

In this regression analysis, I will investigate whether the amount of renewable and nonrenewable energy produced individually and/or collectively affect the prices of energy per EUR/MWH.

For example, would a decrease in the amount of energy produced from solar panels increase the prices of energy per EUR/MWH yet while at the same time; would an increase in the amount of energy produced from fossil fuels decrease the price per EUR/MWH?

These questions are worth pondering over, as figuring out the price of energy per EUR/MWH from the amount of energy produced; based on the type of resource, may indeed lower energy prices as well as appropriately decrease the output volume from pollutants such as fossil fuels.

If one figures out that producing too much energy from one particular source while producing too little energy from one particular source is unnecessarily increasing prices, then this regression analysis can investigate and calculate the direction of energy prices based on the amount produced from several energy sources as a result. Furthermore, this regression analysis can point out which energy sources are increasing the price of energy collectively and individually.

As a result, one can save resources and investigate the causes of higher energy prices from that particular source rather than spend money to investigate all energy sources. A couple factors that can result in overall price increases from certain resources are supply chain issues, labor shortages, technological flaws, etc.

Below is the dataset that I will be analyzing using Python. I downloaded this dataset off of Kaggle. This dataset contains the outputs of energy produced in the scale of megawatts from certain resources followed by the price per EUR/MWH. The observations originated from the country of Spain, and were collected from January 2015 to December 2018.

Below is a index of hashtags to organize the data. Each hashtag will be matched with the conducted analysis within its respective code. For example, the hashtag "#Autocorrelation" would be found within a block of code that calculated the Autocorrelation of the individual dataframes. They are in no particular order. One should insert the hashtags in the search bar in order to examine the graphs, plots, and charts for each resource.

```
In [ ]: #Conclusion  
In [ ]: #Outliers  
In [ ]: # Description Tables  
In [ ]: #Outputs in MWH Histogram  
In [ ]: #Linear OLS regression  
In [ ]: #OLS Linear Summary Table  
In [ ]: #Multiple polynomial regression residual plot  
In [ ]: # Description tables of monthly datasets ("Year_Month" ="0000_0")  
In [ ]: # OLS Logarithmic average monthly predictions versus residuals  
In [ ]: #Predicted OLS average monthly linear values versus actual values  
In [ ]: #OLS Quadratic Summary Table  
In [ ]: # OLS Predicted Quadratic values  
In [ ]: #Linear OLS regression residuals  
In [ ]: #Pie Chart  
In [ ]: #Predicted average monthly OLS quadratic values versus residuals  
In [ ]: # Stem Plot  
In [ ]: #Predicted OLS linear average monthly ratios versus residuals  
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals  
In [ ]: #Rounded Price  
In [ ]: #ADF Tests  
In [ ]: #Autocorrelation Plot
```

```
In [ ]: #Partial autocorrelation Plot  
In [ ]: # Predicted Linear Bell Curve  
In [ ]: # Seasonality values  
In [ ]: #Bell Curves  
In [ ]: #Predicted OLS linear values versus residual values  
In [ ]: #Description table of Dataframes  
In [ ]: # Description tables of monthly datasets ("Year_Month" ="0000_0")  
In [ ]: # Monthly Datasets  
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018  
In [ ]: # Average monthly values  
In [ ]: #Autocorrelations  
In [ ]: #Outliers  
In [ ]: #OLS Quadratic Scatterplot  
In [ ]: #Log  
In [ ]: # Box and Whisker Plot  
In [ ]: #Quadratic OLS regression  
In [ ]: #OLS linear scatterplot  
In [ ]: # Seasonality Plot  
In [ ]: #Predicted OLS average monthly linear values versus actual values  
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
In [ ]: # OLS logarithmic predicted values  
In [ ]: #slope and intercept for OLS linear regression  
In [ ]: #Dataframes analyzed by resource  
In [ ]: #Logarithmic OLS regressions  
In [ ]: #OLS Logarithmic Scatterplots  
In [ ]: #Logarithmic OLS regression residuals  
In [ ]: #OLS Logarithmic summary table  
In [ ]: #Predictive Linear Histogram  
In [ ]: #Predictive Logarithmic Histogram  
In [ ]: #Predictive Quadratic Histogram  
In [ ]: #Output/price per EUR/MWh Histogram  
In [ ]: #Datasets  
In [ ]: #Column Values  
In [ ]: #Predictive Linear Seasonality Plot  
In [ ]: #Predictive Quadratic Seasonality Plot  
In [ ]: #Predictive Logarithmic Seasonality Plot
```

```
In [ ]: import pylab
import numpy as np
from scipy.stats import skew
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import scipy as scipy
import sklearn as sklearn
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt

import pandas as pd
import statsmodels.api as sm
import scipy.stats as stats
```

As one can see, there have been many plug ins downloaded for this regressional analysis. Below is the dataset used for this regressional analysis.

```
In [ ]: df2 = pd.read_csv("./energy_dataset.csv") #Datasets
df2
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0	0.0	0.0	0.0
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0	0.0	0.0
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0	0.0	0.0
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0	0.0	0.0
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0	0.0	0.0
...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	2628.0	178.0	0.0	0.0	0.0
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	2566.0	174.0	0.0	0.0	0.0
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	2422.0	168.0	0.0	0.0	0.0
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	2293.0	163.0	0.0	0.0	0.0
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	2166.0	163.0	0.0	0.0	0.0

35064 rows × 29 columns

Here are the columns within the dataset.

```
In [ ]: print(df2.columns.values) #Column Values
```

```
['time' 'generation biomass' 'generation fossil brown coal/lignite'
 'generation fossil coal-derived gas' 'generation fossil gas'
 'generation fossil hard coal' 'generation fossil oil'
 'generation fossil oil shale' 'generation fossil peat'
 'generation geothermal' 'generation hydro pumped storage aggregated'
 'generation hydro pumped storage consumption'
 'generation hydro run-of-river and poundage'
 'generation hydro water reservoir' 'generation marine'
 'generation nuclear' 'generation other' 'generation other renewable'
 'generation solar' 'generation waste' 'generation wind offshore'
 'generation wind onshore' 'forecast solar day ahead'
 'forecast wind offshore eday ahead' 'forecast wind onshore day ahead'
 'total load forecast' 'total load actual' 'price day ahead'
 'price actual']
```

```
In [ ]:
```

The columns will be extracted per month and analyzed for the sake of the regressional analysis. Although the new data datasets were

extracted based off of the year and month, they are not organized in no particular order. However, the graphs used in this regressional analysis will ensure the chronology of the observations.

```
In [ ]: df3 = pd.read_csv("./energy_dataset.csv") #Datasets
```

```
df3
```

```
Out[ ]:
```

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0	0.0	0.0	0.0
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0	0.0	0.0
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0	0.0	0.0
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0	0.0	0.0
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0	0.0	0.0
...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	2628.0	178.0	0.0	0.0	0.0
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	2566.0	174.0	0.0	0.0	0.0
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	2422.0	168.0	0.0	0.0	0.0
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	2293.0	163.0	0.0	0.0	0.0
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	2166.0	163.0	0.0	0.0	0.0

35064 rows × 29 columns

```
In [ ]: print(df3.columns.values) #Column Values
```

```
['time' 'generation biomass' 'generation fossil brown coal/lignite'  
'generation fossil coal-derived gas' 'generation fossil gas'  
'generation fossil hard coal' 'generation fossil oil'  
'generation fossil oil shale' 'generation fossil peat'  
'generation geothermal' 'generation hydro pumped storage aggregated'  
'generation hydro pumped storage consumption'  
'generation hydro run-of-river and poundage'  
'generation hydro water reservoir' 'generation marine'  
'generation nuclear' 'generation other' 'generation other renewable'  
'generation solar' 'generation waste' 'generation wind offshore'  
'generation wind onshore' 'forecast solar day ahead'  
'forecast wind offshore eday ahead' 'forecast wind onshore day ahead'  
'total load forecast' 'total load actual' 'price day ahead'  
'price actual']
```

Below are line graphs that depict the hourly quantities of energy prices in EUR/MWH and the resources produced from January 2015 to December 2018.

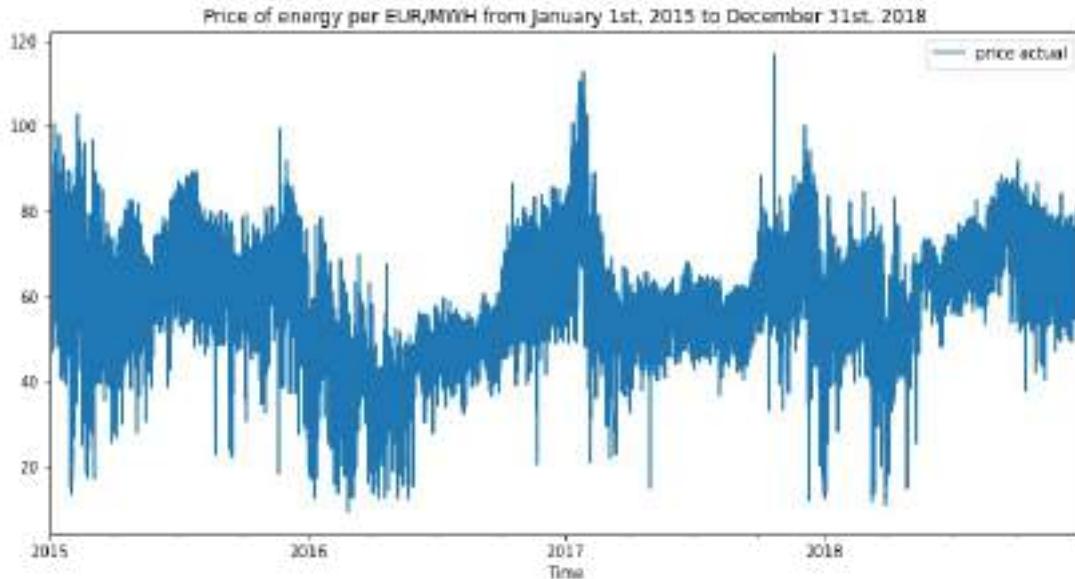
```
In [ ]: print(df3.columns.values) #Column Values
```

```
['time' 'generation biomass' 'generation fossil brown coal/lignite'  
'generation fossil coal-derived gas' 'generation fossil gas'  
'generation fossil hard coal' 'generation fossil oil'  
'generation fossil oil shale' 'generation fossil peat'  
'generation geothermal' 'generation hydro pumped storage aggregated'  
'generation hydro pumped storage consumption'  
'generation hydro run-of-river and poundage'  
'generation hydro water reservoir' 'generation marine'  
'generation nuclear' 'generation other' 'generation other renewable'  
'generation solar' 'generation waste' 'generation wind offshore'  
'generation wind onshore' 'forecast solar day ahead'  
'forecast wind offshore eday ahead' 'forecast wind onshore day ahead'  
'total load forecast' 'total load actual' 'price day ahead'  
'price actual']
```

```
In [ ]: df4 = df3
```

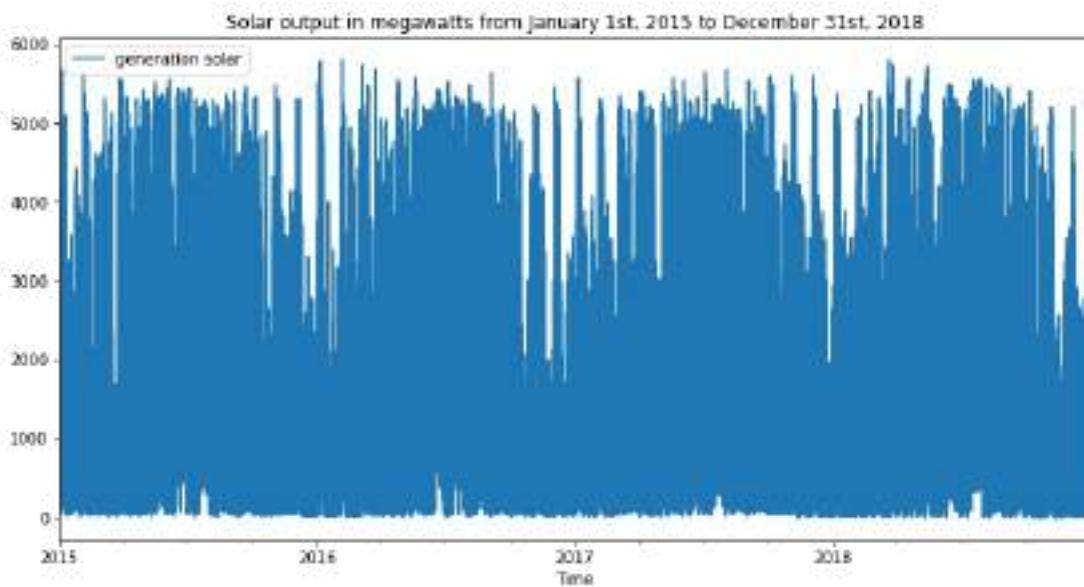
```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018  
df3['Time'] = pd.to_datetime(df3['time'], utc=True)  
ax = df3.plot(x='Time', y='price actual', figsize=(12,6))  
ax.set_title('Price of energy per EUR/MWH from January 1st, 2015 to December 31st, 2018')
```

```
Out[ ]: Text(0.5, 1.0, 'Price of energy per EUR/MWH from January 1st, 2015 to December 31st, 2018')
```



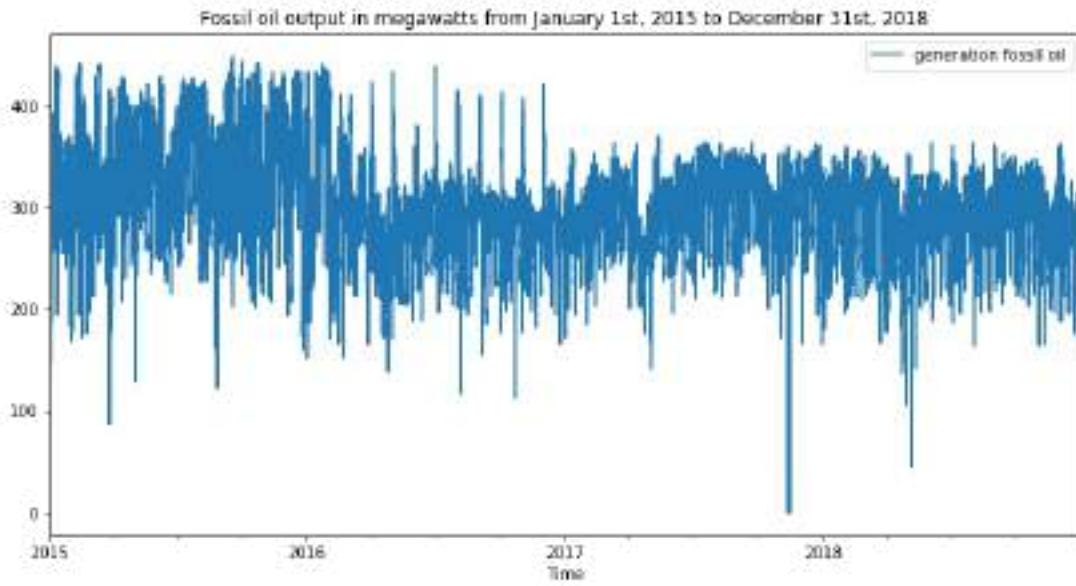
```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018  
df3['Time'] = pd.to_datetime(df3['time'], utc=True)  
ax = df3.plot(x='Time', y='generation solar', figsize=(12,6))  
ax.set_title('Solar output in megawatts from January 1st, 2015 to December 31st, 2018')
```

```
Out[ ]: Text(0.5, 1.0, 'Solar output in megawatts from January 1st, 2015 to December 31st, 2018')
```



```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018  
df3['Time'] = pd.to_datetime(df3['time'], utc=True)  
ax = df3.plot(x='Time', y='generation fossil oil', figsize=(12,6))  
ax.set_title('Fossil oil output in megawatts from January 1st, 2015 to December 31st, 2018')
```

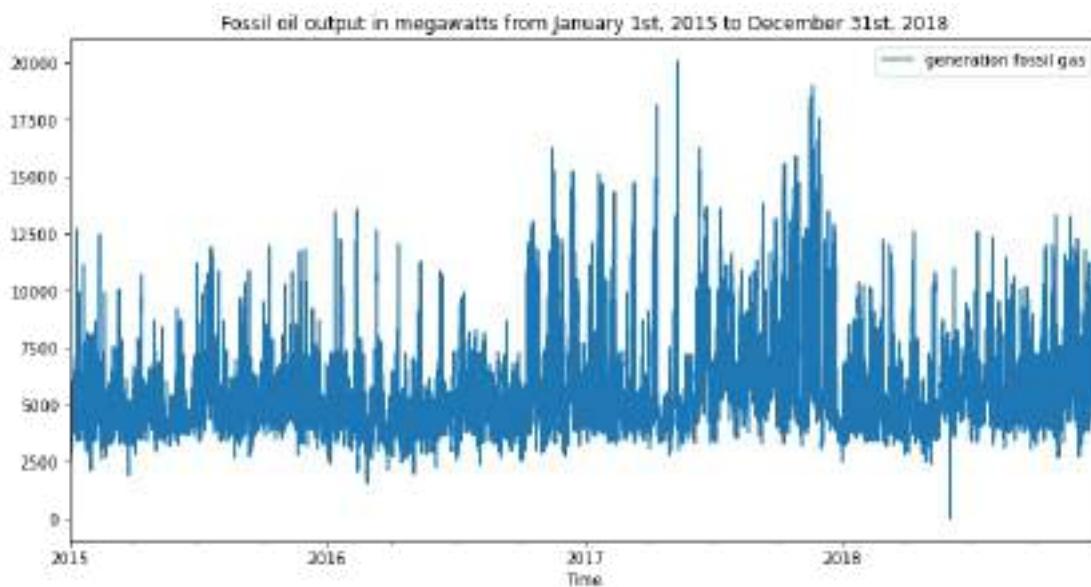
```
Out[ ]: Text(0.5, 1.0, 'Fossil oil output in megawatts from January 1st, 2015 to December 31st, 2018')
```



With the expectation of the sudden output drop right before 2018 and the consistent spike increases between 2016 and 2017, the production of fossil oil was roughly consistent.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation fossil gas', figsize=(12,6))
ax.set_title('Fossil oil output in megawatts from January 1st, 2015 to December 31st, 2018')
```

```
Out[ ]: Text(0.5, 1.0, 'Fossil oil output in megawatts from January 1st, 2015 to December 31st, 2018')
```

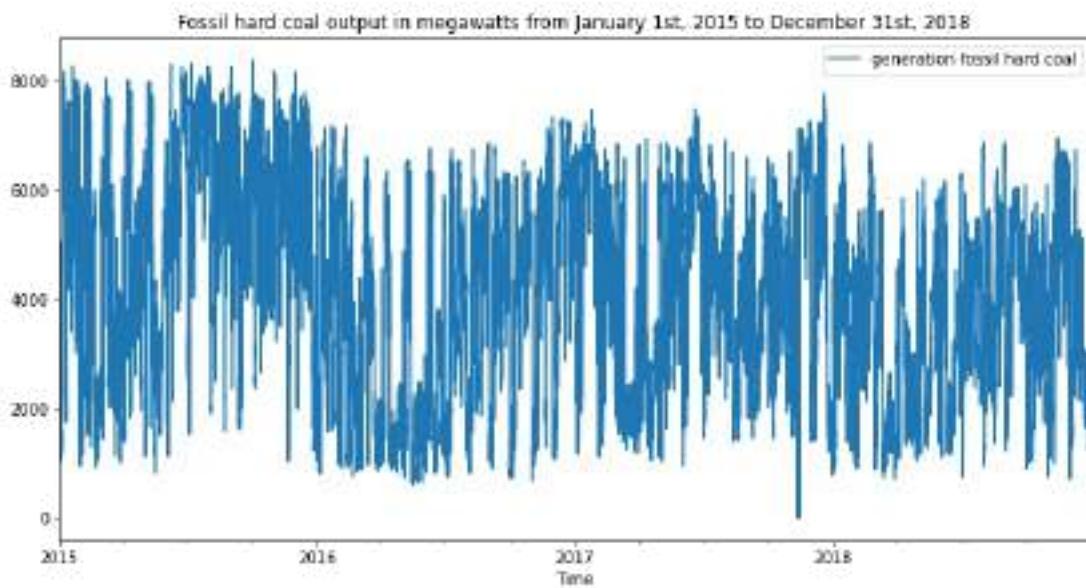


Based off of the constant yet consistent output spikes in the production of fossil oil, it would be reasonable to assume that the production yield was continuously discrete, meaning that the output spikes, which were always followed by periods consolidation, were a frequent and regular occurrence.

```
In [ ]:
```

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation fossil hard coal', figsize=(12,6))
ax.set_title('Fossil hard coal output in megawatts from January 1st, 2015 to December 31st, 2018')
```

```
Out[ ]: Text(0.5, 1.0, 'Fossil hard coal output in megawatts from January 1st, 2015 to December 31st, 2018')
```

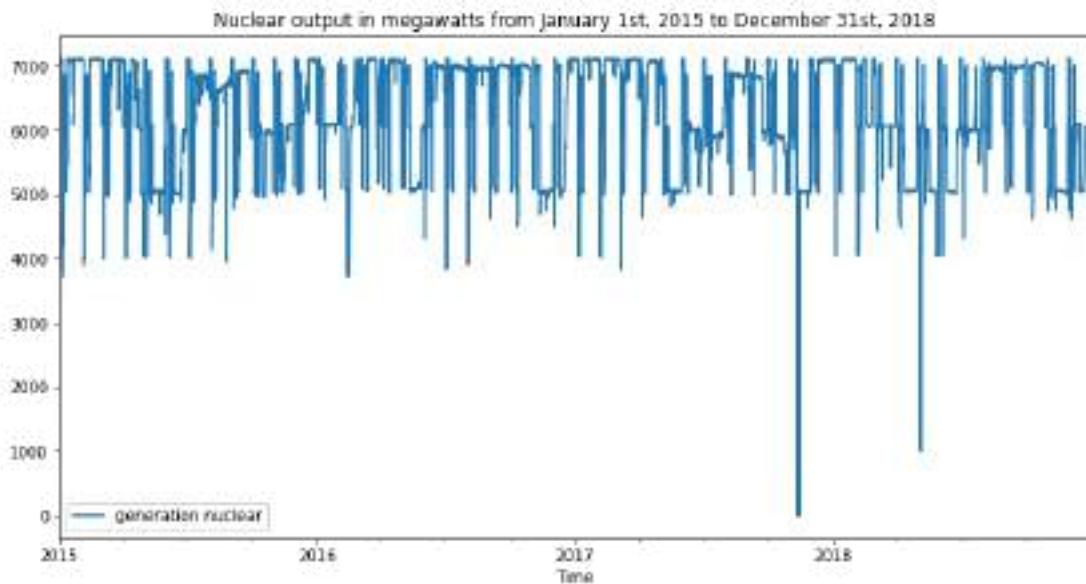


Based on this graph, the production output of fossil hard coal was a consistent roller coaster. There were no unusual output spikes or sharp drops, but production output fluctuated throughout the years.

In []:

```
#Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation nuclear', figsize=(12,6))
ax.set_title('Nuclear output in megawatts from January 1st, 2015 to December 31st, 2018')
```

Out[]: Text(0.5, 1.0, 'Nuclear output in megawatts from January 1st, 2015 to December 31st, 2018')

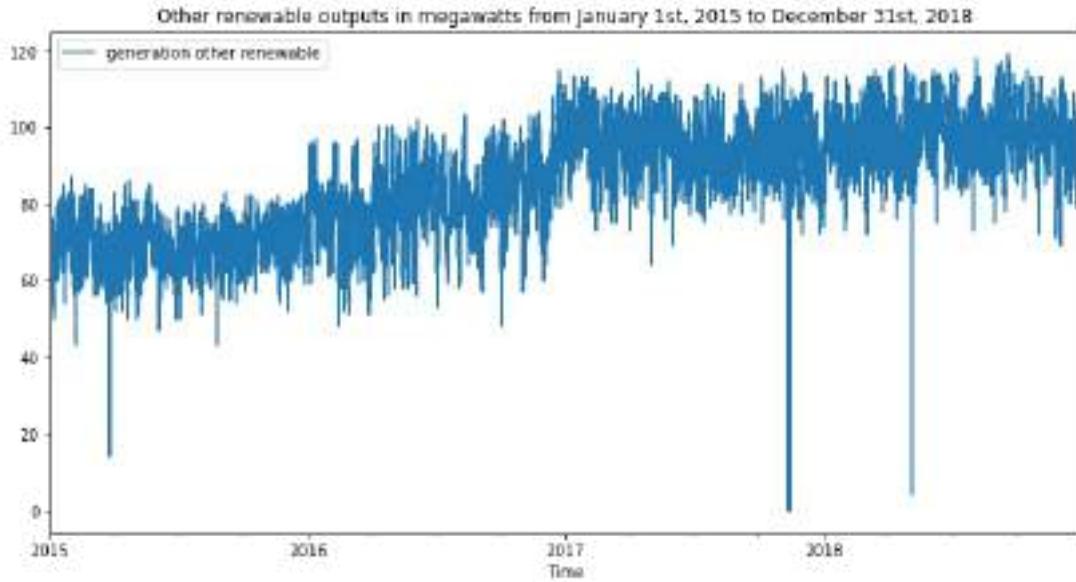


The consistent roller coaster analogy from the previous graph, which depicted the production output of fossil hard coal, can also be applied for the production output of nuclear energy. But unlike fossil hard coal, production outputs of nuclear energy have sharp output drops right before 2018 and right after 2018. Afterwards, the production output consolidated.

In []:

```
#Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation other renewable', figsize=(12,6))
ax.set_title('Other renewable outputs in megawatts from January 1st, 2015 to December 31st, 2018')
```

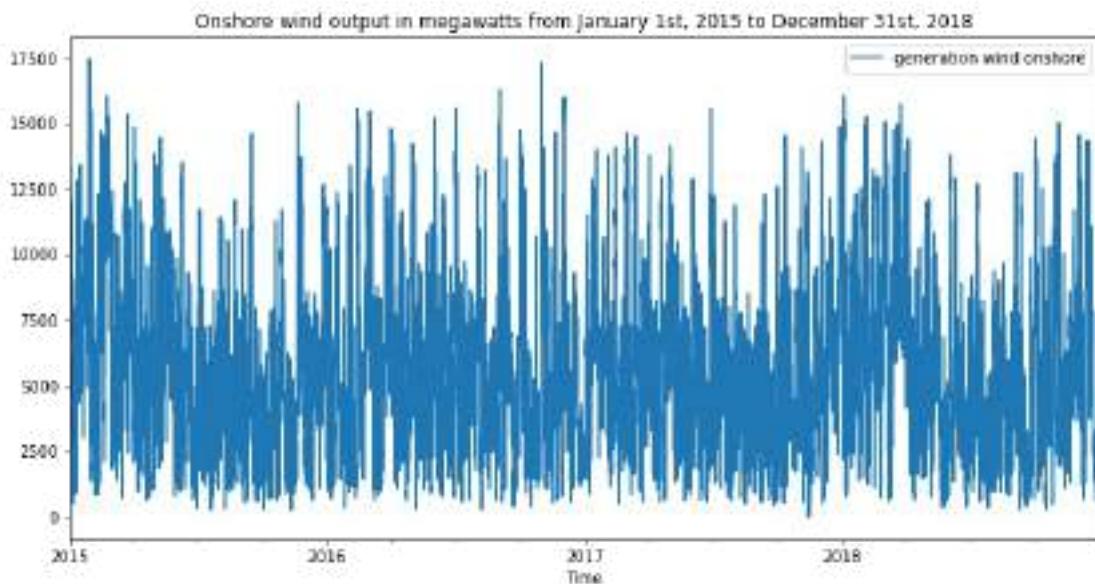
Out[]: Text(0.5, 1.0, 'Other renewable outputs in megawatts from January 1st, 2015 to December 31st, 2018')



Just like nuclear energy, the production outputs of other renewables experienced sharp output drops right before and after 2018, followed by a period of consolidation. In addition, Unlike nuclear energy, though, the production outputs of other renewables gradually increased after an output drop in 2015.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation wind onshore', figsize=(12,6))
ax.set_title('Onshore wind output in megawatts from January 1st, 2015 to December 31st, 2018')
```

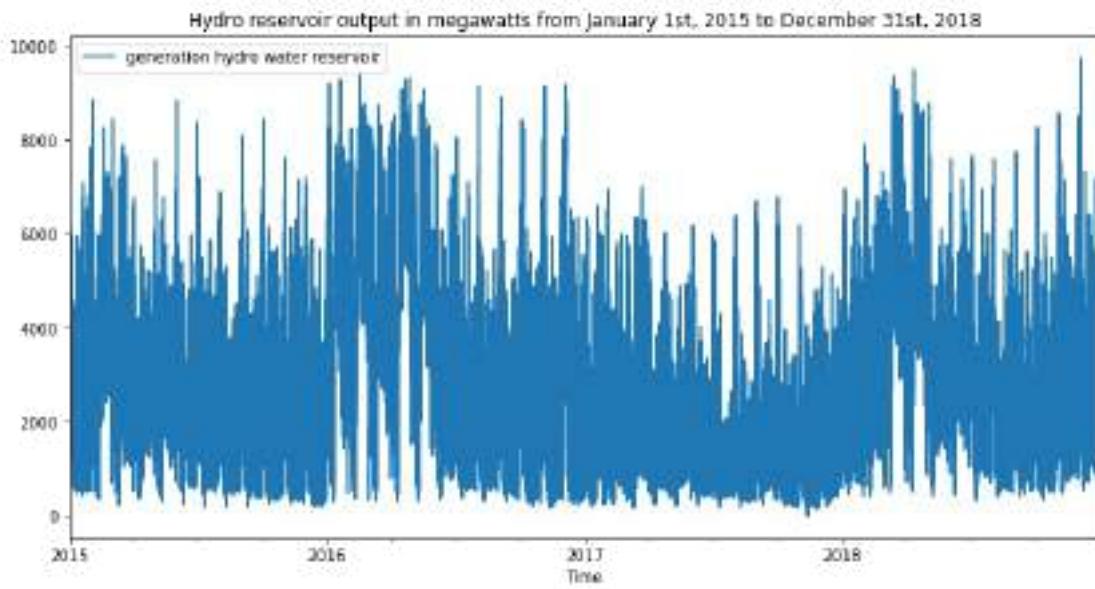
```
Out[ ]: Text(0.5, 1.0, 'Onshore wind output in megawatts from January 1st, 2015 to December 31st, 2018')
```



The production output of onshore wind was continuously discrete, meaning that there were consistent output spikes and drops after a period of consolidation. This pattern repeated itself throughout the years observed.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation hydro water reservoir', figsize=(12,6))
ax.set_title('Hydro reservoir output in megawatts from January 1st, 2015 to December 31st, 2018')
```

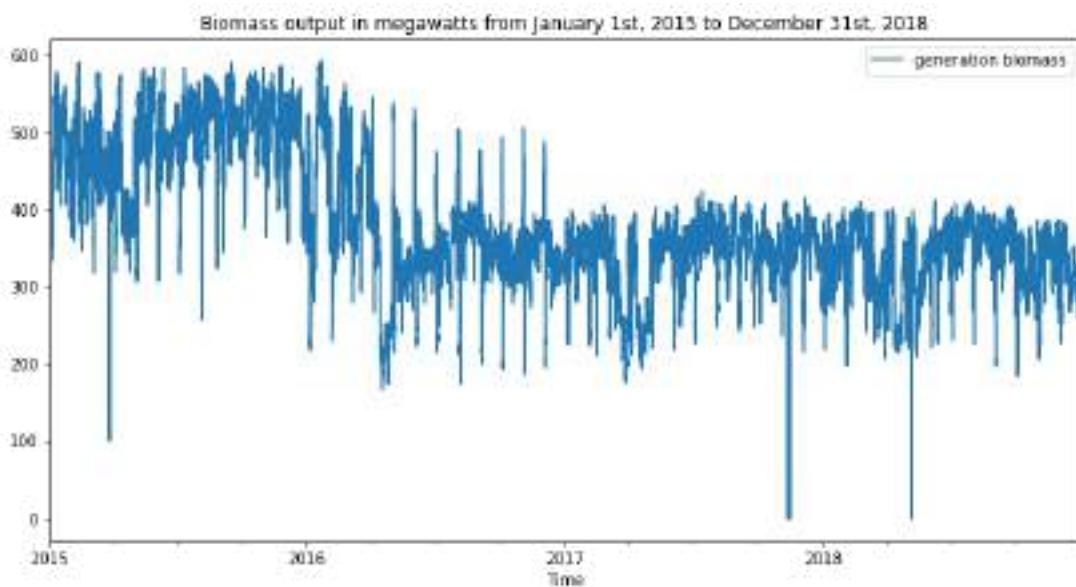
```
Out[ ]: Text(0.5, 1.0, 'Hydro reservoir output in megawatts from January 1st, 2015 to December 31st, 2018')
```



Hydro reservoir production output was discrete; as there were consistent sharp drops and spikes throughout the years observed. But after a period of consolidation from 2015 to 2016, there was a steep increase in the beginning of 2016; followed by a gradual decrease until the middle of 2017. From then on, production output gradually increased while incorporating the continuously discrete drops and spikes.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation biomass', figsize=(12,6))
ax.set_title('Biomass output in megawatts from January 1st, 2015 to December 31st, 2018')
```

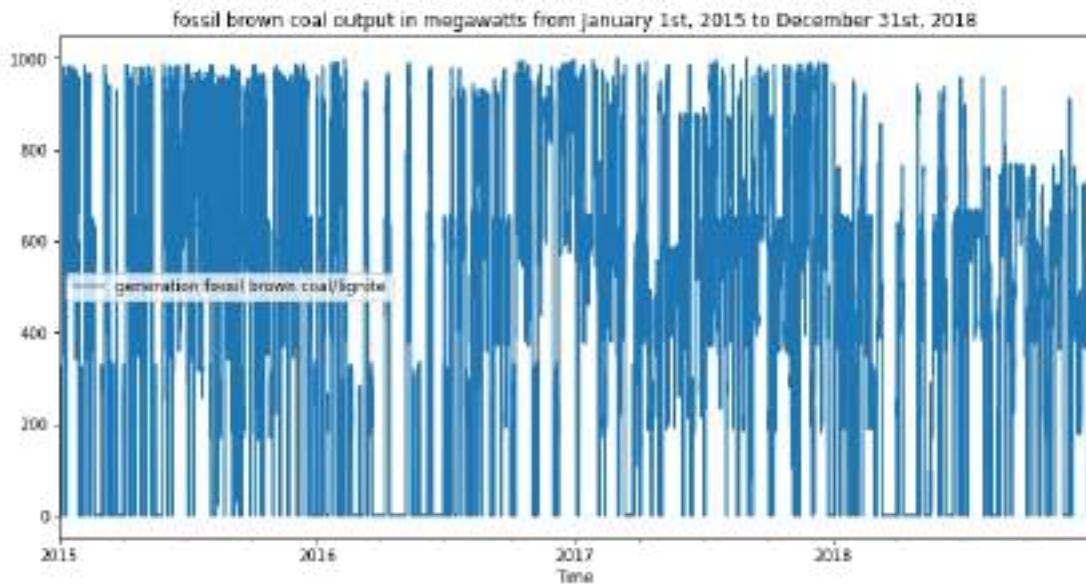
Out[]: Text(0.5, 1.0, 'Biomass output in megawatts from January 1st, 2015 to December 31st, 2018')



Just like the production outputs of other renewables, biomass production experienced sharp output drops right before and after 2018, followed by a period of consolidation. Unlike the production of other renewables, though, the production output of biomass gradually decreased after an output drop in 2015.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st, 2018
df3['Time'] = pd.to_datetime(df3['time'], utc=True)
ax = df3.plot(x='Time', y='generation fossil brown coal/lignite', figsize=(12,6))
ax.set_title('fossil brown coal output in megawatts from January 1st, 2015 to December 31st, 2018')
```

Out[]: Text(0.5, 1.0, 'fossil brown coal output in megawatts from January 1st, 2015 to December 31st, 2018')



Fossil brown coal production output was discrete; as there were consistent sharp drops and spikes throughout the years observed. Some of these time periods had higher consistencies of outputs yet some of these time periods had volatile outputs.

As one can observe, the lines within these graphs are condensed. As such, the dataset has been segmented by monthly averages. By sorting the dataset by the monthly averages, the themes of the variables can be analyzed based off its overall seasonality. In doing this, the observation count changed from 35064 to 48. The dataset can still be confidentially analyzed for statistical significance, since 30 observations is the minimum threshold for this matter. Some of the variable names were changed as well for simplicity. Since there was a lack of observations from off wind production, "Generation Wind Onshore" simply became "Wind". With the exception of "Generation Hydro Water Reservoir", all other hydro production was disqualified from this analysis for their lack of observations.

In []:

Below are the outliers from the columns of the original dataframe. All of these extracted columns were used as independent and dependant variables in the regression extracted. These outliers will be included in the monthly averages since the 35064 observations were collected on an hourly basis. This hourly timeframe is very short in comparison to the monthly timeframe used in the modification process.

```
In [ ]: def find_outliers_IQR(df2): #Outliers
    q1=df2.quantile(0.25)
    q3=df2.quantile(0.75)
    IQR=q3-q1
    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]
    return outliers

outliers_price = find_outliers_IQR(df2['price actual'])
print("number of outliers:"+ str(len(outliers_price)))
print("max outlier value:"+ str(outliers_price.max()))
print("min outlier value:"+ str(outliers_price.min()))
print(outliers_price)
```

```
number of outliers:699
max outlier value:116.8
min outlier value:9.33
154      100.45
322      97.95
697      16.79
698      15.93
699      15.76
...
29149    18.15
29150    14.61
29151    15.05
29152    18.64
29153    17.93
Name: price actual, Length: 699, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers

    q1=df2.quantile(0.25)

    q3=df2.quantile(0.75)

    IQR=q3-q1

    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]

    return outliers

outliers_generation_nuclear = find_outliers_IQR(df2['generation nuclear'])

print("number of outliers:"+ str(len(outliers_generation_nuclear)))

print("max outlier value:"+ str(outliers_generation_nuclear.max()))

print("min outlier value:"+ str(outliers_generation_nuclear.min()))

print(outliers_generation_nuclear)
```

```
number of outliers:74
max outlier value:3862.0
min outlier value:0.0
107      3829.0
114      3712.0
115      3743.0
116      3773.0
117      3803.0
...
19088    3856.0
25125    0.0
25164    0.0
25171    0.0
29315    998.0
Name: generation nuclear, Length: 74, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers

    q1=df2.quantile(0.25)

    q3=df2.quantile(0.75)

    IQR=q3-q1

    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]

    return outliers

outliers_generation_wind = find_outliers_IQR(df2['generation wind onshore'])

print("number of outliers:"+ str(len(outliers_generation_wind)))

print("max outlier value:"+ str(outliers_generation_wind.max()))

print("min outlier value:"+ str(outliers_generation_wind.min()))

print(outliers_generation_wind)
```

```
number of outliers:379
max outlier value:17436.0
min outlier value:14098.0
684      14734.0
685      15841.0
686      16603.0
687      16931.0
688      17256.0
...
34624    14316.0
34625    14288.0
34626    14220.0
34694    14127.0
34695    14305.0
Name: generation wind onshore, Length: 379, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers

    q1=df2.quantile(0.25)

    q3=df2.quantile(0.75)

    IQR=q3-q1

    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]

    return outliers

outliers_generation_hydro = find_outliers_IQR(df2['generation hydro water reservoir'])

print("number of outliers:"+ str(len(outliers_generation_hydro)))

print("max outlier value:"+ str(outliers_generation_hydro.max()))

print("min outlier value:"+ str(outliers_generation_hydro.min()))

print(outliers_generation_hydro)
```

```
number of outliers:343
max outlier value:9728.0
min outlier value:7777.0
765      7833.0
811      8165.0
812      8837.0
813      8320.0
1195     8251.0
...
34402    8383.0
34403    8070.0
34412    8311.0
34413    9728.0
34414    8791.0
Name: generation hydro water reservoir, Length: 343, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers

    q1=df2.quantile(0.25)

    q3=df2.quantile(0.75)

    IQR=q3-q1

    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]

    return outliers

outliers_generation_other_renewable = find_outliers_IQR(df2['generation other renewable'])

print("number of outliers:"+ str(len(outliers_generation_other_renewable)))

print("max outlier value:"+ str(outliers_generation_other_renewable.max()))

print("min outlier value:"+ str(outliers_generation_other_renewable.min()))

print(outliers_generation_other_renewable)
```

```
number of outliers:5
max outlier value:14.0
min outlier value:0.0
2025      14.0
25125      0.0
25164      0.0
25171      0.0
29315      4.0
Name: generation other renewable, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers
```

```
    q1=df2.quantile(0.25)
    q3=df2.quantile(0.75)
    IQR=q3-q1
    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]
    return outliers

outliers_generation_brown = find_outliers_IQR(df2['generation fossil brown coal/lignite'])

print("number of outliers:"+ str(len(outliers_generation_brown)))
print("max outlier value:"+ str(outliers_generation_brown.max()))
print("min outlier value:"+ str(outliers_generation_brown.min()))
print(outliers_generation_brown)
```

```
number of outliers:0
max outlier value:nan
min outlier value:nan
Series([], Name: generation fossil brown coal/lignite, dtype: float64)
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers
```

```
    q1=df2.quantile(0.25)
    q3=df2.quantile(0.75)
    IQR=q3-q1
    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]
    return outliers

outliers_generation_solar = find_outliers_IQR(df2['generation solar'])

print("number of outliers:"+ str(len(outliers_generation_solar)))
print("max outlier value:"+ str(outliers_generation_solar.max()))
print("min outlier value:"+ str(outliers_generation_solar.min()))
print(outliers_generation_solar)
```

```
number of outliers:0
max outlier value:nan
min outlier value:nan
Series([], Name: generation solar, dtype: float64)
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers
```

```
    q1=df2.quantile(0.25)
    q3=df2.quantile(0.75)
    IQR=q3-q1
    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]
    return outliers

outliers_Fossil_Hard = find_outliers_IQR(df2['generation fossil hard coal'])

print("number of outliers:"+ str(len(outliers_Fossil_Hard)))
print("max outlier value:"+ str(outliers_Fossil_Hard.max()))
```

```
print("min outlier value:"+ str(outliers_Fossil_Hard.min()))
```

```
print(outliers_Fossil_Hard)
```

```
number of outliers:0  
max outlier value:nan  
min outlier value:nan  
Series([], Name: generation fossil hard coal, dtype: float64)
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers
```

```
    q1=df2.quantile(0.25)
```

```
    q3=df2.quantile(0.75)
```

```
    IQR=q3-q1
```

```
    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]
```

```
    return outliers
```

```
outliers_fossil_oil = find_outliers_IQR(df2['generation fossil oil'])
```

```
print("number of outliers:"+ str(len(outliers_fossil_oil)))
```

```
print("max outlier value:"+ str(outliers_fossil_oil.max()))
```

```
print("min outlier value:"+ str(outliers_fossil_oil.min()))
```

```
print(outliers_fossil_oil)
```

```
number of outliers:246  
max outlier value:449.0  
min outlier value:0.0  
0      162.0  
1      158.0  
2      157.0  
3      160.0  
4      156.0  
...  
29154     118.0  
29155     142.0  
29315      44.0  
29489     141.0  
29490     147.0  
Name: generation fossil oil, Length: 246, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers
```

```
    q1=df2.quantile(0.25)
```

```
    q3=df2.quantile(0.75)
```

```
    IQR=q3-q1
```

```
    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]
```

```
    return outliers
```

```
outliers_fossil_gas = find_outliers_IQR(df2['generation fossil gas'])
```

```
print("number of outliers:"+ str(len(outliers_fossil_gas)))
```

```
print("max outlier value:"+ str(outliers_fossil_gas.max()))
```

```
print("min outlier value:"+ str(outliers_fossil_gas.min()))
```

```
print(outliers_fossil_gas)
```

```
number of outliers:2185
max outlier value:20034.0
min outlier value:0.0
280      10336.0
281      11171.0
282      12227.0
283      12415.0
284      12532.0
...
34965    13091.0
34966    11128.0
34985    10076.0
34986    10189.0
34987    10265.0
Name: generation fossil gas, Length: 2185, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers

    q1=df2.quantile(0.25)

    q3=df2.quantile(0.75)

    IQR=q3-q1

    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]

    return outliers

outliers_fossil_gas = find_outliers_IQR(df2['generation fossil gas'])

print("number of outliers:"+ str(len(outliers_fossil_gas)))

print("max outlier value:"+ str(outliers_fossil_gas.max()))

print("min outlier value:"+ str(outliers_fossil_gas.min()))

print(outliers_fossil_gas)
```

```
number of outliers:2185
max outlier value:20034.0
min outlier value:0.0
280      10336.0
281      11171.0
282      12227.0
283      12415.0
284      12532.0
...
34965    13091.0
34966    11128.0
34985    10076.0
34986    10189.0
34987    10265.0
Name: generation fossil gas, Length: 2185, dtype: float64
```

```
In [ ]: def find_outliers_IQR(df2): #Outliers

    q1=df2.quantile(0.25)

    q3=df2.quantile(0.75)

    IQR=q3-q1

    outliers = df2[((df2<(q1-1.5*IQR)) | (df2>(q3+1.5*IQR)))]

    return outliers

outliers_biomass = find_outliers_IQR(df2['generation biomass'])

print("number of outliers:"+ str(len(outliers_biomass)))

print("max outlier value:"+ str(outliers_biomass.max()))

print("min outlier value:"+ str(outliers_biomass.min()))

print(outliers_biomass)
```

```

number of outliers:87
max outlier value:592.0
min outlier value:0.0
998      586.0
1000      586.0
1001      586.0
1002      588.0
1003      590.0
...
19643     181.0
25125      0.0
25164      0.0
25171      0.0
29315      0.0
Name: generation biomass, Length: 87, dtype: float64

```

Below this sentence are the modified dataframes extracted from the original dataset. These modified dataframes were organized on a monthly basis.

```
In [ ]: Dates = pd.date_range('2015-1','2018-12',
                           freq='MS').strftime("%y-%m").tolist()
```

```
In [ ]: Months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]

dicDates = {Date: Month for Date, Month in zip(Dates, Months)}
print(dicDates)
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```

```
In [ ]: print(Dates)
```

```
['15-01', '15-02', '15-03', '15-04', '15-05', '15-06', '15-07', '15-08', '15-09', '15-10', '15-11', '15-12', '16-01', '16-02', '16-03', '16-04', '16-05', '16-06', '16-07', '16-08', '16-09', '16-10', '16-11', '16-12', '17-01', '17-02', '17-03', '17-04', '17-05', '17-06', '17-07', '17-08', '17-09', '17-10', '17-11', '17-12', '18-01', '18-02', '18-03', '18-04', '18-05', '18-06', '18-07', '18-08', '18-09', '18-10', '18-11', '18-12']
```

```
In [ ]: df2018_1 = df2[df2['time'].str.contains("2018-01")]
df2018_1
# Monthly Datasets ("Year_Month" = "0000_0")
```

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
26304	2018-01-01 00:00:00+01:00	282.0	0.0	0.0	3471.0	996.0	194.0	0.0	0.0	0.0
26305	2018-01-01 01:00:00+01:00	275.0	0.0	0.0	3269.0	959.0	191.0	0.0	0.0	0.0
26306	2018-01-01 02:00:00+01:00	278.0	0.0	0.0	3541.0	1014.0	191.0	0.0	0.0	0.0
26307	2018-01-01 03:00:00+01:00	278.0	0.0	0.0	3450.0	1043.0	191.0	0.0	0.0	0.0
26308	2018-01-01 04:00:00+01:00	279.0	0.0	0.0	3318.0	1063.0	191.0	0.0	0.0	0.0
...
27043	2018-01-31 19:00:00+01:00	387.0	936.0	0.0	10082.0	4407.0	330.0	0.0	0.0	0.0
27044	2018-01-31 20:00:00+01:00	380.0	918.0	0.0	10207.0	4409.0	333.0	0.0	0.0	0.0
27045	2018-01-31 21:00:00+01:00	382.0	942.0	0.0	9671.0	4411.0	335.0	0.0	0.0	0.0
27046	2018-01-31 22:00:00+01:00	375.0	938.0	0.0	8773.0	4398.0	334.0	0.0	0.0	0.0
27047	2018-01-31 23:00:00+01:00	374.0	816.0	0.0	8081.0	4314.0	335.0	0.0	0.0	0.0

744 rows × 29 columns

```
In [ ]: df2016_1 = df2[df2['time'].str.contains("2016-01")]
df2016_1
```

```
# Monthly Datasets ("Year_Month" = "0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation hard coal	generation fossil oil	generation shale	generation peat	generation geothermal	...
8760	2016-01-01 00:00:00+01:00	410.0	0.0	0.0	4610.0	2596.0	218.0	0.0	0.0	0.0	.
8761	2016-01-01 01:00:00+01:00	404.0	0.0	0.0	4434.0	1866.0	216.0	0.0	0.0	0.0	.
8762	2016-01-01 02:00:00+01:00	395.0	0.0	0.0	4286.0	1422.0	205.0	0.0	0.0	0.0	.
8763	2016-01-01 03:00:00+01:00	391.0	0.0	0.0	3813.0	1146.0	202.0	0.0	0.0	0.0	.
8764	2016-01-01 04:00:00+01:00	384.0	0.0	0.0	3872.0	1081.0	200.0	0.0	0.0	0.0	.
...
9499	2016-01-31 19:00:00+01:00	514.0	0.0	0.0	3532.0	2541.0	316.0	0.0	0.0	0.0	.
9500	2016-01-31 20:00:00+01:00	491.0	0.0	0.0	3725.0	2613.0	316.0	0.0	0.0	0.0	.
9501	2016-01-31 21:00:00+01:00	486.0	0.0	0.0	3926.0	2618.0	316.0	0.0	0.0	0.0	.
9502	2016-01-31 22:00:00+01:00	484.0	0.0	0.0	4158.0	2661.0	316.0	0.0	0.0	0.0	.
9503	2016-01-31 23:00:00+01:00	497.0	0.0	0.0	3826.0	2514.0	316.0	0.0	0.0	0.0	.

744 rows × 29 columns

```
In [ ]: df2016_1.describe() # Description tables of monthly datasets ("Year_Month" = "0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation hard coal	generation fossil oil	generation shale	generation peat	generation geothermal	generation hydro pumped storage aggregated	...
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	744.0	0.0
mean	459.995968	417.927419	0.0	5271.119624	4204.615591	337.465054	0.0	0.0	0.0	0.0	NaN
std	88.130508	394.067088	0.0	1952.831382	1913.785542	67.960672	0.0	0.0	0.0	0.0	NaN
min	217.000000	0.000000	0.0	2417.000000	791.000000	151.000000	0.0	0.0	0.0	0.0	NaN
25%	382.000000	0.000000	0.0	3932.000000	2378.000000	296.000000	0.0	0.0	0.0	0.0	NaN
50%	487.000000	454.000000	0.0	4760.000000	4515.500000	321.000000	0.0	0.0	0.0	0.0	NaN
75%	522.000000	840.250000	0.0	6130.500000	5852.000000	407.250000	0.0	0.0	0.0	0.0	NaN
max	592.000000	992.000000	0.0	13444.000000	7183.000000	441.000000	0.0	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2015_1 = df2[df2['time'].str.contains("2015-01")]
df2015_1
# Monthly Datasets ("Year_Month" = "0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4821.0	162.0	0.0	0.0	0.0	...
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4755.0	158.0	0.0	0.0	0.0	...
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4581.0	157.0	0.0	0.0	0.0	...
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4131.0	160.0	0.0	0.0	0.0	...
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3840.0	156.0	0.0	0.0	0.0	...
...
739	2015-01-31 19:00:00+01:00	373.0	0.0	0.0	3922.0	1320.0	205.0	0.0	0.0	0.0	...
740	2015-01-31 20:00:00+01:00	369.0	0.0	0.0	4206.0	1375.0	205.0	0.0	0.0	0.0	...
741	2015-01-31 21:00:00+01:00	369.0	0.0	0.0	4354.0	1365.0	206.0	0.0	0.0	0.0	...
742	2015-01-31 22:00:00+01:00	365.0	0.0	0.0	4007.0	1358.0	206.0	0.0	0.0	0.0	...
743	2015-01-31 23:00:00+01:00	366.0	0.0	0.0	3189.0	1335.0	206.0	0.0	0.0	0.0	...

744 rows × 29 columns

In []:	df2016_1 = df2[df2['time'].str.contains("2016-01")] df2016_1 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
8760	2016-01-01 00:00:00+01:00	410.0	0.0	0.0	4610.0	2596.0	218.0	0.0	0.0	0.0	...
8761	2016-01-01 01:00:00+01:00	404.0	0.0	0.0	4434.0	1866.0	216.0	0.0	0.0	0.0	...
8762	2016-01-01 02:00:00+01:00	395.0	0.0	0.0	4286.0	1422.0	205.0	0.0	0.0	0.0	...
8763	2016-01-01 03:00:00+01:00	391.0	0.0	0.0	3813.0	1146.0	202.0	0.0	0.0	0.0	...
8764	2016-01-01 04:00:00+01:00	384.0	0.0	0.0	3872.0	1081.0	200.0	0.0	0.0	0.0	...
...
9499	2016-01-31 19:00:00+01:00	514.0	0.0	0.0	3532.0	2541.0	316.0	0.0	0.0	0.0	...
9500	2016-01-31 20:00:00+01:00	491.0	0.0	0.0	3725.0	2613.0	316.0	0.0	0.0	0.0	...
9501	2016-01-31 21:00:00+01:00	486.0	0.0	0.0	3926.0	2618.0	316.0	0.0	0.0	0.0	...
9502	2016-01-31 22:00:00+01:00	484.0	0.0	0.0	4158.0	2661.0	316.0	0.0	0.0	0.0	...
9503	2016-01-31 23:00:00+01:00	497.0	0.0	0.0	3826.0	2514.0	316.0	0.0	0.0	0.0	...

744 rows × 29 columns

In []:	
In []:	df2016_1.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	459.995968	417.927419	0.0	5271.119624	4204.615591	337.465054	0.0	0.0	0.0	NaN
std	88.130508	394.067088	0.0	1952.831382	1913.785542	67.960672	0.0	0.0	0.0	NaN
min	217.000000	0.000000	0.0	2417.000000	791.000000	151.000000	0.0	0.0	0.0	NaN
25%	382.000000	0.000000	0.0	3932.000000	2378.000000	296.000000	0.0	0.0	0.0	NaN
50%	487.000000	454.000000	0.0	4760.000000	4515.500000	321.000000	0.0	0.0	0.0	NaN
75%	522.000000	840.250000	0.0	6130.500000	5852.000000	407.250000	0.0	0.0	0.0	NaN
max	592.000000	992.000000	0.0	13444.000000	7183.000000	441.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2017_1 = df2[df2['time'].str.contains("2017-01")] df2017_1 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
17544	2017-01-01 00:00:00+01:00	341.0	901.0	0.0	5412.0	6157.0	175.0	0.0	0.0	0.0
17545	2017-01-01 01:00:00+01:00	338.0	900.0	0.0	5401.0	5959.0	176.0	0.0	0.0	0.0
17546	2017-01-01 02:00:00+01:00	337.0	908.0	0.0	4753.0	5723.0	175.0	0.0	0.0	0.0
17547	2017-01-01 03:00:00+01:00	335.0	915.0	0.0	4321.0	5523.0	175.0	0.0	0.0	0.0
17548	2017-01-01 04:00:00+01:00	336.0	904.0	0.0	4320.0	5295.0	175.0	0.0	0.0	0.0
...
18283	2017-01-31 19:00:00+01:00	376.0	644.0	0.0	10418.0	6609.0	321.0	0.0	0.0	0.0
18284	2017-01-31 20:00:00+01:00	374.0	656.0	0.0	10127.0	6560.0	321.0	0.0	0.0	0.0
18285	2017-01-31 21:00:00+01:00	376.0	590.0	0.0	8080.0	6618.0	321.0	0.0	0.0	0.0
18286	2017-01-31 22:00:00+01:00	378.0	558.0	0.0	7145.0	6610.0	322.0	0.0	0.0	0.0
18287	2017-01-31 23:00:00+01:00	371.0	591.0	0.0	6118.0	6277.0	321.0	0.0	0.0	0.0

744 rows × 29 columns

In []:	df2017_1.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	347.283602	688.645161	0.0	6412.591398	5533.782258	279.522849	0.0	0.0	0.0	NaN
std	35.523525	230.923176	0.0	2548.380276	1632.328442	41.544632	0.0	0.0	0.0	NaN
min	224.000000	0.000000	0.0	3323.000000	1157.000000	171.000000	0.0	0.0	0.0	NaN
25%	338.000000	595.000000	0.0	4415.000000	4983.500000	256.750000	0.0	0.0	0.0	NaN
50%	358.000000	649.000000	0.0	5552.000000	6157.500000	291.000000	0.0	0.0	0.0	NaN
75%	366.000000	895.250000	0.0	8030.750000	6690.250000	308.000000	0.0	0.0	0.0	NaN
max	399.000000	995.000000	0.0	15092.000000	7452.000000	358.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
df2018_1 = df2[df2['time'].str.contains("2018-01")]
df2018_1
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
26304	2018-01-01 00:00:00+01:00	282.0	0.0	0.0	3471.0	996.0	194.0	0.0	0.0	0.0
26305	2018-01-01 01:00:00+01:00	275.0	0.0	0.0	3269.0	959.0	191.0	0.0	0.0	0.0
26306	2018-01-01 02:00:00+01:00	278.0	0.0	0.0	3541.0	1014.0	191.0	0.0	0.0	0.0
26307	2018-01-01 03:00:00+01:00	278.0	0.0	0.0	3450.0	1043.0	191.0	0.0	0.0	0.0
26308	2018-01-01 04:00:00+01:00	279.0	0.0	0.0	3318.0	1063.0	191.0	0.0	0.0	0.0
...
27043	2018-01-31 19:00:00+01:00	387.0	936.0	0.0	10082.0	4407.0	330.0	0.0	0.0	0.0
27044	2018-01-31 20:00:00+01:00	380.0	918.0	0.0	10207.0	4409.0	333.0	0.0	0.0	0.0
27045	2018-01-31 21:00:00+01:00	382.0	942.0	0.0	9671.0	4411.0	335.0	0.0	0.0	0.0
27046	2018-01-31 22:00:00+01:00	375.0	938.0	0.0	8773.0	4398.0	334.0	0.0	0.0	0.0
27047	2018-01-31 23:00:00+01:00	374.0	816.0	0.0	8081.0	4314.0	335.0	0.0	0.0	0.0

744 rows × 29 columns

```
df2018_1.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	343.848118	379.564516	0.0	5687.715054	3783.622312	285.739247	0.0	0.0	0.0	NaN
std	35.794761	295.186322	0.0	1802.274578	1613.661523	44.679107	0.0	0.0	0.0	NaN
min	219.000000	0.000000	0.0	2452.000000	895.000000	180.000000	0.0	0.0	0.0	NaN
25%	326.750000	0.000000	0.0	4201.500000	2218.500000	251.000000	0.0	0.0	0.0	NaN
50%	354.000000	443.000000	0.0	5269.000000	4114.500000	294.500000	0.0	0.0	0.0	NaN
75%	370.250000	602.250000	0.0	6724.750000	4897.500000	325.000000	0.0	0.0	0.0	NaN
max	402.000000	952.000000	0.0	10326.000000	6827.000000	351.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2015_2 = df2[df2['time'].str.contains("2015-02")] df2015_2 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
744	2015-02-01 00:00:00+01:00	433.0	323.0	0.0	4826.0	4883.0	169.0	0.0	0.0	0.0
745	2015-02-01 01:00:00+01:00	445.0	316.0	0.0	4891.0	4676.0	187.0	0.0	0.0	0.0
746	2015-02-01 02:00:00+01:00	441.0	320.0	0.0	4652.0	4693.0	180.0	0.0	0.0	0.0
747	2015-02-01 03:00:00+01:00	440.0	323.0	0.0	4603.0	4587.0	180.0	0.0	0.0	0.0
748	2015-02-01 04:00:00+01:00	432.0	310.0	0.0	4212.0	4597.0	177.0	0.0	0.0	0.0
...
1411	2015-02-28 19:00:00+01:00	457.0	0.0	0.0	4591.0	2172.0	240.0	0.0	0.0	0.0
1412	2015-02-28 20:00:00+01:00	450.0	0.0	0.0	4652.0	2320.0	239.0	0.0	0.0	0.0
1413	2015-02-28 21:00:00+01:00	458.0	0.0	0.0	4358.0	2102.0	243.0	0.0	0.0	0.0
1414	2015-02-28 22:00:00+01:00	453.0	0.0	0.0	3835.0	2128.0	252.0	0.0	0.0	0.0
1415	2015-02-28 23:00:00+01:00	447.0	0.0	0.0	3747.0	2144.0	254.0	0.0	0.0	0.0

672 rows × 29 columns

In []:	df2015_2.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	672.000000	672.000000	672.0	672.000000	672.000000	672.000000	672.0	672.0	672.0	0.0
mean	470.150298	313.418155	0.0	4674.135417	4045.974702	319.239583	0.0	0.0	0.0	NaN
std	51.148418	375.102898	0.0	1586.403229	2240.121251	65.013267	0.0	0.0	0.0	NaN
min	346.000000	0.000000	0.0	2591.000000	922.000000	169.000000	0.0	0.0	0.0	NaN
25%	440.000000	0.000000	0.0	3577.750000	2008.750000	267.000000	0.0	0.0	0.0	NaN
50%	477.000000	0.000000	0.0	4293.500000	3478.000000	320.000000	0.0	0.0	0.0	NaN
75%	505.250000	605.250000	0.0	5230.750000	5997.500000	373.250000	0.0	0.0	0.0	NaN
max	590.000000	984.000000	0.0	12447.000000	7922.000000	442.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
df2016_2 = df2[df2['time'].str.contains("2016-02")]
df2016_2
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
9504	2016-02-01 00:00:00+01:00	383.0	0.0	0.0	3204.0	967.0	213.0	0.0	0.0	0.0
9505	2016-02-01 01:00:00+01:00	348.0	0.0	0.0	3268.0	1026.0	172.0	0.0	0.0	0.0
9506	2016-02-01 02:00:00+01:00	344.0	0.0	0.0	3361.0	1016.0	170.0	0.0	0.0	0.0
9507	2016-02-01 03:00:00+01:00	343.0	0.0	0.0	3455.0	1086.0	171.0	0.0	0.0	0.0
9508	2016-02-01 04:00:00+01:00	348.0	0.0	0.0	3584.0	1084.0	171.0	0.0	0.0	0.0
...
10195	2016-02-29 19:00:00+01:00	515.0	0.0	0.0	4170.0	2115.0	351.0	0.0	0.0	0.0
10196	2016-02-29 20:00:00+01:00	520.0	0.0	0.0	3944.0	2089.0	345.0	0.0	0.0	0.0
10197	2016-02-29 21:00:00+01:00	519.0	0.0	0.0	3814.0	2118.0	351.0	0.0	0.0	0.0
10198	2016-02-29 22:00:00+01:00	514.0	0.0	0.0	3204.0	2183.0	341.0	0.0	0.0	0.0
10199	2016-02-29 23:00:00+01:00	510.0	0.0	0.0	2802.0	1998.0	338.0	0.0	0.0	0.0

696 rows × 29 columns

```
df2016_2.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	696.000000	696.000000	696.0	696.000000	696.000000	696.000000	696.0	696.0	696.0	0.0
mean	430.344828	191.665230	0.0	4450.360632	3027.655172	297.254310	0.0	0.0	0.0	NaN
std	80.313301	322.133055	0.0	2159.888718	1878.734871	59.034047	0.0	0.0	0.0	NaN
min	230.000000	0.000000	0.0	1518.000000	756.000000	152.000000	0.0	0.0	0.0	NaN
25%	368.750000	0.000000	0.0	3239.000000	1263.500000	250.000000	0.0	0.0	0.0	NaN
50%	416.000000	0.000000	0.0	3925.000000	2457.000000	293.000000	0.0	0.0	0.0	NaN
75%	514.250000	261.500000	0.0	5134.750000	4496.000000	350.000000	0.0	0.0	0.0	NaN
max	563.000000	997.000000	0.0	13566.000000	7176.000000	417.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2017_2 = df2[df2['time'].str.contains("2017-02")] df2017_2 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
18288	2017-02-01 00:00:00+01:00	332.0	838.0	0.0	3806.0	5068.0	199.0	0.0	0.0	0.0
18289	2017-02-01 01:00:00+01:00	334.0	860.0	0.0	3954.0	4567.0	208.0	0.0	0.0	0.0
18290	2017-02-01 02:00:00+01:00	333.0	893.0	0.0	3899.0	4392.0	209.0	0.0	0.0	0.0
18291	2017-02-01 03:00:00+01:00	331.0	939.0	0.0	3879.0	4293.0	209.0	0.0	0.0	0.0
18292	2017-02-01 04:00:00+01:00	332.0	903.0	0.0	4026.0	4344.0	199.0	0.0	0.0	0.0
...
18955	2017-02-28 19:00:00+01:00	389.0	754.0	0.0	4804.0	2627.0	324.0	0.0	0.0	0.0
18956	2017-02-28 20:00:00+01:00	388.0	828.0	0.0	4919.0	2808.0	326.0	0.0	0.0	0.0
18957	2017-02-28 21:00:00+01:00	388.0	871.0	0.0	4661.0	3019.0	329.0	0.0	0.0	0.0
18958	2017-02-28 22:00:00+01:00	391.0	875.0	0.0	4682.0	2985.0	329.0	0.0	0.0	0.0
18959	2017-02-28 23:00:00+01:00	388.0	837.0	0.0	4378.0	2650.0	316.0	0.0	0.0	0.0

672 rows × 29 columns

In []:	df2017_2.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	672.000000	672.000000	672.0	672.000000	672.000000	672.000000	672.0	672.0	672.0	0.0
mean	351.364583	603.415179	0.0	5561.144345	4382.532738	291.401786	0.0	0.0	0.0	NaN
std	39.029197	236.988650	0.0	1938.412409	1444.421345	31.238363	0.0	0.0	0.0	NaN
min	210.000000	0.000000	0.0	3203.000000	1097.000000	185.000000	0.0	0.0	0.0	NaN
25%	342.000000	525.750000	0.0	4224.750000	3404.000000	268.750000	0.0	0.0	0.0	NaN
50%	358.000000	599.000000	0.0	4926.000000	4623.500000	296.000000	0.0	0.0	0.0	NaN
75%	377.000000	821.500000	0.0	6276.750000	5652.750000	315.000000	0.0	0.0	0.0	NaN
max	407.000000	975.000000	0.0	14283.000000	6735.000000	354.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
df2018_2 = df2[df2['time'].str.contains("2018-02")]
df2018_2
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
27048	2018-02-01 00:00:00+01:00	273.0	0.0	0.0	3900.0	1023.0	202.0	0.0	0.0	0.0
27049	2018-02-01 01:00:00+01:00	266.0	0.0	0.0	3753.0	1007.0	202.0	0.0	0.0	0.0
27050	2018-02-01 02:00:00+01:00	268.0	0.0	0.0	3714.0	1033.0	199.0	0.0	0.0	0.0
27051	2018-02-01 03:00:00+01:00	262.0	0.0	0.0	3717.0	1018.0	199.0	0.0	0.0	0.0
27052	2018-02-01 04:00:00+01:00	265.0	0.0	0.0	3692.0	1020.0	197.0	0.0	0.0	0.0
...
27715	2018-02-28 19:00:00+01:00	356.0	0.0	0.0	7479.0	5918.0	307.0	0.0	0.0	0.0
27716	2018-02-28 20:00:00+01:00	360.0	0.0	0.0	7737.0	5890.0	308.0	0.0	0.0	0.0
27717	2018-02-28 21:00:00+01:00	351.0	0.0	0.0	6928.0	5906.0	308.0	0.0	0.0	0.0
27718	2018-02-28 22:00:00+01:00	332.0	0.0	0.0	6239.0	5656.0	303.0	0.0	0.0	0.0
27719	2018-02-28 23:00:00+01:00	335.0	0.0	0.0	5107.0	5103.0	294.0	0.0	0.0	0.0

672 rows × 29 columns

```
df2018_2.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	672.000000	672.000000	672.0	672.000000	672.000000	672.000000	672.0	672.0	672.0	0.0
mean	360.511905	406.950893	0.0	5468.040179	4144.389881	295.900298	0.0	0.0	0.0	NaN
std	39.887993	268.120331	0.0	1726.737880	1466.846267	43.566663	0.0	0.0	0.0	NaN
min	197.000000	0.000000	0.0	3347.000000	901.000000	196.000000	0.0	0.0	0.0	NaN
25%	350.000000	235.000000	0.0	4099.000000	3281.750000	260.000000	0.0	0.0	0.0	NaN
50%	373.000000	398.000000	0.0	4972.500000	4233.500000	313.000000	0.0	0.0	0.0	NaN
75%	388.000000	623.250000	0.0	6541.750000	5380.000000	331.000000	0.0	0.0	0.0	NaN
max	410.000000	923.000000	0.0	12228.000000	6853.000000	360.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2015_3 = df2[df2['time'].str.contains("2015-03")] df2015_3 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
1416	2015-03-01 00:00:00+01:00	484.0	322.0	0.0	5642.0	5969.0	280.0	0.0	0.0	0.0
1417	2015-03-01 01:00:00+01:00	481.0	317.0	0.0	5636.0	5814.0	286.0	0.0	0.0	0.0
1418	2015-03-01 02:00:00+01:00	481.0	306.0	0.0	5285.0	5412.0	275.0	0.0	0.0	0.0
1419	2015-03-01 03:00:00+01:00	479.0	313.0	0.0	5068.0	5427.0	272.0	0.0	0.0	0.0
1420	2015-03-01 04:00:00+01:00	479.0	315.0	0.0	4754.0	5489.0	270.0	0.0	0.0	0.0
...
2154	2015-03-31 19:00:00+02:00	470.0	0.0	0.0	3535.0	2740.0	408.0	0.0	0.0	0.0
2155	2015-03-31 20:00:00+02:00	465.0	0.0	0.0	3616.0	2834.0	408.0	0.0	0.0	0.0
2156	2015-03-31 21:00:00+02:00	448.0	0.0	0.0	3582.0	3141.0	409.0	0.0	0.0	0.0
2157	2015-03-31 22:00:00+02:00	441.0	0.0	0.0	3632.0	2729.0	403.0	0.0	0.0	0.0
2158	2015-03-31 23:00:00+02:00	434.0	0.0	0.0	3781.0	2520.0	377.0	0.0	0.0	0.0

743 rows × 29 columns

In []:	df2015_3.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	468.106326	244.437416	0.0	4614.752355	4233.818304	319.333782	0.0	0.0	0.0	NaN
std	48.253048	355.778435	0.0	1087.721947	1912.194199	55.163628	0.0	0.0	0.0	NaN
min	101.000000	0.000000	0.0	1854.000000	1001.000000	87.000000	0.0	0.0	0.0	NaN
25%	447.000000	0.000000	0.0	3878.500000	2674.500000	277.000000	0.0	0.0	0.0	NaN
50%	470.000000	0.000000	0.0	4368.000000	3807.000000	319.000000	0.0	0.0	0.0	NaN
75%	491.000000	342.500000	0.0	5038.500000	5935.000000	360.000000	0.0	0.0	0.0	NaN
max	577.000000	985.000000	0.0	10036.000000	8031.000000	441.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2016_3 = df2[df2['time'].str.contains("2016-03")] df2016_3 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
10200	2016-03-01 00:00:00+01:00	386.0	0.0	0.0	3900.0	1169.0	273.0	0.0	0.0	0.0
10201	2016-03-01 01:00:00+01:00	377.0	0.0	0.0	3879.0	970.0	268.0	0.0	0.0	0.0
10202	2016-03-01 02:00:00+01:00	362.0	0.0	0.0	3839.0	995.0	272.0	0.0	0.0	0.0
10203	2016-03-01 03:00:00+01:00	364.0	0.0	0.0	3740.0	977.0	271.0	0.0	0.0	0.0
10204	2016-03-01 04:00:00+01:00	368.0	0.0	0.0	3691.0	897.0	265.0	0.0	0.0	0.0
...
10938	2016-03-31 19:00:00+02:00	468.0	0.0	0.0	3651.0	1162.0	362.0	0.0	0.0	0.0
10939	2016-03-31 20:00:00+02:00	469.0	0.0	0.0	3890.0	1258.0	355.0	0.0	0.0	0.0
10940	2016-03-31 21:00:00+02:00	469.0	0.0	0.0	3783.0	1256.0	355.0	0.0	0.0	0.0
10941	2016-03-31 22:00:00+02:00	468.0	0.0	0.0	3797.0	1130.0	348.0	0.0	0.0	0.0
10942	2016-03-31 23:00:00+02:00	468.0	0.0	0.0	3749.0	1130.0	359.0	0.0	0.0	0.0

743 rows × 29 columns

In []:	df2016_3.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	429.744280	173.203230	0.0	4336.594886	2985.886945	294.05249	0.0	0.0	0.0	NaN
std	61.118278	287.462484	0.0	1509.730090	1763.223103	43.82600	0.0	0.0	0.0	NaN
min	279.000000	0.000000	0.0	2139.000000	862.000000	165.00000	0.0	0.0	0.0	NaN
25%	381.000000	0.000000	0.0	3371.000000	1187.000000	268.50000	0.0	0.0	0.0	NaN
50%	437.000000	0.000000	0.0	3897.000000	2828.000000	292.00000	0.0	0.0	0.0	NaN
75%	479.500000	292.000000	0.0	4895.500000	4491.000000	326.00000	0.0	0.0	0.0	NaN
max	534.000000	949.000000	0.0	12635.000000	6593.000000	411.00000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2017_3 = df2[df2['time'].str.contains("2017-03")] df2017_3 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
18960	2017-03-01 00:00:00+01:00	327.0	888.0	0.0	4863.0	6131.0	290.0	0.0	0.0	0.0
18961	2017-03-01 01:00:00+01:00	321.0	888.0	0.0	4918.0	6013.0	290.0	0.0	0.0	0.0
18962	2017-03-01 02:00:00+01:00	341.0	940.0	0.0	4693.0	5766.0	278.0	0.0	0.0	0.0
18963	2017-03-01 03:00:00+01:00	343.0	899.0	0.0	4370.0	5333.0	274.0	0.0	0.0	0.0
18964	2017-03-01 04:00:00+01:00	342.0	908.0	0.0	4380.0	5545.0	283.0	0.0	0.0	0.0
...
19698	2017-03-31 19:00:00+02:00	251.0	402.0	0.0	3962.0	1165.0	333.0	0.0	0.0	0.0
19699	2017-03-31 20:00:00+02:00	250.0	411.0	0.0	4124.0	1313.0	337.0	0.0	0.0	0.0
19700	2017-03-31 21:00:00+02:00	250.0	387.0	0.0	4142.0	1374.0	332.0	0.0	0.0	0.0
19701	2017-03-31 22:00:00+02:00	241.0	294.0	0.0	3839.0	1314.0	323.0	0.0	0.0	0.0
19702	2017-03-31 23:00:00+02:00	239.0	107.0	0.0	3715.0	1336.0	312.0	0.0	0.0	0.0

743 rows × 29 columns

In []:	df2017_3.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	284.402423	335.667564	0.0	5337.418573	2967.84926	302.507402	0.0	0.0	0.0	NaN
std	60.818216	323.278236	0.0	1949.442272	1561.83622	28.266867	0.0	0.0	0.0	NaN
min	175.000000	0.000000	0.0	3312.000000	1165.000000	239.000000	0.0	0.0	0.0	NaN
25%	242.500000	0.000000	0.0	4154.000000	1781.000000	274.000000	0.0	0.0	0.0	NaN
50%	277.000000	328.000000	0.0	4698.000000	2295.000000	309.000000	0.0	0.0	0.0	NaN
75%	339.000000	562.500000	0.0	5517.000000	3956.000000	325.000000	0.0	0.0	0.0	NaN
max	392.000000	999.000000	0.0	14680.000000	6836.000000	364.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2018_3 = df2[df2['time'].str.contains("2018-03")] df2018_3 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
27720	2018-03-01 00:00:00+01:00	292.0	0.0	0.0	3970.0	1103.0	255.0	0.0	0.0	0.0
27721	2018-03-01 01:00:00+01:00	292.0	0.0	0.0	3578.0	1033.0	241.0	0.0	0.0	0.0
27722	2018-03-01 02:00:00+01:00	295.0	0.0	0.0	3394.0	969.0	241.0	0.0	0.0	0.0
27723	2018-03-01 03:00:00+01:00	290.0	0.0	0.0	3337.0	879.0	241.0	0.0	0.0	0.0
27724	2018-03-01 04:00:00+01:00	289.0	0.0	0.0	3591.0	999.0	236.0	0.0	0.0	0.0
...
28458	2018-03-31 19:00:00+02:00	244.0	0.0	0.0	3425.0	1007.0	194.0	0.0	0.0	0.0
28459	2018-03-31 20:00:00+02:00	246.0	0.0	0.0	3815.0	1103.0	194.0	0.0	0.0	0.0
28460	2018-03-31 21:00:00+02:00	259.0	0.0	0.0	4222.0	1340.0	195.0	0.0	0.0	0.0
28461	2018-03-31 22:00:00+02:00	267.0	0.0	0.0	4143.0	1300.0	194.0	0.0	0.0	0.0
28462	2018-03-31 23:00:00+02:00	271.0	0.0	0.0	4022.0	1195.0	188.0	0.0	0.0	0.0

743 rows × 29 columns

In []:	df2018_3.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	307.254374	124.415882	0.0	4936.013459	2157.183042	288.126514	0.0	0.0	0.0	NaN
std	41.145572	243.478477	0.0	1700.369234	1410.681768	48.100609	0.0	0.0	0.0	NaN
min	207.000000	0.000000	0.0	2792.000000	700.000000	167.000000	0.0	0.0	0.0	NaN
25%	282.000000	0.000000	0.0	3929.000000	1222.000000	246.000000	0.0	0.0	0.0	NaN
50%	308.000000	0.000000	0.0	4372.000000	1518.000000	303.000000	0.0	0.0	0.0	NaN
75%	332.500000	0.000000	0.0	5367.000000	2662.000000	327.000000	0.0	0.0	0.0	NaN
max	401.000000	857.000000	0.0	11982.000000	5621.000000	355.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
df2015_4 = df2[df2['time'].str.contains("2015-04")]
df2015_4
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
2159	2015-04-01 00:00:00+02:00	457.0	270.0	0.0	5613.0	6174.0	255.0	0.0	0.0	0.0
2160	2015-04-01 01:00:00+02:00	469.0	273.0	0.0	4989.0	5738.0	251.0	0.0	0.0	0.0
2161	2015-04-01 02:00:00+02:00	466.0	247.0	0.0	4393.0	5248.0	249.0	0.0	0.0	0.0
2162	2015-04-01 03:00:00+02:00	463.0	203.0	0.0	4027.0	4911.0	259.0	0.0	0.0	0.0
2163	2015-04-01 04:00:00+02:00	462.0	182.0	0.0	4020.0	4613.0	260.0	0.0	0.0	0.0
...
2874	2015-04-30 19:00:00+02:00	393.0	891.0	0.0	4062.0	6102.0	397.0	0.0	0.0	0.0
2875	2015-04-30 20:00:00+02:00	388.0	851.0	0.0	4542.0	6110.0	398.0	0.0	0.0	0.0
2876	2015-04-30 21:00:00+02:00	386.0	742.0	0.0	5155.0	6153.0	392.0	0.0	0.0	0.0
2877	2015-04-30 22:00:00+02:00	377.0	679.0	0.0	4804.0	6098.0	397.0	0.0	0.0	0.0
2878	2015-04-30 23:00:00+02:00	377.0	565.0	0.0	4529.0	5521.0	393.0	0.0	0.0	0.0

720 rows × 29 columns

```
df2015_4.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	718.000000	718.000000	718.0	718.000000	718.000000	718.000000	718.0	718.0	718.0	0.0
mean	426.320334	463.119777	0.0	4952.123955	4819.516713	338.781337	0.0	0.0	0.0	NaN
std	56.851839	397.334525	0.0	1136.587141	1667.076972	48.053734	0.0	0.0	0.0	NaN
min	318.000000	0.000000	0.0	2797.000000	1375.000000	208.000000	0.0	0.0	0.0	NaN
25%	388.250000	0.000000	0.0	4211.250000	3517.250000	303.250000	0.0	0.0	0.0	NaN
50%	406.000000	542.500000	0.0	4808.000000	4924.500000	347.000000	0.0	0.0	0.0	NaN
75%	469.000000	902.500000	0.0	5388.000000	5958.500000	375.000000	0.0	0.0	0.0	NaN
max	574.000000	986.000000	0.0	10654.000000	7981.000000	428.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2016_4 = df2[df2['time'].str.contains("2016-04")] df2016_4 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
10943	2016-04-01 00:00:00+02:00	374.0	0.0	0.0	3483.0	1065.0	250.0	0.0	0.0	0.0
10944	2016-04-01 01:00:00+02:00	370.0	0.0	0.0	3384.0	1035.0	245.0	0.0	0.0	0.0
10945	2016-04-01 02:00:00+02:00	372.0	0.0	0.0	3365.0	962.0	245.0	0.0	0.0	0.0
10946	2016-04-01 03:00:00+02:00	369.0	0.0	0.0	3318.0	998.0	239.0	0.0	0.0	0.0
10947	2016-04-01 04:00:00+02:00	370.0	0.0	0.0	3430.0	1124.0	237.0	0.0	0.0	0.0
...
11658	2016-04-30 19:00:00+02:00	243.0	0.0	0.0	2981.0	1001.0	184.0	0.0	0.0	0.0
11659	2016-04-30 20:00:00+02:00	247.0	0.0	0.0	3045.0	980.0	188.0	0.0	0.0	0.0
11660	2016-04-30 21:00:00+02:00	251.0	0.0	0.0	3113.0	1042.0	193.0	0.0	0.0	0.0
11661	2016-04-30 22:00:00+02:00	256.0	0.0	0.0	3073.0	1027.0	185.0	0.0	0.0	0.0
11662	2016-04-30 23:00:00+02:00	252.0	0.0	0.0	3448.0	1007.0	196.0	0.0	0.0	0.0

720 rows × 29 columns

In []:	df2016_4.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	286.890278	143.570833	0.0	4263.481944	2226.186111	259.887500	0.0	0.0	0.0	NaN
std	86.579542	302.080515	0.0	1477.267774	1611.082243	51.375894	0.0	0.0	0.0	NaN
min	167.000000	0.000000	0.0	2451.000000	818.000000	139.000000	0.0	0.0	0.0	NaN
25%	220.000000	0.000000	0.0	3340.500000	1129.500000	217.750000	0.0	0.0	0.0	NaN
50%	246.000000	0.000000	0.0	3867.000000	1528.500000	266.000000	0.0	0.0	0.0	NaN
75%	352.000000	0.000000	0.0	4722.000000	2212.750000	288.000000	0.0	0.0	0.0	NaN
max	546.000000	967.000000	0.0	12017.000000	6320.000000	424.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

--	--	--	--	--	--	--	--	--	--	--

In []:

```
df2017_4 = df2[df2['time'].str.contains("2017-04")]
df2017_4
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
19703	2017-04-01 00:00:00+02:00	362.0	970.0	0.0	4983.0	6732.0	292.0	0.0	0.0	0.0
19704	2017-04-01 01:00:00+02:00	359.0	980.0	0.0	4733.0	6514.0	288.0	0.0	0.0	0.0
19705	2017-04-01 02:00:00+02:00	353.0	898.0	0.0	4727.0	6333.0	288.0	0.0	0.0	0.0
19706	2017-04-01 03:00:00+02:00	346.0	902.0	0.0	4655.0	5983.0	289.0	0.0	0.0	0.0
19707	2017-04-01 04:00:00+02:00	347.0	888.0	0.0	4656.0	5833.0	281.0	0.0	0.0	0.0
...
20418	2017-04-30 19:00:00+02:00	257.0	0.0	0.0	2947.0	1156.0	176.0	0.0	0.0	0.0
20419	2017-04-30 20:00:00+02:00	260.0	0.0	0.0	3057.0	1192.0	181.0	0.0	0.0	0.0
20420	2017-04-30 21:00:00+02:00	266.0	0.0	0.0	3174.0	1231.0	183.0	0.0	0.0	0.0
20421	2017-04-30 22:00:00+02:00	256.0	0.0	0.0	3375.0	1235.0	183.0	0.0	0.0	0.0
20422	2017-04-30 23:00:00+02:00	239.0	0.0	0.0	3586.0	1396.0	190.0	0.0	0.0	0.0

720 rows × 29 columns

--	--	--	--	--	--	--	--	--	--	--

In []:

```
df2017_4.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	280.761111	420.115278	0.0	5169.840278	3119.322222	261.490278	0.0	0.0	0.0	NaN
std	57.848381	275.878406	0.0	2504.057579	1549.673825	41.949999	0.0	0.0	0.0	NaN
min	194.000000	0.000000	0.0	2736.000000	1042.000000	168.000000	0.0	0.0	0.0	NaN
25%	235.000000	226.750000	0.0	3741.750000	1922.250000	233.000000	0.0	0.0	0.0	NaN
50%	257.000000	421.500000	0.0	4331.500000	2820.500000	260.000000	0.0	0.0	0.0	NaN
75%	337.000000	531.000000	0.0	5052.500000	3754.750000	287.000000	0.0	0.0	0.0	NaN
max	396.000000	993.000000	0.0	18149.000000	6915.000000	363.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2018_4 = df2[df2['time'].str.contains("2018-04")] df2018_4 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
28463	2018-04-01 00:00:00+02:00	277.0	0.0	0.0	3969.0	1052.0	284.0	0.0	0.0	0.0
28464	2018-04-01 01:00:00+02:00	269.0	0.0	0.0	3652.0	1033.0	191.0	0.0	0.0	0.0
28465	2018-04-01 02:00:00+02:00	271.0	0.0	0.0	3519.0	1005.0	183.0	0.0	0.0	0.0
28466	2018-04-01 03:00:00+02:00	280.0	0.0	0.0	3634.0	1029.0	181.0	0.0	0.0	0.0
28467	2018-04-01 04:00:00+02:00	280.0	0.0	0.0	3511.0	1033.0	177.0	0.0	0.0	0.0
...
29178	2018-04-30 19:00:00+02:00	338.0	0.0	0.0	3585.0	1746.0	286.0	0.0	0.0	0.0
29179	2018-04-30 20:00:00+02:00	337.0	0.0	0.0	3851.0	1741.0	284.0	0.0	0.0	0.0
29180	2018-04-30 21:00:00+02:00	332.0	0.0	0.0	4499.0	1847.0	290.0	0.0	0.0	0.0
29181	2018-04-30 22:00:00+02:00	332.0	0.0	0.0	4350.0	1712.0	287.0	0.0	0.0	0.0
29182	2018-04-30 23:00:00+02:00	335.0	0.0	0.0	4331.0	1666.0	284.0	0.0	0.0	0.0

720 rows × 29 columns

In []:	df2018_4.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	299.393056	133.979167	0.0	4745.273611	2775.879167	257.629167	0.0	0.0	0.0	NaN
std	53.001558	251.734784	0.0	1759.070145	1368.301099	47.308651	0.0	0.0	0.0	NaN
min	214.000000	0.000000	0.0	2497.000000	1005.000000	106.000000	0.0	0.0	0.0	NaN
25%	247.000000	0.000000	0.0	3644.500000	1604.750000	226.750000	0.0	0.0	0.0	NaN
50%	306.000000	0.000000	0.0	4184.000000	2467.500000	256.000000	0.0	0.0	0.0	NaN
75%	346.000000	0.000000	0.0	5156.500000	3592.750000	296.000000	0.0	0.0	0.0	NaN
max	402.000000	766.000000	0.0	12565.000000	5846.000000	342.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
df2015_5 = df2[df2['time'].str.contains("2015-05")]
df2015_5
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
2879	2015-05-01 00:00:00+02:00	494.0	318.0	0.0	5316.0	6054.0	257.0	0.0	0.0	0.0
2880	2015-05-01 01:00:00+02:00	496.0	317.0	0.0	5341.0	5941.0	258.0	0.0	0.0	0.0
2881	2015-05-01 02:00:00+02:00	493.0	351.0	0.0	4926.0	6029.0	252.0	0.0	0.0	0.0
2882	2015-05-01 03:00:00+02:00	493.0	410.0	0.0	4647.0	5873.0	263.0	0.0	0.0	0.0
2883	2015-05-01 04:00:00+02:00	493.0	514.0	0.0	4305.0	5879.0	265.0	0.0	0.0	0.0
...
3618	2015-05-31 19:00:00+02:00	534.0	580.0	0.0	3601.0	4807.0	260.0	0.0	0.0	0.0
3619	2015-05-31 20:00:00+02:00	534.0	695.0	0.0	3756.0	5043.0	260.0	0.0	0.0	0.0
3620	2015-05-31 21:00:00+02:00	540.0	829.0	0.0	3947.0	5267.0	259.0	0.0	0.0	0.0
3621	2015-05-31 22:00:00+02:00	540.0	931.0	0.0	4337.0	5530.0	260.0	0.0	0.0	0.0
3622	2015-05-31 23:00:00+02:00	541.0	918.0	0.0	4192.0	5473.0	258.0	0.0	0.0	0.0

744 rows × 29 columns

```
df2015_5.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	503.569892	374.280914	0.0	4415.349462	4019.612903	332.567204	0.0	0.0	0.0	NaN
std	57.413618	417.889809	0.0	934.243399	1889.671809	53.402389	0.0	0.0	0.0	NaN
min	306.000000	0.000000	0.0	2901.000000	803.000000	129.000000	0.0	0.0	0.0	NaN
25%	479.000000	0.000000	0.0	3841.250000	2466.500000	286.750000	0.0	0.0	0.0	NaN
50%	510.000000	0.000000	0.0	4175.000000	3793.000000	340.000000	0.0	0.0	0.0	NaN
75%	545.000000	869.000000	0.0	4700.750000	5343.000000	381.000000	0.0	0.0	0.0	NaN
max	585.000000	989.000000	0.0	8673.000000	7957.000000	422.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2016_5 = df2[df2['time'].str.contains("2016-05")] df2016_5 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
11663	2016-05-01 00:00:00+02:00	413.0	0.0	0.0	3790.0	1385.0	362.0	0.0	0.0	0.0
11664	2016-05-01 01:00:00+02:00	411.0	0.0	0.0	3770.0	1561.0	356.0	0.0	0.0	0.0
11665	2016-05-01 02:00:00+02:00	383.0	0.0	0.0	3723.0	1667.0	357.0	0.0	0.0	0.0
11666	2016-05-01 03:00:00+02:00	377.0	0.0	0.0	3463.0	1646.0	351.0	0.0	0.0	0.0
11667	2016-05-01 04:00:00+02:00	372.0	0.0	0.0	3781.0	1644.0	348.0	0.0	0.0	0.0
...
12402	2016-05-31 19:00:00+02:00	328.0	0.0	0.0	4012.0	1844.0	313.0	0.0	0.0	0.0
12403	2016-05-31 20:00:00+02:00	328.0	0.0	0.0	4029.0	1733.0	311.0	0.0	0.0	0.0
12404	2016-05-31 21:00:00+02:00	329.0	0.0	0.0	4574.0	1764.0	315.0	0.0	0.0	0.0
12405	2016-05-31 22:00:00+02:00	332.0	0.0	0.0	5130.0	2061.0	318.0	0.0	0.0	0.0
12406	2016-05-31 23:00:00+02:00	329.0	0.0	0.0	4661.0	2023.0	313.0	0.0	0.0	0.0

744 rows × 29 columns

In []:	df2016_5.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	337.202957	179.002688	0.0	4508.034946	2226.651882	277.915323	0.0	0.0	0.0	NaN
std	47.875123	288.272130	0.0	1406.654443	1677.710361	47.955785	0.0	0.0	0.0	NaN
min	214.000000	0.000000	0.0	1938.000000	576.000000	189.000000	0.0	0.0	0.0	NaN
25%	315.000000	0.000000	0.0	3730.250000	966.000000	230.000000	0.0	0.0	0.0	NaN
50%	328.000000	0.000000	0.0	4217.000000	1716.000000	286.000000	0.0	0.0	0.0	NaN
75%	347.000000	299.250000	0.0	4956.500000	2369.250000	305.000000	0.0	0.0	0.0	NaN
max	538.000000	990.000000	0.0	11232.000000	6535.000000	434.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2017_5 = df2[df2['time'].str.contains("2017-05")] df2017_5 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
20423	2017-05-01 00:00:00+02:00	359.0	919.0	0.0	4689.0	6387.0	304.0	0.0	0.0	0.0
20424	2017-05-01 01:00:00+02:00	361.0	893.0	0.0	4585.0	6041.0	304.0	0.0	0.0	0.0
20425	2017-05-01 02:00:00+02:00	359.0	909.0	0.0	4590.0	6019.0	299.0	0.0	0.0	0.0
20426	2017-05-01 03:00:00+02:00	353.0	923.0	0.0	4475.0	5710.0	295.0	0.0	0.0	0.0
20427	2017-05-01 04:00:00+02:00	355.0	913.0	0.0	4450.0	4930.0	293.0	0.0	0.0	0.0
...
21162	2017-05-31 19:00:00+02:00	337.0	836.0	0.0	6502.0	6605.0	320.0	0.0	0.0	0.0
21163	2017-05-31 20:00:00+02:00	337.0	867.0	0.0	6411.0	6521.0	318.0	0.0	0.0	0.0
21164	2017-05-31 21:00:00+02:00	338.0	849.0	0.0	6631.0	6568.0	319.0	0.0	0.0	0.0
21165	2017-05-31 22:00:00+02:00	336.0	847.0	0.0	6753.0	6552.0	321.0	0.0	0.0	0.0
21166	2017-05-31 23:00:00+02:00	338.0	832.0	0.0	6648.0	6437.0	321.0	0.0	0.0	0.0

744 rows × 29 columns

In []:	df2017_5.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	353.504032	500.770161	0.0	5493.271505	4428.919355	291.920699	0.0	0.0	0.0	NaN
std	32.993208	245.659871	0.0	2571.092763	1626.989616	37.900847	0.0	0.0	0.0	NaN
min	220.000000	0.000000	0.0	2974.000000	1007.000000	141.000000	0.0	0.0	0.0	NaN
25%	346.000000	396.500000	0.0	4011.000000	3185.500000	262.000000	0.0	0.0	0.0	NaN
50%	359.500000	527.500000	0.0	4735.500000	4482.000000	299.500000	0.0	0.0	0.0	NaN
75%	374.000000	594.000000	0.0	6361.750000	5949.000000	320.000000	0.0	0.0	0.0	NaN
max	403.000000	986.000000	0.0	20034.000000	6853.000000	370.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2018_5 = df2[df2['time'].str.contains("2018-05")] df2018_5 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
29183	2018-05-01 00:00:00+02:00	285.0	0.0	0.0	3867.0	1095.0	294.0	0.0	0.0	0.0
29184	2018-05-01 01:00:00+02:00	282.0	0.0	0.0	3939.0	1099.0	239.0	0.0	0.0	0.0
29185	2018-05-01 02:00:00+02:00	283.0	0.0	0.0	3717.0	1097.0	237.0	0.0	0.0	0.0
29186	2018-05-01 03:00:00+02:00	279.0	0.0	0.0	3518.0	1043.0	233.0	0.0	0.0	0.0
29187	2018-05-01 04:00:00+02:00	283.0	0.0	0.0	3539.0	1039.0	234.0	0.0	0.0	0.0
...
29922	2018-05-31 19:00:00+02:00	351.0	421.0	0.0	7121.0	5545.0	304.0	0.0	0.0	0.0
29923	2018-05-31 20:00:00+02:00	350.0	434.0	0.0	7182.0	5545.0	305.0	0.0	0.0	0.0
29924	2018-05-31 21:00:00+02:00	351.0	442.0	0.0	7414.0	5653.0	306.0	0.0	0.0	0.0
29925	2018-05-31 22:00:00+02:00	355.0	442.0	0.0	7558.0	5655.0	306.0	0.0	0.0	0.0
29926	2018-05-31 23:00:00+02:00	339.0	399.0	0.0	7604.0	5639.0	305.0	0.0	0.0	0.0

744 rows × 29 columns

In []:	df2015_6 = df2[df2['time'].str.contains("2015-06")] df2015_6 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	
3623	2015-06-01 00:00:00+02:00	502.0	633.0	0.0	5601.0	6435.0	283.0	0.0	0.0	0.0	.
3624	2015-06-01 01:00:00+02:00	495.0	642.0	0.0	5247.0	6107.0	281.0	0.0	0.0	0.0	.
3625	2015-06-01 02:00:00+02:00	495.0	638.0	0.0	4660.0	5577.0	276.0	0.0	0.0	0.0	.
3626	2015-06-01 03:00:00+02:00	495.0	529.0	0.0	4360.0	5136.0	275.0	0.0	0.0	0.0	.
3627	2015-06-01 04:00:00+02:00	491.0	385.0	0.0	4290.0	4951.0	275.0	0.0	0.0	0.0	.
...
4338	2015-06-30 19:00:00+02:00	497.0	953.0	0.0	5304.0	8126.0	388.0	0.0	0.0	0.0	.
4339	2015-06-30 20:00:00+02:00	500.0	916.0	0.0	5457.0	8102.0	399.0	0.0	0.0	0.0	.
4340	2015-06-30 21:00:00+02:00	504.0	913.0	0.0	6045.0	8124.0	390.0	0.0	0.0	0.0	.
4341	2015-06-30 22:00:00+02:00	508.0	882.0	0.0	6175.0	8003.0	399.0	0.0	0.0	0.0	.
4342	2015-06-30 23:00:00+02:00	508.0	882.0	0.0	5713.0	7930.0	385.0	0.0	0.0	0.0	.

720 rows × 29 columns

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated	
count	719.000000	719.000000	719.0	719.000000	719.000000	719.000000	719.0	719.0	719.0	0.0	
mean	485.998609	665.162726	0.0	4934.930459	6207.095967	319.229485	0.0	0.0	0.0	NaN	
std	44.492762	278.050075	0.0	1117.826707	1569.125048	50.380103	0.0	0.0	0.0	NaN	
min	307.000000	0.000000	0.0	3191.000000	1106.000000	215.000000	0.0	0.0	0.0	NaN	
25%	471.000000	573.500000	0.0	4177.500000	5474.000000	277.000000	0.0	0.0	0.0	NaN	
50%	487.000000	653.000000	0.0	4618.000000	6486.000000	325.000000	0.0	0.0	0.0	NaN	
75%	507.500000	915.000000	0.0	5410.500000	7443.500000	359.500000	0.0	0.0	0.0	NaN	
max	583.000000	983.000000	0.0	9199.000000	8303.000000	429.000000	0.0	0.0	0.0	NaN	

8 rows × 28 columns

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated	
In []:	df2016_6 = df2[df2['time'].str.contains("2016-06")]											
												# Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

		time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
12407		2016-06-01 00:00:00+02:00	376.0	0.0	0.0	3673.0	1354.0	360.0	0.0	0.0	0.0
12408		2016-06-01 01:00:00+02:00	369.0	0.0	0.0	3620.0	1130.0	354.0	0.0	0.0	0.0
12409		2016-06-01 02:00:00+02:00	371.0	0.0	0.0	3478.0	983.0	358.0	0.0	0.0	0.0
12410		2016-06-01 03:00:00+02:00	367.0	0.0	0.0	3654.0	951.0	354.0	0.0	0.0	0.0
12411		2016-06-01 04:00:00+02:00	370.0	0.0	0.0	3626.0	943.0	356.0	0.0	0.0	0.0
...
13122		2016-06-30 19:00:00+02:00	340.0	646.0	0.0	6622.0	5534.0	318.0	0.0	0.0	0.0
13123		2016-06-30 20:00:00+02:00	347.0	643.0	0.0	6367.0	5544.0	320.0	0.0	0.0	0.0
13124		2016-06-30 21:00:00+02:00	350.0	643.0	0.0	6782.0	5578.0	320.0	0.0	0.0	0.0
13125		2016-06-30 22:00:00+02:00	350.0	644.0	0.0	7250.0	5500.0	318.0	0.0	0.0	0.0
13126		2016-06-30 23:00:00+02:00	347.0	654.0	0.0	6758.0	5342.0	318.0	0.0	0.0	0.0

720 rows × 29 columns

In []:	df2016_6.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	---------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	340.337500	175.600000	0.0	4994.327778	2877.080556	289.829167	0.0	0.0	0.0	NaN
std	39.726936	301.926344	0.0	1501.655962	1790.711862	34.856551	0.0	0.0	0.0	NaN
min	223.000000	0.000000	0.0	2258.000000	799.000000	189.000000	0.0	0.0	0.0	NaN
25%	324.000000	0.000000	0.0	4143.750000	1338.750000	265.000000	0.0	0.0	0.0	NaN
50%	337.000000	0.000000	0.0	4691.500000	2200.500000	296.500000	0.0	0.0	0.0	NaN
75%	347.000000	313.500000	0.0	5420.500000	4274.500000	314.000000	0.0	0.0	0.0	NaN
max	531.000000	981.000000	0.0	10849.000000	6749.000000	381.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2017_6 = df2[df2['time'].str.contains("2017-06")] df2017_6 # Monthly Datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
21167	2017-06-01 00:00:00+02:00	356.0	604.0	0.0	5670.0	6236.0	287.0	0.0	0.0	0.0
21168	2017-06-01 01:00:00+02:00	351.0	596.0	0.0	5565.0	5902.0	286.0	0.0	0.0	0.0
21169	2017-06-01 02:00:00+02:00	349.0	634.0	0.0	5399.0	5557.0	282.0	0.0	0.0	0.0
21170	2017-06-01 03:00:00+02:00	349.0	600.0	0.0	5208.0	5090.0	280.0	0.0	0.0	0.0
21171	2017-06-01 04:00:00+02:00	352.0	605.0	0.0	4949.0	4900.0	277.0	0.0	0.0	0.0
...
21882	2017-06-30 19:00:00+02:00	358.0	557.0	0.0	4541.0	3154.0	346.0	0.0	0.0	0.0
21883	2017-06-30 20:00:00+02:00	355.0	563.0	0.0	4686.0	3177.0	347.0	0.0	0.0	0.0
21884	2017-06-30 21:00:00+02:00	350.0	607.0	0.0	4983.0	3367.0	347.0	0.0	0.0	0.0
21885	2017-06-30 22:00:00+02:00	359.0	612.0	0.0	5264.0	3494.0	349.0	0.0	0.0	0.0
21886	2017-06-30 23:00:00+02:00	355.0	583.0	0.0	5418.0	3467.0	346.0	0.0	0.0	0.0

720 rows × 29 columns

In []: df2017_6.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	339.979167	478.811111	0.0	7194.280556	5091.152778	299.393056	0.0	0.0	0.0	NaN
std	36.219250	305.371145	0.0	2855.328760	1778.218245	37.009824	0.0	0.0	0.0	NaN
min	218.000000	0.000000	0.0	3282.000000	947.000000	196.000000	0.0	0.0	0.0	NaN
25%	325.000000	265.000000	0.0	4799.500000	3376.750000	270.000000	0.0	0.0	0.0	NaN
50%	349.000000	466.500000	0.0	6551.000000	5764.000000	309.000000	0.0	0.0	0.0	NaN
75%	360.000000	783.750000	0.0	9252.000000	6585.250000	331.000000	0.0	0.0	0.0	NaN
max	405.000000	985.000000	0.0	16266.000000	7472.000000	352.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []: df2018_6 = df2[df2['time'].str.contains("2018-06")]
df2018_6
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
29927	2018-06-01 00:00:00+02:00	298.0	0.0	0.0	5078.0	1636.0	325.0	0.0	0.0	0.0
29928	2018-06-01 01:00:00+02:00	286.0	0.0	0.0	4699.0	1466.0	333.0	0.0	0.0	0.0
29929	2018-06-01 02:00:00+02:00	285.0	0.0	0.0	4573.0	1342.0	322.0	0.0	0.0	0.0
29930	2018-06-01 03:00:00+02:00	285.0	0.0	0.0	4791.0	1531.0	272.0	0.0	0.0	0.0
29931	2018-06-01 04:00:00+02:00	278.0	0.0	0.0	4635.0	1613.0	271.0	0.0	0.0	0.0
...
30642	2018-06-30 19:00:00+02:00	345.0	618.0	0.0	5523.0	3291.0	239.0	0.0	0.0	0.0
30643	2018-06-30 20:00:00+02:00	358.0	635.0	0.0	5765.0	3449.0	239.0	0.0	0.0	0.0
30644	2018-06-30 21:00:00+02:00	362.0	656.0	0.0	5989.0	3478.0	239.0	0.0	0.0	0.0
30645	2018-06-30 22:00:00+02:00	360.0	663.0	0.0	5996.0	3480.0	239.0	0.0	0.0	0.0
30646	2018-06-30 23:00:00+02:00	363.0	623.0	0.0	6124.0	3330.0	239.0	0.0	0.0	0.0

720 rows × 29 columns

In []: df2018_6.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	351.733333	333.998611	0.0	5807.555556	3059.616667	285.120833	0.0	0.0	0.0	NaN
std	32.707470	303.016432	0.0	1483.951041	1328.116035	34.838121	0.0	0.0	0.0	NaN
min	222.000000	0.000000	0.0	0.000000	1088.000000	189.000000	0.0	0.0	0.0	NaN
25%	345.000000	0.000000	0.0	4803.750000	1851.000000	263.750000	0.0	0.0	0.0	NaN
50%	360.000000	410.000000	0.0	5446.000000	3057.500000	293.000000	0.0	0.0	0.0	NaN
75%	370.000000	631.250000	0.0	6530.750000	3940.500000	310.000000	0.0	0.0	0.0	NaN
max	413.000000	938.000000	0.0	10969.000000	6151.000000	364.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []: df2016_7 = df2[df2['time'].str.contains("2016-07")]
df2016_7
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
13127	2016-07-01 00:00:00+02:00	350.0	0.0	0.0	3616.0	1108.0	332.0	0.0	0.0	0.0
13128	2016-07-01 01:00:00+02:00	345.0	0.0	0.0	3657.0	1105.0	331.0	0.0	0.0	0.0
13129	2016-07-01 02:00:00+02:00	336.0	0.0	0.0	3660.0	1111.0	332.0	0.0	0.0	0.0
13130	2016-07-01 03:00:00+02:00	327.0	0.0	0.0	3600.0	1091.0	328.0	0.0	0.0	0.0
13131	2016-07-01 04:00:00+02:00	327.0	0.0	0.0	3666.0	1123.0	327.0	0.0	0.0	0.0
...
13866	2016-07-31 19:00:00+02:00	373.0	0.0	0.0	3679.0	2824.0	206.0	0.0	0.0	0.0
13867	2016-07-31 20:00:00+02:00	382.0	0.0	0.0	3859.0	3355.0	208.0	0.0	0.0	0.0
13868	2016-07-31 21:00:00+02:00	384.0	0.0	0.0	4095.0	3804.0	213.0	0.0	0.0	0.0
13869	2016-07-31 22:00:00+02:00	381.0	0.0	0.0	4252.0	3986.0	220.0	0.0	0.0	0.0
13870	2016-07-31 23:00:00+02:00	384.0	0.0	0.0	4247.0	3718.0	222.0	0.0	0.0	0.0

744 rows × 29 columns

	count	mean	std	min	25%	50%	75%	max	hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	742.000000	743.0	743.0	743.0
mean	351.889637	398.815612	0.0	5475.643338	4003.916555	286.690027	0.0	0.0	0.0
std	42.165969	385.267866	0.0	1705.003608	1746.961208	41.032799	0.0	0.0	0.0
min	214.000000	0.000000	0.0	2551.000000	745.000000	197.000000	0.0	0.0	0.0
25%	327.000000	0.000000	0.0	4003.500000	2862.500000	258.250000	0.0	0.0	0.0
50%	357.000000	405.000000	0.0	5125.000000	4162.000000	292.000000	0.0	0.0	0.0
75%	379.500000	818.000000	0.0	6861.000000	5497.000000	315.000000	0.0	0.0	0.0
max	475.000000	979.000000	0.0	9899.000000	6735.000000	439.000000	0.0	0.0	NaN

8 rows × 28 columns

	count	mean	std	min	25%	50%	75%	max	hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	742.000000	743.0	743.0	743.0
mean	351.889637	398.815612	0.0	5475.643338	4003.916555	286.690027	0.0	0.0	0.0
std	42.165969	385.267866	0.0	1705.003608	1746.961208	41.032799	0.0	0.0	0.0
min	214.000000	0.000000	0.0	2551.000000	745.000000	197.000000	0.0	0.0	0.0
25%	327.000000	0.000000	0.0	4003.500000	2862.500000	258.250000	0.0	0.0	0.0
50%	357.000000	405.000000	0.0	5125.000000	4162.000000	292.000000	0.0	0.0	0.0
75%	379.500000	818.000000	0.0	6861.000000	5497.000000	315.000000	0.0	0.0	0.0
max	475.000000	979.000000	0.0	9899.000000	6735.000000	439.000000	0.0	0.0	NaN

```
In [ ]: df2015_7 = df2[df2['time'].str.contains("2015-07")]
df2015_7
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
4343	2015-07-01 00:00:00+02:00	494.0	890.0	0.0	5277.0	6703.0	272.0	0.0	0.0	0.0	.
4344	2015-07-01 01:00:00+02:00	497.0	936.0	0.0	5253.0	6619.0	274.0	0.0	0.0	0.0	.
4345	2015-07-01 02:00:00+02:00	487.0	895.0	0.0	5279.0	6450.0	275.0	0.0	0.0	0.0	.
4346	2015-07-01 03:00:00+02:00	484.0	899.0	0.0	5128.0	6366.0	275.0	0.0	0.0	0.0	.
4347	2015-07-01 04:00:00+02:00	489.0	977.0	0.0	5298.0	6363.0	273.0	0.0	0.0	0.0	.
...
5082	2015-07-31 19:00:00+02:00	530.0	570.0	0.0	4859.0	7539.0	406.0	0.0	0.0	0.0	.
5083	2015-07-31 20:00:00+02:00	535.0	486.0	0.0	4916.0	6968.0	416.0	0.0	0.0	0.0	.
5084	2015-07-31 21:00:00+02:00	559.0	528.0	0.0	5489.0	7336.0	415.0	0.0	0.0	0.0	.
5085	2015-07-31 22:00:00+02:00	565.0	529.0	0.0	5148.0	7314.0	418.0	0.0	0.0	0.0	.
5086	2015-07-31 23:00:00+02:00	558.0	469.0	0.0	4914.0	7264.0	416.0	0.0	0.0	0.0	.

744 rows × 29 columns

```
In [1]: df2015[7].describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	512.487903	684.220430	0.0	6529.490591	6748.469086	360.233871	0.0	0.0	0.0	NaN
std	42.861885	298.294673	0.0	2205.578526	1401.111496	49.976405	0.0	0.0	0.0	NaN
min	316.000000	0.000000	0.0	3374.000000	1516.000000	242.000000	0.0	0.0	0.0	NaN
25%	497.000000	565.000000	0.0	4639.750000	6325.000000	316.000000	0.0	0.0	0.0	NaN
50%	523.000000	791.000000	0.0	5987.500000	7260.500000	381.500000	0.0	0.0	0.0	NaN
75%	541.000000	936.000000	0.0	8098.750000	7661.500000	402.000000	0.0	0.0	0.0	NaN
max	584.000000	988.000000	0.0	11895.000000	8313.000000	427.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2017_7 = df2[df2['time'].str.contains("2017-07")]
df2017_7
# Monthly Datasets ("Year_Month" = "0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
21887	2017-07-01 00:00:00+02:00	360.0	590.0	0.0	5615.0	5780.0	246.0	0.0	0.0	0.0
21888	2017-07-01 01:00:00+02:00	357.0	593.0	0.0	4770.0	5402.0	236.0	0.0	0.0	0.0
21889	2017-07-01 02:00:00+02:00	357.0	586.0	0.0	4724.0	5101.0	235.0	0.0	0.0	0.0
21890	2017-07-01 03:00:00+02:00	358.0	585.0	0.0	4507.0	4753.0	235.0	0.0	0.0	0.0
21891	2017-07-01 04:00:00+02:00	357.0	585.0	0.0	4775.0	4649.0	242.0	0.0	0.0	0.0
...
22626	2017-07-31 19:00:00+02:00	383.0	726.0	0.0	9351.0	5230.0	331.0	0.0	0.0	0.0
22627	2017-07-31 20:00:00+02:00	362.0	682.0	0.0	8958.0	5240.0	332.0	0.0	0.0	0.0
22628	2017-07-31 21:00:00+02:00	355.0	734.0	0.0	9205.0	5241.0	341.0	0.0	0.0	0.0
22629	2017-07-31 22:00:00+02:00	359.0	740.0	0.0	9224.0	5232.0	348.0	0.0	0.0	0.0
22630	2017-07-31 23:00:00+02:00	360.0	644.0	0.0	7892.0	4982.0	349.0	0.0	0.0	0.0

744 rows × 29 columns

In []:	df2017_7.describe() # Description tables of monthly datasets ("Year_Month" = "0000_0")
---------	----------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	367.706989	650.677419	0.0	7366.073925	4561.283602	308.200269	0.0	0.0	0.0	NaN
std	36.910829	231.481496	0.0	2242.910240	1101.879716	41.147621	0.0	0.0	0.0	NaN
min	225.000000	0.000000	0.0	3325.000000	1450.000000	210.000000	0.0	0.0	0.0	NaN
25%	356.000000	476.500000	0.0	5617.000000	3828.250000	271.750000	0.0	0.0	0.0	NaN
50%	376.000000	633.000000	0.0	6913.500000	4671.500000	323.000000	0.0	0.0	0.0	NaN
75%	393.000000	867.000000	0.0	9392.000000	5402.750000	343.000000	0.0	0.0	0.0	NaN
max	424.000000	995.000000	0.0	13609.000000	6825.000000	365.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2018_7 = df2[df2['time'].str.contains("2018-07")] df2018_7 # Monthly Datasets ("Year_Month" = "0000_0")
---------	-----------------------------------------------------------------------------------------------------------

Out[]:

		time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
30647		2018-07-01 00:00:00+02:00	289.0	0.0	0.0	5175.0	1867.0	252.0	0.0	0.0	0.0
30648		2018-07-01 01:00:00+02:00	273.0	0.0	0.0	5361.0	1954.0	206.0	0.0	0.0	0.0
30649		2018-07-01 02:00:00+02:00	268.0	0.0	0.0	4835.0	1919.0	203.0	0.0	0.0	0.0
30650		2018-07-01 03:00:00+02:00	266.0	0.0	0.0	4637.0	1916.0	203.0	0.0	0.0	0.0
30651		2018-07-01 04:00:00+02:00	266.0	0.0	0.0	4968.0	2013.0	204.0	0.0	0.0	0.0
...
31386		2018-07-31 19:00:00+02:00	391.0	441.0	0.0	9640.0	5207.0	326.0	0.0	0.0	0.0
31387		2018-07-31 20:00:00+02:00	391.0	441.0	0.0	7630.0	5251.0	320.0	0.0	0.0	0.0
31388		2018-07-31 21:00:00+02:00	396.0	441.0	0.0	7307.0	5237.0	328.0	0.0	0.0	0.0
31389		2018-07-31 22:00:00+02:00	395.0	424.0	0.0	7069.0	5211.0	326.0	0.0	0.0	0.0
31390		2018-07-31 23:00:00+02:00	396.0	439.0	0.0	6380.0	5073.0	324.0	0.0	0.0	0.0

744 rows × 29 columns

In []:	df2018_7.describe() # Description tables of monthly datasets ("Year_Month" = "0000_0")										

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	367.220727	506.368775	0.0	5865.691790	4014.707941	281.545087	0.0	0.0	0.0	NaN
std	34.900007	233.993460	0.0	1536.562105	1178.254070	38.292502	0.0	0.0	0.0	NaN
min	231.000000	0.000000	0.0	3333.000000	747.000000	187.000000	0.0	0.0	0.0	NaN
25%	351.000000	422.000000	0.0	4852.500000	3056.500000	254.000000	0.0	0.0	0.0	NaN
50%	379.000000	614.000000	0.0	5474.000000	4256.000000	289.000000	0.0	0.0	0.0	NaN
75%	392.000000	663.000000	0.0	6602.000000	4891.500000	310.000000	0.0	0.0	0.0	NaN
max	410.000000	957.000000	0.0	12551.000000	6291.000000	361.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2015_8 = df2[df2['time'].str.contains("2015-08")]									

df2015_8
Monthly Datasets ("Year_Month" = "0000_0")

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
5087	2015-08-01 00:00:00+02:00	521.0	882.0	0.0	6796.0	7651.0	323.0	0.0	0.0	0.0
5088	2015-08-01 01:00:00+02:00	527.0	934.0	0.0	6253.0	7663.0	317.0	0.0	0.0	0.0
5089	2015-08-01 02:00:00+02:00	522.0	946.0	0.0	5841.0	7710.0	310.0	0.0	0.0	0.0
5090	2015-08-01 03:00:00+02:00	523.0	953.0	0.0	5437.0	7789.0	303.0	0.0	0.0	0.0
5091	2015-08-01 04:00:00+02:00	522.0	959.0	0.0	5383.0	7815.0	316.0	0.0	0.0	0.0
...
5826	2015-08-31 19:00:00+02:00	504.0	958.0	0.0	7437.0	7890.0	367.0	0.0	0.0	0.0
5827	2015-08-31 20:00:00+02:00	498.0	951.0	0.0	8322.0	7859.0	360.0	0.0	0.0	0.0
5828	2015-08-31 21:00:00+02:00	493.0	957.0	0.0	8683.0	7885.0	371.0	0.0	0.0	0.0
5829	2015-08-31 22:00:00+02:00	491.0	949.0	0.0	8272.0	7752.0	349.0	0.0	0.0	0.0
5830	2015-08-31 23:00:00+02:00	492.0	945.0	0.0	7743.0	7750.0	362.0	0.0	0.0	0.0

744 rows × 29 columns

```
In [ ]: df2015_8.describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	518.947581	585.767473	0.0	4919.491935	5859.370968	323.913978	0.0	0.0	0.0	NaN
std	49.287070	340.411157	0.0	1281.720111	1699.635272	53.829206	0.0	0.0	0.0	NaN
min	257.000000	0.000000	0.0	2630.000000	1570.000000	122.000000	0.0	0.0	0.0	NaN
25%	497.000000	308.750000	0.0	4038.500000	5058.500000	288.000000	0.0	0.0	0.0	NaN
50%	535.000000	635.500000	0.0	4630.000000	6237.500000	332.000000	0.0	0.0	0.0	NaN
75%	552.000000	909.000000	0.0	5439.000000	7215.500000	364.000000	0.0	0.0	0.0	NaN
max	575.000000	985.000000	0.0	10821.000000	8109.000000	425.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2016_8 = df2[df2['time'].str.contains("2016-08")]
df2016_8
# Monthly Datasets ("Year Month" = "0000 0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
13871	2016-08-01 00:00:00+02:00	342.0	0.0	0.0	3459.0	1211.0	286.0	0.0	0.0	0.0
13872	2016-08-01 01:00:00+02:00	310.0	0.0	0.0	3321.0	1255.0	251.0	0.0	0.0	0.0
13873	2016-08-01 02:00:00+02:00	311.0	0.0	0.0	3161.0	1251.0	251.0	0.0	0.0	0.0
13874	2016-08-01 03:00:00+02:00	316.0	0.0	0.0	3424.0	1335.0	270.0	0.0	0.0	0.0
13875	2016-08-01 04:00:00+02:00	322.0	0.0	0.0	3671.0	1331.0	300.0	0.0	0.0	0.0
...
14610	2016-08-31 19:00:00+02:00	387.0	911.0	0.0	5609.0	6683.0	312.0	0.0	0.0	0.0
14611	2016-08-31 20:00:00+02:00	387.0	910.0	0.0	6098.0	6704.0	312.0	0.0	0.0	0.0
14612	2016-08-31 21:00:00+02:00	387.0	912.0	0.0	6562.0	6783.0	314.0	0.0	0.0	0.0
14613	2016-08-31 22:00:00+02:00	389.0	900.0	0.0	5883.0	6561.0	313.0	0.0	0.0	0.0
14614	2016-08-31 23:00:00+02:00	385.0	887.0	0.0	5411.0	6236.0	314.0	0.0	0.0	0.0

744 rows × 29 columns

In []: df2016_8.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	369.645161	464.373656	0.0	4869.987903	4172.501344	283.306452	0.0	0.0	0.0	NaN
std	43.260305	373.010567	0.0	1142.879146	1720.690363	47.762311	0.0	0.0	0.0	NaN
min	173.000000	0.000000	0.0	2339.000000	799.000000	117.000000	0.0	0.0	0.0	NaN
25%	359.000000	0.000000	0.0	4032.000000	2620.750000	246.000000	0.0	0.0	0.0	NaN
50%	380.000000	587.000000	0.0	4693.500000	4900.500000	293.000000	0.0	0.0	0.0	NaN
75%	392.000000	844.750000	0.0	5784.750000	5489.500000	312.000000	0.0	0.0	0.0	NaN
max	504.000000	944.000000	0.0	8109.000000	6852.000000	415.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []: df2017_8 = df2[df2['time'].str.contains("2017-08")]
df2017_8
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
22631	2017-08-01 00:00:00+02:00	358.0	595.0	0.0	5604.0	5528.0	230.0	0.0	0.0	0.0
22632	2017-08-01 01:00:00+02:00	352.0	593.0	0.0	5092.0	5214.0	230.0	0.0	0.0	0.0
22633	2017-08-01 02:00:00+02:00	353.0	598.0	0.0	4704.0	5025.0	224.0	0.0	0.0	0.0
22634	2017-08-01 03:00:00+02:00	358.0	598.0	0.0	4591.0	4753.0	222.0	0.0	0.0	0.0
22635	2017-08-01 04:00:00+02:00	358.0	598.0	0.0	4653.0	4652.0	223.0	0.0	0.0	0.0
...
23370	2017-08-31 19:00:00+02:00	400.0	583.0	0.0	7323.0	3426.0	358.0	0.0	0.0	0.0
23371	2017-08-31 20:00:00+02:00	401.0	655.0	0.0	7679.0	3705.0	352.0	0.0	0.0	0.0
23372	2017-08-31 21:00:00+02:00	400.0	644.0	0.0	7641.0	3843.0	355.0	0.0	0.0	0.0
23373	2017-08-31 22:00:00+02:00	393.0	609.0	0.0	6974.0	3751.0	354.0	0.0	0.0	0.0
23374	2017-08-31 23:00:00+02:00	390.0	564.0	0.0	6213.0	3631.0	348.0	0.0	0.0	0.0

744 rows × 29 columns

```
In [ ]: df2017_8.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	362.879032	511.846774	0.0	6701.935484	3786.232527	306.235215	0.0	0.0	0.0	NaN
std	37.830057	257.066075	0.0	1823.156124	1235.674654	40.192435	0.0	0.0	0.0	NaN
min	248.000000	0.000000	0.0	3520.000000	1382.000000	211.000000	0.0	0.0	0.0	NaN
25%	343.000000	370.750000	0.0	5252.500000	2813.500000	266.000000	0.0	0.0	0.0	NaN
50%	370.000000	589.000000	0.0	6260.500000	3698.500000	319.000000	0.0	0.0	0.0	NaN
75%	393.000000	643.000000	0.0	8300.500000	4726.250000	339.000000	0.0	0.0	0.0	NaN
max	418.000000	985.000000	0.0	10898.000000	6906.000000	363.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2018_8 = df2[df2['time'].str.contains("2018-08")]
df2018_8
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

		time	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
31391		2018-08-01 00:00:00+02:00	276.0	880.0	0.0	5187.0	4251.0	268.0	0.0	0.0	0.0
31392		2018-08-01 01:00:00+02:00	273.0	777.0	0.0	5031.0	4212.0	270.0	0.0	0.0	0.0
31393		2018-08-01 02:00:00+02:00	272.0	852.0	0.0	4734.0	4328.0	269.0	0.0	0.0	0.0
31394		2018-08-01 03:00:00+02:00	274.0	869.0	0.0	4679.0	4441.0	269.0	0.0	0.0	0.0
31395		2018-08-01 04:00:00+02:00	270.0	863.0	0.0	4757.0	4516.0	267.0	0.0	0.0	0.0
...
32130		2018-08-31 19:00:00+02:00	382.0	727.0	0.0	7654.0	5748.0	312.0	0.0	0.0	0.0
32131		2018-08-31 20:00:00+02:00	382.0	727.0	0.0	7816.0	5782.0	310.0	0.0	0.0	0.0
32132		2018-08-31 21:00:00+02:00	383.0	756.0	0.0	7294.0	5681.0	315.0	0.0	0.0	0.0
32133		2018-08-31 22:00:00+02:00	380.0	704.0	0.0	5612.0	5509.0	316.0	0.0	0.0	0.0
32134		2018-08-31 23:00:00+02:00	373.0	683.0	0.0	4749.0	5054.0	305.0	0.0	0.0	0.0

744 rows × 29 columns

```
In [ ]: df2018_8.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	361.325269	300.142473	0.0	6119.346774	4121.565860	283.333333	0.0	0.0	0.0	NaN
std	32.709523	309.837792	0.0	2053.746498	1439.512717	38.787674	0.0	0.0	0.0	NaN
min	225.000000	0.000000	0.0	3086.000000	944.000000	164.000000	0.0	0.0	0.0	NaN
25%	353.000000	0.000000	0.0	4429.500000	2960.000000	259.750000	0.0	0.0	0.0	NaN
50%	369.000000	243.500000	0.0	5457.500000	4134.500000	293.000000	0.0	0.0	0.0	NaN
75%	381.000000	578.000000	0.0	7693.000000	5340.000000	312.000000	0.0	0.0	0.0	NaN
max	403.000000	959.000000	0.0	12347.000000	6860.000000	364.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2015_9 = df2[df2['time'].str.contains("2015-09")]
df2015_9
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
5831	2015-09-01 00:00:00+02:00	532.0	956.0	0.0	5023.0	8001.0	331.0	0.0	0.0	0.0	.
5832	2015-09-01 01:00:00+02:00	532.0	916.0	0.0	4409.0	7595.0	328.0	0.0	0.0	0.0	.
5833	2015-09-01 02:00:00+02:00	531.0	820.0	0.0	4425.0	7384.0	328.0	0.0	0.0	0.0	.
5834	2015-09-01 03:00:00+02:00	525.0	913.0	0.0	4400.0	7369.0	328.0	0.0	0.0	0.0	.
5835	2015-09-01 04:00:00+02:00	524.0	925.0	0.0	4303.0	7349.0	322.0	0.0	0.0	0.0	.
...
6546	2015-09-30 19:00:00+02:00	530.0	906.0	0.0	5343.0	7406.0	442.0	0.0	0.0	0.0	.
6547	2015-09-30 20:00:00+02:00	535.0	924.0	0.0	6297.0	7401.0	445.0	0.0	0.0	0.0	.
6548	2015-09-30 21:00:00+02:00	535.0	929.0	0.0	6894.0	7441.0	445.0	0.0	0.0	0.0	.
6549	2015-09-30 22:00:00+02:00	528.0	912.0	0.0	5482.0	7393.0	441.0	0.0	0.0	0.0	.
6550	2015-09-30 23:00:00+02:00	535.0	887.0	0.0	4957.0	7348.0	433.0	0.0	0.0	0.0	.

720 rows × 29 columns

```
In [1]: df2015_9.describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	517.106944	548.083333	0.0	5076.581944	5653.873611	343.491667	0.0	0.0	0.0	NaN
std	43.157324	363.538515	0.0	1285.125140	1536.541230	55.861416	0.0	0.0	0.0	NaN
min	344.000000	0.000000	0.0	2691.000000	1608.000000	202.000000	0.0	0.0	0.0	NaN
25%	502.000000	189.000000	0.0	4295.250000	4729.000000	300.000000	0.0	0.0	0.0	NaN
50%	521.000000	704.000000	0.0	4806.000000	5921.000000	328.500000	0.0	0.0	0.0	NaN
75%	544.000000	885.000000	0.0	5471.250000	6906.000000	397.250000	0.0	0.0	0.0	NaN
max	590.000000	984.000000	0.0	10853.000000	8244.000000	449.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2016_9 = df2[df2['time'].str.contains("2016-09")]
df2016_9
# Monthly Datasets ("Year_Month" ="0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
14615	2016-09-01 00:00:00+02:00	377.0	0.0	0.0	3619.0	1206.0	346.0	0.0	0.0	0.0
14616	2016-09-01 01:00:00+02:00	354.0	0.0	0.0	3675.0	1031.0	344.0	0.0	0.0	0.0
14617	2016-09-01 02:00:00+02:00	358.0	0.0	0.0	3775.0	1053.0	338.0	0.0	0.0	0.0
14618	2016-09-01 03:00:00+02:00	358.0	0.0	0.0	3942.0	1185.0	335.0	0.0	0.0	0.0
14619	2016-09-01 04:00:00+02:00	354.0	0.0	0.0	3898.0	1274.0	329.0	0.0	0.0	0.0
...
15330	2016-09-30 19:00:00+02:00	357.0	636.0	0.0	5982.0	6077.0	296.0	0.0	0.0	0.0
15331	2016-09-30 20:00:00+02:00	358.0	613.0	0.0	6046.0	6042.0	301.0	0.0	0.0	0.0
15332	2016-09-30 21:00:00+02:00	360.0	596.0	0.0	6016.0	5943.0	299.0	0.0	0.0	0.0
15333	2016-09-30 22:00:00+02:00	356.0	594.0	0.0	5859.0	5657.0	300.0	0.0	0.0	0.0
15334	2016-09-30 23:00:00+02:00	351.0	592.0	0.0	5301.0	5464.0	295.0	0.0	0.0	0.0

720 rows × 29 columns

In []: df2016_9.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	349.391667	473.720833	0.0	4643.550000	4208.013889	281.386111	0.0	0.0	0.0	NaN
std	39.648833	337.968867	0.0	1156.294013	1708.676767	39.528562	0.0	0.0	0.0	NaN
min	199.000000	0.000000	0.0	2640.000000	884.000000	155.000000	0.0	0.0	0.0	NaN
25%	339.000000	48.250000	0.0	3654.000000	2720.750000	254.750000	0.0	0.0	0.0	NaN
50%	354.000000	546.500000	0.0	4551.500000	4557.000000	290.000000	0.0	0.0	0.0	NaN
75%	366.000000	740.250000	0.0	5479.000000	5769.250000	304.000000	0.0	0.0	0.0	NaN
max	477.000000	976.000000	0.0	8645.000000	6794.000000	410.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []: df2017_9 = df2[df2['time'].str.contains("2017-09")]
df2017_9
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
23375	2017-09-01 00:00:00+02:00	352.0	574.0	0.0	4674.0	4117.0	218.0	0.0	0.0	0.0
23376	2017-09-01 01:00:00+02:00	348.0	621.0	0.0	4488.0	3993.0	217.0	0.0	0.0	0.0
23377	2017-09-01 02:00:00+02:00	349.0	551.0	0.0	4475.0	3917.0	217.0	0.0	0.0	0.0
23378	2017-09-01 03:00:00+02:00	342.0	540.0	0.0	4514.0	3593.0	217.0	0.0	0.0	0.0
23379	2017-09-01 04:00:00+02:00	343.0	642.0	0.0	4676.0	3559.0	213.0	0.0	0.0	0.0
...
24090	2017-09-30 19:00:00+02:00	379.0	647.0	0.0	7478.0	4371.0	288.0	0.0	0.0	0.0
24091	2017-09-30 20:00:00+02:00	378.0	662.0	0.0	7669.0	4334.0	291.0	0.0	0.0	0.0
24092	2017-09-30 21:00:00+02:00	379.0	656.0	0.0	7493.0	4128.0	285.0	0.0	0.0	0.0
24093	2017-09-30 22:00:00+02:00	377.0	641.0	0.0	7089.0	4003.0	268.0	0.0	0.0	0.0
24094	2017-09-30 23:00:00+02:00	374.0	645.0	0.0	6800.0	3901.0	265.0	0.0	0.0	0.0

720 rows × 29 columns

In []: df2017_9.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	351.968056	642.630556	0.0	7092.451389	4010.465278	310.801389	0.0	0.0	0.0	NaN
std	36.314272	259.615345	0.0	2174.924207	1240.158034	32.593561	0.0	0.0	0.0	NaN
min	243.000000	0.000000	0.0	3282.000000	1461.000000	213.000000	0.0	0.0	0.0	NaN
25%	331.000000	551.000000	0.0	5393.250000	3093.250000	287.000000	0.0	0.0	0.0	NaN
50%	359.000000	650.000000	0.0	6650.500000	4040.500000	322.500000	0.0	0.0	0.0	NaN
75%	381.000000	887.250000	0.0	8744.000000	5138.000000	336.000000	0.0	0.0	0.0	NaN
max	408.000000	999.000000	0.0	13783.000000	6592.000000	365.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []: df2018_9 = df2[df2['time'].str.contains("2018-09")]
df2018_9
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

		time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
32135		2018-09-01 00:00:00+02:00	283.0	922.0	0.0	6745.0	6657.0	350.0	0.0	0.0	0.0
32136		2018-09-01 01:00:00+02:00	276.0	926.0	0.0	5428.0	6398.0	343.0	0.0	0.0	0.0
32137		2018-09-01 02:00:00+02:00	280.0	936.0	0.0	4826.0	6109.0	342.0	0.0	0.0	0.0
32138		2018-09-01 03:00:00+02:00	281.0	801.0	0.0	4970.0	5509.0	342.0	0.0	0.0	0.0
32139		2018-09-01 04:00:00+02:00	283.0	694.0	0.0	4685.0	5246.0	334.0	0.0	0.0	0.0
...
32850		2018-09-30 19:00:00+02:00	354.0	718.0	0.0	4206.0	4082.0	238.0	0.0	0.0	0.0
32851		2018-09-30 20:00:00+02:00	353.0	679.0	0.0	4212.0	4109.0	241.0	0.0	0.0	0.0
32852		2018-09-30 21:00:00+02:00	355.0	704.0	0.0	4141.0	4049.0	245.0	0.0	0.0	0.0
32853		2018-09-30 22:00:00+02:00	354.0	610.0	0.0	3988.0	3654.0	239.0	0.0	0.0	0.0
32854		2018-09-30 23:00:00+02:00	347.0	572.0	0.0	3950.0	3331.0	247.0	0.0	0.0	0.0

720 rows × 29 columns

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	355.468056	558.161111	0.0	5842.062500	4651.018056	296.136111	0.0	0.0	0.0	NaN
std	37.043583	252.992557	0.0	1599.083686	1329.139806	40.003836	0.0	0.0	0.0	NaN
min	196.000000	0.000000	0.0	3039.000000	1203.000000	205.000000	0.0	0.0	0.0	NaN
25%	343.000000	478.750000	0.0	4533.250000	3935.500000	255.750000	0.0	0.0	0.0	NaN
50%	363.000000	680.000000	0.0	5474.000000	5090.500000	308.000000	0.0	0.0	0.0	NaN
75%	379.000000	744.000000	0.0	7007.000000	5604.750000	331.000000	0.0	0.0	0.0	NaN
max	405.000000	936.000000	0.0	10578.000000	6858.000000	362.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0

In []: df2018_5.describe()

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	318.309140	307.704301	0.0	5777.162634	3674.151882	280.450269	0.0	0.0	0.0	NaN
std	42.403733	273.388593	0.0	1595.993360	1517.692063	42.226974	0.0	0.0	0.0	NaN
min	0.000000	0.000000	0.0	2352.000000	1039.000000	44.000000	0.0	0.0	0.0	NaN
25%	291.750000	0.000000	0.0	4437.750000	2173.500000	247.750000	0.0	0.0	0.0	NaN
50%	323.000000	384.000000	0.0	5500.000000	4055.000000	293.000000	0.0	0.0	0.0	NaN
75%	350.000000	557.750000	0.0	6830.250000	5092.250000	310.000000	0.0	0.0	0.0	NaN
max	399.000000	943.000000	0.0	10778.000000	6185.000000	353.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

--	--	--	--	--	--	--	--	--	--	--

In []: df2018_3.describe()

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	307.254374	124.415882	0.0	4936.013459	2157.183042	288.126514	0.0	0.0	0.0	NaN
std	41.145572	243.478477	0.0	1700.369234	1410.681768	48.100609	0.0	0.0	0.0	NaN
min	207.000000	0.000000	0.0	2792.000000	700.000000	167.000000	0.0	0.0	0.0	NaN
25%	282.000000	0.000000	0.0	3929.000000	1222.000000	246.000000	0.0	0.0	0.0	NaN
50%	308.000000	0.000000	0.0	4372.000000	1518.000000	303.000000	0.0	0.0	0.0	NaN
75%	332.500000	0.000000	0.0	5367.000000	2662.000000	327.000000	0.0	0.0	0.0	NaN
max	401.000000	857.000000	0.0	11982.000000	5621.000000	355.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

--	--	--	--	--	--	--	--	--	--	--

In []: df2015_10 = df2[df2['time'].str.contains("2015-10")]
df2015_10
Monthly Datasets ("Year_Month" ="0000_0")

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
6551	2015-10-01 00:00:00+02:00	506.0	959.0	0.0	5522.0	7661.0	298.0	0.0	0.0	0.0	.
6552	2015-10-01 01:00:00+02:00	506.0	879.0	0.0	5235.0	7313.0	278.0	0.0	0.0	0.0	.
6553	2015-10-01 02:00:00+02:00	509.0	911.0	0.0	4592.0	6754.0	281.0	0.0	0.0	0.0	.
6554	2015-10-01 03:00:00+02:00	511.0	890.0	0.0	4274.0	6372.0	280.0	0.0	0.0	0.0	.
6555	2015-10-01 04:00:00+02:00	512.0	917.0	0.0	4122.0	6109.0	275.0	0.0	0.0	0.0	.
...
7291	2015-10-31 19:00:00+01:00	521.0	0.0	0.0	4976.0	6098.0	225.0	0.0	0.0	0.0	.
7292	2015-10-31 20:00:00+01:00	518.0	0.0	0.0	5079.0	5985.0	224.0	0.0	0.0	0.0	.
7293	2015-10-31 21:00:00+01:00	525.0	0.0	0.0	5115.0	5984.0	225.0	0.0	0.0	0.0	.
7294	2015-10-31 22:00:00+01:00	525.0	0.0	0.0	4825.0	5694.0	226.0	0.0	0.0	0.0	.
7295	2015-10-31 23:00:00+01:00	522.0	0.0	0.0	4578.0	5257.0	214.0	0.0	0.0	0.0	.

745 rows × 29 columns

```
In [1]: df2015_10.describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	519.556452	528.018817	0.0	5231.500000	5788.694892	323.391129	0.0	0.0	0.0	NaN
std	42.329599	352.768164	0.0	1340.424742	1372.217094	61.140938	0.0	0.0	0.0	NaN
min	375.000000	0.000000	0.0	3485.000000	2360.000000	201.000000	0.0	0.0	0.0	NaN
25%	498.000000	216.750000	0.0	4349.250000	4714.250000	277.000000	0.0	0.0	0.0	NaN
50%	529.000000	564.000000	0.0	4895.500000	5985.500000	320.000000	0.0	0.0	0.0	NaN
75%	552.000000	909.500000	0.0	5677.000000	7008.750000	374.000000	0.0	0.0	0.0	NaN
max	585.000000	986.000000	0.0	11956.000000	8359.000000	442.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2016_10 = df2[df2['time'].str.contains("2016-10")]
df2016_10
# Monthly Datasets ("Year Month" = "0000 0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
15335	2016-10-01 00:00:00+02:00	355.0	0.0	0.0	3497.0	902.0	267.0	0.0	0.0	0.0
15336	2016-10-01 01:00:00+02:00	361.0	0.0	0.0	3363.0	844.0	256.0	0.0	0.0	0.0
15337	2016-10-01 02:00:00+02:00	367.0	0.0	0.0	3314.0	842.0	256.0	0.0	0.0	0.0
15338	2016-10-01 03:00:00+02:00	365.0	0.0	0.0	3237.0	862.0	254.0	0.0	0.0	0.0
15339	2016-10-01 04:00:00+02:00	357.0	0.0	0.0	3100.0	843.0	249.0	0.0	0.0	0.0
...
16075	2016-10-31 19:00:00+01:00	362.0	642.0	0.0	6179.0	5578.0	295.0	0.0	0.0	0.0
16076	2016-10-31 20:00:00+01:00	368.0	619.0	0.0	5484.0	5572.0	303.0	0.0	0.0	0.0
16077	2016-10-31 21:00:00+01:00	367.0	626.0	0.0	5460.0	5522.0	306.0	0.0	0.0	0.0
16078	2016-10-31 22:00:00+01:00	365.0	546.0	0.0	4919.0	5131.0	311.0	0.0	0.0	0.0
16079	2016-10-31 23:00:00+01:00	359.0	524.0	0.0	4364.0	4736.0	319.0	0.0	0.0	0.0

745 rows × 29 columns

	df2016_10.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
Out[]:	

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	745.000000	745.000000	745.0	745.000000	745.000000	745.000000	745.0	745.0	745.0	0.0
mean	336.218792	613.769128	0.0	6222.820134	4271.040268	276.995973	0.0	0.0	0.0	NaN
std	41.611544	352.860276	0.0	2697.861633	1740.678858	41.442099	0.0	0.0	0.0	NaN
min	193.000000	0.000000	0.0	2536.000000	715.000000	112.000000	0.0	0.0	0.0	NaN
25%	318.000000	319.000000	0.0	4117.000000	3126.000000	243.000000	0.0	0.0	0.0	NaN
50%	340.000000	697.000000	0.0	5246.000000	4974.000000	284.000000	0.0	0.0	0.0	NaN
75%	356.000000	912.000000	0.0	8627.000000	5562.000000	305.000000	0.0	0.0	0.0	NaN
max	494.000000	995.000000	0.0	12981.000000	6521.000000	414.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

	df2017_10 = df2[df2['time'].str.contains("2017-10")] df2017_10 # Monthly Datasets ("Year_Month" ="0000_0")
In []:	

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
24095	2017-10-01 00:00:00+02:00	349.0	878.0	0.0	4702.0	5324.0	284.0	0.0	0.0	0.0
24096	2017-10-01 01:00:00+02:00	350.0	863.0	0.0	4234.0	5026.0	286.0	0.0	0.0	0.0
24097	2017-10-01 02:00:00+02:00	352.0	879.0	0.0	4348.0	5060.0	282.0	0.0	0.0	0.0
24098	2017-10-01 03:00:00+02:00	351.0	859.0	0.0	4138.0	4755.0	280.0	0.0	0.0	0.0
24099	2017-10-01 04:00:00+02:00	352.0	858.0	0.0	4014.0	4572.0	270.0	0.0	0.0	0.0
...
24835	2017-10-31 19:00:00+01:00	367.0	825.0	0.0	14705.0	5483.0	351.0	0.0	0.0	0.0
24836	2017-10-31 20:00:00+01:00	363.0	800.0	0.0	14058.0	5435.0	350.0	0.0	0.0	0.0
24837	2017-10-31 21:00:00+01:00	367.0	701.0	0.0	10720.0	5431.0	347.0	0.0	0.0	0.0
24838	2017-10-31 22:00:00+01:00	365.0	610.0	0.0	8643.0	5485.0	349.0	0.0	0.0	0.0
24839	2017-10-31 23:00:00+01:00	363.0	645.0	0.0	7793.0	5437.0	346.0	0.0	0.0	0.0

745 rows × 29 columns

In []: df2017_10.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	745.000000	745.000000	745.0	745.000000	745.000000	745.000000	745.0	745.0	745.0	0.0
mean	343.938255	561.322148	0.0	6961.800000	4321.130201	287.418792	0.0	0.0	0.0	NaN
std	28.832785	254.442651	0.0	2701.462979	1319.103218	36.371207	0.0	0.0	0.0	NaN
min	252.000000	0.000000	0.0	3271.000000	1449.000000	200.000000	0.0	0.0	0.0	NaN
25%	330.000000	381.000000	0.0	4799.000000	3253.000000	261.000000	0.0	0.0	0.0	NaN
50%	347.000000	584.000000	0.0	5928.000000	4638.000000	291.000000	0.0	0.0	0.0	NaN
75%	361.000000	761.000000	0.0	8893.000000	5351.000000	316.000000	0.0	0.0	0.0	NaN
max	407.000000	986.000000	0.0	15892.000000	6587.000000	354.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []: df2018_10 = df2[df2['time'].str.contains("2018-10")]
df2018_10
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

		time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
32855		2018-10-01 00:00:00+02:00	265.0	583.0	0.0	4612.0	5821.0	347.0	0.0	0.0	0.0
32856		2018-10-01 01:00:00+02:00	262.0	557.0	0.0	4272.0	5641.0	342.0	0.0	0.0	0.0
32857		2018-10-01 02:00:00+02:00	266.0	605.0	0.0	4250.0	5494.0	333.0	0.0	0.0	0.0
32858		2018-10-01 03:00:00+02:00	266.0	632.0	0.0	4350.0	5212.0	326.0	0.0	0.0	0.0
32859		2018-10-01 04:00:00+02:00	267.0	600.0	0.0	4186.0	4971.0	324.0	0.0	0.0	0.0
...
33595		2018-10-31 19:00:00+01:00	297.0	510.0	0.0	13270.0	5301.0	318.0	0.0	0.0	0.0
33596		2018-10-31 20:00:00+01:00	299.0	510.0	0.0	12980.0	5252.0	321.0	0.0	0.0	0.0
33597		2018-10-31 21:00:00+01:00	298.0	504.0	0.0	12441.0	5287.0	320.0	0.0	0.0	0.0
33598		2018-10-31 22:00:00+01:00	295.0	508.0	0.0	10839.0	5270.0	322.0	0.0	0.0	0.0
33599		2018-10-31 23:00:00+01:00	291.0	500.0	0.0	8416.0	5228.0	321.0	0.0	0.0	0.0

745 rows × 29 columns

In []:	df2018_10.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
---------	----------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	745.000000	745.000000	745.0	745.000000	745.000000	745.000000	745.0	745.0	745.0	0.0
mean	317.927517	405.837584	0.0	6127.481879	3782.816107	291.299329	0.0	0.0	0.0	NaN
std	37.563532	259.168722	0.0	2233.644072	1442.747645	41.250159	0.0	0.0	0.0	NaN
min	185.000000	0.000000	0.0	3197.000000	884.000000	203.000000	0.0	0.0	0.0	NaN
25%	296.000000	175.000000	0.0	4306.000000	2596.000000	249.000000	0.0	0.0	0.0	NaN
50%	317.000000	508.000000	0.0	5498.000000	3990.000000	306.000000	0.0	0.0	0.0	NaN
75%	345.000000	590.000000	0.0	7196.000000	5117.000000	326.000000	0.0	0.0	0.0	NaN
max	388.000000	766.000000	0.0	13270.000000	6067.000000	350.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:	df2015_11 = df2[df2['time'].str.contains("2015-11")] df2015_11 # Monthly Datasets ("Year_Month" ="0000_0")
---------	------------------------------------------------------------------------------------------------------------

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	...
7296	2015-11-01 00:00:00+01:00	525.0	713.0	0.0	5689.0	6807.0	277.0	0.0	0.0	0.0	.
7297	2015-11-01 01:00:00+01:00	527.0	572.0	0.0	4808.0	6059.0	273.0	0.0	0.0	0.0	.
7298	2015-11-01 02:00:00+01:00	518.0	497.0	0.0	4265.0	5338.0	274.0	0.0	0.0	0.0	.
7299	2015-11-01 03:00:00+01:00	511.0	371.0	0.0	4290.0	4521.0	279.0	0.0	0.0	0.0	.
7300	2015-11-01 04:00:00+01:00	513.0	337.0	0.0	4240.0	3865.0	279.0	0.0	0.0	0.0	.
...
8011	2015-11-30 19:00:00+01:00	458.0	859.0	0.0	11821.0	6984.0	425.0	0.0	0.0	0.0	.
8012	2015-11-30 20:00:00+01:00	463.0	913.0	0.0	11586.0	7091.0	424.0	0.0	0.0	0.0	.
8013	2015-11-30 21:00:00+01:00	461.0	948.0	0.0	11430.0	7150.0	426.0	0.0	0.0	0.0	.
8014	2015-11-30 22:00:00+01:00	463.0	961.0	0.0	9825.0	7170.0	427.0	0.0	0.0	0.0	.
8015	2015-11-30 23:00:00+01:00	464.0	937.0	0.0	8403.0	7130.0	421.0	0.0	0.0	0.0	.

720 rows × 29 columns

```
In [1]: df2015[11].describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	503.616667	695.284722	0.0	5387.741667	5982.631944	346.063889	0.0	0.0	0.0	NaN
std	48.514228	293.445189	0.0	1670.962167	1606.453974	57.278849	0.0	0.0	0.0	NaN
min	365.000000	0.000000	0.0	2771.000000	1021.000000	207.000000	0.0	0.0	0.0	NaN
25%	464.000000	568.500000	0.0	4340.500000	5224.750000	302.000000	0.0	0.0	0.0	NaN
50%	510.000000	816.000000	0.0	4875.500000	6517.000000	347.500000	0.0	0.0	0.0	NaN
75%	545.000000	937.000000	0.0	6119.500000	7130.000000	400.000000	0.0	0.0	0.0	NaN
max	586.000000	983.000000	0.0	11821.000000	8159.000000	434.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2016_11 = df2[df2['time'].str.contains("2016-11")]
df2016_11
# Monthly Datasets ("Year Month" = "0000_0")
```

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
16080	2016-11-01 00:00:00+01:00	321.0	0.0	0.0	3218.0	770.0	178.0	0.0	0.0	0.0
16081	2016-11-01 01:00:00+01:00	325.0	0.0	0.0	3058.0	724.0	178.0	0.0	0.0	0.0
16082	2016-11-01 02:00:00+01:00	330.0	0.0	0.0	2923.0	687.0	178.0	0.0	0.0	0.0
16083	2016-11-01 03:00:00+01:00	317.0	0.0	0.0	2845.0	661.0	177.0	0.0	0.0	0.0
16084	2016-11-01 04:00:00+01:00	314.0	0.0	0.0	2948.0	683.0	178.0	0.0	0.0	0.0
...
16795	2016-11-30 19:00:00+01:00	382.0	894.0	0.0	8492.0	6884.0	299.0	0.0	0.0	0.0
16796	2016-11-30 20:00:00+01:00	381.0	892.0	0.0	8542.0	7002.0	300.0	0.0	0.0	0.0
16797	2016-11-30 21:00:00+01:00	382.0	894.0	0.0	8680.0	7066.0	299.0	0.0	0.0	0.0
16798	2016-11-30 22:00:00+01:00	377.0	893.0	0.0	7691.0	6997.0	299.0	0.0	0.0	0.0
16799	2016-11-30 23:00:00+01:00	371.0	892.0	0.0	6747.0	6919.0	298.0	0.0	0.0	0.0

720 rows × 29 columns

```
In [1]: df2016_11.describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	hydro pumped storage aggregated
count	719.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	355.573018	649.545833	0.0	6439.015278	4581.631944	271.569444	0.0	0.0	0.0	NaN
std	44.776541	349.322269	0.0	2947.627392	1961.932734	42.303107	0.0	0.0	0.0	NaN
min	186.000000	0.000000	0.0	2845.000000	661.000000	177.000000	0.0	0.0	0.0	NaN
25%	339.000000	323.750000	0.0	4146.250000	2533.000000	232.000000	0.0	0.0	0.0	NaN
50%	359.000000	879.000000	0.0	5231.000000	5292.000000	278.000000	0.0	0.0	0.0	NaN
75%	377.500000	891.000000	0.0	8352.750000	5956.750000	293.000000	0.0	0.0	0.0	NaN
max	506.000000	987.000000	0.0	16250.000000	7327.000000	408.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2017_11 = df2[df2['time'].str.contains("2017-11")]
df2017_11
# Monthly Datasets ("Year Month" = "0000_0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
24840	2017-11-01 00:00:00+01:00	349.0	865.0	0.0	4318.0	5458.0	288.0	0.0	0.0	0.0
24841	2017-11-01 01:00:00+01:00	347.0	889.0	0.0	4003.0	5386.0	287.0	0.0	0.0	0.0
24842	2017-11-01 02:00:00+01:00	348.0	915.0	0.0	4111.0	5244.0	282.0	0.0	0.0	0.0
24843	2017-11-01 03:00:00+01:00	345.0	940.0	0.0	4287.0	5317.0	271.0	0.0	0.0	0.0
24844	2017-11-01 04:00:00+01:00	344.0	911.0	0.0	4062.0	5265.0	271.0	0.0	0.0	0.0
...
25555	2017-11-30 19:00:00+01:00	382.0	988.0	0.0	14920.0	6888.0	344.0	0.0	0.0	0.0
25556	2017-11-30 20:00:00+01:00	381.0	988.0	0.0	15078.0	6916.0	349.0	0.0	0.0	0.0
25557	2017-11-30 21:00:00+01:00	389.0	985.0	0.0	13571.0	6929.0	351.0	0.0	0.0	0.0
25558	2017-11-30 22:00:00+01:00	385.0	984.0	0.0	11956.0	6926.0	350.0	0.0	0.0	0.0
25559	2017-11-30 23:00:00+01:00	383.0	980.0	0.0	11502.0	6862.0	350.0	0.0	0.0	0.0

720 rows × 29 columns

	df2017_11.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
Out[]:	

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	354.593056	667.647222	0.0	8534.818056	5491.216667	303.543056	0.0	0.0	0.0	NaN
std	39.487111	273.416870	0.0	4059.602836	1678.616981	39.857791	0.0	0.0	0.0	NaN
min	0.000000	0.000000	0.0	3161.000000	0.000000	0.000000	0.0	0.0	0.0	NaN
25%	341.000000	530.500000	0.0	5079.500000	4579.250000	282.000000	0.0	0.0	0.0	NaN
50%	360.000000	657.000000	0.0	7092.500000	6167.500000	314.000000	0.0	0.0	0.0	NaN
75%	379.000000	914.250000	0.0	11854.250000	6763.250000	331.000000	0.0	0.0	0.0	NaN
max	411.000000	988.000000	0.0	18981.000000	7239.000000	360.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

	df2018_11 = df2[df2['time'].str.contains("2018-11")] df2018_11 # Monthly Datasets ("Year_Month" ="0000_0")
In[]:	

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
33600	2018-11-01 00:00:00+01:00	262.0	194.0	0.0	4152.0	3905.0	295.0	0.0	0.0	0.0
33601	2018-11-01 01:00:00+01:00	272.0	217.0	0.0	3597.0	3733.0	281.0	0.0	0.0	0.0
33602	2018-11-01 02:00:00+01:00	262.0	202.0	0.0	3557.0	3540.0	281.0	0.0	0.0	0.0
33603	2018-11-01 03:00:00+01:00	270.0	213.0	0.0	3645.0	3642.0	279.0	0.0	0.0	0.0
33604	2018-11-01 04:00:00+01:00	273.0	201.0	0.0	3632.0	3697.0	269.0	0.0	0.0	0.0
...
34315	2018-11-30 19:00:00+01:00	355.0	0.0	0.0	11477.0	6471.0	319.0	0.0	0.0	0.0
34316	2018-11-30 20:00:00+01:00	357.0	0.0	0.0	11413.0	6508.0	319.0	0.0	0.0	0.0
34317	2018-11-30 21:00:00+01:00	356.0	0.0	0.0	10380.0	6357.0	319.0	0.0	0.0	0.0
34318	2018-11-30 22:00:00+01:00	354.0	0.0	0.0	8134.0	6251.0	316.0	0.0	0.0	0.0
34319	2018-11-30 23:00:00+01:00	352.0	0.0	0.0	6629.0	6301.0	309.0	0.0	0.0	0.0

720 rows × 29 columns

```
In [1]: df2018_11.describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	hydro pumped storage aggregated
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000000	720.0	720.0	720.0	0.0
mean	342.102778	375.736111	0.0	6945.804167	4838.777778	272.983333	0.0	0.0	0.0	NaN
std	37.448461	314.466433	0.0	2616.456572	1696.159079	41.878953	0.0	0.0	0.0	NaN
min	207.000000	0.000000	0.0	2651.000000	745.000000	164.000000	0.0	0.0	0.0	NaN
25%	328.000000	0.000000	0.0	5003.750000	3752.000000	243.000000	0.0	0.0	0.0	NaN
50%	353.000000	468.500000	0.0	6125.500000	5273.000000	286.000000	0.0	0.0	0.0	NaN
75%	368.000000	693.000000	0.0	9575.500000	6263.250000	305.000000	0.0	0.0	0.0	NaN
max	386.000000	797.000000	0.0	13250.000000	6928.000000	346.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2015_12 = df2[df2['time'].str.contains("2015-12")]
df2015_12
# Monthly Datasets ("Year_Month" = "0000_0")
```

Out[1]:

	time	generation biomass	generation fossil brown coal/lignite	fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
8016	2015-12-01 00:00:00+01:00	537.0	647.0	0.0	4947.0	6469.0	245.0	0.0	0.0	0.0
8017	2015-12-01 01:00:00+01:00	530.0	637.0	0.0	4419.0	6250.0	254.0	0.0	0.0	0.0
8018	2015-12-01 02:00:00+01:00	526.0	547.0	0.0	4115.0	5844.0	249.0	0.0	0.0	0.0
8019	2015-12-01 03:00:00+01:00	520.0	493.0	0.0	3996.0	5631.0	254.0	0.0	0.0	0.0
8020	2015-12-01 04:00:00+01:00	517.0	480.0	0.0	4132.0	5311.0	253.0	0.0	0.0	0.0
...
8755	2015-12-31 19:00:00+01:00	363.0	0.0	0.0	6770.0	4083.0	272.0	0.0	0.0	0.0
8756	2015-12-31 20:00:00+01:00	376.0	0.0	0.0	6709.0	4079.0	276.0	0.0	0.0	0.0
8757	2015-12-31 21:00:00+01:00	392.0	0.0	0.0	5791.0	4053.0	247.0	0.0	0.0	0.0
8758	2015-12-31 22:00:00+01:00	401.0	0.0	0.0	4822.0	3709.0	235.0	0.0	0.0	0.0
8759	2015-12-31 23:00:00+01:00	400.0	0.0	0.0	4594.0	3339.0	234.0	0.0	0.0	0.0

744 rows × 29 columns

```
In [1]: df2015_12.describe() # Description tables of monthly datasets ("Year Month" = "0000 0")
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	743.00000	743.00000	743.0	743.000000	743.000000	743.000000	743.0	743.0	743.0	0.0
mean	483.92463	493.480485	0.0	5062.588156	5323.676985	330.651413	0.0	0.0	0.0	NaN
std	59.59178	400.370555	0.0	1245.427021	1931.720695	63.942325	0.0	0.0	0.0	NaN
min	356.00000	0.000000	0.0	3047.000000	1201.000000	159.000000	0.0	0.0	0.0	NaN
25%	435.00000	0.000000	0.0	4134.500000	4020.000000	291.000000	0.0	0.0	0.0	NaN
50%	510.00000	582.000000	0.0	4715.000000	5767.000000	329.000000	0.0	0.0	0.0	NaN
75%	533.00000	917.000000	0.0	5821.000000	7097.000000	383.000000	0.0	0.0	0.0	NaN
max	570.00000	988.000000	0.0	10388.000000	8135.000000	435.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

```
In [ ]: df2016_12 = df2[df2['time'].str.contains("2016-12")]
df2016_12
# Monthly Datasets ("Year Month" = "0000 0")
```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
16800	2016-12-01 00:00:00+01:00	358.0	0.0	0.0	3293.0	1089.0	331.0	0.0	0.0	0.0
16801	2016-12-01 01:00:00+01:00	359.0	0.0	0.0	3195.0	1139.0	324.0	0.0	0.0	0.0
16802	2016-12-01 02:00:00+01:00	362.0	0.0	0.0	3146.0	1161.0	273.0	0.0	0.0	0.0
16803	2016-12-01 03:00:00+01:00	368.0	0.0	0.0	3408.0	1321.0	280.0	0.0	0.0	0.0
16804	2016-12-01 04:00:00+01:00	371.0	0.0	0.0	3358.0	1553.0	311.0	0.0	0.0	0.0
...
17539	2016-12-31 19:00:00+01:00	336.0	977.0	0.0	4880.0	6316.0	188.0	0.0	0.0	0.0
17540	2016-12-31 20:00:00+01:00	337.0	904.0	0.0	4911.0	6321.0	184.0	0.0	0.0	0.0
17541	2016-12-31 21:00:00+01:00	340.0	895.0	0.0	4667.0	6291.0	178.0	0.0	0.0	0.0
17542	2016-12-31 22:00:00+01:00	337.0	896.0	0.0	4564.0	6242.0	177.0	0.0	0.0	0.0
17543	2016-12-31 23:00:00+01:00	337.0	905.0	0.0	5116.0	6211.0	177.0	0.0	0.0	0.0

744 rows × 29 columns

	df2016_12.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
--	----------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	338.776882	670.795699	0.0	6176.176075	4984.295699	276.639785	0.0	0.0	0.0	NaN
std	42.204925	376.661196	0.0	2895.579185	1908.667080	43.100894	0.0	0.0	0.0	NaN
min	195.000000	0.000000	0.0	2703.000000	1053.000000	166.000000	0.0	0.0	0.0	NaN
25%	325.000000	332.500000	0.0	4166.000000	3958.750000	250.000000	0.0	0.0	0.0	NaN
50%	334.000000	892.000000	0.0	5092.000000	5471.000000	282.000000	0.0	0.0	0.0	NaN
75%	346.000000	962.250000	0.0	7146.250000	6469.250000	301.000000	0.0	0.0	0.0	NaN
max	488.000000	996.000000	0.0	15205.000000	7289.000000	422.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

	df2017_12 = df2[df2['time'].str.contains("2017-12")] # Monthly Datasets ("Year_Month" ="0000_0")
--	-----------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
25560	2017-12-01 00:00:00+01:00	360.0	891.0	0.0	5304.0	6559.0	288.0	0.0	0.0	0.0
25561	2017-12-01 01:00:00+01:00	359.0	899.0	0.0	4627.0	6393.0	296.0	0.0	0.0	0.0
25562	2017-12-01 02:00:00+01:00	360.0	924.0	0.0	4580.0	6394.0	292.0	0.0	0.0	0.0
25563	2017-12-01 03:00:00+01:00	359.0	929.0	0.0	4576.0	6266.0	291.0	0.0	0.0	0.0
25564	2017-12-01 04:00:00+01:00	354.0	930.0	0.0	4499.0	6351.0	289.0	0.0	0.0	0.0
...
26299	2017-12-31 19:00:00+01:00	273.0	0.0	0.0	3101.0	806.0	198.0	0.0	0.0	0.0
26300	2017-12-31 20:00:00+01:00	267.0	0.0	0.0	3022.0	764.0	196.0	0.0	0.0	0.0
26301	2017-12-31 21:00:00+01:00	268.0	0.0	0.0	3205.0	784.0	184.0	0.0	0.0	0.0
26302	2017-12-31 22:00:00+01:00	277.0	0.0	0.0	3266.0	791.0	195.0	0.0	0.0	0.0
26303	2017-12-31 23:00:00+01:00	278.0	0.0	0.0	3552.0	926.0	196.0	0.0	0.0	0.0

744 rows × 29 columns

	df2017_12.describe() # Description tables of monthly datasets ("Year_Month" ="0000_0")
--	----------------------------------------------------------------------------------------

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	347.173387	476.543011	0.0	5835.915323	4330.653226	293.926075	0.0	0.0	0.0	NaN
std	41.488337	350.132012	0.0	2573.985107	2255.178294	50.789579	0.0	0.0	0.0	NaN
min	215.000000	0.000000	0.0	2949.000000	764.000000	165.000000	0.0	0.0	0.0	NaN
25%	327.000000	202.500000	0.0	3951.000000	2039.750000	268.000000	0.0	0.0	0.0	NaN
50%	358.000000	522.500000	0.0	4841.500000	4630.000000	310.000000	0.0	0.0	0.0	NaN
75%	379.250000	800.750000	0.0	7087.500000	6540.250000	334.000000	0.0	0.0	0.0	NaN
max	416.000000	990.000000	0.0	13463.000000	7740.000000	363.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

	df2018_12 = df2[df2['time'].str.contains("2018-12")] df2018_12 # Monthly Datasets ("Year_Month" ="0000_0")
--	------------------------------------------------------------------------------------------------------------------

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal
34320	2018-12-01 00:00:00+01:00	272.0	299.0	0.0	4919.0	5502.0	328.0	0.0	0.0	0.0
34321	2018-12-01 01:00:00+01:00	271.0	292.0	0.0	4240.0	5107.0	319.0	0.0	0.0	0.0
34322	2018-12-01 02:00:00+01:00	282.0	247.0	0.0	4038.0	4595.0	317.0	0.0	0.0	0.0
34323	2018-12-01 03:00:00+01:00	306.0	203.0	0.0	4006.0	4436.0	317.0	0.0	0.0	0.0
34324	2018-12-01 04:00:00+01:00	294.0	205.0	0.0	4251.0	4610.0	308.0	0.0	0.0	0.0
...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	2628.0	178.0	0.0	0.0	0.0
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	2566.0	174.0	0.0	0.0	0.0
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	2422.0	168.0	0.0	0.0	0.0
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	2293.0	163.0	0.0	0.0	0.0
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	2166.0	163.0	0.0	0.0	0.0

744 rows × 29 columns

In []: df2018_12.describe() # Description tables of monthly datasets ("Year_Month" = "0000_0")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	generation fossil hard coal	generation fossil oil	generation fossil oil shale	generation fossil peat	generation geothermal	generation hydro pumped storage aggregated
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000	744.0	744.0	744.0	0.0
mean	320.16129	406.228495	0.0	6463.536290	3365.758065	269.021505	0.0	0.0	0.0	NaN
std	33.88101	269.542385	0.0	2102.415078	1415.262858	44.055142	0.0	0.0	0.0	NaN
min	226.000000	0.000000	0.0	2673.000000	704.000000	163.000000	0.0	0.0	0.0	NaN
25%	301.000000	202.750000	0.0	4878.500000	2419.250000	232.000000	0.0	0.0	0.0	NaN
50%	317.000000	448.000000	0.0	6045.000000	2995.500000	279.000000	0.0	0.0	0.0	NaN
75%	343.000000	663.000000	0.0	7747.750000	4676.750000	301.000000	0.0	0.0	0.0	NaN
max	386.000000	910.000000	0.0	13292.000000	6721.000000	363.000000	0.0	0.0	0.0	NaN

8 rows × 28 columns

In []:

df2.columns.values

Below are the appended lists of the mean price of energy per EUR/MWH and the output mean per each resource for each month. However, the first appended list is the mean price of energy per EUR/MWH for each month.

In []:

```
Price_Actual = []
df2018_12.describe(include='all').loc['mean']
Price_Actual.append(df2018_12.describe(include='all').loc['mean'][["price actual"]])
df2018_11.describe(include='all').loc['mean']
Price_Actual.append(df2018_11.describe(include='all').loc['mean'][["price actual"]])
df2018_10.describe(include='all').loc['mean']
Price_Actual.append(df2018_10.describe(include='all').loc['mean'][["price actual"]])
df2018_9.describe(include='all').loc['mean']
Price_Actual.append(df2018_9.describe(include='all').loc['mean'][["price actual"]])
df2018_8.describe(include='all').loc['mean']
Price_Actual.append(df2018_8.describe(include='all').loc['mean'][["price actual"]])
df2018_7.describe(include='all').loc['mean']
Price_Actual.append(df2018_7.describe(include='all').loc['mean'][["price actual"]])
```



```
Price_Actual.append(df2015_1.describe(include='all').loc['mean']["price actual"])

Price_Actual.reverse() # Average monthly values
print(Price_Actual)
```

```
[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777
7, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688
176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944
444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.581055555555556, 67.59513
440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25
82222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.
15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022
, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 66.623513888888
9, 67.04260752688172]
```

In []:

```
In [ ]: solar =[]
df2018_12.describe(include='all').loc['mean']
solar.append(df2018_12.describe(include='all').loc['mean']['generation solar'])
df2018_11.describe(include='all').loc['mean']
solar.append(df2018_11.describe(include='all').loc['mean']['generation solar'])
df2018_10.describe(include='all').loc['mean']
solar.append(df2018_10.describe(include='all').loc['mean']['generation solar'])
df2018_9.describe(include='all').loc['mean']
solar.append(df2018_9.describe(include='all').loc['mean']['generation solar'])
df2018_8.describe(include='all').loc['mean']
solar.append(df2018_8.describe(include='all').loc['mean']['generation solar'])
df2018_7.describe(include='all').loc['mean']
solar.append(df2018_7.describe(include='all').loc['mean']['generation solar'])
df2018_6.describe(include='all').loc['mean']
solar.append(df2018_6.describe(include='all').loc['mean']['generation solar'])
df2018_5.describe(include='all').loc['mean']
solar.append(df2018_5.describe(include='all').loc['mean']['generation solar'])
df2018_4.describe(include='all').loc['mean']
solar.append(df2018_4.describe(include='all').loc['mean']['generation solar'])
df2018_3.describe(include='all').loc['mean']
solar.append(df2018_3.describe(include='all').loc['mean']['generation solar'])
df2018_2.describe(include='all').loc['mean']
solar.append(df2018_2.describe(include='all').loc['mean']['generation solar'])
df2018_1.describe(include='all').loc['mean']
solar.append(df2018_1.describe(include='all').loc['mean']['generation solar'])
df2017_12.describe(include='all').loc['mean']
solar.append(df2017_12.describe(include='all').loc['mean']['generation solar'])
df2017_11.describe(include='all').loc['mean']
solar.append(df2017_11.describe(include='all').loc['mean']['generation solar'])
df2017_10.describe(include='all').loc['mean']
solar.append(df2017_10.describe(include='all').loc['mean']['generation solar'])
df2017_9.describe(include='all').loc['mean']
solar.append(df2017_9.describe(include='all').loc['mean']['generation solar'])
df2017_8.describe(include='all').loc['mean']
solar.append(df2017_8.describe(include='all').loc['mean']['generation solar'])
df2017_7.describe(include='all').loc['mean']
solar.append(df2017_7.describe(include='all').loc['mean']['generation solar'])
df2017_6.describe(include='all').loc['mean']
solar.append(df2017_6.describe(include='all').loc['mean']['generation solar'])
df2017_5.describe(include='all').loc['mean']
solar.append(df2017_5.describe(include='all').loc['mean']['generation solar'])
df2017_4.describe(include='all').loc['mean']
solar.append(df2017_4.describe(include='all').loc['mean']['generation solar'])
df2017_3.describe(include='all').loc['mean']
solar.append(df2017_3.describe(include='all').loc['mean']['generation solar'])
df2017_2.describe(include='all').loc['mean']
solar.append(df2017_2.describe(include='all').loc['mean']['generation solar'])
df2017_1.describe(include='all').loc['mean']
solar.append(df2017_1.describe(include='all').loc['mean']['generation solar'])
df2016_12.describe(include='all').loc['mean']
solar.append(df2016_12.describe(include='all').loc['mean']['generation solar'])
df2016_11.describe(include='all').loc['mean']
solar.append(df2016_11.describe(include='all').loc['mean']['generation solar'])
df2016_10.describe(include='all').loc['mean']
solar.append(df2016_10.describe(include='all').loc['mean']['generation solar'])
df2016_9.describe(include='all').loc['mean']
solar.append(df2016_9.describe(include='all').loc['mean']['generation solar'])
df2016_8.describe(include='all').loc['mean']
solar.append(df2016_8.describe(include='all').loc['mean']['generation solar'])
df2016_7.describe(include='all').loc['mean']
solar.append(df2016_7.describe(include='all').loc['mean']['generation solar'])
df2016_6.describe(include='all').loc['mean']
solar.append(df2016_6.describe(include='all').loc['mean']['generation solar'])
df2016_5.describe(include='all').loc['mean']
```

```

solar.append(df2016_5.describe(include='all').loc['mean']['generation solar'])
df2016_4.describe(include='all').loc['mean']
solar.append(df2016_4.describe(include='all').loc['mean']['generation solar'])
df2016_3.describe(include='all').loc['mean']
solar.append(df2016_3.describe(include='all').loc['mean']['generation solar'])
df2016_2.describe(include='all').loc['mean']
solar.append(df2016_2.describe(include='all').loc['mean']['generation solar'])
df2016_1.describe(include='all').loc['mean']
solar.append(df2016_1.describe(include='all').loc['mean']['generation solar'])
df2015_12.describe(include='all').loc['mean']
solar.append(df2015_12.describe(include='all').loc['mean']['generation solar'])
df2015_11.describe(include='all').loc['mean']
solar.append(df2015_11.describe(include='all').loc['mean']['generation solar'])
df2015_10.describe(include='all').loc['mean']
solar.append(df2015_10.describe(include='all').loc['mean']['generation solar'])
df2015_9.describe(include='all').loc['mean']
solar.append(df2015_9.describe(include='all').loc['mean']['generation solar'])
df2015_8.describe(include='all').loc['mean']
solar.append(df2015_8.describe(include='all').loc['mean']['generation solar'])
df2015_7.describe(include='all').loc['mean']
solar.append(df2015_7.describe(include='all').loc['mean']['generation solar'])
df2015_6.describe(include='all').loc['mean']
solar.append(df2015_6.describe(include='all').loc['mean']['generation solar'])
df2015_5.describe(include='all').loc['mean']
solar.append(df2015_5.describe(include='all').loc['mean']['generation solar'])
df2015_4.describe(include='all').loc['mean']
solar.append(df2015_4.describe(include='all').loc['mean']['generation solar'])
df2015_3.describe(include='all').loc['mean']
solar.append(df2015_3.describe(include='all').loc['mean']['generation solar'])
df2015_2.describe(include='all').loc['mean']
solar.append(df2015_2.describe(include='all').loc['mean']['generation solar'])
df2015_1.describe(include='all').loc['mean']
solar.append(df2015_1.describe(include='all').loc['mean']['generation solar'])

solar.reverse() # Average monthly values
print(solar)

```

[1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657, 1998.6606397774688, 1973.7405913978494, 1738.9502688172042, 1591.2902777777779, 1082.279569892473, 1171.719444444444, 874.3135935397039, 1025.741935483871, 1258.058908045977, 1382.9057873485867, 1390.41666666666667, 1689.1814516129032, 1942.8291666666667, 1915.7752355316286, 1835.8521505376343, 1548.213888888889, 1013.2832214765101, 951.38194444444445, 914.9758064516129, 1088.4180107526881, 1157.4613095238096, 1415.0915208613728, 1468.8277777777778, 1800.0524193548388, 1919.0236111111112, 2025.119623655914, 1754.159946236559, 1731.3666666666666, 1344.3261744966444, 1180.645833333333, 1035.4260752688172, 1051.4193548387098, 1246.110119047619, 1352.3189771197847, 1501.647222222221, 1526.899193548387, 1855.1972222222223, 2010.3458950201884, 1706.9301075268818, 1510.6527777777778, 999.786577181208, 831.6680555555556, 875.5900537634409]

In []:

In []:

```

biomass= []
df2018_12.describe(include='all').loc['mean']
biomass.append(df2018_12.describe(include='all').loc['mean']['generation biomass'])
df2018_11.describe(include='all').loc['mean']
biomass.append(df2018_11.describe(include='all').loc['mean']['generation biomass'])
df2018_10.describe(include='all').loc['mean']
biomass.append(df2018_10.describe(include='all').loc['mean']['generation biomass'])
df2018_9.describe(include='all').loc['mean']
biomass.append(df2018_9.describe(include='all').loc['mean']['generation biomass'])
df2018_8.describe(include='all').loc['mean']
biomass.append(df2018_8.describe(include='all').loc['mean']['generation biomass'])
df2018_7.describe(include='all').loc['mean']
biomass.append(df2018_7.describe(include='all').loc['mean']['generation biomass'])
df2018_6.describe(include='all').loc['mean']
biomass.append(df2018_6.describe(include='all').loc['mean']['generation biomass'])
df2018_5.describe(include='all').loc['mean']
biomass.append(df2018_5.describe(include='all').loc['mean']['generation biomass'])
df2018_4.describe(include='all').loc['mean']
biomass.append(df2018_4.describe(include='all').loc['mean']['generation biomass'])
df2018_3.describe(include='all').loc['mean']
biomass.append(df2018_3.describe(include='all').loc['mean']['generation biomass'])
df2018_2.describe(include='all').loc['mean']
biomass.append(df2018_2.describe(include='all').loc['mean']['generation biomass'])
df2018_1.describe(include='all').loc['mean']
biomass.append(df2018_1.describe(include='all').loc['mean']['generation biomass'])
df2017_12.describe(include='all').loc['mean']
biomass.append(df2017_12.describe(include='all').loc['mean']['generation biomass'])
df2017_11.describe(include='all').loc['mean']
biomass.append(df2017_11.describe(include='all').loc['mean']['generation biomass'])
df2017_10.describe(include='all').loc['mean']
biomass.append(df2017_10.describe(include='all').loc['mean']['generation biomass'])
df2017_9.describe(include='all').loc['mean']
biomass.append(df2017_9.describe(include='all').loc['mean']['generation biomass'])

```

```

df2017_8.describe(include='all').loc['mean']
biomass.append(df2017_8.describe(include='all').loc['mean']['generation biomass'])
df2017_7.describe(include='all').loc['mean']
biomass.append(df2017_7.describe(include='all').loc['mean']['generation biomass'])
df2017_6.describe(include='all').loc['mean']
biomass.append(df2017_6.describe(include='all').loc['mean']['generation biomass'])
df2017_5.describe(include='all').loc['mean']
biomass.append(df2017_5.describe(include='all').loc['mean']['generation biomass'])
df2017_4.describe(include='all').loc['mean']
biomass.append(df2017_4.describe(include='all').loc['mean']['generation biomass'])
df2017_3.describe(include='all').loc['mean']
biomass.append(df2017_3.describe(include='all').loc['mean']['generation biomass'])
df2017_2.describe(include='all').loc['mean']
biomass.append(df2017_2.describe(include='all').loc['mean']['generation biomass'])
df2017_1.describe(include='all').loc['mean']
biomass.append(df2017_1.describe(include='all').loc['mean']['generation biomass'])
df2016_12.describe(include='all').loc['mean']
biomass.append(df2016_12.describe(include='all').loc['mean']['generation biomass'])
df2016_11.describe(include='all').loc['mean']
biomass.append(df2016_11.describe(include='all').loc['mean']['generation biomass'])
df2016_10.describe(include='all').loc['mean']
biomass.append(df2016_10.describe(include='all').loc['mean']['generation biomass'])
df2016_9.describe(include='all').loc['mean']
biomass.append(df2016_9.describe(include='all').loc['mean']['generation biomass'])
df2016_8.describe(include='all').loc['mean']
biomass.append(df2016_8.describe(include='all').loc['mean']['generation biomass'])
df2016_7.describe(include='all').loc['mean']
biomass.append(df2016_7.describe(include='all').loc['mean']['generation biomass'])
df2016_6.describe(include='all').loc['mean']
biomass.append(df2016_6.describe(include='all').loc['mean']['generation biomass'])
df2016_5.describe(include='all').loc['mean']
biomass.append(df2016_5.describe(include='all').loc['mean']['generation biomass'])
df2016_4.describe(include='all').loc['mean']
biomass.append(df2016_4.describe(include='all').loc['mean']['generation biomass'])
df2016_3.describe(include='all').loc['mean']
biomass.append(df2016_3.describe(include='all').loc['mean']['generation biomass'])
df2016_2.describe(include='all').loc['mean']
biomass.append(df2016_2.describe(include='all').loc['mean']['generation biomass'])
df2016_1.describe(include='all').loc['mean']
biomass.append(df2016_1.describe(include='all').loc['mean']['generation biomass'])
df2015_12.describe(include='all').loc['mean']
biomass.append(df2015_12.describe(include='all').loc['mean']['generation biomass'])
df2015_11.describe(include='all').loc['mean']
biomass.append(df2015_11.describe(include='all').loc['mean']['generation biomass'])
df2015_10.describe(include='all').loc['mean']
biomass.append(df2015_10.describe(include='all').loc['mean']['generation biomass'])
df2015_9.describe(include='all').loc['mean']
biomass.append(df2015_9.describe(include='all').loc['mean']['generation biomass'])
df2015_8.describe(include='all').loc['mean']
biomass.append(df2015_8.describe(include='all').loc['mean']['generation biomass'])
df2015_7.describe(include='all').loc['mean']
biomass.append(df2015_7.describe(include='all').loc['mean']['generation biomass'])
df2015_6.describe(include='all').loc['mean']
biomass.append(df2015_6.describe(include='all').loc['mean']['generation biomass'])
df2015_5.describe(include='all').loc['mean']
biomass.append(df2015_5.describe(include='all').loc['mean']['generation biomass'])
df2015_4.describe(include='all').loc['mean']
biomass.append(df2015_4.describe(include='all').loc['mean']['generation biomass'])
df2015_3.describe(include='all').loc['mean']
biomass.append(df2015_3.describe(include='all').loc['mean']['generation biomass'])
df2015_2.describe(include='all').loc['mean']
biomass.append(df2015_2.describe(include='all').loc['mean']['generation biomass'])
df2015_1.describe(include='all').loc['mean']
biomass.append(df2015_1.describe(include='all').loc['mean']['generation biomass'])

biomass.reverse() # Average monthly values

print(biomass)

```

[483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 485.9986091794
1584, 512.4879032258065, 518.9475806451613, 517.106944444445, 519.5564516129032, 503.6166666666667, 483.9246298
7886944, 459.9959677419355, 430.3448275862069, 429.7442799461642, 286.890277777778, 337.2029569892473, 340.3375
, 351.8896366083446, 369.64516129032256, 349.3916666666665, 336.2187919463087, 355.57301808066757, 338.77688172
04301, 347.28360215053766, 351.364583333333, 284.40242261103634, 280.761111111111, 353.5040322580645, 339.9791
666666667, 367.7069892473118, 362.8790322580645, 351.9680555555557, 343.938255033557, 354.5930555555557, 347.1
733870967742, 343.8481182795699, 360.51190476190476, 307.2543741588156, 299.3930555555556, 318.30913978494624, 3
51.73333333333335, 367.2207267833109, 361.3252688172043, 355.4680555555557, 317.9275167785235, 342.102777777777
76, 320.16129032258067]

In []: len(biomass)

Out[]: 48

In []: fossil_gas =[]


```

df2015_7.describe(include='all').loc['mean']
fossil_gas.append(df2015_7.describe(include='all').loc['mean'][ "generation fossil gas"])
df2015_6.describe(include='all').loc['mean']
fossil_gas.append(df2015_6.describe(include='all').loc['mean'][ "generation fossil gas"])
df2015_5.describe(include='all').loc['mean']
fossil_gas.append(df2015_5.describe(include='all').loc['mean'][ "generation fossil gas"])
df2015_4.describe(include='all').loc['mean']
fossil_gas.append(df2015_4.describe(include='all').loc['mean'][ "generation fossil gas"])
df2015_3.describe(include='all').loc['mean']
fossil_gas.append(df2015_3.describe(include='all').loc['mean'][ "generation fossil gas"])
df2015_2.describe(include='all').loc['mean']
fossil_gas.append(df2015_2.describe(include='all').loc['mean'][ "generation fossil gas"])
df2015_1.describe(include='all').loc['mean']
fossil_gas.append(df2015_1.describe(include='all').loc['mean'][ "generation fossil gas"])

fossil_gas.reverse() # Average monthly values

print(fossil_gas)

```

```
[4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915, 4934.930458970
793, 6529.490591397849, 4919.491935483871, 5076.581944444444, 5231.5, 5387.741666666667, 5062.588156123822, 5271
.1196236559135, 4450.360632183908, 4336.594885598924, 4263.481944444445, 4508.034946236559, 4994.327777777778, 5
475.64333781965, 4869.987903225807, 4643.55, 6222.820134228188, 6439.015277777778, 6176.176075268817, 6412.59139
7849463, 5561.144345238095, 5337.418573351279, 5169.840277777777, 5493.271505376344, 7194.280555555555, 7366.073
924731183, 6701.935483870968, 7092.451388888889, 6961.8, 8534.818055555555, 5835.915322580645, 5687.715053763441
, 5468.040178571428, 4936.013458950202, 4745.273611111111, 5777.162634408603, 5807.555555555556, 5865.6917900403
77, 6119.346774193548, 5842.0625, 6127.481879194631, 6945.804166666667, 6463.5362903225805]
```

In []: len(fossil_gas)

Out[]: 48

```

In [ ]: other_renewable= []
df2018_12.describe(include='all').loc['mean']
other_renewable.append(df2018_12.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_11.describe(include='all').loc['mean']
other_renewable.append(df2018_11.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_10.describe(include='all').loc['mean']
other_renewable.append(df2018_10.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_9.describe(include='all').loc['mean']
other_renewable.append(df2018_9.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_8.describe(include='all').loc['mean']
other_renewable.append(df2018_8.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_7.describe(include='all').loc['mean']
other_renewable.append(df2018_7.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_6.describe(include='all').loc['mean']
other_renewable.append(df2018_6.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_5.describe(include='all').loc['mean']
other_renewable.append(df2018_5.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_4.describe(include='all').loc['mean']
other_renewable.append(df2018_4.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_3.describe(include='all').loc['mean']
other_renewable.append(df2018_3.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_2.describe(include='all').loc['mean']
other_renewable.append(df2018_2.describe(include='all').loc['mean'][ 'generation other renewable'])
df2018_1.describe(include='all').loc['mean']
other_renewable.append(df2018_1.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_12.describe(include='all').loc['mean']
other_renewable.append(df2017_12.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_11.describe(include='all').loc['mean']
other_renewable.append(df2017_11.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_10.describe(include='all').loc['mean']
other_renewable.append(df2017_10.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_9.describe(include='all').loc['mean']
other_renewable.append(df2017_9.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_8.describe(include='all').loc['mean']
other_renewable.append(df2017_8.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_7.describe(include='all').loc['mean']
other_renewable.append(df2017_7.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_6.describe(include='all').loc['mean']
other_renewable.append(df2017_6.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_5.describe(include='all').loc['mean']
other_renewable.append(df2017_5.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_4.describe(include='all').loc['mean']
other_renewable.append(df2017_4.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_3.describe(include='all').loc['mean']
other_renewable.append(df2017_3.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_2.describe(include='all').loc['mean']
other_renewable.append(df2017_2.describe(include='all').loc['mean'][ 'generation other renewable'])
df2017_1.describe(include='all').loc['mean']
other_renewable.append(df2017_1.describe(include='all').loc['mean'][ 'generation other renewable'])
df2016_12.describe(include='all').loc['mean']
other_renewable.append(df2016_12.describe(include='all').loc['mean'][ 'generation other renewable'])

```

```

df2016_11.describe(include='all').loc['mean']
other_renewable.append(df2016_11.describe(include='all').loc['mean']['generation other renewable'])
df2016_10.describe(include='all').loc['mean']
other_renewable.append(df2016_10.describe(include='all').loc['mean']['generation other renewable'])
df2016_9.describe(include='all').loc['mean']
other_renewable.append(df2016_9.describe(include='all').loc['mean']['generation other renewable'])
df2016_8.describe(include='all').loc['mean']
other_renewable.append(df2016_8.describe(include='all').loc['mean']['generation other renewable'])
df2016_7.describe(include='all').loc['mean']
other_renewable.append(df2016_7.describe(include='all').loc['mean']['generation other renewable'])
df2016_6.describe(include='all').loc['mean']
other_renewable.append(df2016_6.describe(include='all').loc['mean']['generation other renewable'])
df2016_5.describe(include='all').loc['mean']
other_renewable.append(df2016_5.describe(include='all').loc['mean']['generation other renewable'])
df2016_4.describe(include='all').loc['mean']
other_renewable.append(df2016_4.describe(include='all').loc['mean']['generation other renewable'])
df2016_3.describe(include='all').loc['mean']
other_renewable.append(df2016_3.describe(include='all').loc['mean']['generation other renewable'])
df2016_2.describe(include='all').loc['mean']
other_renewable.append(df2016_2.describe(include='all').loc['mean']['generation other renewable'])
df2016_1.describe(include='all').loc['mean']
other_renewable.append(df2016_1.describe(include='all').loc['mean']['generation other renewable'])
df2015_12.describe(include='all').loc['mean']
other_renewable.append(df2015_12.describe(include='all').loc['mean']['generation other renewable'])
df2015_11.describe(include='all').loc['mean']
other_renewable.append(df2015_11.describe(include='all').loc['mean']['generation other renewable'])
df2015_10.describe(include='all').loc['mean']
other_renewable.append(df2015_10.describe(include='all').loc['mean']['generation other renewable'])
df2015_9.describe(include='all').loc['mean']
other_renewable.append(df2015_9.describe(include='all').loc['mean']['generation other renewable'])
df2015_8.describe(include='all').loc['mean']
other_renewable.append(df2015_8.describe(include='all').loc['mean']['generation other renewable'])
df2015_7.describe(include='all').loc['mean']
other_renewable.append(df2015_7.describe(include='all').loc['mean']['generation other renewable'])
df2015_6.describe(include='all').loc['mean']
other_renewable.append(df2015_6.describe(include='all').loc['mean']['generation other renewable'])
df2015_5.describe(include='all').loc['mean']
other_renewable.append(df2015_5.describe(include='all').loc['mean']['generation other renewable'])
df2015_4.describe(include='all').loc['mean']
other_renewable.append(df2015_4.describe(include='all').loc['mean']['generation other renewable'])
df2015_3.describe(include='all').loc['mean']
other_renewable.append(df2015_3.describe(include='all').loc['mean']['generation other renewable'])
df2015_2.describe(include='all').loc['mean']
other_renewable.append(df2015_2.describe(include='all').loc['mean']['generation other renewable'])
df2015_1.describe(include='all').loc['mean']
other_renewable.append(df2015_1.describe(include='all').loc['mean']['generation other renewable'])

other_renewable.reverse() # Average monthly values
print(other_renewable)

```

[70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306407, 70.56586021505376, 65.51182197496523, 68.3252688172043, 68.1760752688172, 67.96666666666667, 70.66935483870968, 70.3125, 71.62314939434724, 77.95430107526882, 74.45402298850574, 73.40242261103634, 78.275, 79.28225806451613, 83.48333333333333, 78.6271870794078, 78.43010752688173, 83.79166666666667, 83.78120805369127, 85.95694444444445, 90.09005376344086, 99.20430107526882, 94.90922619047619, 95.6850605652759, 95.97083333333333, 96.74596774193549, 92.10555555555555, 93.39650537634408, 92.24731182795699, 94.87222222222222, 95.6268456375839, 94.90138888888889, 91.81586021505376, 96.98252688172043, 96.75744047619048, 97.64199192462988, 97.15138888888889, 100.59005376344086, 101.10972222222222, 96.71736204576042, 96.61559139784946, 100.19444444444444, 98.61744966442953, 96.4625, 95.88172043010752]

In []:

```

In [ ]:
nuclear = []
df2018_12.describe(include='all').loc['mean']
nuclear.append(df2018_12.describe(include='all').loc['mean']['generation nuclear'])
df2018_11.describe(include='all').loc['mean']
nuclear.append(df2018_11.describe(include='all').loc['mean']['generation nuclear'])
df2018_10.describe(include='all').loc['mean']
nuclear.append(df2018_10.describe(include='all').loc['mean']['generation nuclear'])
df2018_9.describe(include='all').loc['mean']
nuclear.append(df2018_9.describe(include='all').loc['mean']['generation nuclear'])
df2018_8.describe(include='all').loc['mean']
nuclear.append(df2018_8.describe(include='all').loc['mean']['generation nuclear'])
df2018_7.describe(include='all').loc['mean']
nuclear.append(df2018_7.describe(include='all').loc['mean']['generation nuclear'])
df2018_6.describe(include='all').loc['mean']
nuclear.append(df2018_6.describe(include='all').loc['mean']['generation nuclear'])
df2018_5.describe(include='all').loc['mean']
nuclear.append(df2018_5.describe(include='all').loc['mean']['generation nuclear'])
df2018_4.describe(include='all').loc['mean']
nuclear.append(df2018_4.describe(include='all').loc['mean']['generation nuclear'])
df2018_3.describe(include='all').loc['mean']
nuclear.append(df2018_3.describe(include='all').loc['mean']['generation nuclear'])
df2018_2.describe(include='all').loc['mean']
nuclear.append(df2018_2.describe(include='all').loc['mean']['generation nuclear'])

```



```
[6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 5659.276773296245, 6483.623655913979, 6437.845430107527, 6466.175, 5854.012096774193, 5973.075, 6626.029609690444, 6223.8494623655915, 6159.264367816092, 6699.888290713325, 6706.406944444445, 5714.009408602151, 6647.4638888888885, 6628.9220430107525, 6633.3494623655915, 6675.7555555555555, 6576.6859060402685, 5534.297222222222, 6325.970430107527, 6769.916666666667, 6739.267857142857, 6755.951547779273, 6676.383333333333, 5561.240591397849, 6063.859722222222, 6117.403225806452, 6675.846774193548, 6427.688888888889, 6003.754362416107, 5565.216666666666, 6815.138440860215, 6723.689516129032, 6429.3735119047615, 6091.685060565276, 5504.175, 5465.200268817204, 5603.227777777778, 6121.515477792732, 6655.32123655914, 6621.998611111111, 6539.120805369127, 5403.497222222222, 5821.1518817204305]
```

In []:

In []:

```
wind = []
df2018_12.describe(include='all').loc['mean']
wind.append(df2018_12.describe(include='all').loc['mean']['generation wind onshore'])
df2018_11.describe(include='all').loc['mean']
wind.append(df2018_11.describe(include='all').loc['mean']['generation wind onshore'])
df2018_10.describe(include='all').loc['mean']
wind.append(df2018_10.describe(include='all').loc['mean']['generation wind onshore'])
df2018_9.describe(include='all').loc['mean']
wind.append(df2018_9.describe(include='all').loc['mean']['generation wind onshore'])
df2018_8.describe(include='all').loc['mean']
wind.append(df2018_8.describe(include='all').loc['mean']['generation wind onshore'])
df2018_7.describe(include='all').loc['mean']
wind.append(df2018_7.describe(include='all').loc['mean']['generation wind onshore'])
df2018_6.describe(include='all').loc['mean']
wind.append(df2018_6.describe(include='all').loc['mean']['generation wind onshore'])
df2018_5.describe(include='all').loc['mean']
wind.append(df2018_5.describe(include='all').loc['mean']['generation wind onshore'])
df2018_4.describe(include='all').loc['mean']
wind.append(df2018_4.describe(include='all').loc['mean']['generation wind onshore'])
df2018_3.describe(include='all').loc['mean']
wind.append(df2018_3.describe(include='all').loc['mean']['generation wind onshore'])
df2018_2.describe(include='all').loc['mean']
wind.append(df2018_2.describe(include='all').loc['mean']['generation wind onshore'])
df2018_1.describe(include='all').loc['mean']
wind.append(df2018_1.describe(include='all').loc['mean']['generation wind onshore'])
df2017_12.describe(include='all').loc['mean']
wind.append(df2017_12.describe(include='all').loc['mean']['generation wind onshore'])
df2017_11.describe(include='all').loc['mean']
wind.append(df2017_11.describe(include='all').loc['mean']['generation wind onshore'])
df2017_10.describe(include='all').loc['mean']
wind.append(df2017_10.describe(include='all').loc['mean']['generation wind onshore'])
df2017_9.describe(include='all').loc['mean']
wind.append(df2017_9.describe(include='all').loc['mean']['generation wind onshore'])
df2017_8.describe(include='all').loc['mean']
wind.append(df2017_8.describe(include='all').loc['mean']['generation wind onshore'])
df2017_7.describe(include='all').loc['mean']
wind.append(df2017_7.describe(include='all').loc['mean']['generation wind onshore'])
df2017_6.describe(include='all').loc['mean']
wind.append(df2017_6.describe(include='all').loc['mean']['generation wind onshore'])
df2017_5.describe(include='all').loc['mean']
wind.append(df2017_5.describe(include='all').loc['mean']['generation wind onshore'])
df2017_4.describe(include='all').loc['mean']
wind.append(df2017_4.describe(include='all').loc['mean']['generation wind onshore'])
df2017_3.describe(include='all').loc['mean']
wind.append(df2017_3.describe(include='all').loc['mean']['generation wind onshore'])
df2017_2.describe(include='all').loc['mean']
wind.append(df2017_2.describe(include='all').loc['mean']['generation wind onshore'])
df2017_1.describe(include='all').loc['mean']
wind.append(df2017_1.describe(include='all').loc['mean']['generation wind onshore'])
df2016_12.describe(include='all').loc['mean']
wind.append(df2016_12.describe(include='all').loc['mean']['generation wind onshore'])
df2016_11.describe(include='all').loc['mean']
wind.append(df2016_11.describe(include='all').loc['mean']['generation wind onshore'])
df2016_10.describe(include='all').loc['mean']
wind.append(df2016_10.describe(include='all').loc['mean']['generation wind onshore'])
df2016_9.describe(include='all').loc['mean']
wind.append(df2016_9.describe(include='all').loc['mean']['generation wind onshore'])
df2016_8.describe(include='all').loc['mean']
wind.append(df2016_8.describe(include='all').loc['mean']['generation wind onshore'])
df2016_7.describe(include='all').loc['mean']
wind.append(df2016_7.describe(include='all').loc['mean']['generation wind onshore'])
df2016_6.describe(include='all').loc['mean']
wind.append(df2016_6.describe(include='all').loc['mean']['generation wind onshore'])
df2016_5.describe(include='all').loc['mean']
wind.append(df2016_5.describe(include='all').loc['mean']['generation wind onshore'])
df2016_4.describe(include='all').loc['mean']
wind.append(df2016_4.describe(include='all').loc['mean']['generation wind onshore'])
df2016_3.describe(include='all').loc['mean']
wind.append(df2016_3.describe(include='all').loc['mean']['generation wind onshore'])
df2016_2.describe(include='all').loc['mean']
wind.append(df2016_2.describe(include='all').loc['mean']['generation wind onshore'])
df2016_1.describe(include='all').loc['mean']
```

```

wind.append(df2016_1.describe(include='all').loc['mean']['generation wind onshore'])
df2015_12.describe(include='all').loc['mean']
wind.append(df2015_12.describe(include='all').loc['mean']['generation wind onshore'])
df2015_11.describe(include='all').loc['mean']
wind.append(df2015_11.describe(include='all').loc['mean']['generation wind onshore'])
df2015_10.describe(include='all').loc['mean']
wind.append(df2015_10.describe(include='all').loc['mean']['generation wind onshore'])
df2015_9.describe(include='all').loc['mean']
wind.append(df2015_9.describe(include='all').loc['mean']['generation wind onshore'])
df2015_8.describe(include='all').loc['mean']
wind.append(df2015_8.describe(include='all').loc['mean']['generation wind onshore'])
df2015_7.describe(include='all').loc['mean']
wind.append(df2015_7.describe(include='all').loc['mean']['generation wind onshore'])
df2015_6.describe(include='all').loc['mean']
wind.append(df2015_6.describe(include='all').loc['mean']['generation wind onshore'])
df2015_5.describe(include='all').loc['mean']
wind.append(df2015_5.describe(include='all').loc['mean']['generation wind onshore'])
df2015_4.describe(include='all').loc['mean']
wind.append(df2015_4.describe(include='all').loc['mean']['generation wind onshore'])
df2015_3.describe(include='all').loc['mean']
wind.append(df2015_3.describe(include='all').loc['mean']['generation wind onshore'])
df2015_2.describe(include='all').loc['mean']
wind.append(df2015_2.describe(include='all').loc['mean']['generation wind onshore'])
df2015_1.describe(include='all').loc['mean']
wind.append(df2015_1.describe(include='all').loc['mean']['generation wind onshore'])

wind.reverse() # Average monthly values
print(wind)

```

```
[7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.49513212795
6, 3955.2540322580644, 4856.173387096775, 4323.8, 4597.236559139785, 4493.763888888889, 4943.14131897712, 4993.3
99193548387, 6534.337643678161, 5884.690444145357, 5343.191666666667, 5309.384408602151, 5638.5375, 5715.8331090
17497, 5103.193548387097, 5484.695833333333, 4788.928859060403, 5766.394444444444, 4519.7661290322585, 6483.2983
87096775, 5330.943452380952, 5623.149394347241, 5741.669444444445, 5031.903225806452, 5586.527777777777, 4799.47
7150537635, 4348.514784946236, 3883.879166666666, 5234.8, 5200.188888888889, 7257.2553763440865, 6197.444892473
119, 6864.555059523809, 8771.240915208613, 5289.247222222222, 4389.743279569892, 4779.433333333333, 4059.9071332
43607, 4358.579301075269, 4265.988888888889, 6156.939597315436, 6057.558333333333, 5886.720430107527]
```

In []:

```

In [ ]:
fossil_oil= []
df2018_12.describe(include='all').loc['mean']
fossil_oil.append(df2018_12.describe(include='all').loc['mean']['generation fossil oil'])
df2018_11.describe(include='all').loc['mean']
fossil_oil.append(df2018_11.describe(include='all').loc['mean']['generation fossil oil'])
df2018_10.describe(include='all').loc['mean']
fossil_oil.append(df2018_10.describe(include='all').loc['mean']['generation fossil oil'])
df2018_9.describe(include='all').loc['mean']
fossil_oil.append(df2018_9.describe(include='all').loc['mean']['generation fossil oil'])
df2018_8.describe(include='all').loc['mean']
fossil_oil.append(df2018_8.describe(include='all').loc['mean']['generation fossil oil'])
df2018_7.describe(include='all').loc['mean']
fossil_oil.append(df2018_7.describe(include='all').loc['mean']['generation fossil oil'])
df2018_6.describe(include='all').loc['mean']
fossil_oil.append(df2018_6.describe(include='all').loc['mean']['generation fossil oil'])
df2018_5.describe(include='all').loc['mean']
fossil_oil.append(df2018_5.describe(include='all').loc['mean']['generation fossil oil'])
df2018_4.describe(include='all').loc['mean']
fossil_oil.append(df2018_4.describe(include='all').loc['mean']['generation fossil oil'])
df2018_3.describe(include='all').loc['mean']
fossil_oil.append(df2018_3.describe(include='all').loc['mean']['generation fossil oil'])
df2018_2.describe(include='all').loc['mean']
fossil_oil.append(df2018_2.describe(include='all').loc['mean']['generation fossil oil'])
df2018_1.describe(include='all').loc['mean']
fossil_oil.append(df2018_1.describe(include='all').loc['mean']['generation fossil oil'])
df2017_12.describe(include='all').loc['mean']
fossil_oil.append(df2017_12.describe(include='all').loc['mean']['generation fossil oil'])
df2017_11.describe(include='all').loc['mean']
fossil_oil.append(df2017_11.describe(include='all').loc['mean']['generation fossil oil'])
df2017_10.describe(include='all').loc['mean']
fossil_oil.append(df2017_10.describe(include='all').loc['mean']['generation fossil oil'])
df2017_9.describe(include='all').loc['mean']
fossil_oil.append(df2017_9.describe(include='all').loc['mean']['generation fossil oil'])
df2017_8.describe(include='all').loc['mean']
fossil_oil.append(df2017_8.describe(include='all').loc['mean']['generation fossil oil'])
df2017_7.describe(include='all').loc['mean']
fossil_oil.append(df2017_7.describe(include='all').loc['mean']['generation fossil oil'])
df2017_6.describe(include='all').loc['mean']
fossil_oil.append(df2017_6.describe(include='all').loc['mean']['generation fossil oil'])
df2017_5.describe(include='all').loc['mean']
fossil_oil.append(df2017_5.describe(include='all').loc['mean']['generation fossil oil'])
df2017_4.describe(include='all').loc['mean']
fossil_oil.append(df2017_4.describe(include='all').loc['mean']['generation fossil oil'])
df2017_3.describe(include='all').loc['mean']

```

```

fossil_oil.append(df2017_3.describe(include='all').loc['mean']['generation fossil oil'])
df2017_2.describe(include='all').loc['mean']
fossil_oil.append(df2017_2.describe(include='all').loc['mean']['generation fossil oil'])
df2017_1.describe(include='all').loc['mean']
fossil_oil.append(df2017_1.describe(include='all').loc['mean']['generation fossil oil'])
df2016_12.describe(include='all').loc['mean']
fossil_oil.append(df2016_12.describe(include='all').loc['mean']['generation fossil oil'])
df2016_11.describe(include='all').loc['mean']
fossil_oil.append(df2016_11.describe(include='all').loc['mean']['generation fossil oil'])
df2016_10.describe(include='all').loc['mean']
fossil_oil.append(df2016_10.describe(include='all').loc['mean']['generation fossil oil'])
df2016_9.describe(include='all').loc['mean']
fossil_oil.append(df2016_9.describe(include='all').loc['mean']['generation fossil oil'])
df2016_8.describe(include='all').loc['mean']
fossil_oil.append(df2016_8.describe(include='all').loc['mean']['generation fossil oil'])
df2016_7.describe(include='all').loc['mean']
fossil_oil.append(df2016_7.describe(include='all').loc['mean']['generation fossil oil'])
df2016_6.describe(include='all').loc['mean']
fossil_oil.append(df2016_6.describe(include='all').loc['mean']['generation fossil oil'])
df2016_5.describe(include='all').loc['mean']
fossil_oil.append(df2016_5.describe(include='all').loc['mean']['generation fossil oil'])
df2016_4.describe(include='all').loc['mean']
fossil_oil.append(df2016_4.describe(include='all').loc['mean']['generation fossil oil'])
df2016_3.describe(include='all').loc['mean']
fossil_oil.append(df2016_3.describe(include='all').loc['mean']['generation fossil oil'])
df2016_2.describe(include='all').loc['mean']
fossil_oil.append(df2016_2.describe(include='all').loc['mean']['generation fossil oil'])
df2016_1.describe(include='all').loc['mean']
fossil_oil.append(df2016_1.describe(include='all').loc['mean']['generation fossil oil'])
df2015_12.describe(include='all').loc['mean']
fossil_oil.append(df2015_12.describe(include='all').loc['mean']['generation fossil oil'])
df2015_11.describe(include='all').loc['mean']
fossil_oil.append(df2015_11.describe(include='all').loc['mean']['generation fossil oil'])
df2015_10.describe(include='all').loc['mean']
fossil_oil.append(df2015_10.describe(include='all').loc['mean']['generation fossil oil'])
df2015_9.describe(include='all').loc['mean']
fossil_oil.append(df2015_9.describe(include='all').loc['mean']['generation fossil oil'])
df2015_8.describe(include='all').loc['mean']
fossil_oil.append(df2015_8.describe(include='all').loc['mean']['generation fossil oil'])
df2015_7.describe(include='all').loc['mean']
fossil_oil.append(df2015_7.describe(include='all').loc['mean']['generation fossil oil'])
df2015_6.describe(include='all').loc['mean']
fossil_oil.append(df2015_6.describe(include='all').loc['mean']['generation fossil oil'])
df2015_5.describe(include='all').loc['mean']
fossil_oil.append(df2015_5.describe(include='all').loc['mean']['generation fossil oil'])
df2015_4.describe(include='all').loc['mean']
fossil_oil.append(df2015_4.describe(include='all').loc['mean']['generation fossil oil'])
df2015_3.describe(include='all').loc['mean']
fossil_oil.append(df2015_3.describe(include='all').loc['mean']['generation fossil oil'])
df2015_2.describe(include='all').loc['mean']
fossil_oil.append(df2015_2.describe(include='all').loc['mean']['generation fossil oil'])
df2015_1.describe(include='all').loc['mean']
fossil_oil.append(df2015_1.describe(include='all').loc['mean']['generation fossil oil'])

fossil_oil.reverse() # Average monthly values
print(fossil_oil)

```

[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963
839, 360.23387096774195, 323.9139784946237, 343.4916666666667, 323.39112903225805, 346.06388888888887, 330.65141
31897712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875, 277.9153225806452, 289.829166666
66665, 286.6900269541779, 283.30645161290323, 281.386111111111, 276.9959731543624, 271.56944444444446, 276.6397
8494623655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.4902777777778, 291.9206989247312, 299
.39305555555556, 308.2002688172043, 306.23521505376345, 310.8013888888889, 287.4187919463087, 303.543055555555555,
293.9260752688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.629166666666666, 280.45026881720
43, 285.12083333333334, 281.5450874831763, 283.333333333333, 296.136111111111, 291.2993288590604, 272.98333333
33335, 269.02150537634407]

```

In [ ]: Fossil_Hard = []
df2018_12.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_12.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_11.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_11.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_10.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_10.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_9.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_9.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_8.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_8.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_7.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_7.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_6.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_6.describe(include='all').loc['mean']['generation fossil hard coal'])
df2018_5.describe(include='all').loc['mean']
Fossil_Hard.append(df2018_5.describe(include='all').loc['mean']['generation fossil hard coal'])

```



```
[5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.6129032258063, 6207.095966620  
306, 6748.469086021505, 5859.370967741936, 5653.873611111111, 5788.694892473119, 5982.631944444444, 5323.6769851  
95155, 4204.615591397849, 3027.655172413793, 2985.886944818304, 2226.186111111111, 2226.65188172043, 2877.080555  
5555557, 4003.9165545087485, 4172.501344086021, 4208.013888888889, 4271.040268456376, 4581.631944444444, 4984.29  
5698924731, 5533.782258064516, 4382.5327380952385, 2967.8492597577388, 3119.322222222222, 4428.919354838709, 509  
1.152777777777, 4561.283602150537, 3786.2325268817203, 4010.465277777778, 4321.130201342282, 5491.216666666666,  
4330.653225806452, 3783.6223118279568, 4144.389880952381, 2157.1830417227457, 2775.8791666666666, 3674.151881720  
43, 3059.616666666667, 4014.707940780619, 4121.565860215053, 4651.018055555555, 3782.8161073825504, 4838.7777777  
77777, 3365.7580645161293]
```

In []:

In []:

```
fossil_brown = []
df2018_12.describe(include='all').loc['mean']
fossil_brown.append(df2018_12.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_11.describe(include='all').loc['mean']
fossil_brown.append(df2018_11.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_10.describe(include='all').loc['mean']
fossil_brown.append(df2018_10.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_9.describe(include='all').loc['mean']
fossil_brown.append(df2018_9.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_8.describe(include='all').loc['mean']
fossil_brown.append(df2018_8.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_7.describe(include='all').loc['mean']
fossil_brown.append(df2018_7.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_6.describe(include='all').loc['mean']
fossil_brown.append(df2018_6.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_5.describe(include='all').loc['mean']
fossil_brown.append(df2018_5.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_4.describe(include='all').loc['mean']
fossil_brown.append(df2018_4.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_3.describe(include='all').loc['mean']
fossil_brown.append(df2018_3.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_2.describe(include='all').loc['mean']
fossil_brown.append(df2018_2.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2018_1.describe(include='all').loc['mean']
fossil_brown.append(df2018_1.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_12.describe(include='all').loc['mean']
fossil_brown.append(df2017_12.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_11.describe(include='all').loc['mean']
fossil_brown.append(df2017_11.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_10.describe(include='all').loc['mean']
fossil_brown.append(df2017_10.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_9.describe(include='all').loc['mean']
fossil_brown.append(df2017_9.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_8.describe(include='all').loc['mean']
fossil_brown.append(df2017_8.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_7.describe(include='all').loc['mean']
fossil_brown.append(df2017_7.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_6.describe(include='all').loc['mean']
fossil_brown.append(df2017_6.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_5.describe(include='all').loc['mean']
fossil_brown.append(df2017_5.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_4.describe(include='all').loc['mean']
fossil_brown.append(df2017_4.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_3.describe(include='all').loc['mean']
fossil_brown.append(df2017_3.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_2.describe(include='all').loc['mean']
fossil_brown.append(df2017_2.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2017_1.describe(include='all').loc['mean']
fossil_brown.append(df2017_1.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_12.describe(include='all').loc['mean']
fossil_brown.append(df2016_12.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_11.describe(include='all').loc['mean']
fossil_brown.append(df2016_11.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_10.describe(include='all').loc['mean']
fossil_brown.append(df2016_10.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_9.describe(include='all').loc['mean']
fossil_brown.append(df2016_9.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_8.describe(include='all').loc['mean']
fossil_brown.append(df2016_8.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_7.describe(include='all').loc['mean']
fossil_brown.append(df2016_7.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_6.describe(include='all').loc['mean']
fossil_brown.append(df2016_6.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_5.describe(include='all').loc['mean']
fossil_brown.append(df2016_5.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_4.describe(include='all').loc['mean']
fossil_brown.append(df2016_4.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_3.describe(include='all').loc['mean']
fossil_brown.append(df2016_3.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
df2016_2.describe(include='all').loc['mean']
fossil_brown.append(df2016_2.describe(include='all').loc['mean'][['generation fossil brown coal/lignite']])
```

```

df2016_1.describe(include='all').loc['mean']
fossil_brown.append(df2016_1.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_12.describe(include='all').loc['mean']
fossil_brown.append(df2015_12.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_11.describe(include='all').loc['mean']
fossil_brown.append(df2015_11.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_10.describe(include='all').loc['mean']
fossil_brown.append(df2015_10.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_9.describe(include='all').loc['mean']
fossil_brown.append(df2015_9.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_8.describe(include='all').loc['mean']
fossil_brown.append(df2015_8.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_7.describe(include='all').loc['mean']
fossil_brown.append(df2015_7.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_6.describe(include='all').loc['mean']
fossil_brown.append(df2015_6.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_5.describe(include='all').loc['mean']
fossil_brown.append(df2015_5.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_4.describe(include='all').loc['mean']
fossil_brown.append(df2015_4.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_3.describe(include='all').loc['mean']
fossil_brown.append(df2015_3.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_2.describe(include='all').loc['mean']
fossil_brown.append(df2015_2.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])
df2015_1.describe(include='all').loc['mean']
fossil_brown.append(df2015_1.describe(include='all').loc['mean']['generation fossil brown coal/lignite'])

fossil_brown.reverse() # Average monthly values
print(fossil_brown)

```

[572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.2809139784946, 665.162726008
345, 684.2204301075269, 585.7674731182796, 548.0833333333334, 528.0188172043011, 695.284722222222, 493.48048452
22073, 417.9274193548387, 191.66522988505747, 173.20323014804845, 143.57083333333333, 179.002688172043, 175.6, 3
98.8156123822342, 464.3736559139785, 473.7208333333336, 613.7691275167786, 649.5458333333333, 670.7956989247311
, 688.6451612903226, 603.4151785714286, 335.66756393001344, 420.1152777777778, 500.77016129032256, 478.811111111
11113, 650.6774193548387, 511.8467741935484, 642.6305555555556, 561.3221476510067, 667.647222222222, 476.543010
7526882, 379.56451612903226, 406.95089285714283, 124.41588156123822, 133.97916666666666, 307.7043010752688, 333.
99861111111113, 506.36877523553164, 300.14247311827955, 558.161111111112, 405.83758389261743, 375.7361111111111
, 406.22849462365593]

In []:

In []: hydro_water= []

```

df2018_12.describe(include='all').loc['mean']
hydro_water.append(df2018_12.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_11.describe(include='all').loc['mean']
hydro_water.append(df2018_11.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_10.describe(include='all').loc['mean']
hydro_water.append(df2018_10.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_9.describe(include='all').loc['mean']
hydro_water.append(df2018_9.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_8.describe(include='all').loc['mean']
hydro_water.append(df2018_8.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_7.describe(include='all').loc['mean']
hydro_water.append(df2018_7.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_6.describe(include='all').loc['mean']
hydro_water.append(df2018_6.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_5.describe(include='all').loc['mean']
hydro_water.append(df2018_5.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_4.describe(include='all').loc['mean']
hydro_water.append(df2018_4.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_3.describe(include='all').loc['mean']
hydro_water.append(df2018_3.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_2.describe(include='all').loc['mean']
hydro_water.append(df2018_2.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2018_1.describe(include='all').loc['mean']
hydro_water.append(df2018_1.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_12.describe(include='all').loc['mean']
hydro_water.append(df2017_12.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_11.describe(include='all').loc['mean']
hydro_water.append(df2017_11.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_10.describe(include='all').loc['mean']
hydro_water.append(df2017_10.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_9.describe(include='all').loc['mean']
hydro_water.append(df2017_9.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_8.describe(include='all').loc['mean']
hydro_water.append(df2017_8.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_7.describe(include='all').loc['mean']
hydro_water.append(df2017_7.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_6.describe(include='all').loc['mean']
hydro_water.append(df2017_6.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_5.describe(include='all').loc['mean']
hydro_water.append(df2017_5.describe(include='all').loc['mean']['generation hydro water reservoir'])

```

```

df2017_4.describe(include='all').loc['mean']
hydro_water.append(df2017_4.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_3.describe(include='all').loc['mean']
hydro_water.append(df2017_3.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_2.describe(include='all').loc['mean']
hydro_water.append(df2017_2.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2017_1.describe(include='all').loc['mean']
hydro_water.append(df2017_1.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_12.describe(include='all').loc['mean']
hydro_water.append(df2016_12.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_11.describe(include='all').loc['mean']
hydro_water.append(df2016_11.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_10.describe(include='all').loc['mean']
hydro_water.append(df2016_10.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_9.describe(include='all').loc['mean']
hydro_water.append(df2016_9.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_8.describe(include='all').loc['mean']
hydro_water.append(df2016_8.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_7.describe(include='all').loc['mean']
hydro_water.append(df2016_7.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_6.describe(include='all').loc['mean']
hydro_water.append(df2016_6.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_5.describe(include='all').loc['mean']
hydro_water.append(df2016_5.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_4.describe(include='all').loc['mean']
hydro_water.append(df2016_4.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_3.describe(include='all').loc['mean']
hydro_water.append(df2016_3.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_2.describe(include='all').loc['mean']
hydro_water.append(df2016_2.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2016_1.describe(include='all').loc['mean']
hydro_water.append(df2016_1.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_12.describe(include='all').loc['mean']
hydro_water.append(df2015_12.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_11.describe(include='all').loc['mean']
hydro_water.append(df2015_11.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_10.describe(include='all').loc['mean']
hydro_water.append(df2015_10.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_9.describe(include='all').loc['mean']
hydro_water.append(df2015_9.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_8.describe(include='all').loc['mean']
hydro_water.append(df2015_8.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_7.describe(include='all').loc['mean']
hydro_water.append(df2015_7.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_6.describe(include='all').loc['mean']
hydro_water.append(df2015_6.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_5.describe(include='all').loc['mean']
hydro_water.append(df2015_5.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_4.describe(include='all').loc['mean']
hydro_water.append(df2015_4.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_3.describe(include='all').loc['mean']
hydro_water.append(df2015_3.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_2.describe(include='all').loc['mean']
hydro_water.append(df2015_2.describe(include='all').loc['mean']['generation hydro water reservoir'])
df2015_1.describe(include='all').loc['mean']
hydro_water.append(df2015_1.describe(include='all').loc['mean']['generation hydro water reservoir'])

hydro_water.reverse() # Average monthly values
print(hydro_water)

```

[2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.184139784946, 2531.1682892906815, 2412.5255376344085, 1963.0631720430108, 2261.1569444444444, 2276.9193548387098, 2404.390277777776, 1943.42799461642, 4075.5846774193546, 4881.568965517241, 3901.5450874831763, 5119.295833333334, 4581.743279569892, 2983.793055555554, 2556.6971736204578, 2460.0201612903224, 2303.461111111111, 2500.489932885906, 2501.297222222224, 2805.2970430107525, 2156.951612903226, 1874.8497023809523, 2348.4589502018844, 1611.4152777777779, 1639.616935483871, 1448.644444444444, 1290.983870967742, 1260.6908602150538, 1570.509722222223, 1229.2845637583894, 1371.33888888889, 1452.9569892473119, 2271.896505376344, 2773.2038690476193, 4378.419919246298, 4159.898611111111, 2931.19623655914, 3362.5291666666667, 2789.228802153432, 2296.0295698924733, 2267.641666666667, 2172.1919463087247, 2665.9, 2763.0241935483873]

Below are the plots and regressions for the mean price of energy per EUR/MWH for each month and the mean megawatts (MW) output per resource, given the month and its mean price of energy per EUR/MWH. The months, prices, megawatts, and MW-to-price ratio were all accounted for. The significance behind the MW-to-price ratio is that it depicts the MW of the resource for one euro. Descriptions were given for each analysis.

One discretion that the viewer should account for is that the dates from January 2015 to December 2018 for some of the plots were depicted as values from 0 to 48. Ex: January 2015 would be considered month 0.

In []: #Outputs in MWH Histogram

```

Price_Actual_Dict = {key: i for i, key in enumerate(Price_Actual)}
def Hist_Price_Actual(Price_Actual_Dict):

```

```

    for k, v in Price_Actual.items(): print(f"v:{k}")
print(Price_Actual_Dict)
plt.suptitle("Average monthly price of energy per EUR/MWH from January 2015 to December 2018 in Spain")
plt.ylabel('Average amount')
plt.xlabel('Months')
plt.bar(list(Price_Actual_Dict.values()), Price_Actual_Dict.keys(), color='g')
print(dicDates)
import matplotlib.pyplot as plt
plt.show()

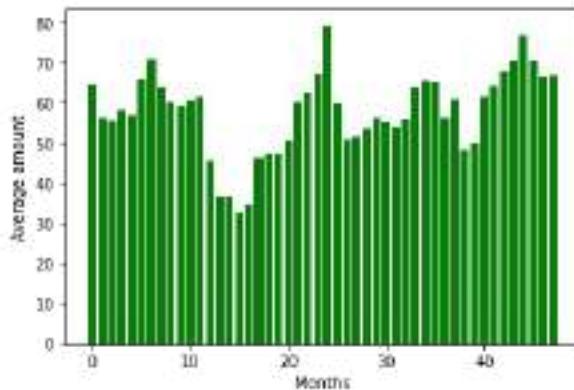
```

```

{64.9490188172043: 0, 56.38385416666667: 1, 55.52246298788695: 2, 58.35408333333335: 3, 57.29405913978494: 4, 6
5.97490277777777: 5, 71.0720430107527: 6, 63.99806451612903: 7, 60.25479166666667: 8, 59.40676510067113: 9, 60.7
2679166666667: 10, 61.901760752688176: 11, 45.57872311827957: 12, 36.75208333333333: 13, 36.818008075370116: 14,
32.61866666666666: 15, 34.69137096774194: 16, 46.26631944444444: 17, 47.502016129032256: 18, 47.60233870967742:
19, 50.40559722222222: 20, 60.18242953020134: 21, 62.58105555555556: 22, 67.59513440860215: 23, 79.492083333333
4: 24, 59.83779761904762: 25, 50.95989232839838: 26, 51.71791666666666: 27, 53.772620967741936: 28, 56.258222222
22222: 29, 55.25258064516129: 30, 54.08432795698924: 31, 55.81655555555556: 32, 63.92528859060402: 33, 65.430652
7777778: 34, 65.15127688172043: 35, 56.511975806451616: 36, 60.877098214285716: 37, 48.279717362045766: 38, 50.
40073611111111: 39, 61.63376344086022: 40, 64.34813888888888: 41, 67.78344086021505: 42, 70.36391129032258: 43,
76.91404166666666: 44, 70.36221476510067: 45, 66.6235138888889: 46, 67.04260752688172: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15
-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18
, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03
': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '
17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08':
44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}

```

Average monthly price of energy per EUR/MWH from January 2015 to December 2018 in Spain



The green bars represent the observation value for each respective month. This histogram has a significant trench between month 10 and month 20, meaning that there was a price drop. In addition, the histogram has a multimodal distribution, which could indicate that there wasn't any external factors that led to a singular price increase. However, the price fluctuations within the histogram appear to be seasonal.

But within this histogram, there seems to be higher inversions as the months progress, indicating that the prices seem to be increasing overtime.

Below is the residual plot for the average monthly prices of energy per EUR/MWH.

```
In [ ]: Months = [i for i in range(48)]
print(Months)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 3
0, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]
```

```
In [ ]: Months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]
```

```
In [ ]: Months1 = Months
Months1 = sm.add_constant(Months1)
Price_Actual1 = Price_Actual
Price_Actual1 = sm.add_constant(Price_Actual1)
```

As one can observe this residual plot, one may notice the rebound in the observations, which formed a nonlinear pattern. However, the residuals are spread out, indicating a lack of bias, homoscedasticity, and constant variance. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy of the observed regression value versus the actual value of the observation itself.

```
In [ ]: import numpy as np
from sklearn.linear_model import LinearRegression
```

```
y = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
x = np.array([64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65
```

```
In [ ]:
```

```
In [ ]: modelprice = stats.linregress ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22], [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.974902777777])
```

```
In [ ]:
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly prices of energy per EUR/MWH.

```
In [ ]: from sklearn.preprocessing import PolynomialFeatures
```

```
In [ ]: #Quadratic OLS regression
polynomial_features = PolynomialFeatures(degree=2)

modelpricequad = np.poly1d(np.polyfit(Months, Price_Actual, 2))
print(modelpricequad)

Price_Actual1 = sm.add_constant(Price_Actual)
xp = polynomial_features.fit_transform(Price_Actual1)

poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(Months1)

Quad = poly.fit(X_poly, Price_Actual)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
Price_Quad = sm.OLS(Months, xp).fit()

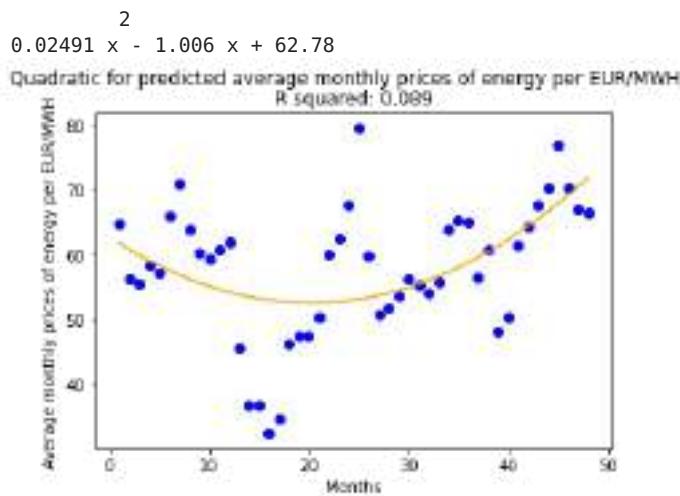
ypred = Price_Quad.predict(xp)

Price_Quad.summary()

# #OLS Quadratic Scatterplot
plt.scatter(Months, Price_Actual, color = 'blue')

plt.plot(Months, lin2.predict(poly.fit_transform(Months1)), color = 'orange')
plt.title("R squared: 0.089")
plt.suptitle('Quadratic for predicted average monthly prices of energy per EUR/MWH')
plt.xlabel('Months')
plt.ylabel('Average monthly prices of energy per EUR/MWH')

plt.show()
```



The blue dots represent the observations and the orange line is the model of best fit. This is a very weak and positive correlation between the months themselves and the average monthly prices of energy per EUR/MWH. The blue dots represent the observations and the red line represents the model of best fit.

```
In [ ]:
```

```
#Quadratic OLS regression
polynomial_features = PolynomialFeatures(degree=2)

modelpricequad = np.poly1d(np.polyfit(Months, Price_Actual, 2))
print(modelpricequad)
```

```

Price_Actual1 = sm.add_constant(Price_Actual1)
xp = polynomial_features.fit_transform(Price_Actual1)

poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(Months1)

Quad = poly.fit(X_poly, Price_Actual)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
Price_Quad = sm.OLS(Months, xp).fit()

```

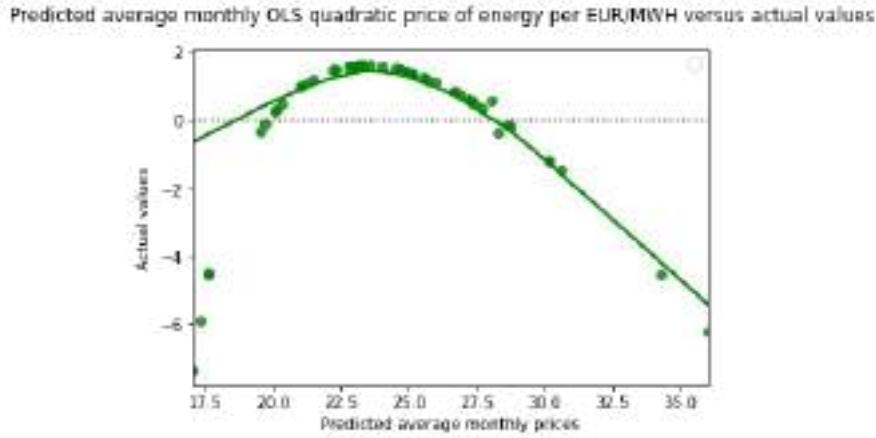
$$0.02491 x^2 - 1.006 x + 62.78$$

```

In [ ]: plt.suptitle("Predicted average monthly OLS quadratic price of energy per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend("#")
sns.residplot(x = ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b7ee590>



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []: Price_Quad.summary()

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.089			
Model:	OLS	Adj. R-squared:	0.048			
Method:	Least Squares	F-statistic:	2.196			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.123			
Time:	05:20:48	Log-Likelihood:	-192.04			
No. Observations:	48	AIC:	390.1			
Df Residuals:	45	BIC:	395.7			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	6.7433	13.956	0.483	0.631	-21.366	34.852
x1	6.7433	13.956	0.483	0.631	-21.366	34.852
x2	-0.1512	0.763	-0.198	0.844	-1.688	1.385
x3	6.7433	13.956	0.483	0.631	-21.366	34.852
x4	-0.1512	0.763	-0.198	0.844	-1.688	1.385
x5	0.0063	0.014	0.461	0.647	-0.021	0.034
Omnibus:	7.220	Durbin-Watson:	0.060			
Prob(Omnibus):	0.027	Jarque-Bera (JB):	3.163			
Skew:	-0.348	Prob(JB):	0.206			
Kurtosis:	1.953	Cond. No.	1.10e+21			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 5.21e-34. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
influencePriceQuad = Price_Quad.get_influence() #Quadratic OLS residuals

standardized_residualsPriceQuad = influencePriceQuad.resid_studentized_internal

print(standardized_residualsPriceQuad)
```

```
[-1.94818180e+00 -1.58080642e+00 -1.48116586e+00 -1.49198489e+00
 -1.38475535e+00 -1.61850705e+00 -1.79586250e+00 -1.39016989e+00
 -1.18190814e+00 -1.07949414e+00 -1.04922965e+00 -1.01576654e+00
 -4.92123149e-01 -2.90959482e-01 -2.11519954e-01 -9.48333730e-02
 -2.73241639e-02 -1.30367715e-01 -8.19309856e-02 -9.24182566e-03
 -5.97110015e-04 -2.13226132e-01 -2.22885756e-01 -3.43979459e-01
 -9.95883892e-01 9.57368580e-02 4.33609888e-01 4.88555949e-01
 5.07031872e-01 5.08389692e-01 6.13222108e-01 7.21877887e-01
 7.45315671e-01 5.45133056e-01 5.62864078e-01 6.47896743e-01
 1.02189398e+00 9.51980864e-01 1.39763134e+00 1.41915961e+00
 1.14827745e+00 1.12416870e+00 1.06989706e+00 1.04768394e+00
 8.90420462e-01 1.19926009e+00 1.41205911e+00 1.47124764e+00]
```

In []:

```
# OLS Predicted Quadratic values
print(ypred)

[27.18807769 23.22554817 22.87823628 24.05511741 23.60270863 27.72472243
 30.58783893 26.70248166 24.90180391 24.51841487 25.11912136 25.67229968
 19.54636257 17.6334062 17.64405232 17.07534443 17.32825464 19.7366291
 20.09354637 20.12336851 21.00799158 24.86873542 26.00005506 28.59930479
 36.03506727 24.71214951 21.19464431 21.45617264 22.20151115 23.17431072
 22.77134503 22.31923277 22.99576214 26.66578905 27.43836997 27.29282952
 23.278006 25.18891499 20.32804924 21.00637179 25.54459336 26.87991763
 28.70309766 30.1704781 34.27227994 30.16948579 28.0708618 28.29733761]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
LogMonthsvalues = ((np.log(Months)))
Log = np.polyfit(np.log(Months), Price_Actual1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(Price_Actual1), Months)
Price_Log = sm.OLS(Months, Price_Actual1).fit()

Logpred = Price_Log.predict(Price_Actual1)

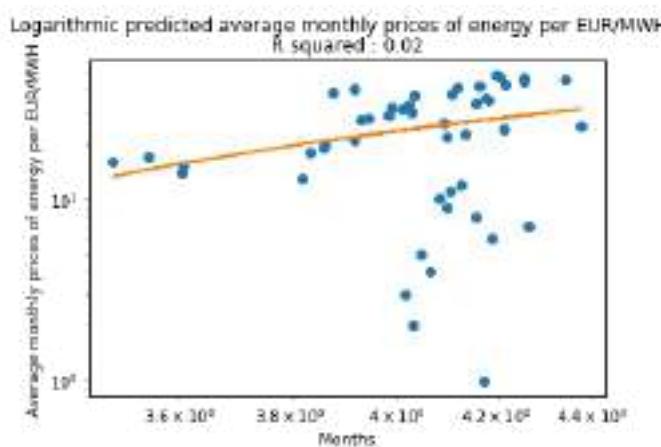
Price_Log.summary() #OLS Logarithmic summary table
#Log
Log = np.polyfit(np.log(Price_Actual), Months, 1)
print(Log)

y = 20.14460424 * Logpricevalues -56.89509997

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic predicted average monthly prices of energy per EUR/MWH")
plt.title("R squared : 0.02")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Months")
plt.yscale("log")
plt.xscale("log")
plt.plot(Logpricevalues, Months, "o")
plt.plot(Logpricevalues, y)
```

[20.14460424 -56.89509997]

Out[]: [<matplotlib.lines.Line2D at 0x7ff60b635150>]



This is a very weak and positive correlation between the months themselves and the average monthly prices of energy per EUR/MWH.

```
In [ ]: influencePriceLog = Price_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsPriceLog = influencePriceLog.resid_studentized_internal

print(standardized_residualsPriceLog)

print(Logpred)# OLS logarithmic predicted values
```

[-1.97297595 -1.63629344 -1.53677617 -1.54471644 -1.43886443 -1.63084231
-1.72635178 -1.41780683 -1.2281771 -1.1281171 -1.09299655 -1.05380313
-0.50447184 -0.16995032 -0.09367512 0.1168339 0.12988847 -0.14571288
-0.10662895 -0.03413869 -0.04193175 -0.25514659 -0.25158444 -0.32722807
-0.63066263 0.05372586 0.39172857 0.44384706 0.45703911 0.4578122
0.56234125 0.67197706 0.69491683 0.53243968 0.56404848 0.64712841
0.9728795 0.91962957 1.37739607 1.38449082 1.12204091 1.11992381
1.09966813 1.10496964 1.0069336 1.2567342 1.43255598 1.49651333]
[27.29734838 23.91758071 23.57768041 24.69502249 24.27674267 27.70215664
29.71346062 26.92210695 25.44503172 25.11040502 25.63128037 26.09491682
19.65393442 16.17099008 16.19700362 14.53996621 15.35784412 19.92525621
20.41285544 20.45244219 21.55859284 25.41647801 26.36296287 28.34149128
33.03596301 25.28048812 21.77731469 22.076427 22.88720221 23.86800702
23.4711863 23.0102001 23.69372775 26.89338996 27.48739853 27.37715831
23.96813682 25.69059052 20.71973214 21.55667467 25.98916653 27.0602444
28.41579599 29.43403567 32.0186817 29.43336623 27.95809508 28.12346716]

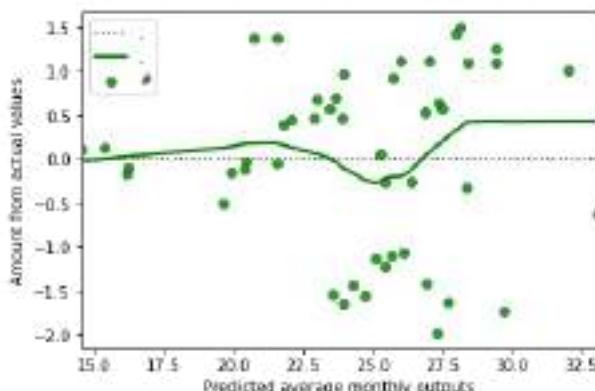
In []: # OLS Logarithmic average monthly predictions versus residuals

```
sns.residplot(x = Logpred, y =standardized_residualsPriceLog, lowess = True, color="g")
plt.suptitle("Price residuals from quadratic model versus predicted average monthly outputs")
plt.xlabel("Predicted average monthly outputs")
```

```
plt.ylabel("Amount from actual values")
plt.rcParams["figure.figsize"] = [20, 10]
plt.legend(..#)
```

Out[]: <matplotlib.legend.Legend at 0x7ff60b4cc450>

Price residuals from quadratic model versus predicted average monthly outputs

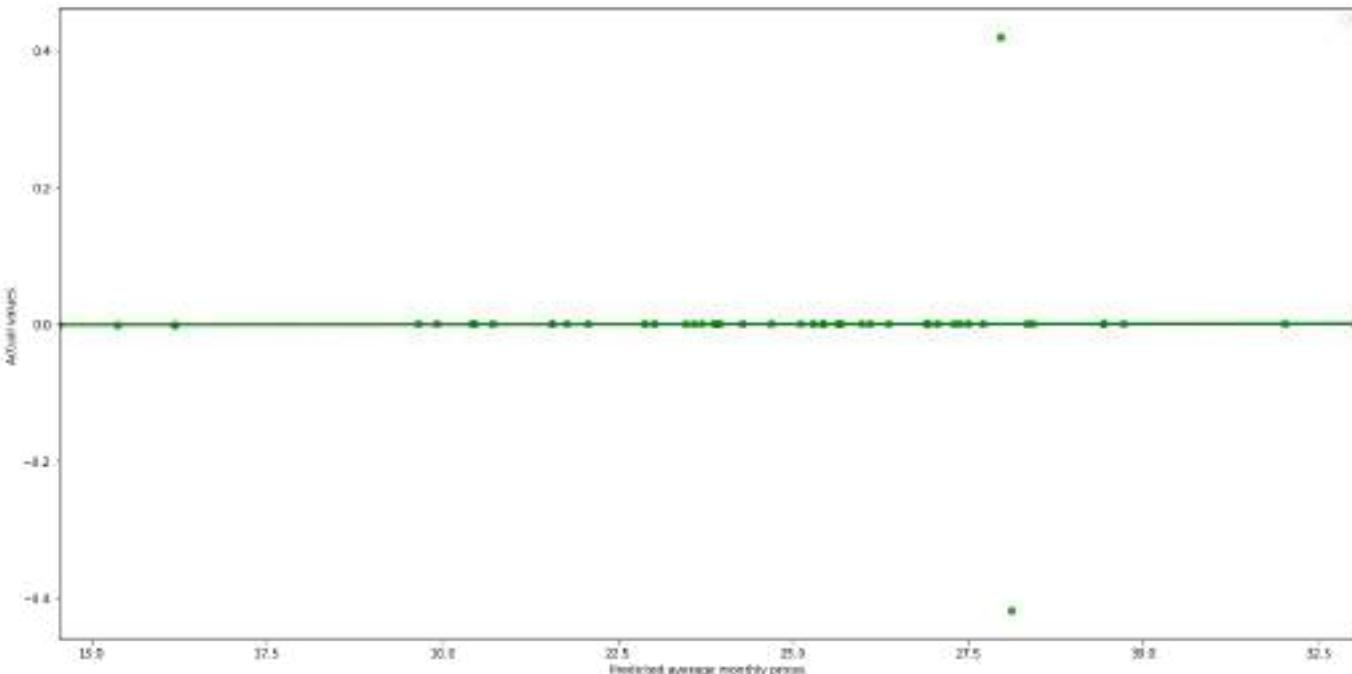


As one can observe this residual plot, one may notice a slightly curved rebound in the observations, which formed a nonlinear pattern, indicating that a nonlinear model would fit the observation values better than a linear model. However, the residuals are spread out; indicating homoscedasticity and constant variance. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly OLS logarithmic price of energy per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Logpred, y = Price_Actual, lowess = True, color="g")
# OLS predicted logarithmic average monthly values versus actual values
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b45f510>

Predicted average monthly OLS logarithmic price of energy per EUR/MWH versus actual values



The predictions seem to be nearly the same as the actual values. This indicates a lack of bias, constant variance, and homoscedasticity. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Linear OLS regression
Months1 = sm.add_constant(Months1)
modelpricereg = sm.OLS(Price_Actual, Months1).fit()
predictions = modelpricereg.predict(Months1)
```

```
modelpricereg.summary()  
#OLS Linear Summary Table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.085		
Model:	OLS	Adj. R-squared:	0.065		
Method:	Least Squares	F-statistic:	4.250		
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.0449		
Time:	05:20:49	Log-Likelihood:	-177.52		
No. Observations:	48	AIC:	359.0		
Df Residuals:	46	BIC:	362.8		
Df Model:	1				
Covariance Type:	nonrobust				
coef	std err	t	P> t	[0.025	0.975]
const	52.6072	2.927	17.973	0.000	46.716 58.499
x1	0.2144	0.104	2.062	0.045	0.005 0.424
Omnibus:	1.718	Durbin-Watson:	0.455		
Prob(Omnibus):	0.424	Jarque-Bera (JB):	1.379		
Skew:	-0.414	Prob(JB):	0.502		
Kurtosis:	2.927	Cond. No.	57.2		

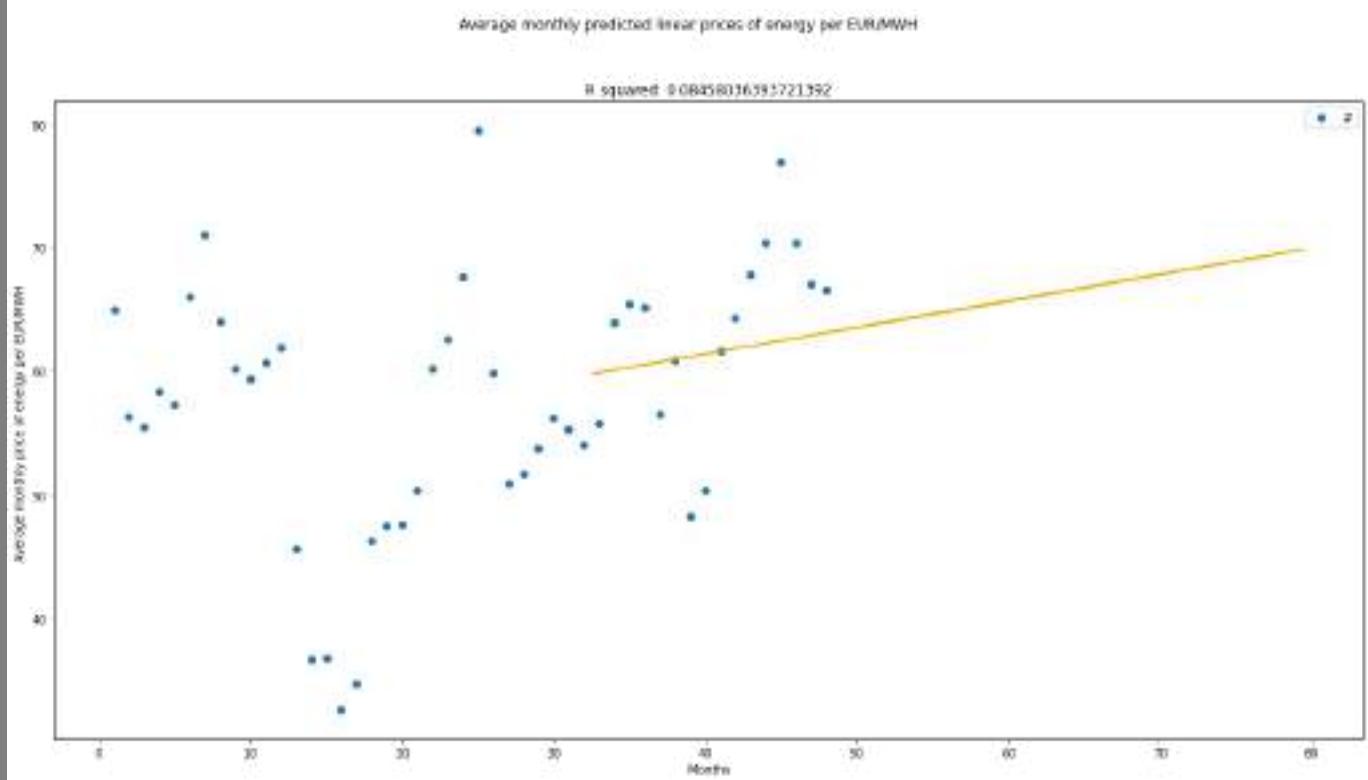
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [ ]: modelprice = stats.linregress ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.974902777777])
```

```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(Months,Price_Actual)  
print("slope: %f      intercept: %f" % (slope, intercept))  
  
#OLS Linear Scatterplot  
plt.suptitle("Average monthly predicted linear prices of energy per EUR/MWH")  
plt.plot(Months, Price_Actual, "o")  
f = lambda x: 0.214393*x + 52.821613  
plt.rcParams["figure.figsize"] = [10, 10]  
plt.plot(x,f(x), c="orange", label="line of best fit")  
plt.title(f"R squared: {modelprice.rvalue**2}")  
plt.ylabel('Average monthly price of energy per EUR/MWH')  
plt.legend("#")  
plt.xlabel('Months')  
plt.show()
```

```
slope: 0.214393      intercept: 52.607220
```



As one can observe this scatterplot, one may notice the positive yet weak correlation between the mean price of energy per EUR/MWH and their respective months. The blue dots represent the observations and the orange line is the model of best fit.

```
In [ ]: #Linear OLS Predicted values
print(predictions)
```

```
[52.82161316 53.03600615 53.25039913 53.46479211 53.67918509 53.89357808
 54.10797106 54.32236404 54.53675703 54.75115001 54.96554299 55.17993597
 55.39432896 55.60872194 55.82311492 56.03750791 56.25190089 56.46629387
 56.68068685 56.89507984 57.10947282 57.3238658 57.53825879 57.75265177
 57.96704475 58.18143773 58.39583072 58.6102237 58.82461668 59.03900967
 59.25340265 59.46779563 59.68218861 59.8965816 60.11097458 60.32536756
 60.53976055 60.75415353 60.96854651 61.18293949 61.39733248 61.61172546
 61.82611844 62.04051143 62.25490441 62.46929739 62.68369037 62.89808336]
```

```
In [ ]: #Linear OLS regression residuals
influencePricereg = modelpricereg.get_influence()

standardized_residualsPricereg = influencePricereg.resid_studentized_internal

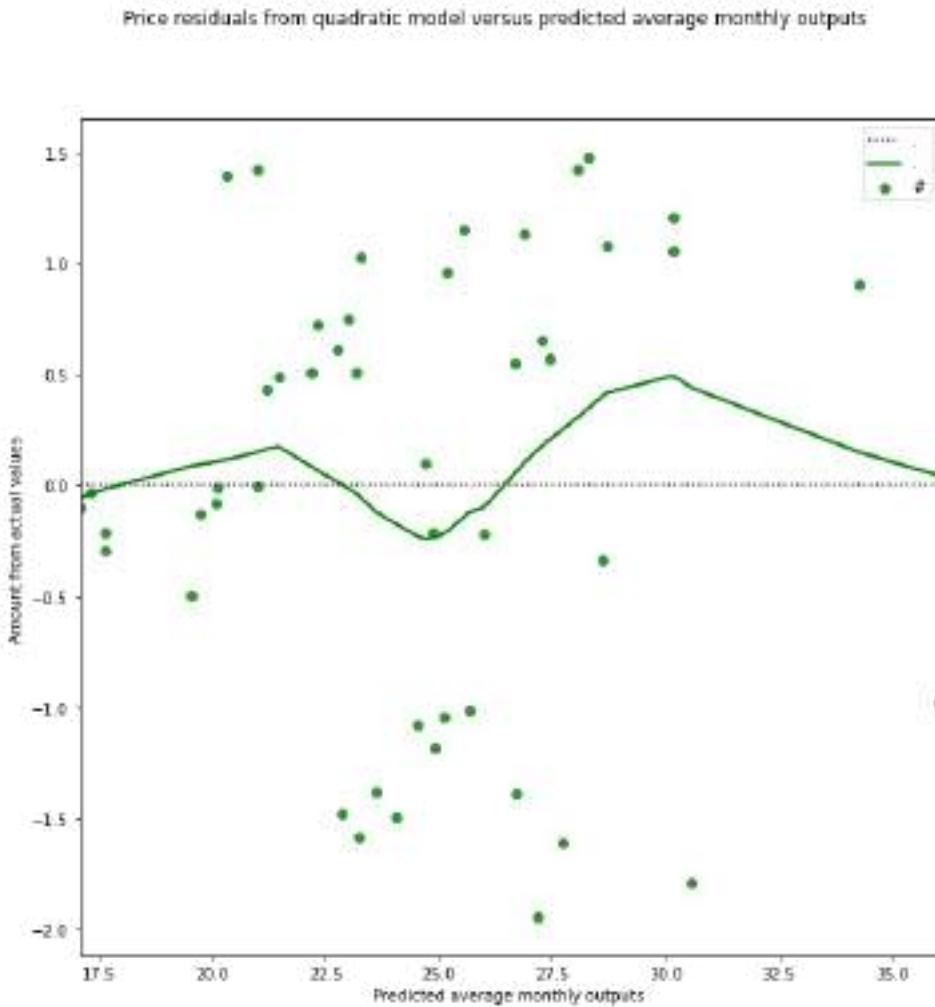
print(standardized_residualsPricereg)
```

```
[ 1.26728638  0.34889615  0.23617375  0.50698359  0.37396719  1.24710183
 1.74750527  0.99477572  0.58680885  0.47696469  0.58930039  0.68654563
-1.00118001 -1.92097848 -1.93393401 -2.38067149 -2.18980557 -1.03515724
-0.93088704 -0.94193214 -0.67921738  0.28952504  0.51063829  0.99654844
 2.17940375  0.16772454 -0.75313708 -0.69830871 -0.51208111 -0.2820233
-0.40602846 -0.54677448 -0.39296575  0.40995578  0.54193049  0.49223696
-0.41138502  0.01257563 -1.29996217 -1.10651523  0.02430788  0.28188379
 0.6149481   0.86107521  1.52004486  0.82044346  0.45426477  0.38929904]
```

```
In [ ]: modelpriceresidual = ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.974902777777]
```

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = ypred, y =standardized_residualsPriceQuad, lowess = True, color="g")
plt.suptitle("Price residuals from quadratic model versus predicted average monthly outputs")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.rcParams["figure.figsize"] = [20, 10]
plt.legend(..#)
```

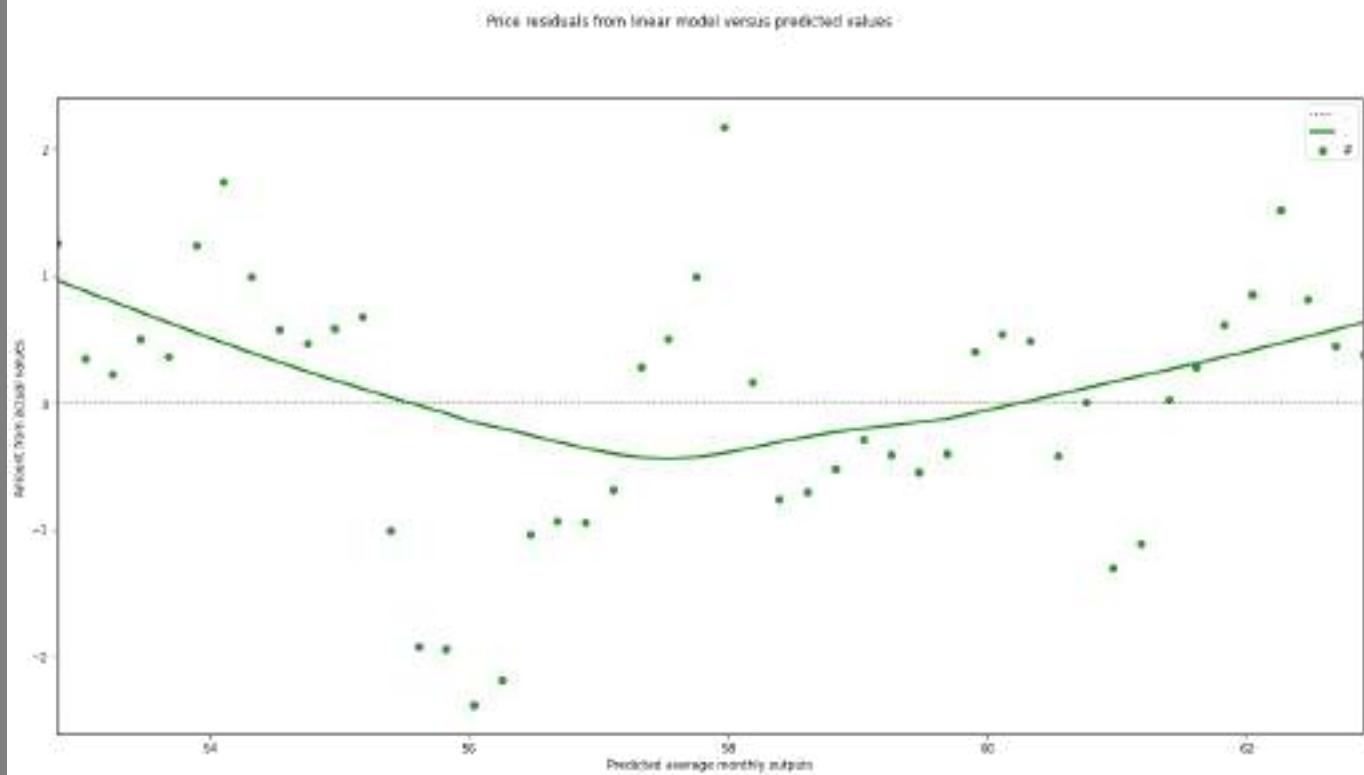
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff6380e5090>
```



As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. Given these circumstances, it would be reasonable to assume that the quadratic model of fit is suitable. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictions, y =standardized_residualsPricereg, lowess = True, color="g")
plt.suptitle("Price residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.rcParams["figure.figsize"] = [20, 10]
plt.legend(..#)
```

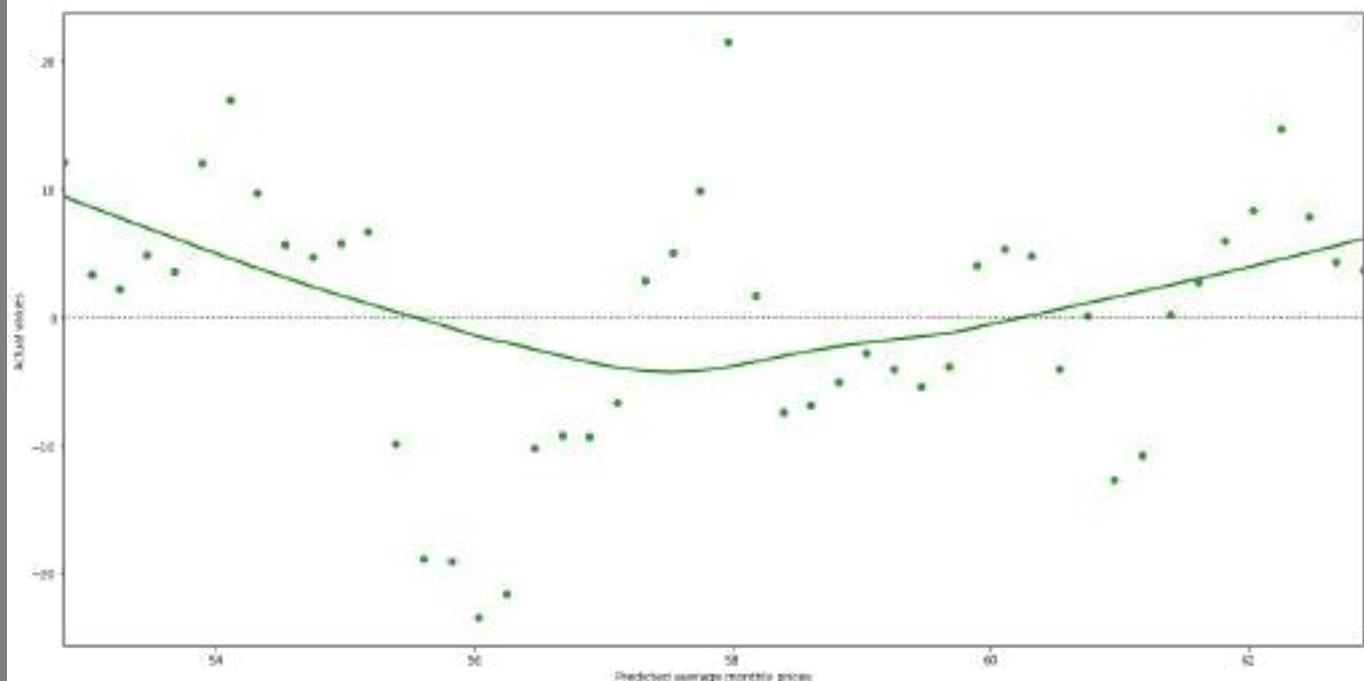
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff6103b3d10>
```



As one can observe, there is a subtle hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly OLS linear prices of energy per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictions, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b2fa550>
Predicted average monthly OLS linear prices of energy per EUR/MWh versus actual values
```



As one can observe this residual plot, one may notice the slight hump in the fitted model, which form a nonlinear pattern. The slight hump

is too subtle to suggest a different model. In addition, the observations are spread out without the distinct pattern, indicating constant variance, a lack of bias, and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(Months,Price_Actual)  
print("slope: %f      intercept: %f" % (slope, intercept))
```

```
slope: 0.214393      intercept: 52.607220
```

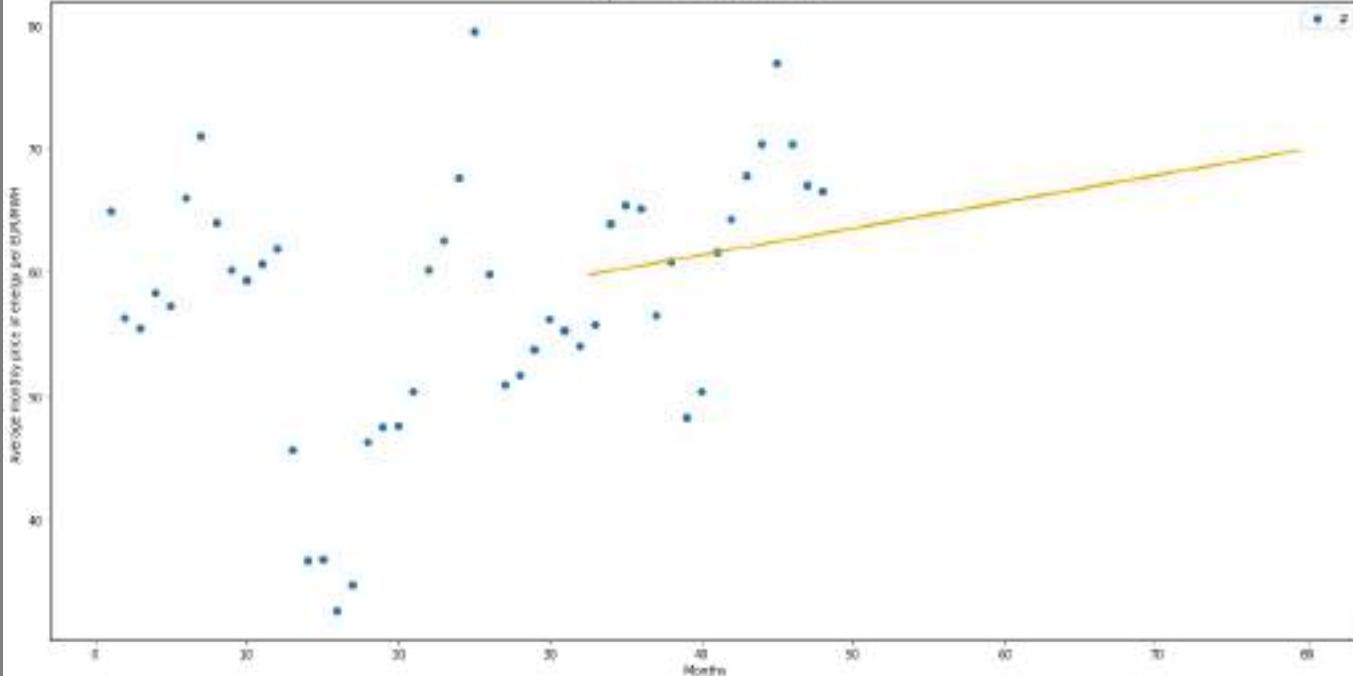
Below is a scatter graph depicting the relationship between the average monthly prices of energy per EUR/MWH.

```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(Months,Price_Actual)  
print("slope: %f      intercept: %f" % (slope, intercept))  
  
#OLS Linear Scatterplot  
plt.suptitle("Average monthly price of energy per EUR/MWH")  
plt.plot(Months, Price_Actual, "o")  
f = lambda x: 0.214393*x + 52.821613  
plt.rcParams["figure.figsize"] = [10, 10]  
plt.plot(x,f(x), c="orange", label="line of best fit")  
plt.title(f"R squared: {modelprice.rvalue**2}")  
plt.ylabel('Average monthly price of energy per EUR/MWH ')  
plt.legend("#")  
plt.xlabel('Months')  
plt.show()
```

```
slope: 0.214393      intercept: 52.607220
```

Average monthly price of energy per EUR/MWH

R squared: 0.08458036393721392



As one can observe this scatterplot, one may notice the positive yet weak correlation between the mean price of energy per EUR/MWH and their respective months. The blue dots represent the observations and the orange line is the model of best fit.

As one can observe this scatterplot, one may notice the positive yet weak correlation between the mean price of energy per EUR/MWH and their respective months. The blue dots represent the observations and the orange line is the model of best fit.

```
In [ ]:
```

Below are the prices of energy per EUR/MWH.

```
In [ ]: print(Price_Actual)
```

```
[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777  
7, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688  
176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944  
444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556, 67.59513  
440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25  
82222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.  
15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022  
, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.042607526881  
72, 66.6235138888889]
```

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, then seasonality and trends are indeed factors after all.

```
In [ ]: #Dataframes analyzed by resource  
dfPrice = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.974  
90277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.90  
1760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 4  
6.26631944444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556  
, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741  
936, 56.2582222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.430652777  
77778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.6337  
6344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04  
260752688172, 66.6235138888889], 'Months': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2  
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 4  
8]}  
Price Months  
0 64.949019 1  
1 56.383854 2  
2 55.522463 3  
3 58.354083 4  
4 57.294059 5  
5 65.974903 6  
6 71.072043 7  
7 63.998065 8  
8 60.254792 9  
9 59.406765 10  
10 60.726792 11  
11 61.901761 12  
12 45.578723 13  
13 36.752083 14  
14 36.818008 15  
15 32.618667 16  
16 34.691371 17  
17 46.266319 18  
18 47.502016 19  
19 47.602339 20  
20 50.405597 21  
21 60.182430 22  
22 62.581056 23  
23 67.595134 24  
24 79.492083 25  
25 59.837798 26  
26 50.959892 27  
27 51.717917 28  
28 53.772621 29  
29 56.258222 30  
30 55.252581 31  
31 54.084328 32  
32 55.816556 33  
33 63.925289 34  
34 65.430653 35  
35 65.151277 36  
36 56.511976 37  
37 60.877098 38  
38 48.279717 39  
39 50.400736 40  
40 61.633763 41  
41 64.348139 42  
42 67.783441 43  
43 70.363911 44  
44 76.914042 45  
45 70.362215 46  
46 67.042608 47  
47 66.623514 48
```

```
In [ ]: #ADF Tests
```

```

from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_Price['Price'])

test_result=adfuller(df_Price['Price'])

df_Price['First Difference Price'] = df_Price["Price"]- df_Price["Price"].shift(1) # Seasonality values
df_Price['Seasonal Difference Price']=df_Price["Price"]- df_Price["Price"].shift(12)
df_Price.head()

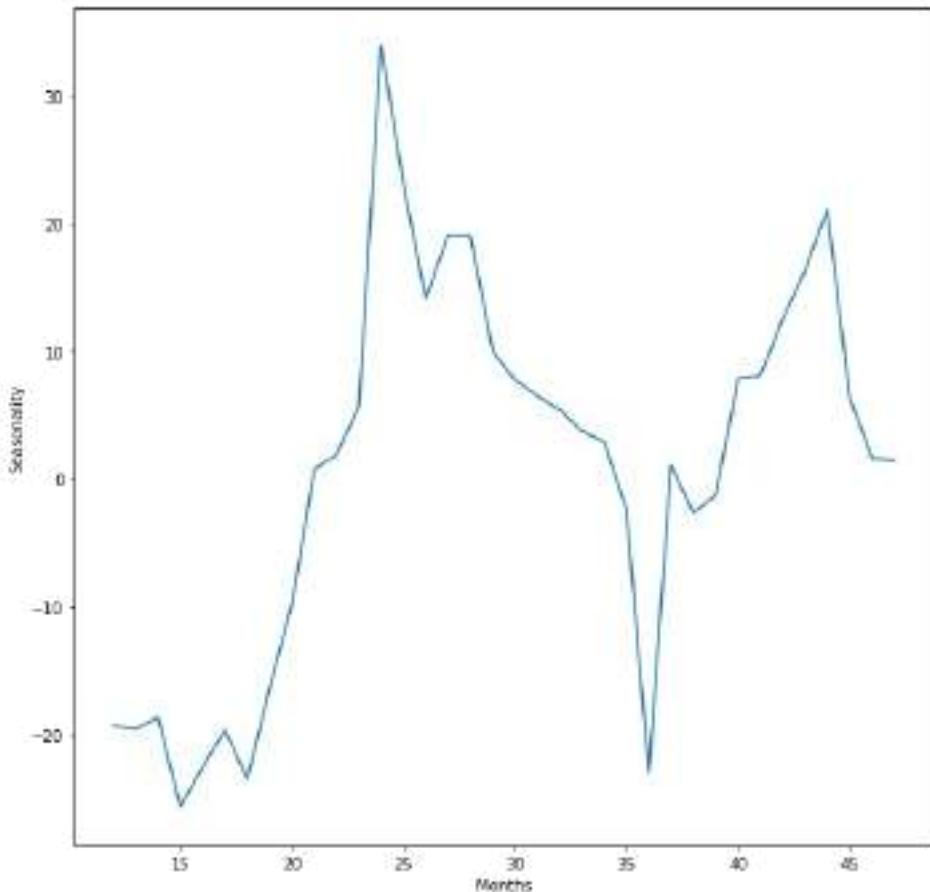
plt.suptitle("Seasonal Difference of average monthly prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Price['Seasonal Difference Price'].plot() # Seasonality Plot

```

ADF Test Statistic : -2.5921164784806066
 p-value : 0.09463760250608577
 #Lags Used : 1
 Number of Observations : 46
 weak evidence against null hypothesis,indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60baa5c10>

Seasonal Difference of average monthly prices of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there is volatility depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

In []:

In []: `import statsmodels.api as sm`

In []: `from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot`
`plot_acf(df_Price["Price"])`

```

plt.suptitle(" Autocorrelations of average monthly Price")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_Price["Price"])
plt.suptitle("Partial Autocorrelations of average monthly Price")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Price_Autocorrelations = sm.tsa.acf(df_Price["Price"], fft=False) #Autocorrelations
print(Price_Autocorrelations)

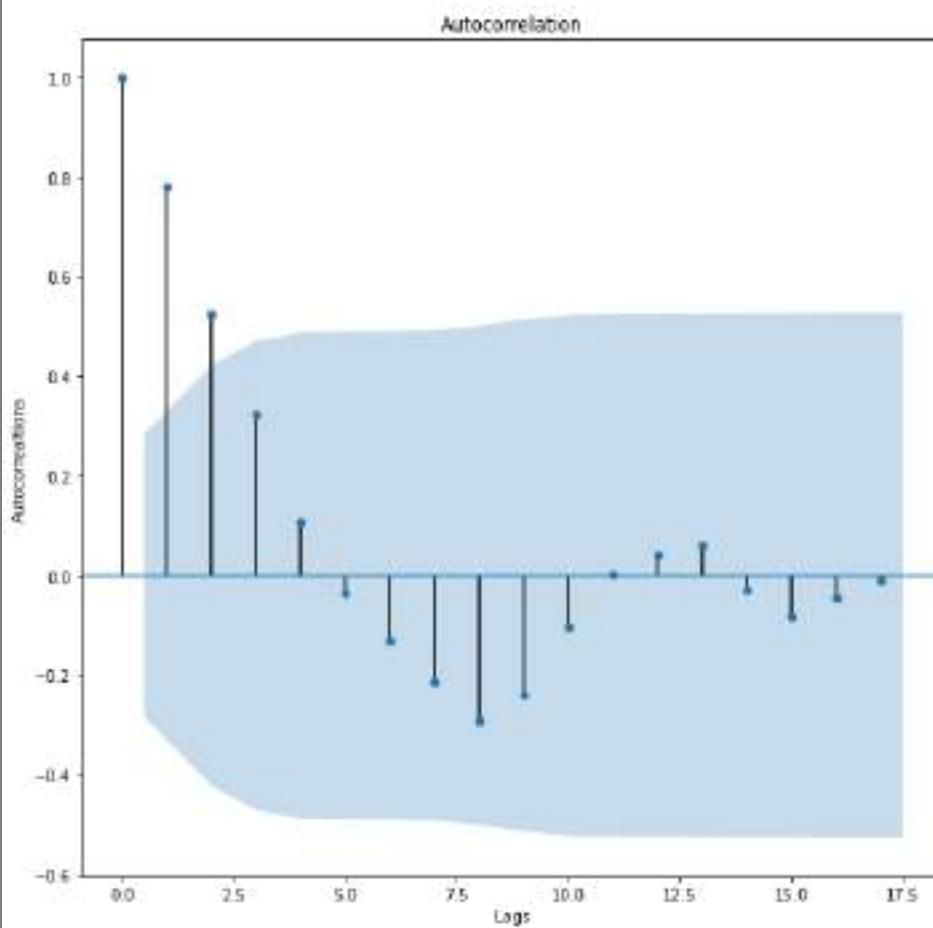
```

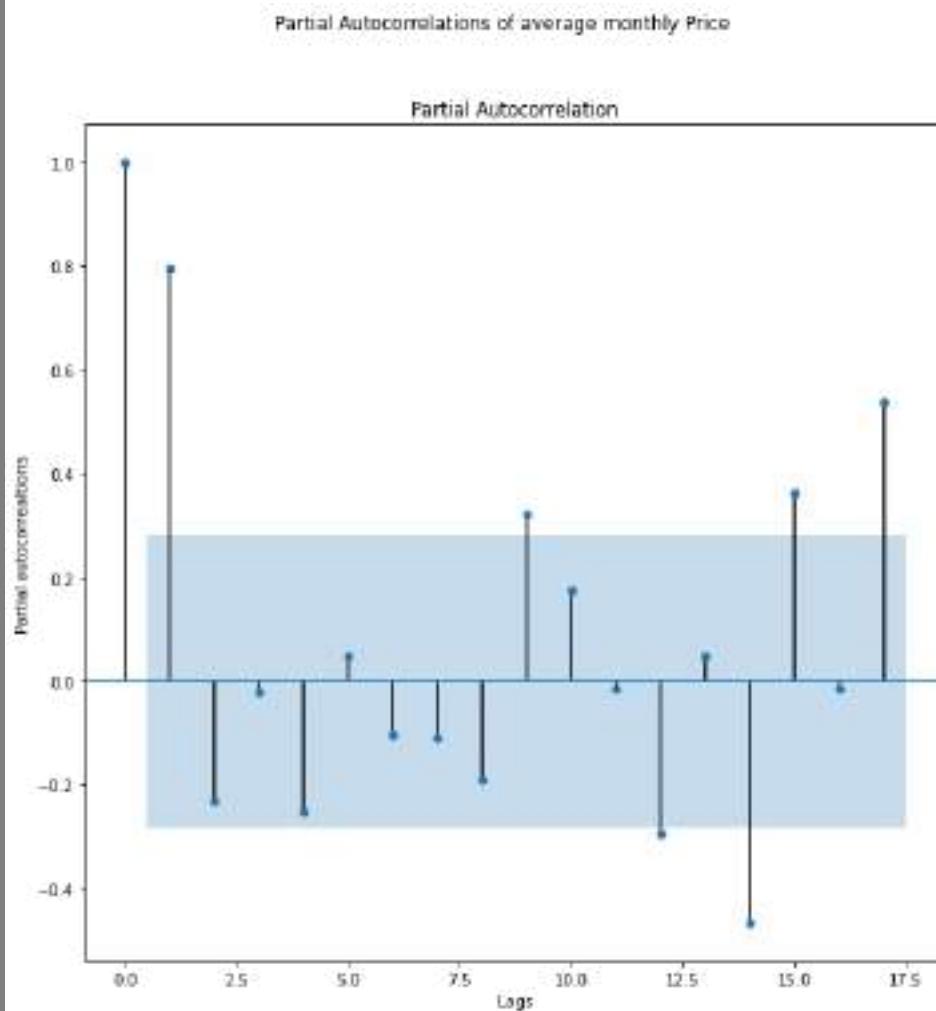
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,

```
[ 1.0000000e+00  7.79003572e-01  5.24798989e-01  3.23431778e-01
 1.06812553e-01 -3.63408274e-02 -1.30482038e-01 -2.15611711e-01
 -2.92669631e-01 -2.39305975e-01 -1.03497550e-01  5.93248145e-04
 4.12744055e-02  5.94044358e-02 -3.04091169e-02 -8.27466921e-02
 -4.41617039e-02 -9.53603418e-03  2.59026285e-02  1.17122133e-02
 -4.31383466e-03 -3.89309060e-03  3.13907136e-02  7.27751977e-02
 2.30875020e-02 -7.95848665e-02 -1.45607736e-01 -2.22179281e-01
 -2.72171549e-01 -2.77147782e-01 -2.72392641e-01 -2.54716900e-01
 -1.99535402e-01 -1.05617993e-01 -1.34550477e-02  4.95647338e-02
 7.78643480e-02  9.65062814e-02  7.93771347e-02  6.41965653e-02
 5.31257219e-02]
```

Autocorrelations of average monthly Price



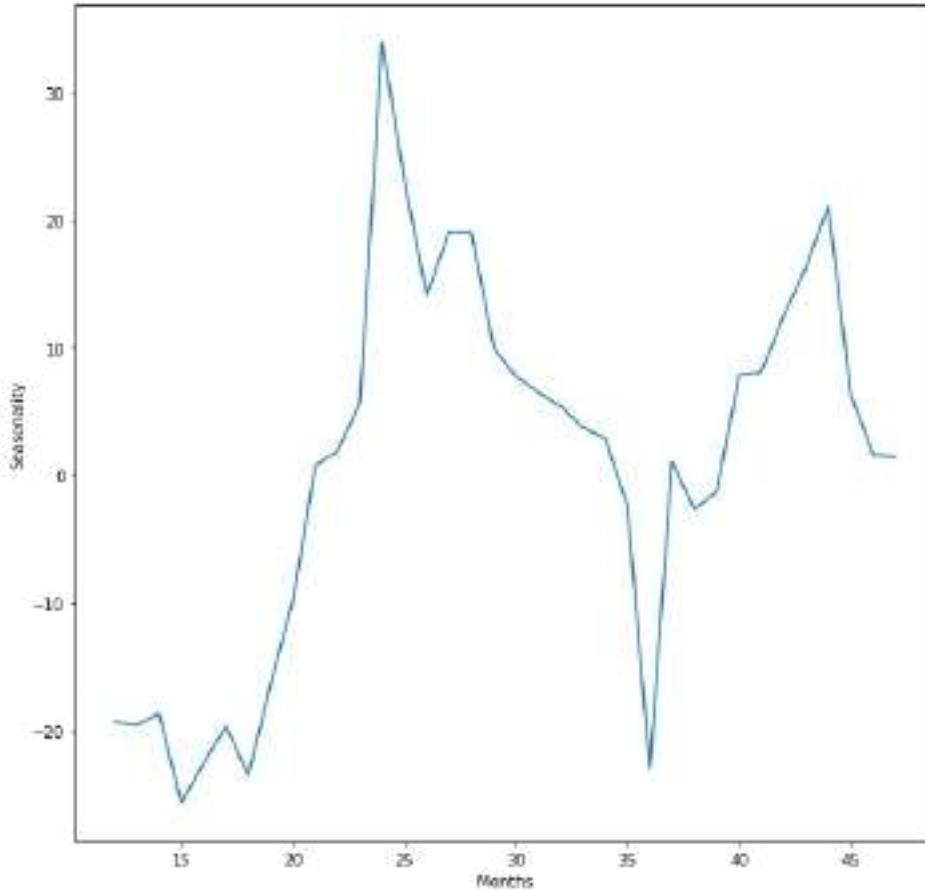


```
In [ ]: df_Price['First Difference Price'] = df_Price["Price"]- df_Price["Price"].shift(1) # Seasonality values
df_Price['Seasonal Difference Price']=df_Price["Price"]- df_Price["Price"].shift(12)
df_Price.head()

plt.suptitle("Seasonal Difference of average monthly prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Price['Seasonal Difference Price'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b3e2b50>
```

Seasonal Difference of average monthly prices of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the months and the average monthly prices of energy per EUR/MWH.

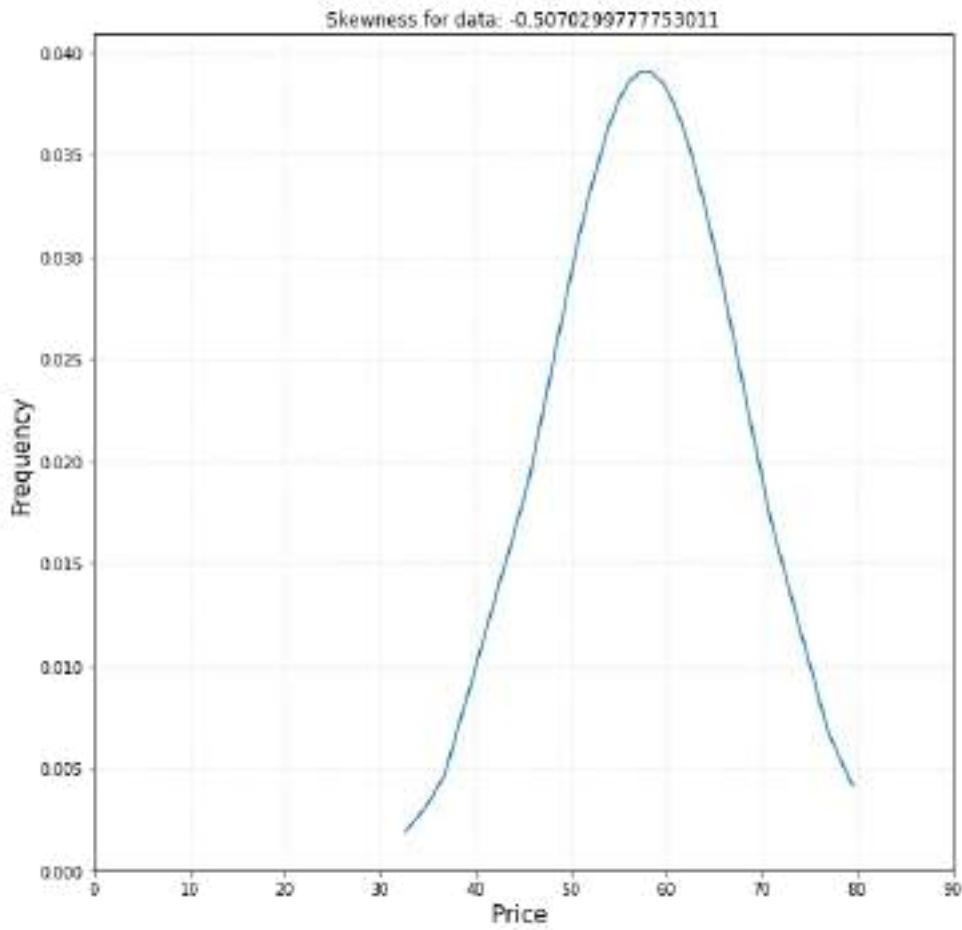
In []: #Bell Curves

```
PriceResults_mean = np.mean(df_Price["Price"])
PriceResults_std = np.std(df_Price["Price"])

PriceResultspdf = stats.norm.pdf(df_Price["Price"].sort_values(), PriceResults_mean, PriceResults_std)

plt.plot(df_Price["Price"].sort_values(), PriceResultspdf)
plt.xlim([0,90])
plt.xlabel("Price ", size=15)
plt.ylabel("Frequency", size=15)
plt.suptitle("Frequency distribution of energy price per EUR/MWH (scaled in decimals)")
plt.title(f'Skewness for data: {skew(df_Price["Price"])}')
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

Frequency distribution of energy price per EUR/MWH (scaled in decimals)



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: df_Price.describe(include = 'all') # Description Tables
```

```
Out[ ]:
```

	Price	Months	First Difference Price	Seasonal Difference Price
count	48.000000	48.00	47.000000	36.000000
mean	57.859848	24.50	0.035628	0.702849
std	10.320573	14.00	6.733390	15.177103
min	32.618667	1.00	-19.654286	-25.735417
25%	51.528411	12.75	-2.243930	-11.485827
50%	59.622281	24.50	1.235697	2.351931
75%	64.999583	36.25	2.817439	8.565413
max	79.492083	48.00	11.896949	33.913360

Below are the respective seasonality and distribution analysis' for the predictive models of the average monthly prices of energy per EUR/MWH; given all other variables constant.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model for the predicted average monthly prices of energy per EUR/MWH ; given all other variables constant.

```
In [ ]: print(list(predictions))
```

```
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445  
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739  
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293  
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75  
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.  
03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,  
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988  
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373  
92337, 62.89808335677895]
```

```
In [ ]: dfPriceReg = ({ "Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139  
"PriceReg" : [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896  
"Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10'],  
print(dfPriceReg) #Dataframes  
df_PriceReg= pd.DataFrame.from_dict(dfPriceReg, orient = "columns")  
print(df_PriceReg)  
  
PriceReg = list(df_PriceReg["PriceReg"])  
print(PriceReg)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.095989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'PriceReg': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.4647921113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.7515000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.8246168252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.45000826687, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.511725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price      PriceReg     Dates
0   64.949019  52.821613  2015-01
1   56.383854  53.036006  2015-02
2   55.522463  53.250399  2015-03
3   58.354083  53.464792  2015-04
4   57.294059  53.679185  2015-05
5   65.974903  53.893578  2015-06
6   71.072043  54.107971  2015-07
7   63.998065  54.322364  2015-08
8   60.254792  54.536757  2015-09
9   59.406765  54.751150  2015-10
10  60.726792  54.965543  2015-11
11  61.901761  55.179936  2015-12
12  45.578723  55.394329  2016-01
13  36.752083  55.608722  2016-02
14  36.818008  55.823115  2016-03
15  32.618667  56.037508  2016-04
16  34.691371  56.251901  2016-05
17  46.266319  56.466294  2016-06
18  47.502016  56.680687  2016-07
19  47.602339  56.895080  2016-08
20  50.405597  57.109473  2016-09
21  60.182430  57.323866  2016-10
22  62.581056  57.538259  2016-11
23  67.595134  57.752652  2016-12
24  79.492083  57.967045  2017-01
25  59.837798  58.181438  2017-02
26  50.959892  58.395831  2017-03
27  51.717917  58.610224  2017-04
28  53.772621  58.824617  2017-05
29  56.258222  59.039010  2017-06
30  55.252581  59.253403  2017-07
31  54.084328  59.467796  2017-08
32  55.816556  59.682189  2017-09
33  63.925289  59.896582  2017-10
34  65.430653  60.110975  2017-11
35  65.151277  60.325368  2017-12
36  56.511976  60.539761  2018-01
37  60.877098  60.754154  2018-02
38  48.279717  60.968547  2018-03
39  50.400736  61.182939  2018-04
40  61.633763  61.397332  2018-05
41  64.348139  61.611725  2018-06
42  67.783441  61.826118  2018-07
43  70.363911  62.040511  2018-08
44  76.914042  62.254904  2018-09
45  70.362215  62.469297  2018-10
46  67.042608  62.683690  2018-11
47  66.623514  62.898083  2018-12
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.4647921113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895]
```

In []: #ADF Tests
from statsmodels.tsa.stattools import adfuller

```

def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_PriceReg["PriceReg"])

test_result=adfuller(df_PriceReg["PriceReg"])

```

ADF Test Statistic : -0.08785253051074082
 p-value : 0.9506351073236636
 #Lags Used : 0
 Number of Observations : 47
 weak evidence against null hypothesis,indicating it is non-stationary

```

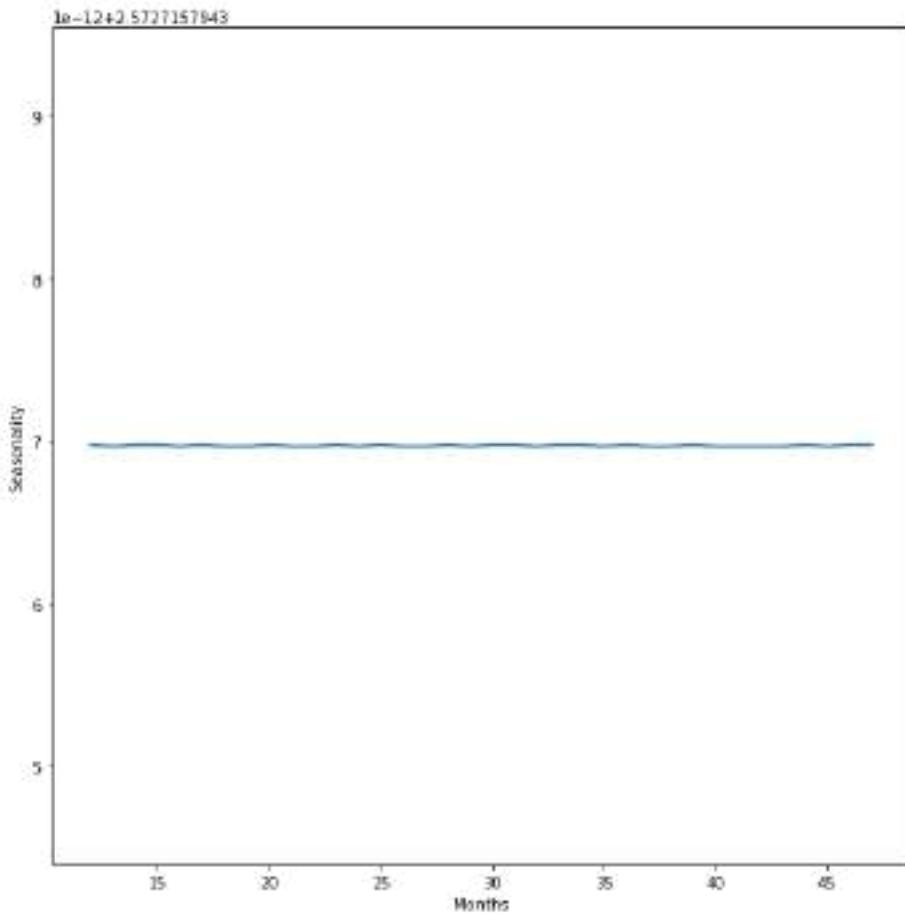
In [ ]: df_PriceReg['First Difference'] = df_PriceReg["PriceReg"]- df_PriceReg["PriceReg"].shift(1) # Seasonality value
df_PriceReg['Seasonal Difference']=df_PriceReg["PriceReg"]- df_PriceReg["PriceReg"].shift(12)

plt.suptitle("Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_PriceReg['Seasonal Difference'].plot()
# Seasonality Plot
#Predictive Linear Seasonality Plot

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b366850>
 Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted in the average monthly predicted prices per EUR/MWH.

In []: df_PriceReg.head()

	Price	PriceReg	Dates	First Difference	Seasonal Difference
0	64.949019	52.821613	2015-01	NaN	NaN
1	56.383854	53.036006	2015-02	0.214393	NaN
2	55.522463	53.250399	2015-03	0.214393	NaN
3	58.354083	53.464792	2015-04	0.214393	NaN
4	57.294059	53.679185	2015-05	0.214393	NaN

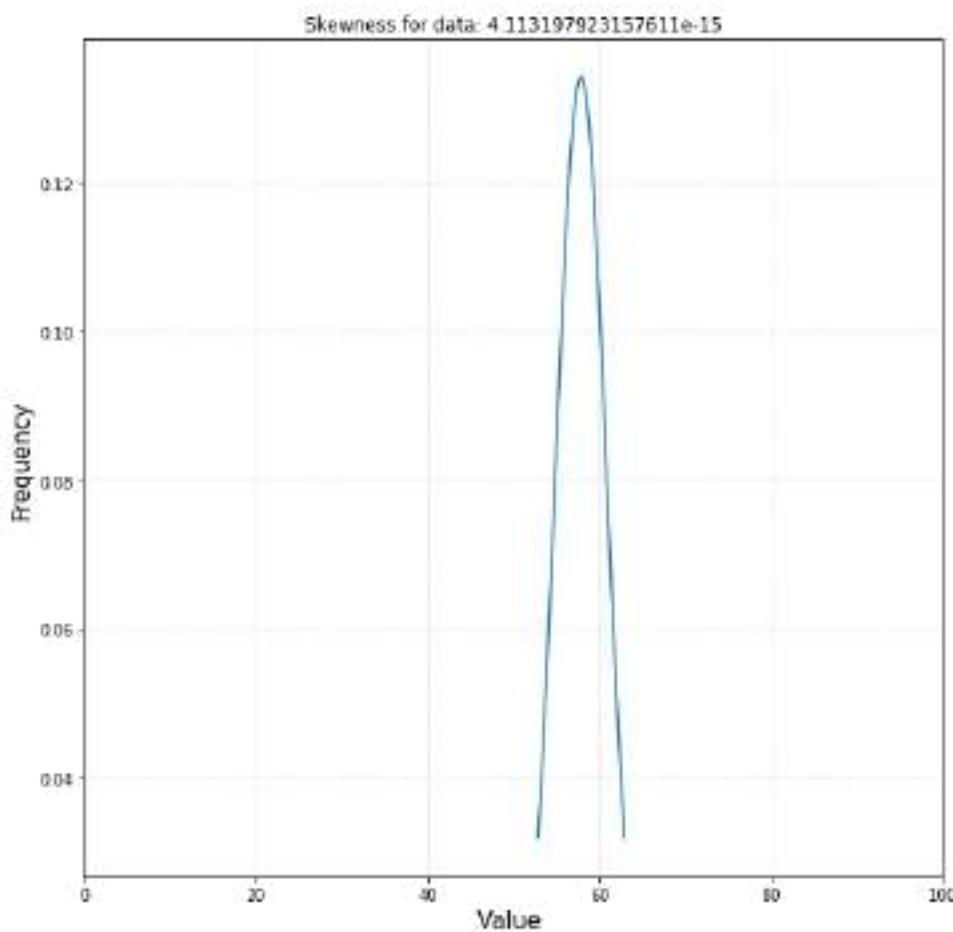
In []: #Bell Curves

```
PriceRegResults_mean = np.mean(df_PriceReg["PriceReg"])
PriceRegResults_std = np.std(df_PriceReg["PriceReg"])

PriceRegResultspdf = stats.norm.pdf(df_PriceReg["PriceReg"].sort_values(), PriceRegResults_mean, PriceRegResults_std)

plt.plot(df_PriceReg["PriceReg"].sort_values(), PriceRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_PriceReg["PriceReg"])}')
plt.suptitle("Frequency distribution of average monthly predicted linear prices per EUR/MWH (scaled in decimals")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

Frequency distribution of average monthly predicted linear prices per EUR/MWH (scaled in decimals)



This bell shaped curve is skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
PriceReg_Dict = {key: i for i, key in enumerate(PriceReg)}

def Hist_PriceReg(PriceReg_Dict):
    for k, v in PriceReg_Dict.items(): print(f"{v}:{k}")
```

```

print(PriceReg_Dict)

plt.bar(list(PriceReg_Dict.values()), PriceReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted linear prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

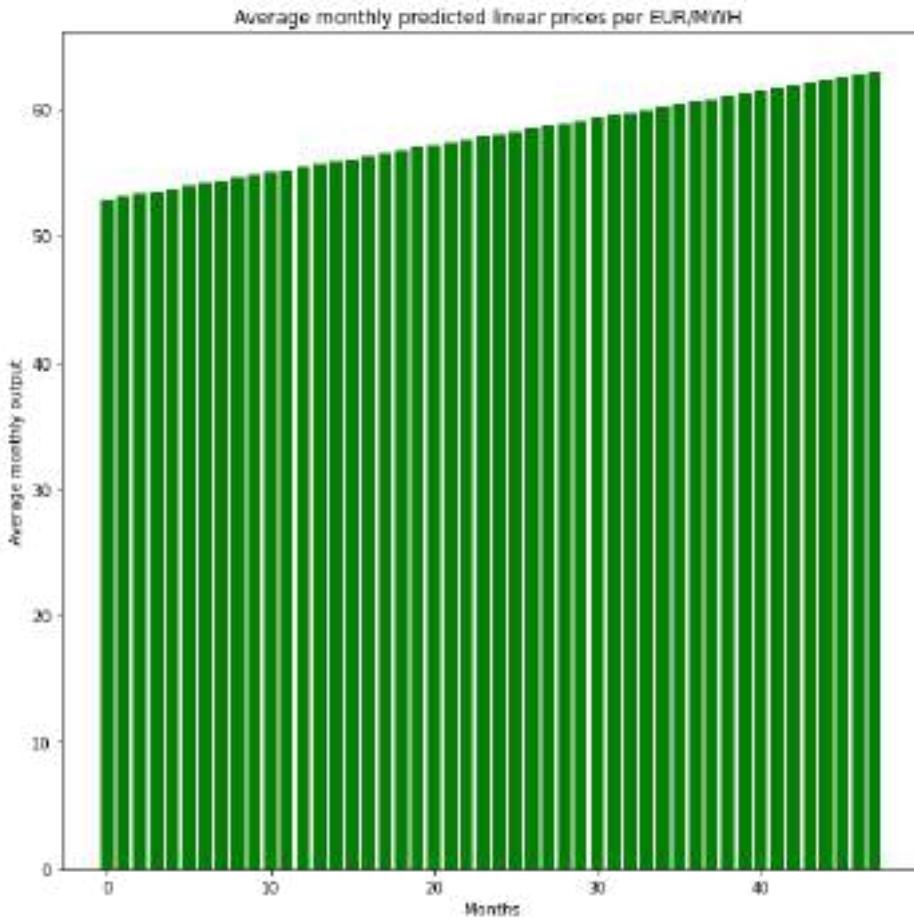
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{52.821613162566635: 0, 53.03600614542222: 1, 53.2503991282778: 2, 53.46479211113338: 3, 53.67918509398896: 4, 53.89357807684454: 5, 54.10797105970013: 6, 54.322364042555705: 7, 54.53675702541128: 8, 54.75115000826687: 9, 54.96554299112245: 10, 55.179935973978026: 11, 55.39432895683361: 12, 55.60872193968919: 13, 55.823114922544775: 14, 56.03750790540035: 15, 56.25190088825593: 16, 56.46629387111152: 17, 56.680686853967096: 18, 56.895079836822674: 19, 57.10947281967826: 20, 57.32386580253384: 21, 57.53825878538942: 22, 57.752651768245: 23, 57.96704475110058: 24, 58.181437733956166: 25, 58.395830716811744: 26, 58.61022369966732: 27, 58.82461668252291: 28, 59.03900966537849: 29, 59.25340264823407: 30, 59.46779563108965: 31, 59.68218861394523: 32, 59.896581596800814: 33, 60.11097457965639: 34, 60.32536756251197: 35, 60.53976054536756: 36, 60.754153528223135: 37, 60.96854651107871: 38, 61.1829394939343: 39, 61.39733247678988: 40, 61.611725459645456: 41, 61.82611844250104: 42, 62.04051142535662: 43, 62.254904408212205: 44, 62.469297391067784: 45, 62.68369037392337: 46, 62.89808335677895: 47}

```



The green bars represent the observation value for each respective month. As one can observe, the price is steadily increasing each month in this predictive model.

```
In [ ]: df_PriceReg.describe(include = 'all') # Description Tables
```

Out[]:

	Price	PriceReg	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	4.700000e+01	3.600000e+01
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	57.859848	NaN	2.143930e-01	2.572716e+00
std	10.320573	3.001502	NaN	3.491384e-15	3.606236e-15
min	32.618667	52.821613	NaN	2.143930e-01	2.572716e+00
25%	51.528411	55.340731	NaN	2.143930e-01	2.572716e+00
50%	59.622281	57.859848	NaN	2.143930e-01	2.572716e+00
75%	64.999583	60.378966	NaN	2.143930e-01	2.572716e+00
max	79.492083	62.898083	NaN	2.143930e-01	2.572716e+00

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model for the predicted average monthly prices of energy per EUR/MWH; given all other variables constant.

In []:

print(list(ypred))

```
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]
```

In []:

```
dfPriceQuad = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.2940591, 'PriceQuad' : [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398]}, #Dataframes
print(dfPriceQuad)
df_PriceQuad= pd.DataFrame.from_dict(dfPriceQuad, orient = "columns")
print(df_PriceQuad)

PriceQuad = list(df_PriceQuad["PriceQuad"])
print(PriceQuad)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.08432795698924, 55.81655555555556, 63.92528859060402, 65.430652777936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'PriceQuad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.0079917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	PriceQuad	Dates
0	64.949019	27.188078	2015-01
1	56.383854	23.225548	2015-02
2	55.522463	22.878236	2015-03
3	58.354083	24.055117	2015-04
4	57.294059	23.602709	2015-05
5	65.974903	27.724722	2015-06
6	71.072043	30.587839	2015-07
7	63.998065	26.702482	2015-08
8	60.254792	24.901804	2015-09
9	59.406765	24.518415	2015-10
10	60.726792	25.119121	2015-11
11	61.901761	25.672300	2015-12
12	45.578723	19.546363	2016-01
13	36.752083	17.633406	2016-02
14	36.818008	17.644052	2016-03
15	32.618667	17.075344	2016-04
16	34.691371	17.328255	2016-05
17	46.266319	19.736629	2016-06
18	47.502016	20.093546	2016-07
19	47.602339	20.123369	2016-08
20	50.405597	21.007992	2016-09
21	60.182430	24.868735	2016-10
22	62.581056	26.000055	2016-11
23	67.595134	28.599305	2016-12
24	79.492083	36.035067	2017-01
25	59.837798	24.712150	2017-02
26	50.959892	21.194644	2017-03
27	51.717917	21.456173	2017-04
28	53.772621	22.201511	2017-05
29	56.258222	23.174311	2017-06
30	55.252581	22.771345	2017-07
31	54.084328	22.319233	2017-08
32	55.816556	22.995762	2017-09
33	63.925289	26.665789	2017-10
34	65.430653	27.438370	2017-11
35	65.151277	27.292830	2017-12
36	56.511976	23.278006	2018-01
37	60.877098	25.188915	2018-02
38	48.279717	20.328049	2018-03
39	50.400736	21.006372	2018-04
40	61.633763	25.544593	2018-05
41	64.348139	26.879918	2018-06
42	67.783441	28.703098	2018-07
43	70.363911	30.170478	2018-08
44	76.914042	34.272280	2018-09
45	70.362215	30.169486	2018-10
46	67.042608	28.070862	2018-11
47	66.623514	28.297338	2018-12

[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.59930478662582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511565746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.43836996505977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]

In []: #ADF Tests
from statsmodels.tsa.stattools import adfuller

```

def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_PriceQuad["PriceQuad"])

test_result=adfuller(df_PriceQuad["PriceQuad"])

```

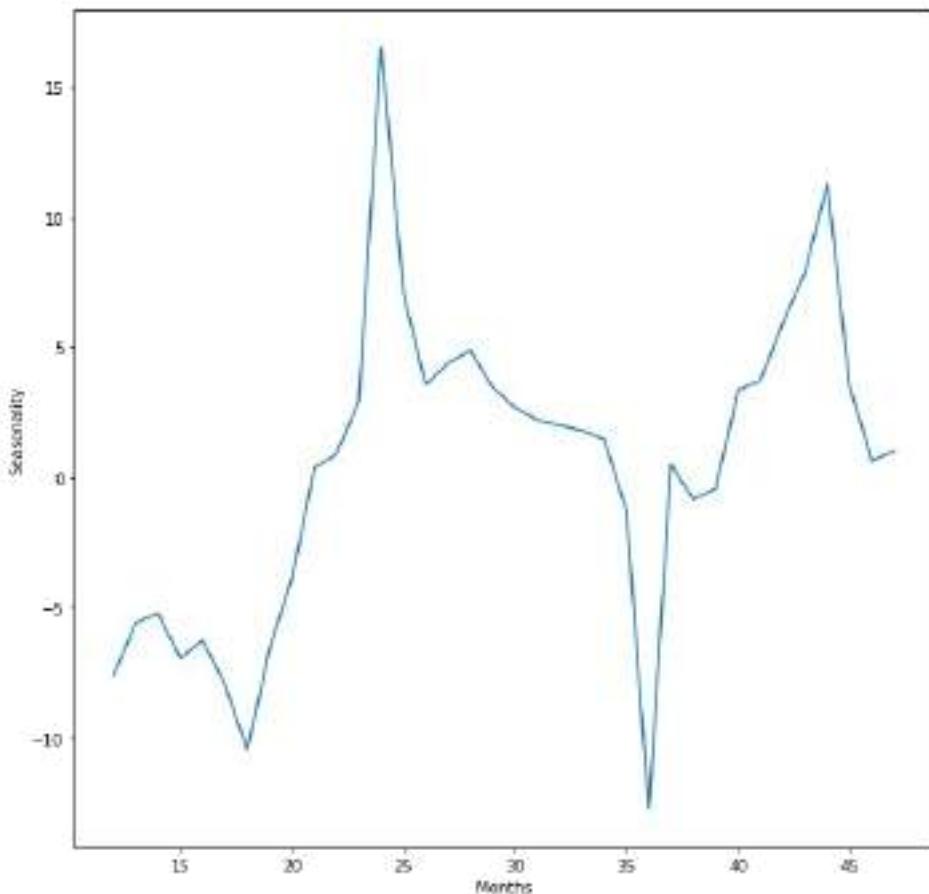
ADF Test Statistic : -2.672042104752486
 p-value : 0.07900240726821062
 #Lags Used : 0
 Number of Observations : 47
 weak evidence against null hypothesis,indicating it is non-stationary

```

In [ ]: df_PriceQuad['First Difference'] = df_PriceQuad["PriceQuad"]- df_PriceQuad["PriceQuad"].shift(1) # Seasonality
df_PriceQuad['Seasonal Difference']=df_PriceQuad["PriceQuad"]- df_PriceQuad["PriceQuad"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_PriceQuad.head() #Predictive Quadratic Seasonality Plot
df_PriceQuad['Seasonal Difference'].plot()
# Seasonality Plot

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b010410>
 Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there are sharp trenches depicted between the predicted average monthly prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

```

In [ ]: #Bell Curves

PriceQuadResults_mean = np.mean(df_PriceQuad["PriceQuad"])
PriceQuadResults_std = np.std(df_PriceQuad["PriceQuad"])

```

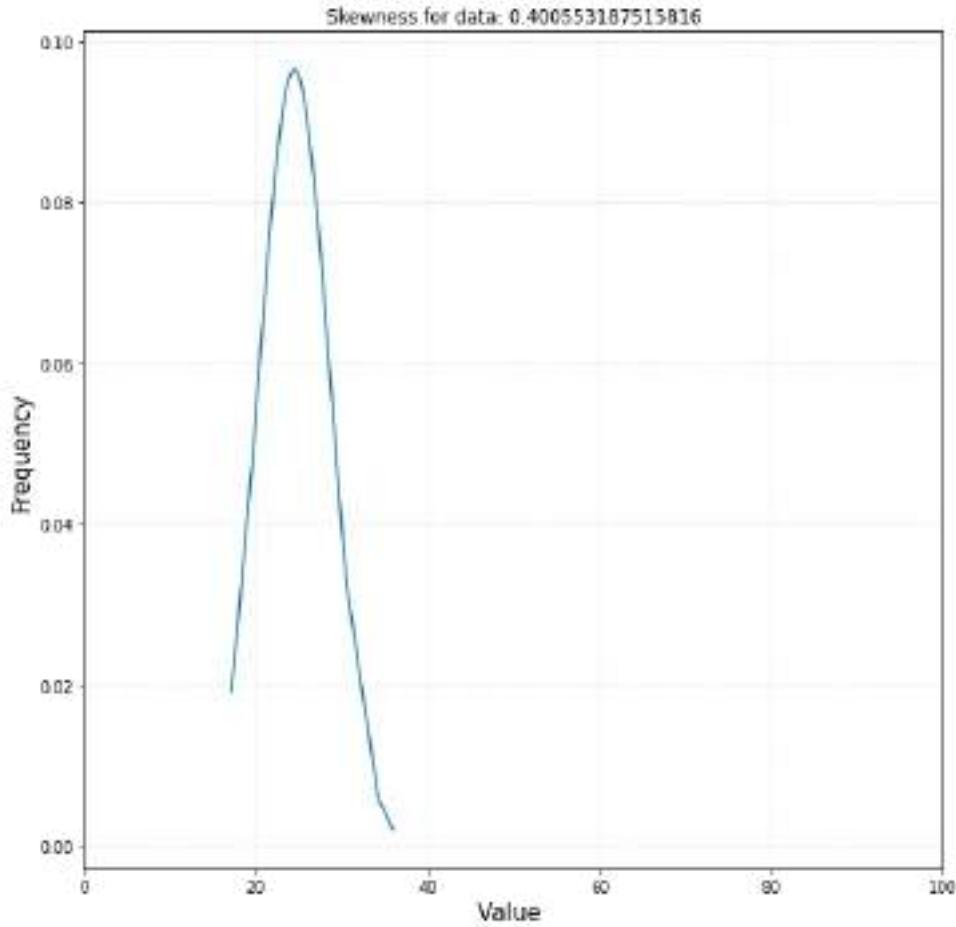
```

PriceQuadResultspdf = stats.norm.pdf(df_PriceQuad["PriceQuad"].sort_values(), PriceQuadResults_mean, PriceQuadResults_std)

plt.plot(df_PriceQuad["PriceQuad"].sort_values(), PriceQuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_PriceQuad["PriceQuad"])}')
plt.suptitle("Frequency distribution of average monthly predicted quadratic prices per EUR/MWH (scaled in decimals)", size=10)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

```

Frequency distribution of average monthly predicted quadratic prices per EUR/MWH (scaled in decimals)



This bell shaped curve is roughly symmetrical, hence they have a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```

In [ ]:
print(dicDates)
PriceQuad_Dict = {key: i for i, key in enumerate(PriceQuad)}

def Hist_PriceQuad(PriceQuad_Dict):
    for k, v in PriceQuad_Dict.items(): print(f"{v}:{k}")
print(PriceQuad_Dict)

plt.bar(list(PriceQuad_Dict.values()), PriceQuad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

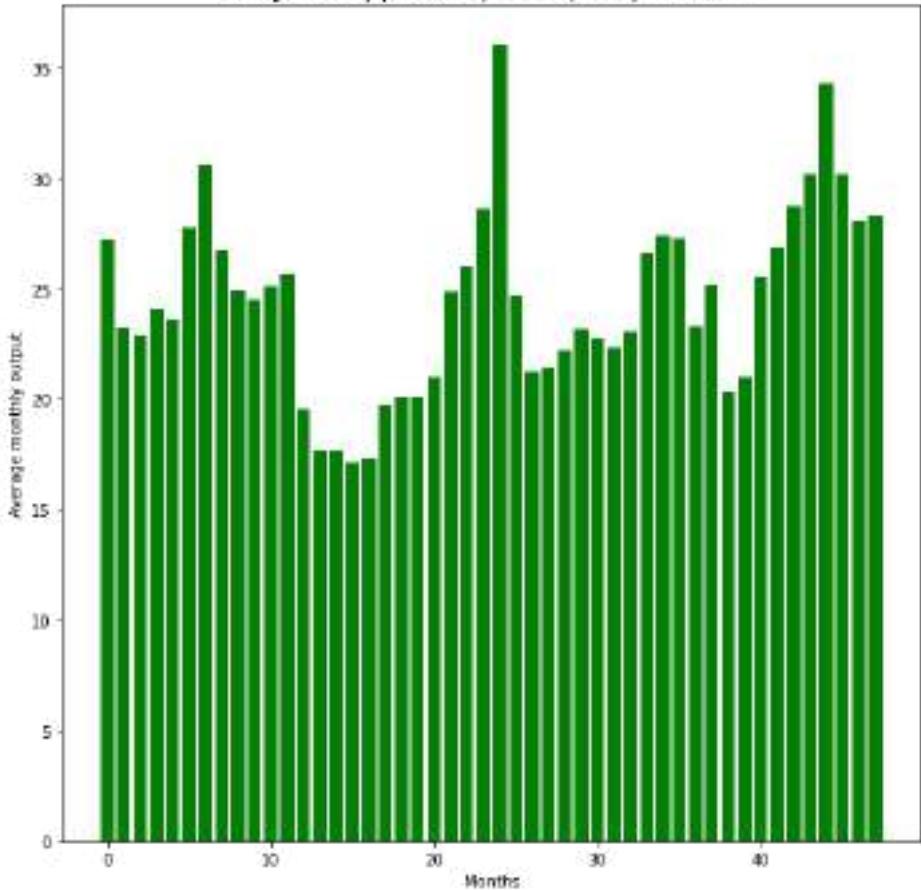
```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{27.188077690295593: 0, 23.22554817302482: 1, 22.87823628204505: 2, 24.055117412542895: 3, 23.602708631972398: 4, 27.724722433871626: 5, 30.587838932854936: 6, 26.70248165923099: 7, 24.901803909409455: 8, 24.518414867253377: 9, 25.119121355777786: 10, 25.672299682598407: 11, 19.546362569195352: 12, 17.633406195018726: 13, 17.644052319077353: 14, 17.075344431983726: 15, 17.328254644736486: 16, 19.736629098849825: 17, 20.09354637020233: 18, 20.123368510962656: 19, 21.007991576184747: 20, 24.868735417613472: 21, 26.000055062885394: 22, 28.599304786622582: 23, 36.03506726766325: 24, 24.71214950611809: 25, 21.194644314739115: 26, 21.456172637404084: 27, 22.201511151656746: 28, 23.17431072214061: 29, 22.771345027936817: 30, 22.31923276693945: 31, 22.99576213803661: 32, 26.665789045581803: 33, 27.438369965065977: 34, 27.292829521192132: 35, 23.2780059994575: 36, 25.18891499253064: 37, 20.32804924075167: 38, 21.006371788590553: 39, 25.54459336089593: 40, 26.879917631774212: 41, 28.703097663581467: 42, 30.17047810264582: 43, 34.27227943527835: 44, 30.169485792181256: 45, 28.070861800194585: 46, 28.297337605159356: 47}

```

Average monthly predicted quadratic prices per EUR/MWH



The green bars represent the observation value for each respective month. This is a multimodal histogram with troughs occurring roughly every ten months followed by sharp increases.

In []: df_PriceQuad.describe(include = 'all') # Description Tables

	Price	PriceQuad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	24.500000	NaN	0.023601	0.437028
std	10.320573	4.174498	NaN	3.145508	6.044904
min	32.618667	17.075344	NaN	-11.322918	-12.757061
25%	51.528411	21.390791	NaN	-0.510558	-4.228905
50%	59.622281	24.615282	NaN	0.356917	1.221411
75%	64.999583	27.214266	NaN	1.401352	3.515421
max	79.492083	36.035067	NaN	7.435762	16.488705

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(Logpred))
```

[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]

```
In [ ]: dfPriceLog = ({ "Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139, "PriceLog" : [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.27674267217683], print(dfPriceLog) #Dataframes df_PriceLog= pd.DataFrame.from_dict(dfPriceLog, orient = "columns") print(df_PriceLog)

PriceLog = list(df_PriceLog["PriceLog"])
print(PriceLog)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.430652777777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'PriceLog': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639, 23.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price      PriceLog      Dates
0   64.949019  27.297348  2015-01
1   56.383854  23.917581  2015-02
2   55.522463  23.577680  2015-03
3   58.354083  24.695022  2015-04
4   57.294059  24.276743  2015-05
5   65.974903  27.702157  2015-06
6   71.072043  29.713461  2015-07
7   63.998065  26.922107  2015-08
8   60.254792  25.445032  2015-09
9   59.406765  25.110405  2015-10
10  60.726792  25.631280  2015-11
11  61.901761  26.094917  2015-12
12  45.578723  19.653934  2016-01
13  36.752083  16.170990  2016-02
14  36.818008  16.197004  2016-03
15  32.618667  14.539966  2016-04
16  34.691371  15.357844  2016-05
17  46.266319  19.925256  2016-06
18  47.502016  20.412855  2016-07
19  47.602339  20.452442  2016-08
20  50.405597  21.558593  2016-09
21  60.182430  25.416478  2016-10
22  62.581056  26.362963  2016-11
23  67.595134  28.341491  2016-12
24  79.492083  33.035963  2017-01
25  59.837798  25.280488  2017-02
26  50.959892  21.777315  2017-03
27  51.717917  22.076427  2017-04
28  53.772621  22.887202  2017-05
29  56.258222  23.868007  2017-06
30  55.252581  23.471186  2017-07
31  54.084328  23.010200  2017-08
32  55.816556  23.693728  2017-09
33  63.925289  26.893390  2017-10
34  65.430653  27.487399  2017-11
35  65.151277  27.377158  2017-12
36  56.511976  23.968137  2018-01
37  60.877098  25.690591  2018-02
38  48.279717  20.719732  2018-03
39  50.400736  21.556675  2018-04
40  61.633763  25.989167  2018-05
41  64.348139  27.060244  2018-06
42  67.783441  28.415796  2018-07
43  70.363911  29.434036  2018-08
44  76.914042  32.018682  2018-09
45  70.362215  29.433366  2018-10
46  67.042608  27.958095  2018-11
47  66.623514  28.123467  2018-12
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]
```

In []: #ADF Tests
from statsmodels.tsa.stattools import adfuller

```

def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_PriceLog["PriceLog"])

test_result=adfuller(df_PriceLog["PriceLog"])

```

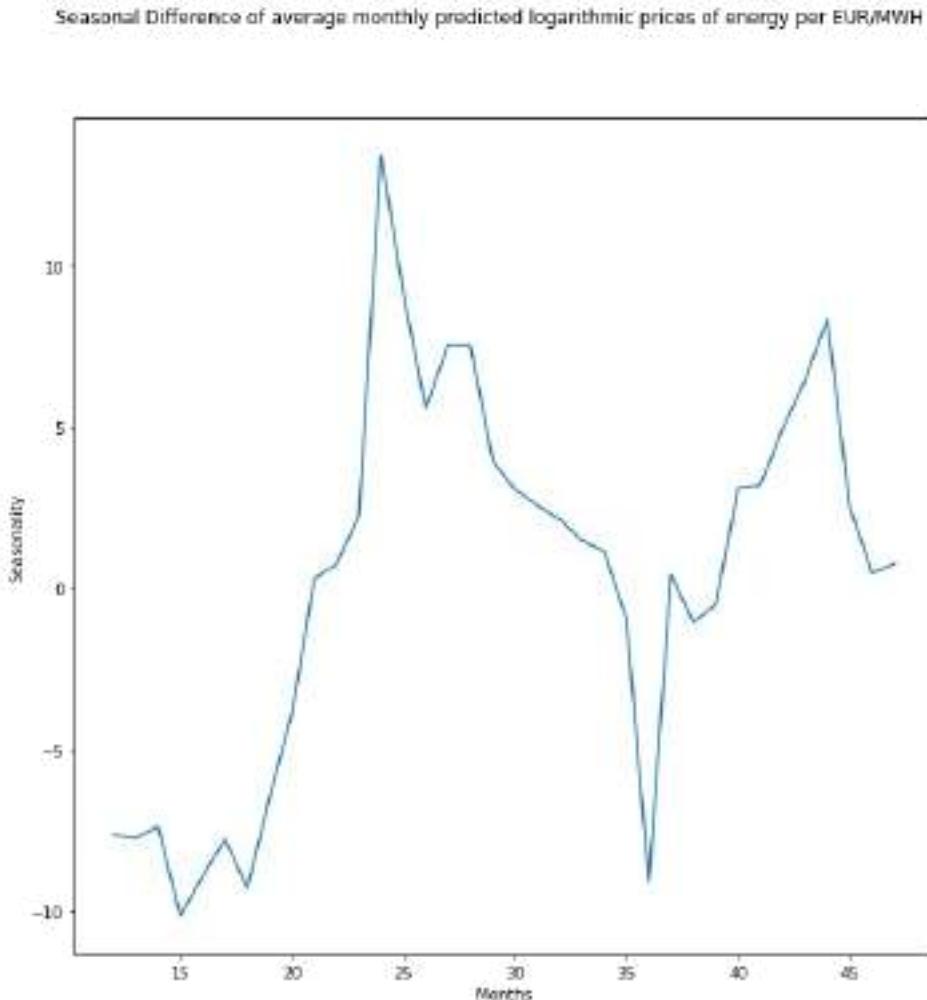
ADF Test Statistic : -2.583213988915304
 p-value : 0.09651646085287002
 #Lags Used : 1
 Number of Observations : 46
 weak evidence against null hypothesis,indicating it is non-stationary

```

In [ ]: df_PriceLog['First Difference'] = df_PriceLog["PriceLog"]- df_PriceLog["PriceLog"].shift(1) # Seasonality value
df_PriceLog['Seasonal Difference']=df_PriceLog["PriceLog"]- df_PriceLog["PriceLog"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_PriceLog.head() #Predictive Logarithmic Seasonality Plot
df_PriceLog['Seasonal Difference'].plot()
# Seasonality Plot

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60ad54890>



The blue line represents the trend line among the values themselves. As one can observe, there is volatility depicted between the predicted average monthly prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```

PriceLogResults_mean = np.mean(df_PriceLog["PriceLog"])
PriceLogResults_std = np.std(df_PriceLog["PriceLog"])

```

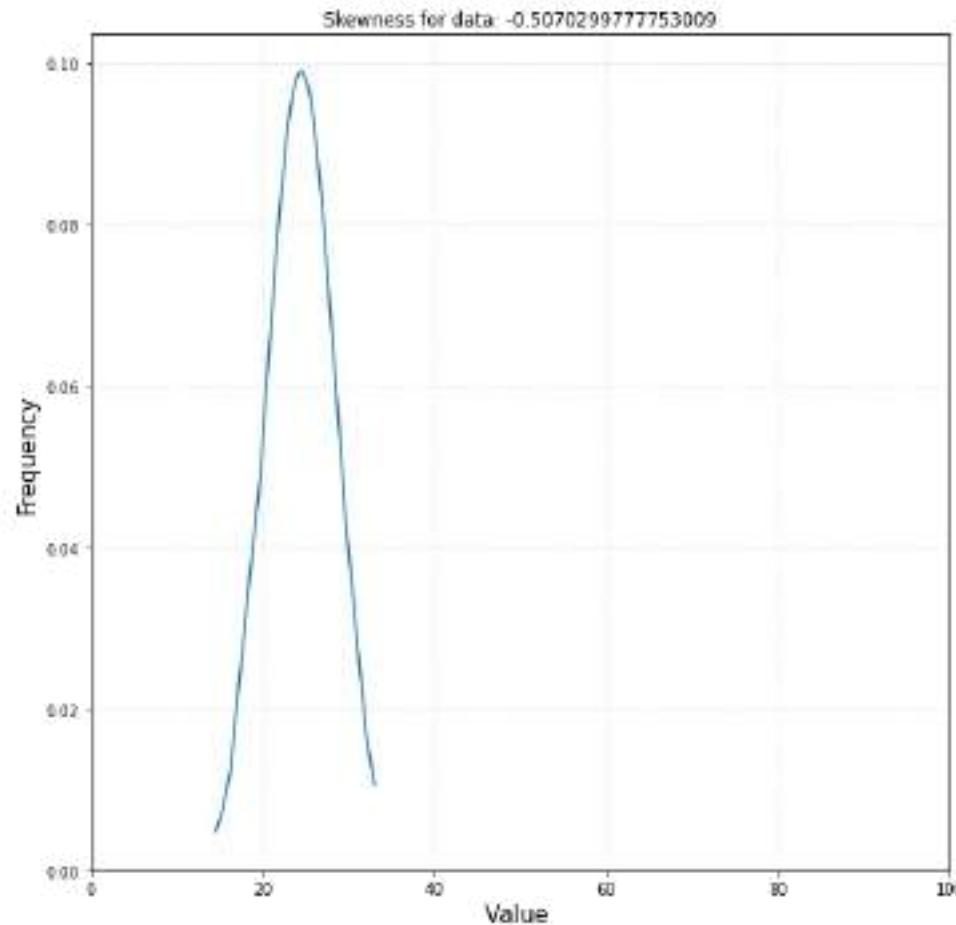
```

PriceLogResultspdf = stats.norm.pdf(df_PriceLog["PriceLog"].sort_values(), PriceLogResults_mean, PriceLogResults_std)

plt.plot(df_PriceLog["PriceLog"].sort_values(), PriceLogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_PriceLog["PriceLog"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic prices per EUR/MWH (scaled in decimals)")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

```

Frequency distribution of average monthly predicted logarithmic prices per EUR/MWH (scaled in decimals)



This bell shaped curve is slightly skewed to the left, hence it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring. If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```

In [ ]:
print(dicDates)
PriceLog_Dict = {key: i for i, key in enumerate(PriceLog)}

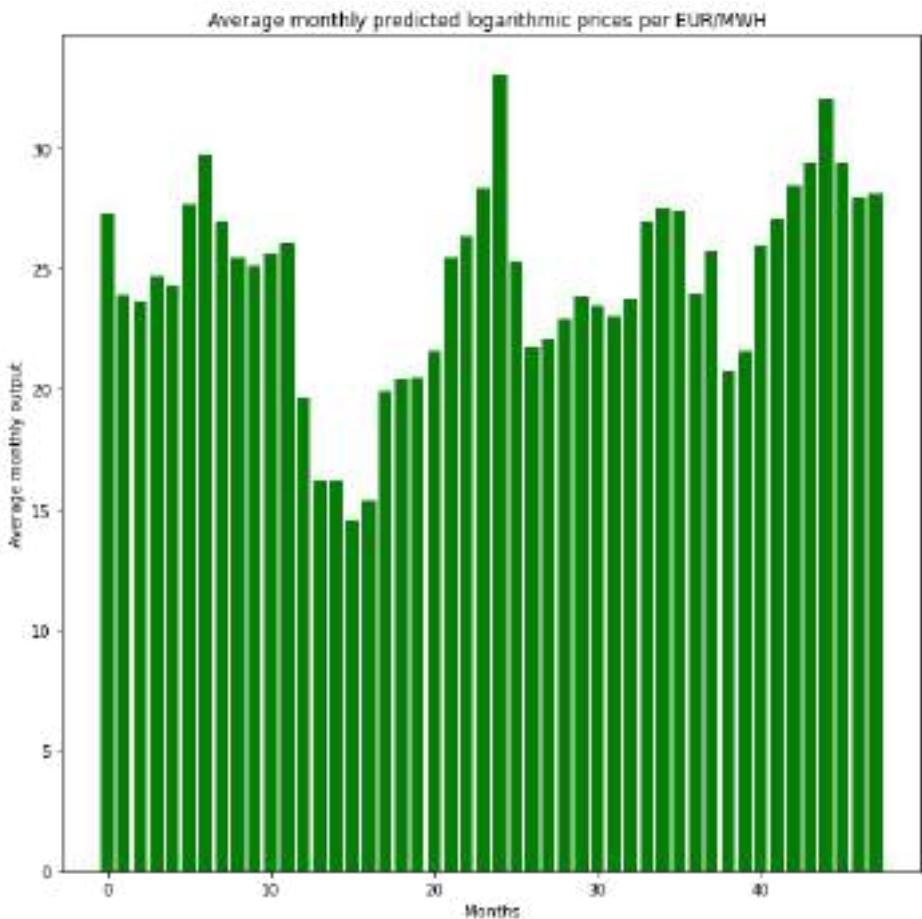
def Hist_PriceLog(PriceLog_Dict):
    for k, v in PriceLog_Dict.items(): print(f"{v}:{k}")
print(PriceLog_Dict)

plt.bar(list(PriceLog_Dict.values()), PriceLog_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly predicted logarithmic prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{27.297348375081718: 0, 23.917580710720195: 1, 23.57768040860245: 2, 24.695022488031967: 3, 24.276742672176834: 4, 27.70215663958084: 5, 29.71346061831328: 6, 26.922106949120572: 7, 25.44503172036812: 8, 25.110405022200517: 9, 25.631280368545422: 10, 26.094916817531914: 11, 19.653934415930614: 12, 16.170990077171673: 13, 16.197003623963596: 14, 14.5399662074255: 15, 15.357844118752933: 16, 19.925256208811728: 17, 20.412855439874555: 18, 20.452442187812732: 19, 21.55859284131481: 20, 25.416478012537848: 21, 26.362962874201227: 22, 28.341491281477328: 23, 33.03596300746373: 24, 25.28048812361594: 25, 21.777314693468632: 26, 22.076426999163463: 27, 22.887202207806165: 28, 23.868007024659466: 29, 23.471186295255887: 30, 23.01020010054923: 31, 23.693727745821242: 32, 26.893389962364463: 33, 27.48739853201623: 34, 27.37715831385097: 35, 23.968136817101986: 36, 25.690590519617974: 37, 20.71973214185461: 38, 21.556674673151495: 39, 25.98916652718635: 40, 27.060244403968117: 41, 28.415795989223025: 42, 29.434035669877737: 43, 32.01868170373189: 44, 29.433366230197276: 45, 27.958095077371567: 46, 28.123467161134005: 47}
```



The green bars represent the observation value for each respective month. There are troughs followed by gradual increases in this histogram.

In []: df_PriceLog.describe(include = 'all') # Description Tables

	Price	PriceLog	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	24.500000	NaN	0.017577	0.277340
std	10.320573	4.072442	NaN	2.658822	5.988890
min	32.618667	14.539966	NaN	-7.755475	-10.155056
25%	51.528411	22.001649	NaN	-0.968129	-4.532245
50%	59.622281	25.195447	NaN	0.487599	0.935372
75%	64.999583	27.317301	NaN	1.111746	3.379866
max	79.492083	33.035963	NaN	4.694472	13.382029

In []: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Logpred)
plt.suptitle(" Autocorrelations of average monthly Price Log")

```

plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Price Log")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
Log_Autocorrelations = sm.tsa.acf(Logpred, fft=False) #Autocorrelations
print(Log_Autocorrelations)

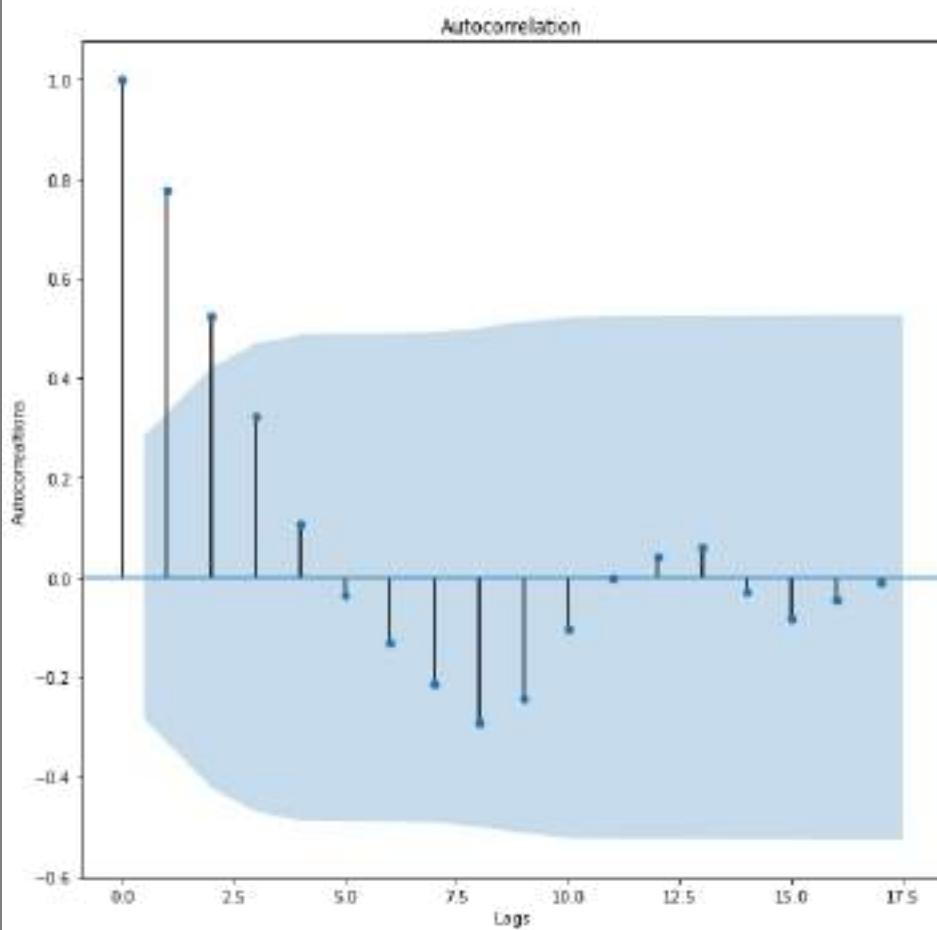
```

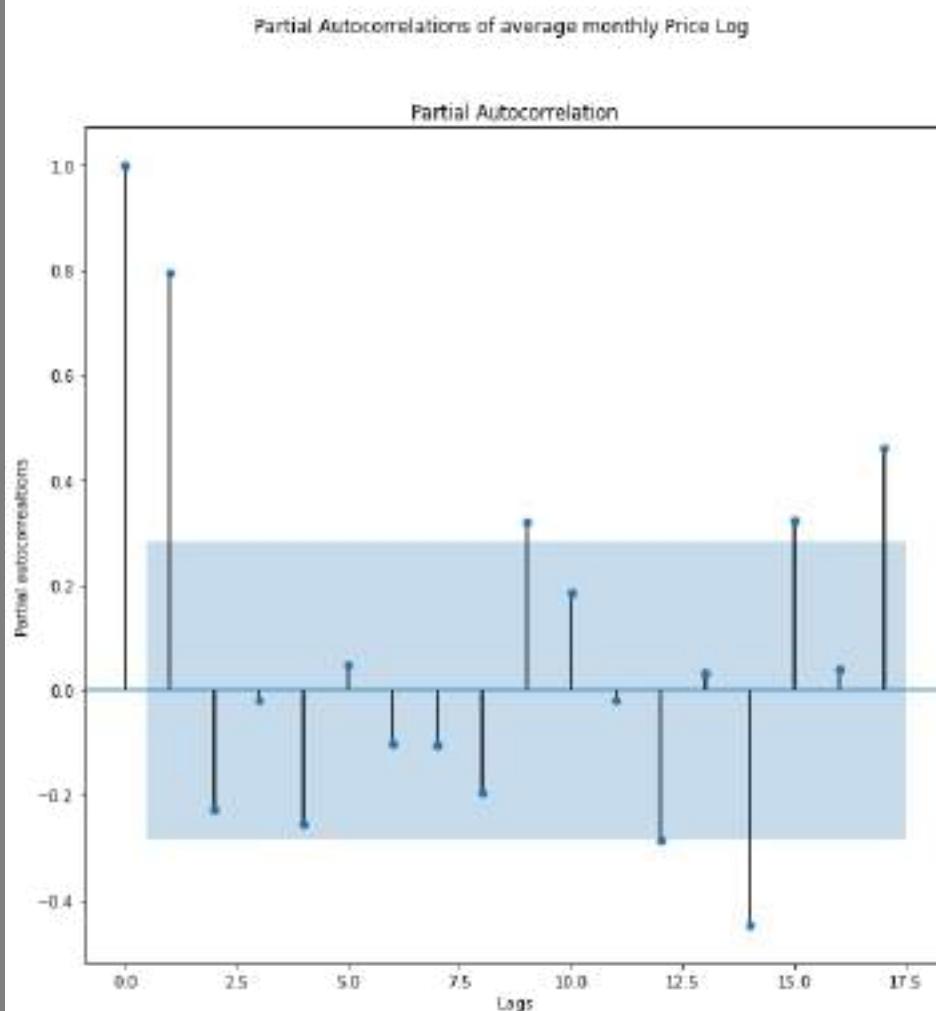
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,

```
[ 1.0000000e+00  7.77956931e-01  5.24250500e-01  3.23980125e-01
 1.07028578e-01 -3.60532396e-02 -1.30254803e-01 -2.14671334e-01
-2.92492069e-01 -2.40360570e-01 -1.03132122e-01 -1.29994731e-04
 4.12510174e-02  5.95304581e-02 -2.97302907e-02 -8.26016779e-02
-4.42595047e-02 -9.62022191e-03  2.61107117e-02  1.18842238e-02
-4.25037632e-03 -4.63630833e-03  2.97453465e-02  7.37711560e-02
 2.35072578e-02 -7.93840645e-02 -1.44789264e-01 -2.21944605e-01
-2.72163150e-01 -2.77044335e-01 -2.71423639e-01 -2.54543382e-01
-1.99886952e-01 -1.05612474e-01 -1.41939737e-02  4.81982436e-02
 7.79627111e-02  9.66167880e-02  7.93061418e-02  6.38831956e-02
 5.25335207e-02]
```

Autocorrelations of average monthly Price Log





The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

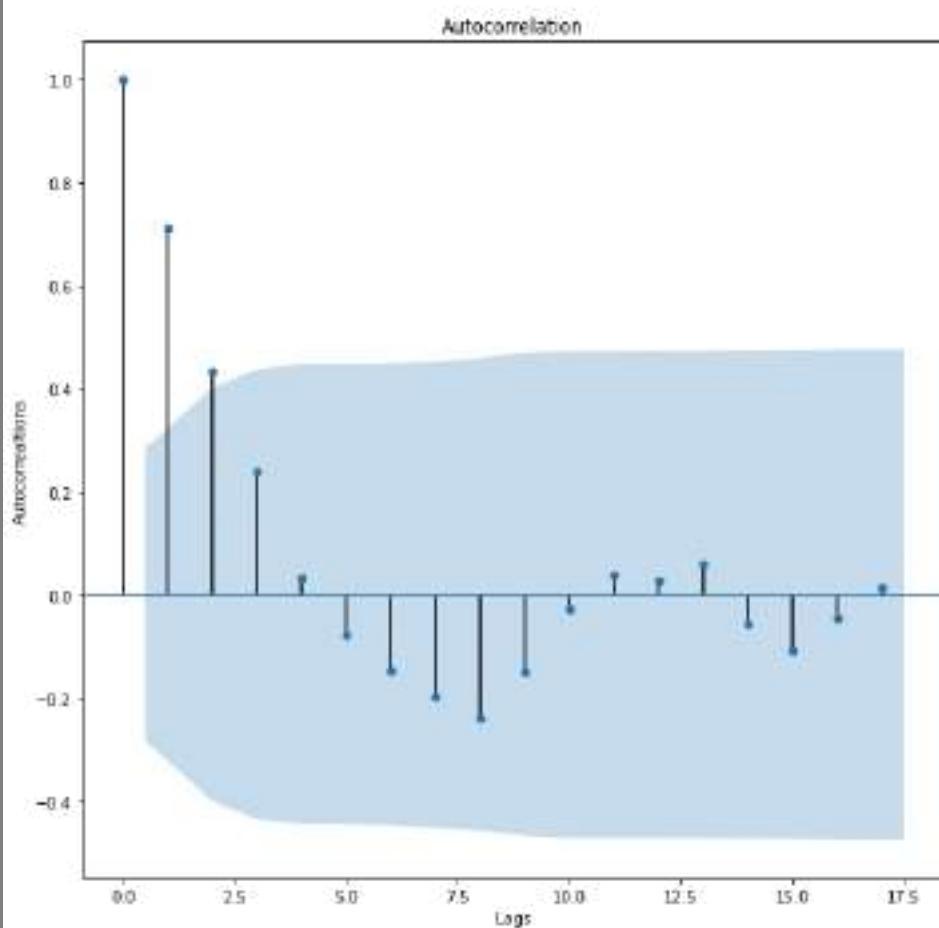
```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(ypred)
plt.suptitle(" Autocorrelations of average monthly Quadratic Price")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(ypred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic Price")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
```

```
Quad_Autocorrelations = sm.tsa.acf(ypred, fft=False) #Autocorrelations
print(Quad_Autocorrelations)
```

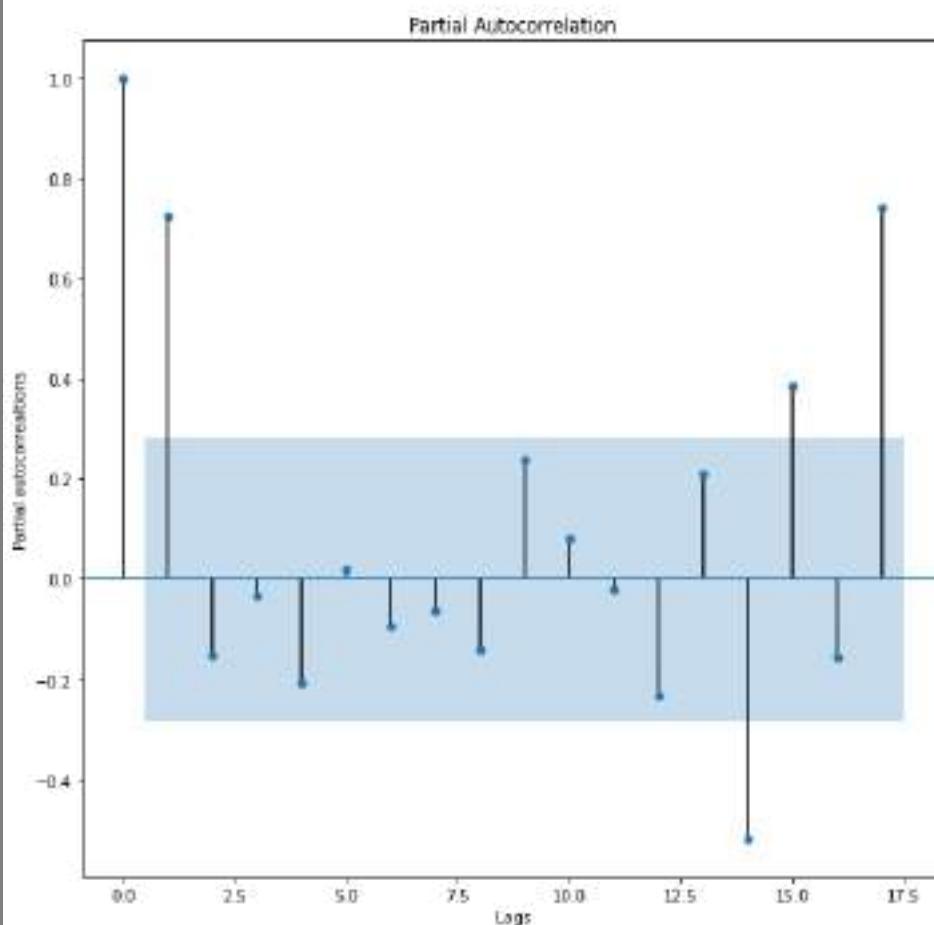
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.          0.70892475  0.43315396  0.23686375  0.03076224 -0.0790933
 -0.14892593 -0.19870526 -0.23893082 -0.1499733 -0.0269835  0.03800657
 0.02600565  0.0577859 -0.05656382 -0.10733708 -0.0462279  0.01278301
 0.09118926  0.07673635  0.06473332  0.03585042  0.03113278  0.05801957
 0.00858007 -0.10403746 -0.15328618 -0.20440041 -0.24098566 -0.24728244
 -0.25736766 -0.2478468 -0.21108429 -0.11914686 -0.02341841  0.03833327
 0.07284715  0.10320258  0.095823   0.06578207  0.03669171]
```

Autocorrelations of average monthly Quadratic Price



Partial Autocorrelations of average monthly Quadratic Price



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

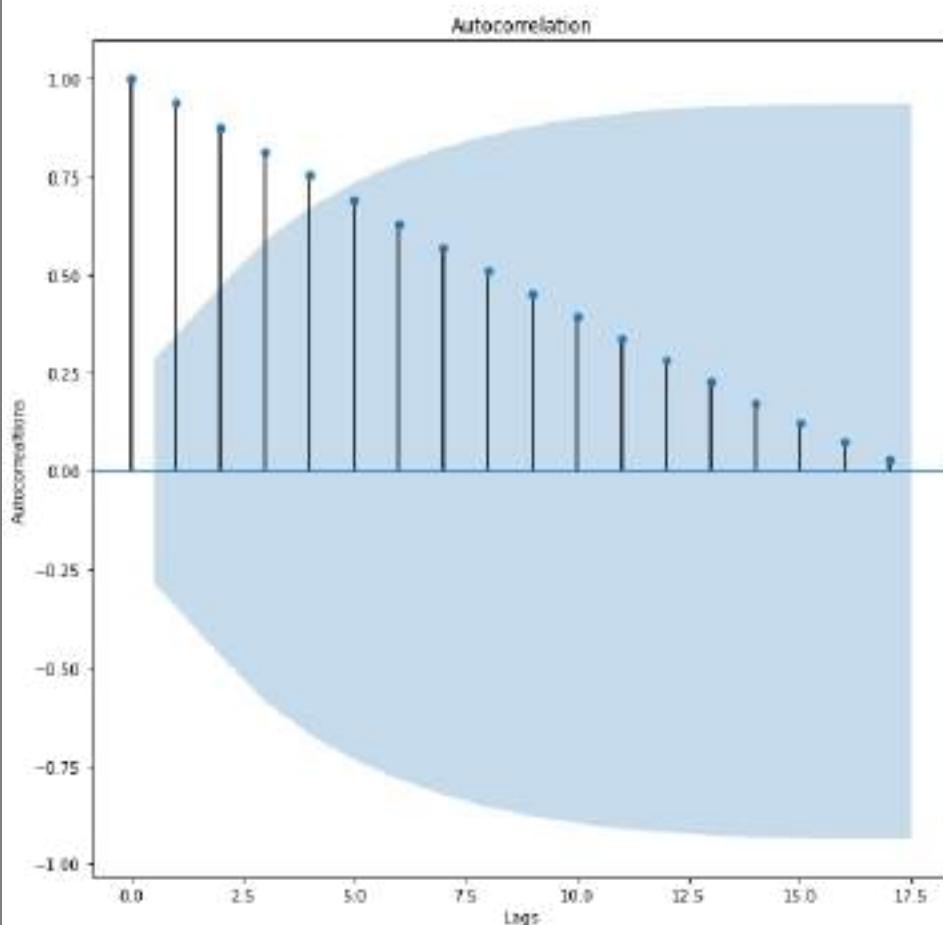
```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(predictions)
plt.suptitle(" Autocorrelations of average monthly Linear Price")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(predictions)
plt.suptitle("Partial Autocorrelations of average monthly Linear Price")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Pred_Autocorrelations = sm.tsa.acf(predictions, fft=False) #Autocorrelations
print(Pred_Autocorrelations)
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

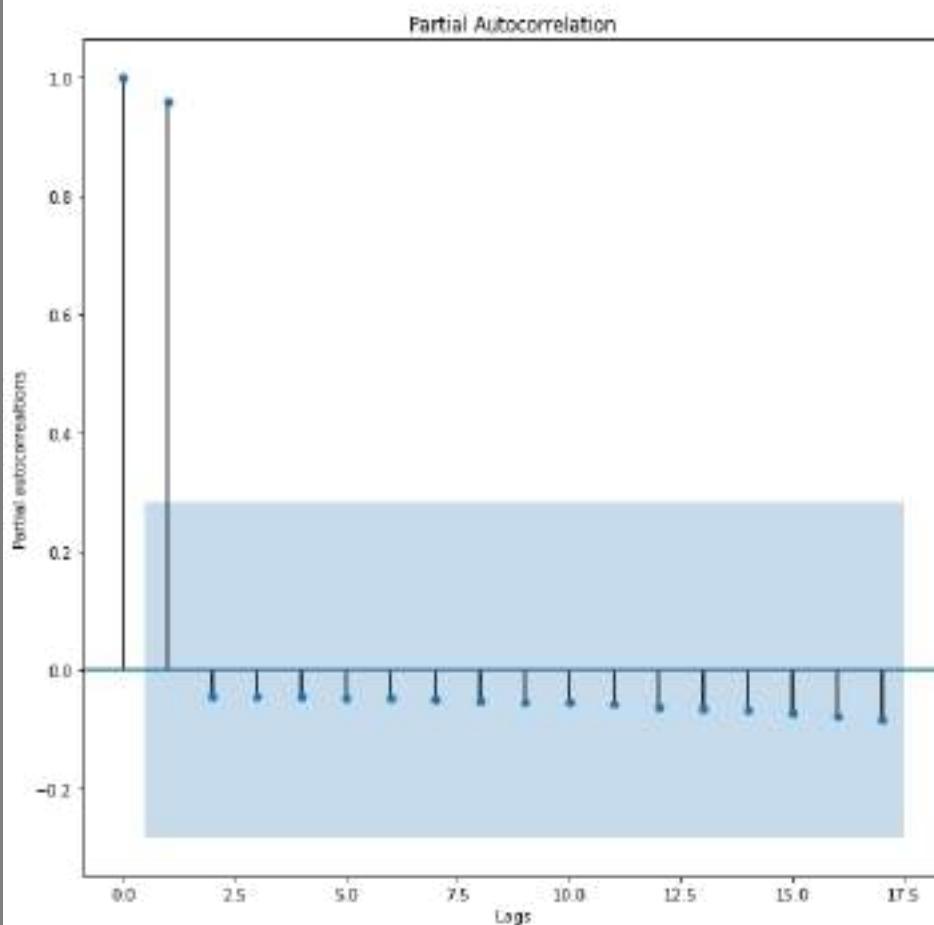
FutureWarning,

```
[ 1.          0.9375    0.87510855  0.81293422  0.75108554  0.68967108
 0.62879939  0.56857903  0.50911854  0.45052649  0.39291142  0.33638189
 0.28104646  0.22701368  0.1743921   0.12329027  0.07381676  0.02608011
-0.01981112 -0.06374837 -0.1056231   -0.14532675 -0.18275076 -0.21778658
-0.25032566 -0.28025944 -0.30747937 -0.3318769  -0.35334347 -0.37177052
-0.3870495  -0.39907186 -0.40772905 -0.41291251 -0.41451368 -0.41242401
-0.40653495 -0.39673795 -0.38292445 -0.36498589 -0.34281372]
```

Autocorrelations of average monthly Linear Price



Partial Autocorrelations of average monthly Linear Price



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

And below are the prices of energy per EUR/MWH rounded to the hundredth.

```
In [ ]: Rounded_Y = [round(item, 2) for item in Price_Actual]
print(Rounded_Y)
#Rounded Price
```

```
[64.95, 56.38, 55.52, 58.35, 57.29, 65.97, 71.07, 64.0, 60.25, 59.41, 60.73, 61.9, 45.58, 36.75, 36.82, 32.62, 34.69, 46.27, 47.5, 47.6, 50.41, 60.18, 62.58, 67.6, 79.49, 59.84, 50.96, 51.72, 53.77, 56.26, 55.25, 54.08, 55.82, 63.93, 65.43, 65.15, 56.51, 60.88, 48.28, 50.4, 61.63, 64.35, 67.78, 70.36, 76.91, 70.36, 67.04, 66.62]
```

The average monthly prices of energy per EUR/MWH were rounded for the sake of the box and whisker graphs displayed throughout the

analysis.

```
In [ ]: import pandas as pd
import statsmodels.api as sm
import scipy.stats as stats
```

Below is the analysis of all the resources included in the regression. All of these resources have been analyzed separately and collectively. Quadratic, logarithmic and linear models have been constructed as references for predicted prices per resource as well; given all other variables constant. To further elaborate, the predictive models per resource does not take any other resources into consideration, yet the price of energy per EUR/MWH adjusts to that of the respectively analyzed resource to limit bias. It is as if all other resources discontinued while the outputs and demand of that one resource remained the same nonetheless. However, the multipolynomial and multilinear regression models do take all resources into consideration.

A consideration to keep in mind when reading this analysis is that the conducted dataframes, plots, graphs and tests are in no particular order. It would be best to seek the dataframes, plots, graphs and tests by their respective hashtags. That being said, these analysis' begins below this paragraph.

```
In [ ]:
```

The first analyzed resource was fossil oil.

The results from the list directly below will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Dataframes analyzed by resource
dfFossilOil = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.2940591,
                           "Fossil_Oil"] : [306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 332.5672043010752],
                print(dfFossilOil)
                df_FossilOil= pd.DataFrame.from_dict(dfFossilOil, orient = "columns")
                print(df_FossilOil)
                df_FossilOil[Ratio] = df_FossilOil["Fossil_Oil"]/df_FossilOil["Price"]
                pdToListFossilOil = list(df_FossilOil["Ratio"])
                print(pdToListFossilOil)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.09589232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.9252885906402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Oil': [306.0204638472033, 319.23958333333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963839, 360.23387096774195, 323.9139784946237, 343.4916666666667, 323.39112903225805, 346.06388888888887, 330.6514131897712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875, 277.9153225806452, 289.82916666666665, 286.6900269541779, 283.30645161290323, 281.386111111111, 276.9959731543624, 271.56944444444446, 276.63978494623655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.4902777777778, 291.9206989247312, 299.3930555555556, 308.2002688172043, 306.23521505376345, 310.801388888889, 287.4187919463087, 303.54305555555555, 293.9260752688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916666666666, 280.4502688172043, 285.12083333333334, 281.5450874831763, 283.3333333333333, 296.1361111111111, 291.299328590604, 269.02150537634407, 272.98333333333335], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	Fossil_Oil	Dates
0	64.949019	306.020464	2015-01
1	56.383854	319.239583	2015-02
2	55.522463	319.333782	2015-03
3	58.354083	338.781337	2015-04
4	57.294059	332.567204	2015-05
5	65.974903	319.229485	2015-06
6	71.072043	360.233871	2015-07
7	63.998065	323.913978	2015-08
8	60.254792	343.491667	2015-09
9	59.406765	323.391129	2015-10
10	60.726792	346.063889	2015-11
11	61.901761	330.651413	2015-12
12	45.578723	337.465054	2016-01
13	36.752083	297.254310	2016-02
14	36.818008	294.052490	2016-03
15	32.618667	259.887500	2016-04
16	34.691371	277.915323	2016-05
17	46.266319	289.829167	2016-06
18	47.502016	286.690027	2016-07
19	47.602339	283.306452	2016-08
20	50.405597	281.386111	2016-09
21	60.182430	276.995973	2016-10
22	62.581056	271.569444	2016-11
23	67.595134	276.639785	2016-12
24	79.492083	279.522849	2017-01
25	59.837798	291.401786	2017-02
26	50.959892	302.507402	2017-03
27	51.717917	261.490278	2017-04
28	53.772621	291.920699	2017-05
29	56.258222	299.393056	2017-06
30	55.252581	308.200269	2017-07
31	54.084328	306.235215	2017-08
32	55.816556	310.801389	2017-09
33	63.925289	287.418792	2017-10
34	65.430653	303.543056	2017-11
35	65.151277	293.926075	2017-12
36	56.511976	285.739247	2018-01
37	60.877098	295.900298	2018-02
38	48.279717	288.126514	2018-03
39	50.400736	257.629167	2018-04
40	61.633763	280.450269	2018-05
41	64.348139	285.120833	2018-06
42	67.783441	281.545087	2018-07
43	70.363911	283.333333	2018-08
44	76.914042	296.136111	2018-09
45	70.362215	291.299329	2018-10
46	67.042608	269.021505	2018-11
47	66.623514	272.983333	2018-12

[4.711702646478498, 5.661897152147204, 5.751434010315323, 5.805614923503275, 5.804566988170348, 4.838650334531596, 5.068573460217559, 5.061308977758065, 5.700653129246291, 5.443675118216539, 5.6987019961215175, 5.341551018408021, 7.4040041202492315, 8.088094153705416, 7.986648525467014, 7.967447065075214, 8.01107926345912, 6.264366177099672, 6.035323346601258, 5.951523796777401, 5.582437796948807, 4.6026053669926625, 4.339483283457277, 4.092599080785782, 3.516360846780745, 4.869861480689363, 5.9361860592871185, 5.056086838592902, 5.42879803273591, 5.321765312329655, 5.578024867227532, 5.662180277016627, 5.568265289669134, 4.4961672959667265, 4.6391567662709905, 4.511439979947413, 5.056260079995415, 4.8606176427379415, 5.967858344556159, 5.11161515773717, 4.550270065632229, 4.4309103302219315, 4.153596865402372, 4.026685386551291, 3.8502216850665367, 4.139996585263025, 4.012694543070627, 4.097402214308303]

In []: fossil_oil1 = fossil_oil
fossil_oil1 = sm.add_constant(fossil_oil1)

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the average monthly outputs of fossil oil versus the predicted average monthly prices of energy per EUR/MWH.

In []: `print(fossil_oil)`

```
[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963  
839, 360.23387096774195, 323.9139784946237, 343.4916666666667, 323.39112903225805, 346.06388888888887, 330.65141  
31897712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875, 277.9153225806452, 289.829166666  
66665, 286.6900269541779, 283.30645161290323, 281.3861111111111, 276.9959731543624, 271.569444444444446, 276.6397  
8494623655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.4902777777778, 291.9206989247312, 299  
.3930555555556, 308.2002688172043, 306.23521505376345, 310.8013888888889, 287.4187919463087, 303.543055555555555,  
293.9260752688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916666666666, 280.45026881720  
43, 285.12083333333334, 281.5450874831763, 283.3333333333333, 296.1361111111111, 291.2993288590604, 272.983333333  
33335, 269.02150537634407]
```

In []: `#Logarithmic OLS regressions`

```
Logpricevalues = ((np.log(Price_Actual)))  
LogFossilOilvalues = ((np.log(fossil_oil)))  
Log = np.polyfit(np.log(Price_Actual), fossil_oil1, 1)  
lin2 = LinearRegression()  
lin2.fit(np.log(fossil_oil1), Price_Actual)  
FossilOil_Log = sm.OLS(Price_Actual, fossil_oil1).fit()
```

```
FossilOil_Logpred = FossilOil_Log.predict(fossil_oil1)
```

```
FossilOil_Log.summary() #OLS Logarithmic summary table
```

```
#Log  
Log = np.polyfit(np.log(fossil_oil), Price_Actual, 1)  
print(Log)
```

```
y = 18.80220214 * LogFossilOilvalues - 49.22000295
```

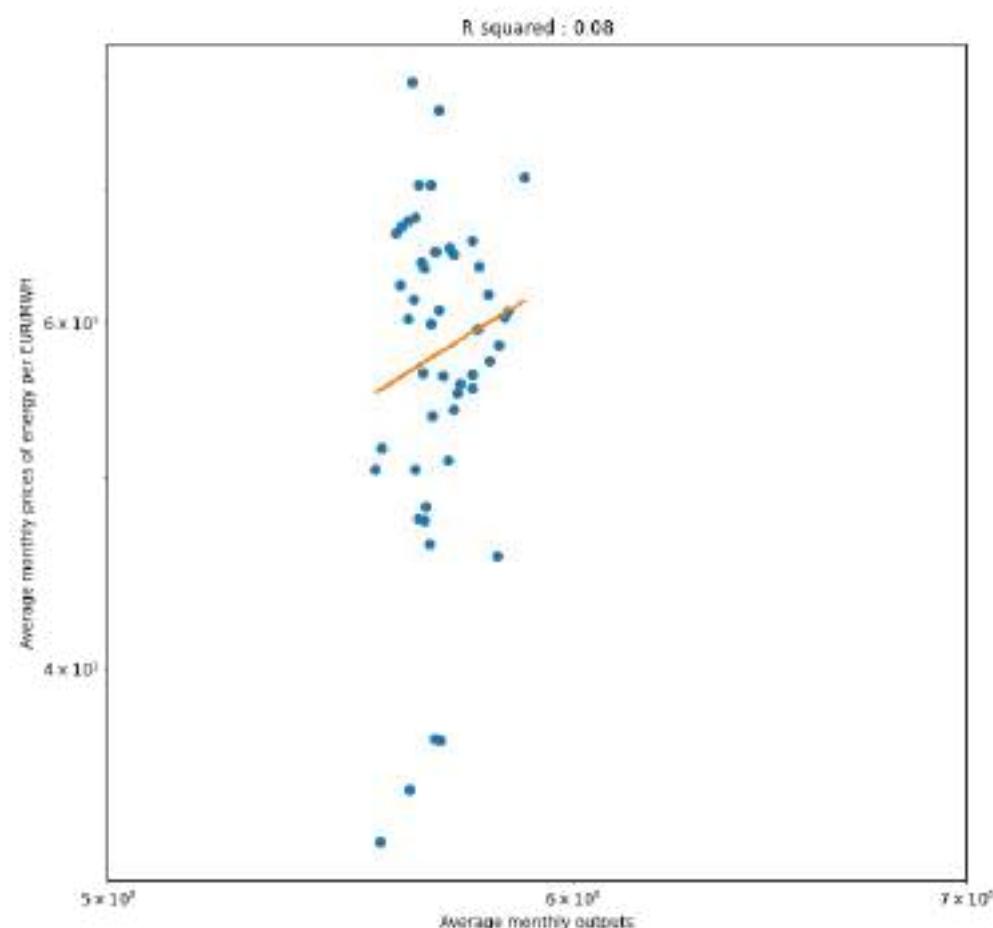
```
#OLS Logarithmic Scatterplots
```

```
plt.suptitle("Logarithmic average monthly fossil oil outputs versus predicted average monthly prices of energy in EUR/MWH")  
plt.title("R squared : 0.08")  
plt.ylabel("Average monthly prices of energy per EUR/MWH")  
plt.xlabel("Average monthly outputs")  
plt.yscale("log")  
plt.xscale("log")  
plt.plot(LogFossilOilvalues, Price_Actual, "o")  
plt.plot(LogFossilOilvalues, y)  
  
plt.xlim([5, 7])
```

```
[ 18.80220214 -49.22000295]
```

Out[]: (5, 7)

Logarithmic average monthly fossil oil outputs versus predicted average monthly prices of energy per EUR/MWH



The blue dots represent the observations and the orange line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH. The blue dots represent the observations and the orange line is the model of best fit.

In []: FossilOil_Log.summary() #OLS Logarithmic summary table

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.020			
Model:	OLS	Adj. R-squared:	-0.001			
Method:	Least Squares	F-statistic:	0.9576			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.333			
Time:	05:20:57	Log-Likelihood:	-179.15			
No. Observations:	48	AIC:	362.3			
Df Residuals:	46	BIC:	366.0			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	39.5996	18.719	2.115	0.040	1.919	77.280
x1	0.0612	0.063	0.979	0.333	-0.065	0.187
Omnibus:	1.353	Durbin-Watson:	0.413			
Prob(Omnibus):	0.508	Jarque-Bera (JB):	0.921			
Skew:	-0.339	Prob(JB):	0.631			
Kurtosis:	3.043	Cond. No.	3.76e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 3.76e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceFossilOilLog = FossilOil_Log.get_influence()
#Logarithmic OLS regression residuals
```

```
standardized_residualsFossilOilLog = influenceFossilOilLog.resid_studentized_internal

print(standardized_residualsFossilOilLog)
```

```
[ 6.48464911e-01 -2.72004425e-01 -3.57608810e-01 -2.00240571e-01
-2.66443802e-01  6.74566839e-01  9.96604805e-01  4.53066360e-01
-3.76568603e-02  1.24549118e-03 -5.64805685e-03  2.06007759e-01
-1.47961111e+00 -2.05955884e+00 -2.03457635e+00 -2.30493392e+00
-2.16229313e+00 -1.08529385e+00 -9.46478822e-01 -9.17872866e-01
-6.31516208e-01  3.58163113e-01  6.30892475e-01  1.09241818e+00
2.24481145e+00  2.35273838e-01 -7.00625171e-01 -3.90533514e-01
-3.61953943e-01 -1.63166888e-01 -3.14929761e-01 -4.17413817e-01
-2.75542580e-01  6.60461039e-01  7.10089047e-01  7.40265153e-01
-5.66974068e-02  3.09868127e-01 -8.78276584e-01 -5.02094124e-01
 4.79330244e-01  7.16485215e-01  1.07749272e+00  1.31921067e+00
 1.87820637e+00  1.26692622e+00  1.06345147e+00  1.05033093e+00]
```

In []:

```
print(FossilOil_Logpred) # OLS logarithmic predicted values
```

```
[58.33094074 59.14007647 59.14584232 60.3362175 59.95585352 59.13945838
61.64931643 59.42619387 60.62453443 59.39419052 60.78197887 59.8385889
60.25564845 57.79436884 57.598387 55.50716496 56.61063901 57.33987949
57.14773431 56.94062734 56.82308425 56.55436609 56.22221097 56.53256399
56.70903493 57.43613872 58.11590799 55.60527019 57.46790114 57.92528037
58.46436552 58.34408555 58.62357878 57.19234165 58.17929981 57.59064923
57.08953756 57.71149039 57.23566097 55.3689335 56.76580181 57.05168474
56.83281512 56.94227276 57.72592442 57.42986738 56.3087544 56.06625288]
```

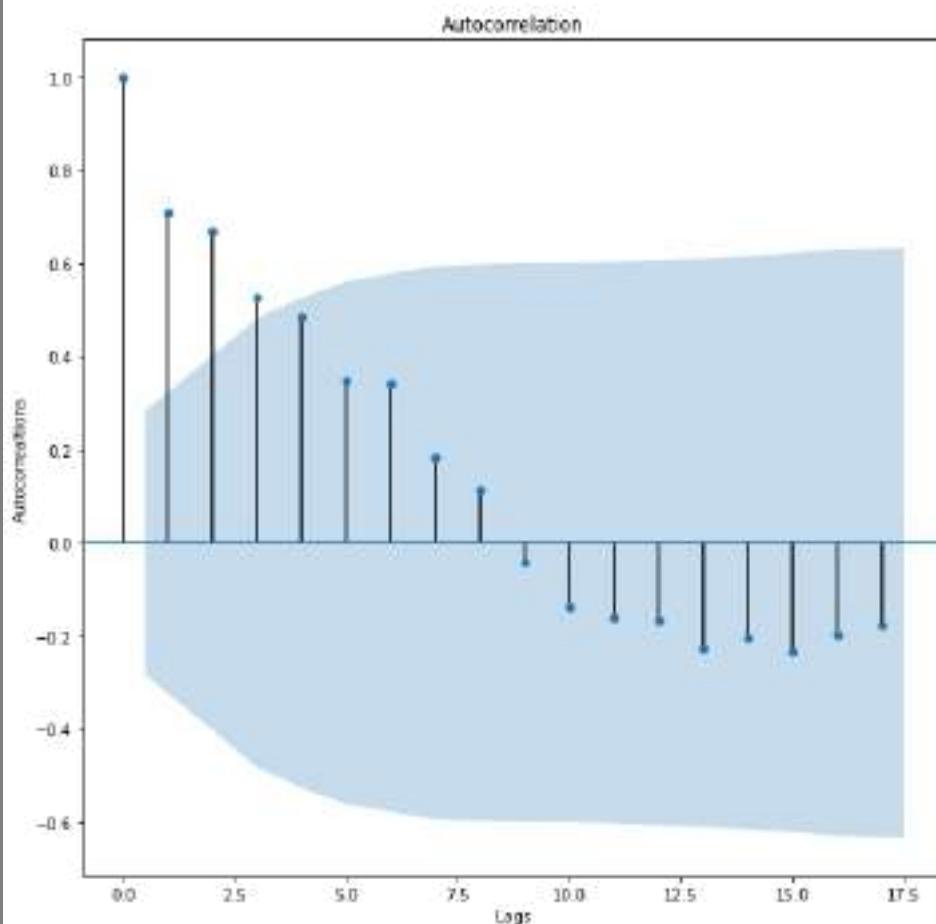
```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilOil_Logpred)
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilOil Log")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partial autocorrelation Plot
plot_pacf(FossilOil_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilOil Log")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
FossilOil_Log_Autocorrelations = sm.tsa.acf(FossilOil_Logpred, fft=False) #Autocorrelations
```

```
print(FossilOil_Log_Autocorrelations)
```

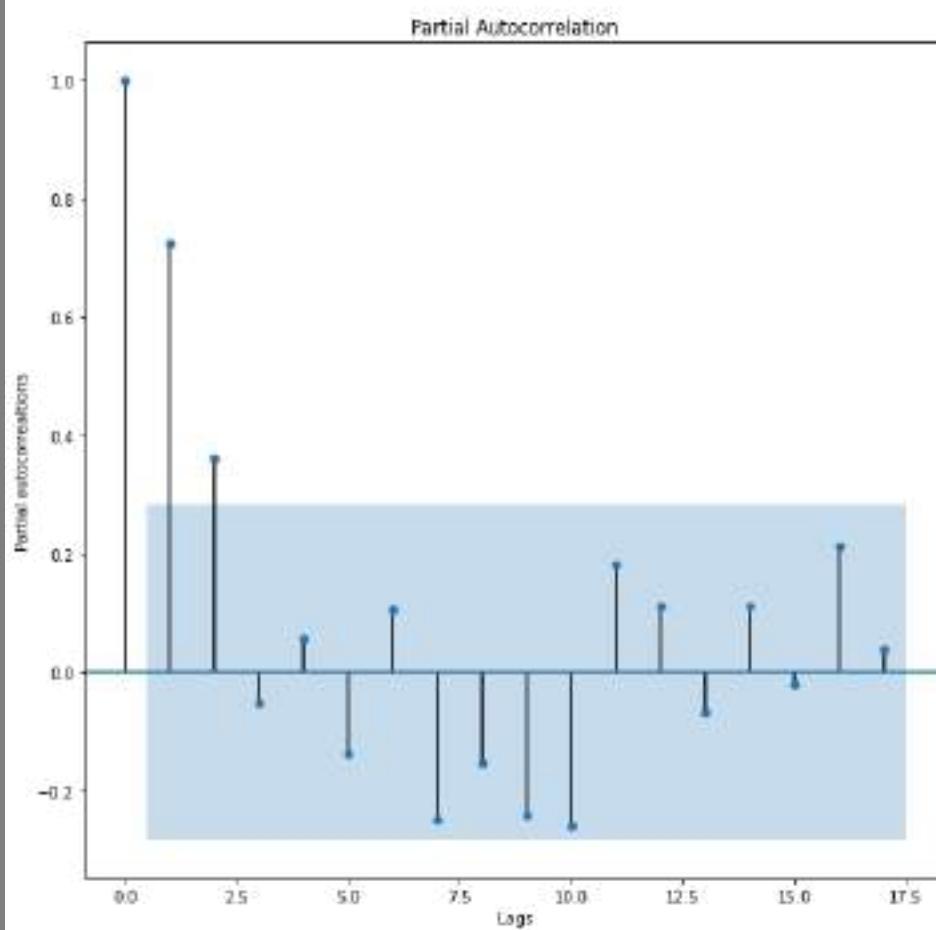
```
[ 1.          0.70952824  0.66802752  0.52851278  0.48281801  0.3500967
  0.3432444   0.18192468  0.11216314 -0.04078178 -0.13797814 -0.16229098
 -0.16583677 -0.22594728 -0.2037795  -0.23451912 -0.19590997 -0.17666594
 -0.06884229 -0.04458249  0.03861587 -0.04748505  0.05017233  0.07068622
  0.10413612  0.07025746  0.05493628 -0.02032984 -0.02640435 -0.09210102
 -0.11185708 -0.10870195 -0.13292692 -0.21194383 -0.20126559 -0.24983907
 -0.24773538 -0.21829587 -0.18061395 -0.16915039 -0.1436068 ]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,
```

Autocorrelations of average monthly Quadratic FossilOil Log



Partial Autocorrelations of average monthly Quadratic FossilOil Log



In []:

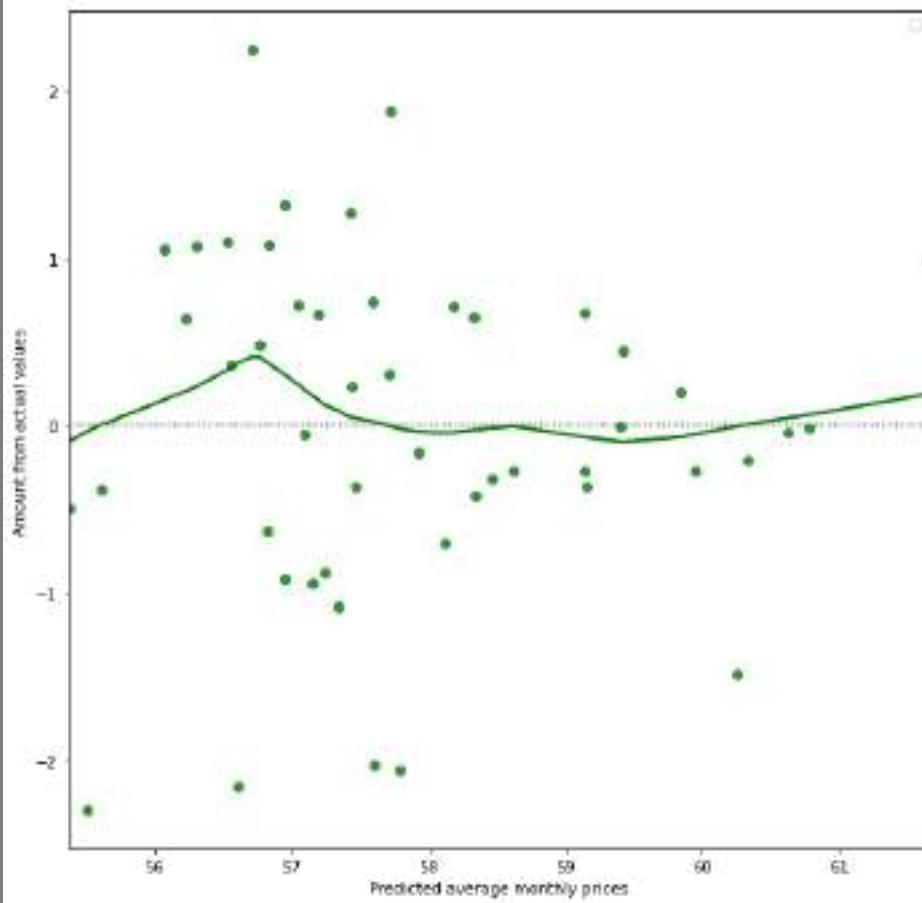
In []:

In []: # OLS Logarithmic average monthly predictions versus residuals

```
plt.suptitle("Output residuals from logarithmic model versus predicted average monthly fossil oil outputs")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("...#")
sns.residplot(x = FossilOil_Logpred, y = standardized_residualsFossilOilLog, lowess = True, color="g")
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a7eef10>
```

Output residuals from logarithmic model versus predicted average monthly fossil oil outputs



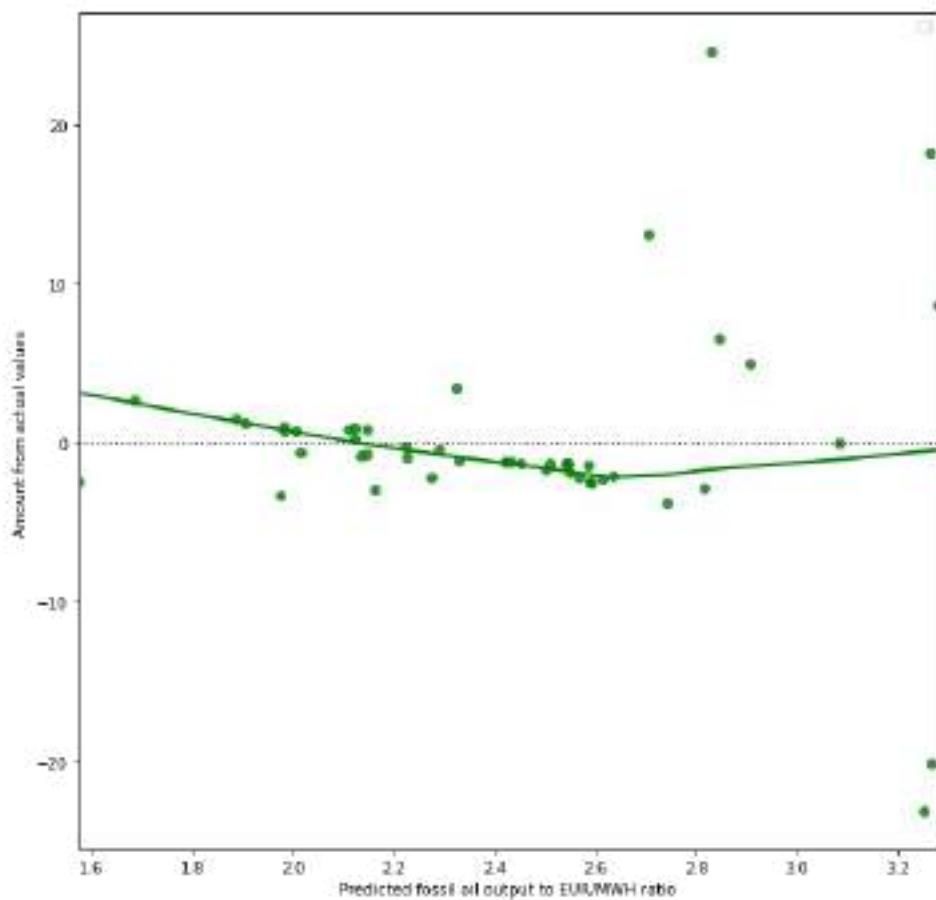
As one can observe, there is a decreasing trend in the variance of the predicted values versus the actual values. This indicates that there is heteroskedasticity but no bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: FossilOilLogRatioPredict = FossilOil_Logpred/ypred
```

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
plt.suptitle("Predicted average monthly logarithmic fossil oil output to price of energy per EUR/MWH ratios versus Predicted fossil oil output to EUR/MWH ratio")
plt.xlabel("Predicted fossil oil output to EUR/MWH ratio")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = FossilOilLogRatioPredict, y = standardized_residualsFossilOilLog/standardized_residualsPriceL)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a588ed0>
```

Predicted average monthly logarithmic fossil oil output to price of energy per EUR/MWH ratios versus respective fossil oil residuals



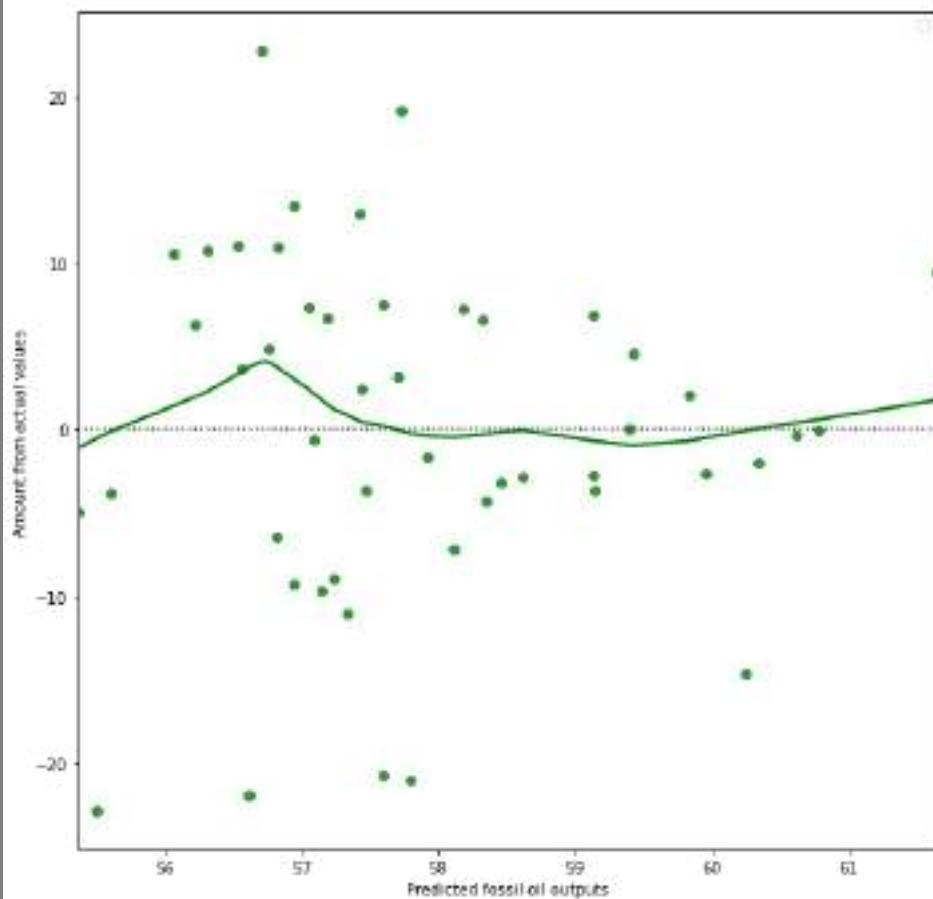
With the exception of a few heteroskedastic outliers, the predictions seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []:

```
In [ ]: plt.suptitle("Predicted average monthly logarithmic fossil oil energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted fossil oil outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = FossilOil_Logpred, y = Price_Actual , lowess = True, color="g")
# OLS predicted logarithmic average monthly values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a505a90>
```

Predicted average monthly logarithmic fossil oil energy prices per EUR/MWh versus actual values



As one can observe, there is a decreasing trend in the variance of the predicted values versus the actual values. This indicates that there is heteroskedasticity but no bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]:
```

```
In [ ]: modelFossilOil = stats.linregress([306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 64.9490188172043, 56.383854166666666, 55.522462987886975, 58.3540833333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.75208333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.4055972222224, 60.182429530201404, 62.5810555555558, 67.5951344086021, 79.4920833333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.2582222222224, 55.252580645161316, 54.08432795698931,
```

```
55.81655555555558,
63.92528859060403,
65.43065277777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881] )
```

In []:

```
In [ ]: modelFossilOil = stats.linregress([306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376,
[64.9490188172043, 56.383854166666666,
55.522462987886975,
58.35408333333333,
57.29405913978498,
65.9749027777778,
71.07204301075271,
63.99806451612899,
60.254791666666634,
59.40676510067113,
60.72679166666668,
61.901760752688226,
45.57872311827956,
36.752083333333374,
36.81800807537014,
32.61866666666666,
34.691370967741896,
46.266319444444434,
47.50201612903221,
47.6023387096774,
50.40559722222224,
60.182429530201404,
62.58105555555558,
67.5951344086021,
79.4920833333331,
59.83779761904767,
50.95989232839837,
51.71791666666662,
53.77262096774189,
56.2582222222224,
55.252580645161316,
54.08432795698931,
55.81655555555558,
63.92528859060403,
65.4306527777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881] )
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly fossil oil outputs versus the average monthly prices of energy per EUR/MWH.

In []:

```
#Linear OLS regression
fossil_oil1 = fossil_oil
fossil_oil1 = sm.add_constant(fossil_oil1)
modelFossilOilreg = sm.OLS(Price_Actual, fossil_oil1).fit()
predictionsFossilOil = modelFossilOilreg.predict(fossil_oil1)
modelFossilOilreg.summary()
#OLS Linear Summary Table
```

Out[]:

OLS Regression Results									
Dep. Variable:	y	R-squared:	0.020 <th data-cs="3" data-kind="parent"></th> <th data-kind="ghost"></th> <th data-kind="ghost"></th>						
Model:	OLS	Adj. R-squared:	-0.001 <th data-cs="3" data-kind="parent"></th> <th data-kind="ghost"></th> <th data-kind="ghost"></th>						
Method:	Least Squares	F-statistic:	0.9576						
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.333						
Time:	05:20:59	Log-Likelihood:	-179.15						
No. Observations:	48	AIC:	362.3						
Df Residuals:	46	BIC:	366.0						
Df Model:	1								
Covariance Type:	nonrobust								
	coef	std err	t	P> t	[0.025	0.975]			
const	39.5996	18.719	2.115	0.040	1.919	77.280			
x1	0.0612	0.063	0.979	0.333	-0.065	0.187			
Omnibus:	1.353	Durbin-Watson:	0.413						
Prob(Omnibus):	0.508	Jarque-Bera (JB):	0.921						
Skew:	-0.339	Prob(JB):	0.631						
Kurtosis:	3.043	Cond. No.	3.76e+03						

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 3.76e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

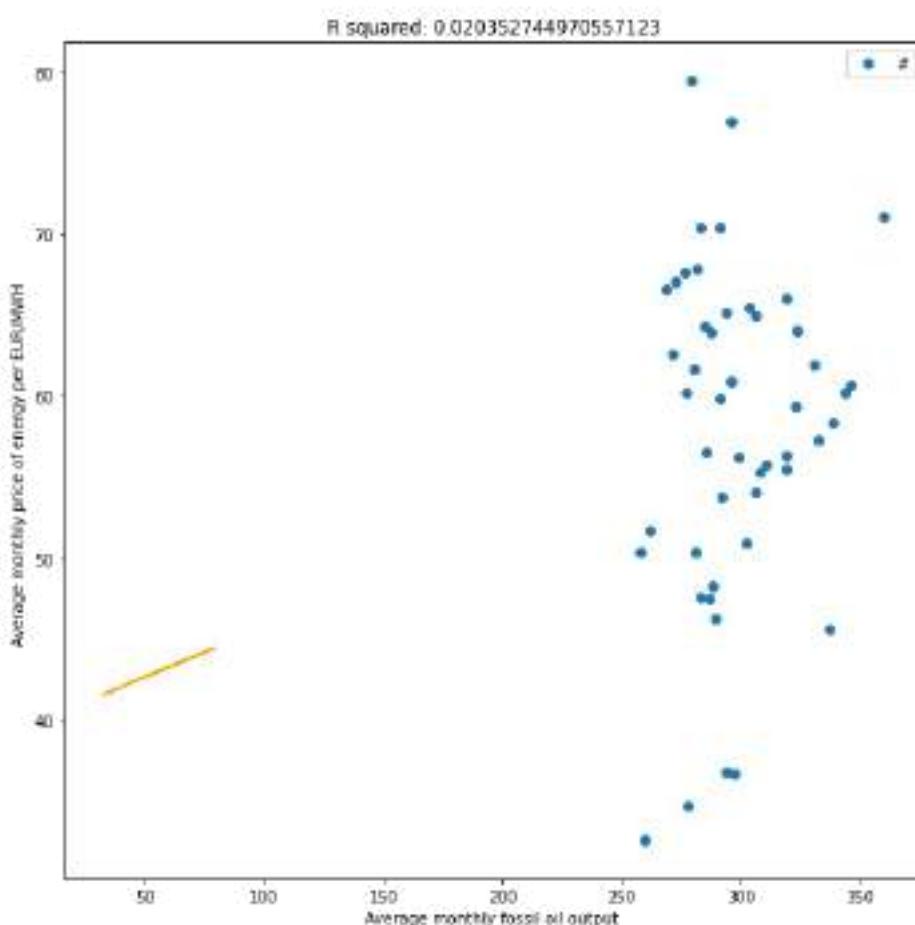
```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(fossil_oil,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(fossil_oil,Price_Actual, "o")
f = lambda x: 0.061210 *x + 39.599580
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.legend('#')
plt.title(f"R squared: {modelFossilOil.rvalue**2}")
plt.suptitle("Average monthly fossil oil outputs versus predicted average monthly prices of energy per EUR/MWH")
plt.ylabel('Average monthly price of energy per EUR/MWH ')
plt.xlabel('Average monthly fossil oil output ')

plt.show()
```

slope: 0.061210 intercept: 39.599580

Average monthly fossil oil outputs versus predicted average monthly prices of energy per EUR/MWH



There is a very weak yet positive correlation between the average monthly outputs and the respective months. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: #Predicted OLS Linear values  
print(predictionsFossilOil)
```

```
[58.33094074 59.14007647 59.14584232 60.3362175 59.95585352 59.13945838  
61.64931643 59.42619387 60.62453443 59.39419052 60.78197887 59.8385889  
60.25564845 57.79436884 57.598387 55.50716496 56.61063901 57.33987949  
57.14773431 56.94062734 56.82308425 56.55436609 56.22221097 56.53256399  
56.70903493 57.43613872 58.11590799 55.60527019 57.46790114 57.92528037  
58.46436552 58.34408555 58.62357878 57.19234165 58.17929981 57.59064923  
57.08953756 57.71149039 57.23566097 55.3689335 56.76580181 57.05168474  
56.83281512 56.94227276 57.72592442 57.42986738 56.3087544 56.06625288]
```

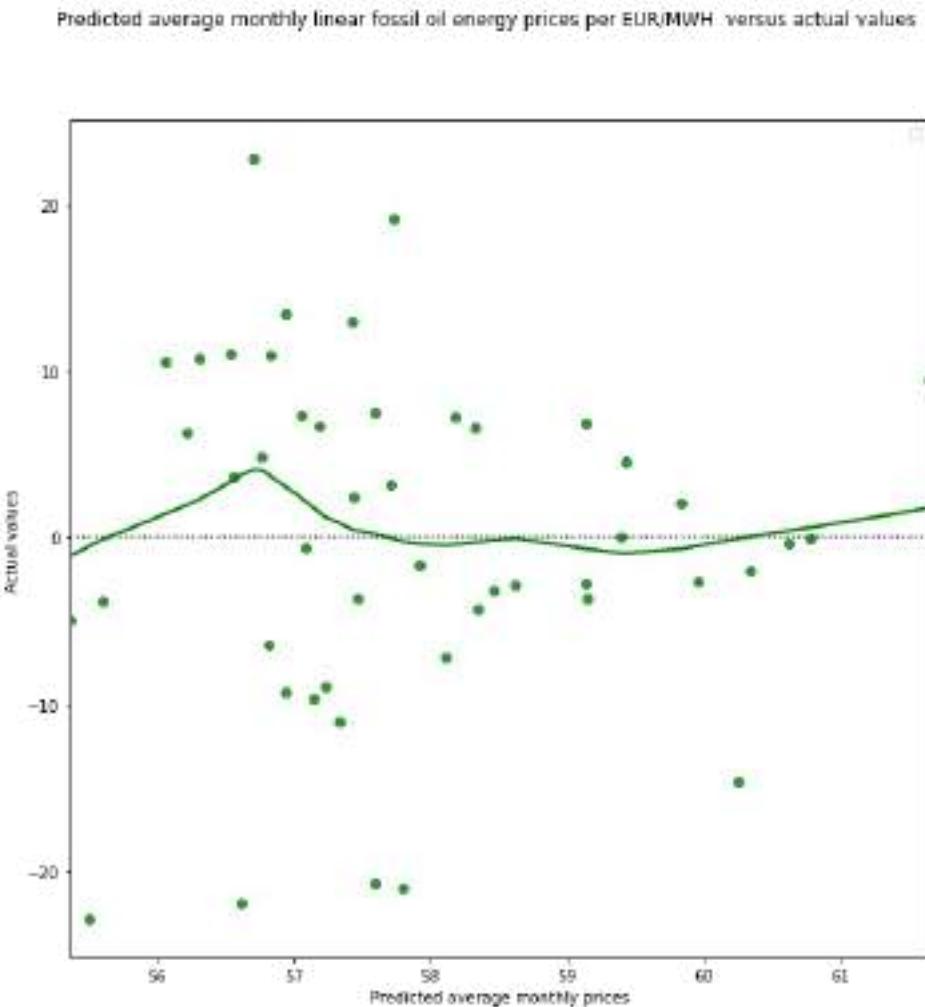
```
In [ ]: #Linear OLS regression residuals  
influenceFossilOilreg = modelFossilOilreg.get_influence()
```

```
standardized_residualsFossilOil = influenceFossilOilreg.resid_studentized_internal  
  
print(standardized_residualsFossilOil)
```

```
[ 6.48464911e-01 -2.72004425e-01 -3.57608810e-01 -2.00240571e-01  
-2.66443802e-01  6.74566839e-01  9.96604805e-01  4.53066360e-01  
-3.76568603e-02  1.24549118e-03 -5.64805685e-03  2.06007759e-01  
-1.47961111e+00 -2.05955884e+00 -2.03457635e+00 -2.30493392e+00  
-2.16229313e+00 -1.08529385e+00 -9.46478822e-01 -9.17872866e-01  
-6.31516208e-01  3.58163113e-01  6.30892475e-01  1.09241818e+00  
2.24481145e+00  2.35273838e-01 -7.00625171e-01 -3.90533514e-01  
-3.61953943e-01 -1.63166888e-01 -3.14929761e-01 -4.17413817e-01  
-2.75542580e-01  6.60461039e-01  7.10089047e-01  7.40265153e-01  
-5.66974068e-02  3.09868127e-01 -8.78276584e-01 -5.02094124e-01  
4.79330244e-01  7.16485215e-01  1.07749272e+00  1.31921067e+00  
1.87820637e+00  1.26692622e+00  1.06345147e+00  1.05033093e+00]
```

```
In [ ]: plt.suptitle("Predicted average monthly linear fossil oil energy prices per EUR/MWH versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend("#")  
sns.residplot(x = predictionsFossilOil, y = Price_Actual, lowess = True, color="g")  
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a4a23d0>
```



As one can observe, there is a decreasing trend in the variance of the predicted values versus the actual values. This indicates that there is heteroskedasticity but no bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly fossil oil outputs versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression  
from sklearn.preprocessing import PolynomialFeatures  
polynomial_features = PolynomialFeatures(degree=3)
```

```

modelFossilOilquad = np.poly1d(np.polyfit(fossil_oil, Price_Actual, 2))
print(modelFossilOilquad)

fossil_oil1 = fossil_oil

fossil_oil1 = sm.add_constant(fossil_oil1)
fossil_oil2 = polynomial_features.fit_transform(fossil_oil1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(fossil_oil1)

FossilOil_Q = poly.fit(X_poly, fossil_oil)
lin2 = LinearRegression()
lin2.fit(X_poly, fossil_oil)
FossilOil_Quad = sm.OLS(Price_Actual, fossil_oil2).fit()

# OLS Predicted Quadratic values
FossilOil_ypred = FossilOil_Quad.predict(fossil_oil2)

#OLS Quadratic Summary Table
FossilOil_Quad.summary()

```

2
-0.0003178 x + 0.2555 x + 10.1

Out[]: OLS Regression Results

Dep. Variable:	y	R-squared:	0.083		
Model:	OLS	Adj. R-squared:	0.021		
Method:	Least Squares	F-statistic:	1.332		
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.276		
Time:	05:20:59	Log-Likelihood:	-177.56		
No. Observations:	48	AIC:	363.1		
Df Residuals:	44	BIC:	370.6		
Df Model:	3				
Covariance Type:	nonrobust				
coef	std err	t	P> t	[0.025	0.975]
const	-889.6660	517.985	-1.718	0.093	-1933.595 154.263
x1	-889.6660	517.985	-1.718	0.093	-1933.595 154.263
x2	11.8719	6.824	1.740	0.089	-1.881 25.625
x3	-889.6660	517.985	-1.718	0.093	-1933.595 154.263
x4	11.8719	6.824	1.740	0.089	-1.881 25.625
x5	-0.0583	0.034	-1.735	0.090	-0.126 0.009
x6	-889.6660	517.985	-1.718	0.093	-1933.595 154.263
x7	11.8719	6.824	1.740	0.089	-1.881 25.625
x8	-0.0583	0.034	-1.735	0.090	-0.126 0.009
x9	0.0001	7.32e-05	1.731	0.090	-2.08e-05 0.000
Omnibus:	2.855	Durbin-Watson:	0.522		
Prob(Omnibus):	0.240	Jarque-Bera (JB):	2.154		
Skew:	-0.514	Prob(JB):	0.341		
Kurtosis:	3.137	Cond. No.	3.71e+37		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 2.71e-59. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []: ##OLS Quadratic Scatterplot

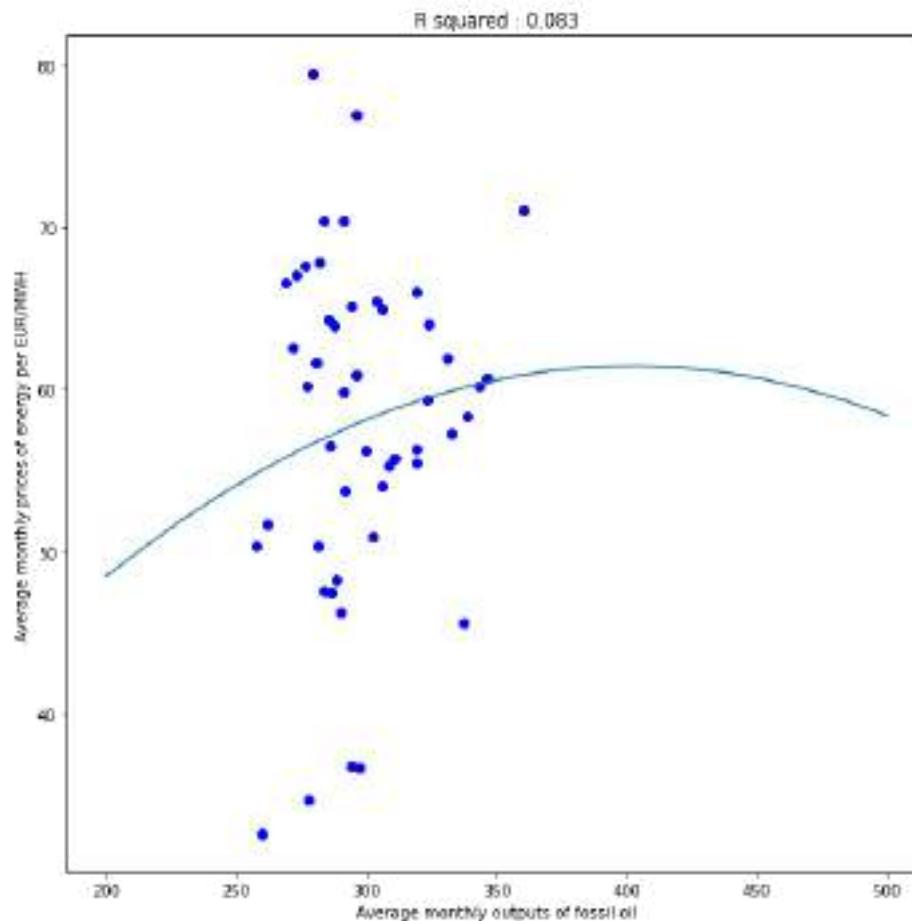
```

polyline = np.linspace(start = 200, stop = 500 , num = 20)
plt.plot(polyline, modelFossilOilquad(polyline))
plt.scatter(fossil_oil,Price_Actual, color = 'blue')
plt.title("R squared : 0.083")
plt.suptitle('Quadratic for average monthly outputs of fossil oil versus predicted average monthly prices of energy')
plt.ylabel('Average monthly prices of energy per EUR/MWh')

```

```
plt.xlabel('Average monthly outputs of fossil oil')
plt.show()
```

Quadratic for average monthly outputs of fossil oil versus predicted average monthly prices of energy per EUR/MWH



The blue dots represent the observations and the blue line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: influenceFossilOilQuad = FossilOil_Quad.get_influence()
#Logarithmic OLS regression residuals
```

```
standardized_residualsFossilOilQuad = influenceFossilOilQuad.resid_studentized_internal
```

```
print(standardized_residualsFossilOilQuad)
```

```
[ 0.71883581 -0.00966458 -0.09664904  0.02042371  0.04246333  0.96459529
 0.02189838  0.78291168  0.0536967  0.31337593 -0.01481201  0.53955002
-1.2637671 -2.18831435 -2.20447283 -1.9921564 -2.33944951 -1.2769549
-1.15056615 -1.12054539 -0.82009132  0.23278399  0.63162634  0.9838309
 2.11698304  0.08097297 -0.72006639  0.04399874 -0.52271298 -0.22300626
-0.22670095 -0.36619392 -0.14095325  0.48787061  0.73733629  0.62179431
-0.24660447  0.20800309 -1.075986  0.22786767  0.31551733  0.53973657
 0.91586073  1.15451558  1.80966679  1.13114609  1.03175513  1.14941123]
```

```
In [ ]: print(FossilOil_ypred) # OLS quadratic predicted values
```

```
[57.76733268 56.4789898 56.4737367 58.15671299 56.8812611 56.47955798
70.9593487 56.33389956 59.74259823 56.33729429 60.86601843 56.65009444
57.8139044 58.68617772 58.91233553 50.21801966 58.04965378 59.05675515
59.01974913 58.81197102 58.60567134 57.85964143 56.33064954 57.78094041
58.33793264 59.02652586 58.16850564 51.3164425 59.01012722 58.49289521
57.51346653 57.74229099 57.21864497 59.04078779 58.05277555 58.91941834
58.980158 58.79217109 59.05379563 48.51471951 58.47982965 58.946745
58.6253621 58.81438244 58.77473301 59.02939873 56.79970016 55.34944541]
```

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilOil_Logpred)
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilOilRatio Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
```

```

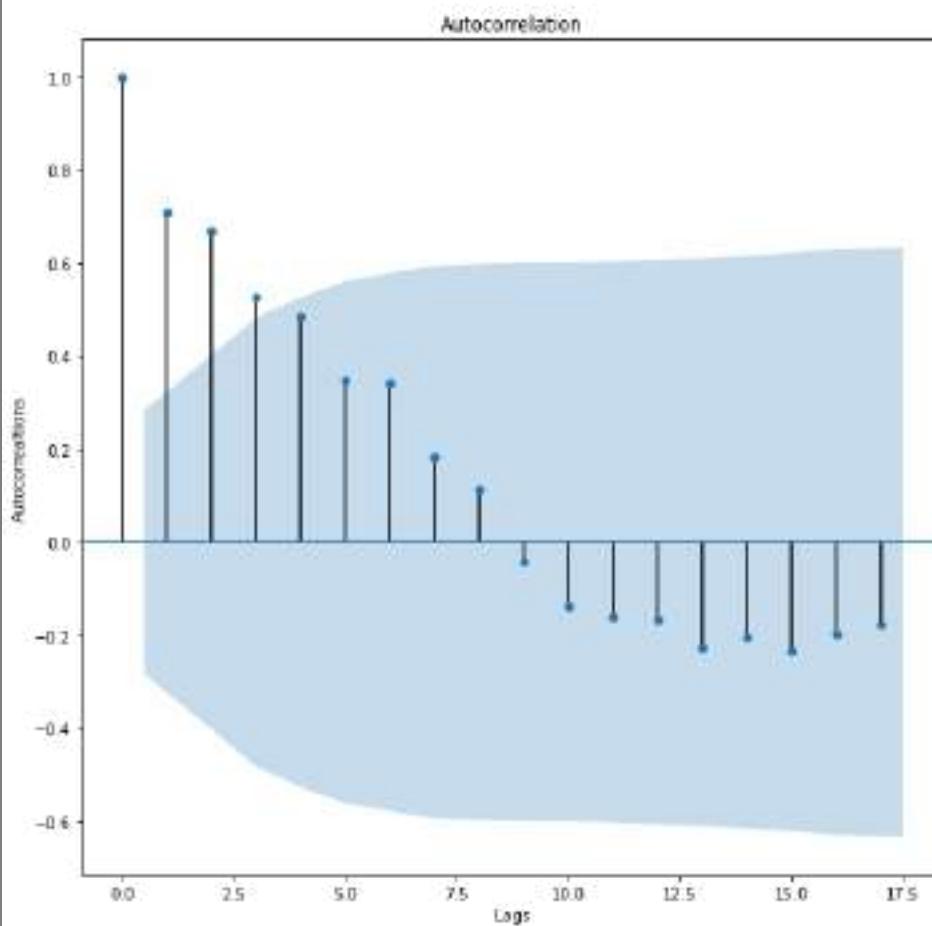
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilOil_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilOil Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Fossiloil_Log_Autocorrelations = sm.tsa.acf(FossilOil_Logpred, fft=False) #Autocorrelations
print(FossilOil_Log_Autocorrelations)

```

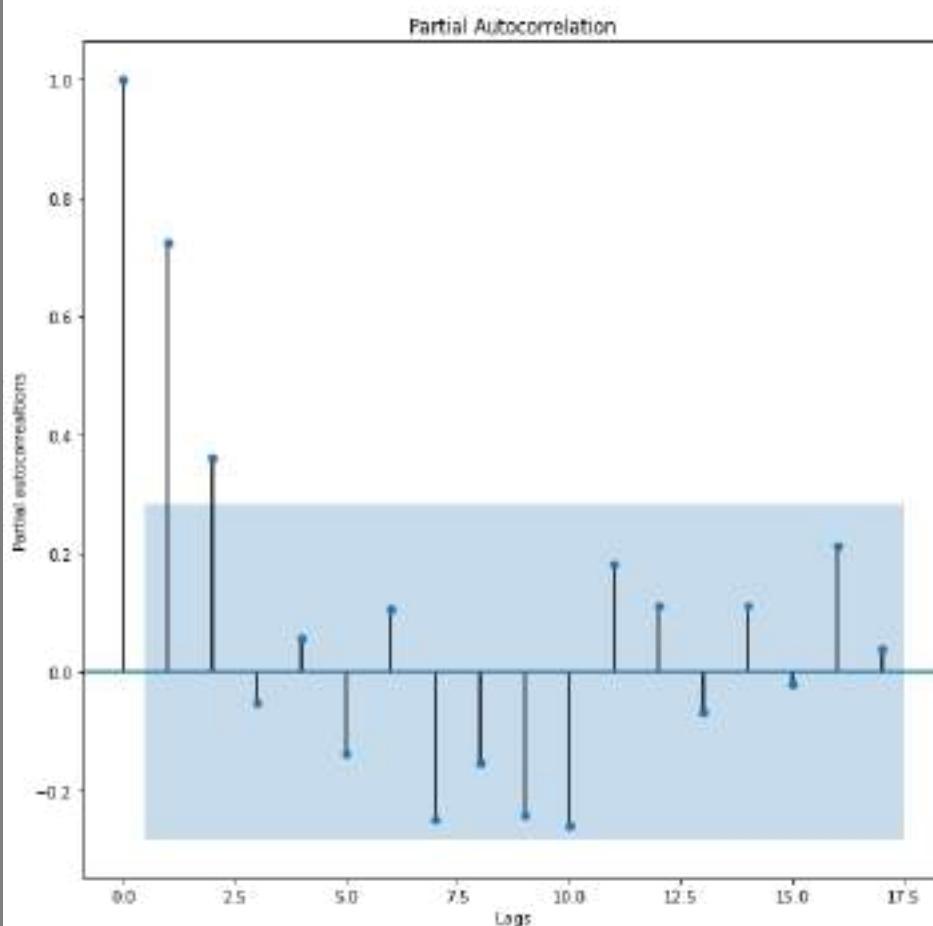
```
[ 1.          0.70952824  0.66802752  0.52851278  0.48281801  0.3500967
  0.3432444   0.18192468  0.11216314 -0.04078178 -0.13797814 -0.16229098
 -0.16583677 -0.22594728 -0.2037795  -0.23451912 -0.19590997 -0.17666594
 -0.06884229 -0.04458249  0.03861587 -0.04748505  0.05017233  0.07068622
  0.10413612  0.07025746  0.05493628 -0.02032984 -0.02640435 -0.09210102
 -0.11185708 -0.10870195 -0.13292692 -0.21194383 -0.20126559 -0.24983907
 -0.24773538 -0.21829587 -0.18061395 -0.16915039 -0.1436068 ]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

Autocorrelations of average monthly Quadratic FossilOilRatio Log



Partial Autocorrelations of average monthly Quadratic FossilOil Log

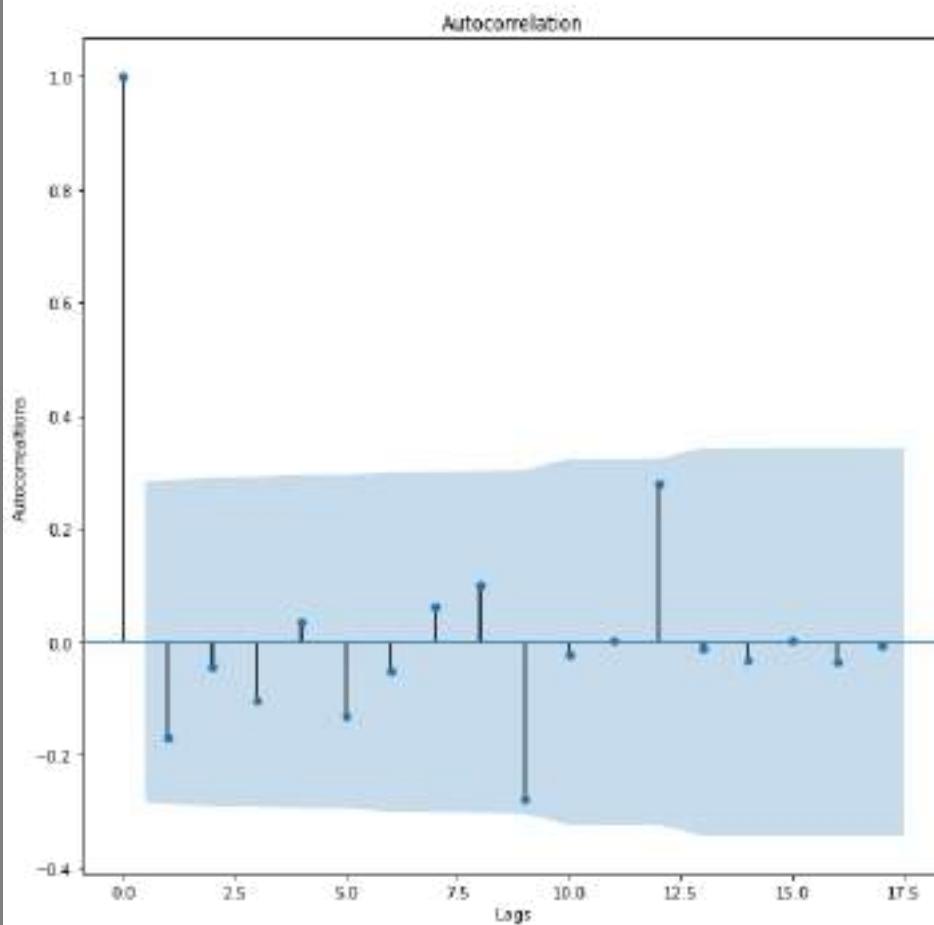


```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilOil_ypred)
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilOil")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilOil_ypred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilOil")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
FossilOil_Quad_Autocorrelations = sm.tsa.acf(FossilOil_ypred, fft=False) #Autocorrelations
print(FossilOil_Quad_Autocorrelations)
```

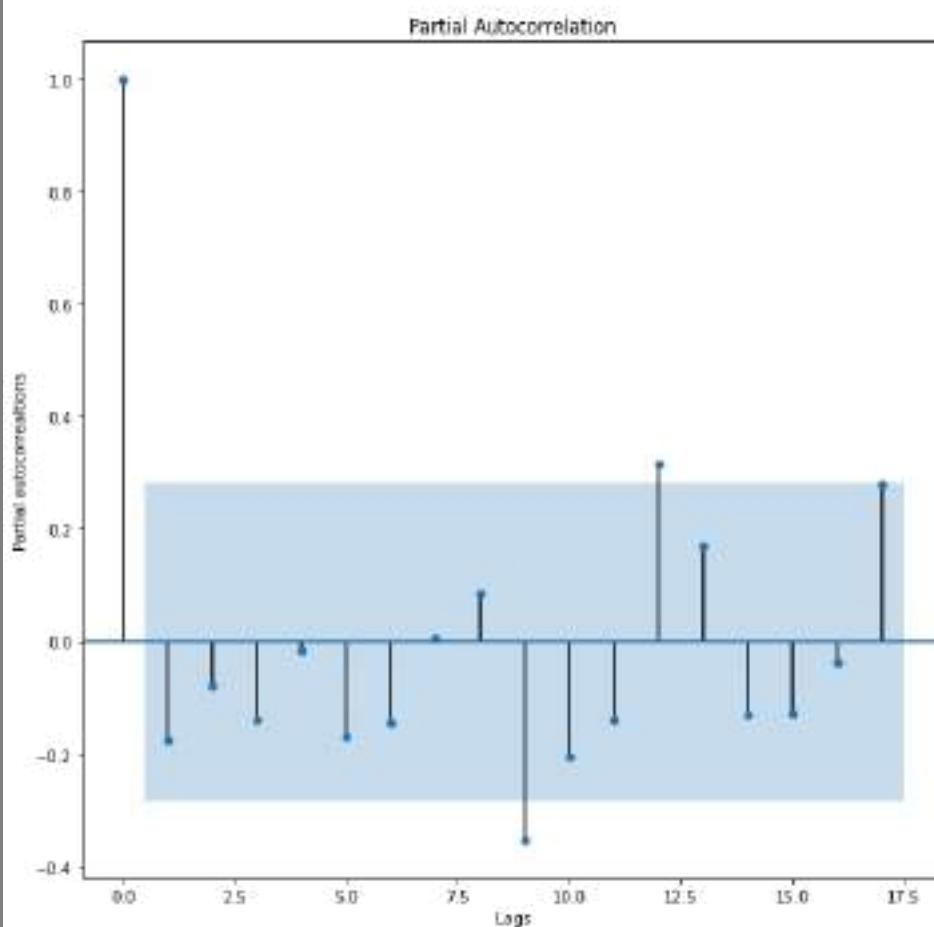
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.0000000e+00 -1.69774560e-01 -4.42600488e-02 -1.04701236e-01
 3.44679492e-02 -1.30927697e-01 -5.08113849e-02  6.32138162e-02
 1.00079245e-01 -2.76620993e-01 -2.29045509e-02  2.62990601e-03
 2.80508694e-01 -1.06618714e-02 -3.33242164e-02  2.26397838e-03
 -3.45101567e-02 -5.72346039e-03  4.17032032e-03  2.77039109e-03
 4.39664758e-02 -2.46388762e-01  7.12110106e-04  1.68645991e-02
 1.55992979e-01  1.46753200e-02 -3.32732037e-02  2.84229929e-02
 1.74941547e-02 -6.50876896e-02  4.71068581e-02  1.18850035e-03
 1.09514870e-01 -3.07981927e-01  3.53516246e-02  5.78087827e-02
 1.04585210e-03  3.99712737e-02  5.33042979e-02  2.09796439e-02
 -3.57548479e-02]
```

Autocorrelations of average monthly Quadratic Fossiloil



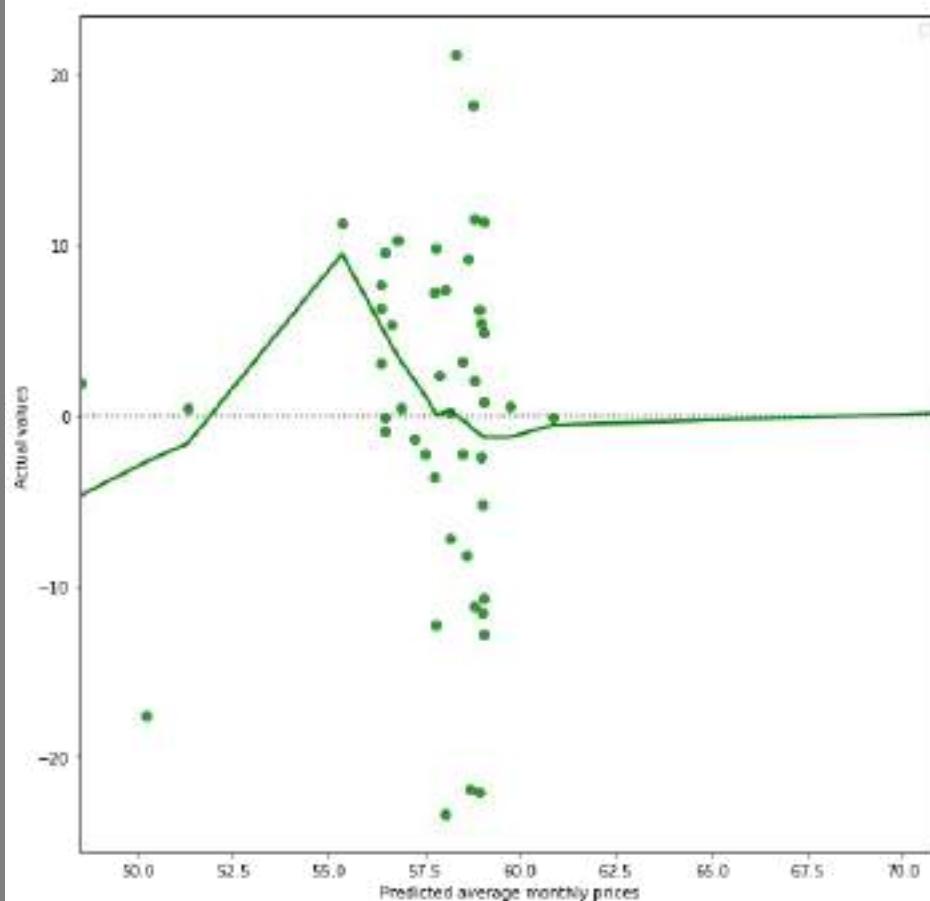
Partial Autocorrelations of average monthly Quadratic FossilOil



```
In [ ]: plt.suptitle("Predicted average monthly quadratic fossil oil energy prices per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = FossilOil_ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a25ba50>
```

Predicted average monthly quadratic fossil oil energy prices per EUR/MWH versus actual values



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []:

In []:

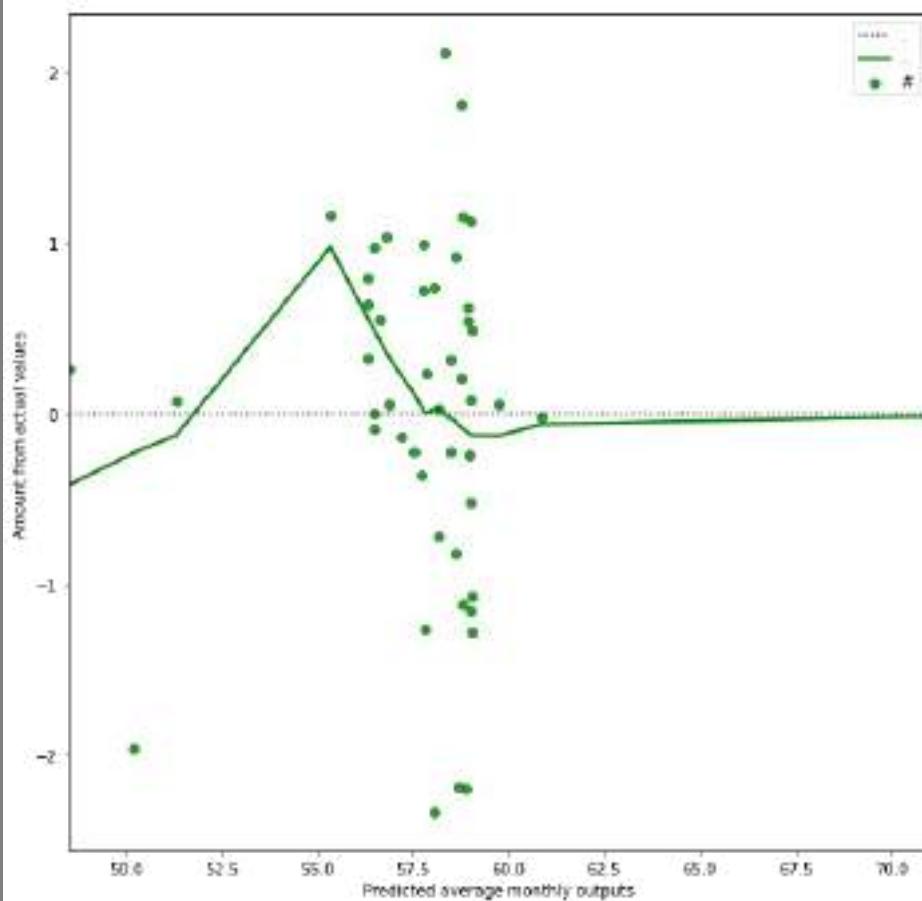
```
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = FossilOil_ypred, y = standardized_residualsFossilOilQuad, lowess = True, color="g")

plt.suptitle("Fossil oil output residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(". #")
```

Out[]: <matplotlib.legend.Legend at 0x7ff60a10f850>

Fossil oil output residuals from quadratic model versus predicted values



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: modelFossilOilregpredictions = stats.linregress([predictionsFossilOil],[fossil_oil])
```

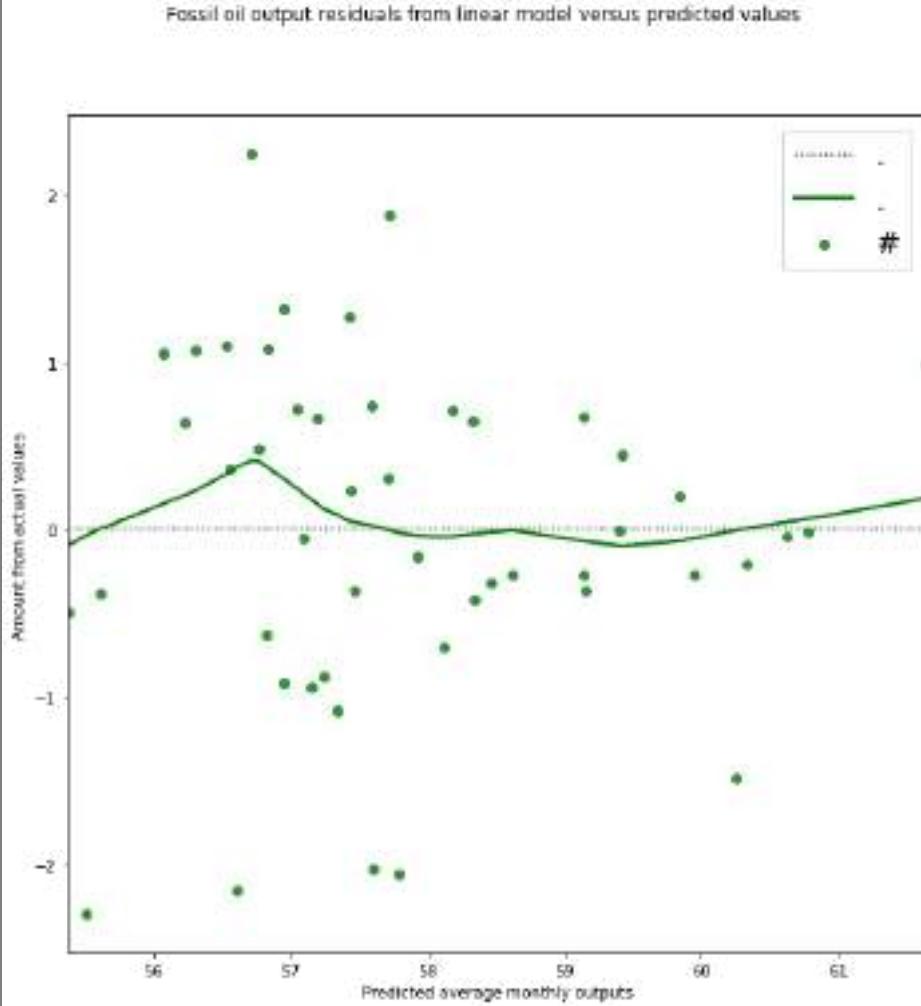
```
/usr/local/lib/python3.7/dist-packages/scipy/stats/_stats_mstats_common.py:184: RuntimeWarning: invalid value encountered in sqrt
t = r * np.sqrt(df / ((1.0 - r + TINY)*(1.0 + r + TINY)))
```

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionsFossilOil, y = standardized_residualsFossilOil, lowess = True, color="g")
plt.rcParams["figure.figsize"] = [20, 10]

plt.suptitle("Fossil oil output residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.rcParams.update({'font.size': 19})
modelFossilOilregpredictions = stats.linregress([predictionsFossilOil],[fossil_oil])
```

```
plt.legend("..#")  
  
/usr/local/lib/python3.7/dist-packages/scipy/stats/_stats_mstats_common.py:184: RuntimeWarning: invalid value encountered in sqrt  
t = r * np.sqrt(df / ((1.0 - r + TINY)*(1.0 + r + TINY)))
```

Out[]: <matplotlib.legend.Legend at 0x7ff60a0b5a50>



Due to the sudden and punctual hump in the lowess line, there is indication that there is heteroskedasticity in this plot. However, there is a lack of bias in this plot since there is a decreasing trend in the variance. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, then seasonality and trends are indeed factors after all.

```
In [ ]:  
#Dataframes analyzed by resource  
dfFossilOil = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.2940591,  
"Fossil_Oil"] : [306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 332.56720430107520],  
print(dfFossilOil)  
df_FossilOil= pd.DataFrame.from_dict(dfFossilOil, orient = "columns")  
print(df_FossilOil)  
df_FossilOil["Ratio"] = df_FossilOil["Fossil_Oil"]/df_FossilOil["Price"]  
pdToListFossilOil = list(df_FossilOil["Ratio"])  
  
print(pdToListFossilOil)  
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value))  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis, indicating it is non-stationary ")  
  
adfuller_test(df_FossilOil["Ratio"])
```

```

test_result=adfuller(df_FossilOil["Ratio"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilOil["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Fossil Oil Ratios")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show

from statsmodels.graphics.tsaplots import plot_pacf # Partialautocorrelation Plot
plot_pacf(df_FossilOil["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Fossil Oil Ratios")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.2582222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Oil': [306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963839, 360.23387096774195, 323.9139784946237, 343.4916666666667, 323.39112903225805, 346.06388888888887, 330.6514131897712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875, 277.9153225806452, 289.82916666666665, 286.6900269541779, 283.30645161290323, 281.38611111111111, 276.9959731543624, 271.56944444444446, 276.63978494623655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.4902777777778, 291.9206989247312, 299.3930555555556, 308.2002688172043, 306.23521505376345, 310.801388888889, 287.4187919463087, 303.54305555555555, 293.9260752688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916666666666, 280.4502688172043, 285.12083333333334, 281.5450874831763, 283.3333333333333, 296.13611111111111, 291.2993288590604, 269.02150537634407, 272.98333333333335], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

	Price	Fossil_Oil	Dates
0	64.949019	306.020464	2015-01
1	56.383854	319.239583	2015-02
2	55.522463	319.333782	2015-03
3	58.354083	338.781337	2015-04
4	57.294059	332.567204	2015-05
5	65.974903	319.229485	2015-06
6	71.072043	360.233871	2015-07
7	63.998065	323.913978	2015-08
8	60.254792	343.491667	2015-09
9	59.406765	323.391129	2015-10
10	60.726792	346.063889	2015-11
11	61.901761	330.651413	2015-12
12	45.578723	337.465054	2016-01
13	36.752083	297.254310	2016-02
14	36.818008	294.052490	2016-03
15	32.618667	259.887500	2016-04
16	34.691371	277.915323	2016-05
17	46.266319	289.829167	2016-06
18	47.502016	286.690027	2016-07
19	47.602339	283.306452	2016-08
20	50.405597	281.386111	2016-09
21	60.182430	276.995973	2016-10
22	62.581056	271.569444	2016-11
23	67.595134	276.639785	2016-12
24	79.492083	279.522849	2017-01
25	59.837798	291.401786	2017-02
26	50.959892	302.507402	2017-03
27	51.717917	261.490278	2017-04
28	53.772621	291.920699	2017-05
29	56.258222	299.393056	2017-06
30	55.252581	308.200269	2017-07
31	54.084328	306.235215	2017-08
32	55.816556	310.801389	2017-09
33	63.925289	287.418792	2017-10
34	65.430653	303.543056	2017-11
35	65.151277	293.926075	2017-12
36	56.511976	285.739247	2018-01
37	60.877098	295.900298	2018-02
38	48.279717	288.126514	2018-03
39	50.400736	257.629167	2018-04
40	61.633763	280.450269	2018-05
41	64.348139	285.120833	2018-06
42	67.783441	281.545087	2018-07
43	70.363911	283.333333	2018-08

```
44 76.914042 296.136111 2018-09
45 70.362215 291.299329 2018-10
46 67.042608 269.021505 2018-11
47 66.623514 272.983333 2018-12
[4.711702646478498, 5.661897152147204, 5.751434010315323, 5.805614923503275, 5.804566988170348, 4.83865033453159
6, 5.068573460217559, 5.061308977758065, 5.700653129246291, 5.443675118216539, 5.6987019961215175, 5.34155101840
8021, 7.4040041202492315, 8.088094153705416, 7.986648525467014, 7.967447065075214, 8.01107926345912, 6.264366177
099672, 6.035323346601258, 5.951523796777401, 5.582437796948807, 4.6026053669926625, 4.339483283457277, 4.092599
8080785782, 3.516360846780745, 4.869861480689363, 5.9361860592871185, 5.056086838592902, 5.42879803273591, 5.3217
65312329655, 5.578024867227532, 5.662180277016627, 5.568265289669134, 4.4961672959667265, 4.6391567662709905, 4.
511439979947413, 5.056260079995415, 4.8606176427379415, 5.967858344556159, 5.11161515773717, 4.550270065632229,
4.4309103302219315, 4.153596865402372, 4.026685386551291, 3.8502216850665367, 4.139996585263025, 4.0126945430706
27, 4.097402214308303]
```

ADF Test Statistic : -1.959432227746659

p-value : 0.3046388732428561

#Lags Used : 0

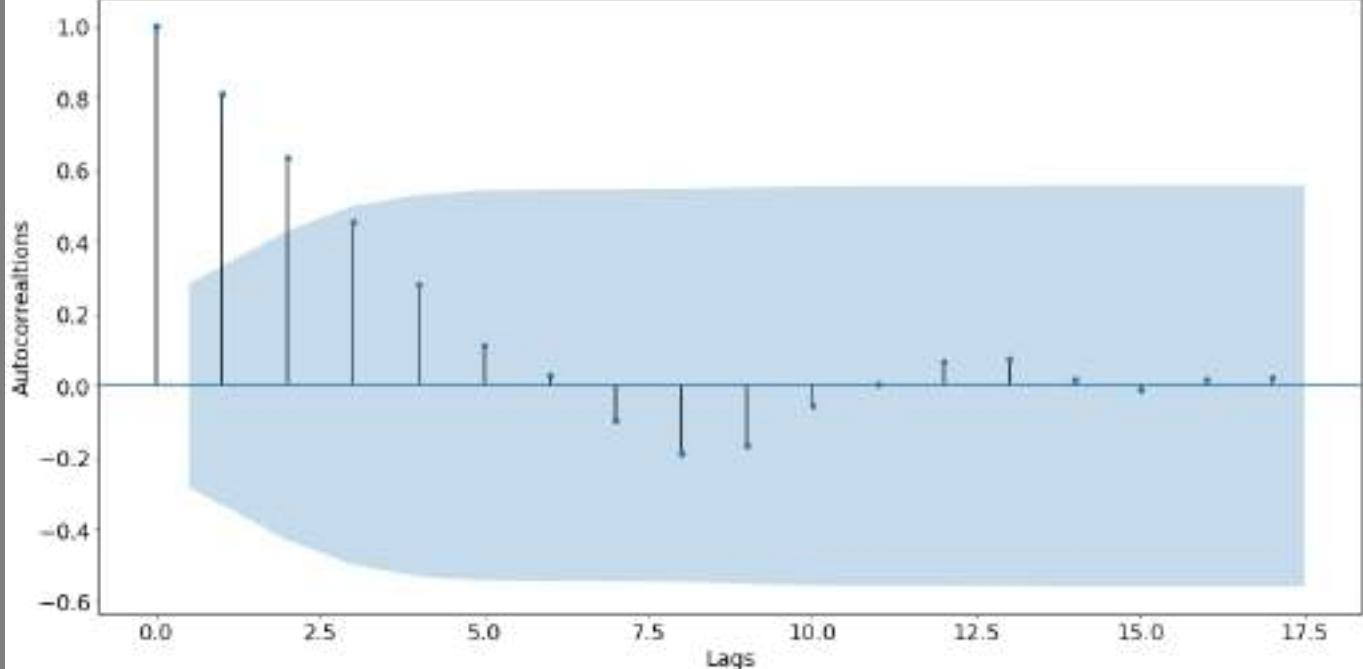
Number of Observations : 47

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <function matplotlib.pyplot.show(*args, **kw)>

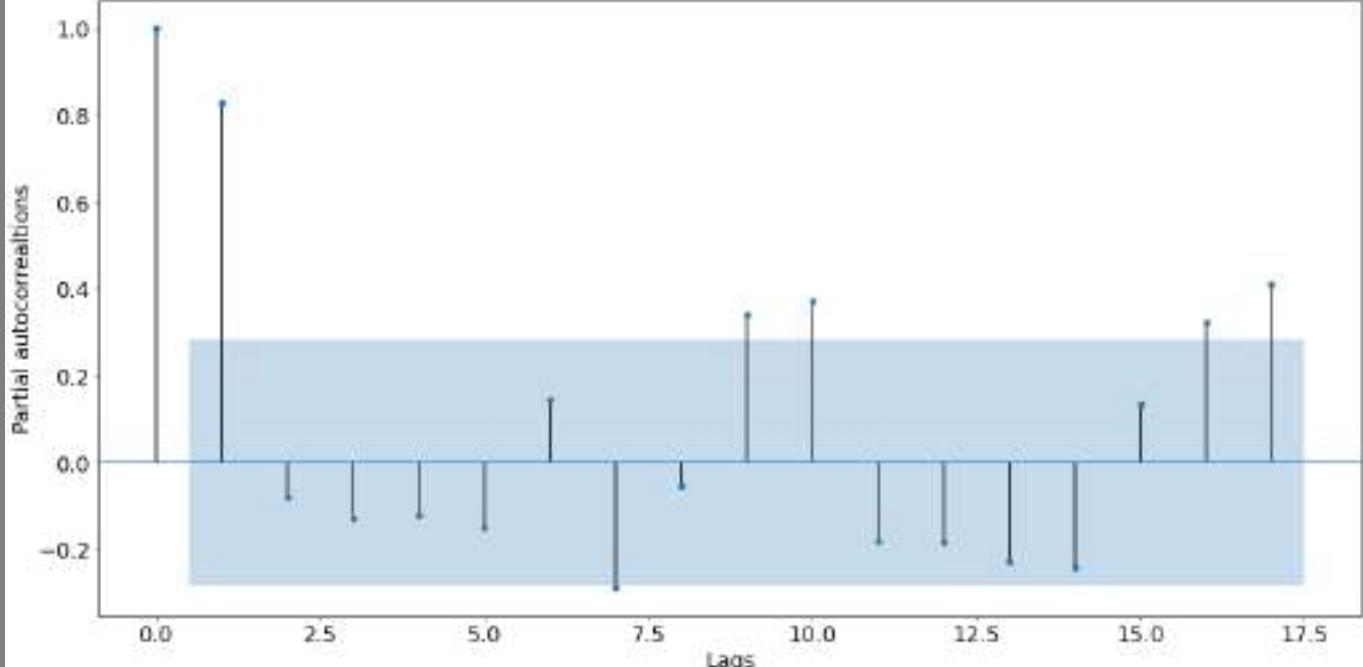
Autocorrelations of average monthly Fossil Oil Ratios

Autocorrelation



Autocorrelations of average monthly Fossil Oil Ratios

Partial Autocorrelation



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each

spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly fossil oil outputs and the average monthly prices of energy per EUR/MWH.

In []:

```
In [ ]: Fossil_Oil_Ratio_Autocorrelations = sm.tsa.acf(df_FossilOil["Ratio"], fft=False) #Autocorrelations
print(Fossil_Oil_Ratio_Autocorrelations)
```

```
[ 1.          0.81081747  0.63316925  0.45361825  0.28114559  0.11184454
 0.02962111 -0.09649995 -0.18764493 -0.16552601 -0.05508444  0.00235592
 0.06637247  0.07307397  0.01661988 -0.01278033  0.0154729   0.02281436
 0.02625199 -0.00942336 -0.04532448 -0.06266455  0.01955626  0.016538
 -0.02197949 -0.1016855  -0.17447587 -0.25393697 -0.28078036 -0.30058861
 -0.310156  -0.3170961  -0.25995126 -0.18623869 -0.1136211  -0.03931396
 -0.00407666 -0.00134708 -0.01769854 -0.01215895 -0.00573077]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,

```
In [ ]: df_FossilOil['First Difference Fossil Oil Ratio'] = df_FossilOil["Ratio"]- df_FossilOil["Ratio"].shift(1) # Seasonal difference
df_FossilOil['Seasonal Difference Fossil Oil Ratio']=df_FossilOil["Ratio"]- df_FossilOil["Ratio"].shift(12)
df_FossilOil.head()
```

	Price	Fossil_Oil	Dates	Ratio	First Difference Fossil Oil Ratio	Seasonal Difference Fossil Oil Ratio
0	64.949019	306.020464	2015-01	4.711703		NaN
1	56.383854	319.239583	2015-02	5.661897		0.950195
2	55.522463	319.333782	2015-03	5.751434		0.089537
3	58.354083	338.781337	2015-04	5.805615		0.054181
4	57.294059	332.567204	2015-05	5.804567		-0.001048

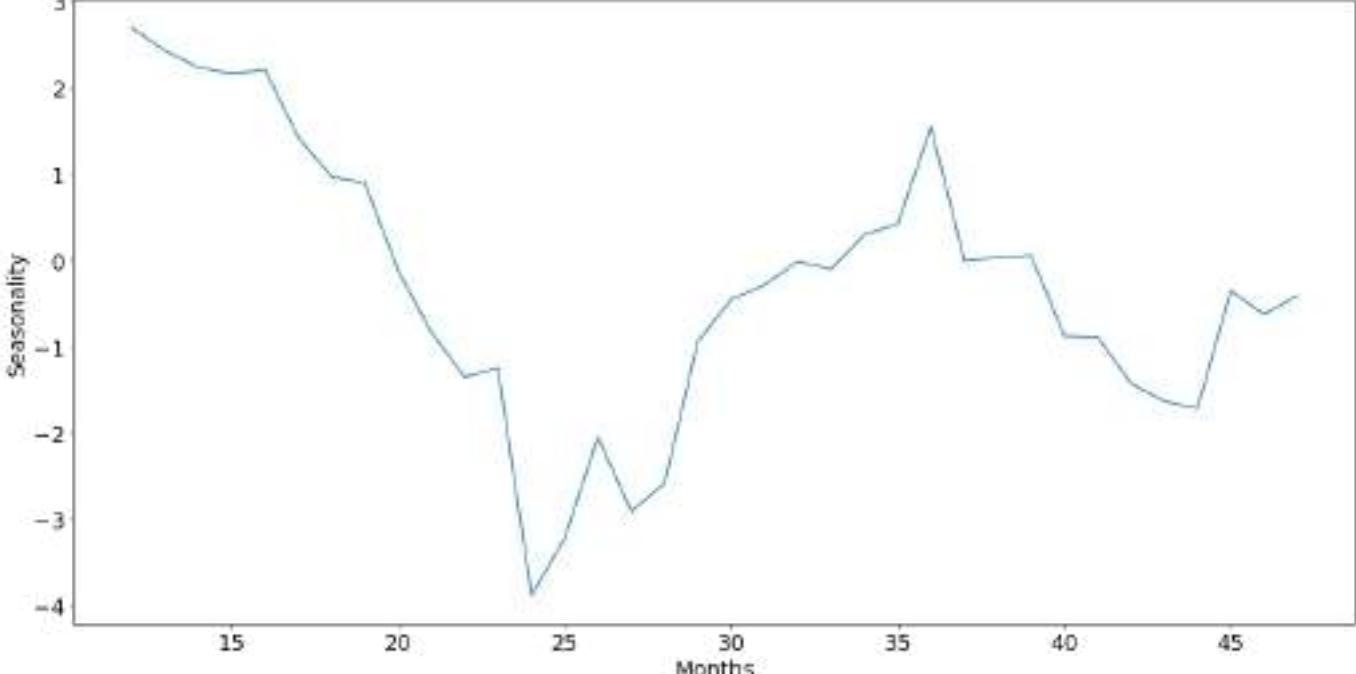
```
In [ ]: plt.suptitle("Seasonal Difference of average monthly fossil oil outputs to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilOil['Seasonal Difference Fossil Oil Ratio'].plot()

#Seasonality Plot
```

Out[]:

<matplotlib.axes._subplots.AxesSubplot at 0x7ff609e96890>

Seasonal Difference of average monthly fossil oil outputs to price of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted

between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

In []:

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

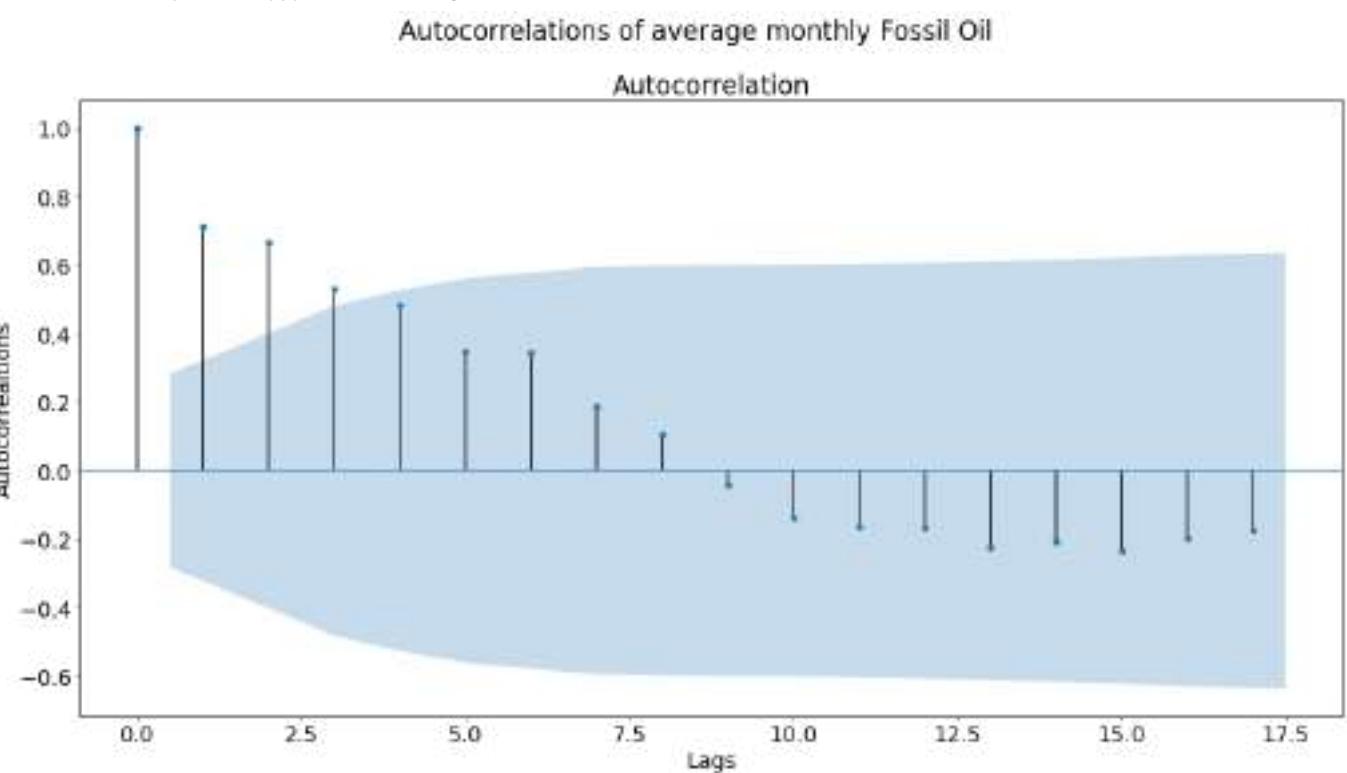
adfuller_test(df_FossilOil["Fossil_Oil"])

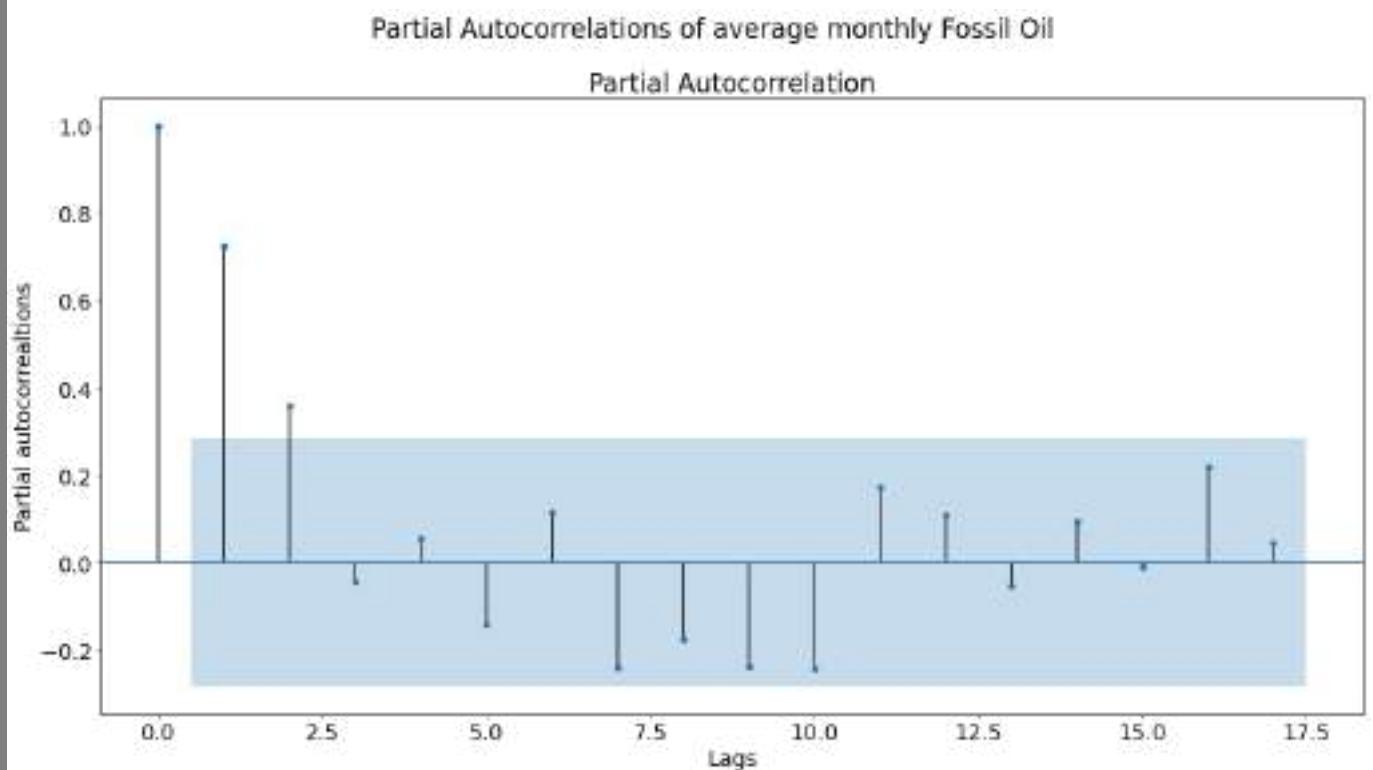
test_result=adfuller(df_FossilOil["Fossil_Oil"])


from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilOil["Fossil_Oil"])
plt.suptitle(" Autocorrelations of average monthly Fossil Oil")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_FossilOil["Fossil_Oil"])
plt.suptitle("Partial Autocorrelations of average monthly Fossil Oil")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
plt.show
```

ADF Test Statistic : -2.598946146304935
p-value : 0.09321527324439388
#Lags Used : 0
Number of Observations : 47
weak evidence against null hypothesis, indicating it is non-stationary

Out[]:





The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly fossil oil outputs and the average monthly prices of energy per EUR/MWH.

In []:

[]

[]

In []: df_FossilOil['First Difference'] = df_FossilOil["Fossil_Oil"] - df_FossilOil["Fossil_Oil"].shift(1) # Seasonality
df_FossilOil['Seasonal Difference']=df_FossilOil["Fossil_Oil"]- df_FossilOil["Fossil_Oil"].shift(12)
df_FossilOil.head()

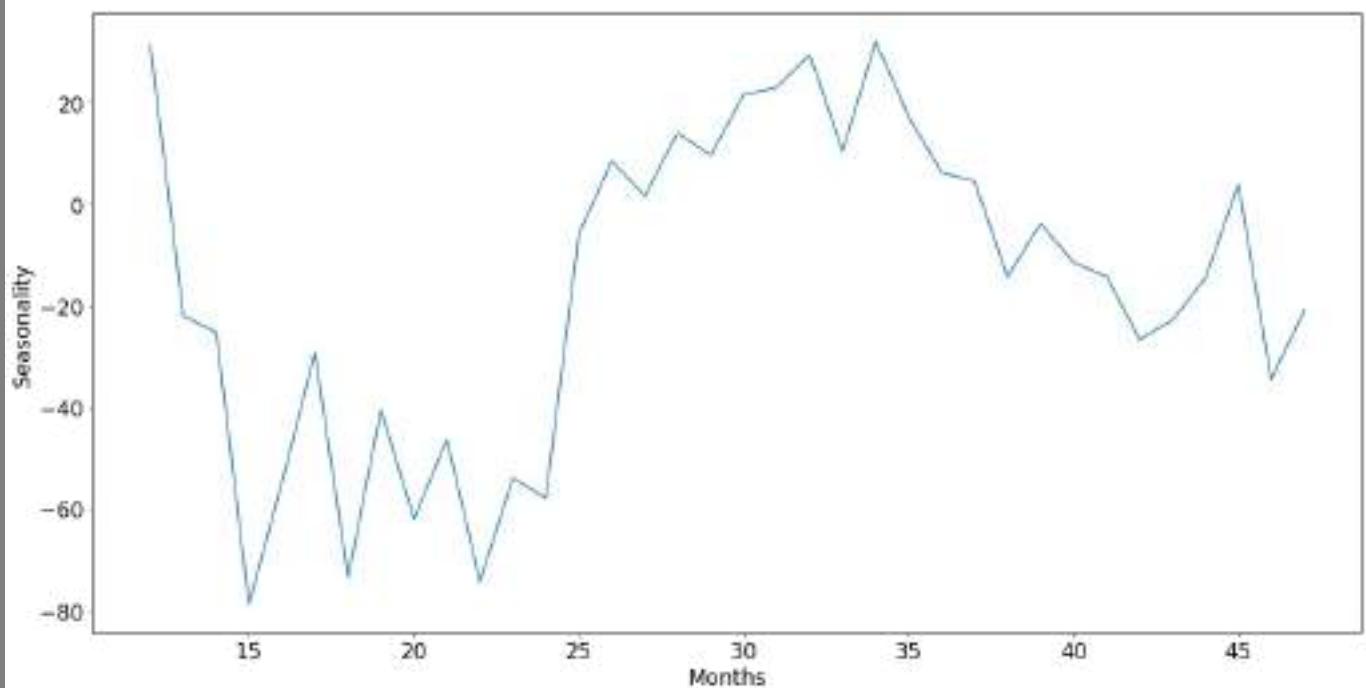
Out[]:

	Price	Fossil_Oil	Dates	Ratio	First Difference Fossil Oil Ratio	Seasonal Difference Fossil Oil Ratio	First Difference	Seasonal Difference
0	64.949019	306.020464	2015-01	4.711703		NaN	NaN	NaN
1	56.383854	319.239583	2015-02	5.661897		0.950195	NaN	13.219119
2	55.522463	319.333782	2015-03	5.751434		0.089537	NaN	0.094199
3	58.354083	338.781337	2015-04	5.805615		0.054181	NaN	19.447555
4	57.294059	332.567204	2015-05	5.804567	-0.001048		NaN	-6.214133

In []: plt.suptitle("Seasonal Difference of average monthly fossil oil outputs")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilOil['Seasonal Difference'].plot() # Seasonality Plot

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609d56810>
```

Seasonal Difference of average monthly fossil oil outputs



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted in the average monthly fossil oil outputs.

```
In [ ]: Fossil_Oil_Autocorrelations = sm.tsa.acf(df_FossilOil["Fossil_Oil"], fft=False) #Autocorrelations
print(Fossil_Oil_Autocorrelations)
```

```
[ 1.          0.71054958  0.66732429  0.5303742   0.483078   0.34957682
 0.34392347  0.18524268  0.10772907  -0.04191202  -0.13650081  -0.16348128
 -0.167235  -0.22360295  -0.20717914  -0.23385523  -0.19619567  -0.17538544
 -0.06775588  -0.04015816  0.03265232  -0.04587039  0.05189943  0.0711054
 0.10487331  0.06946849  0.05429799  -0.02060905  -0.0268963  -0.09255742
 -0.11012491  -0.10608085  -0.13789422  -0.21240935  -0.2071119  -0.24884842
 -0.24997623  -0.21499944  -0.1835364  -0.16630396  -0.14888741]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]: from scipy.stats import skew
```

```
#Bell Curves
```

```
FossilOilResults_mean = np.mean(df_FossilOil["Ratio"])
FossilOilResults_std = np.std(df_FossilOil["Ratio"])
```

```
FossilOilResultspdf = stats.norm.pdf(df_FossilOil["Ratio"].sort_values(), FossilOilResults_mean, FossilOilResults_std)
```

```
plt.plot(df_FossilOil["Ratio"].sort_values(), FossilOilResultspdf)
plt.xlim([0,20])
```

```
plt.xlabel("Output to energy price per EUR/MWH", size=15)
```

```
plt.ylabel("Frequency", size=15)
```

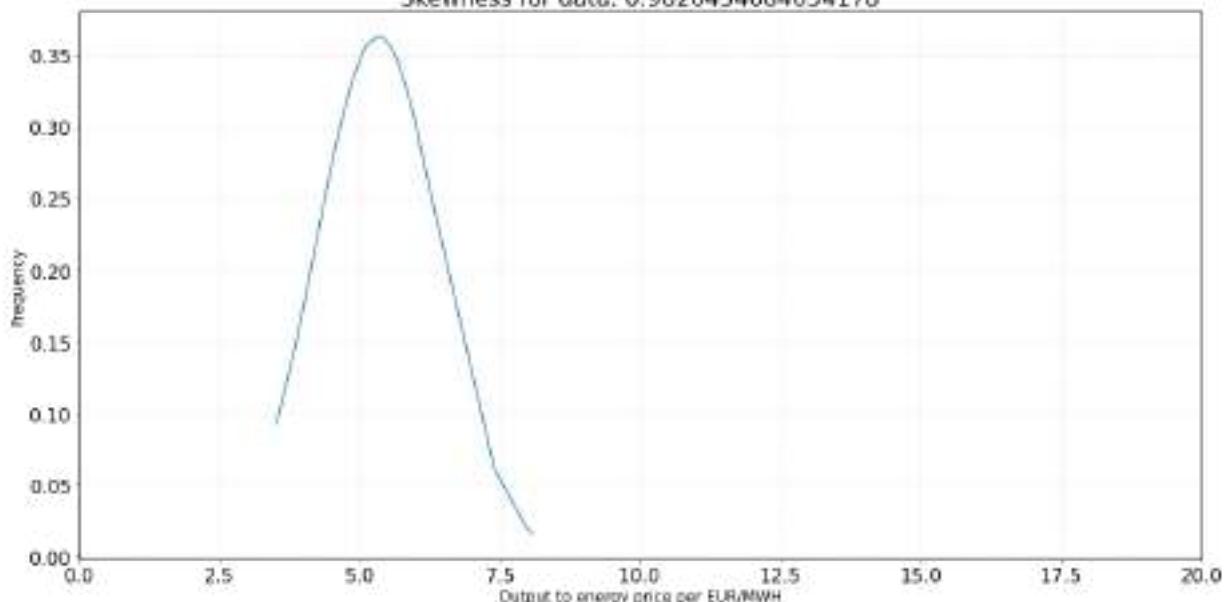
```
plt.grid(True, alpha=0.3, linestyle="--")
```

```
plt.title(f'Skewness for data: {skew(df_FossilOil["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
```

```
plt.suptitle("Frequency distribution of average monthly fossil oil outputs to energy prices per EUR/MWH ratios")
plt.show()
```

Frequency distribution of average monthly fossil oil outputs to energy prices per EUR/MWH ratios (scaled in decimals)

Skewness for data: 0.9820454084054178



In []: #Bell Curves

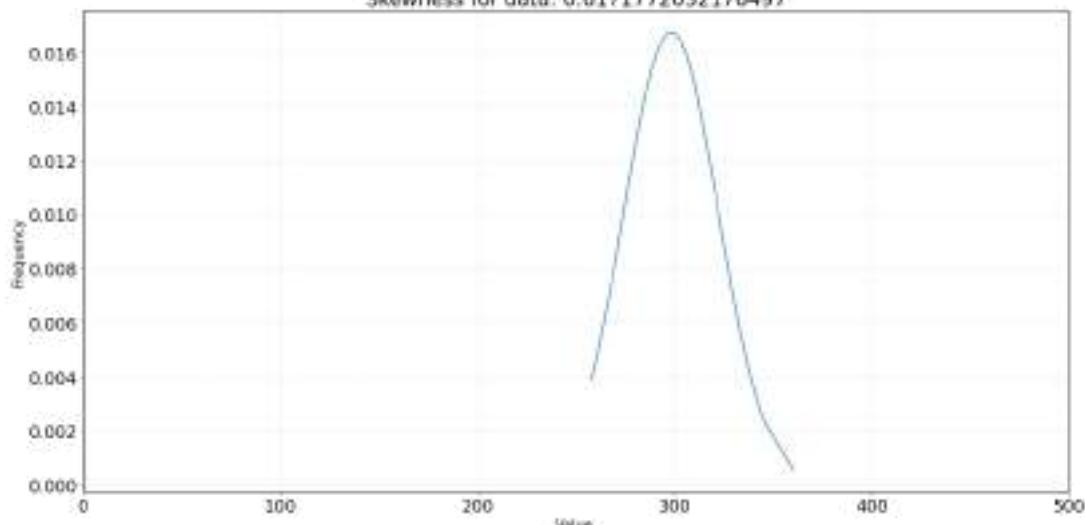
```
FossilOilResults_mean = np.mean(df_FossilOil["Fossil_Oil"])
FossilOilResults_std = np.std(df_FossilOil["Fossil_Oil"])

FossilOilResultspdf = stats.norm.pdf(df_FossilOil["Fossil_Oil"].sort_values(), FossilOilResults_mean, FossilOilResults_std)

plt.plot(df_FossilOil["Fossil_Oil"].sort_values(), FossilOilResultspdf)
plt.xlim([0,500])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilOil["Fossil_Oil"])}')
plt.suptitle("Frequency distribution of average monthly fossil oil outputs versus average monthly energy prices per EUR/MWH")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

Frequency distribution of average monthly fossil oil outputs versus average monthly energy prices per EUR/MWH (scaled in decimals)

Skewness for data: 0.6171772032176497



These bell-shaped curves are respectively skewed to the right. Hence, they have asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

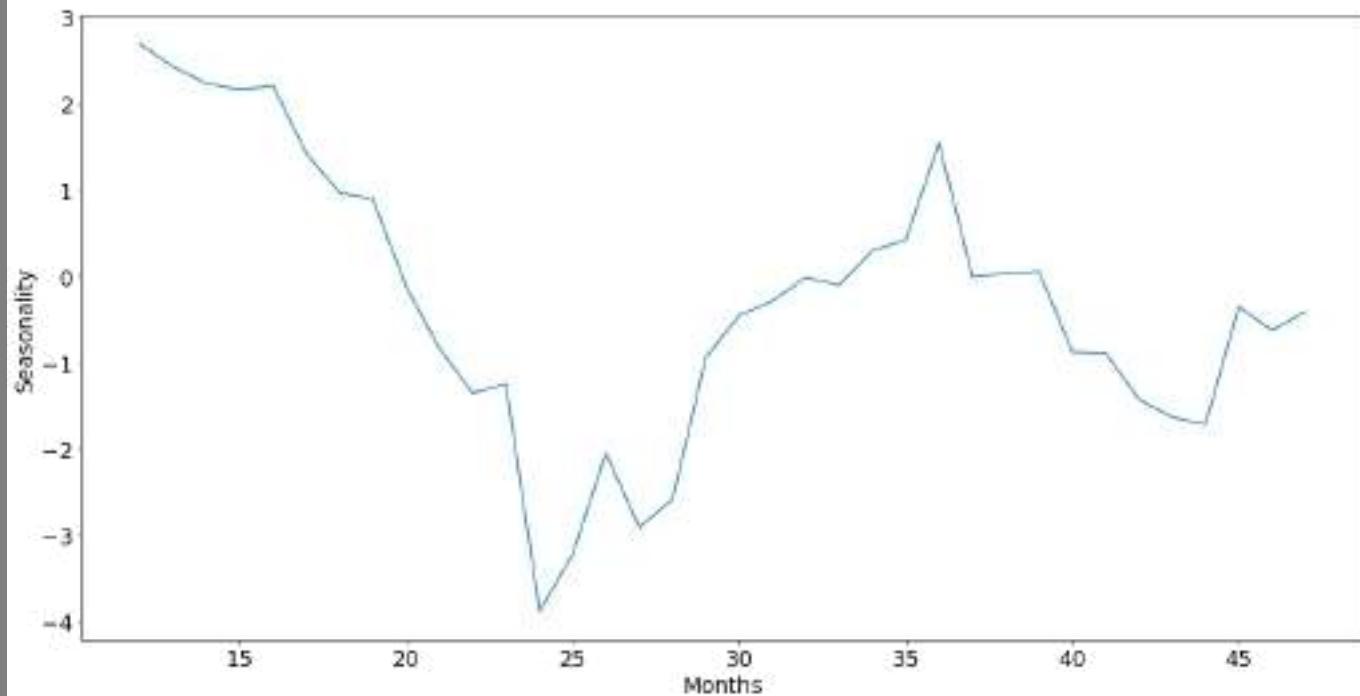
```
df_FossilOil['First Difference ratios'] = df_FossilOil["Ratio"]- df_FossilOil["Ratio"].shift(1) # Seasonality variable
df_FossilOil['Seasonal Difference ratios']=df_FossilOil["Ratio"]- df_FossilOil["Ratio"].shift(12)
df_FossilOil.head()

plt.suptitle("Seasonal Difference of average monthly fossil oil outputs to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
```

```
df_FossilOil['Seasonal Difference ratios'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609c554d0>
```

Seasonal Difference of average monthly fossil oil outputs to price of energy per EUR/MWH



```
In [ ]:
```

```
In [ ]:
```

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly megawatts from fossil oil and the average monthly prices of energy per EUR/MWH.

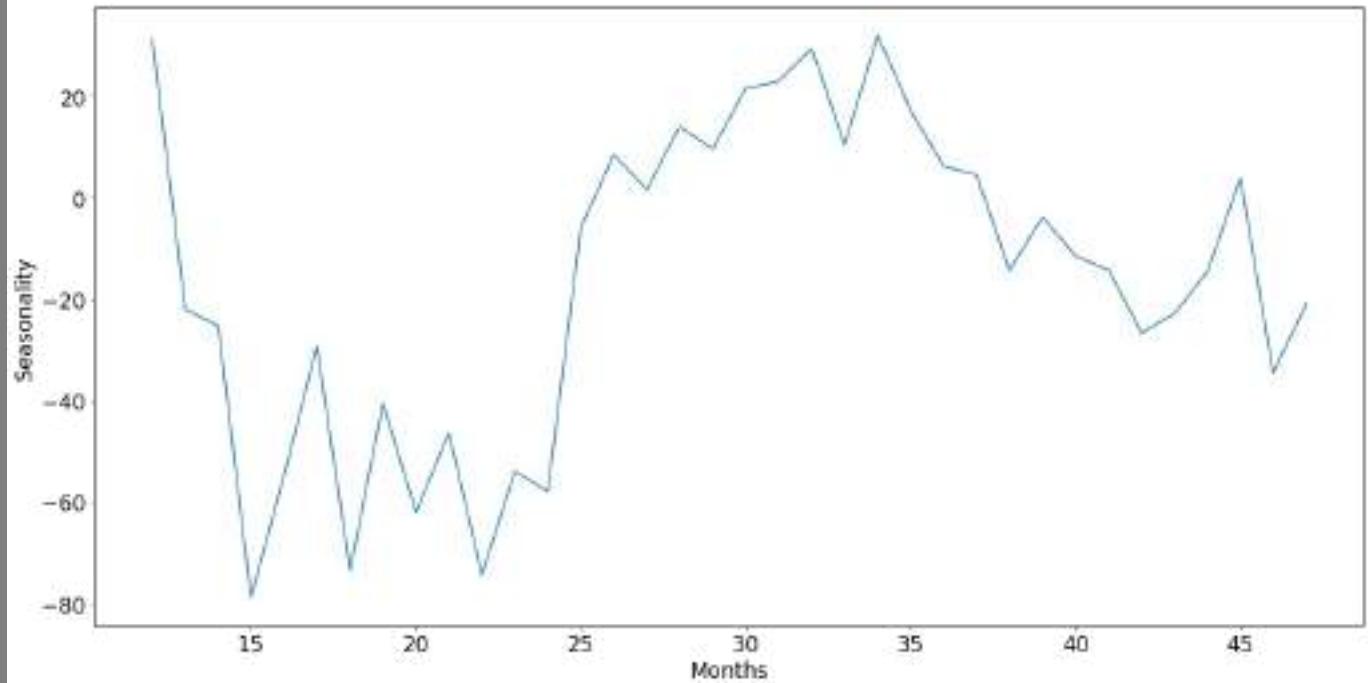
```
In [ ]:
```

```
In [ ]:
```

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly Price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilOil['Seasonal Difference'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609b4df90>
```

Seasonal Difference of average monthly Price of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly megawatts from fossil oil and the average monthly prices of energy per EUR/MWH.

In []: df_FossilOil.describe(include = 'all') # Description Tables

Out[]:

	Price	Fossil_Oil	Dates	Ratio	First Difference Fossil Oil Ratio	Seasonal Difference Fossil Oil Ratio	First Difference	Seasonal Difference	First Difference ratios	Seasonal Difference ratios
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	57.859848	298.324069	NaN	5.334508	-0.013070	-0.295283	-0.702918	-15.989799	-0.013070	-0.295283
std	10.320573	24.078477	NaN	1.109870	0.659037	1.612341	18.087981	31.741994	0.659037	1.612341
min	32.618667	257.629167	NaN	3.516361	-1.746713	-3.887643	-41.017125	-78.893837	-1.746713	-3.887643
25%	51.528411	281.505343	NaN	4.540563	-0.260050	-1.276519	-7.980306	-36.043044	-0.260050	-1.276519
50%	59.622281	292.923387	NaN	5.216690	-0.093915	-0.322757	0.094199	-14.326555	-0.093915	-0.322757
75%	64.999583	312.908413	NaN	5.764717	0.242475	0.536684	11.492276	8.732157	0.242475	0.536684
max	79.492083	360.233871	NaN	8.088094	2.062453	2.692301	41.004386	31.973611	2.062453	2.692301

Below is a histogram of average monthly fossil oil outputs.

In []:

```
fossil_oil_Dict = {key: i for i, key in enumerate(fossil_oil)}

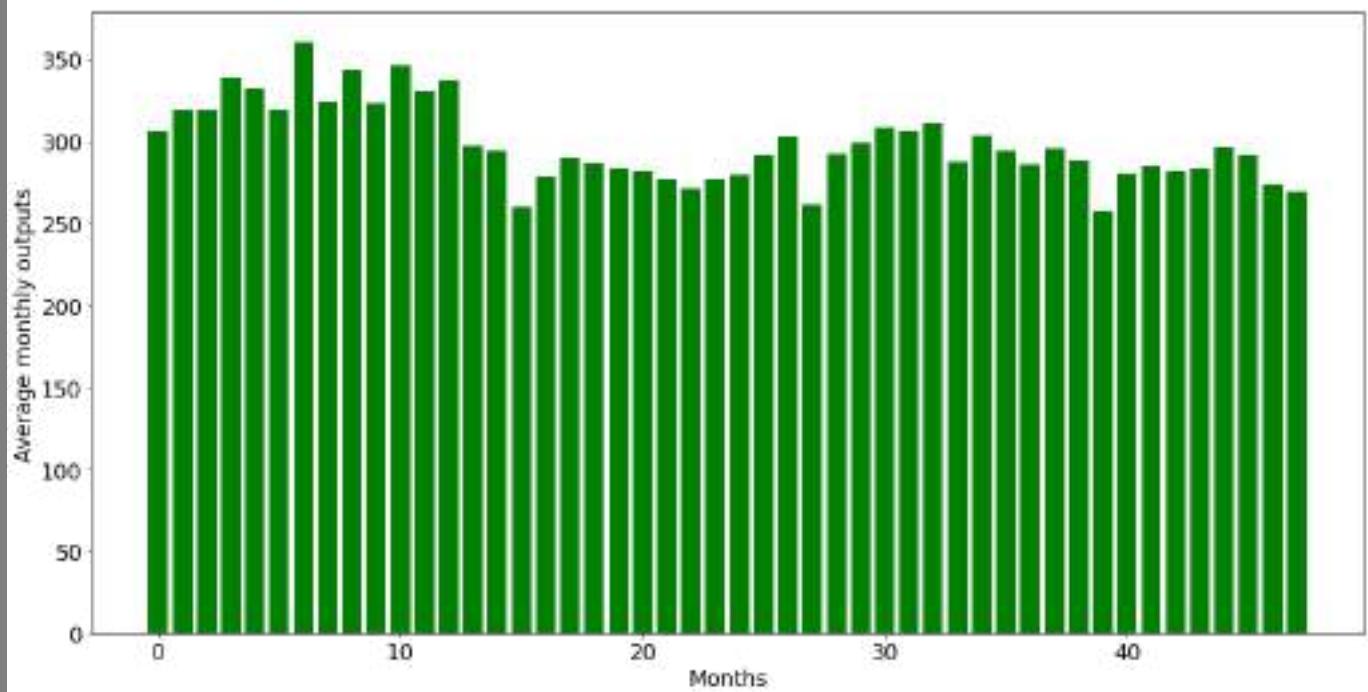
def Hist_fossil_oil(fossil_oil_Dict):
    for k, v in fossil_oil_Dict.items(): print(f"{v}:{k}")
print(fossil_oil_Dict)

plt.bar(list(fossil_oil_Dict.values()), fossil_oil_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of fossil oil")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')
#Outputs in MWH Histogram

plt.show()
plt.show()
```

```
{306.0204638472033: 0, 319.2395833333333: 1, 319.3337819650067: 2, 338.78133704735376: 3, 332.56720430107526: 4, 319.2294853963839: 5, 360.23387096774195: 6, 323.9139784946237: 7, 343.4916666666667: 8, 323.39112903225805: 9, 346.06388888888887: 10, 330.6514131897712: 11, 337.46505376344084: 12, 297.25431034482756: 13, 294.05248990578735: 14, 259.8875: 15, 277.9153225806452: 16, 289.8291666666665: 17, 286.6900269541779: 18, 283.30645161290323: 19, 281.386111111111: 20, 276.9959731543624: 21, 271.56944444444446: 22, 276.63978494623655: 23, 279.52284946236557: 24, 291.4017857142857: 25, 302.50740242261105: 26, 261.4902777777778: 27, 291.9206989247312: 28, 299.393055555556: 29, 308.2002688172043: 30, 306.23521505376345: 31, 310.8013888888889: 32, 287.4187919463087: 33, 303.54305555555555: 34, 293.9260752688172: 35, 285.73924731182797: 36, 295.9002976190476: 37, 288.1265141318977: 38, 257.62916666666666: 39, 280.4502688172043: 40, 285.120833333333334: 41, 281.5450874831763: 42, 283.33333333333333: 43, 296.136111111111: 44, 291.2993288590604: 45, 272.983333333333335: 46, 269.02150537634407: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```

Average monthly outputs of fossil oil



The green bars represent the observation value for each respective month. The histogram is roughly symmetrical, with a unimodal in between months 0 and 10. Based off of this histogram, the output of fossil oil has been roughly the same from months 0 to 48. As one can observe the histogram, one may notice that it is mostly uniform with a slight decrease in output overtime.

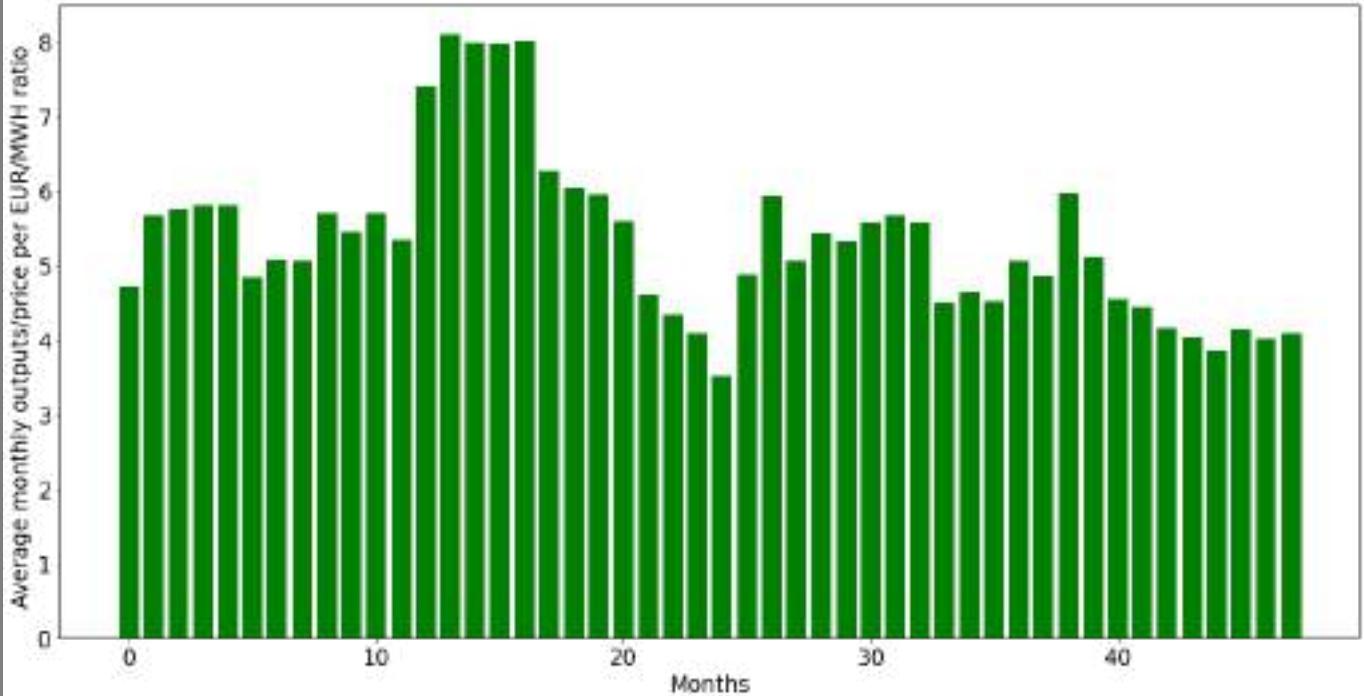
```
In [ ]: pdToListFossil_Oil_Dict = {key: i for i, key in enumerate(pdToListFossilOil)}
```

```
def Hist_pdToList(pdToListFossil_Oil_Dict):
    for k, v in pdToListFossil_Oil_Dict.items(): print(f"{v}:{k}")
print(dicDates)
#Output/price per EUR/MWH Histogram
print(pdToListFossil_Oil_Dict)
plt.bar(list(pdToListFossil_Oil_Dict.values()), pdToListFossil_Oil_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of fossil oil")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')
```

```
plt.show()
plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{4.711702646478498: 0, 5.661897152147204: 1, 5.751434010315323: 2, 5.805614923503275: 3, 5.804566988170348: 4, 4.838650334531596: 5, 5.068573460217559: 6, 5.061308977758065: 7, 5.700653129246291: 8, 5.443675118216539: 9, 5.6987019961215175: 10, 5.341551018408021: 11, 7.4040641202492315: 12, 8.088094153705416: 13, 7.986648525467014: 14, 7.967447065075214: 15, 8.01107926345912: 16, 6.264366177099672: 17, 6.035323346601258: 18, 5.951523796777401: 19, 5.582437796948807: 20, 4.6026053669926625: 21, 4.339483283457277: 22, 4.092599080785782: 23, 3.516360846780745: 24, 4.869861480689363: 25, 5.9361860592871185: 26, 5.056086838592902: 27, 5.42879803273591: 28, 5.321765312329655: 29, 5.578024867227532: 30, 5.662180277016627: 31, 5.56826528969134: 32, 4.4961672959667265: 33, 4.6391567662709905: 34, 4.511439979947413: 35, 5.056260079995415: 36, 4.8606176427379415: 37, 5.967858344556159: 38, 5.11161515773717: 39, 4.550270065632229: 40, 4.4309103302219315: 41, 4.153596865402372: 42, 4.026685386551291: 43, 3.8502216850665367: 44, 4.139996585263025: 45, 4.012694543070627: 46, 4.097402214308303: 47}
```

Average monthly outputs/price per EUR/MWH ratio of fossil oil



In []:

The green bars represent the observation value for each respective month. The histogram above displays a bimodal distribution. In addition, there is a large trench in the center of the histogram between months 20 and 30. This indicates that there was a sharp decline in the output produced per EUR/MWH.

Below is a box and whisker plot for monthly fossil oil outputs and the prices of energy per EUR/MWH.

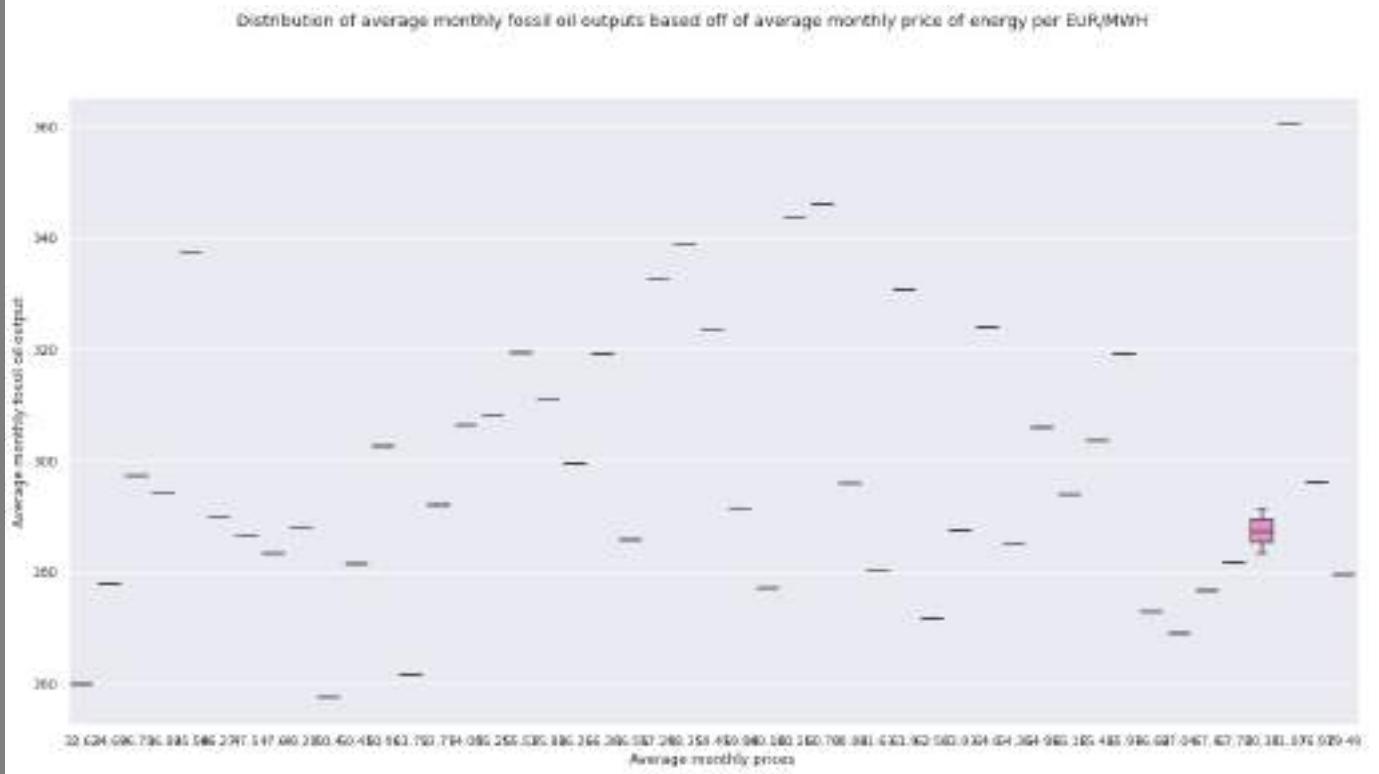
In []:

```
# Box and Whisker Plot
sns.set(style="darkgrid")
plt.suptitle('Distribution of average monthly fossil oil outputs based off of average monthly price of energy')
plt.xlabel('Average monthly prices')

plt.rcParams["figure.figsize"] = [40, 20]
plt.ylabel('Average monthly fossil oil output')

sns.boxplot(x=Rounded_Y, y=[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 338.8133704735376])
plt.show
```

Out[]: <function matplotlib.pyplot.show(*args, **kw)>



This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil oil produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil oil produced, which was in between 280 and 300 units, was when the price of energy per EUR/MWH was at roughy 70. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 units), fossil oil output was at 260.The lowest amount produced, which was slightly below 260 units, was at the price of approximently 50.4 EUR/MWH. When the price of energy per EUR/MWH was at its highest, at approximately 79.49 EUR/MWH fossil oil output was at 280. At approximately 71.07 EUR/MWH, fossil oil produced the most units at roughly 360.

```
In [ ]: import matplotlib.pyplot as plt

plt.suptitle('Distribution of average monthly fossil oil outputs based off of average monthly price of energy per EUR/MWH')
plt.xlabel('Average monthly prices')

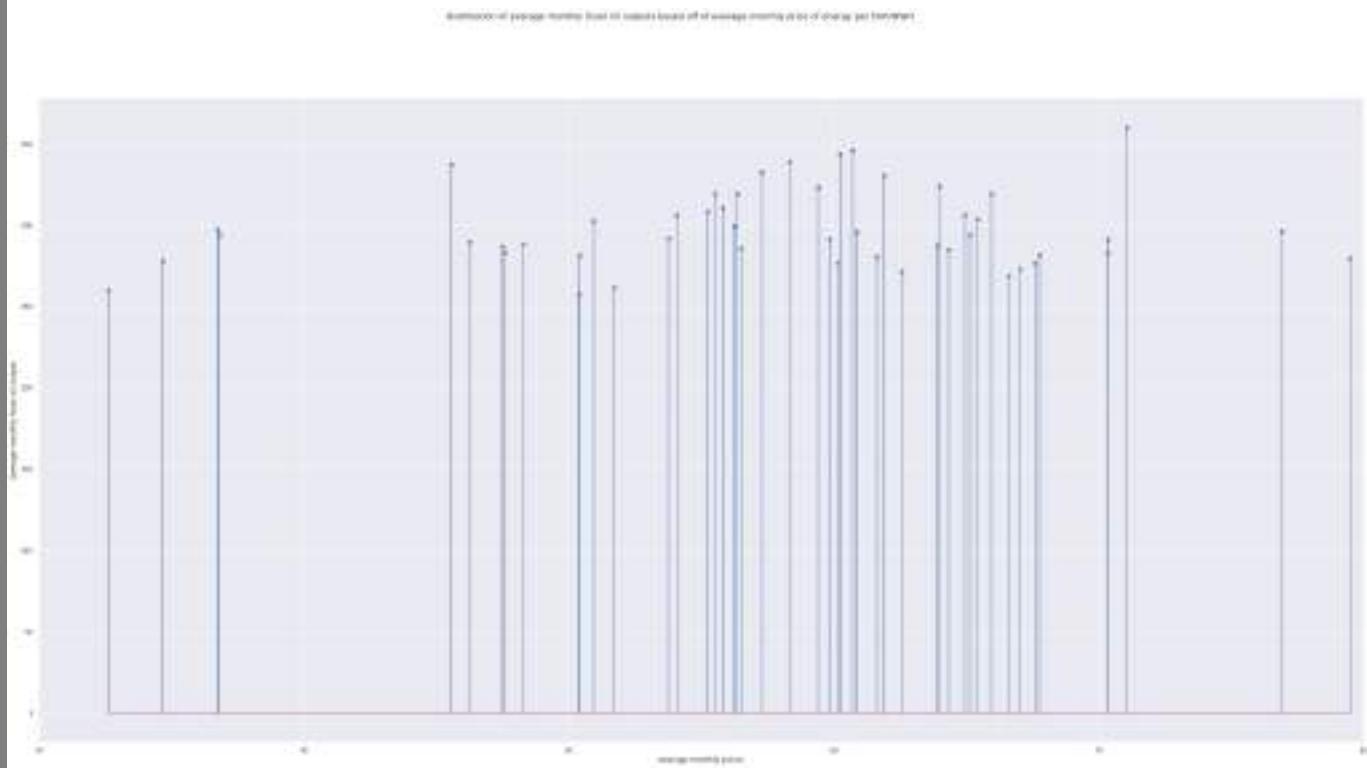
plt.rcParams["figure.figsize"] = [40, 15]
plt.ylabel('Average monthly fossil oil output')

plt.xlim(30, 80)

# Stem Plot
plt.stem(Rounded_Y, fossil_oil)
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.
from ipykernel import kernelapp as app

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of fossil oil produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil oil produced, which was in between 280 and 300 units, was when the price of energy per EUR/MWH was at roughly 70. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 units), fossil oil output was at 260. The lowest amount produced, which was slightly below 260 units, was at the price of approximately 50.4 EUR/MWH. When the price of energy per EUR/MWH was at its highest, at approximately 79.49 EUR/MWH fossil oil output was at 280. At approximately 71.07 EUR/MWH, fossil oil produced the most units at roughly 360. Each blue dot and the end of the blue lines represent an observation.

In []: `print(fossil_oil)`

```
[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.229485396389, 360.23387096774195, 323.9139784946237, 343.4916666666667, 323.39112903225805, 346.06388888888887, 330.6514131897712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875, 277.9153225806452, 289.82916666666665, 286.6900269541779, 283.30645161290323, 281.3861111111111, 276.9959731543624, 271.56944444444446, 276.63978494623655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.49027777777778, 291.9206989247312, 299.3930555555556, 308.2002688172043, 306.23521505376345, 310.8013888888889, 287.4187919463087, 303.54305555555555, 293.9260752688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916666666666, 280.4502688172043, 285.1208333333334, 281.5450874831763, 283.333333333333, 296.1361111111111, 291.2993288590604, 272.9833333333335, 269.02150537634407]
```

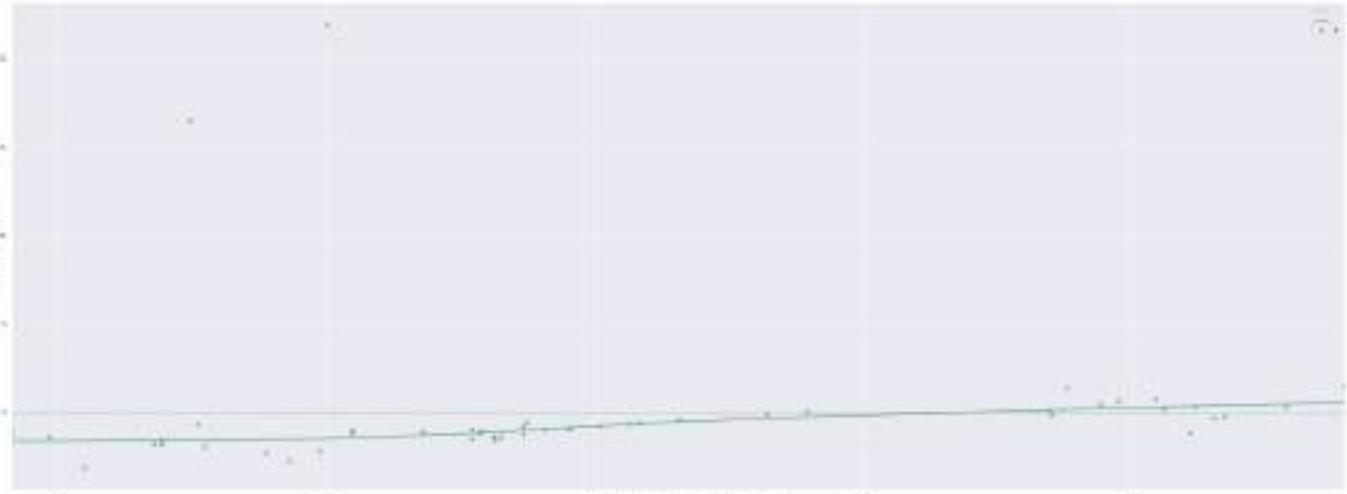
In []:

In []: `#Predicted OLS linear average monthly ratios versus residuals`
`FossilOilRegRatioPredict = predictionsFossilOil/predictions`

```
sns.residplot(x = FossilOilRegRatioPredict, y = standardized_residualsFossilOil/standardized_residualsPricereg,
plt.suptitle("Predicted fossil oil output to EUR/MWH versus respective linear model residuals ")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")
```

```
plt.legend(..#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff609349590>
```



With the exception of a few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
```

```
FossilOilQuadRatioPredict = FossilOil_ypred/ypred
```

```
sns.residplot(x = FossilOilQuadRatioPredict , y = standardized_residualsFossilOilQuad/standardized_residualsPrice)
```

```
plt.suptitle("Predicted average monthly fossil oil energy prices to EUR/MWH versus respective quadratic model results")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")
```

```
plt.legend(..#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60932f210>
```



With the exception of a few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]:
```

Below is a scatterplot depicting the relationship between the average monthly prices of energy per EUR/MWH and the average monthly outputs of fossil oil.

```
In [ ]: modelFossilOil = stats.linregress([306.0204638472033, 319.2395833333333, 319.3337819650067, 338.78133704735376, [64.9490188172043, 56.383854166666666, 55.522462987886975, 58.3540833333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.752083333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.4055972222224, 60.182429530201404, 62.58105555555558, 67.5951344086021, 79.4920833333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.2582222222224, 55.252580645161316, 54.08432795698931, 55.81655555555558, 63.92528859060403, 65.43065277777781, 65.15127688172035, 56.51197580645163, 60.877098214285674, 48.279717362045766, 50.4007361111113, 61.633763440860214, 64.34813888888884, 67.78344086021498, 70.36391129032262, 76.9140416666666, 70.36221476510062, 67.0426075268817, 66.62351388888881])
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

The results from the list directly below will be analyzed for seasonality since the price of energy per EUR/MWH for each resource and the respective average monthly prices of energy per EUR/MWH from the original models are taken into account simultaneously.

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(predictionsFossilOil))
print(list(predictions))
```

```
[58.33094073693212, 59.1400764681692, 59.145842319578634, 60.33621749689952, 59.95585352114911, 59.139458378469186, 61.64931643202709, 59.42619387189568, 60.62453443093506, 59.39419051626356, 60.78197887425564, 59.838588899752125, 60.25564845154355, 57.794368839558516, 57.59838700242096, 55.507164957024386, 56.61063901380311, 57.339879486778784, 57.14773430574877, 56.94062734141346, 56.82308425408665, 56.55436609251828, 56.22221096823898, 56.53256398934674, 56.70903493497538, 57.4361387167818, 58.11590799381706, 55.605270187905575, 57.46790113639098, 57.92528037052908, 58.464365515352995, 58.344085551511974, 58.62357878167124, 57.1923416485772, 58.17929980716078, 57.59064922534502, 57.08953755656549, 57.71149039382483, 57.23566097167962, 55.368933496528015, 56.765801812574765, 57.05168474441551, 56.83281511877816, 56.942272758155475, 57.725924420428235, 57.429867383614706, 56.3087544041463, 56.066252884708675]  
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895]
```

```
In [ ]:  
dfFossilOilReg = ({ "Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895]  
  
#Dataframes  
df_FossilOilReg= pd.DataFrame.from_dict(dfFossilOilReg, orient = "columns")  
print(df_FossilOilReg)  
  
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis,indicating it is non-stationary ")  
  
adfuller_test(df_FossilOilReg["Fossil_OilReg"])  
  
test_result=adfuller(df_FossilOilReg["Fossil_OilReg"])  
  
df_FossilOilReg['First Difference'] = df_FossilOilReg["Fossil_OilReg"]- df_FossilOilReg["Fossil_OilReg"].shift()  
df_FossilOilReg['Seasonal Difference']=df_FossilOilReg["Fossil_OilReg"]- df_FossilOilReg["Fossil_OilReg"].shift()  
plt.suptitle("Seasonal Difference of average monthly predicted fossil oil linear prices of energy per EUR/MWH")  
plt.ylabel('Seasonality')  
plt.xlabel('Months')  
df_FossilOilReg.head() #Predictive Linear Seasonality Plot  
df_FossilOilReg['Seasonal Difference'].plot()  
# Seasonality Plot
```

```
{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.1109745796539, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'Fossil_OilReg': [58.33094073693212, 59.1400764681692, 59.145842319578634, 60.33621749689952, 59.95585352114911, 59.139458378469186, 61.64931643202709, 59.42619387189568, 60.62453443093506, 59.39419051626356, 60.78197887425564, 59.838588899752125, 60.25564845154355, 57.794368839558516, 57.59838700242096, 55.507164957024386, 56.61063901380311, 57.339879486778784, 57.14773430574877, 56.94062734141346, 56.82308425408665, 56.55436609251828, 56.22221096823898, 56.532563989394674, 56.70903493497538, 57.4361387167818, 58.11590799381706, 55.605270187905575, 57.46790113639098, 57.92528037052908, 58.464365515352995, 58.344085551511974, 58.62357878167124, 57.1923416485772, 58.17929980716078, 57.59064922534502, 57.08953755656549, 57.71149039382483, 57.23566097167962, 55.368933496528015, 56.765801812574765, 57.05168474441551, 56.83281511877816, 56.942272758155475, 57.725924420428235, 57.429867383614706, 56.3087544041463, 56.066252884708675]}
```

```
    Price Fossil_OilReg
```

0	52.821613	58.330941
1	53.036006	59.140076
2	53.250399	59.145842
3	53.464792	60.336217
4	53.679185	59.955854
5	53.893578	59.139458
6	54.107971	61.649316
7	54.322364	59.426194
8	54.536757	60.624534
9	54.751150	59.394191
10	54.965543	60.781979
11	55.179936	59.838589
12	55.394329	60.255648
13	55.608722	57.794369
14	55.823115	57.598387
15	56.037508	55.507165
16	56.251901	56.610639
17	56.466294	57.339879
18	56.680687	57.147734
19	56.895080	56.940627
20	57.109473	56.823084
21	57.323866	56.554366
22	57.538259	56.222211
23	57.752652	56.532564
24	57.967045	56.709035
25	58.181438	57.436139
26	58.395831	58.115908
27	58.610224	55.605270
28	58.824617	57.467901
29	59.039010	57.925280
30	59.253403	58.464366
31	59.467796	58.344086
32	59.682189	58.623579
33	59.896582	57.192342
34	60.110975	58.179300
35	60.325368	57.590649
36	60.539761	57.089538
37	60.754154	57.711490
38	60.968547	57.235661
39	61.182939	55.368933
40	61.397332	56.765802
41	61.611725	57.051685
42	61.826118	56.832815
43	62.040511	56.942273
44	62.254904	57.725924
45	62.469297	57.429867
46	62.683690	56.308754
47	62.898083	56.066253

```
ADF Test Statistic : -2.5474323265829337
```

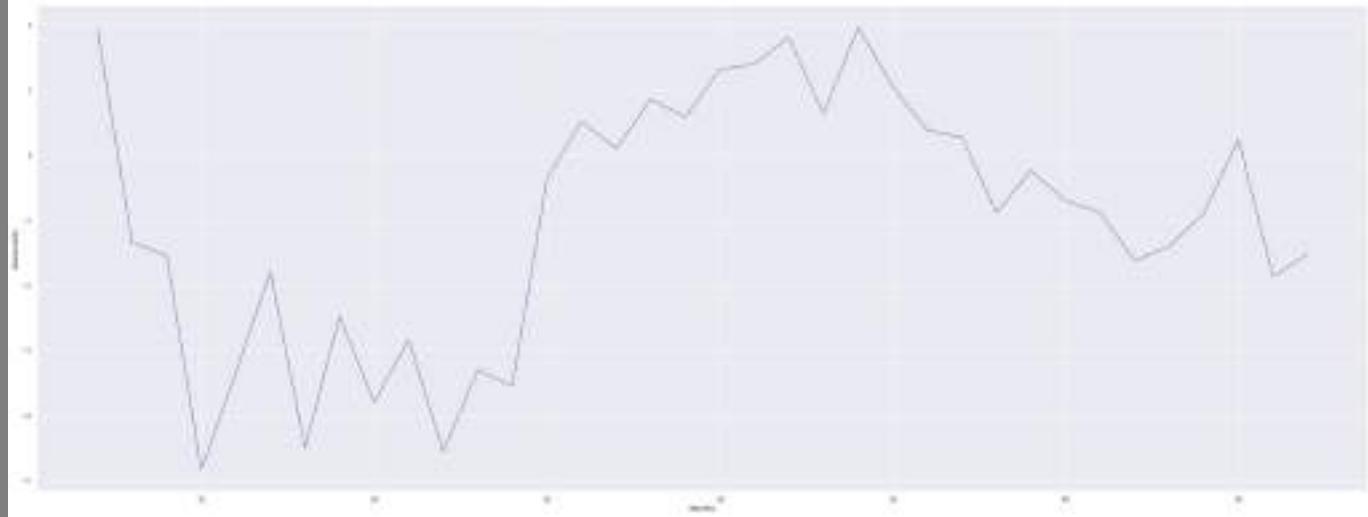
```
p-value : 0.10435578418262581
```

```
#Lags Used : 0
```

```
Number of Observations : 47
```

```
weak evidence against null hypothesis, indicating it is non-stationary
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60924ae90>
```



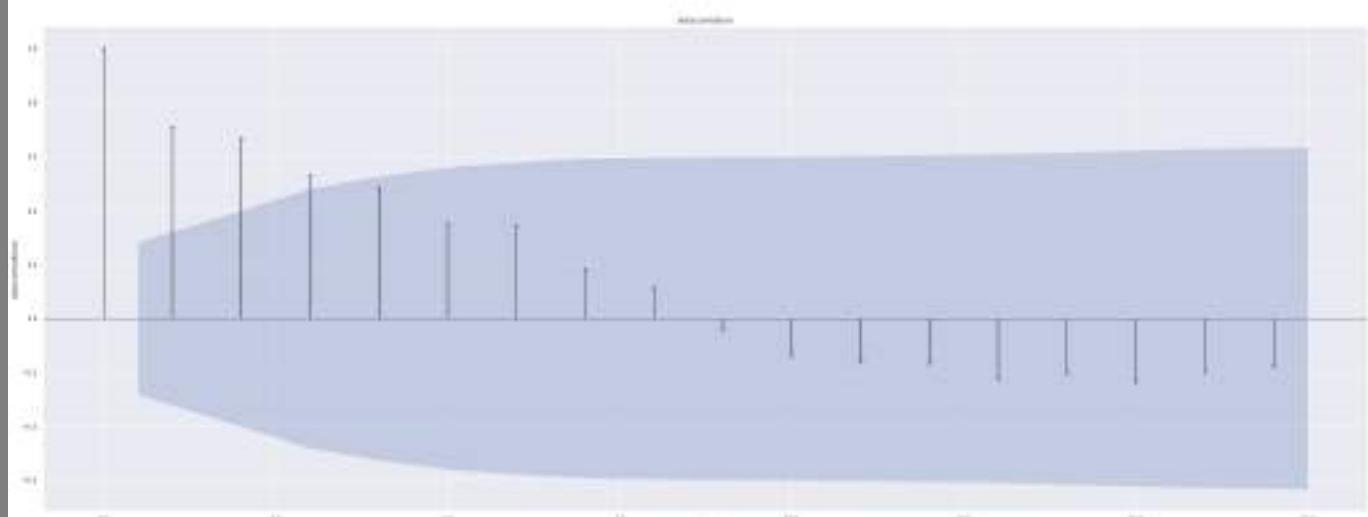
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

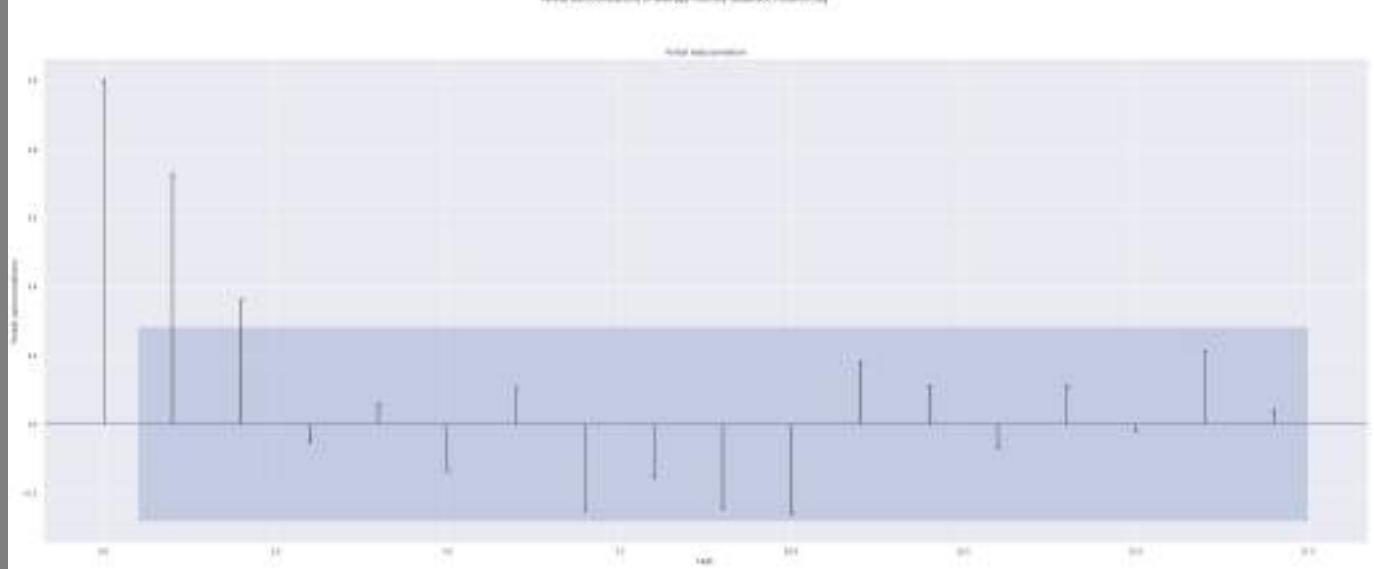
```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(predictionsFossilOil)
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilOil Log")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(predictionsFossilOil)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilOil Log")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
FossilOil_Reg_Autocorrelations = sm.tsa.acf(predictionsFossilOil, fft=False) #Autocorrelations
print(FossilOil_Reg_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.70952824  0.66802752  0.52851278  0.48281801  0.3500967
  0.3432444   0.18192468  0.11216314 -0.04078178 -0.13797814 -0.16229098
 -0.16583677 -0.22594728 -0.2037795  -0.23451912 -0.19590997 -0.17666594
 -0.06884229 -0.04458249  0.03861587 -0.04748505  0.05017233  0.07068622
  0.10413612  0.07025746  0.05493628 -0.02032984 -0.02640435 -0.09210102
 -0.11185708 -0.10870195 -0.13292692 -0.21194383 -0.20126559 -0.24983907
 -0.24773538 -0.21829587 -0.18061395 -0.16915039 -0.1436068 ]
```





The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

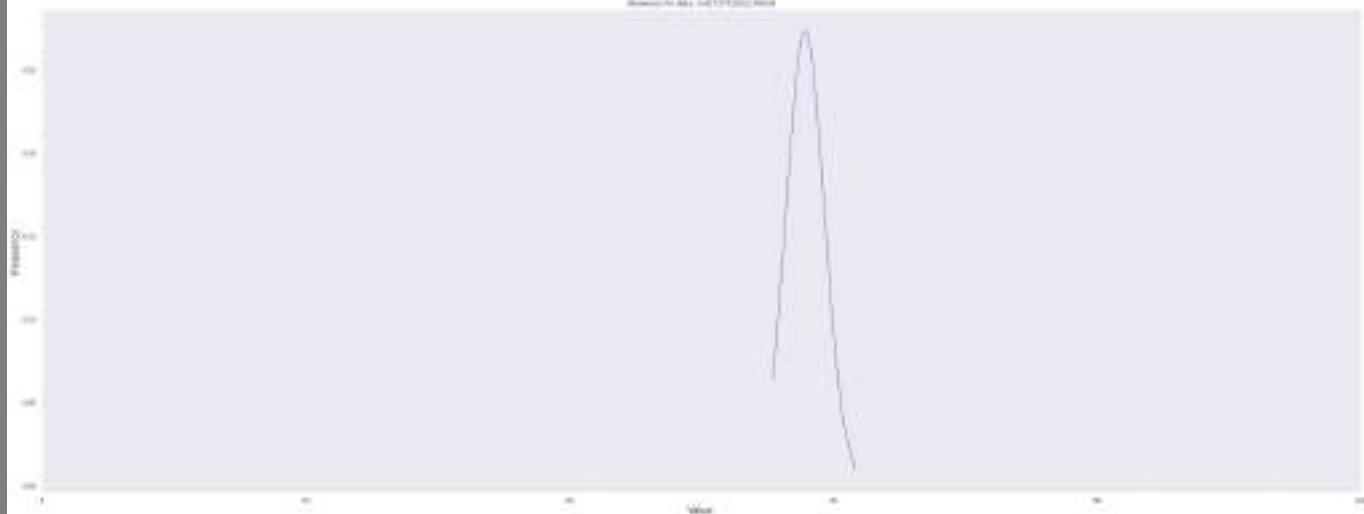
As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

In []: #Bell Curves

```
FossilOilRegResults_mean = np.mean(df_FossilOilReg["Fossil_OilReg"])
FossilOilRegResults_std = np.std(df_FossilOilReg["Fossil_OilReg"])

FossilOilRegResultspdf = stats.norm.pdf(df_FossilOilReg["Fossil_OilReg"].sort_values(), FossilOilRegResults_mean)

plt.plot(df_FossilOilReg["Fossil_OilReg"].sort_values(), FossilOilRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilOilReg["Fossil_OilReg"])}')
plt.suptitle("Frequency distribution of predicted linear average monthly fossil oil energy prices per EUR/MWH (:")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve has a symmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

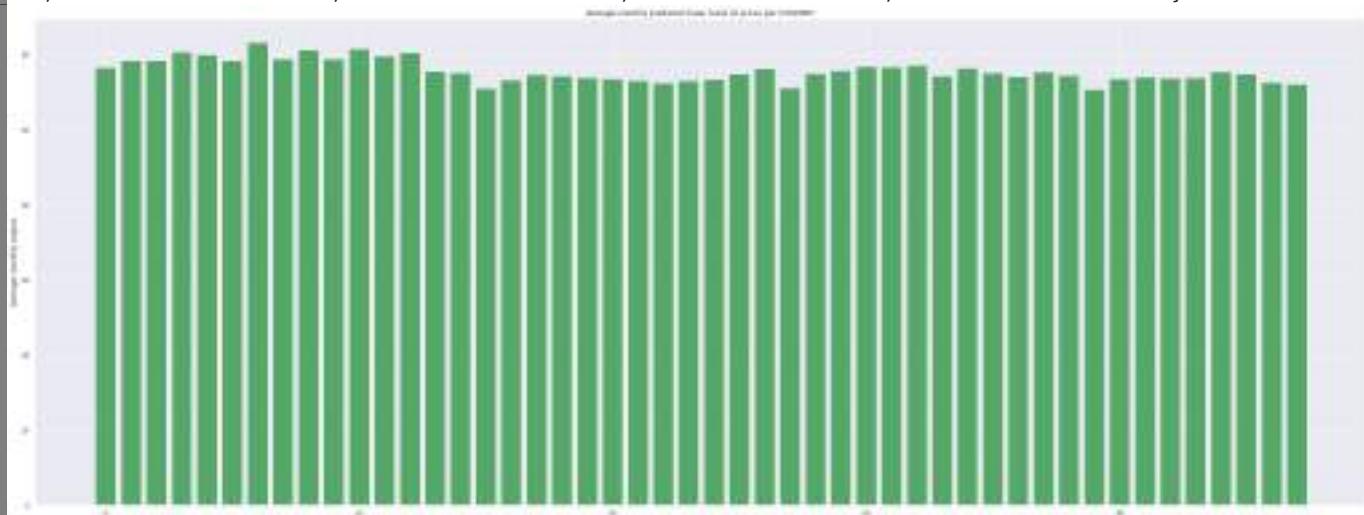
```
In [ ]:
print(dicDates)
Fossil_OilReg_Dict = {key: i for i, key in enumerate(predictionsFossilOil)}

def Hist_Fossil_OilReg(Fossil_OilReg_Dict):
    for k, v in Fossil_OilReg_Dict.items(): print(f"{v}:{k}")
print(Fossil_OilReg_Dict)

plt.bar(list(Fossil_OilReg_Dict.values()), Fossil_OilReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted linear fossil oil prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{58.33094073693212: 0, 59.1400764681692: 1, 59.145842319578634: 2, 60.33621749689952: 3, 59.95585352114911: 4, 59.139458378469186: 5, 61.64931643202709: 6, 59.42619387189568: 7, 60.62453443093506: 8, 59.39419051626356: 9, 60.78197887425564: 10, 59.838588899752125: 11, 60.25564845154355: 12, 57.794368839558516: 13, 57.59838700242096: 14, 55.507164957024386: 15, 56.61063901380311: 16, 57.339879486778784: 17, 57.14773430574877: 18, 56.940627341413: 19, 56.82308425408665: 20, 56.55436609251828: 21, 56.22221096823898: 22, 56.532563989394674: 23, 56.70903493: 24, 57.4361387167818: 25, 58.11590799381706: 26, 55.605270187905575: 27, 57.46790113639098: 28, 57.92528037052908: 29, 58.464365515352995: 30, 58.344085551511974: 31, 58.62357878167124: 32, 57.1923416485772: 33, 58.17929980716078: 34, 57.59064922534502: 35, 57.08953755656549: 36, 57.71149039382483: 37, 57.23566097167962: 38, 55.368933496528015: 39, 56.765801812574765: 40, 57.05168474441551: 41, 56.83281511877816: 42, 56.942272758155475: 43, 57.725924420428235: 44, 57.429867383614706: 45, 56.3087544041463: 46, 56.066252884708675: 47}
```



The green bars represent the observation value for each respective month. This histogram is uniform throughout.

```
In [ ]: df_FossilOilReg.describe(include = 'all') # Description Tables
```

	Price	Fossil_OilReg	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	57.859848	57.859848	-0.048185	-0.978728
std	3.001502	1.473832	1.101007	1.940811
min	52.821613	55.368933	-2.510638	-4.829053
25%	55.340731	56.830382	-0.488471	-2.024301
50%	57.859848	57.529275	-0.117543	-0.876921
75%	60.378966	58.752549	0.703437	0.534491
max	62.898083	61.649316	2.509858	1.957089

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(FossilOil_Logpred))
```

```
print(list(Logpred))
```

```
[58.33094073693212, 59.1400764681692, 59.145842319578634, 60.33621749689952, 59.95585352114911, 59.139458378469186, 61.64931643202709, 59.42619387189568, 60.62453443093506, 59.39419051626356, 60.78197887425564, 59.838588899752125, 60.25564845154355, 57.794368839558516, 57.59838700242096, 55.507164957024386, 56.61063901380311, 57.339879486778784, 57.14773430574877, 56.94062734141346, 56.82308425408665, 56.55436609251828, 56.22221096823898, 56.532563989394674, 56.70903493497538, 57.4361387167818, 58.11590799381706, 55.605270187905575, 57.46790113639098, 57.92528037052908, 58.464365515352995, 58.344085551511974, 58.62357878167124, 57.1923416485772, 58.17929980716078, 57.59064922534502, 57.08953755656549, 57.71149039382483, 57.23566097167962, 55.368933496528015, 56.765801812574765, 57.05168474441551, 56.83281511877816, 56.942272758155475, 57.725924420428235, 57.429867383614706, 56.308754404163, 56.066252884708675] [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.43336623019726, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: dfFossilOil_Log = ({'Price_Log':[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 'Fossil_Oil_Log' :[58.33094073693212, 59.1400764681692, 59.145842319578634, 60.336217496899]} print(dfFossilOil_Log) #Dataframes df_FossilOil_Log= pd.DataFrame.from_dict(dfFossilOil_Log, orient = "columns") print(df_FossilOil_Log)
```

```
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Fossil_Oil_Log': [58.33094073693212, 59.1400764681692, 59.145842319578634, 60.33621749689952, 59.95585352114911, 59.139458378469186, 61.64931643202709, 59.42619387189568, 60.62453443093506, 59.39419051626356, 60.78197887425564, 59.838588899752125, 60.25564845154355, 57.794368839558516, 57.59838700242096, 55.507164957024386, 56.61063901380311, 57.339879486778784, 57.14773430574877, 56.94062734141346, 56.82308425408665, 56.55436609251828, 56.22221096823898, 56.532563989394674, 56.70903493497538, 57.4361387167818, 58.11590799381706, 55.605270187905575, 57.46790113639098, 57.92528037052908, 58.464365515352995, 58.344085551511974, 58.62357878167124, 57.1923416485772, 58.17929980716078, 57.59064922534502, 57.08953755656549, 57.71149039382483, 57.23566097167962, 55.368933496528015, 56.765801812574765, 57.05168474441551, 56.83281511877816, 56.942272758155475, 57.725924420428235, 57.429867383614706, 56.3087544041463, 56.066252884708675]}

    Price_Log      Fossil_Oil_Log
0    27.297348      58.330941
1    23.917581      59.140076
2    23.577680      59.145842
3    24.695022      60.336217
4    24.276743      59.955854
5    27.702157      59.139458
6    29.713461      61.649316
7    26.922107      59.426194
8    25.445032      60.624534
9    25.110405      59.394191
10   25.631280      60.781979
11   26.094917      59.838589
12   19.653934      60.255648
13   16.170990      57.794369
14   16.197004      57.598387
15   14.539966      55.507165
16   15.357844      56.610639
17   19.925256      57.339879
18   20.412855      57.147734
19   20.452442      56.940627
20   21.558593      56.823084
21   25.416478      56.554366
22   26.362963      56.222211
23   28.341491      56.532564
24   33.035963      56.709035
25   25.280488      57.436139
26   21.777315      58.115908
27   22.076427      55.605270
28   22.887202      57.467901
29   23.868007      57.925280
30   23.471186      58.464366
31   23.010200      58.344086
32   23.693728      58.623579
33   26.893390      57.192342
34   27.487399      58.179300
35   27.377158      57.590649
36   23.968137      57.089538
37   25.690591      57.711490
38   20.719732      57.235661
39   21.556675      55.368933
40   25.989167      56.765802
41   27.060244      57.051685
42   28.415796      56.832815
43   29.434036      56.942273
44   32.018682      57.725924
45   29.433366      57.429867
46   27.958095      56.308754
47   28.123467      56.066253
```

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_FossilOil_Log["Fossil_Oil_Log"])
```

```
test_result=adfuller(df_FossilOil_Log["Fossil_Oil_Log"])
```

```
ADF Test Statistic : -2.5474323265829337
p-value : 0.10435578418262581
#Lags Used : 0
Number of Observations : 47
weak evidence against null hypothesis, indicating it is non-stationary
```

```
In [ ]: df_FossilOil_Log['First Difference'] = df_FossilOil_Log["Fossil_Oil_Log"]- df_FossilOil_Log["Fossil_Oil_Log"].shift(1)
df_FossilOil_Log['Seasonal Difference']=df_FossilOil_Log["Fossil_Oil_Log"]- df_FossilOil_Log["Fossil_Oil_Log"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic fossil oil prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilOil_Log.head() #Predictive Logarithmic Seasonality Plot
df_FossilOil_Log['Seasonal Difference'].plot()
# Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608fa3290>
```



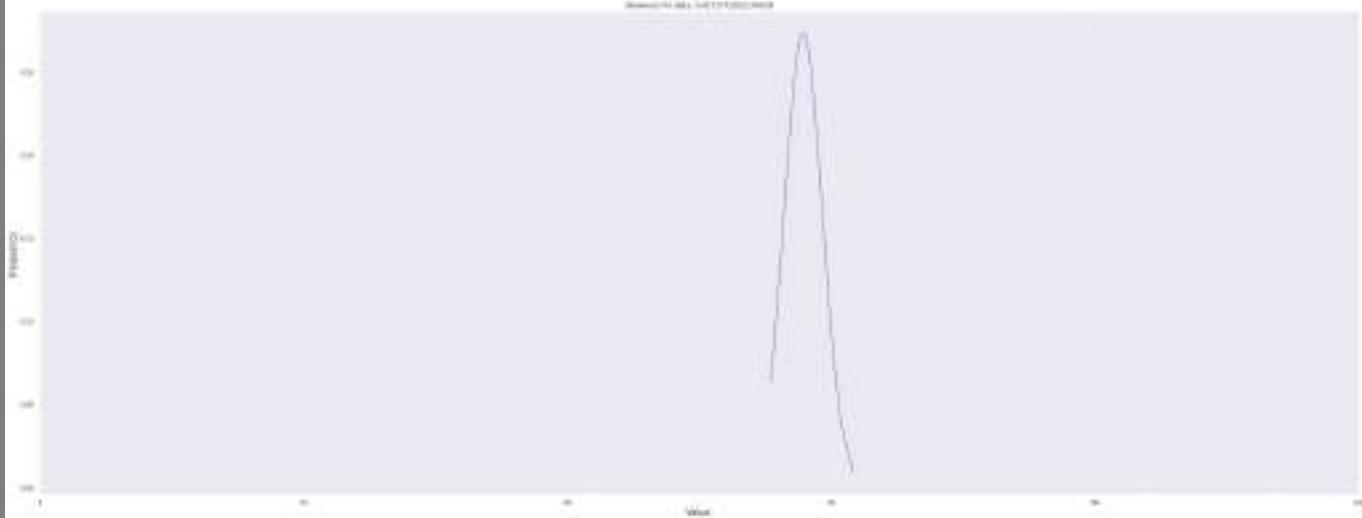
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Bell Curves
```

```
FossilOil_LogResults_mean = np.mean(df_FossilOil_Log["Fossil_Oil_Log"])
FossilOil_LogResults_std = np.std(df_FossilOil_Log["Fossil_Oil_Log"])

FossilOil_LogResultspdf = stats.norm.pdf(df_FossilOil_Log["Fossil_Oil_Log"].sort_values(), FossilOil_LogResults_mean)

plt.plot(df_FossilOil_Log["Fossil_Oil_Log"].sort_values(), FossilOil_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilOil_Log["Fossil_Oil_Log"])}')
plt.suptitle("Frequency distribution of average monthly fossil oil predicted logarithmic energy prices per EUR/MWH")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

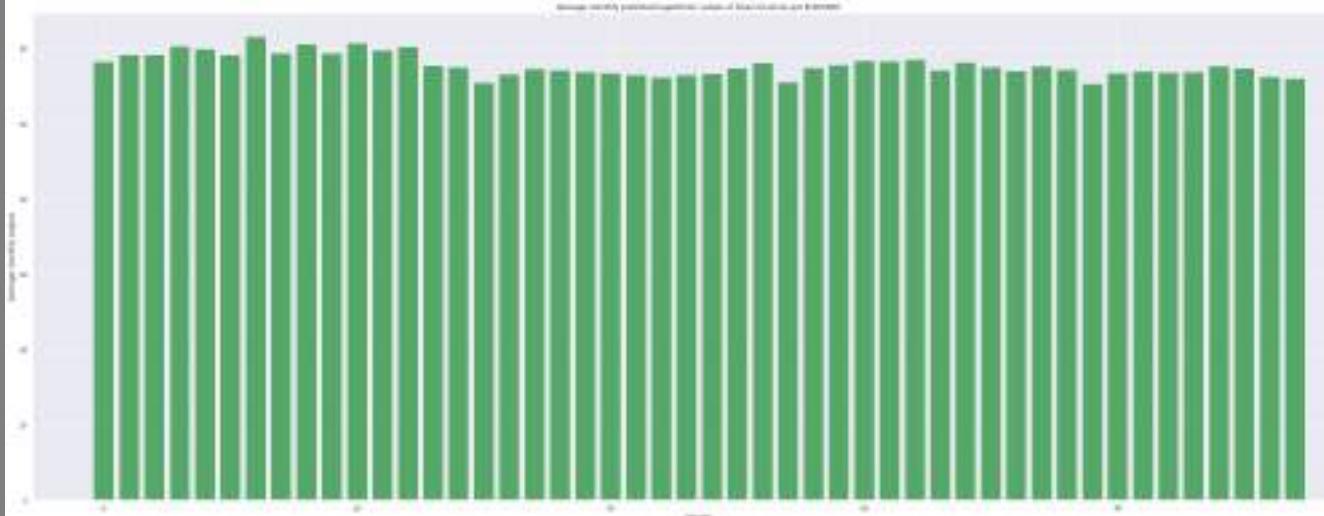
```
In [ ]:
print(dicDates)
Fossil_Oil_Log_Dict = {key: i for i, key in enumerate(df_FossilOil_Log["Fossil_Oil_Log"])}

def Hist_Fossil_Oil_Log(Fossil_Oil_Log_Dict):
    for k, v in Fossil_Oil_Log_Dict.items(): print(f"{v}:{k}")
print(Fossil_Oil_Log_Dict)

plt.bar(list(Fossil_Oil_Log_Dict.values()), Fossil_Oil_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly predicted logarithmic values of fossil oil prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{58.33094073693212: 0, 59.1400764681692: 1, 59.145842319578634: 2, 60.33621749689952: 3, 59.95585352114911: 4, 59.139458378469186: 5, 61.64931643202709: 6, 59.42619387189568: 7, 60.62453443093506: 8, 59.39419051626356: 9, 60.78197887425564: 10, 59.838588899752125: 11, 60.25564845154355: 12, 57.794368839558516: 13, 57.59838700242096: 14, 55.507164957024386: 15, 56.61063901380311: 16, 57.339879486778784: 17, 57.14773430574877: 18, 56.940627341413: 19, 56.82308425408665: 20, 56.55436609251828: 21, 56.22221096823898: 22, 56.532563989394674: 23, 56.70903493: 24, 57.4361387167818: 25, 58.11590799381706: 26, 55.605270187905575: 27, 57.46790113639098: 28, 57.92528037052908: 29, 58.464365515352995: 30, 58.344085551511974: 31, 58.62357878167124: 32, 57.1923416485772: 33, 58.17929980716078: 34, 57.59064922534502: 35, 57.08953755656549: 36, 57.71149039382483: 37, 57.23566097167962: 38, 55.368933496528015: 39, 56.765801812574765: 40, 57.05168474441551: 41, 56.83281511877816: 42, 56.942272758155475: 43, 57.725924420428235: 44, 57.429867383614706: 45, 56.3087544041463: 46, 56.066252884708675: 47}
```



The green bars represent the observation value for each respective month. This histogram is multimodal with troughs roughly every ten months followed by inclines.

The green bars represent the observation value for each respective month.

```
In [ ]: df_FossilOil_Log.describe(include = 'all') # Description Tables
```

```
Out[ ]:
```

	Price_Log	Fossil_Oil_Log	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	-0.048185	-0.978728
std	4.072442	1.473832	1.101007	1.940811
min	14.539966	55.368933	-2.510638	-4.829053
25%	22.001649	56.830382	-0.488471	-2.024301
50%	25.195447	57.529275	-0.117543	-0.876921
75%	27.317301	58.752549	0.703437	0.534491
max	33.035963	61.649316	2.509858	1.957089

```
In [ ]: print(FossilOil_ypred)

print(ypred)

Rounded_ypred = [round(item, 2) for item in ypred]
print(Rounded_ypred)
#Rounded Price
```

```
[57.76733268 56.4789898 56.4737367 58.15671299 56.8812611 56.47955798
70.9593487 56.33389956 59.74259823 56.33729429 60.86601843 56.65009444
57.8139044 58.68617772 58.91233553 50.21801966 58.04965378 59.05675515
59.01974913 58.81197102 58.60567134 57.85964143 56.33064954 57.78094041
58.33793264 59.02652586 58.16850564 51.3164425 59.01012722 58.49289521
57.51346653 57.74229099 57.21864497 59.04078779 58.05277555 58.91941834
58.980158 58.79217109 59.05379563 48.51471951 58.47982965 58.946745
58.6253621 58.81438244 58.77473301 59.02939873 56.79970016 55.34944541]
[27.18807769 23.22554817 22.87823628 24.05511741 23.60270863 27.72472243
30.58783893 26.70248166 24.90180391 24.51841487 25.11912136 25.67229968
19.54636257 17.6334062 17.64405232 17.07534443 17.32825464 19.7366291
20.09354637 20.12336851 21.00799158 24.86873542 26.00005506 28.59930479
36.03506727 24.71214951 21.19464431 21.45617264 22.20151115 23.17431072
22.77134503 22.31923277 22.99576214 26.66578905 27.43836997 27.29282952
23.278006 25.18891499 20.32804924 21.00637179 25.54459336 26.87991763
28.70309766 30.1704781 34.27227994 30.16948579 28.0708618 28.29733761]
[27.19, 23.23, 22.88, 24.06, 23.6, 27.72, 30.59, 26.7, 24.9, 24.52, 25.12, 25.67, 19.55, 17.63, 17.64, 17.08, 17
.33, 19.74, 20.09, 20.12, 21.01, 24.87, 26.0, 28.6, 36.04, 24.71, 21.19, 21.46, 22.2, 23.17, 22.77, 22.32, 23.0,
26.67, 27.44, 27.29, 23.28, 25.19, 20.33, 21.01, 25.54, 26.88, 28.7, 30.17, 34.27, 30.17, 28.07, 28.3]
```

```
In [ ]: print(list(FossilOil_ypred))
```

```
[57.76733267539112, 56.47898979986485, 56.47373670241859, 58.15671299207497, 56.88126110318717, 56.4795579839092
2, 70.9593486998865, 56.333899555701464, 59.74259822586828, 56.33729428573042, 60.86601842627351, 56.65009444443
422, 57.813904403153174, 58.686177724237496, 58.9123355258007, 50.21801966432395, 58.049653775917704, 59.0567551
4930499, 59.01974913145523, 58.81197101561247, 58.605671341899324, 57.85964143434967, 56.33064953535677, 57.7809
4041238046, 58.33793264155565, 59.026525859551384, 58.1685056371748, 51.316442504021325, 59.01012721676443, 58.4
9289521060473, 57.51346652965185, 57.742290987514025, 57.2186449742062, 59.04078779136853, 58.05277555240764, 58
.91941834315958, 58.98015800173971, 58.79217109417732, 59.05379563309498, 48.51471951282383, 58.479829652519584,
58.946744997989754, 58.625362104561646, 58.81438244183755, 58.7747330071561, 59.029398727245734, 56.799700159377
61, 55.34944541371351]
```

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(ypred))
```

```
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338
71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672
299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486
, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506
2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20
1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,
27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905
53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857
92181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]:
```

```
In [ ]: dfFossilOil_Quad = {"Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895
    "Fossil_Oil_Quad" : [57.76733267539112, 56.47898979986485, 56.47373670241859, 58.15671299207497, 56.88126110318717
    print(dfFossilOil_Quad) #Dataframes
    df_FossilOil_Quad= pd.DataFrame.from_dict(dfFossilOil_Quad, orient = "columns")
    print(df_FossilOil_Quad)
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279493527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Fossil_Oil_Quad': [57.76733267539112, 56.4789879986485, 56.47373670241859, 58.15671299207497, 56.88126110318717, 56.47955798390922, 70.9593486998865, 56.333899555701464, 59.74259822586828, 56.33729428573042, 60.86601842627351, 56.65009444443422, 57.813904403153174, 58.686177724237496, 58.9123355258007, 50.21801966432395, 58.049653775917704, 59.05675514930499, 59.01974913145523, 58.81197101561247, 58.605671341899324, 57.85964143434967, 56.33064953535677, 57.78094041238046, 58.33793264155565, 59.026525859551384, 58.1685056371748, 51.316442504021325, 59.01012721676443, 58.49289521060473, 57.51346652965185, 57.742290987514025, 57.2186449742062, 59.04078779136853, 58.05277555240764, 58.91941834315958, 58.98015800173971, 58.79217109417732, 59.05379563309498, 48.51471951282383, 58.479829652519584, 58.946744997989754, 58.625362104561646, 58.81438244183755, 58.7747330071561, 59.029398727245734, 56.79970015937761, 55.34944541371351], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price_Quad Fossil_Oil_Quad Dates
0    27.188078      57.767333 2015-01
1    23.225548      56.478990 2015-02
2    22.878236      56.473737 2015-03
3    24.055117      58.156713 2015-04
4    23.602709      56.881261 2015-05
5    27.724722      56.479558 2015-06
6    30.587839      70.959349 2015-07
7    26.702482      56.333900 2015-08
8    24.901804      59.742598 2015-09
9    24.518415      56.337294 2015-10
10   25.119121      60.866018 2015-11
11   25.672300      56.650094 2015-12
12   19.546363      57.813904 2016-01
13   17.633406      58.686178 2016-02
14   17.644052      58.912336 2016-03
15   17.075344      50.218020 2016-04
16   17.328255      58.049654 2016-05
17   19.736629      59.056755 2016-06
18   20.093546      59.019749 2016-07
19   20.123369      58.811971 2016-08
20   21.007992      58.605671 2016-09
21   24.868735      57.859641 2016-10
22   26.000055      56.330650 2016-11
23   28.599305      57.780940 2016-12
24   36.035067      58.337933 2017-01
25   24.712150      59.026526 2017-02
26   21.194644      58.168506 2017-03
27   21.456173      51.316443 2017-04
28   22.201511      59.010127 2017-05
29   23.174311      58.492895 2017-06
30   22.771345      57.513467 2017-07
31   22.319233      57.742291 2017-08
32   22.995762      57.218645 2017-09
33   26.665789      59.040788 2017-10
34   27.438370      58.052776 2017-11
35   27.292830      58.919418 2017-12
36   23.278006      58.980158 2018-01
37   25.188915      58.792171 2018-02
38   20.328049      59.053796 2018-03
39   21.006372      48.514720 2018-04
40   25.544593      58.479830 2018-05
41   26.879918      58.946745 2018-06
42   28.703098      58.625362 2018-07
43   30.170478      58.814382 2018-08
44   34.272280      58.774733 2018-09
45   30.169486      59.029399 2018-10
46   28.070862      56.799700 2018-11
47   28.297338      55.349445 2018-12
```

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
```

```
print("weak evidence against null hypothesis, indicating it is non-stationary ")

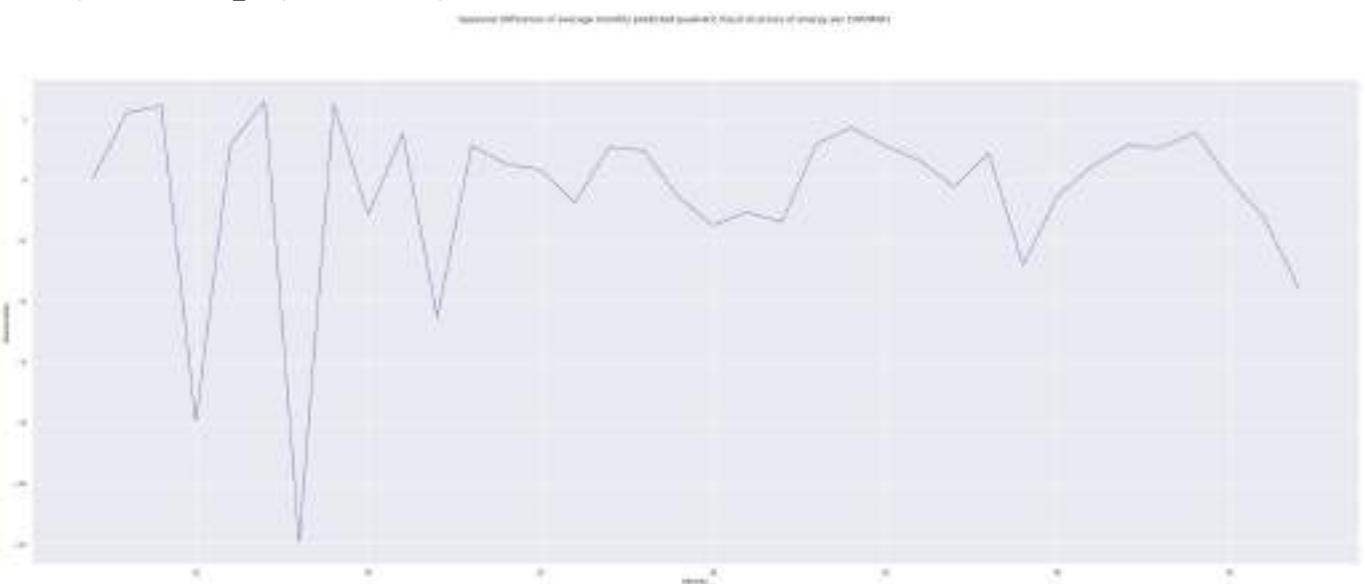
adfuller_test(df_FossilOil_Quad["Fossil_Oil_Quad"])

test_result=adfuller(df_FossilOil_Quad["Fossil_Oil_Quad"])
```

ADF Test Statistic : -7.920973459752062
p-value : 3.725157354382535e-12
#Lags Used : 0
Number of Observations : 47
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```
In [ ]: df_FossilOil_Quad['First Difference'] = df_FossilOil_Quad["Fossil_Oil_Quad"]- df_FossilOil_Quad["Fossil_Oil_Quad"]
df_FossilOil_Quad['Seasonal Difference']=df_FossilOil_Quad["Fossil_Oil_Quad"]- df_FossilOil_Quad["Fossil_Oil_Quad"]
plt.suptitle("Seasonal Difference of average monthly predicted quadratic fossil oil prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilOil_Quad.head() #Predictive Quadratic Seasonality Plot
df_FossilOil_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608d9f150>

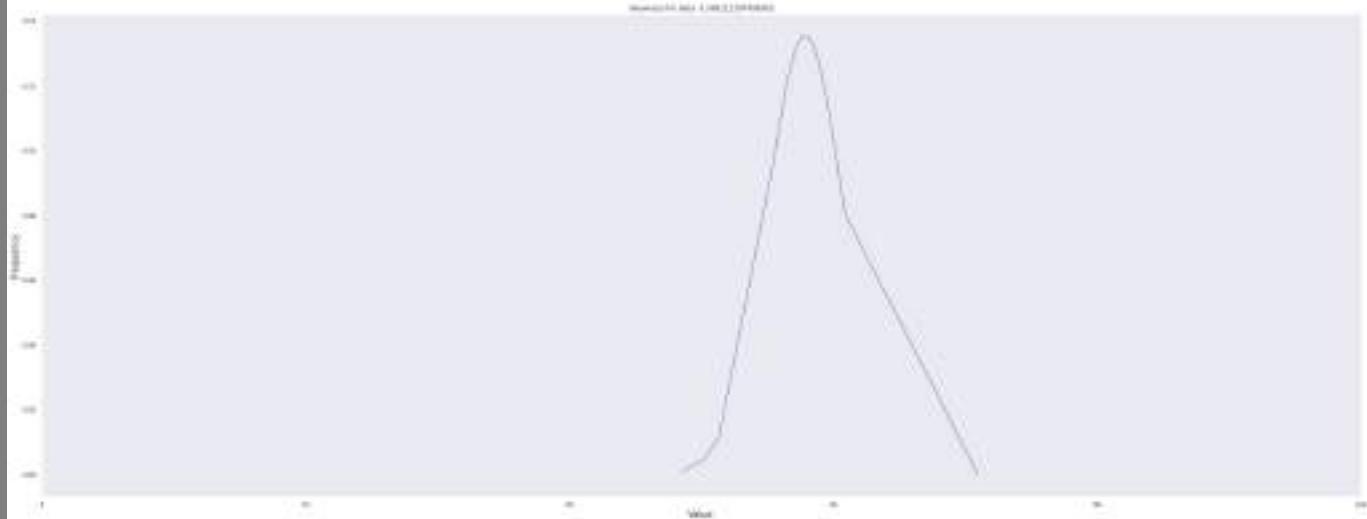


The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: FossilOil_QuadResults_mean = np.mean(df_FossilOil_Quad["Fossil_Oil_Quad"])
FossilOil_QuadResults_std = np.std(df_FossilOil_Quad["Fossil_Oil_Quad"])

FossilOil_QuadResultspdf = stats.norm.pdf(df_FossilOil_Quad["Fossil_Oil_Quad"].sort_values(), FossilOil_QuadResults_mean, FossilOil_QuadResults_std)

plt.plot(df_FossilOil_Quad["Fossil_Oil_Quad"].sort_values(), FossilOil_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilOil_Quad["Fossil_Oil_Quad"])}')
plt.suptitle("Frequency distribution of average monthly predicted quadratic fossil oil energy prices per EUR/MWH")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]:
print(dicDates)
Fossil_Oil_Quad_Dict = {key: i for i, key in enumerate(df_FossilOil_Quad["Fossil_Oil_Quad"])}
def Hist_Fossil_Oil_Quad(Fossil_Oil_Quad_Dict):
    for k, v in Fossil_Oil_Quad_Dict.items(): print(f"{v}:{k}")
print(Fossil_Oil_Quad_Dict)

plt.bar(list(Fossil_Oil_Quad_Dict.values()), Fossil_Oil_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly quadratic fossil oil energy prices per EUR/MWh")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{57.76733267539112: 0, 56.47898979986485: 1, 56.47373670241859: 2, 58.15671299207497: 3, 56.88126110318717: 4, 56.47955798390922: 5, 70.9593486998865: 6, 56.333899555701464: 7, 59.74259822586828: 8, 56.33729428573042: 9, 60.86601842627351: 10, 56.65009444443422: 11, 57.813904403153174: 12, 58.686177724237496: 13, 58.9123355258007: 14, 50.21801966432395: 15, 58.049653775917704: 16, 59.05675514930499: 17, 59.01974913145523: 18, 58.81197101561247: 19, 58.605671341899324: 20, 57.85964143434967: 21, 56.33064953535677: 22, 57.78094041238046: 23, 58.337932641555: 24, 59.026525859551384: 25, 58.1685056371748: 26, 51.316442504021325: 27, 59.01012721676443: 28, 58.49289521: 29, 57.51346652965185: 30, 57.742290987514025: 31, 57.2186449742062: 32, 59.04078779136853: 33, 58.05277: 34, 58.91941834315958: 35, 58.98015800173971: 36, 58.79217109417732: 37, 59.05379563309498: 38, 48.51: 471951282383: 39, 58.479829652519584: 40, 58.946744997989754: 41, 58.625362104561646: 42, 58.81438244183755: 43, 58.7747330071561: 44, 59.029398727245734: 45, 56.79970015937761: 46, 55.34944541371351: 47}
```



The green bars represent the observation value for each respective month. This histogram is unimodal yet uniform at the same time.

```
In [ ]: df_FossilOil_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Fossil_Oil_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	Nan	Nan	48	Nan	Nan
top	Nan	Nan	2015-01	Nan	Nan
freq	Nan	Nan	1	Nan	Nan
mean	24.500000	57.859845	Nan	-0.051444	-0.360178
std	4.174498	2.977889	Nan	4.588881	2.900386
min	17.075344	48.514720	Nan	-14.625449	-11.939600
25%	21.390791	56.860871	Nan	-0.983720	-1.086492
50%	24.615282	58.253219	Nan	-0.037006	0.488939
75%	27.214266	58.926250	Nan	0.869458	1.145957
max	36.035067	70.959349	Nan	14.479791	2.577197

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
```

```
plot_acf(FossilOil_Logpred)
```

```
plt.suptitle(" Autocorrelations of average monthly FossilOil Log")
```

```
plt.ylabel('Autocorrealtions')
```

```
plt.xlabel('Lags')
```

```
plt.show
```

```
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
```

```
plot_pacf(FossilOil_Logpred)
```

```
plt.suptitle("Partial Autocorrelations of average monthly FossilOil Log")
```

```
plt.ylabel('Partial autocorrealtions')
```

```
plt.xlabel('Lags')
```

```
plt.show
```

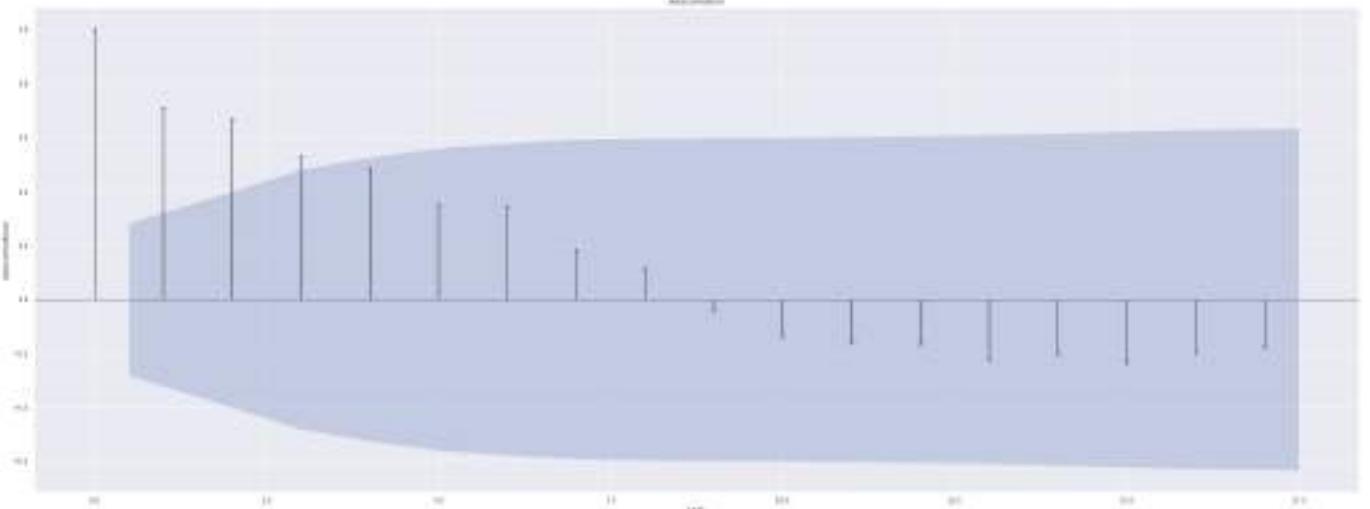
```
Fossiloil_Log_Autocorrelations = sm.tsa.acf(FossilOil_Logpred, fft=False) #Autocorrelations
```

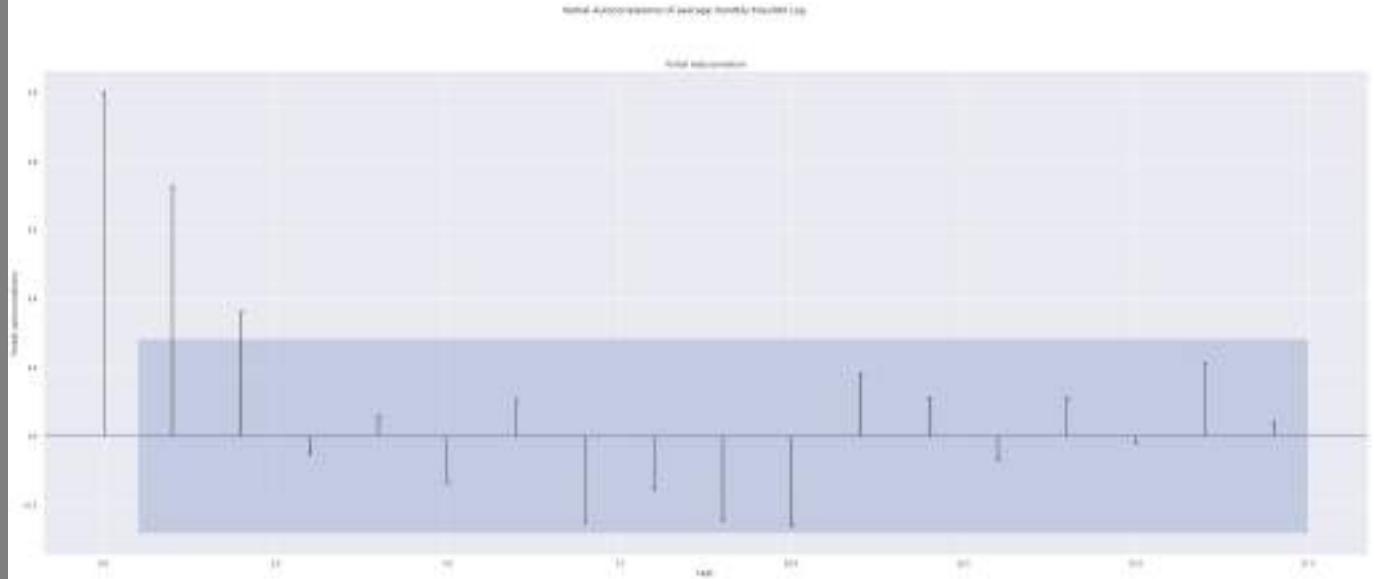
```
print(FossilOil_Log_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.70952824  0.66802752  0.52851278  0.48281801  0.3500967
 0.3432444   0.18192468  0.11216314 -0.04078178 -0.13797814 -0.16229098
-0.16583677 -0.22594728 -0.2037795  -0.23451912 -0.19590997 -0.17666594
-0.06884229 -0.04458249  0.03861587 -0.04748505  0.05017233  0.07068622
 0.10413612  0.07025746  0.05493628 -0.02032984 -0.02640435 -0.09210102
-0.11185708 -0.10870195 -0.13292692 -0.21194383 -0.20126559 -0.24983907
-0.24773538 -0.21829587 -0.18061395 -0.16915039 -0.1436068 ]
```





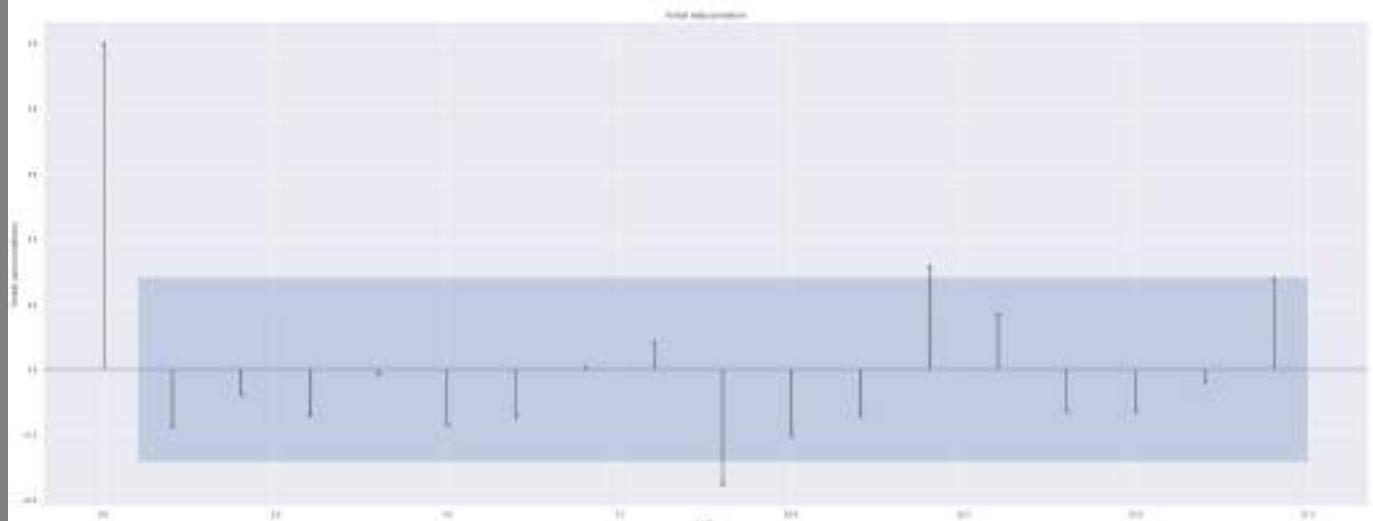
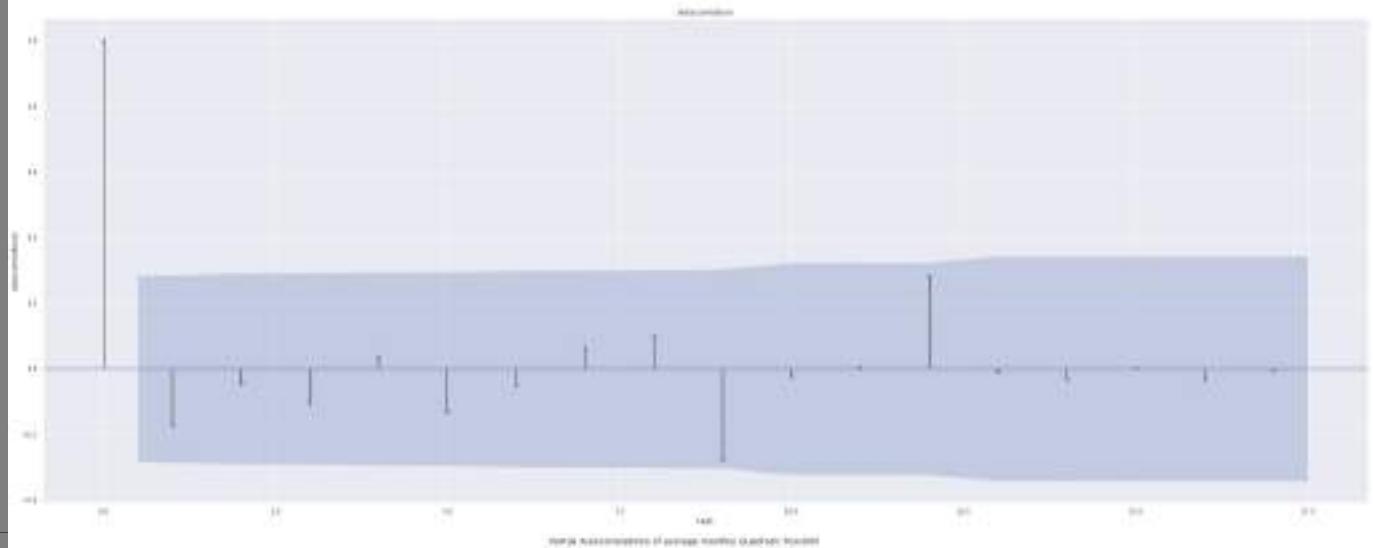
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilOil_ypred)
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilOil")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilOil_ypred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilOil")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
FossilOil_Quad_Autocorrelations = sm.tsa.acf(FossilOil_ypred, fft=False) #Autocorrelations
print(FossilOil_Quad_Autocorrelations)
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.0000000e+00 -1.69774560e-01 -4.42600488e-02 -1.04701236e-01
 3.44679492e-02 -1.30927697e-01 -5.08113849e-02  6.32138162e-02
 1.00079245e-01 -2.76620993e-01 -2.29045509e-02  2.62990601e-03
 2.80508694e-01 -1.06618714e-02 -3.33242164e-02  2.26397838e-03
-3.45101567e-02 -5.72346039e-03  4.17032032e-03  2.77039109e-03
 4.39664758e-02 -2.46388762e-01  7.12110106e-04  1.68645991e-02
 1.55992979e-01  1.46753200e-02 -3.32732037e-02  2.84229929e-02
 1.74941547e-02 -6.50876896e-02  4.71068581e-02  1.18850035e-03
 1.09514870e-01 -3.07981927e-01  3.53516246e-02  5.78087827e-02
 1.04585210e-03  3.99712737e-02  5.33042979e-02  2.09796439e-02
-3.57548479e-02]
```



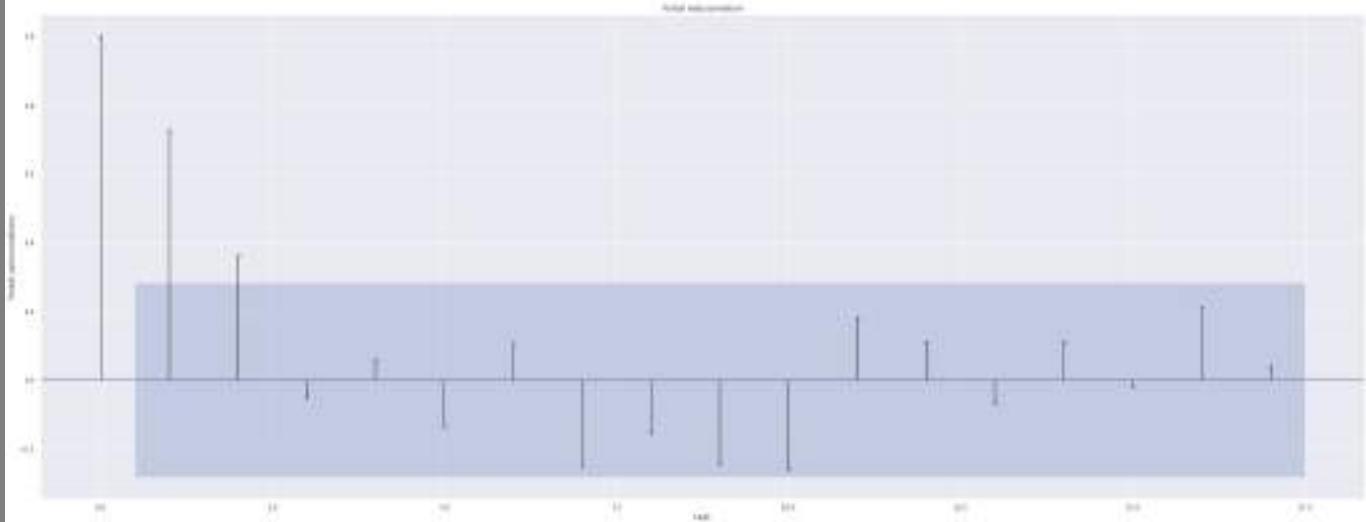
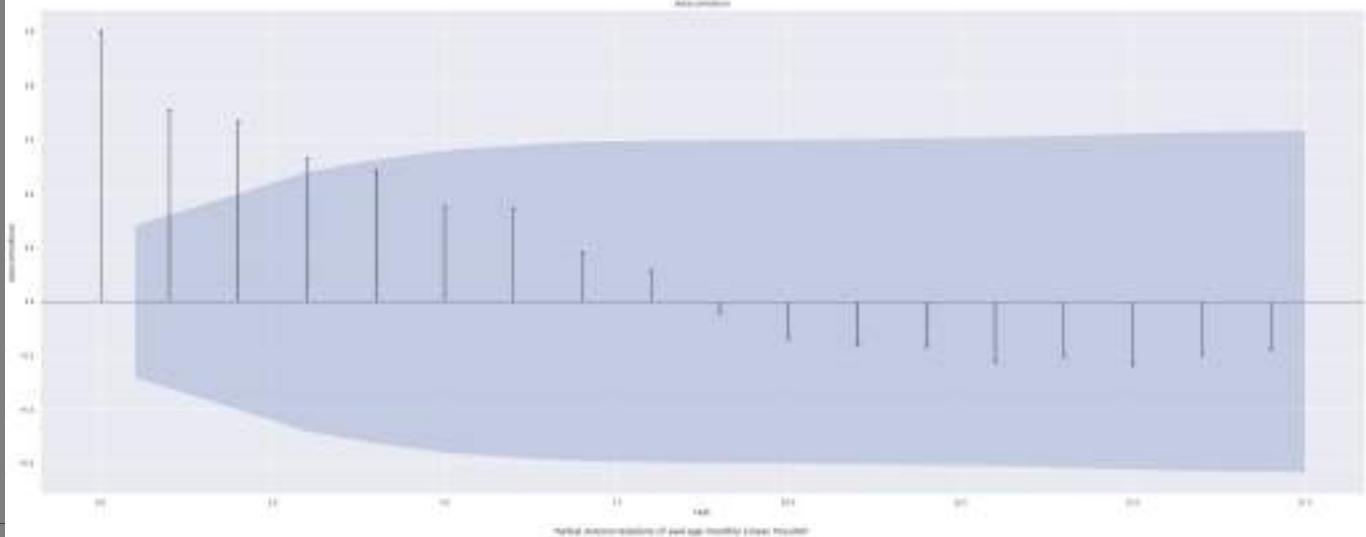
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(predictionsFossilOil)
plt.suptitle(" Autocorrelations of average monthly Linear FossilOil")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show()
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(predictionsFossilOil)
plt.suptitle("Partial Autocorrelations of average monthly Linear FossilOil")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show()
FossilOil_Pred_Autocorrelations = sm.tsa.acf(predictionsFossilOil, fft=False) #Autocorrelations
print(FossilOil_Pred_Autocorrelations)
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.      0.70952824  0.66802752  0.52851278  0.48281801  0.3500967
 0.3432444  0.18192468  0.11216314 -0.04078178 -0.13797814 -0.16229098
-0.16583677 -0.22594728 -0.2037795 -0.23451912 -0.19590997 -0.17666594
-0.06884229 -0.04458249  0.03861587 -0.04748505  0.05017233  0.07068622
 0.10413612  0.07025746  0.05493628 -0.02032984 -0.02640435 -0.09210102
-0.11185708 -0.10870195 -0.13292692 -0.21194383 -0.20126559 -0.24983907
-0.24773538 -0.21829587 -0.18061395 -0.16915039 -0.1436068 ]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resource analyzed is brown fossil coal.

```
In [ ]: #Dataframes analyzed by resource
dffossilreg = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.2940591,
 "Fossil_Brown" : [572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.280913978,
 dffossilreg["Dates"] += ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08'],
 print(dffossilreg)
 df_FossilBrown= pd.DataFrame.from_dict(dffossilreg, orient = "columns")
 print(df_FossilBrown)
 df_FossilBrown[ "Ratio" ] = df_FossilBrown[ "Fossil_Brown" ]/df_FossilBrown[ "Price" ]
 pdToList = list(df_FossilBrown[ "Ratio" ])
 print(pdToList)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.08432795698924, 55.81655555555556, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Brown': [572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.2809139784946, 665.162726008345, 684.2204301075269, 585.7674731182796, 548.0833333333334, 528.0188172043011, 695.284722222222, 493.480484522073, 417.9274193548387, 191.66522988505747, 173.20323014804845, 143.57083333333333, 179.002688172043, 175.6, 398.8156123822342, 464.3736559139785, 473.7208333333336, 613.7691275167786, 649.545833333333, 670.7956989247311, 688.6451612903226, 603.4151785714286, 335.66756393001344, 420.115277777778, 500.77016129032256, 478.8111111111113, 650.6774193548387, 511.8467741935484, 642.630555555556, 561.3221476510067, 667.647222222222, 476.5430107526882, 379.56451612903226, 406.95089285714283, 124.41588156123822, 133.97916666666666, 307.7043010752688, 333.9986111111113, 506.36877523553164, 300.14247311827955, 558.16111111112, 405.83758389261743, 406.22849462365593, 375.7361111111111], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price Fossil_Brown Dates
0 64.949019 572.851296 2015-01
1 56.383854 313.418155 2015-02
2 55.522463 244.437416 2015-03
3 58.354083 463.119777 2015-04
4 57.294059 374.280914 2015-05
5 65.974903 665.162726 2015-06
6 71.072043 684.220430 2015-07
7 63.998065 585.767473 2015-08
8 60.254792 548.083333 2015-09
9 59.406765 528.018817 2015-10
10 60.726792 695.284722 2015-11
11 61.901761 493.480485 2015-12
12 45.578723 417.927419 2016-01
13 36.752083 191.665230 2016-02
14 36.818008 173.203230 2016-03
15 32.618667 143.570833 2016-04
16 34.691371 179.002688 2016-05
17 46.266319 175.600000 2016-06
18 47.502016 398.815612 2016-07
19 47.602339 464.373656 2016-08
20 50.405597 473.720833 2016-09
21 60.182430 613.769128 2016-10
22 62.581056 649.545833 2016-11
23 67.595134 670.795699 2016-12
24 79.492083 688.645161 2017-01
25 59.837798 603.415179 2017-02
26 50.959892 335.667564 2017-03
27 51.717917 420.115278 2017-04
28 53.772621 500.770161 2017-05
29 56.258222 478.811111 2017-06
30 55.252581 650.677419 2017-07
31 54.084328 511.846774 2017-08
32 55.816556 642.630556 2017-09
33 63.925289 561.322148 2017-10
34 65.430653 667.647222 2017-11
35 65.151277 476.543011 2017-12
36 56.511976 379.564516 2018-01
37 60.877098 406.950893 2018-02
38 48.279717 124.415882 2018-03
39 50.400736 133.979167 2018-04
40 61.633763 307.704301 2018-05
41 64.348139 333.998611 2018-06
42 67.783441 506.368775 2018-07
43 70.363911 300.142473 2018-08
44 76.914042 558.161111 2018-09
45 70.362215 405.837584 2018-10
46 67.042608 406.228495 2018-11
47 66.623514 375.736111 2018-12
[8.820014628025667, 5.558650776789095, 4.402495903953125, 7.936373098576781, 6.532630426225016, 10.082056933813183, 9.62713890191688, 9.152893568689914, 9.096095400434958, 8.888193395306352, 11.449390016167584, 7.971994310368259, 9.169353390403051, 5.215084765309653, 4.704307462627643, 4.401493010137345, 5.15986204000096, 3.795417532852522, 8.395761798802605, 9.755269772482263, 9.398179159446304, 10.198477068938715, 10.37927257006247, 9.923727569944232, 8.663065960954047, 10.084180945512422, 6.586897039869847, 8.123205744839126, 9.312734850524272, 8.510953460629953, 11.776416807272176, 9.46386492221178, 11.513260701225642, 8.780909089763766, 10.203890590695366, 7.314407845265, 6.716532393576297, 6.684794525269369, 2.5769803213273477, 2.6582779737840028, 4.992463284681974, 5.190493724889739, 7.470390538004405, 4.265574036666141, 7.256946833324577, 5.767834131536049, 6.059258576134178, 5.639692192425388]
```

In []: modelFossilBrowncoal = stats.linregress([572.8512960436562, 313.41815476190476, 244.43741588156124, 463.119777164.9490188172043,

```
56.383854166666666,
55.522462987886975,
58.354083333333333,
57.29405913978498,
65.9749027777778,
71.07204301075271,
63.99806451612899,
60.254791666666634,
59.40676510067113,
60.72679166666668,
61.901760752688226,
45.57872311827956,
36.75208333333374,
36.81800807537014,
32.61866666666666,
34.691370967741896,
46.266319444444434,
47.50201612903221,
47.6023387096774,
50.4055972222224,
60.182429530201404,
62.58105555555558,
67.5951344086021,
79.49208333333331,
59.83779761904767,
50.95989232839837,
51.71791666666662,
53.77262096774189,
56.2582222222224,
55.252580645161316,
54.08432795698931,
55.81655555555558,
63.92528859060403,
65.43065277777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881])
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly fossil brown coal outputs versus the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Linear OLS regression
fossil_brown1 = fossil_brown

fossil_brown1 = sm.add_constant(fossil_brown1)
modelFossilBrownreg = sm.OLS(Price_Actual, fossil_brown1).fit()
predictionsFossilBrown = modelFossilBrownreg.predict(Price_Actual1)

modelFossilBrownreg.summary()
#OLS Linear Summary Table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.408			
Model:	OLS	Adj. R-squared:	0.395			
Method:	Least Squares	F-statistic:	31.73			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	1.03e-06			
Time:	05:21:22	Log-Likelihood:	-167.05			
No. Observations:	48	AIC:	338.1			
Df Residuals:	46	BIC:	341.8			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	40.2219	3.339	12.047	0.000	33.501	46.943
x1	0.0394	0.007	5.633	0.000	0.025	0.053
	Omnibus:	1.572	Durbin-Watson:	0.516		
Prob(Omnibus):	0.456	Jarque-Bera (JB):	1.395			
Skew:	0.273	Prob(JB):	0.498			
Kurtosis:	2.367	Cond. No.	1.38e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.38e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

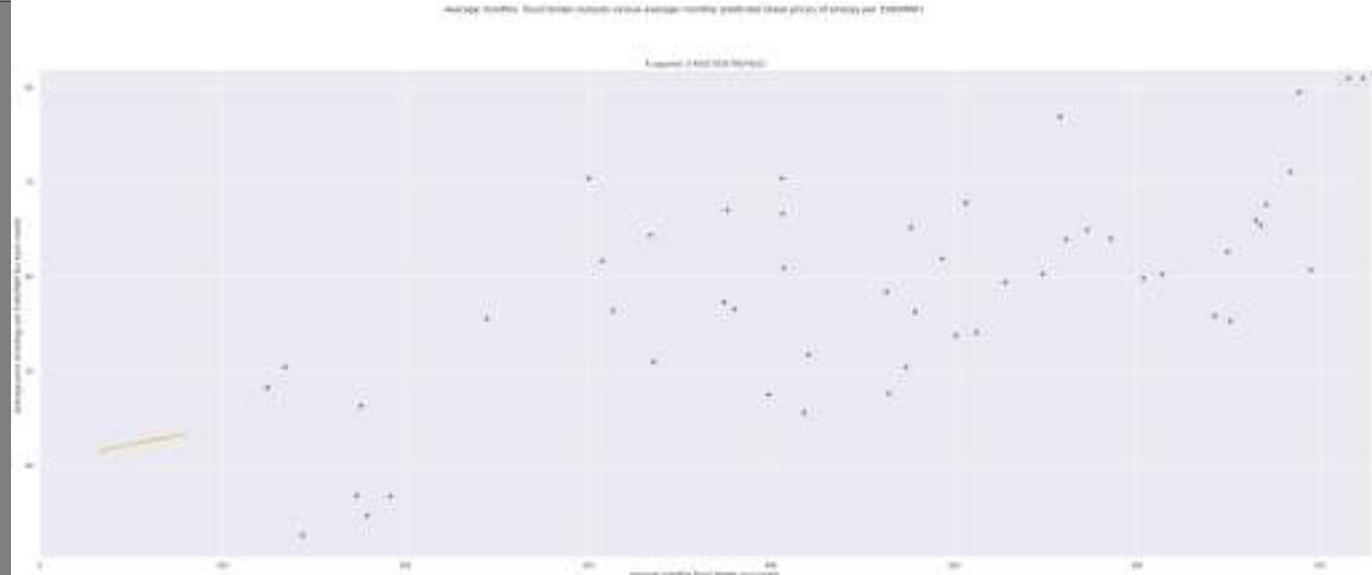
```
modelFossilBrowncoal = stats.linregress([572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977719,
[64.9490188172043,
56.383854166666666,
55.522462987886975,
58.35408333333333,
57.29405913978498,
65.9749027777778,
71.07204301075271,
63.99806451612899,
60.254791666666634,
59.40676510067113,
60.72679166666668,
61.901760752688226,
45.57872311827956,
36.75208333333374,
36.81800807537014,
32.61866666666666,
34.691370967741896,
46.266319444444434,
47.50201612903221,
47.6023387096774,
50.4055972222224,
60.182429530201404,
62.58105555555558,
67.5951344086021,
79.4920833333331,
59.83779761904767,
50.95989232839837,
51.71791666666662,
53.77262096774189,
56.2582222222224,
55.252580645161316,
54.08432795698931,
55.81655555555558,
63.92528859060403,
65.43065277777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
```

```
[67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881])
```

```
In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(fossil_brown,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(fossil_brown,Price_Actual, "o")
f = lambda x: 0.039383 *x + 40.221857
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average monthly fossil brown outputs versus average monthly predicted linear prices of energy per EUR/MWH")
plt.legend('#')
plt.title(f"R squared: {modelFossilBrownreg.rvalue**2}")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly fossil brown coal output')
plt.show()
```

```
slope: 0.039383      intercept: 40.221857
```



There is a moderately positive correlation between the average monthly outputs of brown fossil coal and their respective average monthly prices of energy per EUR/MWH. It appears that as the average monthly price of energy per EUR/MWH increases, the average monthly outputs of brown fossil coal increase as well. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: print(predictionsFossilBrown)
#Linear OLS Predicted Values
```

```
[42.77973086 42.44241073 42.40848674 42.52000386 42.47825715 42.82013304
43.02087267 42.74227962 42.59485907 42.56146141 42.61344775 42.65972132
42.01687446 41.66925671 41.67185301 41.50647121 41.5881001 42.04395393
42.09261912 42.0965701 42.20697026 42.59200925 42.68647384 42.88394229
43.3524774 42.57843668 42.22879995 42.25865307 42.33957307 42.43746299
42.39785801 42.35184896 42.42006892 42.7394135 42.79869895 42.78769635
42.44745652 42.61936724 42.12324717 42.20677881 42.64916683 42.75606653
42.89135832 42.99298446 43.25094691 42.99291765 42.84567716 42.86218223]
```

```
In [ ]: #Linear OLS regression residuals
influencefossilreg = modelFossilBrownreg.get_influence()

standardized_residualsFossilBrown = influencefossilreg.resid_studentized_internal

print(standardized_residualsFossilBrown)
```

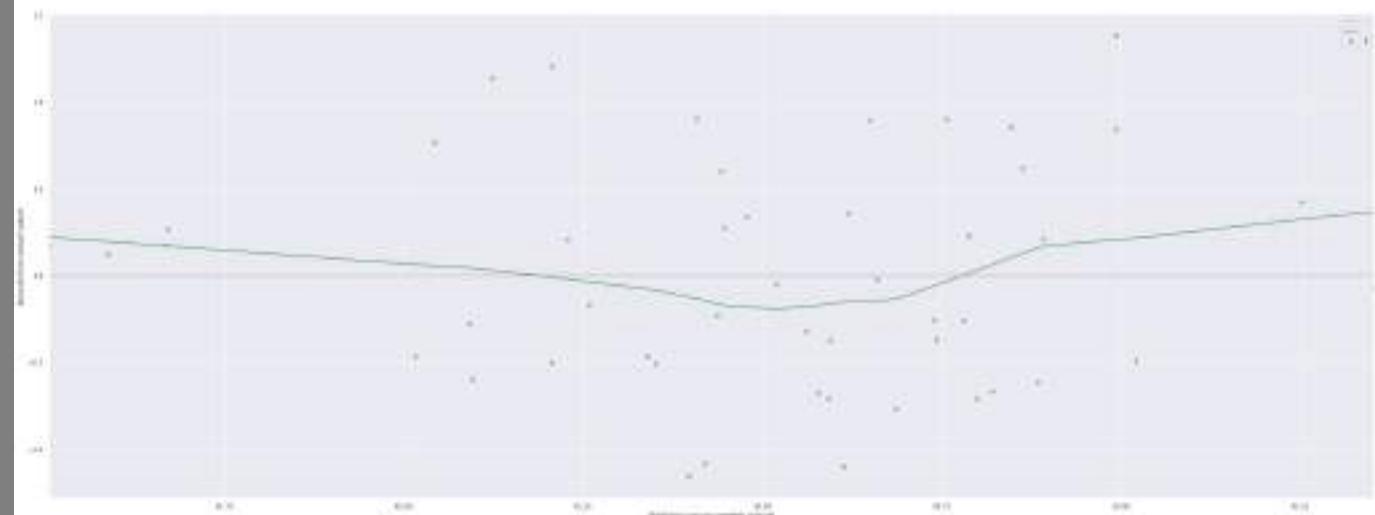
```
[ 0.27449969  0.48426507  0.72622188 -0.01344027  0.29426516 -0.05682334
 0.50256118  0.08969648 -0.19621151 -0.20323495 -0.88731681  0.28295657
 -1.39850883 -1.42412307 -1.32694542 -1.73274324 -1.63045272 -0.11299218
 -1.06204996 -1.37368487 -1.06717828 -0.53605465 -0.41223875  0.12270134
 1.56546771 -0.52732363 -0.31401039 -0.63600258 -0.77790547 -0.35530218
 -1.35590907 -0.7940019 -1.2415852  0.20210812 -0.13926004  0.77615744
 0.16926948  0.5831935  0.4148406  0.64233649  1.17928938  1.38867383
 0.96072222  2.32685837  1.86114154  1.78394353  1.51705007  1.3108793 ]
```

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionsFossilBrown, y= standardized_residualsFossilBrown, lowess = True, color="g")
```

```
plt.suptitle("fossil brown coal residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#")
```

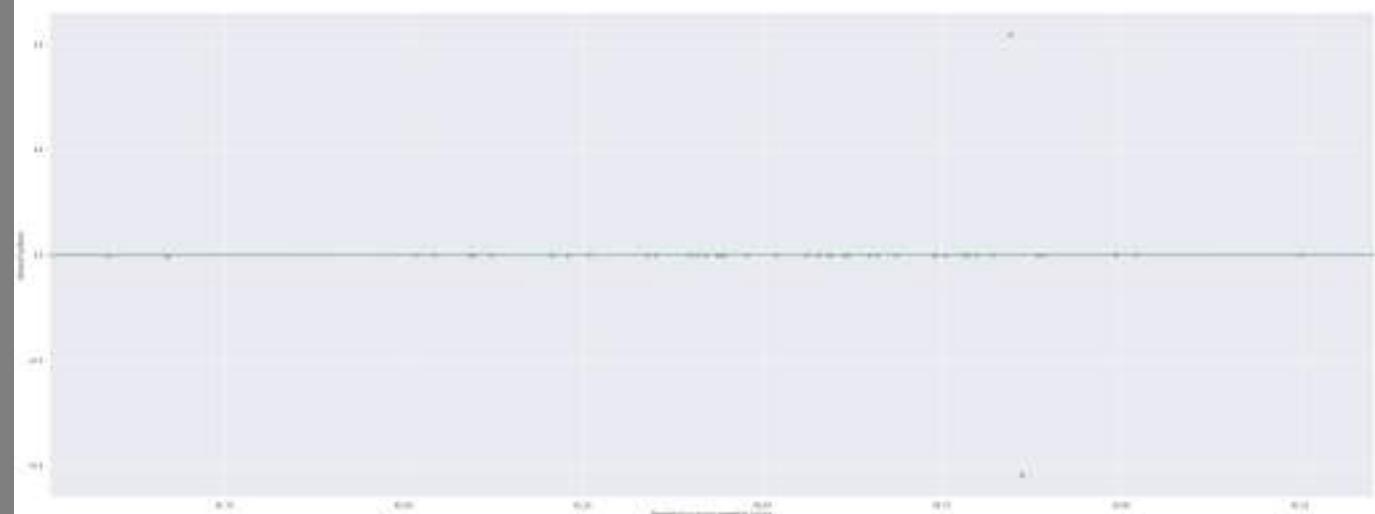
Out[]: <matplotlib.legend.Legend at 0x7ff60b9ba310>



As one can observe this residual plot, one may notice that the lowess line is subtle and that the residuals are spread out; indicating homoscedasticity, a lack of bias, and constant variance. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly linear fossil brown coal energy prices per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#")
sns.residplot(x = predictionsFossilBrown, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6088e0b10>



With the exception of a few outliers, the predictions seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: print(fossil_brown)
```

```
[572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.2809139784946, 665.162726008  
345, 684.2204301075269, 585.7674731182796, 548.083333333334, 528.0188172043011, 695.284722222222, 493.48048452  
22073, 417.9274193548387, 191.66522988505747, 173.20323014804845, 143.57083333333333, 179.002688172043, 175.6, 3  
98.8156123822342, 464.3736559139785, 473.72083333333336, 613.7691275167786, 649.5458333333333, 670.7956989247311  
, 688.6451612903226, 603.4151785714286, 335.66756393001344, 420.11527777777778, 500.77016129032256, 478.811111111  
11113, 650.6774193548387, 511.8467741935484, 642.6305555555556, 561.3221476510067, 667.647222222222, 476.543010  
7526882, 379.56451612903226, 406.95089285714283, 124.41588156123822, 133.97916666666666, 307.7043010752688, 333.  
99861111111113, 506.36877523553164, 300.14247311827955, 558.161111111112, 405.83758389261743, 375.7361111111111  
, 406.22849462365593]
```

This is the average monthly logarithmic outputs of fossil brown coal the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions  
Logpricevalues = ((np.log(Price_Actual)))  
LogFossilBrownvalues = ((np.log(fossil_brown)))  
Log = np.polyfit(np.log(Price_Actual), fossil_brown1, 1)  
lin2 = LinearRegression()  
lin2.fit(np.log(fossil_brown1), Price_Actual)  
FossilBrown_Log = sm.OLS(Price_Actual, fossil_brown1).fit()  
  
FossilBrown_Logpred = FossilBrown_Log.predict(fossil_brown1)  
#OLS Logarithmic summary table  
FossilBrown_Log.summary()  
#Log  
Log = np.polyfit(np.log(fossil_brown), Price_Actual, 1)  
print(Log)  
  
y = 14.56449676 * LogFossilBrownvalues - 29.7139618  
  
#OLS Logarithmic Scatterplots  
plt.suptitle("Logarithmic average monthly outputs versus predicted average monthly prices of energy per EUR/MWH")  
plt.title("R squared : 0.408")  
plt.ylabel("Average monthly prices of energy per EUR/MWH")  
plt.xlabel("Average monthly outputs")  
plt.yscale("log")  
plt.xscale("log")  
plt.plot(LogFossilBrownvalues, Price_Actual, "o")  
plt.plot(LogFossilBrownvalues, y)  
  
plt.xlim([1, 7])
```

```
[ 14.56449676 -29.7139618 ]
```

```
Out[ ]: (1, 7)
```



The blue dots represent the observations and the orange line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: FossilBrown_Log.summary() #OLS Logarithmic summary table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.408			
Model:	OLS	Adj. R-squared:	0.395			
Method:	Least Squares	F-statistic:	31.73			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	1.03e-06			
Time:	05:21:26	Log-Likelihood:	-167.05			
No. Observations:	48	AIC:	338.1			
Df Residuals:	46	BIC:	341.8			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	40.2219	3.339	12.047	0.000	33.501	46.943
x1	0.0394	0.007	5.633	0.000	0.025	0.053
	Omnibus:	1.572	Durbin-Watson:	0.516		
	Prob(Omnibus):	0.456	Jarque-Bera (JB):	1.395		
	Skew:	0.273	Prob(JB):	0.498		
	Kurtosis:	2.367	Cond. No.	1.38e+03		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 [2] The condition number is large, 1.38e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceFossilBrownLog = FossilBrown_Log.get_influence()
#Logarithmic OLS regression residuals
```

```
standardized_residualsFossilBrownLog = influenceFossilBrownLog.resid_studentized_internal
```

```
print(standardized_residualsFossilBrownLog)
```

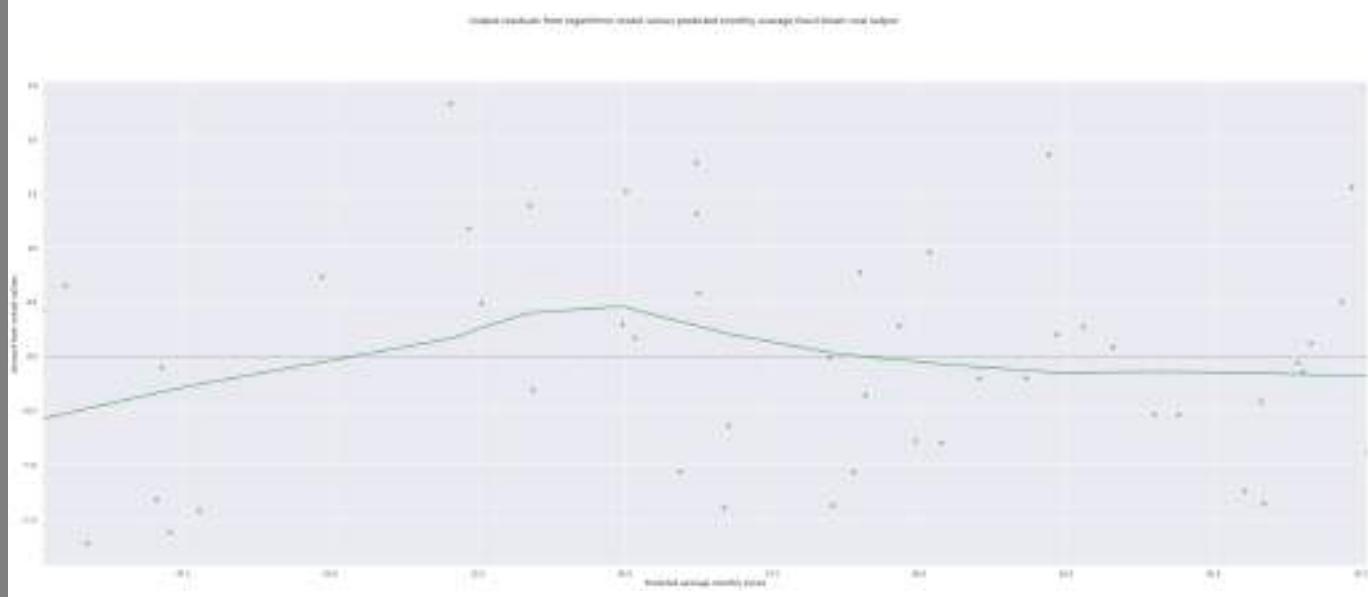
```
print(FossilBrown_Logpred) # OLS logarithmic predicted values
```

```
[ 0.27449969  0.48426507  0.72622188 -0.01344027  0.29426516 -0.05682334
 0.50256118  0.08969648 -0.19621151 -0.20323495 -0.88731681  0.28295657
-1.39850883 -1.42412307 -1.32694542 -1.73274324 -1.63045272 -0.11299218
-1.06204996 -1.37368487 -1.06717828 -0.53605465 -0.41223875  0.12270134
 1.56546771 -0.52732363 -0.31401039 -0.63600258 -0.77790547 -0.35530218
-1.35590907 -0.7940019 -1.2415852  0.20210812 -0.13926004  0.77615744
 0.16926948  0.5831935  0.4148406  0.64233649  1.17928938  1.38867383
 0.96072222  2.32685837  1.86114154  1.78394353  1.51705007  1.3108793 ]
[62.78234245 52.56514006 49.84848572 58.46080847 54.96208567 66.41782463
67.1683703 63.29101761 61.80691084 61.0167141 67.60411305 59.656498
56.68100707 47.7701695 47.04308434 45.87607771 47.27148321 47.13747584
55.92833069 58.51018972 58.8783077 64.39380105 65.80278774 66.63966685
67.34262858 63.98603359 53.441384 56.76717106 59.94358585 59.07877707
65.84735276 60.37981383 65.53044477 62.32829236 66.51567104 58.98945293
55.1701687 56.24872077 45.12170216 45.49833106 52.34011253 53.37565597
60.16407492 52.04230661 62.2038019 56.20487556 55.0193954 56.22027071]
```

In []:

```
plt.suptitle("Output residuals from logarithmic model versus predicted monthly average fossil brown coal output")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = FossilBrown_Logpred, y = standardized_residualsFossilBrownLog, lowess = True, color="g")
# OLS Logarithmic average monthly predictions versus residuals
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608aff2d0>

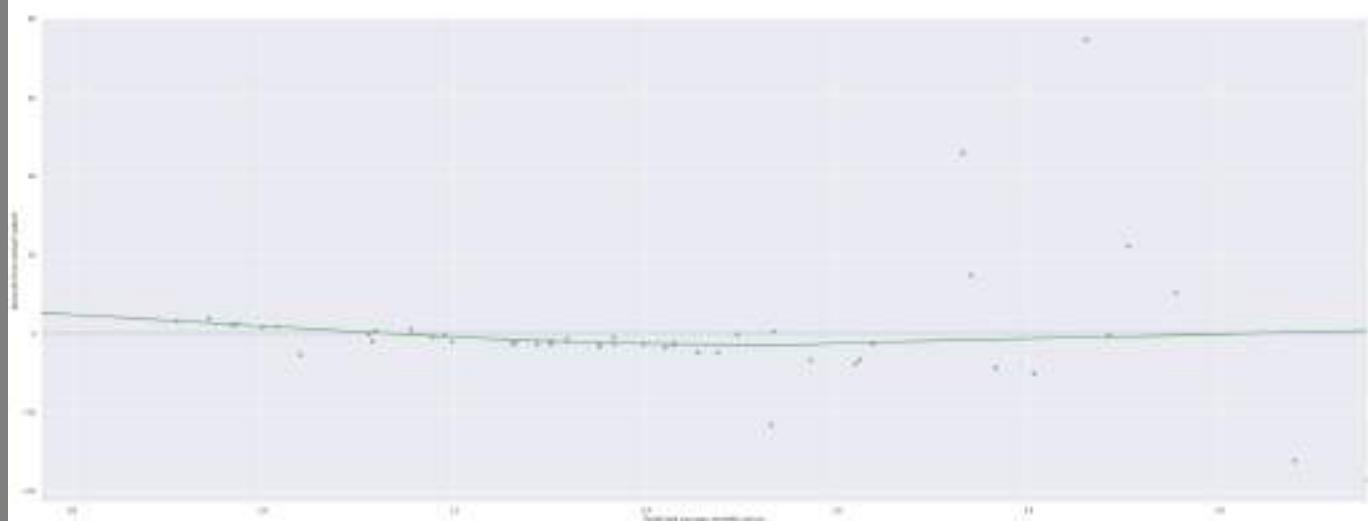


As one can observe this residual plot, one may notice the slight hump in the fitted model, which form a nonlinear pattern. The slight hump is too subtle to suggest a different model. In addition, the residuals are spread out without the distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: FossilBrownLogRatioPredict = FossilBrown_Logpred/Logpred
```

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
plt.suptitle("Predicted average monthly logarithmic fossil brown coal output to price of energy per EUR/MWh ratio")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Amount from actual values")  
plt.legend(..#)  
sns.residplot(x = FossilBrownLogRatioPredict, y = standardized_residualsFossilBrownLog/standardized_residualsPr:
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60899ad50>
```

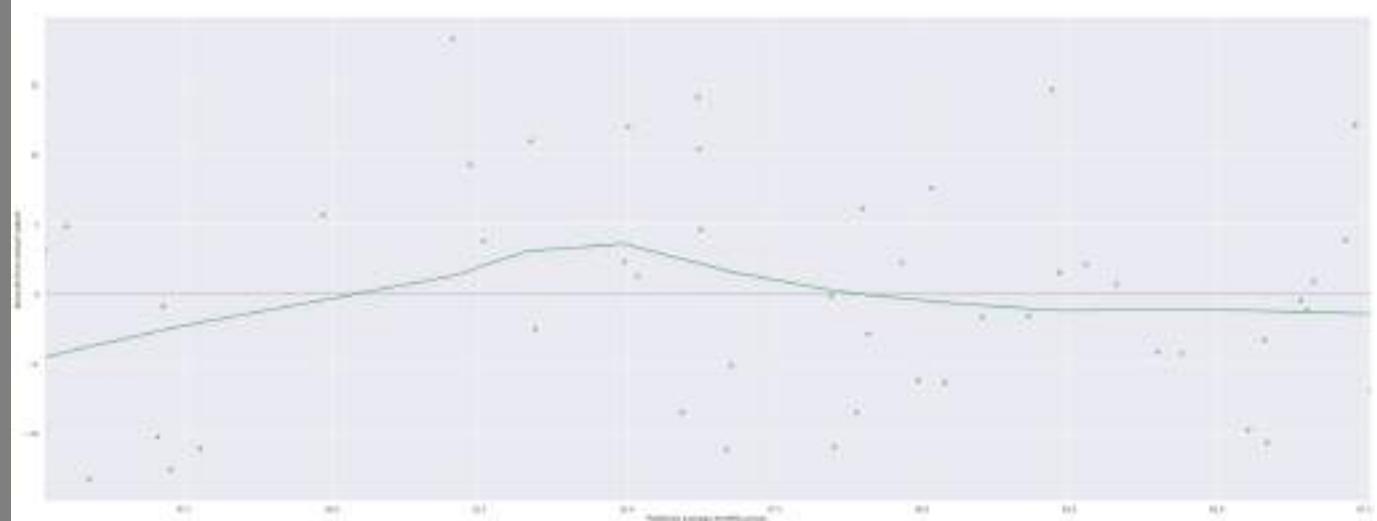


With the exception of a few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS predicted logarithmic average monthly values versus actual values
```

```
plt.suptitle("Predicted logarithmic monthly average fossil brown coal energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = FossilBrown_Logpred, y = Price_Actual, lowess = True, color="g")
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608d0b1d0>



As one can observe this residual plot, one may notice the positive slope in the fitted model. In addition, the observations are spread out in a distinct pattern, indicating bias and homoscedasticity. It would be reasonable to assume that this model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly fossil brown output versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)

modelFossilBrownquad = np.poly1d(np.polyfit(fossil_brown,Price_Actual,2))
print(modelFossilBrownquad)

fossil_brown1 = fossil_brown

fossil_brown1 = sm.add_constant(fossil_brown1)
fossil_brown2 = polynomial_features.fit_transform(fossil_brown1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(fossil_brown1)

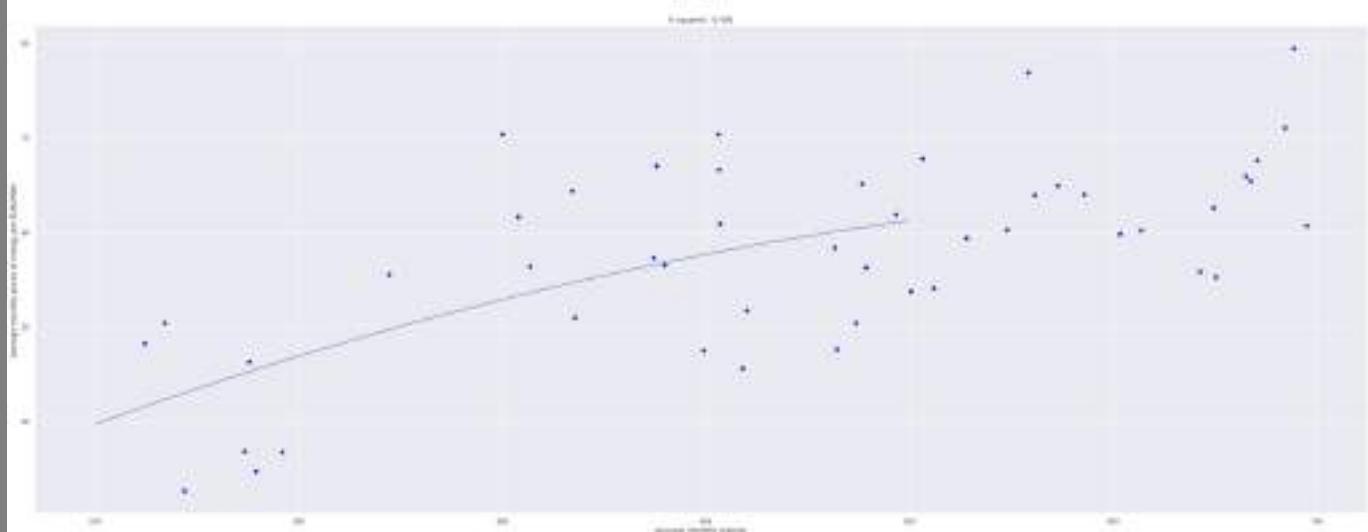
FossilBrown_Q = poly.fit(X_poly, fossil_brown)
lin2 = LinearRegression()
lin2.fit(X_poly, fossil_brown)
FossilBrown_Quad = sm.OLS(Price_Actual, fossil_brown2).fit()

# OLS Predicted Quadratic values
FossilBrown_ypred = FossilBrown_Quad.predict(fossil_brown2)

#OLS Quadratic Summary Table
FossilBrown_Quad.summary()

##OLS Quadratic Scatterplot
polyline = np.linspace(start = 100, stop =500 , num = 100)
plt.title("R squared : 0.436")
plt.plot(polyline, modelFossilBrownquad(polyline))
plt.scatter(fossil_brown,Price_Actual, color = 'blue')
plt.suptitle('Quadratic for predicted average monthly prices of energy per EUR/MWH versus average monthly fossil brown coal energy prices per EUR/MWH')
plt.xlabel('Average monthly outputs')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()
```

$$-6.058e-05 x^2 + 0.09019 x + 31.28$$



The blue dots represent the observations and the blue line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

In []: FossilBrown_Quad.summary()

Out[]: OLS Regression Results

Dep. Variable:	y	R-squared:	0.436
Model:	OLS	Adj. R-squared:	0.410
Method:	Least Squares	F-statistic:	17.36
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	2.58e-06
Time:	05:21:28	Log-Likelihood:	-165.92
No. Observations:	48	AIC:	337.8
Df Residuals:	45	BIC:	343.4
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	10.4265	2.298	4.537	0.000	5.798	15.055
x1	10.4265	2.298	4.537	0.000	5.798	15.055
x2	0.0451	0.018	2.570	0.014	0.010	0.080
x3	10.4265	2.298	4.537	0.000	5.798	15.055
x4	0.0451	0.018	2.570	0.014	0.010	0.080
x5	-6.058e-05	4.1e-05	-1.477	0.147	-0.000	2.2e-05

Omnibus:	2.025	Durbin-Watson:	0.550
Prob(Omnibus):	0.363	Jarque-Bera (JB):	1.569
Skew:	0.262	Prob(JB):	0.456
Kurtosis:	2.285	Cond. No.	1.89e+23

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 9.68e-35. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []: print(FossilBrown_ypred) # OLS quadratic predicted values

```
[63.06731283 53.59687265 49.70642367 60.05675062 56.55074829 64.47010715  
64.6311407 63.3257131 62.51526526 62.01355917 64.70444221 61.03569664  
58.39273582 46.3408971 45.08382205 42.97982311 45.48315497 45.24935033  
57.61457683 60.09939138 60.41125896 63.81650409 64.30534409 64.52228556  
64.66223446 63.64609726 54.72875591 58.47899399 61.25411764 60.5766436  
64.31827563 61.57368303 64.22294671 62.81958841 64.49359499 60.5033401  
56.78600786 57.9512235 41.56314623 42.27599184 53.29651552 54.64593282  
61.41749812 52.89293557 62.74885485 57.90562715 56.6158804 57.92165428]
```

```
In [ ]: influenceFossilBrownquad = FossilBrown_Quad.get_influence() #Quadratic OLS residuals
```

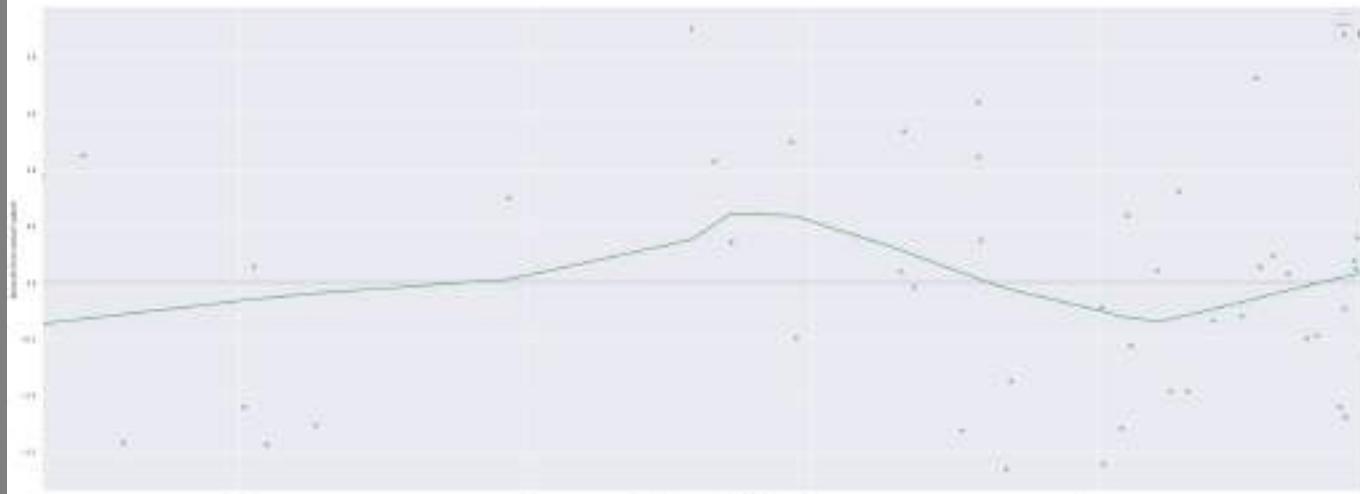
```
standardized_residualsFossilBrownquad = influenceFossilBrownquad.resid_studentized_internal  
#Quadratic OLS residuals  
print(standardized_residualsFossilBrownquad)
```

```
[ 0.24151186  0.35938576  0.75396049 -0.21925071  0.09590346  0.19845127  
 0.8616789  0.08638358 -0.28995122 -0.33451973 -0.53774368  0.11133079  
-1.65288886 -1.26539187 -1.10326914 -1.42015968 -1.43475754  0.13551199  
-1.30477666 -1.60912659 -1.28767612 -0.46906945 -0.22540135  0.40678445  
1.99188545 -0.4905027 -0.48604244 -0.87207374 -0.96133072 -0.5555973  
-1.18574592 -0.96176715 -1.09539151  0.14184633  0.12377999  0.59806951  
-0.0353574  0.37748177  0.94333107  1.12636439  1.07515631  1.25120023  
 0.81773583  2.25332744  1.8170791  1.60710913  1.34529197  1.12268077]
```

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals  
sns.residplot(x = FossilBrown_ypred, y = standardized_residualsFossilBrownquad, lowess = True, color="g")
```

```
plt.suptitle("fossil brown coal residuals from quadratic model versus predicted values")  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Amount from actual values")  
plt.legend(..#)
```

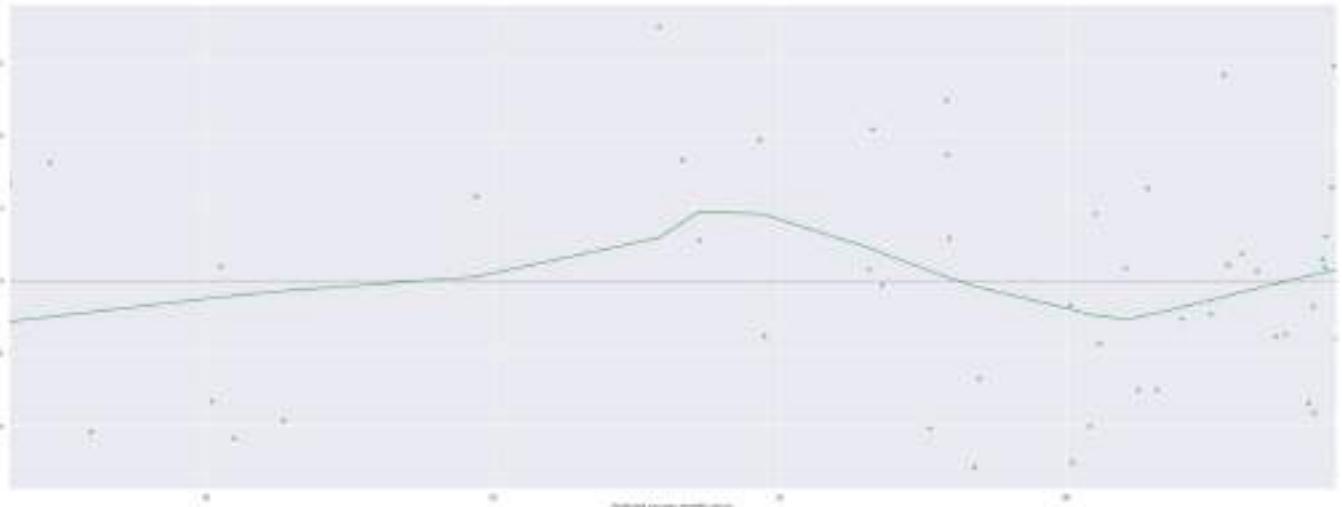
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff608f95910>
```



As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance and homoscedasticity. Given these circumstances, it would be reasonable to assume that the quadratic model of fit is suitable. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly quadratic fossil brown coal energy prices per EUR/MWH versus actual va  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend(..#)  
sns.residplot(x = FossilBrown_ypred, y = Price_Actual, lowess = True, color="g")  
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609064410>
```



As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance and homoscedasticity. Given these circumstances, it would be reasonable to assume that the quadratic model of fit is suitable. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The ratios from the list directly below will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective the average monthly prices of energy per EUR/MWH.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

```
In [ ]: #Dataframes analyzed by resource
dffossilreg = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.2940591,
 "Fossil_Brown" : [572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.280913978,
 dffossilreg["Dates"] += ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08'],
 print(dffossilreg),
 df_FossilBrown= pd.DataFrame.from_dict(dffossilreg, orient = "columns"),
 print(df_FossilBrown),
 df_FossilBrown["Ratio"] = df_FossilBrown["Fossil_Brown"]/df_FossilBrown["Price"],
 pdToList = list(df_FossilBrown["Ratio"]),
 print(pdToList),
 #ADF Tests
 from statsmodels.tsa.stattools import adfuller
 def adfuller_test(test_result):
 result=adfuller(test_result)
 labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
 for value,label in zip(result,labels):
 print(label+' : '+str(value) )
 if result[1] <= 0.05:
 print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
 else:
 print("weak evidence against null hypothesis, indicating it is non-stationary ")
 adfuller_test(df_FossilBrown["Ratio"])
 test_result=adfuller(df_FossilBrown["Ratio"])
 df_FossilBrown['First Difference'] = df_FossilBrown["Ratio"]- df_FossilBrown["Ratio"].shift(1) # Seasonality variable
 df_FossilBrown['Seasonal Difference']=df_FossilBrown["Ratio"]- df_FossilBrown["Ratio"].shift(12)
 df_FossilBrown.head()
 from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
 plot_acf(df_FossilBrown["Ratio"])
 plt.show()
 from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
```

```

plot_pacf(df_FossilBrown["Ratio"])
plt.show

df_FossilBrown['Seasonal Difference'].plot() # Seasonality Plot

```

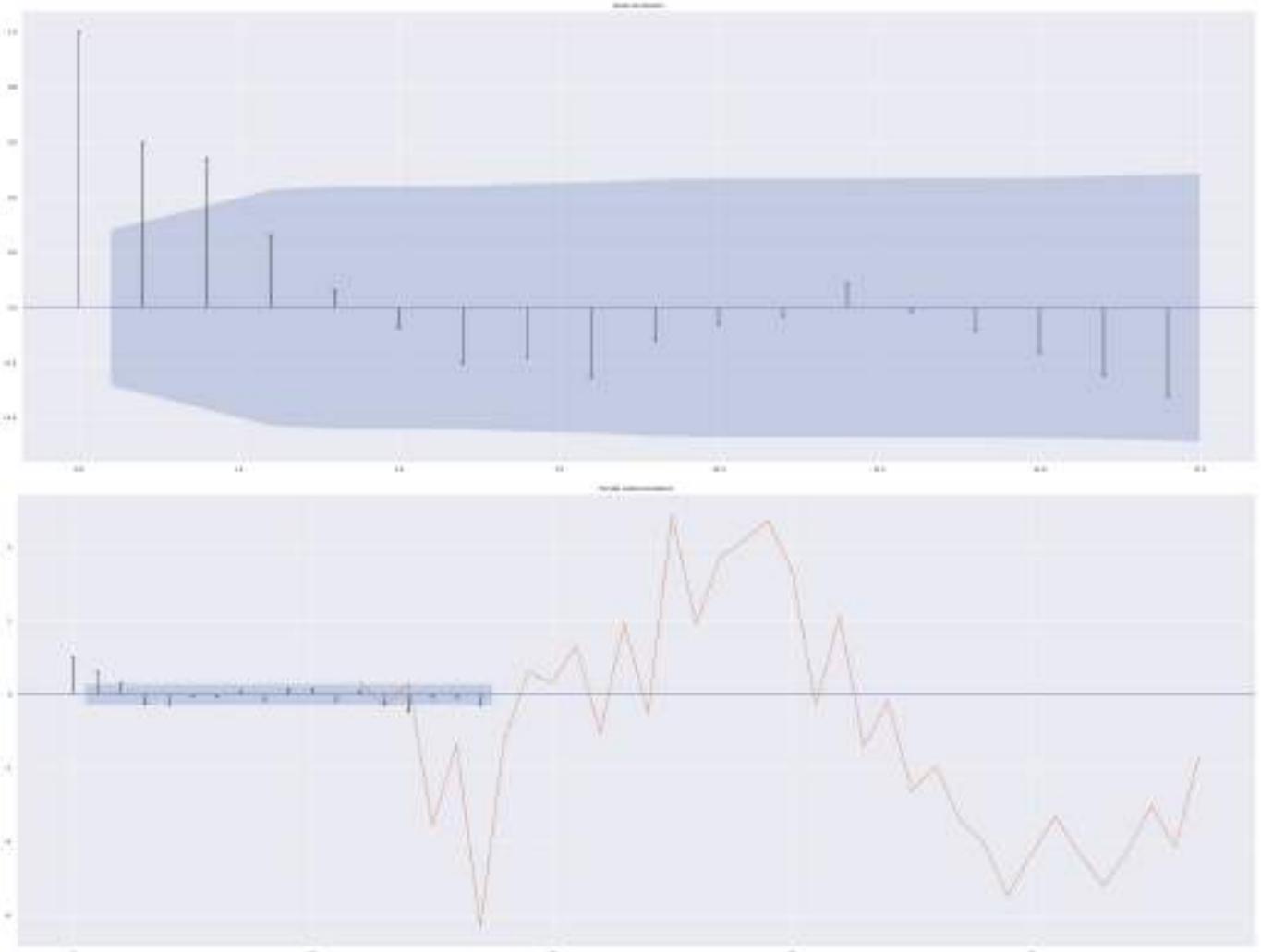
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Brown': [572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.2809139784946, 665.162726008345, 684.2204301075269, 585.7674731182796, 548.0833333333334, 528.0188172043011, 695.2847222222222, 493.4804845222073, 417.9274193548387, 191.66522988505747, 173.20323014804845, 143.5708333333333, 179.002688172043, 175.6, 398.8156123822342, 464.3736559139785, 473.7208333333336, 613.7691275167786, 649.545833333333, 670.7956989247311, 688.6451612903226, 603.4151785714286, 335.66756393001344, 420.115277777778, 500.77016129032256, 478.81111111111113, 650.6774193548387, 511.8467741935484, 642.6305555555556, 561.3221476510067, 667.647222222222, 476.5430107526882, 379.56451612903226, 406.95089285714283, 124.41588156123822, 133.97916666666666, 307.7043010752688, 333.99861111111113, 506.36877523553164, 300.14247311827955, 558.161111111112, 405.83758389261743, 406.22849462365593, 375.73611111111111], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

	Price	Fossil_Brown	Dates
0	64.949019	572.851296	2015-01
1	56.383854	313.418155	2015-02
2	55.522463	244.437416	2015-03
3	58.354083	463.119777	2015-04
4	57.294059	374.280914	2015-05
5	65.974903	665.162726	2015-06
6	71.072043	684.220430	2015-07
7	63.998065	585.767473	2015-08
8	60.254792	548.083333	2015-09
9	59.406765	528.018817	2015-10
10	60.726792	695.284722	2015-11
11	61.901761	493.480485	2015-12
12	45.578723	417.927419	2016-01
13	36.752083	191.665230	2016-02
14	36.818008	173.203230	2016-03
15	32.618667	143.570833	2016-04
16	34.691371	179.002688	2016-05
17	46.266319	175.600000	2016-06
18	47.502016	398.815612	2016-07
19	47.602339	464.373656	2016-08
20	50.405597	473.720833	2016-09
21	60.182430	613.769128	2016-10
22	62.581056	649.545833	2016-11
23	67.595134	670.795699	2016-12
24	79.492083	688.645161	2017-01
25	59.837798	603.415179	2017-02
26	50.959892	335.667564	2017-03
27	51.717917	420.115278	2017-04
28	53.772621	500.770161	2017-05
29	56.258222	478.811111	2017-06
30	55.252581	650.677419	2017-07
31	54.084328	511.846774	2017-08
32	55.816556	642.630556	2017-09
33	63.925289	561.322148	2017-10
34	65.430653	667.647222	2017-11
35	65.151277	476.543011	2017-12
36	56.511976	379.564516	2018-01
37	60.877098	406.950893	2018-02
38	48.279717	124.415882	2018-03
39	50.400736	133.979167	2018-04
40	61.633763	307.704301	2018-05
41	64.348139	333.998611	2018-06
42	67.783441	506.368775	2018-07
43	70.363911	300.142473	2018-08
44	76.914042	558.161111	2018-09
45	70.362215	405.837584	2018-10
46	67.042608	406.228495	2018-11
47	66.623514	375.736111	2018-12

[8.820014628025667, 5.558650776789095, 4.402495903953125, 7.936373098576781, 6.532630426225016, 10.082056933813183, 9.62713890191688, 9.152893568689914, 9.096095400434958, 8.888193395306352, 11.449390016167584, 7.971994310368259, 9.169353390403051, 5.215084765309653, 4.704307462627643, 4.401493010137345, 5.15986204000096, 3.795417532852522, 8.395761798802605, 9.755269772482263, 9.398179159446304, 10.198477068938715, 10.37927257006247, 9.92372756994232, 8.663065960954047, 10.084180945512422, 6.586897039869847, 8.123205744839126, 9.312734850524272, 8.510953460629953, 11.776416807272176, 9.46386492221178, 11.513260701225642, 8.780909089763766, 10.203890590695366, 7.314407845265, 6.716532393576297, 6.684794525269369, 2.5769803213273477, 2.6582779737840028, 4.992463284681974, 5.190493724889739, 7.470390538004405, 4.265574036666141, 7.256946833324577, 5.767834131536049, 6.059258576134178,

```
5.639692192425388]
ADF Test Statistic : -3.0595018940509013
p-value : 0.029699047143613967
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609905510>
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partialautocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial autocorrelation and autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly fossil brown coal outputs and the average monthly prices of energy per EUR/MWh.

```
In [ ]: df_FossilBrown.describe(include = 'all') # Description Tables
```

```
Out[ ]:
```

	Price	Fossil_Brown	Dates	Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN
mean	57.859848	447.860317	NaN	7.617232	-0.067666	-0.951075
std	10.320573	167.425524	NaN	2.378039	2.135393	3.022350
min	32.618667	124.415882	NaN	2.576980	-4.107814	-6.286639
25%	51.528411	335.250326	NaN	5.619432	-1.312553	-3.433260
50%	59.622281	469.047245	NaN	8.047600	-0.207902	-1.150747
75%	64.999583	576.080340	NaN	9.414601	1.390311	0.779353
max	79.492083	695.284722	NaN	11.776417	4.600344	4.869096

In []:

In []: #Bell Curves

```
patientsResults_mean = np.mean(df_FossilBrown["Ratio"])
patientsResults_std = np.std(df_FossilBrown["Ratio"])

patientsResultspdf = stats.norm.pdf(df_FossilBrown["Ratio"].sort_values(), patientsResults_mean, patientsResults_std)

plt.plot(df_FossilBrown["Ratio"].sort_values(), patientsResultspdf)
plt.xlim([0,20])
plt.xlabel("Output to energy price per EUR/MWh", size=15)
plt.title(f'Skewness for data: {skew(df_FossilBrown["Ratio"])}') # Output/Price per EUR/MWh Bell Curve
plt.suptitle("Frequency distribution of average monthly fossil brown outputs to energy prices per EUR/MWh ratios")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves
patientsResults_mean = np.mean(df_FossilBrown["Fossil_Brown"])
patientsResults_std = np.std(df_FossilBrown["Fossil_Brown"])

patientsResultspdf = stats.norm.pdf(df_FossilBrown["Fossil_Brown"].sort_values(), patientsResults_mean, patientsResults_std)

plt.plot(df_FossilBrown["Fossil_Brown"].sort_values(), patientsResultspdf)
plt.xlim([0,1000])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilBrown["Fossil_Brown"])}')
plt.suptitle("Frequency distribution of average monthly fossil brown outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

Frequency distribution of average monthly fossil brown outputs to energy prices per EUR/MWh ratios (scaled in decimals).

Bell curve fit - 4.0000000000000005



Frequency distribution of average monthly fossil brown outputs to energy prices per EUR/MWh ratios (scaled in decimals).

Bell curve fit - 4.0000000000000005



These bell shaped curves are roughly symmetrical. Hence, they have a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: Fossil_Brown_Ratio_Autocorrelations = sm.tsa.acf(df_FossilBrown["Ratio"],fft=False) #Autocorrelations  
print(Fossil_Brown_Ratio_Autocorrelations)
```

```
[ 1. 0.59492536 0.53968667 0.26116372 0.06245553 -0.07330085  
-0.19951145 -0.18060914 -0.25131173 -0.11760908 -0.06090059 -0.03161517  
0.08525619 -0.01290166 -0.08311451 -0.16472315 -0.24356617 -0.32204956  
-0.23641718 -0.19583051 -0.08613301 -0.01229077 0.15340733 0.15124208  
0.20276741 0.18970582 0.08452253 0.03762272 -0.06517607 -0.07641704  
-0.06101094 -0.11031999 -0.07195769 -0.14584492 -0.0511969 -0.07313725  
-0.0106611 0.02650699 -0.01758551 0.00340967 -0.03421846]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,
```

```
In [ ]: df_FossilBrown['First Difference fossil brown coal Ratio'] = df_FossilBrown["Ratio"]- df_FossilBrown["Ratio"].shift()  
df_FossilBrown['Seasonal Difference fossil brown coal Ratio']=df_FossilBrown["Ratio"]- df_FossilBrown["Ratio"].shift(12)  
df_FossilBrown.head()
```

Out[]:	Price	Fossil_Brown	Dates	Ratio	First Difference	Seasonal Difference	First Difference fossil brown coal Ratio	Seasonal Difference fossil brown coal Ratio
0	64.949019	572.851296	2015-01	8.820015	NaN	NaN	NaN	NaN
1	56.383854	313.418155	2015-02	5.558651	-3.261364	NaN	-3.261364	NaN
2	55.522463	244.437416	2015-03	4.402496	-1.156155	NaN	-1.156155	NaN
3	58.354083	463.119777	2015-04	7.936373	3.533877	NaN	3.533877	NaN
4	57.294059	374.280914	2015-05	6.532630	-1.403743	NaN	-1.403743	NaN

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly fossil brown coal output to price of energy per EUR/MWH")  
plt.ylabel('Seasonality')  
plt.xlabel('Months')  
df_FossilBrown['Seasonal Difference fossil brown coal Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609af1990>
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly fossil brown output and the average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

```
In [ ]: #ADF Tests  
from statsmodels.tsa.stattools import adfuller
```

```

def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_FossilBrown['Fossil_Brown'])

test_result=adfuller(df_FossilBrown['Fossil_Brown'])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilBrown["Fossil_Brown"])
plt.suptitle(" Autocorrelations of average monthly fossil brown coal")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_FossilBrown["Fossil_Brown"])
plt.suptitle("Partial Autocorrelations of average monthly fossil brown coal")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Fossil_Brown_Autocorrelations = sm.tsa.acf(df_FossilBrown["Fossil_Brown"], fft=False) #Autocorrelations
print(Fossil_Brown_Autocorrelations)
df_FossilBrown['First Difference'] = df_FossilBrown["Fossil_Brown"]- df_FossilBrown["Fossil_Brown"].shift(1) # :
df_FossilBrown['Seasonal Difference']=df_FossilBrown["Fossil_Brown"]- df_FossilBrown["Fossil_Brown"].shift(12)
df_FossilBrown.head()
df_FossilBrown['First Difference fossil brown coal'] = df_FossilBrown["Fossil_Brown"]- df_FossilBrown["Fossil_B"]
df_FossilBrown['Seasonal Difference fossil brown coal']=df_FossilBrown["Fossil_Brown"]- df_FossilBrown["Fossil_I
df_FossilBrown.head()
plt.suptitle("Seasonal Difference of average monthly fossil brown coal output")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_FossilBrown['Seasonal Difference fossil brown coal'].plot() # Seasonality Plot

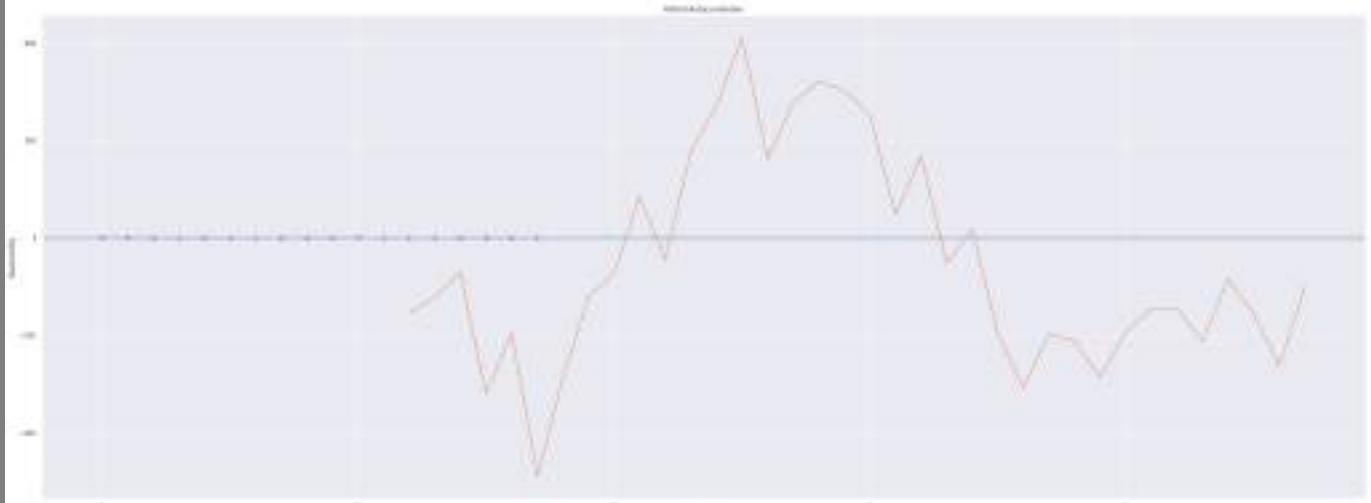
```

ADF Test Statistic : -3.4811300328069645
 p-value : 0.008485147743162468
 #Lags Used : 3
 Number of Observations : 44
 strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
 [1. 0.64083638 0.46521629 0.17735177 -0.06784148 -0.20529574
 -0.35932642 -0.371149 -0.40080123 -0.27304364 -0.10861713 -0.01816965
 0.1241744 0.06349395 -0.0101791 -0.12351309 -0.19058787 -0.23192705
 -0.16046812 -0.11875434 -0.0689481 0.01467396 0.15923135 0.23214047
 0.30902489 0.28227076 0.16260112 0.09218351 -0.02367384 -0.06767177
 -0.0765852 -0.11960189 -0.10123388 -0.143868 -0.0130803 -0.01985048
 0.0236005 0.05774521 0.00186864 0.01184143 -0.04053259]

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
 FutureWarning,

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609ebcf0>





The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partialautocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation plot, indicating that the lags beforehand influenced the average monthly fossil brown coal outputs.

In []: df_FossilBrown.describe(include = 'all') # Description Tables

Out[]:

	Price	Fossil_Brown	Dates	Ratio	First Difference	Seasonal Difference	First Difference fossil brown coal Ratio	Seasonal Difference fossil brown coal Ratio	First Difference fossil brown coal	Seasonal Difference fossil brown coal
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	57.859848	447.860317	NaN	7.617232	-4.193940	-53.584378	-0.067666	-0.951075	-4.193940	-53.584378
std	10.320573	167.425524	NaN	2.378039	141.783995	216.041609	2.135393	3.022350	141.783995	216.041609
min	32.618667	124.415882	NaN	2.576980	-282.535011	-489.562726	-4.107814	-6.286639	-282.535011	-489.562726
25%	51.528411	335.250326	NaN	5.619432	-87.034423	-195.574741	-1.312553	-3.433260	-87.034423	-195.574741
50%	59.622281	469.047245	NaN	8.047600	0.390911	-111.100358	-0.207902	-1.150747	0.390911	-111.100358
75%	64.999583	576.080340	NaN	9.414601	82.551299	104.928816	1.390311	0.779353	82.551299	104.928816
max	79.492083	695.284722	NaN	11.776417	290.881812	411.749949	4.600344	4.869096	290.881812	411.749949

Below is the box and whisker plot for the distribution of the average monthly outputs of fossil brown coal and its the average monthly prices of energy per EUR/MWH.

In []: # Box and Whisker Plot
 sns.set(style="darkgrid")
 plt.suptitle('Distribution of average monthly fossil brown coal outputs by average monthly price of energy per MWH')
 plt.ylabel('Average monthly fossil brown coal outputs')
 plt.xlabel('Average monthly price of energy per EUR/MWH')
 sns.boxplot(x=Rounded_Y, y=[572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.1])
 plt.show()



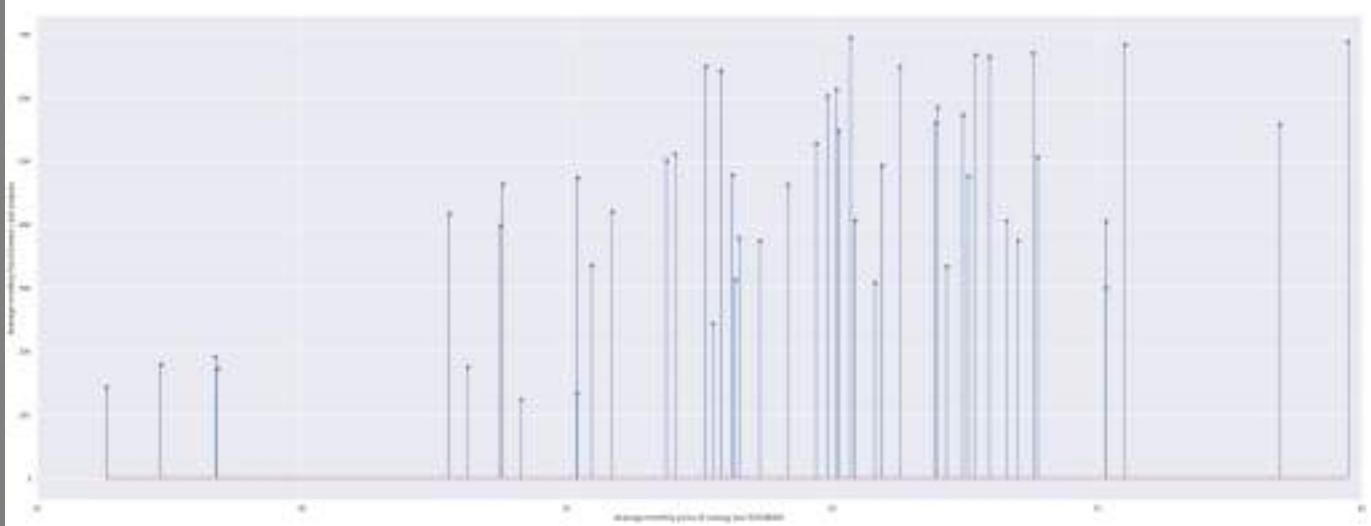
This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil brown coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil brown coal produced, which was in between 300 and 400 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), fossil brown coal output was slightly below 50 units. When the price of energy per EUR/MWH was at its highest, at approximately 79.49 EUR/MWH, fossil brown coal output was slightly below 700 units. At approximately 60.73 EUR/MWH, fossil brown coal produced the highest amount at roughly 700 units.The lowest amount produced, whcih was slightly above 100 units, was at the price of approximently 48.28 EUR/MWH.

```
In [ ]: plt.suptitle('Distribution of average monthly fossil brown coal outputs by average monthly price of energy per I')
plt.ylabel('Average monthly fossil brown coal outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

plt.xlim(30, 80)
plt.rcParams.update({'font.size': 19})
# Stem Plot
plt.stem(Rounded_Y, fossil_brown)
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of fossil brown coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil brown coal produced, which was in between 300 and 400 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), fossil brown coal output was slightly below 50 units. When the price of energy per EUR/MWH was at its highest, at approximately 79.49 EUR/MWH, fossil brown coal output was slightly below 700 units. At approximately 60.73 EUR/MWH, fossil brown coal produced the highest amount at roughly 700 units.The lowest amount produced, whcih was slightly above 100 units, was at the price of approximently 48.28 EUR/MWH.

```
In [ ]: fossil_brown_Dict = {key: i for i, key in enumerate(fossil_brown)}

def Hist_fossil_brown(fossil_brown_Dict):
    for k, v in fossil_brown.items(): print(f"{v}:{k}")

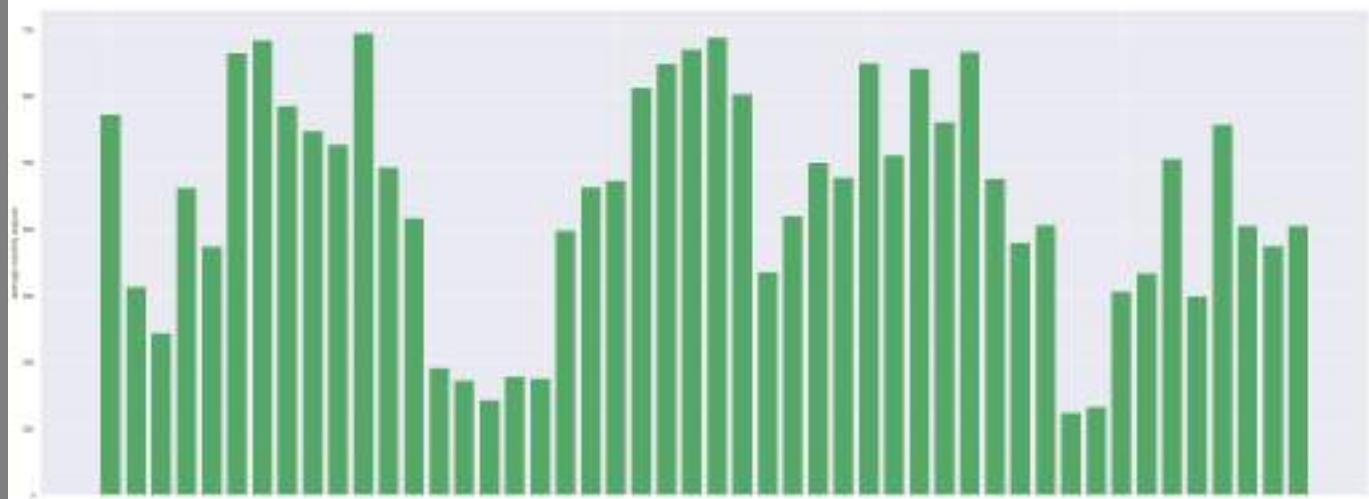
print(fossil_brown_Dict)

plt.bar(list(fossil_brown_Dict.values()), fossil_brown_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of fossil brown coal")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.rcParams.update({'font.size': 19})
#Outputs in MWH Histogram
plt.show()
```

```
{572.8512960436562: 0, 313.41815476190476: 1, 244.43741588156124: 2, 463.11977715877435: 3, 374.2809139784946: 4, 665.162726008345: 5, 684.2204301075269: 6, 585.7674731182796: 7, 548.0833333333334: 8, 528.0188172043011: 9, 695.2847222222222: 10, 493.4804845222073: 11, 417.9274193548387: 12, 191.66522988505747: 13, 173.20323014804845: 14, 143.57083333333333: 15, 179.002688172043: 16, 175.6: 17, 398.8156123822342: 18, 464.3736559139785: 19, 473.7208333333336: 20, 613.7691275167786: 21, 649.5458333333333: 22, 670.7956989247311: 23, 688.6451612903226: 24, 603.4151785714286: 25, 335.66756393001344: 26, 420.1152777777778: 27, 500.77016129032256: 28, 478.8111111111113: 29, 650.6774193548387: 30, 511.8467741935484: 31, 642.6305555555556: 32, 561.3221476510067: 33, 667.647222222222: 34, 476.5430107526882: 35, 379.56451612903226: 36, 406.95089285714283: 37, 124.41588156123822: 38, 133.979166666666: 39, 307.7043010752688: 40, 333.99861111111113: 41, 506.36877523553164: 42, 300.14247311827955: 43, 558.1611111111112: 44, 405.83758389261743: 45, 375.73611111111111: 46, 406.22849462365593: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```

Average monthly outputs of fossil brown coal



In []:

The green bars represent the observation value for each respective month. This histogram has significant troughs between month 10 and month 20 as well as in between months 37 and 40, meaning that there were sharp decreases in fossil oil output. In addition, the histogram has a multimodal shape, which means that there wasn't any external factors that led to a singular price increase. However, the price fluctuations within the histogram appear to be seasonal.

```
In [ ]: pdToList_Dict = {key: i for i, key in enumerate(pdToList)}

def Hist_pdToList(pdToList_Dict):
    for k, v in pdToList_Dict.items(): print(f"{v}:{k}")

#Output/price per EUR/MWH Histogram
print(pdToList_Dict)

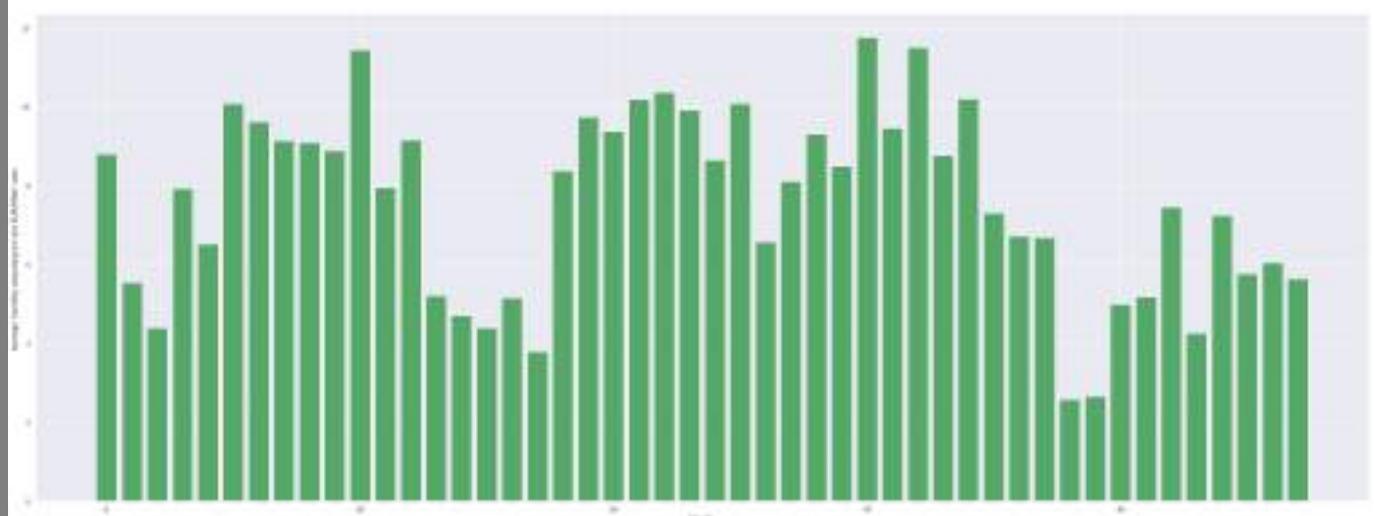
plt.bar(list(pdToList_Dict.values()), pdToList_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of fossil brown coal")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')
import matplotlib

plt.rcParams.update({'font.size': 19})

plt.show()
plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```

Average monthly output price per EUR/MWh ratio of fossil brown coal



The green bars represent the observation value for each respective month. This displays a roughly symmetric yet multimodal distribution. It appears that there are deep trenches occurring roughly every ten months, meaning that the output produced per EUR/MWH sharply declined for the time being. It would be fair to assume that this fluctuating pattern is seasonal.

```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals  
FossilBrownQuadRatioPredict = FossilBrown_ypred/ypred
```

```
sns.residplot(x = FossilBrownQuadRatioPredict , y = standardized_residualsFossilBrownquad/standardized_residuals)

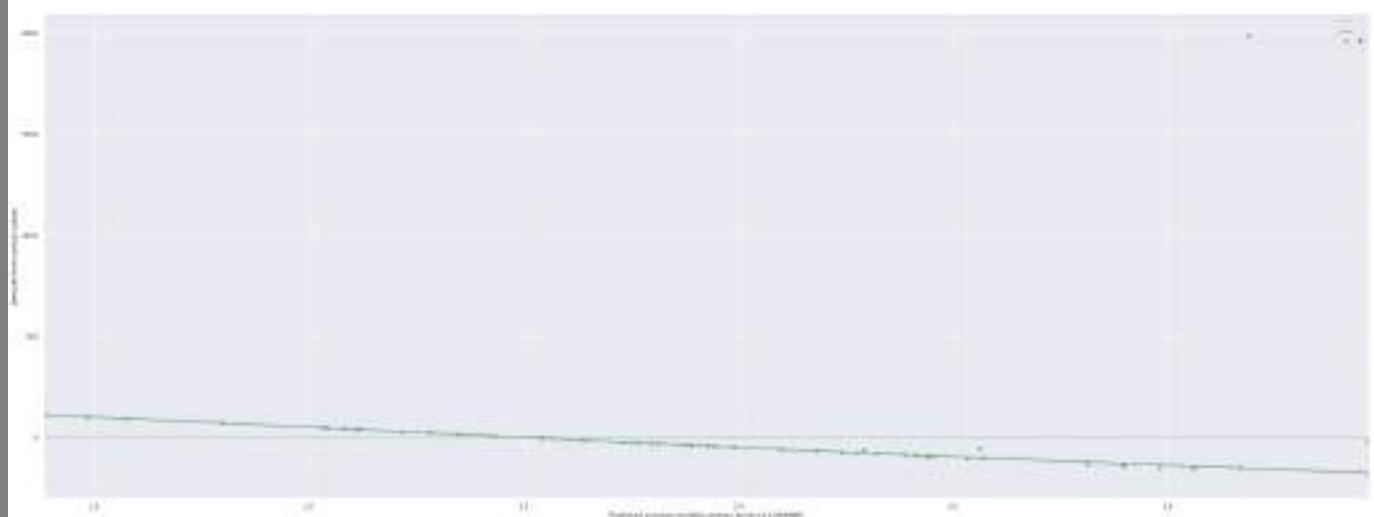
plt.suptitle("Predicted average monthly quadratic fossil brown coal energy prices to EUR/MWH versus respective residuals")
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")
plt.ylabel("Amount from actual values")

plt.legend(..#)

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff6088abe90>
```

Predicted average monthly quadratic fossil brown coal energy prices to EUR/MWH versus respective quadratic model residuals



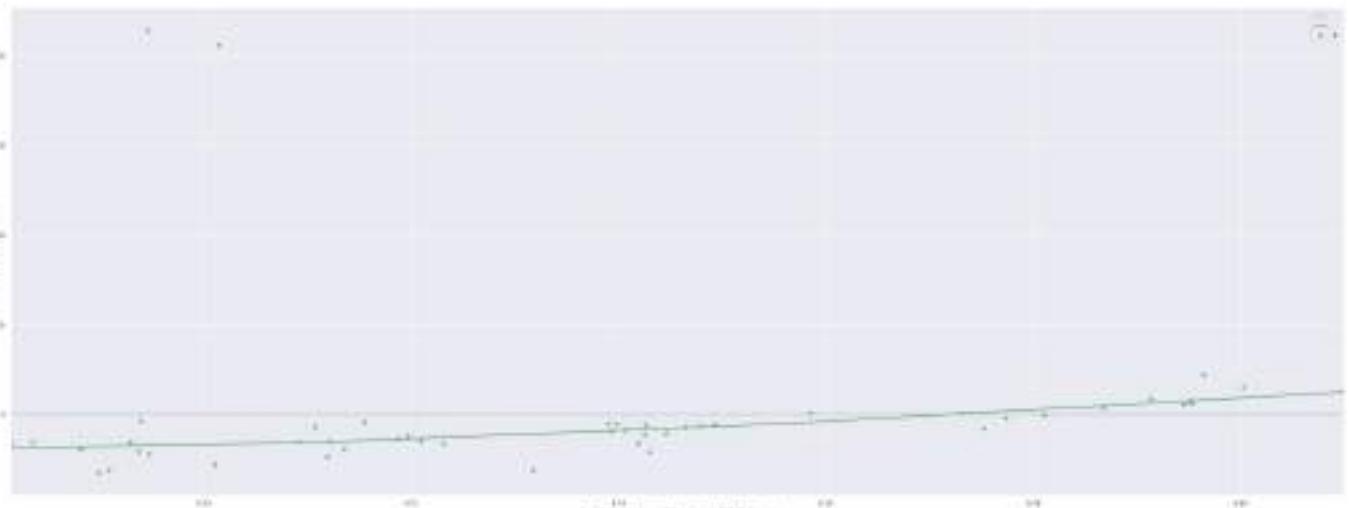
With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals
FossilBrownRegRatioPredict = predictionsFossilBrown/predictions

sns.residplot(x = FossilBrownRegRatioPredict, y = standardized_residualsFossilBrown/standardized_residualsPrice
plt.suptitle("Predicted linear fossil brown coal output to EUR/MWH versus respective linear model residuals ")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff609a87c90>
Predicted linear fossil brown coal output to EUR/MWH versus respective linear model residuals
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

Below is a scatterplot depicting the relationship between the average monthly prices of energy per EUR/MWH and the average monthly outputs of fossil brown coal.

```
In [ ]: modelFossilBrowncoal = stats.linregress([572.8512960436562, 313.41815476190476, 244.43741588156124, 463.1197771
[64.9490188172043,
56.383854166666666,
55.522462987886975,
58.35408333333333,
57.29405913978498,
65.9749027777778,
71.07204301075271,
63.99806451612899,
60.254791666666634,
59.40676510067113,
60.72679166666668,
61.901760752688226,
45.57872311827956,
36.752083333333374,
36.81800807537014,
32.61866666666666,
34.691370967741896,
46.266319444444434,
47.50201612903221,
47.6023387096774,
50.4055972222224,
60.182429530201404,
62.58105555555558,
67.5951344086021,
79.4920833333331,
59.83779761904767,
50.95989232839837,
51.71791666666662,
53.77262096774189,
56.2582222222224,
55.252580645161316,
54.08432795698931,
55.81655555555558,
63.92528859060403,
65.43065277777781,
```

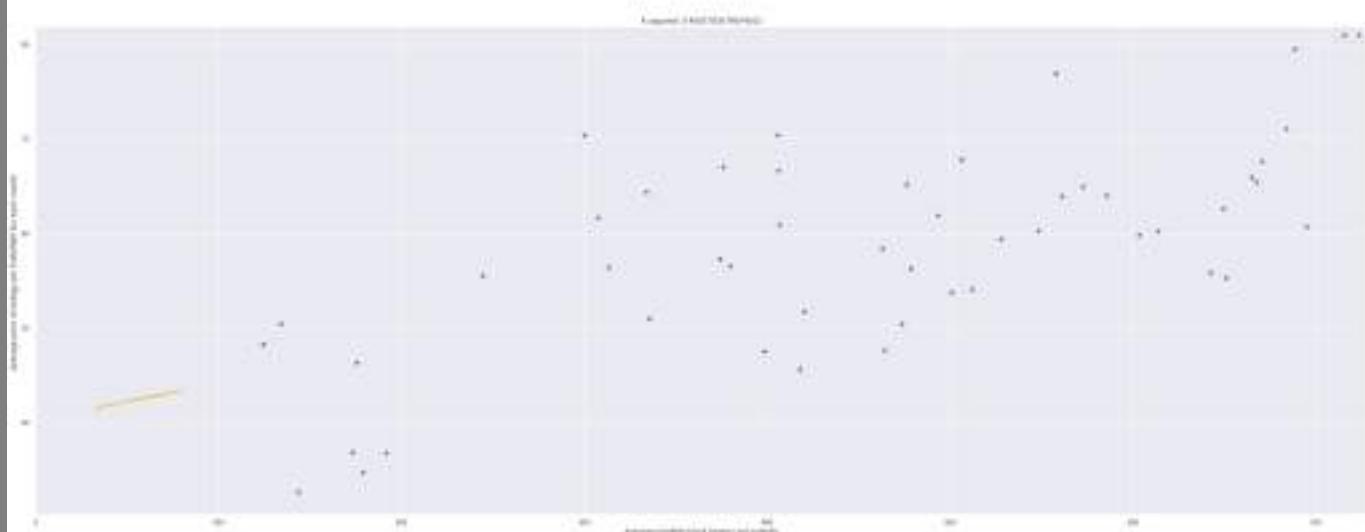
```
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881])
```

In []:

```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(fossil_brown,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(fossil_brown,Price_Actual, "o")
f = lambda x: 0.039383 *x + 40.221857
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average monthly fossil brown outputs versus average predicted linear monthly price of energy per MWH")
plt.legend('#')
plt.title(f"R squared: {modelFossilBrowncoal.rvalue**2}")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly fossil brown coal outputs')
plt.show()
```

```
slope: 0.039383      intercept: 40.221857
```



There is a moderately positive correlation the average monthly outputs of brown fossil coal and their respective the average monthly prices of energy per EUR/MWH. It appears that as the average monthly price of energy per EUR/MWH increases, the average monthly outputs of brown fossil coal increases as well. The blue dots are the observations and orange line is the model of best fit.

Below are the predicted average monthly fossil brown coal outputs to the prices of energy per EUR/MWH and their respective seasonality and distribution analysis'.

In []:

```
print(predictions)

print(predictionsFossilBrown)

Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price
```

```
[52.82161316 53.03600615 53.25039913 53.46479211 53.67918509 53.89357808  
54.10797106 54.32236404 54.53675703 54.75115001 54.96554299 55.17993597  
55.39432896 55.60872194 55.82311492 56.03750791 56.25190089 56.46629387  
56.68068685 56.89507984 57.10947282 57.3238658 57.53825879 57.75265177  
57.96704475 58.18143773 58.39583072 58.6102237 58.82461668 59.03900967  
59.25340265 59.46779563 59.68218861 59.8965816 60.11097458 60.32536756  
60.53976055 60.75415353 60.96854651 61.18293949 61.39733248 61.61172546  
61.82611844 62.04051143 62.25490441 62.46929739 62.68369037 62.89808336]  
[42.77973086 42.44241073 42.40848674 42.52000386 42.47825715 42.82013304  
43.02087267 42.74227962 42.59485907 42.56146141 42.61344775 42.65972132  
42.01687446 41.66925671 41.67185301 41.50647121 41.5881001 42.04395393  
42.09261912 42.0965701 42.20697026 42.59200925 42.68647384 42.88394229  
43.3524774 42.57843668 42.22879995 42.25865307 42.33957307 42.43746299  
42.39785801 42.35184896 42.42006892 42.7394135 42.79869895 42.78769635  
42.44745652 42.61936724 42.12324717 42.20677881 42.64916683 42.75606653  
42.89135832 42.99298446 43.25094691 42.99291765 42.84567716 42.86218223]  
[52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04,  
56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59.  
.68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]
```

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(predictionsFossilBrown))
```

```
[42.77973085791807, 42.442410728794634, 42.40848673602778, 42.52000386144516, 42.478257145248726, 42.82013303629  
587, 43.02087266848022, 42.74227961900825, 42.59485906932324, 42.561461412370576, 42.613447748869085, 42.6597213  
1627657, 42.01687446076636, 41.669256709857486, 41.671853010501266, 41.5064712059462, 41.58810009587413, 42.0439  
5392636801, 42.09261911638002, 42.09657010007245, 42.20697025722046, 42.59200924609017, 42.686473844682624, 42.8  
8394228751502, 43.35247739585217, 42.57843667795943, 42.22879994811567, 42.25865306572589, 42.33957306533216, 42  
.43746299159972, 42.39785801487522, 42.351848957991734, 42.42006892352186, 42.739413499605206, 42.79869894976446  
, 42.78769634594106, 42.44745651715237, 42.61936724091852, 42.123247165114975, 42.20677881307495, 42.64916683291  
064, 42.75606652649225, 42.891358322018085, 42.99298446164267, 43.250946907721854, 42.99291764773654, 42.8456771  
5513977, 42.86218223424221]
```

```
In [ ]: print(list(predictions))
```

```
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445  
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739  
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293  
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75  
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.  
03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,  
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988  
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373  
92337, 62.89808335677895]
```

```
In [ ]: dfFossilBrownReg = ({ "Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895] )  
"Fossil_BrownReg" : [42.77973085791807, 42.442410728794634, 42.40848673602778, 42.52000386144516, 42.478257145248726, 42.82013303629587, 43.02087266848022, 42.74227961900825, 42.59485906932324, 42.561461412370576, 42.613447748869085, 42.65972131627657, 42.01687446076636, 41.669256709857486, 41.671853010501266, 41.5064712059462, 41.58810009587413, 42.04395392636801, 42.09261911638002, 42.09657010007245, 42.20697025722046, 42.59200924609017, 42.686473844682624, 42.8394228751502, 43.35247739585217, 42.57843667795943, 42.22879994811567, 42.25865306572589, 42.33957306533216, 42.43746299159972, 42.39785801487522, 42.351848957991734, 42.42006892352186, 42.739413499605206, 42.79869894976446, 42.78769634594106, 42.44745651715237, 42.61936724091852, 42.123247165114975, 42.20677881307495, 42.64916683291064, 42.75606652649225, 42.891358322018085, 42.99298446164267, 43.250946907721854, 42.99291764773654, 42.84567715513977, 42.86218223424221]  
print(dfFossilBrownReg) #Dataframes  
df_FossilBrownReg= pd.DataFrame.from_dict(dfFossilBrownReg, orient = "columns")  
print(df_FossilBrownReg)
```

```
{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'Fossil_BrownReg': [42.776016707996575, 42.43861349878181, 42.40468115072224, 42.51622574219387, 42.47446874401099, 42.81642883720856, 43.017217910459536, 42.738556245052735, 42.591099386497675, 42.557693503885915, 42.609692644332995, 42.65597760866545, 42.01297242351269, 41.66526905626491, 41.667865996363254, 41.50244345917991, 41.58409245385418, 42.04005855863769, 42.08873573461816, 42.09268769141613, 42.203115039515254, 42.58824886136883, 42.68273672607203, 42.88025380429511, 43.34890431025082, 42.574672950389264, 42.22495010694319, 42.25481057721197, 42.3357505069688, 42.433664542986456, 42.39404981177415, 42.348029423113175, 42.41626619084536, 42.735689419740254, 42.7949894715796, 42.78398415787543, 42.44366052988945, 42.61561359432074, 42.11937132687296, 42.2029235482181, 42.64542052578836, 42.75234654817616, 42.88767166532666, 42.98932283491012, 43.24734881572006, 42.989256004548096, 42.84197924742835, 42.85848839164093]}

Price Fossil_BrownReg
```

0	52.821613	42.776017
1	53.036006	42.438613
2	53.250399	42.404681
3	53.464792	42.516226
4	53.679185	42.474469
5	53.893578	42.816429
6	54.107971	43.017218
7	54.322364	42.738556
8	54.536757	42.591099
9	54.751150	42.557694
10	54.965543	42.609693
11	55.179936	42.655978
12	55.394329	42.012972
13	55.608722	41.665269
14	55.823115	41.667866
15	56.037508	41.502443
16	56.251901	41.584092
17	56.466294	42.040059
18	56.680687	42.088736
19	56.895080	42.092688
20	57.109473	42.203115
21	57.323866	42.588249
22	57.538259	42.682737
23	57.752652	42.880254
24	57.967045	43.348904
25	58.181438	42.574673
26	58.395831	42.224950
27	58.610224	42.254811
28	58.824617	42.335751
29	59.039010	42.433665
30	59.253403	42.394050
31	59.467796	42.348029
32	59.682189	42.416266
33	59.896582	42.735689
34	60.110975	42.794989
35	60.325368	42.783984
36	60.539761	42.443661
37	60.754154	42.615614
38	60.968547	42.119371
39	61.182939	42.202924
40	61.397332	42.645421
41	61.611725	42.752347
42	61.826118	42.887672
43	62.040511	42.989323
44	62.254904	43.247349
45	62.469297	42.989256
46	62.683690	42.841979
47	62.898083	42.858488

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_FossilBrownReg["Fossil_BrownReg"])
```

```

test_result=adfuller(df_FossilBrownReg["Fossil_BrownReg"])

df_FossilBrownReg['First Difference'] = df_FossilBrownReg["Fossil_BrownReg"]- df_FossilBrownReg["Fossil_BrownReg"].shift(1)
df_FossilBrownReg['Seasonal Difference']=df_FossilBrownReg["Fossil_BrownReg"]- df_FossilBrownReg["Fossil_BrownReg"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted linear fossil brown prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilBrownReg.head() #Predictive Linear Seasonality Plot
df_FossilBrownReg['Seasonal Difference'].plot()
# Seasonality Plot

```

ADF Test Statistic : -2.5832139889153054
p-value : 0.09651646085286963
#Lags Used : 1
Number of Observations : 46
weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608d04ed0>



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resource and the average monthly prices of energy per EUR/MWH.

In []:

In []:

In []: FossilBrownRegResults_mean = np.mean(df_FossilBrownReg["Fossil_BrownReg"])

FossilBrownRegResults_std = np.std(df_FossilBrownReg["Fossil_BrownReg"])

FossilBrownRegResultspdf = stats.norm.pdf(df_FossilBrownReg["Fossil_BrownReg"].sort_values(), FossilBrownRegResu

plt.plot(df_FossilBrownReg["Fossil_BrownReg"].sort_values(), FossilBrownRegResultspdf)

plt.xlim([0,100])

plt.xlabel("Value ", size=15)

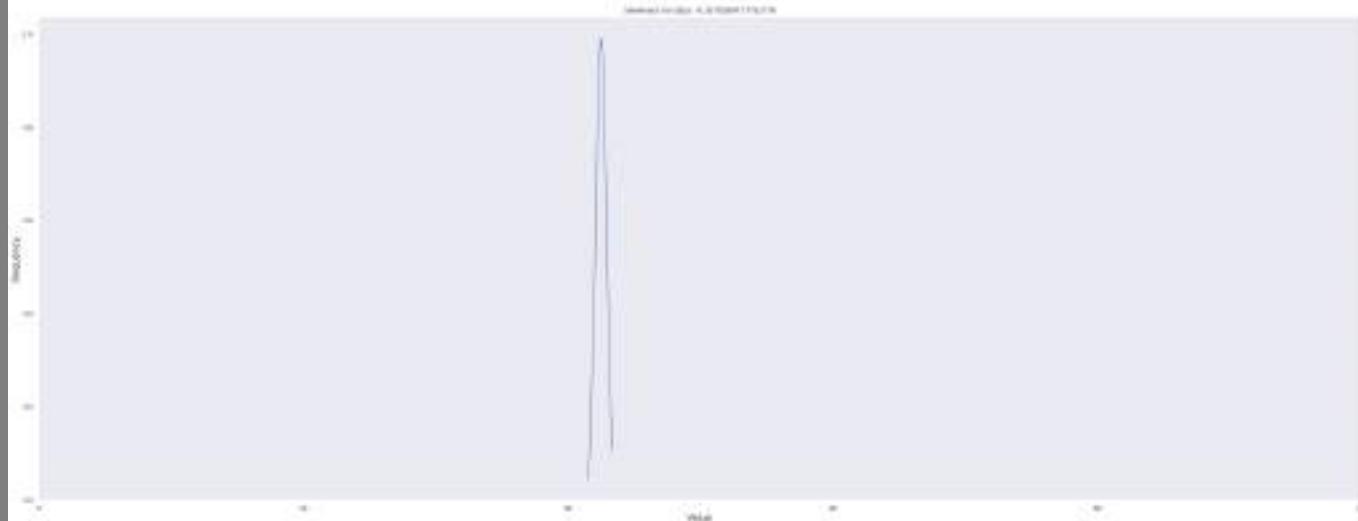
plt.title(f'Skewness for data: {skew(df_FossilBrownReg["Fossil_BrownReg"])}')

plt.suptitle("Frequency distribution of average monthly predicted linear fossil brown energy prices per EUR/MWH")

plt.ylabel("Frequency", size=15)

plt.grid(True, alpha=0.3, linestyle="--")

plt.show()



This bell shaped curve is roughly symmetrical. Hence, it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

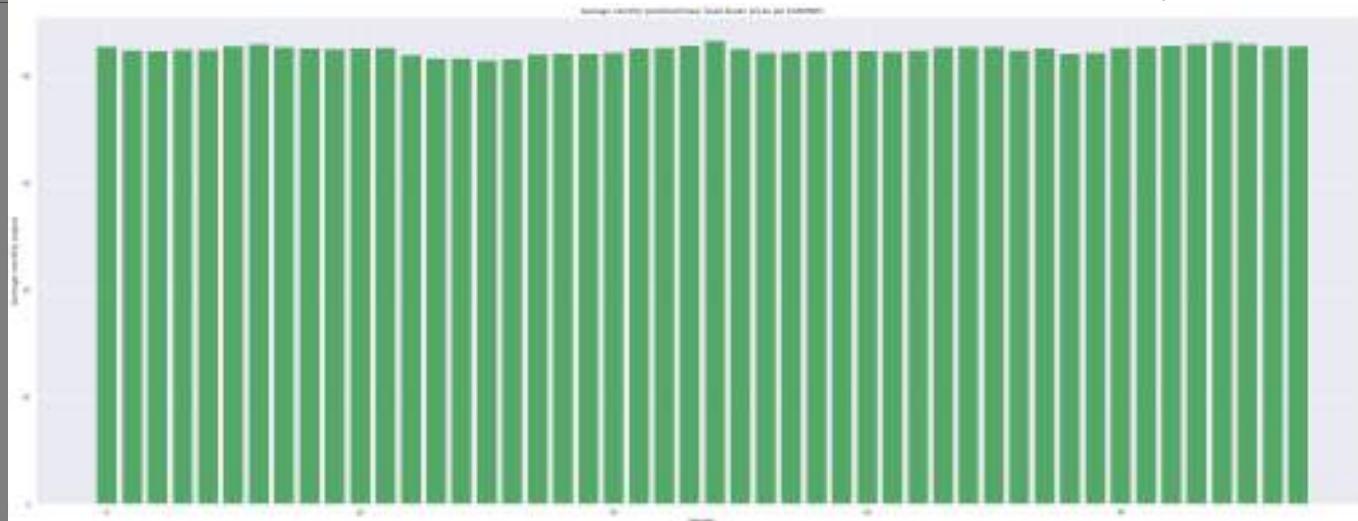
If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Fossil_BrownReg_Dict = {key: i for i, key in enumerate(df_FossilBrownReg["Fossil_BrownReg"])}
def Hist_Fossil_BrownReg(Fossil_BrownReg_Dict):
    for k, v in Fossil_BrownReg_Dict.items(): print(f"{v}:{k}")
print(Fossil_BrownReg_Dict)

plt.bar(list(Fossil_BrownReg_Dict.values()), Fossil_BrownReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted linear fossil brown prices per EUR/MWh")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{42.776016707996575: 0, 42.43861349878181: 1, 42.40468115072224: 2, 42.51622574219387: 3, 42.47446874401099: 4, 42.81642883720856: 5, 43.017217910459536: 6, 42.738556245052735: 7, 42.591099386497675: 8, 42.557693503885915: 9, 42.609692644332995: 10, 42.65597760866545: 11, 42.01297242351269: 12, 41.66526905626491: 13, 41.66786599636325: 14, 41.50244345917991: 15, 41.58409245385418: 16, 42.04005855863769: 17, 42.08873573461816: 18, 42.0926876914: 19, 42.203115039515254: 20, 42.58824886136883: 21, 42.68273672607203: 22, 42.88025380429511: 23, 43.348904: 31025082: 24, 42.574672950389264: 25, 42.22495010694319: 26, 42.25481057721197: 27, 42.3357505069688: 28, 42.433664542986456: 29, 42.39404981177415: 30, 42.348029423113175: 31, 42.41626619084536: 32, 42.735689419740254: 33, 42.7949894715796: 34, 42.78398415787543: 35, 42.44366052988945: 36, 42.61561359432074: 37, 42.11937132687296: 38, 42.2029235482181: 39, 42.64542052578836: 40, 42.75234654817616: 41, 42.88767166532666: 42, 42.98932283491012: 43, 43.24734881572006: 44, 42.989256004548096: 45, 42.84197924742835: 46, 42.85848839164093: 47}
```



The green bars represent the observation value for each respective month.

```
In [ ]:
```

```
In [ ]: print(FossilBrown_Logpred)
```

```
print(FossilBrown_Logpred/Logpred)
```

```
print(Logpred)
```

```
Rounded_Logpred = [round(item, 2) for item in Logpred]
```

```
print(Rounded_Logpred)
```

```
#Rounded Price
```

```
[62.78234245 52.56514006 49.84848572 58.46080847 54.96208567 66.41782463  
67.1683703 63.29101761 61.80691084 61.0167141 67.60411305 59.656498  
56.68100707 47.7701695 47.04308434 45.87607771 47.27148321 47.13747584  
55.92833069 58.51018972 58.8783077 64.39380105 65.80278774 66.63966685  
67.34262858 63.98603359 53.441384 56.76717106 59.94358585 59.07877707  
65.84735276 60.37981383 65.53044477 62.32829236 66.51567104 58.98945293  
55.1701687 56.24872077 45.12170216 45.49833106 52.34011253 53.37565597  
60.16407492 52.04230661 62.2038019 56.20487556 55.0193954 56.22027071]  
[2.2999429 2.19776158 2.11422349 2.36731141 2.26398106 2.39756873  
2.26053677 2.35089392 2.4290365 2.42993747 2.63756286 2.28613482  
2.88395218 2.95406585 2.9044313 3.15517086 3.07800254 2.36571492  
2.73985826 2.86079233 2.73108306 2.5335454 2.49603157 2.35131123  
2.03846422 2.53104423 2.45399328 2.57139305 2.61908753 2.47522874  
2.80545482 2.62404558 2.76572963 2.31760639 2.41986054 2.15469598  
2.30181299 2.1894678 2.17771648 2.11063774 2.0139204 1.97247501  
2.11727572 1.7680996 1.94273463 1.90956329 1.96792361 1.99905191]  
[27.29734838 23.91758071 23.57768041 24.69502249 24.27674267 27.70215664  
29.71346062 26.92210695 25.44503172 25.11040502 25.63128037 26.09491682  
19.65393442 16.17099008 16.19700362 14.53996621 15.35784412 19.92525621  
20.41285544 20.45244219 21.55859284 25.41647801 26.36296287 28.34149128  
33.03596301 25.28048812 21.77731469 22.076427 22.88720221 23.86800702  
23.4711863 23.0102001 23.69372775 26.89338996 27.48739853 27.37715831  
23.96813682 25.69059052 20.71973214 21.55667467 25.98916653 27.0602444  
28.41579599 29.43403567 32.0186817 29.43336623 27.95809508 28.12346716]  
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.  
36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.89, 23.87, 23.47, 23.01, 23.  
69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(FossilBrown_Logpred))
```

```
[62.7823424510713, 52.56514006365846, 49.848485720687194, 58.460808466685506, 54.962085672457654, 66.41782463334  
35, 67.16837029940126, 63.29101761335433, 61.80691083703386, 61.016714098548576, 67.60411305474386, 59.656498000  
50771, 56.6810070744269, 47.77016950368968, 47.04308434079398, 45.87607771248019, 47.271483211022, 47.137475838  
08863, 55.92833068597696, 58.510189717469046, 58.87830769566092, 64.39380104633578, 65.80278774046707, 66.639666  
85460014, 67.34262858337054, 63.98603359087144, 53.44138399821439, 56.76717105562001, 59.943585851006226, 59.078  
77706517459, 65.84735276151012, 60.37981383344575, 65.53044476889583, 62.32829235592835, 66.51567104002281, 58.9  
8945293224972, 55.17016869623751, 56.248720774459294, 45.12170216119285, 45.498331064223315, 52.34011253155522,  
53.37565597040144, 60.16407491992233, 52.042306606399926, 62.20380189984184, 56.20487555501632, 55.0193954037590  
2, 56.22027071245341]
```

```
In [ ]: print(list(Logpred))
```

```
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639  
58084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.0949  
16817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 1  
9.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.36296287420  
1227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.88720  
2207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 2  
7.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.5566746731514  
95, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.4333662  
30197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]:
```

```
In [ ]: dfFossilBrown_Log = {"Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639  
58084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.0949  
16817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 1  
9.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.36296287420  
1227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.88720  
2207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 2  
7.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.5566746731514  
95, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.4333662  
30197276, 27.958095077371567, 28.123467161134005], #Dataframes  
df_FossilBrown_Log= pd.DataFrame.from_dict(dfFossilBrown_Log, orient = "columns")  
print(df_FossilBrown_Log)
```

```
#ADF Tests
```

```
from statsmodels.tsa.stattools import adfuller
```

```
def adfuller_test(test_result):
```

```
    result=adfuller(test_result)
```

```
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
```

```
    for value,label in zip(result,labels):
```

```
print(label+' : '+str(value) )

if result[1] <= 0.05:
    print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
else:
    print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_FossilBrown_Log["Fossil_Brown_Log"])

test_result=adfuller(df_FossilBrown_Log["Fossil_Brown_Log"])

df_FossilBrown_Log['First Difference'] = df_FossilBrown_Log["Fossil_Brown_Log"]- df_FossilBrown_Log["Fossil_Brown_Log"].shift(1)
df_FossilBrown_Log['Seasonal Difference']=df_FossilBrown_Log["Fossil_Brown_Log"]- df_FossilBrown_Log["Fossil_Brown_Log"].shift(12)

plt.suptitle("Seasonal Difference of average monthly predicted logarithmic fossil brown prices of energy per barrel")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilBrown_Log.head() #Predictive Logarithmic Seasonality Plot
df_FossilBrown_Log['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Fossil_Brown_Log': [62.7823424510713, 52.56514006365846, 49.848485720687194, 58.460808466685506, 54.962085672457654, 66.4178246333435, 67.16837029940126, 63.29101761335433, 61.80691083703386, 61.016714098548576, 67.60411305474386, 59.65649800050771, 56.68100707444269, 47.77016950368968, 47.04308434079398, 45.87607771248019, 47.271483211022, 47.13747583808863, 55.92833068597696, 58.510189717469046, 58.87830769566092, 64.39380104633578, 65.80278774046707, 66.63966685460014, 67.34262858337054, 63.98603359087144, 53.44138399821439, 56.76717105562001, 59.943585851006226, 59.07877706517459, 65.84735276151012, 60.37981383344575, 65.53044476889583, 62.32829235592835, 66.51567104002281, 58.98945293224972, 55.17016869623751, 56.248720774459294, 45.12170216119285, 45.498331064223315, 52.34011253155522, 53.37565597040144, 60.16407491992233, 52.042306606399926, 62.20380189984184, 56.20487555501632, 55.01939540375902, 56.22027071245341], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Log	Fossil_Brown_Log	Dates
0	27.297348	62.782342	2015-01
1	23.917581	52.565140	2015-02
2	23.577680	49.848486	2015-03
3	24.695022	58.460808	2015-04
4	24.276743	54.962086	2015-05
5	27.702157	66.417825	2015-06
6	29.713461	67.168370	2015-07
7	26.922107	63.291018	2015-08
8	25.445032	61.806911	2015-09
9	25.110405	61.016714	2015-10
10	25.631280	67.604113	2015-11
11	26.094917	59.656498	2015-12
12	19.653934	56.681007	2016-01
13	16.170990	47.770170	2016-02
14	16.197004	47.043084	2016-03
15	14.539966	45.876078	2016-04
16	15.357844	47.271483	2016-05
17	19.925256	47.137476	2016-06
18	20.412855	55.928331	2016-07
19	20.452442	58.510190	2016-08
20	21.558593	58.878308	2016-09
21	25.416478	64.393801	2016-10
22	26.362963	65.802788	2016-11
23	28.341491	66.639667	2016-12
24	33.035963	67.342629	2017-01
25	25.280488	63.986034	2017-02
26	21.777315	53.441384	2017-03
27	22.076427	56.767171	2017-04
28	22.887202	59.943586	2017-05
29	23.868007	59.078777	2017-06
30	23.471186	65.847353	2017-07
31	23.010200	60.379814	2017-08
32	23.693728	65.530445	2017-09
33	26.893390	62.328292	2017-10
34	27.487399	66.515671	2017-11
35	27.377158	58.989453	2017-12
36	23.968137	55.170169	2018-01
37	25.690591	56.248721	2018-02
38	20.719732	45.121702	2018-03
39	21.556675	45.498331	2018-04
40	25.989167	52.340113	2018-05
41	27.060244	53.375656	2018-06
42	28.415796	60.164075	2018-07
43	29.434036	52.042307	2018-08
44	32.018682	62.203802	2018-09
45	29.433366	56.204876	2018-10
46	27.958095	55.019395	2018-11
47	28.123467	56.220271	2018-12

ADF Test Statistic : -3.5212357152630314

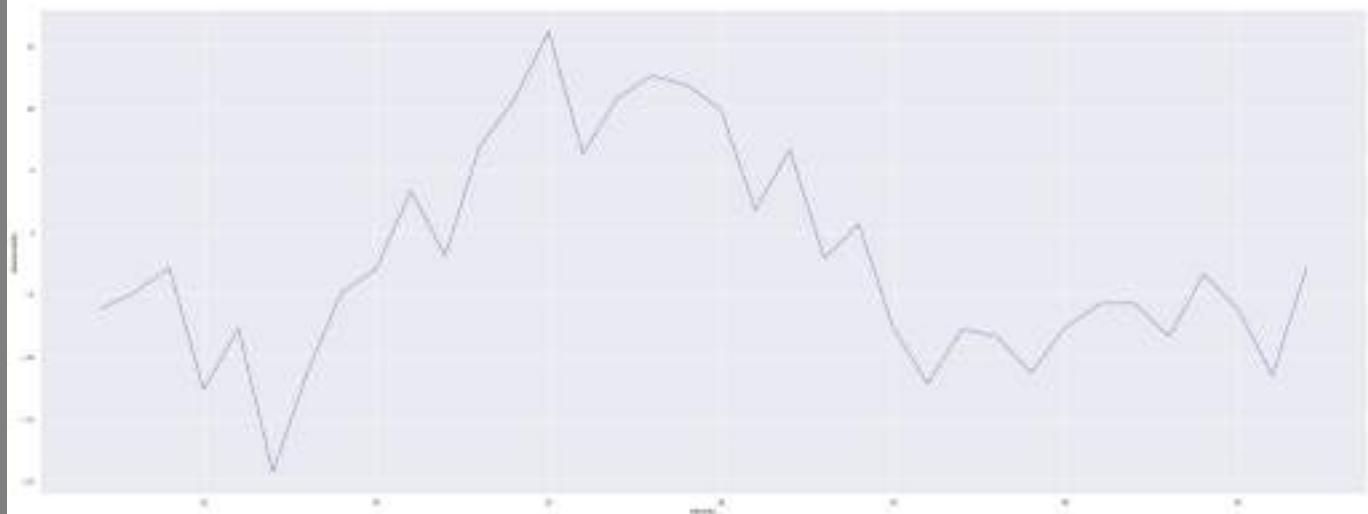
p-value : 0.007458173826222194

#Lags Used : 3

Number of Observations : 44

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60ae7ac50>



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

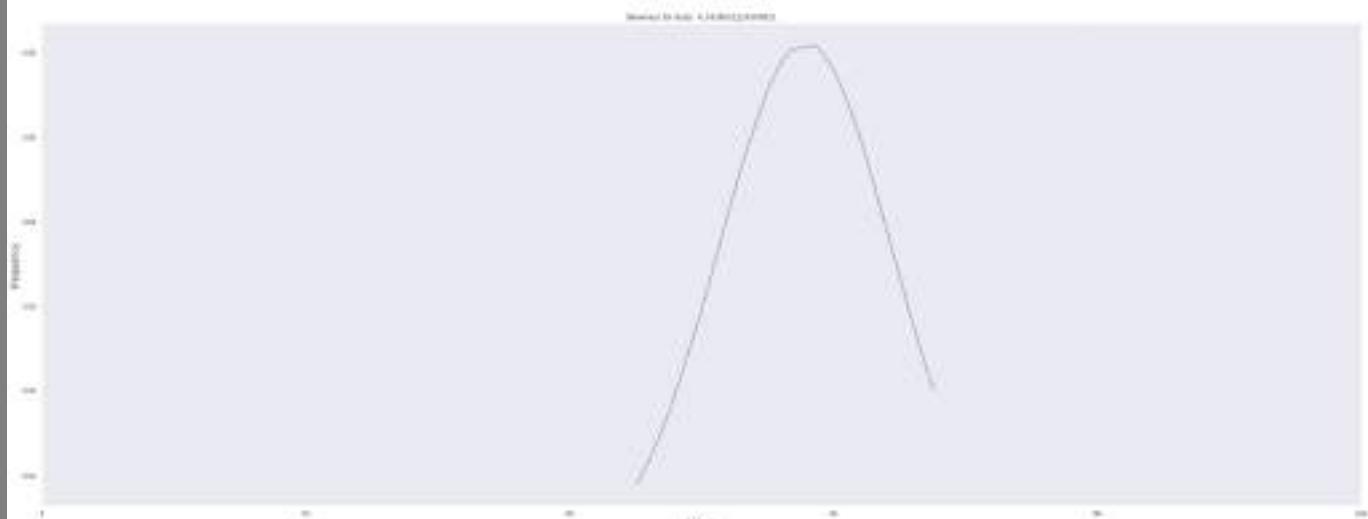
In []: #Bell Curves

```
FossilBrown_LogResults_mean = np.mean(df_FossilBrown_Log["Fossil_Brown_Log"])
FossilBrown_LogResults_std = np.std(df_FossilBrown_Log["Fossil_Brown_Log"])

FossilBrown_LogResultspdf = stats.norm.pdf(df_FossilBrown_Log["Fossil_Brown_Log"].sort_values(), FossilBrown_LogResults_mean, FossilBrown_LogResults_std)

plt.plot(df_FossilBrown_Log["Fossil_Brown_Log"].sort_values(), FossilBrown_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilBrown_Log["Fossil_Brown_Log"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic fossil brown energy prices per EUR/MWH")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

Frequency distribution of average monthly predicted logarithmic fossil brown energy prices per EUR/MWH(scaled in decimals)



This bell shaped curve is roughly symmetrical, hence it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and -0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Fossil_Brown_Log_Dict = {key: i for i, key in enumerate(df_FossilBrown_Log["Fossil_Brown_Log"])}

def Hist_Fossil_Brown_Log(Fossil_Brown_Log_Dict):
    for k, v in Fossil_Brown_Log_Dict.items(): print(f"{v}:{k}")
print(Fossil_Brown_Log_Dict)
```

```

plt.bar(list(Fossil_Brown_Log_Dict.values()), Fossil_Brown_Log_Dict.keys(), color='g') #Predictive Logarithmic
plt.title("Average monthly predicted logarithmic fossil brown logarithmic prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-010': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{62.7823424510713: 0, 52.56514006365846: 1, 49.848485720687194: 2, 58.460808466685506: 3, 54.962085672457654: 4, 66.4178246333435: 5, 67.16837029940126: 6, 63.29101761335433: 7, 61.80691083703386: 8, 61.016714098548576: 9, 67.60411305474386: 10, 59.65649800050771: 11, 56.68100707444269: 12, 47.77016950368968: 13, 47.04308434079398: 14, 45.87607771248019: 15, 47.271483211022: 16, 47.13747583808863: 17, 55.92833068597696: 18, 58.510189717469046: 19, 58.87830769566092: 20, 64.39380104633578: 21, 65.80278774046707: 22, 66.63966685460014: 23, 67.34262858337054: 24, 63.98603359087144: 25, 53.44138399821439: 26, 56.76717105562001: 27, 59.943585851006226: 28, 59.078777065174 59: 29, 65.84735276151012: 30, 60.37981383344575: 31, 65.53044476889583: 32, 62.32829235592835: 33, 66.515671040 02281: 34, 58.98945293224972: 35, 55.17016869623751: 36, 56.248720774459294: 37, 45.12170216119285: 38, 45.49833 1064223315: 39, 52.34011253155522: 40, 53.37565597040144: 41, 60.16407491992233: 42, 52.042306606399926: 43, 62. 20380189984184: 44, 56.20487555501632: 45, 55.01939540375902: 46, 56.22027071245341: 47}

```



The green bars represent the observation value for each respective month. There are slight trenches in this multimodal histogram.

```
In [ ]: df_FossilBrown_Log.describe(include = 'all') # Description Tables
```

	Price_Log	Fossil_Brown_Log	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	-0.139619	-2.110303
std	4.072442	6.593685	NaN	5.587297	8.538619
min	14.539966	45.121702	NaN	-11.127019	-19.280349
25%	22.001649	53.424952	NaN	-3.427659	-7.702280
50%	25.195447	58.694249	NaN	0.368118	-4.053735
75%	27.317301	62.909511	NaN	3.251101	4.132390
max	33.035963	67.604113	NaN	11.455739	16.215864

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(ypred))
```

```
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.73662909849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: print(list(FossilBrown_ypred))
```

```
[63.06731283062955, 53.59687265001218, 49.706423671387874, 60.056750622987295, 56.55074828770661, 64.4701071510188, 64.63114069751558, 63.325713100570056, 62.515265261018286, 62.01355917043847, 64.7044422146824, 61.03569663922449, 58.392735824455045, 46.3408970994633, 45.08382205204928, 42.9798231063273, 45.4831549684863, 45.24935032569962, 57.61457682875125, 60.099391384635005, 60.41125895765093, 63.8165040913031, 64.30534408955643, 64.52228556316315, 64.66223445562477, 63.64609725932625, 54.72875591318566, 58.47899399166512, 61.25411763909487, 60.576643602385644, 64.3182756319178, 61.573683028856706, 64.22294670634315, 62.819588405376024, 64.4935949858076, 60.5033401041932, 56.78600785665822, 57.95122349633304, 41.5631462339131, 42.27599184009283, 53.29651552339284, 54.645932822997636, 61.41749811637763, 52.89293556552304, 62.74885484784244, 57.90562714953958, 56.61588040436445, 57.921654278615954]
```

```
In [ ]: dfFossilBrown_Quad = ({ "Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, "Fossil_Brown_Quad" : [63.06731283062955, 53.59687265001218, 49.706423671387874, 60.056750622987295], "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07']}, {"Date": "2015-01", "Price": 63.06731283062955, "Fossil_Brown": 53.59687265001218, "Date": "2015-02", "Price": 53.59687265001218, "Fossil_Brown": 49.706423671387874, "Date": "2015-03", "Price": 49.706423671387874, "Fossil_Brown": 49.706423671387874, "Date": "2015-04", "Price": 49.706423671387874, "Fossil_Brown": 45.08382205204928, "Date": "2015-05", "Price": 45.08382205204928, "Fossil_Brown": 42.9798231063273, "Date": "2015-06", "Price": 42.9798231063273, "Fossil_Brown": 42.9798231063273, "Date": "2015-07", "Price": 42.9798231063273, "Fossil_Brown": 42.9798231063273})  
print(dfFossilBrown_Quad) #Dataframes  
df_FossilBrown_Quad= pd.DataFrame.from_dict(dfFossilBrown_Quad, orient = "columns")  
print(df_FossilBrown_Quad)
```

```
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis,indicating it is non-stationary ")  
  
adfuller_test(df_FossilBrown_Quad["Fossil_Brown_Quad"])  
  
test_result=adfuller(df_FossilBrown_Quad["Fossil_Brown_Quad"])
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.888735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Fossil_Brown_Quad': [63.06731283062955, 53.59687265001218, 49.706423671387874, 60.056750622987295, 56.55074828770661, 64.4701071510188, 64.63114069751558, 63.325713100570056, 62.515265261018286, 62.01355917043847, 64.7044422146824, 61.03569663922449, 58.392735824455045, 46.3408970994633, 45.08382205204928, 42.9798231063273, 45.4831549684863, 45.24935032569962, 57.61457682875125, 60.099391384635005, 60.41125895765093, 63.8165040913031, 64.30534408955643, 64.52228556316315, 64.66223445562477, 63.64609725932625, 54.72875591318566, 58.47899399166512, 61.25411763909487, 60.576643602385644, 64.3182756319178, 61.573683028856706, 64.22294670634315, 62.819588405376024, 64.4935949858076, 60.5033401041932, 56.78600785665822, 57.95122349633304, 41.5631462339131, 42.27599184009283, 53.29651552339284, 54.645932822997636, 61.41749811637763, 52.89293556552304, 62.74885484784244, 57.90562714953958, 56.61588040436445, 57.921654278615954], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Quad	Fossil_Brown_Quad	Dates
0	27.188078	63.067313	2015-01
1	23.225548	53.596873	2015-02
2	22.878236	49.706424	2015-03
3	24.055117	60.056751	2015-04
4	23.602709	56.550748	2015-05
5	27.724722	64.470107	2015-06
6	30.587839	64.631141	2015-07
7	26.702482	63.325713	2015-08
8	24.901804	62.515265	2015-09
9	24.518415	62.013559	2015-10
10	25.119121	64.704442	2015-11
11	25.672300	61.035697	2015-12
12	19.546363	58.392736	2016-01
13	17.633406	46.340897	2016-02
14	17.644052	45.083822	2016-03
15	17.075344	42.979823	2016-04
16	17.328255	45.483155	2016-05
17	19.736629	45.249350	2016-06
18	20.093546	57.614577	2016-07
19	20.123369	60.099391	2016-08
20	21.007992	60.411259	2016-09
21	24.868735	63.816504	2016-10
22	26.000055	64.305344	2016-11
23	28.599305	64.522286	2016-12
24	36.035067	64.662234	2017-01
25	24.712150	63.646097	2017-02
26	21.194644	54.728756	2017-03
27	21.456173	58.478994	2017-04
28	22.201511	61.254118	2017-05
29	23.174311	60.576644	2017-06
30	22.771345	64.318276	2017-07
31	22.319233	61.573683	2017-08
32	22.995762	64.222947	2017-09
33	26.665789	62.819588	2017-10
34	27.438370	64.493595	2017-11
35	27.292830	60.503340	2017-12
36	23.278006	56.786008	2018-01
37	25.188915	57.951223	2018-02
38	20.328049	41.563146	2018-03
39	21.006372	42.275992	2018-04
40	25.544593	53.296516	2018-05
41	26.879918	54.645933	2018-06
42	28.703098	61.417498	2018-07
43	30.170478	52.892936	2018-08
44	34.272280	62.748855	2018-09
45	30.169486	57.905627	2018-10
46	28.070862	56.615880	2018-11
47	28.297338	57.921654	2018-12

ADF Test Statistic : -3.061383654560723

p-value : 0.029546124471154987

#Lags Used : 0

Number of Observations : 47

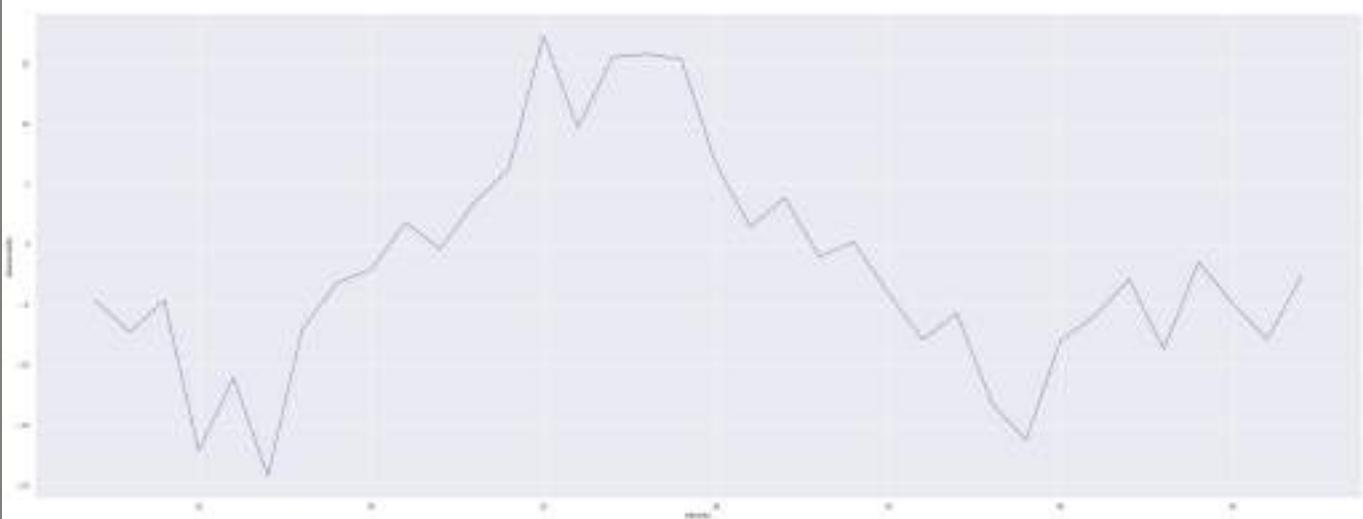
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```
In [ ]: df_FossilBrown_Quad['First Difference'] = df_FossilBrown_Quad["Fossil_Brown_Quad"]- df_FossilBrown_Quad["Fossil_Brown_Quad"].shift(1)
df_FossilBrown_Quad['Seasonal Difference']=df_FossilBrown_Quad["Fossil_Brown_Quad"]- df_FossilBrown_Quad["Fossil_Brown_Quad"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted quadratic fossil brown prices of energy per EUR")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilBrown_Quad.head() #Predictive Quadratic Seasonality Plot
```

```
df_FossilBrown_Quad['Seasonal Difference'].plot()  
# Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60b10f910>

Seasonal Difference of average monthly predicted quadratic fossil brown prices of energy per EUR/MWH

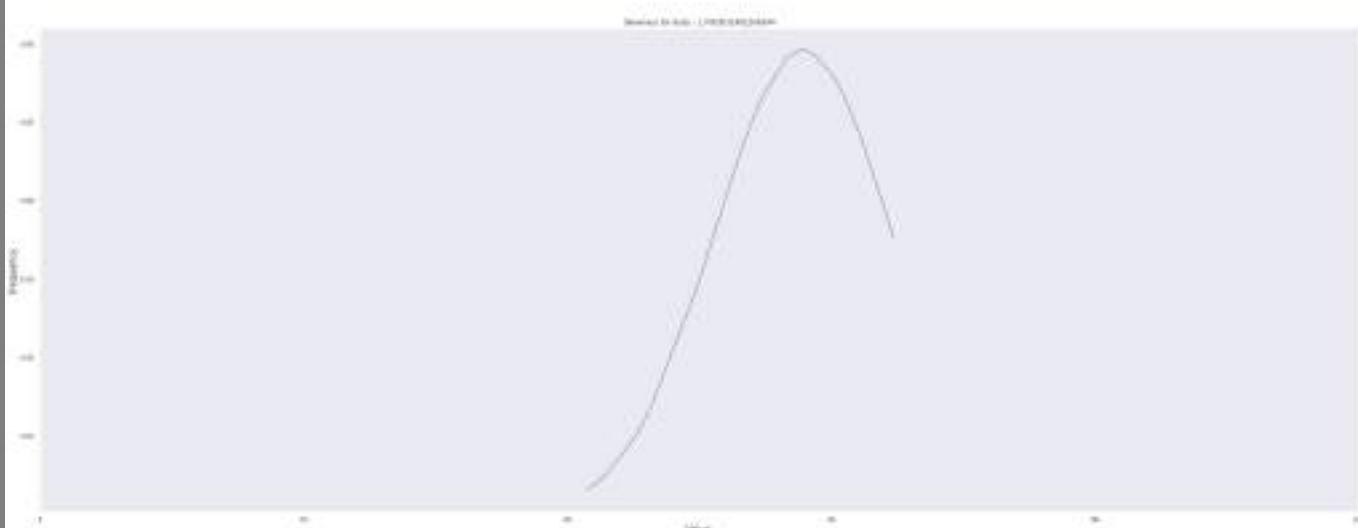


The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
FossilBrown_QuadResults_mean = np.mean(df_FossilBrown_Quad["Fossil_Brown_Quad"])  
FossilBrown_QuadResults_std = np.std(df_FossilBrown_Quad["Fossil_Brown_Quad"])  
  
FossilBrown_QuadResultspdf = stats.norm.pdf(df_FossilBrown_Quad["Fossil_Brown_Quad"].sort_values(), FossilBrown_QuadResults_mean, FossilBrown_QuadResults_std)  
  
plt.plot(df_FossilBrown_Quad["Fossil_Brown_Quad"].sort_values(), FossilBrown_QuadResultspdf)  
plt.xlim([0,100])  
plt.xlabel("Value ", size=15)  
plt.title(f'Skewness for data: {skew(df_FossilBrown_Quad["Fossil_Brown_Quad"])}')  
plt.suptitle("Frequency distribution of average monthly fossil brown quadratic energy prices per EUR/MWH (scaled in decimal)", size=10)  
plt.ylabel("Frequency", size=15)  
plt.grid(True, alpha=0.3, linestyle="--")  
plt.show()
```

Frequency distribution of average monthly fossil brown quadratic energy prices per EUR/MWH (scaled in decimal).



This bell shaped curve is skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)
Fossil_Brown_Quad_Dict = {key: i for i, key in enumerate(df_FossilBrown_Quad["Fossil_Brown_Quad"])}

```

def Hist_Fossil_Brown_Quad(Fossil_Brown_Quad_Dict):
    for k, v in Fossil_Brown_Quad_Dict.items(): print(f"{v}:{k}")
print(Fossil_Brown_Quad_Dict)

plt.bar(list(Fossil_Brown_Quad_Dict.values()), Fossil_Brown_Quad_Dict.keys(), color='g') #Predictive Quadratic /
plt.title("Average monthly fossil brown quadratic fossil brown energy prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

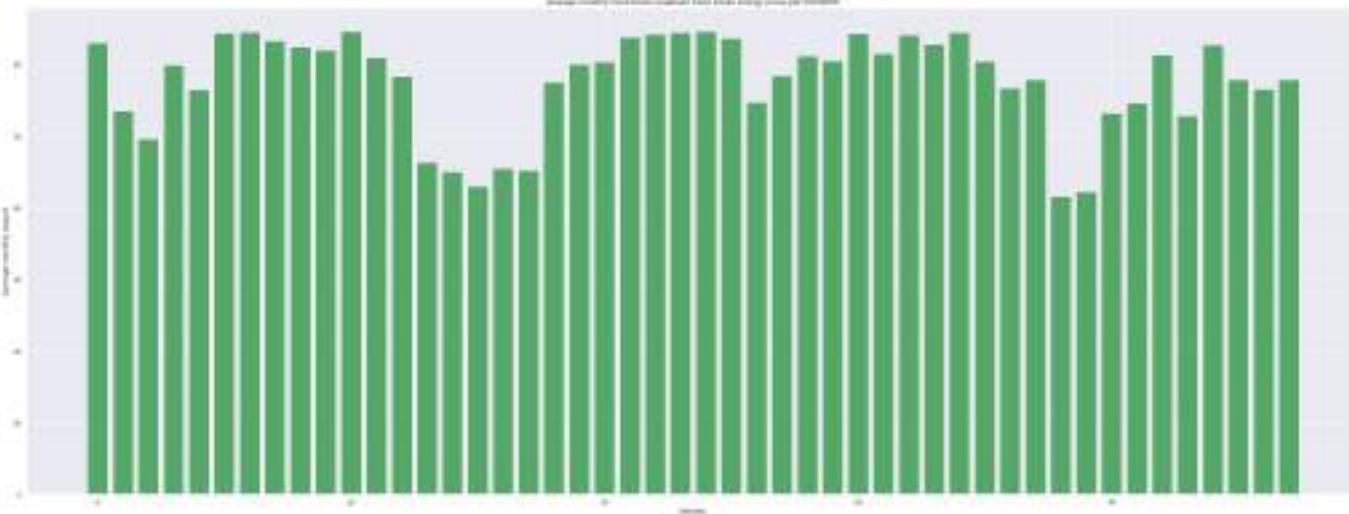
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{63.06731283062955: 0, 53.59687265001218: 1, 49.706423671387874: 2, 60.056750622987295: 3, 56.55074828770661: 4, 64.4701071510188: 5, 64.63114069751558: 6, 63.325713100570056: 7, 62.515265261018286: 8, 62.01355917043847: 9, 64.7044422146824: 10, 61.03569663922449: 11, 58.392735824455045: 12, 46.3408970994633: 13, 45.08382205204928: 14, 42.9798231063273: 15, 45.4831549684863: 16, 45.24935032569962: 17, 57.61457682875125: 18, 60.099391384635005: 19, 60.41125895765093: 20, 63.8165040913031: 21, 64.30534408955643: 22, 64.52228556316315: 23, 64.66223445562477: 24, 63.64609725932625: 25, 54.72875591318566: 26, 58.47899399166512: 27, 61.25411763909487: 28, 60.5766436023856: 29, 64.3182756319178: 30, 61.573683028856706: 31, 64.22294670634315: 32, 62.819588405376024: 33, 64.49359498: 34, 60.5033401041932: 35, 56.78600785665822: 36, 57.95122349633304: 37, 41.5631462339131: 38, 42.27599184: 39, 53.29651552339284: 40, 54.645932822997636: 41, 61.41749811637763: 42, 52.89293556552304: 43, 62.7488: 44, 57.90562714953958: 45, 56.61588040436445: 46, 57.921654278615954: 47}

```



The green bars represent the observation value for each respective month. There are trenches within this roughly uniform histogram.

```
In [ ]: df_FossilBrown_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Fossil_Brown_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	-0.109482	-1.934799
std	4.174498	6.811092	NaN	5.609395	9.049760
min	17.075344	41.563146	NaN	-16.388077	-19.220757
25%	21.390791	54.708050	NaN	-2.693777	-7.411038
50%	24.615282	60.255325	NaN	0.139949	-3.063550
75%	27.214266	63.131913	NaN	2.576298	2.223856
max	36.035067	64.704442	NaN	12.365227	17.305200

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
```

```

plot_acf(FossilBrown_Logpred)
plt.suptitle(" Autocorrelations of average monthly FossilBrown Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show()

from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilBrown_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly FossilBrown Log")

```

```

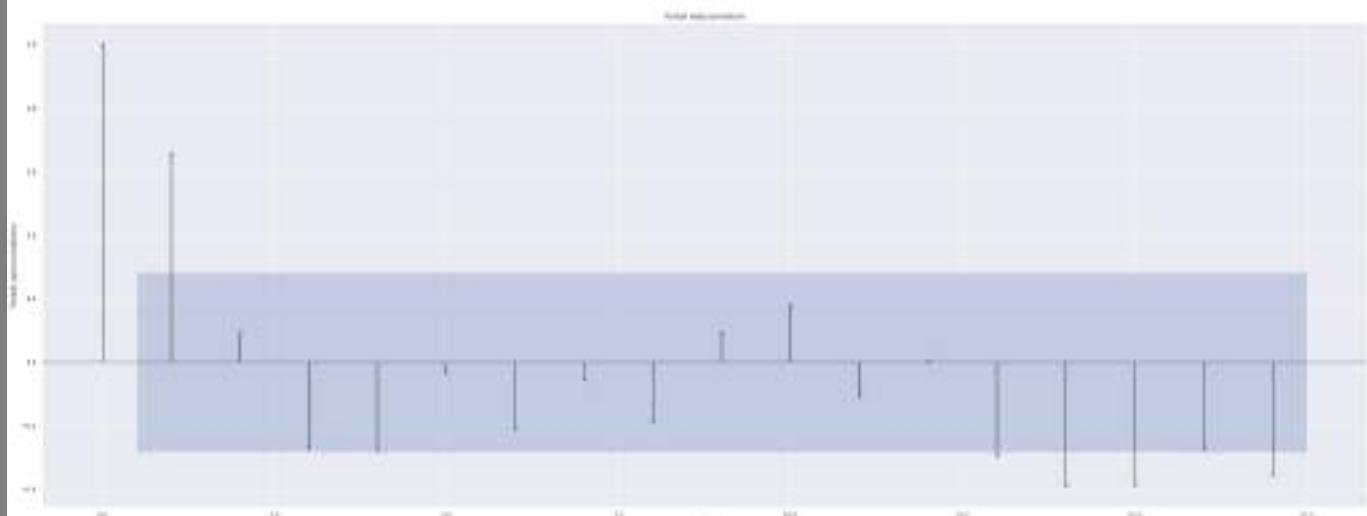
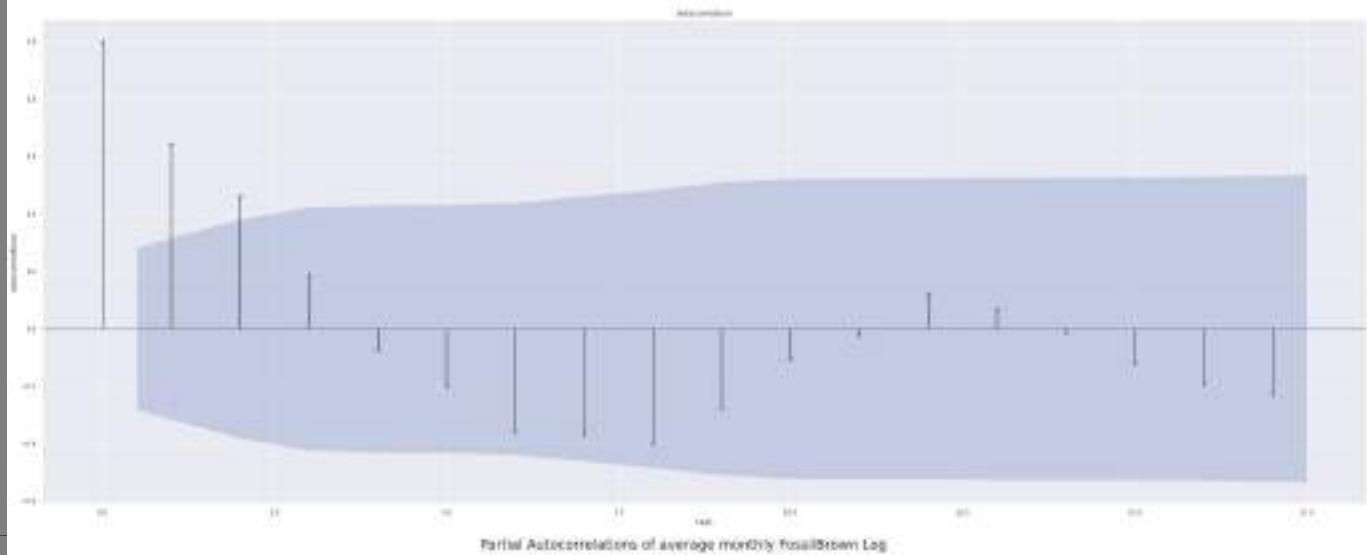
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
FossilBrown_Log_Autocorrelations = sm.tsa.acf(FossilBrown_Logpred, fft=False) #Autocorrelations
print(FossilBrown_Log_Autocorrelations)

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.          0.64180898  0.46169082  0.18332352 -0.07261451 -0.2013063
 -0.35871785 -0.3671282 -0.40057989 -0.27958281 -0.10798328 -0.02041418
 0.11975137  0.0659548 -0.01206095 -0.12048615 -0.19380105 -0.22794926
 -0.16097635 -0.11688761 -0.06699359  0.00847704  0.15725873  0.23255359
 0.30951671  0.2830988  0.16584249  0.09239984 -0.02215652 -0.06250553
 -0.07666395 -0.11878183 -0.10191971 -0.1442953 -0.01831705 -0.02159913
 0.01892982  0.06161651  0.00140425  0.01096924 -0.04281125]
```

Autocorrelations of average monthly FossilBrown Log



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```

In [ ]:
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilBrown_ypred)
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilBrown")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilBrown_ypred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilBrown")

```

```

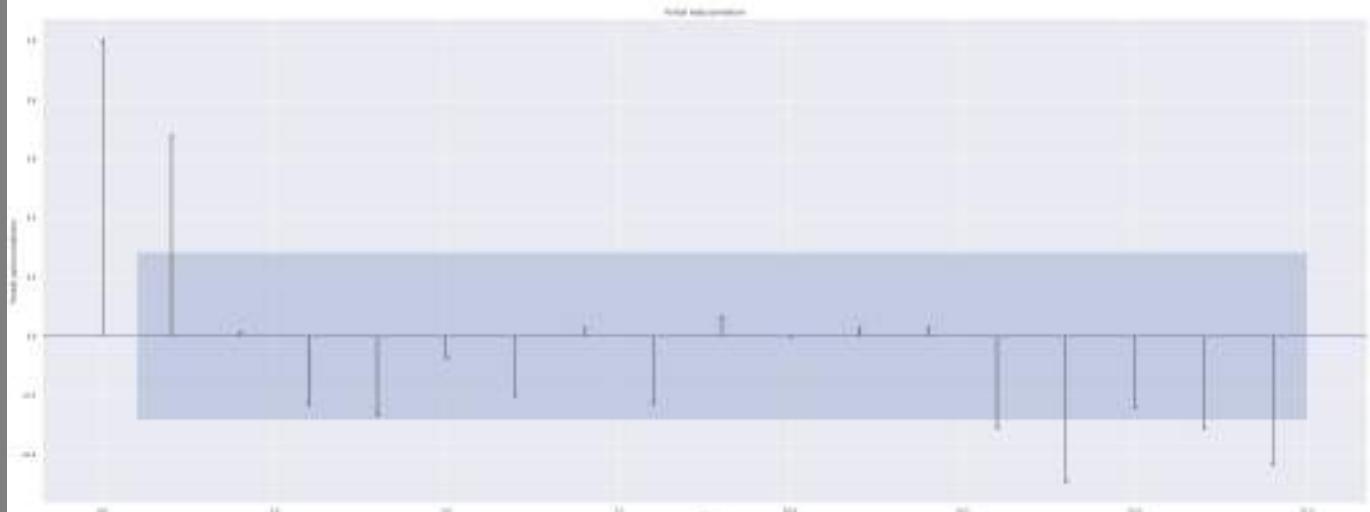
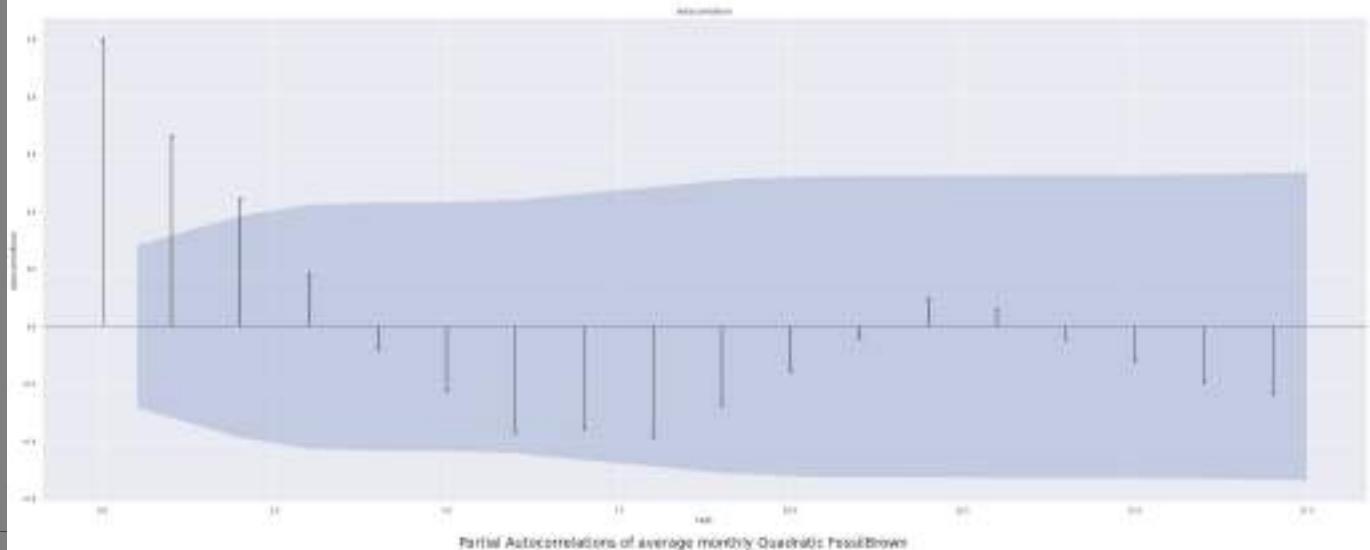
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
FossilBrown_Qquad_Autocorrelations = sm.tsa.acf(FossilBrown_ypred, fft=False) #Autocorrelations
print(FossilBrown_Qquad_Autocorrelations)

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.00000000e+00  6.61735060e-01  4.43975255e-01  1.80218966e-01
 -7.91306306e-02 -2.24093786e-01 -3.67593425e-01 -3.54885830e-01
 -3.84424823e-01 -2.74760713e-01 -1.55092569e-01 -3.84341067e-02
 9.68695962e-02  5.82632796e-02 -4.20930128e-02 -1.19912563e-01
 -1.93410369e-01 -2.32808113e-01 -1.85246204e-01 -1.27716453e-01
 -7.13165285e-02  7.18341988e-02  2.26577001e-01  2.97590791e-01
 3.42986671e-01  3.05101903e-01  1.70655714e-01  3.39921713e-02
 -5.40444766e-02 -9.42958471e-02 -8.83442642e-02 -1.26892402e-01
 -1.09510125e-01 -1.23724248e-01 -3.19892920e-02 -1.68339142e-02
 4.07998507e-02  9.32263583e-02  9.38362952e-04  2.26537171e-03
 -2.94281536e-02]
```

Autocorrelations of average monthly Ququad of FossilBrown



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

In []: `from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot`

In []: `from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot`
`plot_acf(predictionsFossilBrown)`

```

plt.suptitle(" Autocorrelations of average monthly Linear FossilBrown")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show()
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(predictionsFossilBrown)
plt.suptitle("Partial Autocorrelations of average monthly Linear FossilBrown")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show()
FossilBrown_Pred_Autocorrelations = sm.tsa.acf(predictionsFossilBrown, fft=False) #Autocorrelations
print(FossilBrown_Pred_Autocorrelations)

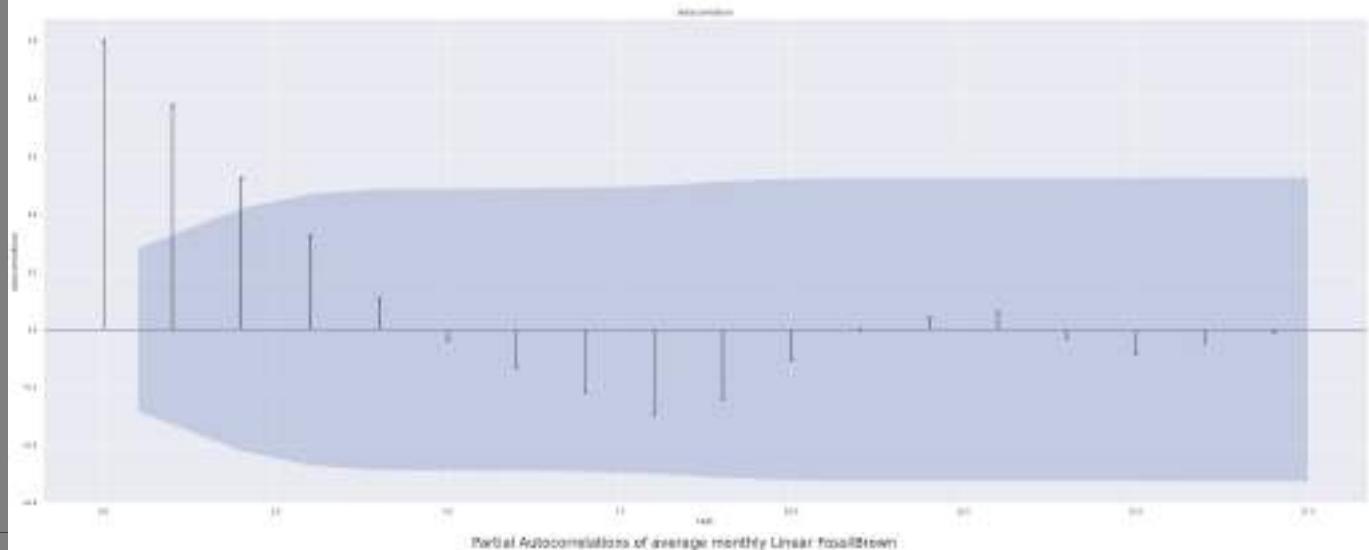
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

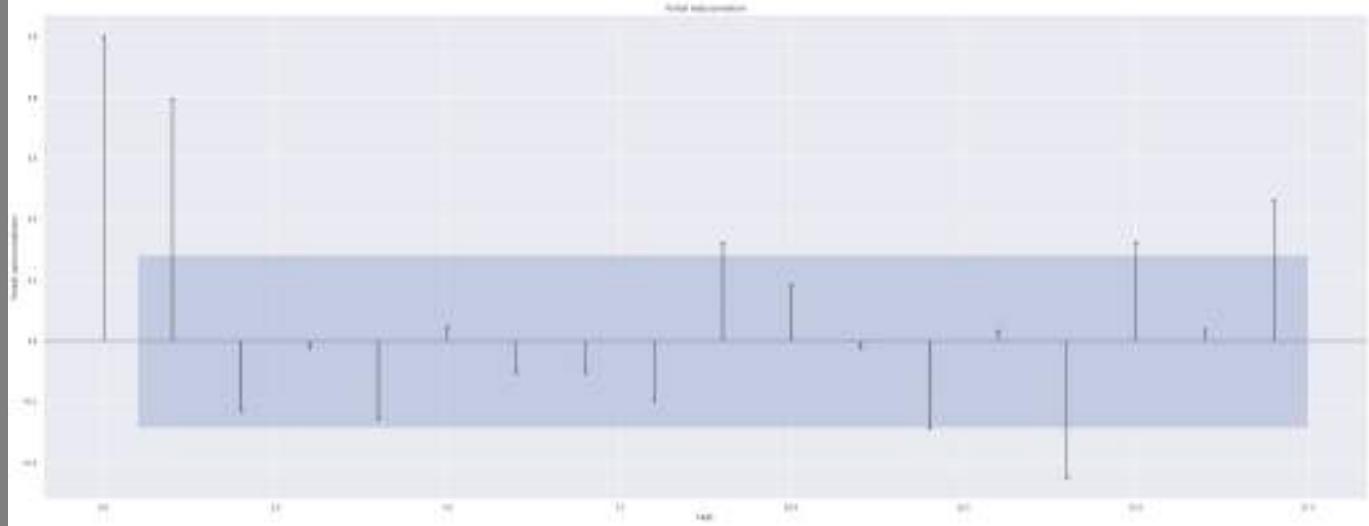
FutureWarning,

```
[ 1.0000000e+00  7.77956931e-01  5.24250500e-01  3.23980125e-01
 1.07028578e-01 -3.60532396e-02 -1.30254803e-01 -2.14671334e-01
 -2.92492069e-01 -2.40360570e-01 -1.03132122e-01 -1.29994731e-04
 4.12510174e-02  5.95304581e-02 -2.97302907e-02 -8.26016779e-02
 -4.42595047e-02 -9.62022191e-03  2.61107117e-02  1.18842238e-02
 -4.25037632e-03 -4.63630833e-03  2.97453465e-02  7.37711560e-02
 2.35072578e-02 -7.93840645e-02 -1.44789264e-01 -2.21944605e-01
 -2.72163150e-01 -2.77044335e-01 -2.71423639e-01 -2.54543382e-01
 -1.99886952e-01 -1.05612474e-01 -1.41939737e-02  4.81982436e-02
 7.79627111e-02  9.66167880e-02  7.93061418e-02  6.38831956e-02
 5.25335207e-02]
```

Autocorrelations of average monthly Linear FossilBrown



Partial Autocorrelations of average monthly Linear FossilBrown



In []: FossilBrown_Pred_Autocorrelations = sm.tsa.acf(predictionsFossilBrown, fft=False) #Autocorrelations
print(FossilBrown_Pred_Autocorrelations)

```
[ 1.00000000e+00  7.77956931e-01  5.24250500e-01  3.23980125e-01
 1.07028578e-01 -3.60532396e-02 -1.30254803e-01 -2.14671334e-01
-2.92492069e-01 -2.40360570e-01 -1.03132122e-01 -1.29994731e-04
 4.12510174e-02  5.95304581e-02 -2.97302907e-02 -8.26016779e-02
-4.42595047e-02 -9.62022191e-03  2.61107117e-02  1.18842238e-02
-4.25037632e-03 -4.63630833e-03  2.97453465e-02  7.37711560e-02
 2.35072578e-02 -7.93840645e-02 -1.44789264e-01 -2.21944605e-01
-2.72163150e-01 -2.77044335e-01 -2.71423639e-01 -2.54543382e-01
-1.99886952e-01 -1.05612474e-01 -1.41939737e-02  4.81982436e-02
 7.79627111e-02  9.66167880e-02  7.93061418e-02  6.38831956e-02
 5.25335207e-02]
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

In []:

The next resource analyzed was hydro reservoir power.

In []:

```
modelhydrowater = stats.linregress([2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.012534818941,
[64.9490188172043,
 56.383854166666666,
 55.522462987886975,
 58.35408333333333,
 57.29405913978498,
 65.9749027777778,
 71.07204301075271,
 63.99806451612899,
 60.254791666666634,
 59.40676510067113,
 60.72679166666668,
 61.901760752688226,
 45.57872311827956,
 36.75208333333374,
 36.81800807537014,
 32.61866666666666,
 34.691370967741896,
 46.266319444444434,
 47.50201612903221,
 47.6023387096774,
 50.4055972222224,
 60.182429530201404,
 62.58105555555558,
 67.5951344086021,
 79.4920833333331,
 59.83779761904767,
 50.95989232839837,
 51.71791666666662,
 53.77262096774189,
 56.2582222222224,
 55.252580645161316,
 54.08432795698931,
 55.81655555555558,
 63.92528859060403,
 65.43065277777781,
 65.15127688172035,
 56.51197580645163,
 60.877098214285674,
 48.279717362045766,
 50.40073611111113,
 61.633763440860214,
 64.34813888888884,
 67.78344086021498,
 70.363911290932262,
 76.9140416666666,
 70.36221476510062,
 67.0426075268817,
 66.62351388888881])
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per

EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly hydro reservoir outputs versus the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Dataframes analyzed by resource
dfhydro = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139784
                 "Hydro" : [2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.184139784946, 2516.0125348189417]
print(dfhydro)
df_hydro= pd.DataFrame.from_dict(dfhydro, orient = "columns")
print(df_hydro)
df_hydro["Ratio"] = df_hydro["Hydro"]/df_hydro["Price"]
pdToListHydro = list(df_hydro["Ratio"])
print(pdToListHydro)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.095989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Hydro': [2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.184139784946, 2531.1682892906815, 2412.5255376344085, 1963.0631720430108, 2261.15694444444444, 2276.9193548387098, 2404.3902777777776, 1943.42799461642, 4075.5846774193546, 4881.568965517241, 3901.5450874831763, 5119.295833333334, 4581.743279569892, 2983.793055555554, 2556.6971736204578, 2460.0201612903224, 2303.46111111111, 2500.489932885906, 2501.2972222222224, 2805.2970430107525, 2156.951612903226, 1874.8497023809523, 2348.4589502018844, 1611.415277777779, 1639.616935483871, 1448.644444444444, 1290.983870967742, 1260.6908602150538, 1570.509722222223, 1229.2845637583894, 1371.33888888889, 1452.9569892473119, 2271.896505376344, 2773.2038690476193, 4378.419919246298, 4159.898611111111, 2931.19623655914, 3362.529166666667, 2789.228802153432, 2296.0295698924733, 2267.641666666667, 2172.1919463087247, 2763.0241935483873, 2665.9], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}
```

	Price	Hydro	Dates
--	-------	-------	-------

0	64.949019	2572.339700	2015-01
1	56.383854	3712.690476	2015-02
2	55.522463	3081.620458	2015-03
3	58.354083	2516.012535	2015-04
4	57.294059	2798.184140	2015-05
5	65.974903	2531.168289	2015-06
6	71.072043	2412.525538	2015-07
7	63.998065	1963.063172	2015-08
8	60.254792	2261.156944	2015-09
9	59.406765	2276.919355	2015-10
10	60.726792	2404.390278	2015-11
11	61.901761	1943.427995	2015-12
12	45.578723	4075.584677	2016-01
13	36.752083	4881.568966	2016-02
14	36.818008	3901.545087	2016-03
15	32.618667	5119.295833	2016-04
16	34.691371	4581.743280	2016-05
17	46.266319	2983.793056	2016-06
18	47.502016	2556.697174	2016-07
19	47.602339	2460.020161	2016-08
20	50.405597	2303.461111	2016-09
21	60.182430	2500.489933	2016-10
22	62.581056	2501.297222	2016-11
23	67.595134	2805.297043	2016-12
24	79.492083	2156.951613	2017-01
25	59.837798	1874.849702	2017-02
26	50.959892	2348.458950	2017-03
27	51.717917	1611.415278	2017-04
28	53.772621	1639.616935	2017-05
29	56.258222	1448.644444	2017-06
30	55.252581	1290.983871	2017-07
31	54.084328	1260.690860	2017-08
32	55.816556	1570.509722	2017-09
33	63.925289	1229.284564	2017-10
34	65.430653	1371.338889	2017-11
35	65.151277	1452.956989	2017-12
36	56.511976	2271.896505	2018-01
37	60.877098	2773.203869	2018-02
38	48.279717	4378.419919	2018-03
39	50.400736	4159.898611	2018-04
40	61.633763	2931.196237	2018-05
41	64.348139	3362.529167	2018-06
42	67.783441	2789.228802	2018-07
43	70.363911	2296.029570	2018-08
44	76.914042	2267.641667	2018-09
45	70.362215	2172.191946	2018-10
46	67.042608	2763.024194	2018-11
47	66.623514	2665.900000	2018-12

[39.60552055610406, 65.8466954957713, 55.50222903974214, 43.1163063679167, 48.83899276464233, 38.365623634435295, 33.94478947607304, 30.67378969794114, 37.52659136145235, 38.32761051675219, 39.59356672382155, 31.395358887784525, 89.41857951665217, 132.8242788644791, 105.96839132351556, 156.94374897809027, 132.07155415766834, 64.49168836821842, 53.82291915095909, 51.67855672583178, 45.6985183799308, 41.548504312726116, 39.96892030690864, 41.50146408546457, 27.13416886884827, 31.332197657357437, 46.08445667561115, 31.157776291796583, 30.4916685475955, 25.749915074142248, 23.36513255839751, 23.309725900960125, 28.13698743303958, 19.230019775602145, 20.95866127984951, 22.30128185952668, 40.20203634637343, 45.55414023326174, 90.68859882531778, 82.53646537900543, 47.55828742101278, 52.25526681467521, 41.149117937306535, 32.63078370414942, 29.48280466776494, 30.871568690104986, 41.21295837785713, 40.014400988306384]

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy

per EUR/MWH. This is the linear model used for the average monthly hydro reservoir outputs versus the average monthly prices of energy per EUR/MWH.

In []:

```
#Linear OLS regression
hydro1 = hydro_water
hydro1 = sm.add_constant(hydro1)
modelhydropowerreg = sm.OLS(Price_Actual, hydro1).fit()
predictionshydropower = modelhydropowerreg.predict(hydro1)

modelhydropowerreg.summary()
#OLS Linear Summary Table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.325		
Model:	OLS	Adj. R-squared:	0.310		
Method:	Least Squares	F-statistic:	22.11		
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	2.37e-05		
Time:	05:24:32	Log-Likelihood:	-170.22		
No. Observations:	48	AIC:	344.4		
Df Residuals:	46	BIC:	348.2		
Df Model:	1				
Covariance Type:	nonrobust				
coef	std err	t	P> t	[0.025	0.975]
const	73.9980	3.648	20.283	0.000	66.655 81.341
x1	-0.0062	0.001	-4.702	0.000	-0.009 -0.004
Omnibus:	3.427	Durbin-Watson:	0.386		
Prob(Omnibus):	0.180	Jarque-Bera (JB):	1.866		
Skew:	0.195	Prob(JB):	0.393		
Kurtosis:	2.117	Cond. No.	8.18e+03		

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 8.18e+03. This might indicate that there are strong multicollinearity or other numerical problems.

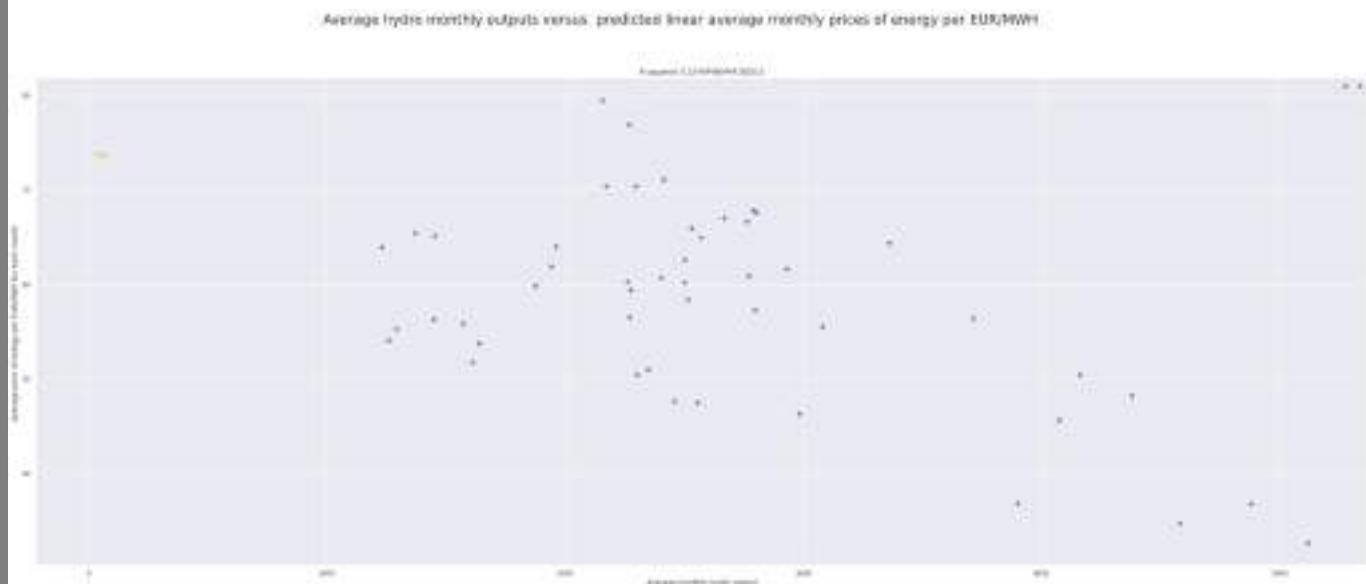
In []:

In []:

```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(hydro_water,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(hydro_water,Price_Actual,"o")
f = lambda x: -0.006186 *x + 73.998036
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.title(f"R squared: {modelhydropowerreg.rvalue**2}")
plt.legend("#")
plt.suptitle("Average hydro monthly outputs versus predicted linear average monthly prices of energy per EUR/MWH")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly hydro output')
plt.show()
```

slope: -0.006186 intercept: 73.998036



The blue dots represent the observations and the orange line is the model of best fit. There is a moderately negative correlation between the output and the respective the average monthly prices of energy per EUR/MWH. As the price decreases, the output increases. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: print(predictionshydropower)
#Linear OLS Predicted Values
```

```
[58.08650622 51.03272344 54.9362862 58.43492496 56.6895171 58.34117714
59.07505676 61.85526261 60.01136619 59.91386582 59.12537844 61.97671846
48.78799584 43.80247892 49.86453955 42.33198956 45.65708831 55.54140987
58.18326503 58.78127281 59.74968846 58.53094196 58.52594837 56.64551935
60.65594134 62.40091809 59.47134868 64.03042477 63.85597988 65.03726418
66.01249343 66.19987465 64.28345121 66.39414224 65.51544739 65.01058839
59.94493528 56.8440357 46.91477046 48.26646143 55.86675409 53.19869015
56.74491147 59.79565719 59.97125412 60.56167034 57.50777724 56.90700336]
```

```
In [ ]: #Linear OLS regression residuals
influenceHydroreg = modelhydropowerreg.get_influence()

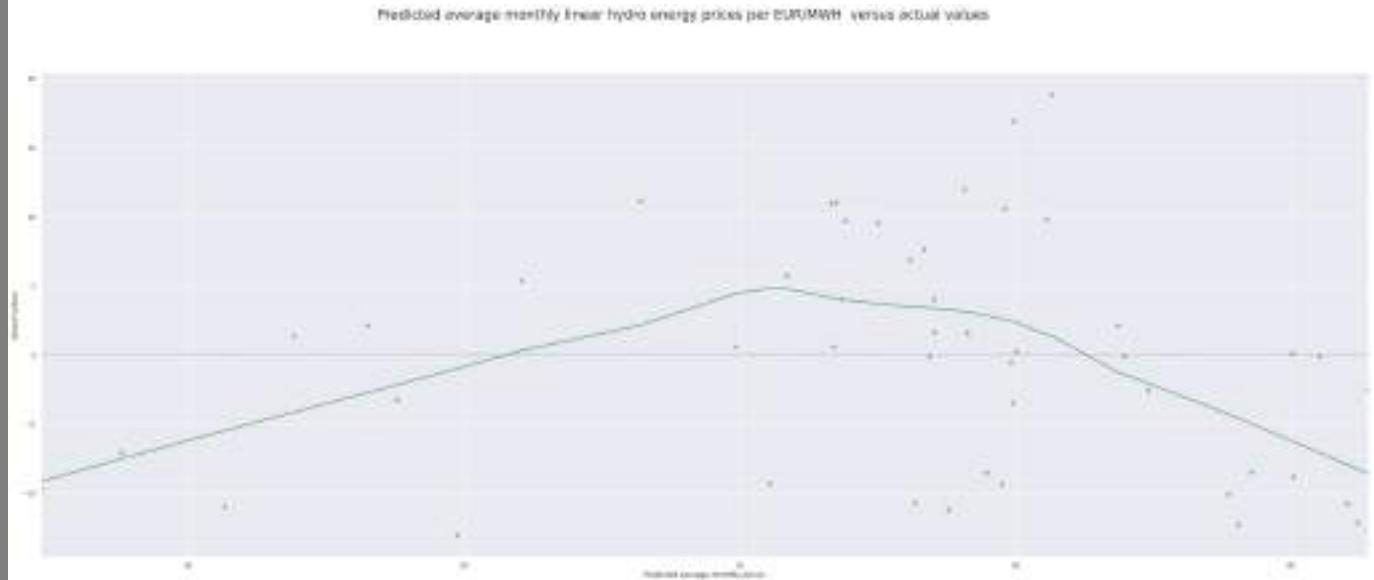
standardized_residualsHydro = influenceHydroreg.resid_studentized_internal

print(standardized_residualsHydro)
```

```
[ 0.80895074  0.6402248   0.06928352 -0.00953042  0.07129275  0.89991207
 1.4148346   0.25386521   0.02873628 -0.0598553   0.18886608 -0.00888328
-0.38848049 -0.88804896 -1.56974736 -1.24301769 -1.35769508 -1.09517813
-1.25912319 -1.31809949 -1.10269872  0.19470122  0.47807364  1.29131463
 2.22583673 -0.30411172 -1.00413115 -1.46905208 -1.20226272 -1.05202182
-1.29570143 -1.46042657 -1.01125531 -0.29792045 -0.0101847   0.01685708
-0.40522352  0.47556262  0.16731775  0.25918997  0.68065227  1.32333672
 1.30170574  1.24723159  1.99998134  1.15792516  1.12398831  1.14568746]
```

```
In [ ]: plt.suptitle("Predicted average monthly linear hydro energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend("..#")
sns.residplot(x = predictionshydropower, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60aaba7d0>
```



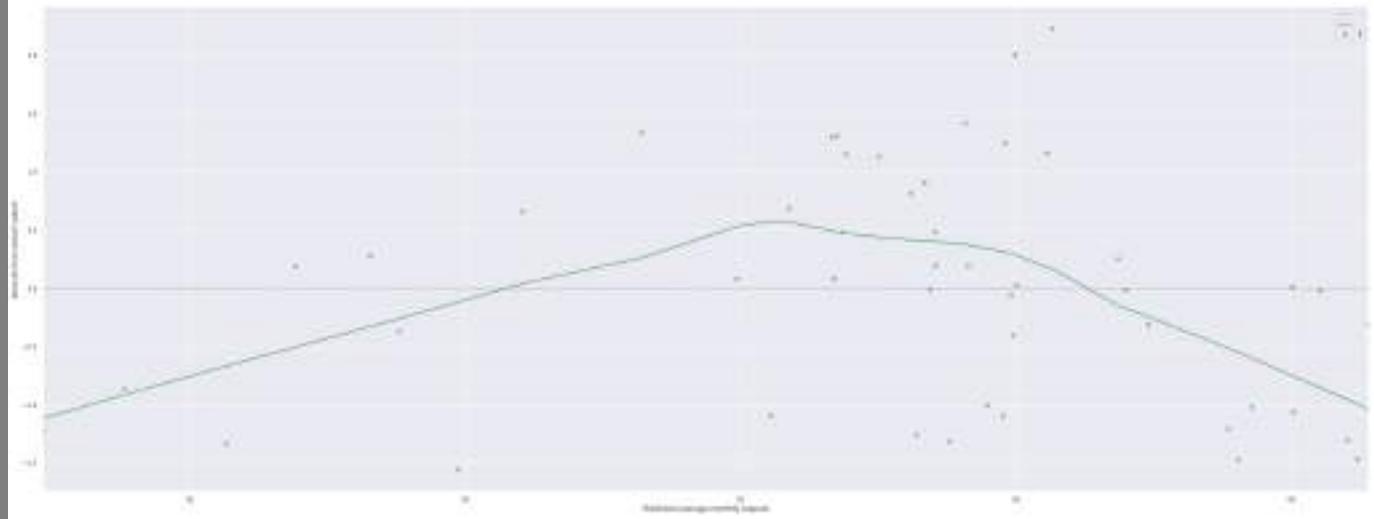
As one can observe, there is a hump along the lowess line in the residual plot. Furthermore, the observations are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionshydropower, y =standardized_residualsHydro, lowess = True, color="g")

plt.suptitle("Hydro power residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60aab1d0>
Hydro power residuals from linear model versus predicted values
```



As one can observe, there is a hump along the lowess line in the residual plot. Furthermore, the observations are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

This is the average monthly logarithmic outputs of hydro power the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
Loghydrovalues = ((np.log(hydro_water)))
Log = np.polyfit(np.log(Price_Actual), hydro1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(hydro1), Price_Actual)
```

```
Hydro_Log = sm.OLS(Price_Actual, hydro1).fit()

Hydro_Logpred = Hydro_Log.predict(hydro1)

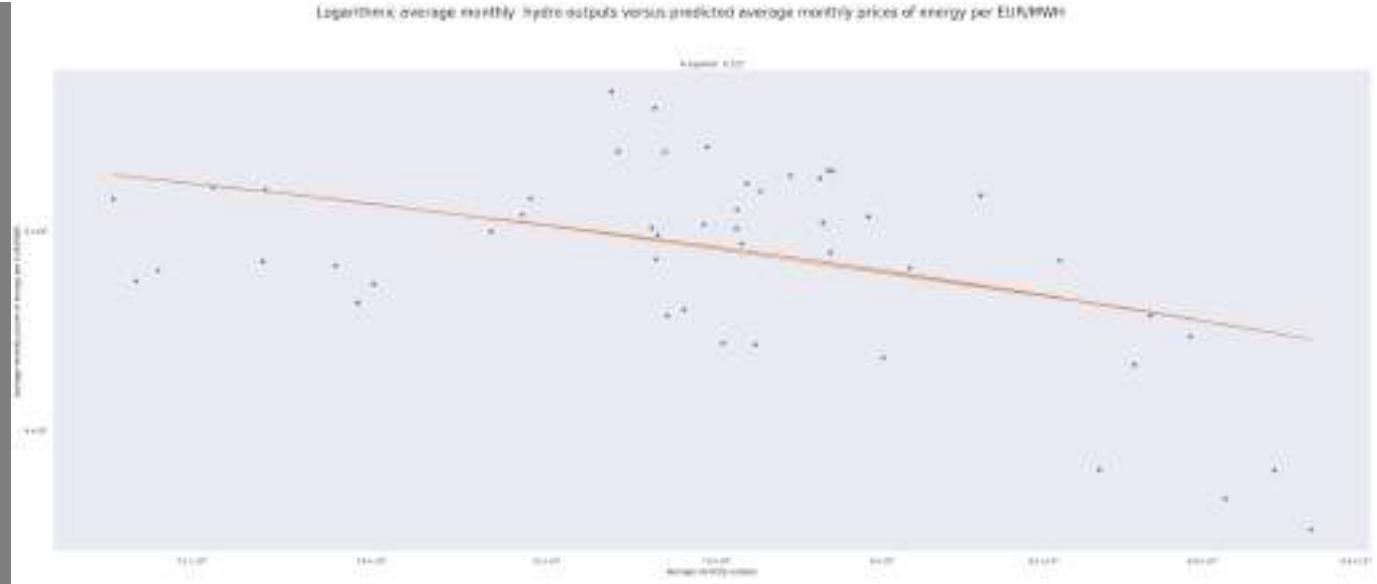
#OLS Logarithmic summary table
Hydro_Log.summary()
#Log
Log = np.polyfit(np.log(hydro_water), Price_Actual, 1)
print(Log)

y = -13.40692102 * Loghydrovalues + 162.49392679

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly hydro outputs versus predicted average monthly prices of energy per MWH")
plt.title("R squared : 0.325")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(Loghydrovalues, Price_Actual, "o")
plt.plot(Loghydrovalues, y)
```

[-13.40692102 162.49392679]

Out[]: [<matplotlib.lines.Line2D at 0x7ff6099859d0>]



This is a moderate and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH. The blue dots represent the observations and the orange line is the model of best fit.

In []: Hydro_Log.summary() #OLS Logarithmic summary table

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.325			
Model:	OLS	Adj. R-squared:	0.310			
Method:	Least Squares	F-statistic:	22.11			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	2.37e-05			
Time:	05:24:36	Log-Likelihood:	-170.22			
No. Observations:	48	AIC:	344.4			
Df Residuals:	46	BIC:	348.2			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	73.9980	3.648	20.283	0.000	66.655	81.341
x1	-0.0062	0.001	-4.702	0.000	-0.009	-0.004
Omnibus:	3.427	Durbin-Watson:	0.386			
Prob(Omnibus):	0.180	Jarque-Bera (JB):	1.866			
Skew:	0.195	Prob(JB):	0.393			
Kurtosis:	2.117	Cond. No.	8.18e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.18e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceHydroLog = Hydro_Log.get_influence()
#Logarithmic OLS regression residuals
```

```
standardized_residualsHydroLog = influenceHydroLog.resid_studentized_internal
```

```
print(standardized_residualsHydroLog)
```

```
print(Hydro_Logpred) # OLS logarithmic predicted values
```

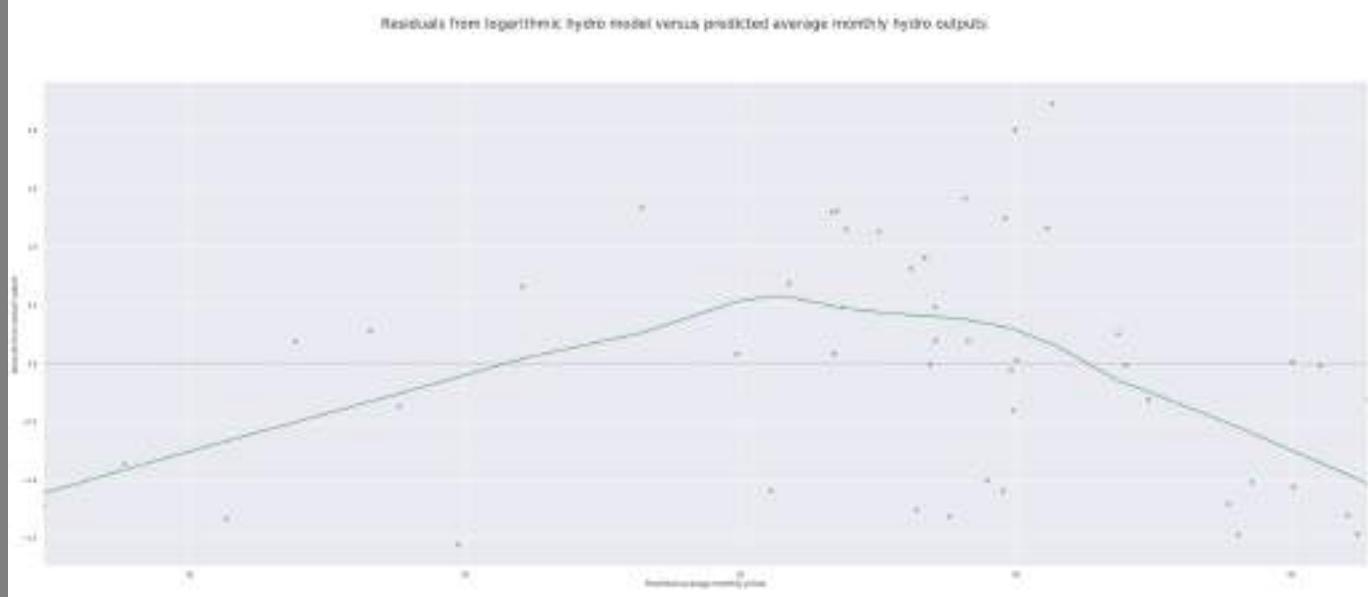
```
[ 0.80895074  0.6402248   0.06928352 -0.00953042  0.07129275  0.89991207
 1.4148346   0.25386521  0.02873628 -0.0598553   0.18886608 -0.00888328
-0.38848049 -0.88804896 -1.56974736 -1.24301769 -1.35769508 -1.09517813
-1.25912319 -1.31809949 -1.10269872  0.19470122  0.47807364  1.29131463
 2.22583673 -0.30411172 -1.00413115 -1.46905208 -1.20226272 -1.05202182
-1.29570143 -1.46042657 -1.01125531 -0.29792045 -0.0101847   0.01685708
-0.40522352  0.47556262  0.16731775  0.25918997  0.68065227  1.32333672
 1.30170574  1.24723159  1.99998134  1.15792516  1.12398831  1.14568746]
[ 58.08650622 51.03272344 54.9362862 58.43492496 56.6895171 58.34117714
 59.07505676 61.85526261 60.01136619 59.91386582 59.12537844 61.97671846
 48.78799584 43.80247892 49.86453955 42.33198956 45.65708831 55.54140987
 58.18326503 58.78127281 59.74968846 58.53094196 58.52594837 56.64551935
 60.65594134 62.40091809 59.47134868 64.03042477 63.85597988 65.03726418
 66.01249343 66.19987465 64.28345121 66.39414224 65.51544739 65.01058839
 59.94493528 56.8440357 46.91477046 48.26646143 55.86675409 53.19869015
 56.74491147 59.79565719 59.97125412 60.56167034 57.50777724 56.90700336]
```

In []:

```
# OLS Logarithmic average monthly predictions versus residuals
plt.suptitle("Residuals from logarithmic hydro model versus predicted average monthly hydro outputs")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("#")
sns.residplot(x = Hydro_Logpred, y = standardized_residualsHydroLog, lowess = True, color="g")
```

Out[]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff60949a650>
```



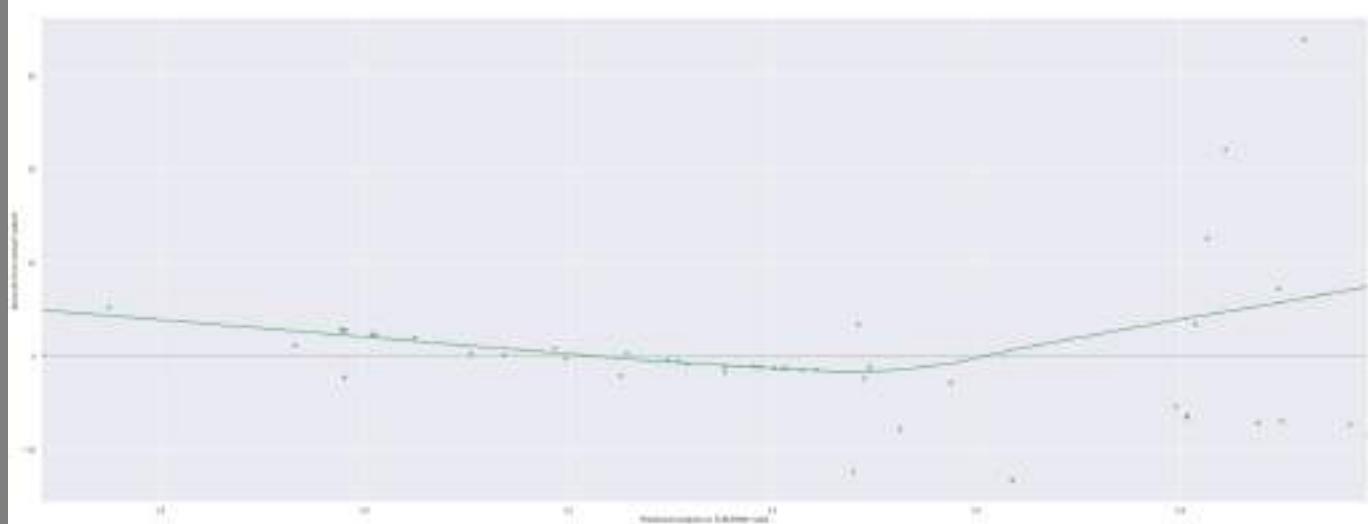
As one can observe this residual plot, one may notice that the lowess line has an arching hump; indicating heteroskedasticity, decreasing variance, and bias. This would indicate that this model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: HydroLogRatioPredict = Hydro_Logpred/ypred
```

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
```

```
plt.suptitle("Predicted average monthly hydro outputs to prices of energy per EUR/MWH ratio versus respective hydro outputs")
plt.xlabel("Predicted outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")
plt.legend("...#")
sns.residplot(x = HydroLogRatioPredict, y = standardized_residualsHydroLog/standardized_residualsPriceLog, lowess=True)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6094ffa50>
```



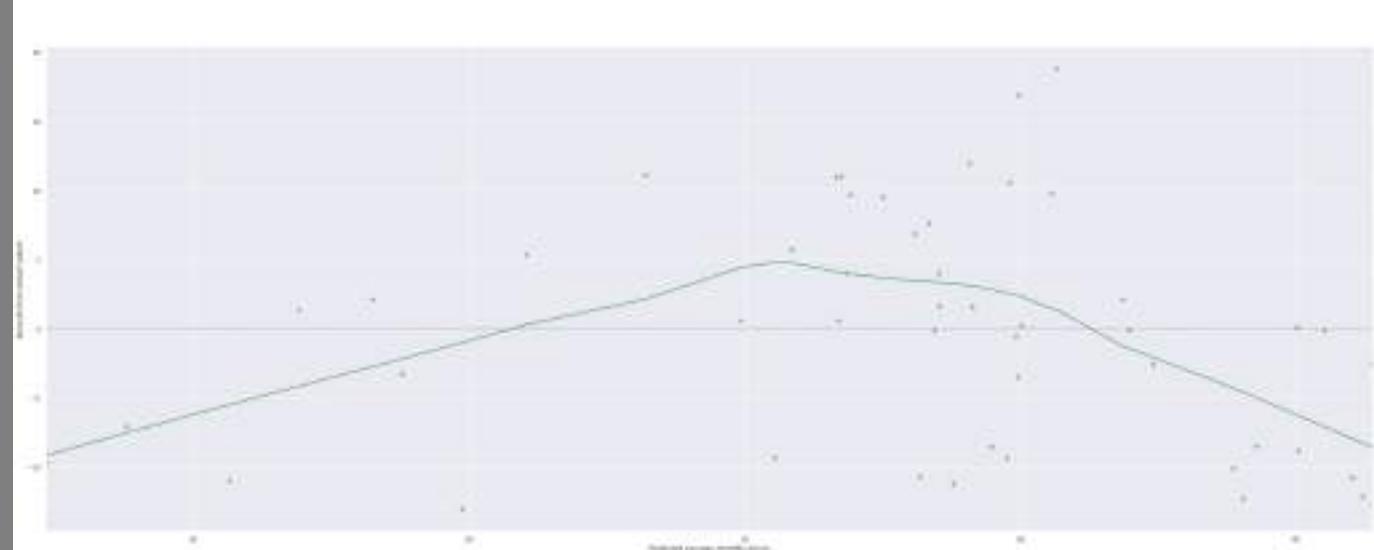
With the exception of few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS predicted logarithmic average monthly values versus actual values
```

```
plt.suptitle("Predicted average monthly logarithmic hydro energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
```

```
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = Hydro_Logpred, y = Price_Actual, lowess = True, color="g")
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608a3c190>



As one can observe this residual plot, one may notice that the lowess line has an arching hump and that the observations are spread out in a pattern; indicating heteroscedasticity and bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: influenceHydroLog = Hydro_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsHydroLog = influenceHydroLog.resid_studentized_internal

print(standardized_residualsHydroLog)
```

```
[ 0.80895074  0.6402248   0.06928352 -0.00953042  0.07129275  0.89991207
 1.4148346   0.25386521  0.02873628 -0.0598553   0.18886608 -0.00888328
-0.38848049 -0.88804896 -1.56974736 -1.24301769 -1.35769508 -1.09517813
-1.25912319 -1.31809949 -1.10269872  0.19470122  0.47807364  1.29131463
 2.22583673 -0.30411172 -1.00413115 -1.46905208 -1.20226272 -1.05202182
-1.29570143 -1.46042657 -1.01125531 -0.29792045 -0.0101847   0.01685708
-0.40522352  0.47556262  0.16731775  0.25918997  0.68065227  1.32333672
 1.30170574  1.24723159  1.99998134  1.15792516  1.12398831  1.14568746]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly hydro reservoir outputs versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)

modelHydroquad = np.poly1d(np.polyfit(hydro_water,Price_Actual,2))
print(modelHydroquad)

hydro1 = hydro_water
hydro1 = sm.add_constant(hydro1)
hydro2 = polynomial_features.fit_transform(hydro1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(hydro1)

Hydro_Q = poly.fit(X_poly, hydro_water)
lin2 = LinearRegression()
lin2.fit(X_poly, hydro_water)
Hydro_Quad = sm.OLS(Price_Actual, hydro2).fit()

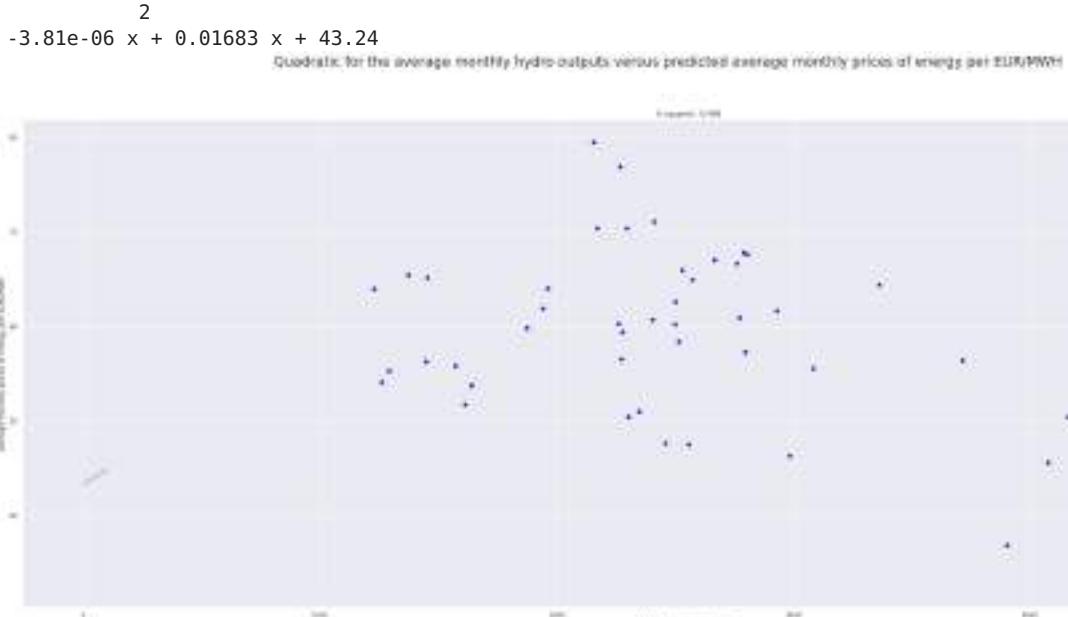
# OLS Predicted Quadratic values
Hydro_ypred = Hydro_Quad.predict(hydro2)
```

```
#OLS Quadratic Summary Table
```

```
Hydro_Quad.summary()
```

```
##OLS Quadratic Scatterplot
```

```
polyline = np.linspace(start = 0, stop = 100 , num = 10)
plt.title("R squared : 0.498")
plt.plot(polyline, modelHydroquad(polyline))
plt.scatter(hydro_water,Price_Actual, color = 'blue')
plt.suptitle('Quadratic for the average monthly hydro outputs versus predicted average monthly prices of energy')
plt.xlabel('Average monthly outputs ')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()
```



The blue dots represent the observations and the orange line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

```
In [ ]:
```

```
Hydro_Quad.summary()
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.498			
Model:	OLS	Adj. R-squared:	0.475			
Method:	Least Squares	F-statistic:	22.30			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	1.87e-07			
Time:	05:24:39	Log-Likelihood:	-163.11			
No. Observations:	48	AIC:	332.2			
Df Residuals:	45	BIC:	337.8			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	14.4138	2.811	5.128	0.000	8.752	20.075
x1	14.4138	2.811	5.128	0.000	8.752	20.075
x2	0.0084	0.003	2.826	0.007	0.002	0.014
x3	14.4138	2.811	5.128	0.000	8.752	20.075
x4	0.0084	0.003	2.826	0.007	0.002	0.014
x5	-3.81e-06	9.67e-07	-3.938	0.000	-5.76e-06	-1.86e-06
Omnibus:	0.081	Durbin-Watson:		0.821		
Prob(Omnibus):	0.960	Jarque-Bera (JB):		0.116		
Skew:	-0.082	Prob(JB):		0.944		
Kurtosis:	2.823	Cond. No.		4.41e+23		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 2.29e-32. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []: `print(Hydro_ypred) # OLS quadratic predicted values`

```
[61.33760765 53.23032422 58.94131093 61.48126682 60.51882213 61.44499016
 61.68219643 61.60803683 61.82912405 61.82196732 61.69453226 61.56970637
 48.57205984 34.63696257 50.93132503 29.58151062 40.39897314 59.55496108
 61.37992756 61.60011257 61.80563869 61.51660734 61.51481463 60.48672199
 61.82881071 61.41278538 61.76569175 60.47678139 60.60225776 59.63414779
 58.62550365 58.41001068 60.28401251 58.17921592 59.16324593 59.65907643
 61.82445339 60.62850248 43.91667166 47.34612102 59.85474759 56.77391286
 60.55868887 61.81075166 61.82640895 61.83402199 61.04556169 60.6718346 ]
```

In []: `influenceHydroQuad = Hydro_Quad.get_influence() #Quadratic OLS residuals`

```
standardized_residualsHydroQuad = influenceHydroQuad.resid_studentized_internal
print(standardized_residualsHydroQuad)

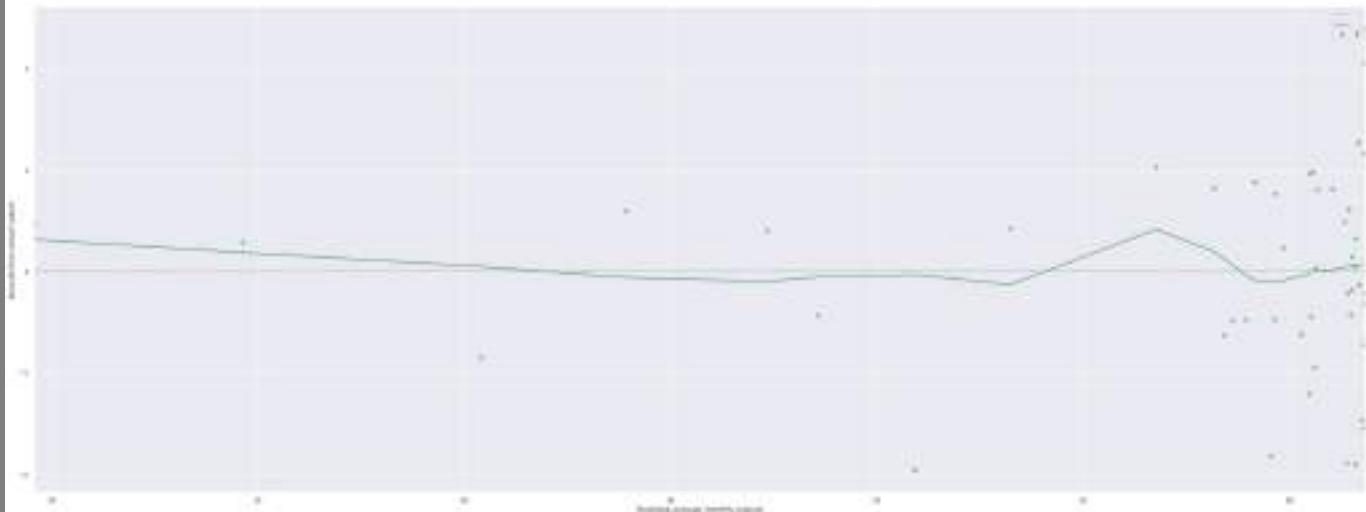
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = Hydro_ypred, y = standardized_residualsHydroQuad, lowess = True, color="g")

plt.suptitle("Hydro power residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend("..#")
```

```
[ 0.49132642  0.4340034  -0.46792924 -0.42515881 -0.43998477  0.61597838
 1.27518268  0.32476721 -0.21357131 -0.32766525 -0.13141329  0.0451379
 -0.41558803  0.32444334 -1.9489456   0.50662324 -0.82546889 -1.81700338
 -1.88770316 -1.90188025 -1.54683999 -0.18135659  0.14493691  0.96995332
 2.39584587 -0.21445043 -1.46665945 -1.20783397 -0.93996954 -0.4724592
 -0.48230315 -0.62173014 -0.61797779  0.83062659  0.88551316  0.76826288
 -0.72071535  0.03390765  0.61695032  0.42568353  0.24311166  1.03903337
 0.98563081  1.16050451  2.04681924  1.15674115  0.81685611  0.81168527]
```

Out[]: `<matplotlib.legend.Legend at 0x7ff608c90f50>`



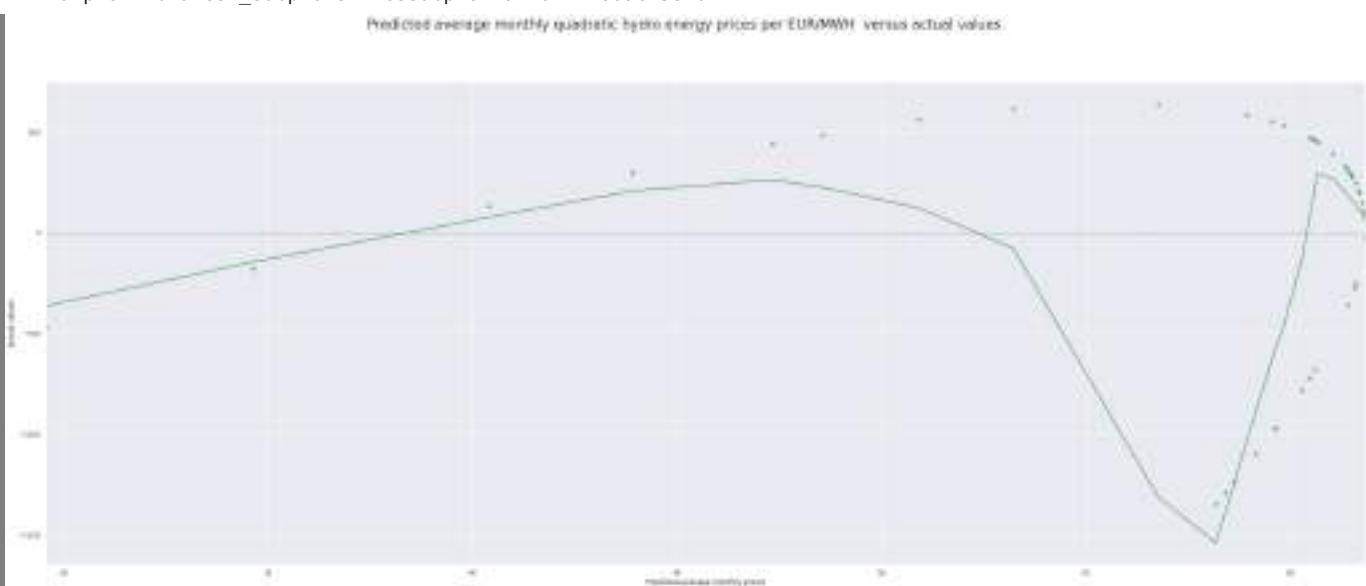
As one can observe this residual plot, one may notice that the lowess line has a slight hump and that the residuals are clustered on the right side; indicating an increasing trend in variance, heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []: `print(standardized_residualsHydroQuad)`

```
[ 0.49132642  0.4340034 -0.46792924 -0.42515881 -0.43998477  0.61597838
 1.27518268  0.32476721 -0.21357131 -0.32766525 -0.13141329  0.0451379
-0.41558803  0.32444334 -1.9489456  0.50662324 -0.82546889 -1.81700338
-1.88770316 -1.90188025 -1.54683999 -0.18135659  0.14493691  0.96995332
 2.39584587 -0.21445043 -1.46665945 -1.20783397 -0.93996954 -0.4724592
-0.48230315 -0.62173014 -0.61797779  0.83062659  0.88551316  0.76826288
-0.72071535  0.03390765  0.61695032  0.42568353  0.24311166  1.03903337
 0.98563081  1.16050451  2.04681924  1.15674115  0.81685611  0.81168527]
```

In []: `plt.suptitle("Predicted average monthly quadratic hydro energy prices per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Hydro_ypred, y = hydro_water, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values`

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60ac45810>



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

In []:

```
#Dataframes analyzed by resource
dfhydro = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.62351388888889], 'Hydro': [2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.184139784946, 2531.1682892906815, 2412.5255376344085, 1963.0631720430108, 2261.15694444444444, 2276.09193548387098, 2404.3902777777776, 1943.42799461642, 4075.5846774193546, 4881.568965517241, 3901.5450874831763, 5119.295833333334, 4581.743279569892, 2983.793055555554, 2556.6971736204578, 2460.0201612903224, 2303.4611111111, 2500.489932885906, 2501.297222222224, 2805.2970430107525, 2156.951612903226, 1874.8497023809523, 2348.4589502018844, 1611.4152777777779, 1639.616935483871, 1448.64444444444444, 1290.983870967742, 1260.6908602150538, 1570.5097222222223, 1229.2845637583894, 1371.338888888889, 1452.9569892473119, 2271.896505376344, 2773.2038690476193, 4378.419919246298, 4159.898611111111, 2931.19623655914, 3362.5291666666667, 2789.228802153432, 2296.0295698924733, 2267.641666666667, 2172.1919463087247, 2763.0241935483873, 2665.9], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

print(dfhydro)
df_hydro= pd.DataFrame.from_dict(dfhydro, orient = "columns")
print(df_hydro)
df_hydro["Ratio"] = df_hydro["Hydro"]/df_hydro["Price"]
pdToListHydro = list(df_hydro["Ratio"])

print(pdToListHydro)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

test_result=adfuller(df_hydro["Ratio"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_hydro["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Hydro Ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_hydro["Ratio"])
plt.suptitle("Partialatocorrelations of Hydro Ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show

df_hydro['First Difference Ratio'] = df_hydro["Ratio"]- df_hydro["Ratio"].shift(1) # Seasonality values
df_hydro['Seasonal Difference Ratio']=df_hydro["Ratio"]- df_hydro["Ratio"].shift(12)
df_hydro.head()

Hydro_Ratio_Autocorrelations = sm.tsa.acf(df_hydro["Ratio"],fft=False) #Autocorrelations
print(Hydro_Ratio_Autocorrelations)
```

	Price	Hydro	Dates
0	64.949019	2572.339700	2015-01
1	56.383854	3712.690476	2015-02
2	55.522463	3081.620458	2015-03
3	58.354083	2516.012535	2015-04
4	57.294059	2798.184140	2015-05
5	65.974903	2531.168289	2015-06
6	71.072043	2412.525538	2015-07
7	63.998065	1963.063172	2015-08
8	60.254792	2261.156944	2015-09
9	59.406765	2276.919355	2015-10

```

10 60.726792 2404.390278 2015-11
11 61.901761 1943.427995 2015-12
12 45.578723 4075.584677 2016-01
13 36.752083 4881.568966 2016-02
14 36.818008 3901.545087 2016-03
15 32.618667 5119.295833 2016-04
16 34.691371 4581.743280 2016-05
17 46.266319 2983.793056 2016-06
18 47.502016 2556.697174 2016-07
19 47.602339 2460.020161 2016-08
20 50.405597 2303.461111 2016-09
21 60.182430 2500.489933 2016-10
22 62.581056 2501.297222 2016-11
23 67.595134 2805.297043 2016-12
24 79.492083 2156.951613 2017-01
25 59.837798 1874.849702 2017-02
26 50.959892 2348.458950 2017-03
27 51.717917 1611.415278 2017-04
28 53.772621 1639.616935 2017-05
29 56.258222 1448.644444 2017-06
30 55.252581 1290.983871 2017-07
31 54.084328 1260.690860 2017-08
32 55.816556 1570.509722 2017-09
33 63.925289 1229.284564 2017-10
34 65.430653 1371.338889 2017-11
35 65.151277 1452.956989 2017-12
36 56.511976 2271.896505 2018-01
37 60.877098 2773.203869 2018-02
38 48.279717 4378.419919 2018-03
39 50.400736 4159.898611 2018-04
40 61.633763 2931.196237 2018-05
41 64.348139 3362.529167 2018-06
42 67.783441 2789.228802 2018-07
43 70.363911 2296.029570 2018-08
44 76.914042 2267.641667 2018-09
45 70.362215 2172.191946 2018-10
46 67.042608 2763.024194 2018-11
47 66.623514 2665.900000 2018-12

```

```

[39.60552055610406, 65.8466954957713, 55.50222903974214, 43.1163063679167, 48.83899276464233, 38.365623634435295
, 33.94478947607304, 30.67378969794114, 37.52659136145235, 38.32761051675219, 39.59356672382155, 31.395358887784
525, 89.41857951665217, 132.8242788644791, 105.96839132351556, 156.94374897809027, 132.07155415766834, 64.491688
36821842, 53.82291915095909, 51.67855672583178, 45.6985183799308, 41.548504312726116, 39.96892030690864, 41.5014
6408546457, 27.13416886884827, 31.332197657357437, 46.08445667561115, 31.157776291796583, 30.4916685475955, 25.7
49915074142248, 23.36513255839751, 23.309725900960125, 28.13698743303958, 19.230019775602145, 20.95866127984951,
22.30128185952668, 40.20203634637343, 45.55414023326174, 90.68859882531778, 82.53646537900543, 47.55828742101278
, 52.25526681467521, 41.149117937306535, 32.63078370414942, 29.48280466776494, 30.871568690104986, 41.2129583778
5713, 40.014400988306384]
[ 1.          0.76546089  0.52829648  0.35337438  0.07193558 -0.0955247
-0.14636035 -0.16906723 -0.19832371 -0.20060354 -0.13586914 -0.07872197
-0.08172419 -0.086693 -0.10571597 -0.15896144 -0.18047796 -0.18395757
-0.18618104 -0.16771139 -0.12988296 -0.06521704  0.06150582  0.18108682
0.18443088  0.17499047  0.12626167 -0.01026066 -0.08910114 -0.12437026
-0.14237346 -0.13869013 -0.11470274 -0.0602108  -0.02136209  0.00695068
0.02680938  0.04745409  0.0266263  0.01108395  0.01432111]

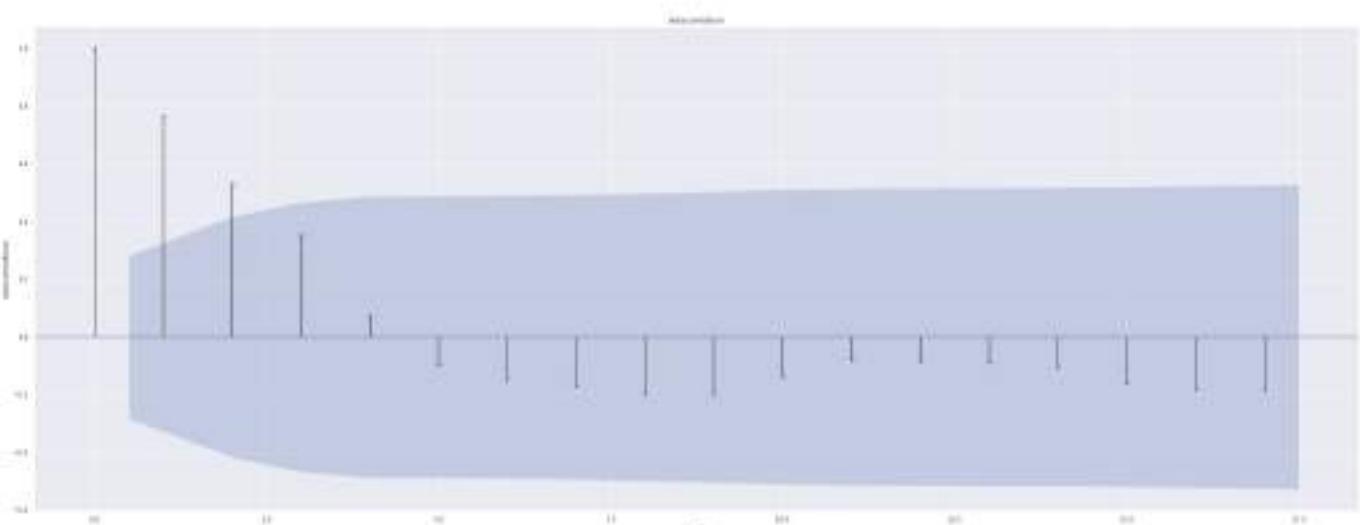
```

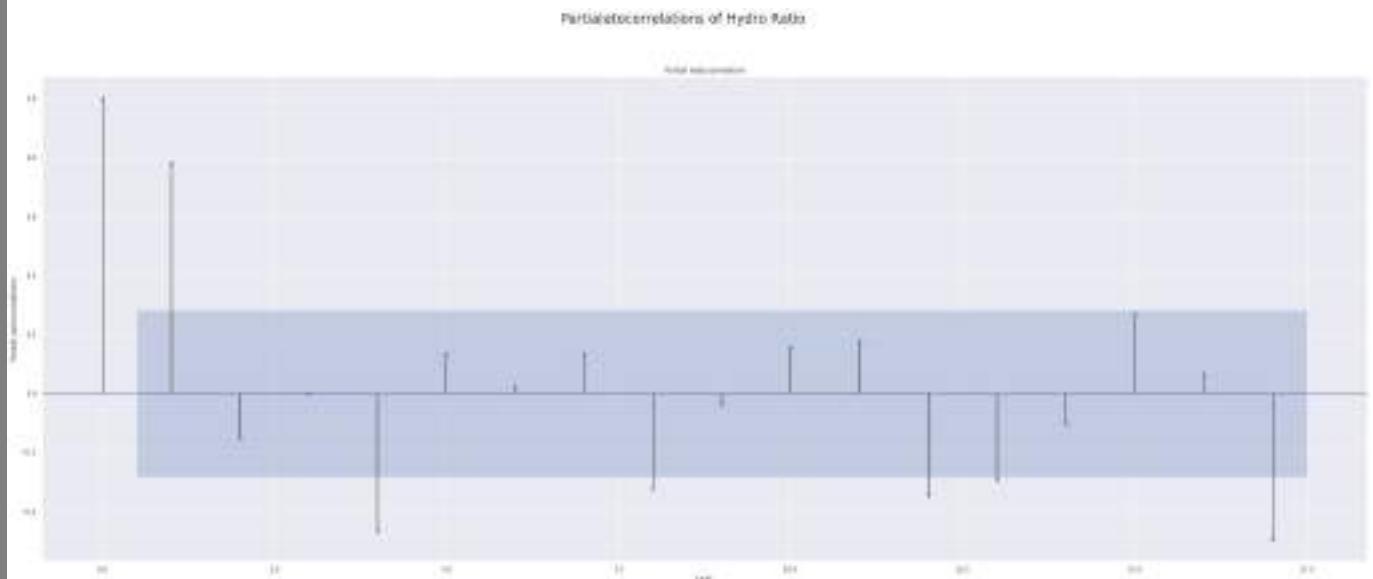
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```

Autocorrelations of average monthly Hydro Ratio:





The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation and partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly hydro reservoir outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: df_hydro.describe(include = 'all') # Description Tables
```

	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN
mean	57.859848	2608.982390	NaN	49.210055	0.008700	1.983871
std	10.320573	950.659135	NaN	30.241054	20.843801	49.826097
min	32.618667	1229.284564	NaN	19.230020	-67.579866	-125.785973
25%	51.528411	2108.479503	NaN	31.086224	-8.712651	-19.979758
50%	59.622281	2480.255047	NaN	40.108219	-1.579584	10.873827
75%	64.999583	2836.771841	NaN	51.822734	4.762120	22.285091
max	79.492083	5119.295833	NaN	156.943749	58.023221	113.827443

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly hydro output to price of energy per EUR/MWH ratio")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_hydro['Seasonal Difference Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60aca9250>
```

Seasonal Difference of average monthly hydro output to price of energy per EUR/MWH ratio.



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Bell Curves

hydroResults_mean = np.mean(df_hydro["Ratio"])
hydroResults_std = np.std(df_hydro["Ratio"])

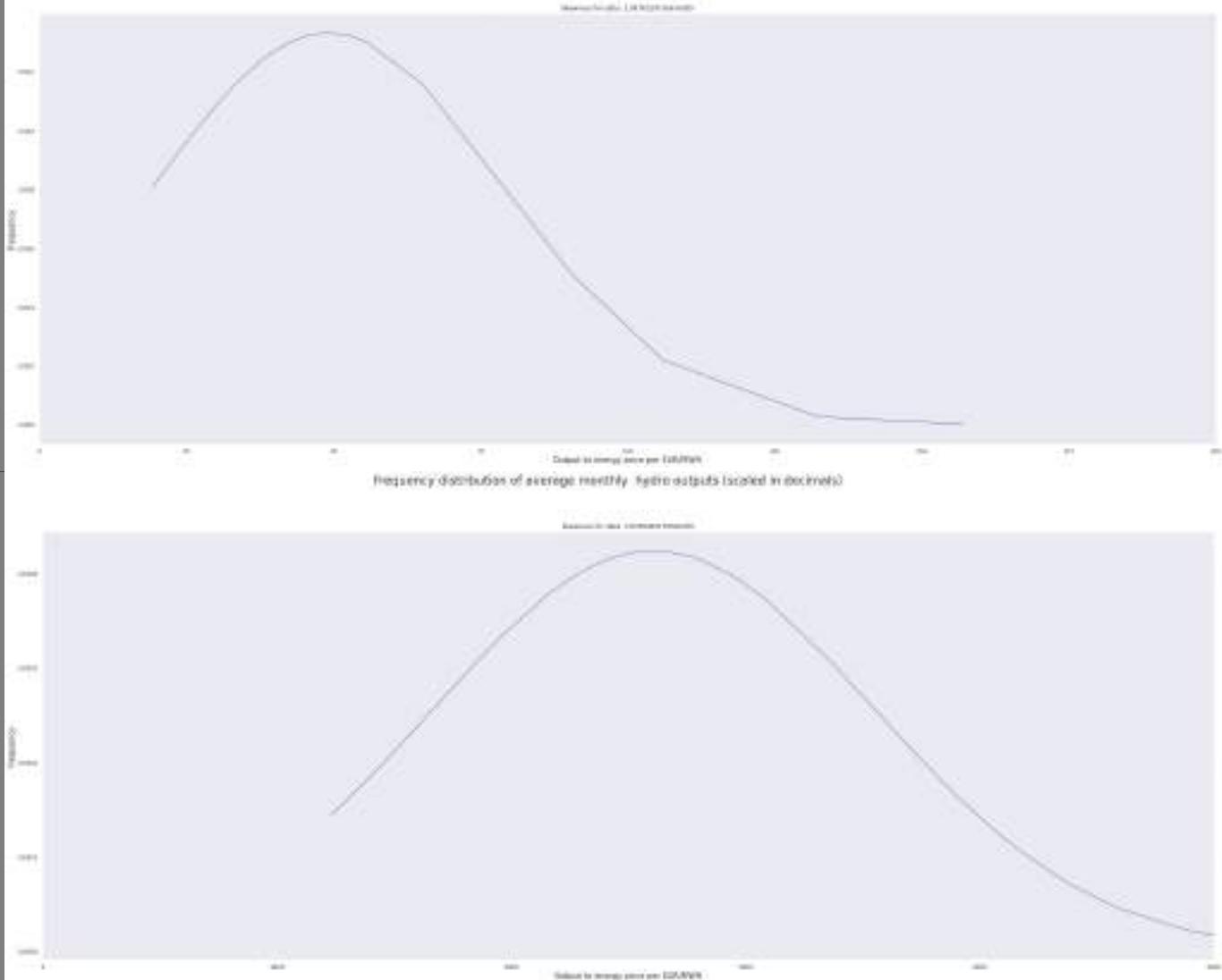
hydroResultspdf = stats.norm.pdf(df_hydro["Ratio"].sort_values(), hydroResults_mean, hydroResults_std)

plt.plot(df_hydro["Ratio"].sort_values(), hydroResultspdf)
plt.xlim([0,200])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_hydro["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
plt.suptitle("Frequency distribution of average monthly hydro outputs to energy prices per EUR/MWH ratios (scaled)", size=10)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves
hydroResults_mean = np.mean(df_hydro[ "Hydro"])
hydroResults_std = np.std(df_hydro[ "Hydro"])

hydroResultspdf = stats.norm.pdf(df_hydro["Hydro"].sort_values(), hydroResults_mean, hydroResults_std)

plt.plot(df_hydro["Hydro"].sort_values(), hydroResultspdf)
plt.xlim([0,5000])
plt.xlabel("Value ", size=15)
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_hydro["Hydro"])}')
plt.suptitle("Frequency distribution of average monthly hydro outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



These bell shaped curves are both skewed to the right. Hence, they have a asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective the average monthly prices of energy per EUR/MWH.

In []:

In []:

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_hydro['Hydro'])

test_result=adfuller(df_hydro['Hydro'])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_hydro["Hydro"])
plt.suptitle(" Autocorrelations of average monthly Hydro")
plt.ylabel('Autocorrealtions')
```

```
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_hydro["Hydro"])
plt.suptitle("Partialatocorrelations of Hydro")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show
Hydro_Autocorrelations = sm.tsa.acf(df_hydro["Hydro"], fft=False) #Autocorrelations
print(Hydro_Autocorrelations)
df_hydro['First Difference'] = df_hydro["Hydro"]- df_hydro["Hydro"].shift(1) # Seasonality values
df_hydro['Seasonal Difference']=df_hydro["Hydro"]- df_hydro["Hydro"].shift(12)
df_hydro.head()
```

```

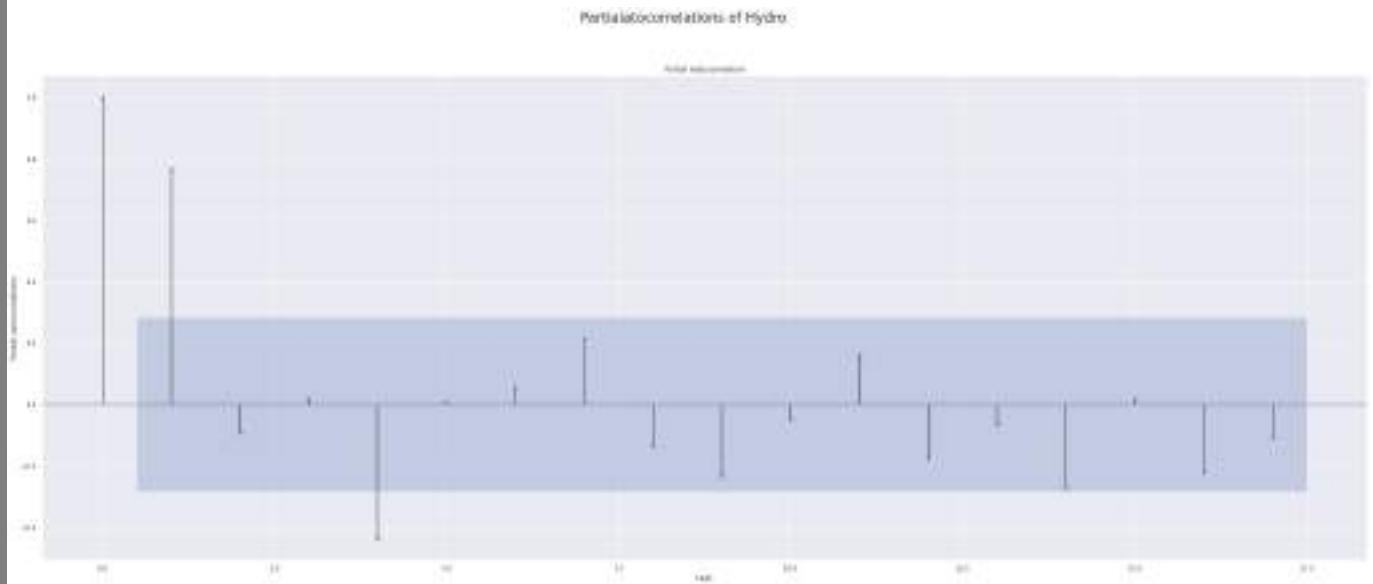
ADF Test Statistic : -3.1673141085291867
p-value : 0.021956352091091067
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
[ 1. ] 0.74939834 0.52473279 0.36956636 0.09716137 -0.0835217
-0.1434708 -0.14428342 -0.14292616 -0.15713115 -0.13020363 -0.09117332
-0.1374441 -0.14945181 -0.15462303 -0.19054601 -0.23209519 -0.25967188
-0.26366714 -0.24411201 -0.17871772 -0.09417803 0.05593604 0.17769478
0.21695225 0.24333289 0.21138179 0.05674265 -0.03283555 -0.08010896
-0.12580332 -0.12294714 -0.11136326 -0.06350859 -0.02316694 -0.00282879
0.02940809 0.07415011 0.04859925 0.01346777 0.01969459]

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning
```

FutureWarning

Dat[]:	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	First Difference	Seasonal Difference
0	64.949019	2572.339700	2015-01	39.605521	NaN	NaN	NaN	NaN
1	56.383854	3712.690476	2015-02	65.846695	26.241175	NaN	1140.350776	NaN
2	55.522463	3081.620458	2015-03	55.502229	-10.344466	NaN	-631.070019	NaN
3	58.354083	2516.012535	2015-04	43.116306	-12.385923	NaN	-565.607923	NaN
4	57.294059	2798.184140	2015-05	48.838993	5.722686	NaN	282.171605	NaN



In []:

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation and partial autocorrelation plot, indicating that the lags directly beforehand influenced the average monthly hydro reservoir outputs.

In []:

```
df_hydro['First Difference'] = df_hydro["Ratio"] - df_hydro["Ratio"].shift(1) # Seasonality values
df_hydro['Seasonal Difference']=df_hydro["Ratio"]- df_hydro["Ratio"].shift(12)
df_hydro.head()
```

Out[]:

	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	First Difference	Seasonal Difference
0	64.949019	2572.339700	2015-01	39.605521	NaN	NaN	NaN	NaN
1	56.383854	3712.690476	2015-02	65.846695	26.241175	NaN	26.241175	NaN
2	55.522463	3081.620458	2015-03	55.502229	-10.344466	NaN	-10.344466	NaN
3	58.354083	2516.012535	2015-04	43.116306	-12.385923	NaN	-12.385923	NaN
4	57.294059	2798.184140	2015-05	48.838993	5.722686	NaN	5.722686	NaN

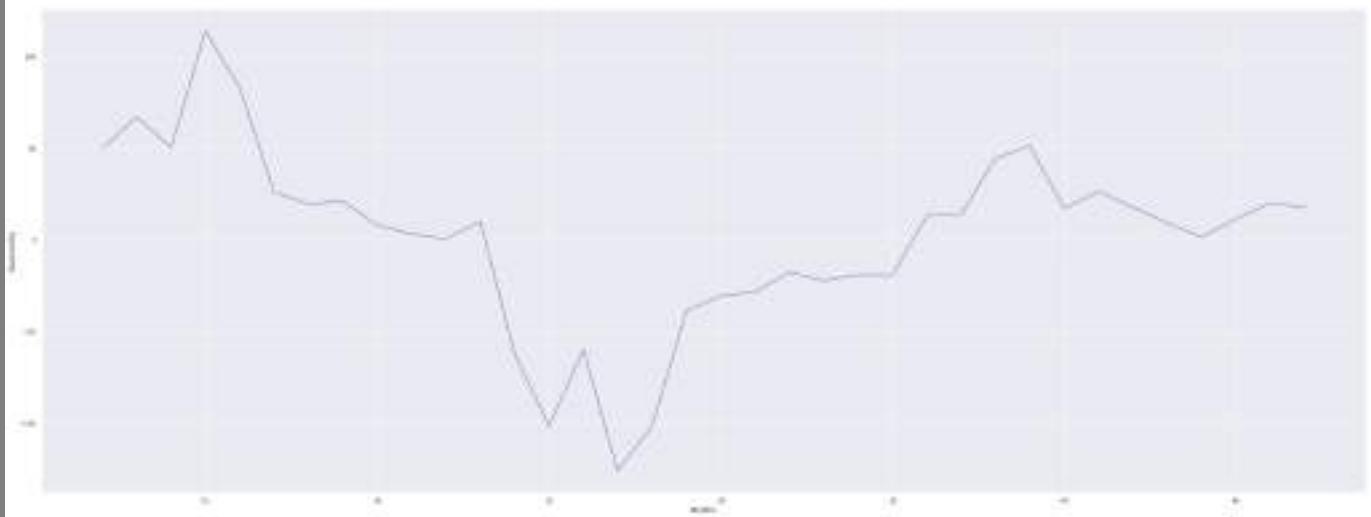
In []:

```
plt.suptitle("Seasonal Difference of average monthly Hydro output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_hydro['Seasonal Difference'].plot() # Seasonality Plot
```

Out[]:

<matplotlib.axes._subplots.AxesSubplot at 0x7ff608c9ec90>

Seasonal Difference of average monthly Hydro output to price of energy per EUR/MWH



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted in the average monthly hydro reservoir outputs.

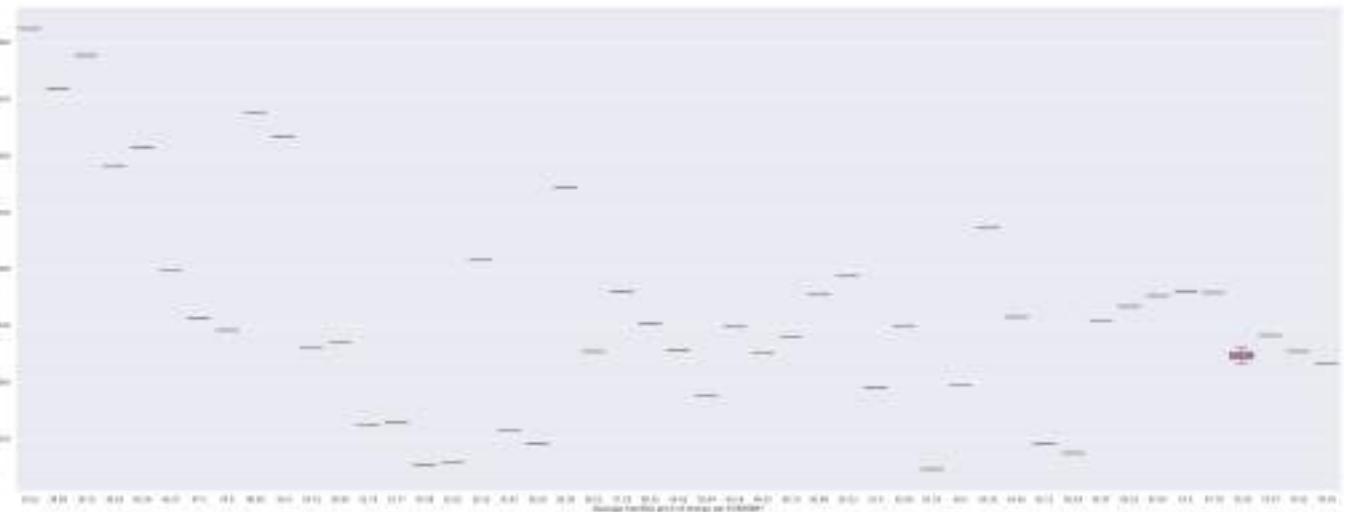
In []: `df_hydro.describe(include = 'all') # Description Tables`

	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	2608.982390	NaN	49.210055	0.008700	1.983871	0.008700	1.983871
std	10.320573	950.659135	NaN	30.241054	20.843801	49.826097	20.843801	49.826097
min	32.618667	1229.284564	NaN	19.230020	-67.579866	-125.785973	-67.579866	-125.785973
25%	51.528411	2108.479503	NaN	31.086224	-8.712651	-19.979758	-8.712651	-19.979758
50%	59.622281	2480.255047	NaN	40.108219	-1.579584	10.873827	-1.579584	10.873827
75%	64.999583	2836.771841	NaN	51.822734	4.762120	22.285091	4.762120	22.285091
max	79.492083	5119.295833	NaN	156.943749	58.023221	113.827443	58.023221	113.827443

Below is the box and whisker plot for the distribution of the average monthly outputs of hydro power and its the average monthly prices of energy per EUR/MWH.

In []: `# Box and Whisker Plot
sns.set(style="darkgrid")

plt.suptitle('Distribution of average hydro power outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly hydro power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')
sns.boxplot(x=Rounded_Y, y=[2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.])
plt.show()`



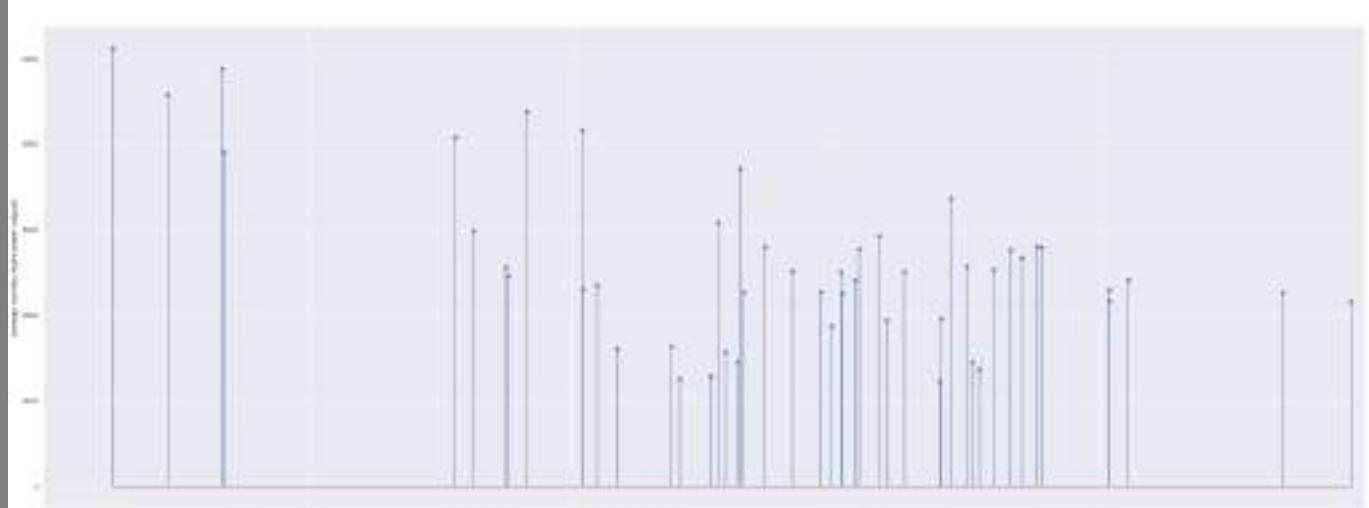
This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil brown coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of hydro power produced, which was in between 2000 and 2500 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest (at approximately 32.62 euros per MWH), hydro power output was at its highest, at slightly above 5000 units. At approximately 60.73 EUR/MWH, hydropower produced the most units at roughly 700 units. The lowest amount produced, which was slightly above 1000 units, was at the price of approximately 63.93 EUR/MWH.

```
In [ ]: plt.xlabel('Average monthly price of energy per EUR/MWH')
plt.xlim(30, 80)
plt.suptitle('Distribution of average hydro power outputs by average monthly price of energy per EUR/MWH ')
plt.ylabel('Average monthly hydro power outputs')

# Stem Plot
plt.stem(Rounded_Y, hydro_water)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.
  if __name__ == '__main__':
```

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of fossil brown coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of hydro power produced, which was in between 2000 and 2500 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest (at approximately 32.62 euros per MWH), hydro power output was at its highest, at slightly above 5000 units. At approximately 60.73 EUR/MWH, hydropower produced the most units at roughly 700 units. The lowest amount produced, which was slightly above 1000 units, was at the price of approximately 63.93 EUR/MWH. Each blue dot at the end of the line respents an observation.

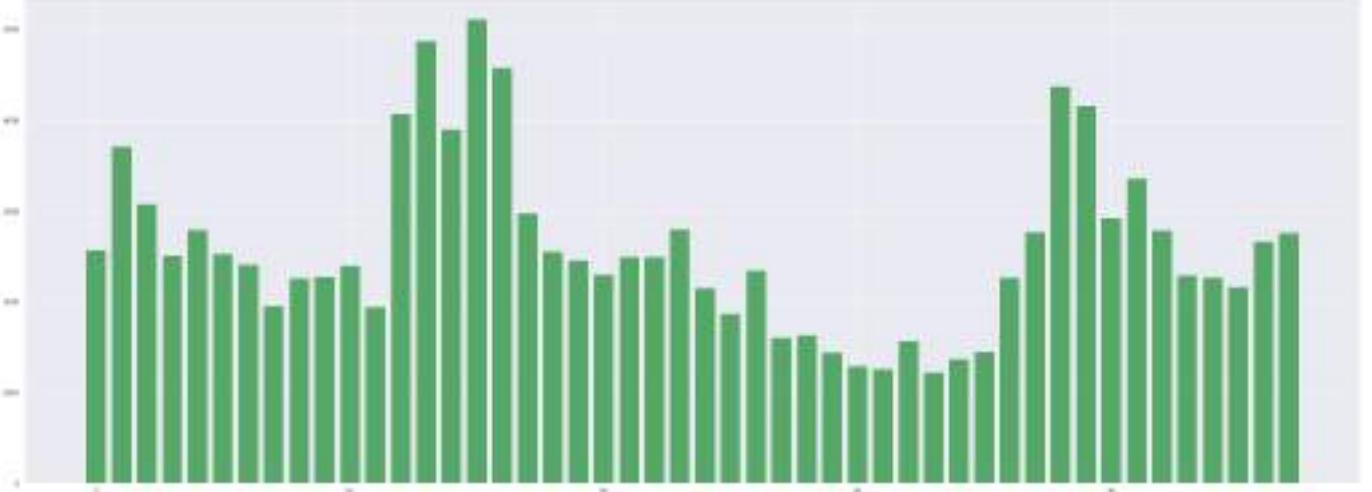
In []:

```
#Outputs in MWH Histogram
hydro_water_Dict = {key: i for i, key in enumerate(hydro_water)}

def Hist_hydro_water(hydro_water_Dict):
    for k, v in hydro_water_Dict.items(): print(f"{v}:{k}")
print(hydro_water_Dict)

plt.bar(list(hydro_water_Dict.values()), hydro_water_Dict.keys(), color='g')
print(dicDates)
plt.show()
```

```
{2572.3396998635744: 0, 3712.690476190476: 1, 3081.620457604307: 2, 2516.0125348189417: 3, 2798.184139784946: 4, 2531.1682892906815: 5, 2412.5255376344085: 6, 1963.0631720430108: 7, 2261.1569444444444: 8, 2276.9193548387098: 9, 2404.3902777777776: 10, 1943.42799461642: 11, 4075.5846774193546: 12, 4881.568965517241: 13, 3901.5450874831763: 14, 5119.295833333334: 15, 4581.743279569892: 16, 2983.7930555555554: 17, 2556.6971736204578: 18, 2460.0201612903224: 19, 2303.461111111111: 20, 2500.489932885906: 21, 2501.297222222224: 22, 2805.2970430107525: 23, 2156.951612903226: 24, 1874.8497023809523: 25, 2348.4589502018844: 26, 1611.4152777777779: 27, 1639.616935483871: 28, 1448.6444444444444: 29, 1290.983870967742: 30, 1260.6908602150538: 31, 1570.5097222222223: 32, 1229.2845637583894: 33, 1371.3388888888889: 34, 1452.9569892473119: 35, 2271.896505376344: 36, 2773.2038690476193: 37, 4378.419919246298: 38, 4159.898611111111: 39, 2931.19623655914: 40, 3362.5291666666667: 41, 2789.228802153432: 42, 2296.0295698924733: 43, 2267.6416666666667: 44, 2172.1919463087247: 45, 2665.9: 46, 2763.0241935483873: 47}
```



In []:

In []:

The green bars represent the observation value for each respective month. This histogram is multimodal, containing fluctuating patterns of gradual decrease in the average monthly outputs followed by a sudden increase in the average monthly outputs. This histogram is multimodal; there isn't an indication of a singular external factor increasing the average monthly outputs.

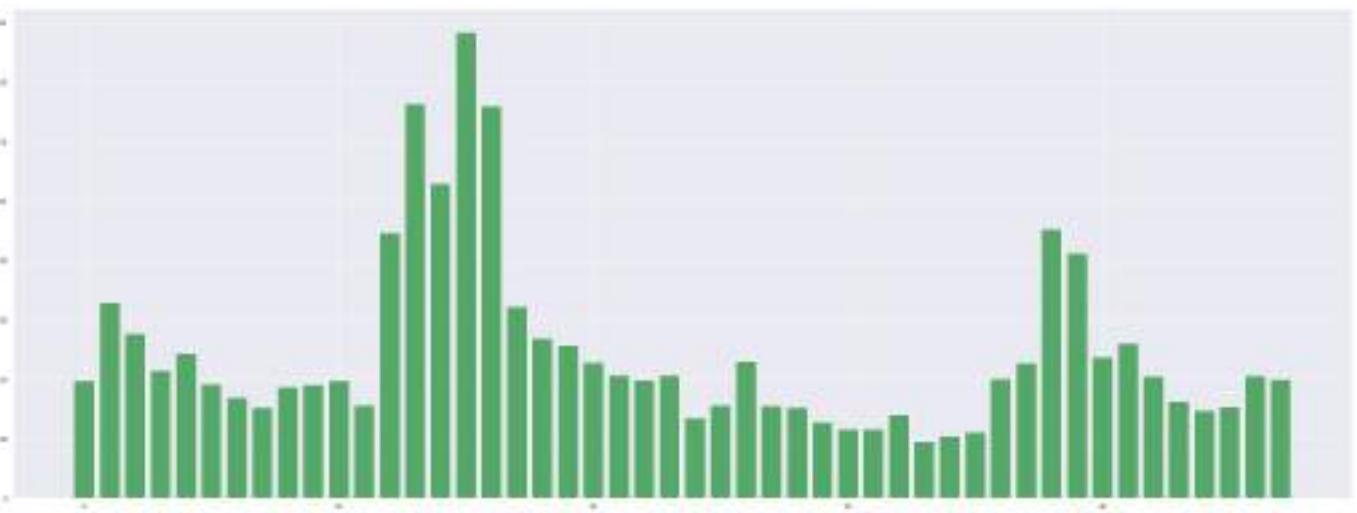
In []:

```
pdToListHydro_Dict = {key: i for i, key in enumerate(pdToListHydro)}

def Hist_pdToListHydro_water(pdToListHydro_Dict):
    for k, v in pdToListHydro_Dict.items(): print(f"{v}:{k}")
#Output/price per EUR/MWH Histogram
    print(pdToListHydro_Dict)

plt.bar(list(pdToListHydro_Dict.values()), pdToListHydro_Dict.keys(), color='g')
print(dicDates)
plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is bimodal in its distribution, with sharp increases in the output produced per EUR/MWH that always followed by gradual declines.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of hydro power.

In []:

```
In [ ]: modelhydrowater = stats.linregress([2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.012534818941, 64.9490188172043, 56.383854166666666, 55.522462987886975, 58.35408333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.752083333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.40559722222224, 60.182429530201404, 62.58105555555558, 67.5951344086021, 79.49208333333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.25822222222224, 55.252580645161316, 54.08432795698931, 55.81655555555558, 63.92528859060403, 65.43065277777781, 65.15127688172035, 56.51197580645163, 60.877098214285674, 48.279717362045766, 50.40073611111113, 61.633763440860214, 64.34813888888884,
```

```
67.78344086021498,  
70.36391129032262,  
76.9140416666666,  
70.36221476510062,  
67.0426075268817,  
66.62351388888881])
```

In []:

```
#slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(hydro_water,Price_Actual)  
print("slope: %f      intercept: %f" % (slope, intercept))  
  
#OLS Linear Scatterplot  
plt.plot(hydro_water,Price_Actual,"o")  
f = lambda x: -0.006186 *x + 73.998036  
plt.plot(x,f(x), c="orange", label="line of best fit")  
plt.title(f"R squared: {modelhydrowater.rvalue**2}")  
plt.legend("#")  
plt.suptitle("Average hydro monthly output versus predicted linear average monthly prices of energy per EUR/MWH")  
plt.ylabel('Average price of energy per EUR/MWH for each month ')  
plt.xlabel('Average monthly hydro output')  
plt.show()
```

```
slope: -0.006186      intercept: 73.998036
```



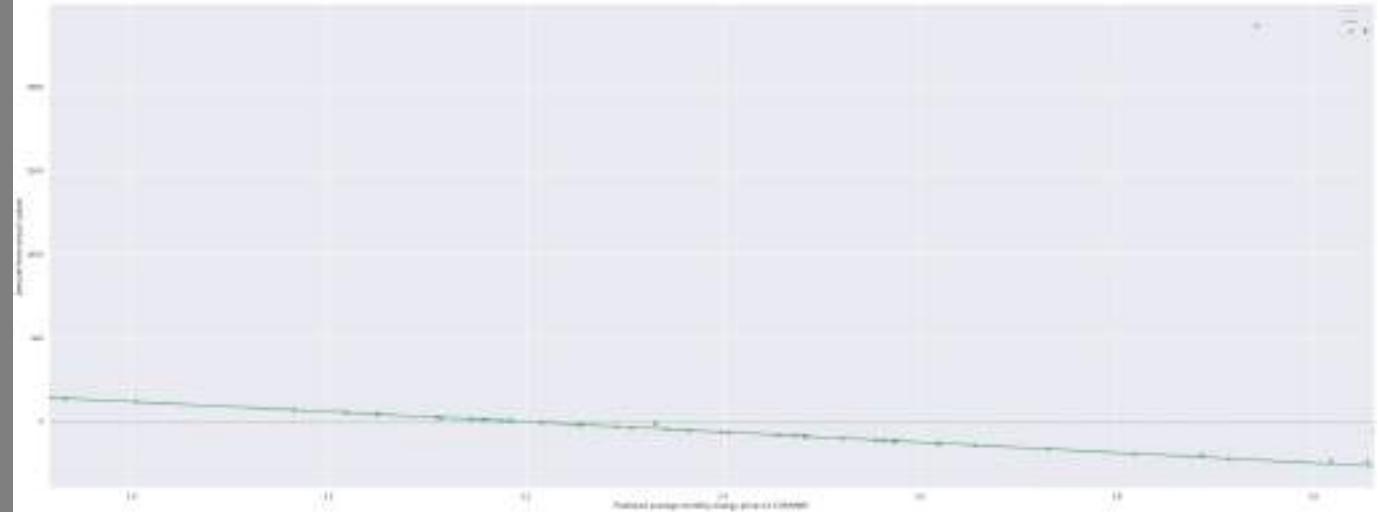
The blue dots represent the observations and the orange line is the model of best fit. There is a moderately negative correlation between the output and the respective the average monthly prices of energy per EUR/MWH. As the price decreases, the output increases. The blue dots are the observations and orange line is the model of best fit.

In []:

```
#OLS predicted quadratic average monthly ratios versus residuals  
HydroQuadRatioPredict = Hydro_ypred/ypred  
  
sns.residplot(x = HydroQuadRatioPredict , y = standardized_residualsHydroQuad/standardized_residualsPriceQuad ,  
plt.suptitle("Predicted average monthly quadratic hydro energy prices to EUR/MWH versus respective quadratic model")  
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")  
plt.ylabel("Amount from actual values")  
plt.legend("..#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff609f06ed0>
```

```
HydroRegRatioPredict = predictionsHydropower/predictions
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals
```

```
HydroRegRatioPredict = predictionsHydropower/predictions
```

```
sns.residplot(x = HydroRegRatioPredict, y = standardized_residualsHydro/standardized_residualsPricereg, lowess = True)
plt.suptitle("Predicted linear hydro power output to EUR/MWH versus respective linear model residuals ")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#..)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60991bad0>
```

```
HydroRegRatioPredict = predictionsHydropower/predictions
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

observation itself.

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(Hydro_ypred))
print(list(ypred))

[61.3376076524368, 53.23032421639376, 58.94131093358947, 61.481266818688226, 60.51882212939074, 61.4449901574589
3, 61.68219642784088, 61.60803683034199, 61.82912405339957, 61.82196732038972, 61.694532263137944, 61.5697063665
14396, 48.572059842755415, 34.63696257388266, 50.93132503376881, 29.581510622142275, 40.39897314063682, 59.55496
108363039, 61.379927561757256, 61.600112569182286, 61.80563868511031, 61.516607337639925, 61.51481462879916, 60.
48672198576486, 61.82881070824977, 61.41278538169082, 61.76569175276494, 60.47678138618451, 60.6022577646758, 59
.634147791875286, 58.62550365257845, 58.410010679349355, 60.2840125073846, 58.17921591815906, 59.1632459278975,
59.65907642754619, 61.824453393966806, 60.628502478567874, 43.916671656036094, 47.346121018397795, 59.8547475864
4083, 56.77391285910282, 60.55868887159784, 61.81075166227316, 61.82640894900889, 61.834021985154315, 61.0455616
9294959, 60.67183459826377]
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338
71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672
299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486
, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506
2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20
1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,
27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905
53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857
92181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: print(Hydro_Logpred)
print(Hydro_Logpred/Logpred)

print(Logpred)

Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price
```

```
[58.08650622 51.03272344 54.9362862 58.43492496 56.6895171 58.34117714
59.07505676 61.85526261 60.01136619 59.91386582 59.12537844 61.97671846
48.78799584 43.80247892 49.86453955 42.33198956 45.65708831 55.54140987
58.18326503 58.78127281 59.74968846 58.53094196 58.52594837 56.64551935
60.65594134 62.40091809 59.47134868 64.03042477 63.85597988 65.03726418
66.01249343 66.19987465 64.28345121 66.39414224 65.51544739 65.01058839
59.94493528 56.8440357 46.91477046 48.26646143 55.86675409 53.19869015
56.74491147 59.79565719 59.97125412 60.56167034 57.50777724 56.90700336]
[2.12791753 2.13369086 2.33001234 2.36626328 2.33513688 2.10601571
1.98815808 2.29756396 2.35847087 2.3860175 2.30676648 2.37504947
2.48235263 2.7087073 3.07862742 2.91142283 2.97288395 2.78748786
2.85032465 2.87404664 2.77150224 2.30287383 2.22000648 1.99867815
1.83605791 2.46834309 2.7308853 2.900398 2.79002996 2.72487201
2.81249071 2.87697953 2.71309994 2.46879037 2.3834721 2.37462879
2.50102608 2.21264029 2.26425564 2.23904949 2.149617 1.96593532
1.99694957 2.03151406 1.87300822 2.05758559 2.05692759 2.0234704 ]
[27.29734838 23.91758071 23.57768041 24.69502249 24.27674267 27.70215664
29.71346062 26.92210695 25.44503172 25.11040502 25.63128037 26.09491682
19.65393442 16.17099008 16.19700362 14.53996621 15.35784412 19.92525621
20.41285544 20.45244219 21.55859284 25.41647801 26.36296287 28.34149128
33.03596301 25.28048812 21.77731469 22.076427 22.88720221 23.86800702
23.4711863 23.0102001 23.69372775 26.89338996 27.48739853 27.37715831
23.96813682 25.69059052 20.71973214 21.55667467 25.98916653 27.0602444
28.41579599 29.43403567 32.0186817 29.43336623 27.95809508 28.12346716]
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.
36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.
69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

```
In [ ]: print(Hydro_ypred/ypred)
print(predictions)

print(predictionshydropower)

Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price
```

```
[2.25604798 2.29188667 2.57630484 2.55584979 2.56406259 2.21625267  
2.01655948 2.30720266 2.48291747 2.52145041 2.45607843 2.39829338  
2.48496669 1.96428088 2.8866002 1.73241077 2.33139309 3.01748393  
3.05470853 3.06112332 2.94200607 2.47365241 2.3659494 2.11497176  
1.71579562 2.4851252 2.91421223 2.81861926 2.72964562 2.5732868  
2.57452968 2.61702592 2.62152705 2.1817924 2.15622306 2.18588829  
2.65591707 2.40695173 2.16039774 2.25389332 2.3431474 2.11213121  
2.10983113 2.04871635 1.80397712 2.04955505 2.17469496 2.14408279]  
[52.82161316 53.03600615 53.25039913 53.46479211 53.67918509 53.89357808  
54.10797106 54.32236404 54.53675703 54.75115001 54.96554299 55.17993597  
55.39432896 55.60872194 55.82311492 56.03750791 56.25190089 56.46629387  
56.68068685 56.89507984 57.10947282 57.3238658 57.53825879 57.75265177  
57.96704475 58.18143773 58.39583072 58.6102237 58.82461668 59.03900967  
59.25340265 59.46779563 59.68218861 59.8965816 60.11097458 60.32536756  
60.53976055 60.75415353 60.96854651 61.18293949 61.39733248 61.61172546  
61.82611844 62.04051143 62.25490441 62.46929739 62.68369037 62.89808336]  
[58.08650622 51.03272344 54.9362862 58.43492496 56.6895171 58.34117714  
59.07505676 61.85526261 60.01136619 59.91386582 59.12537843 61.97671846  
48.78799584 43.80247892 49.86453955 42.33198956 45.65708831 55.54140987  
58.18326503 58.78127281 59.74968846 58.53094196 58.52594837 56.64551935  
60.65594134 62.40091809 59.47134868 64.03042477 63.85597988 65.03726418  
66.01249343 66.19987465 64.28345121 66.39414224 65.51544739 65.01058839  
59.94493528 56.8440357 46.91477046 48.26646143 55.86675409 53.19869015  
56.74491147 59.79565719 59.97125412 60.56167034 57.50777724 56.90700336]  
[52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04,  
56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59.  
.68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]
```

```
In [ ]: print(list(predictionshydropower))
```

```
[58.086506216931475, 51.032723442515774, 54.93628620492427, 58.43492496208573, 56.689517103506034, 58.3411771416  
3843, 59.07505676474696, 61.8552626083986, 60.01136618584462, 59.91386581908864, 59.125378435090305, 61.97671846  
143919, 48.78799583877904, 43.802478920776956, 49.8645395541369, 42.331989559057746, 45.65708831182323, 55.54140  
987192436, 58.18326502523207, 58.7812728123209, 59.74968845622487, 58.53094196352282, 58.52594837406535, 56.6455  
1934834873, 60.655941335778145, 62.40091809063356, 59.47134867792718, 64.03042476965678, 63.855979878254175, 65.  
.03726417747669, 66.01249343191415, 66.19987464992916, 64.2834512148047, 66.39414223631847, 65.51544738869949, 65  
.01058839062595, 59.94493528451246, 56.84403570171733, 46.91477046231597, 48.266461425452476, 55.86675409365672,  
53.19869014737627, 56.744911467797806, 59.7956571866573, 59.97125412308502, 60.56167034172636, 57.50777724408225  
6, 56.90700336147279]
```

```
In [ ]: print(list(predictions))
```

```
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445  
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739  
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293  
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75  
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.  
03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,  
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988  
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373  
92337, 62.89808335677895]
```

```
In [ ]: dfHydroReg = ({ "Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509  
"HydroReg" : [58.08647110309061, 51.03378110039499, 54.936739123030414, 58.434835871228046, 56.68969841115775,  
print(dfHydroReg) #Dataframes  
df_HydroReg= pd.DataFrame.from_dict(dfHydroReg, orient = "columns")  
print(df_HydroReg)
```

```
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary  
    else:  
        print("weak evidence against null hypothesis, indicating it is non-stationary ")  
  
adfuller_test(df_HydroReg["HydroReg"])  
  
test_result=adfuller(df_HydroReg["HydroReg"])
```

```
{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'HydroReg': [58.08647110309061, 51.03378110039499, 54.936739123030414, 58.434835871228046, 56.68969841115775, 58.34110257418901, 59.074868504698294, 61.85464363898176, 60.01103287278123, 59.91354761077782, 59.125182379210656, 61.976080676086895, 48.78940124976083, 43.80465668782247, 49.86577818701321, 42.33439513425111, 45.65897876286269, 55.54176904429836, 58.18321492152083, 58.781130065269345, 59.749395682263945, 58.53083799771545, 58.525845181864646, 56.64570747213075, 60.65550816516061, 62.400214588293196, 59.4710990243528, 64.02946882421269, 63.85505095780506, 65.03615225251541, 66.01123042447925, 66.19858261340414, 64.28245607051475, 66.39282010386744, 65.51426138361596, 65.00948059827641, 59.94461226292141, 56.844193071354546, 46.91646607329147, 48.26794763252286, 55.867062863718076, 53.19941225378794, 56.745084193757016, 59.795357291222665, 59.97092702418064, 60.561251775567115, 57.50783178690864, 56.907150976162896]}
```

	Price	HydroReg
0	52.821613	58.086471
1	53.036006	51.033781
2	53.250399	54.936739
3	53.464792	58.434836
4	53.679185	56.689698
5	53.893578	58.341103
6	54.107971	59.074869
7	54.322364	61.854644
8	54.536757	60.011033
9	54.751150	59.913548
10	54.965543	59.125182
11	55.179936	61.976081
12	55.394329	48.789401
13	55.608722	43.804657
14	55.823115	49.865778
15	56.037508	42.334395
16	56.251901	45.658979
17	56.466294	55.541769
18	56.680687	58.183215
19	56.895080	58.781130
20	57.109473	59.749396
21	57.323866	58.530838
22	57.538259	58.525845
23	57.752652	56.645707
24	57.967045	60.655508
25	58.181438	62.400215
26	58.395831	59.471099
27	58.610224	64.029469
28	58.824617	63.855051
29	59.039010	65.036152
30	59.253403	66.011230
31	59.467796	66.198583
32	59.682189	64.282456
33	59.896582	66.392820
34	60.110975	65.514261
35	60.325368	65.009481
36	60.539761	59.944612
37	60.754154	56.844193
38	60.968547	46.916466
39	61.182939	48.267948
40	61.397332	55.867063
41	61.611725	53.199412
42	61.826118	56.745084
43	62.040511	59.795357
44	62.254904	59.970927
45	62.469297	60.561252
46	62.683690	57.507832
47	62.898083	56.907151

ADF Test Statistic : -3.1850778204255854

p-value : 0.020864845170417996

#Lags Used : 3

Number of Observations : 44

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```
In [ ]: df_HydroReg['First Difference'] = df_HydroReg["HydroReg"]- df_HydroReg["HydroReg"].shift(1) # Seasonality value
df_HydroReg['Seasonal Difference']=df_HydroReg["HydroReg"]- df_HydroReg["HydroReg"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted linear hydro prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_HydroReg.head() #Predictive Linear Seasonality Plot
df_HydroReg['Seasonal Difference'].plot()
# Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a1d4710>
```



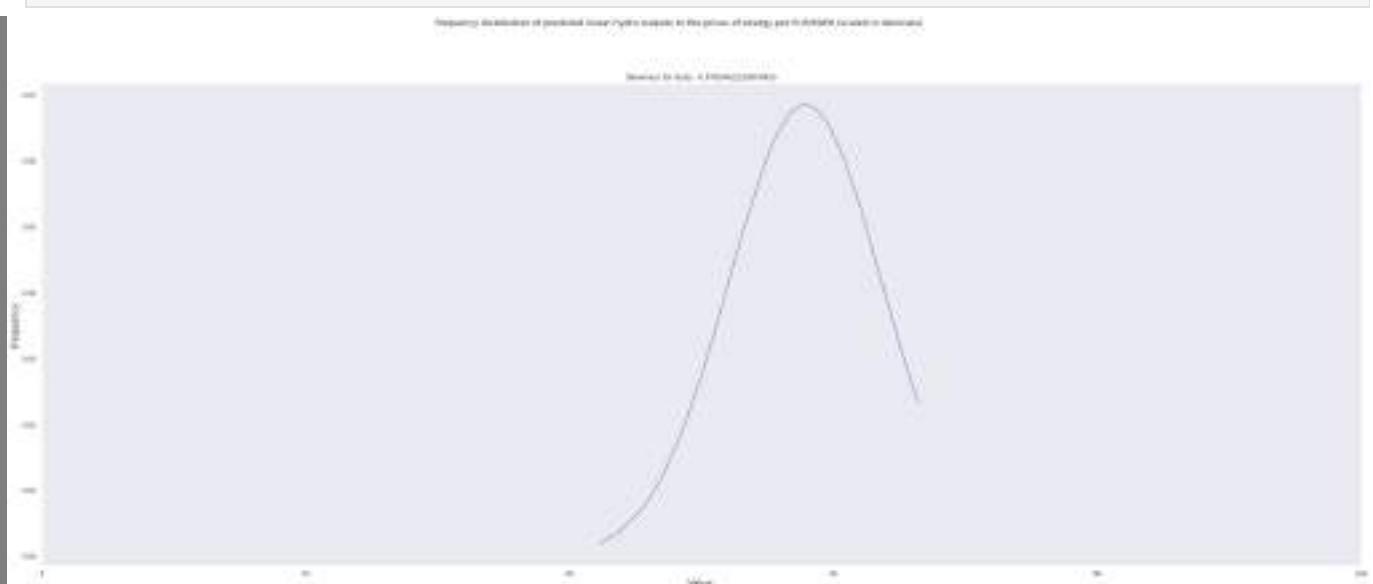
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
HydroRegResults_mean = np.mean(df_HydroReg["HydroReg"])
HydroRegResults_std = np.std(df_HydroReg["HydroReg"])

HydroRegResultspdf = stats.norm.pdf(df_HydroReg["HydroReg"].sort_values(), HydroRegResults_mean, HydroRegResults_std)

plt.plot(df_HydroReg["HydroReg"].sort_values(), HydroRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_HydroReg["HydroReg"])}')
plt.suptitle("Frequency distribution of predicted linear hydro outputs to the prices of energy per EUR/MWH (scaled)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the left, hence it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
HydroReg_Dict = {key: i for i, key in enumerate(df_HydroReg["HydroReg"])}

def Hist_HydroReg(HydroReg_Dict):
    for k, v in HydroReg_Dict.items(): print(f"{v}:{k}")
```

```

print(HydroReg_Dict)

plt.bar(list(HydroReg_Dict.values()), HydroReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted hydro linear prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

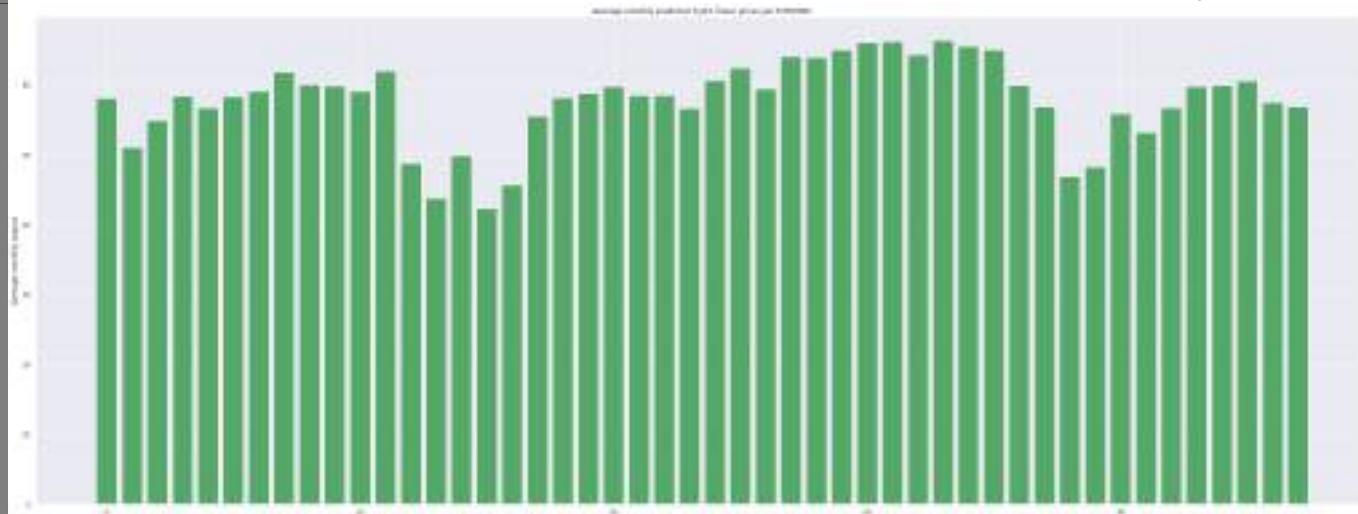
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{58.08647110309061: 0, 51.03378110039499: 1, 54.936739123030414: 2, 58.434835871228046: 3, 56.68969841115775: 4, 58.34110257418901: 5, 59.074868504698294: 6, 61.85464363898176: 7, 60.01103287278123: 8, 59.91354761077782: 9, 59.125182379210656: 10, 61.976080676086895: 11, 48.78940124976083: 12, 43.80465668782247: 13, 49.86577818701321: 14, 42.33439513425111: 15, 45.65897876286269: 16, 55.54176904429836: 17, 58.18321492152083: 18, 58.7811300652693: 19, 59.749395682263945: 20, 58.53083799771545: 21, 58.525845181864646: 22, 56.64570747213075: 23, 60.6555081: 24, 62.400214588293196: 25, 59.47109902435258: 26, 64.02946882421269: 27, 63.85505095780506: 28, 65.036: 29, 66.01123042447925: 30, 66.19858261340414: 31, 64.28245607051475: 32, 66.39282010386744: 33, 65.51426138361596: 34, 65.00948059827641: 35, 59.94461226292141: 36, 56.844193071354546: 37, 46.91646607329147: 38, 48.26794763252286: 39, 55.867062863718076: 40, 53.19941225378794: 41, 56.745084193757016: 42, 59.795357291222665: 43, 59.97092702418064: 44, 60.561251775567115: 45, 57.50783178690864: 46, 56.907150976162896: 47}

```



The green bars represent the observation value for each respective month. This multimodal histogram has trenches roughly every ten months. The values are decreasing overtime.

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: df_HydroReg.describe(include = 'all') # Description Tables
```

	Price	HydroReg	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	57.859848	57.859848	-0.025092	-0.748630
std	3.001502	5.879510	4.196490	9.728802
min	52.821613	42.334395	-13.186679	-16.100441
25%	55.340731	56.451046	-1.861874	-7.992598
50%	57.859848	58.655984	0.187352	-2.936424
75%	60.378966	60.955292	2.710611	7.520093
max	62.898083	66.392820	9.882790	21.695074

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(ypred))
```

```
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.67299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: print(list(Hydro_ypred))
```

```
[61.3376076524368, 53.23032421639376, 58.94131093358947, 61.481266818688226, 60.51882212939074, 61.44499015745893, 61.68219642784088, 61.60803683034199, 61.82912405339957, 61.82196732038972, 61.694532263137944, 61.569706366514396, 48.572059842755415, 34.63696257388266, 50.93132503376881, 29.581510622142275, 40.39897314063682, 59.55496108363039, 61.379927561757256, 61.600112569182286, 61.80563868511031, 61.516607337639925, 61.51481462879916, 60.48672198576486, 61.82881070824977, 61.41278538169082, 61.76569175276494, 60.47678138618451, 60.6022577646758, 59.634147791875286, 58.62550365257845, 58.410010679349355, 60.2840125073846, 58.17921591815906, 59.1632459278975, 59.65907642754619, 61.824453393966806, 60.628502478567874, 43.916671656036094, 47.346121018397795, 59.85474758644083, 56.77391285910282, 60.55868887159784, 61.81075166227316, 61.82640894900889, 61.834021985154315, 61.04556169294959, 60.67183459826377]
```

```
In [ ]: dfHydro_Quad = ({ "Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.67299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]})
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + ' )')

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_Hydro_Quad["Hydro_Quad"])

test_result=adfuller(df_Hydro_Quad["Hydro_Quad"])

df_Hydro_Quad['First Difference'] = df_Hydro_Quad["Hydro_Quad"]- df_Hydro_Quad["Hydro_Quad"].shift(1) # Seasonal
df_Hydro_Quad['Seasonal Difference']=df_Hydro_Quad["Hydro_Quad"]- df_Hydro_Quad["Hydro_Quad"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Hydro_Quad.head() #Predictive Quadratic Seasonality Plot
df_Hydro_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.2722794943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Hydro_Quad': [61.3376076524368, 53.23032421639376, 58.94131093358947, 61.481266818688226, 60.51882212939074, 61.44499015745893, 61.68219642784088, 61.60803683034199, 61.82912405339957, 61.82196732038972, 61.694532263137944, 61.569706366514396, 48.572059842755415, 34.63696257388266, 50.93132503376881, 29.581510622142275, 40.39897314063682, 59.55496108363039, 61.379927561757256, 61.600112569182286, 61.80563868511031, 61.516607337639925, 61.51481462879916, 60.48672198576486, 61.82881070824977, 61.41278538169082, 61.76569175276494, 60.47678138618451, 60.6022577646758, 59.634147791875286, 58.62550365257845, 58.410010679349355, 60.2840125073846, 58.17921591815906, 59.1632459278975, 59.65907642754619, 61.824453393966806, 60.628502478567874, 43.916671656036094, 47.346121018397795, 59.85474758644083, 56.77391285910282, 60.55868887159784, 61.81075166227316, 61.82640894900889, 61.834021985154315, 61.04556169294959, 60.67183459826377], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Quad	Hydro_Quad	Dates
0	27.188078	61.337608	2015-01
1	23.225548	53.230324	2015-02
2	22.878236	58.941311	2015-03
3	24.055117	61.481267	2015-04
4	23.602709	60.518822	2015-05
5	27.724722	61.444990	2015-06
6	30.587839	61.682196	2015-07
7	26.702482	61.608037	2015-08
8	24.901804	61.829124	2015-09
9	24.518415	61.821967	2015-10
10	25.119121	61.694532	2015-11
11	25.672300	61.569706	2015-12
12	19.546363	48.572060	2016-01
13	17.633406	34.636963	2016-02
14	17.644052	50.931325	2016-03
15	17.075344	29.581511	2016-04
16	17.328255	40.398973	2016-05
17	19.736629	59.554961	2016-06
18	20.093546	61.379928	2016-07
19	20.123369	61.600113	2016-08
20	21.007992	61.805639	2016-09
21	24.868735	61.516607	2016-10
22	26.000055	61.514815	2016-11
23	28.599305	60.486722	2016-12
24	36.035067	61.828811	2017-01
25	24.712150	61.412785	2017-02
26	21.194644	61.765692	2017-03
27	21.456173	60.476781	2017-04
28	22.201511	60.602258	2017-05
29	23.174311	59.634148	2017-06
30	22.771345	58.625504	2017-07
31	22.319233	58.410011	2017-08
32	22.995762	60.284013	2017-09
33	26.665789	58.179216	2017-10
34	27.438370	59.163246	2017-11
35	27.292830	59.659076	2017-12
36	23.278006	61.824453	2018-01
37	25.188915	60.628502	2018-02
38	20.328049	43.916672	2018-03
39	21.006372	47.346121	2018-04
40	25.544593	59.854748	2018-05
41	26.879918	56.773913	2018-06
42	28.703098	60.558689	2018-07
43	30.170478	61.810752	2018-08
44	34.272280	61.826409	2018-09
45	30.169486	61.834022	2018-10
46	28.070862	61.045562	2018-11
47	28.297338	60.671835	2018-12

ADF Test Statistic : -3.559779906181209

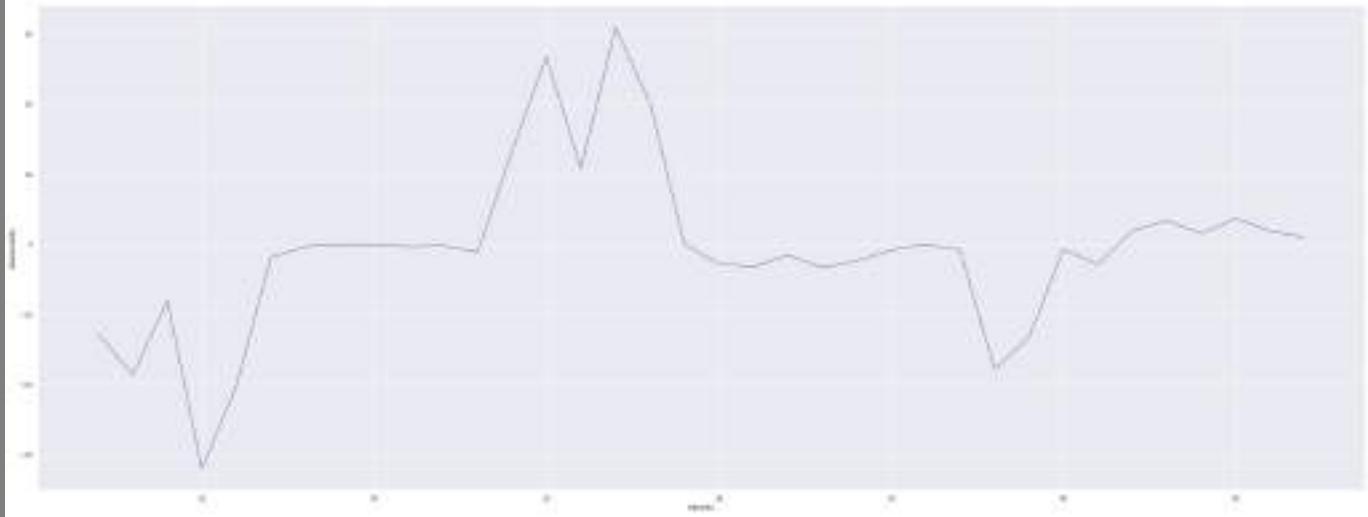
p-value : 0.006578174720780082

#Lags Used : 0

Number of Observations : 47

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a8cee50>



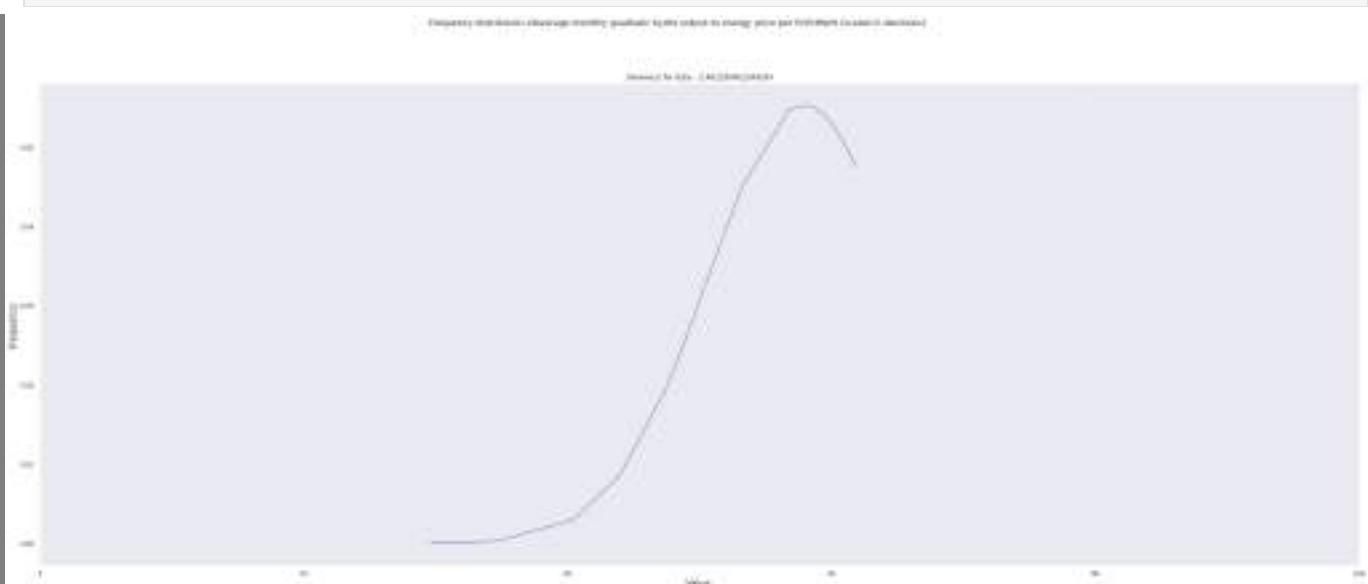
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Hydro_QuadResults_mean = np.mean(df_Hydro_Quad["Hydro_Quad"])
Hydro_QuadResults_std = np.std(df_Hydro_Quad["Hydro_Quad"])

Hydro_QuadResultspdf = stats.norm.pdf(df_Hydro_Quad["Hydro_Quad"].sort_values(), Hydro_QuadResults_mean, Hydro_QuadResults_std)

plt.plot(df_Hydro_Quad["Hydro_Quad"].sort_values(), Hydro_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Hydro_Quad["Hydro_Quad"])}')
plt.suptitle("Frequency distribution of average monthly quadratic hydro output to energy price per EUR/MWH (scaled)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
print(dicDates)
Hydro_Quad_Dict = {key: i for i, key in enumerate(df_Hydro_Quad["Hydro_Quad"])}

def Hist_Hydro_Quad(Hydro_Quad_Dict):
    for k, v in Hydro_Quad_Dict.items(): print(f"{v}:{k}")
print(Hydro_Quad_Dict)
```

```

plt.bar(list(Hydro_Quad_Dict.values()), Hydro_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic hydro prices per EUR/MWh ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

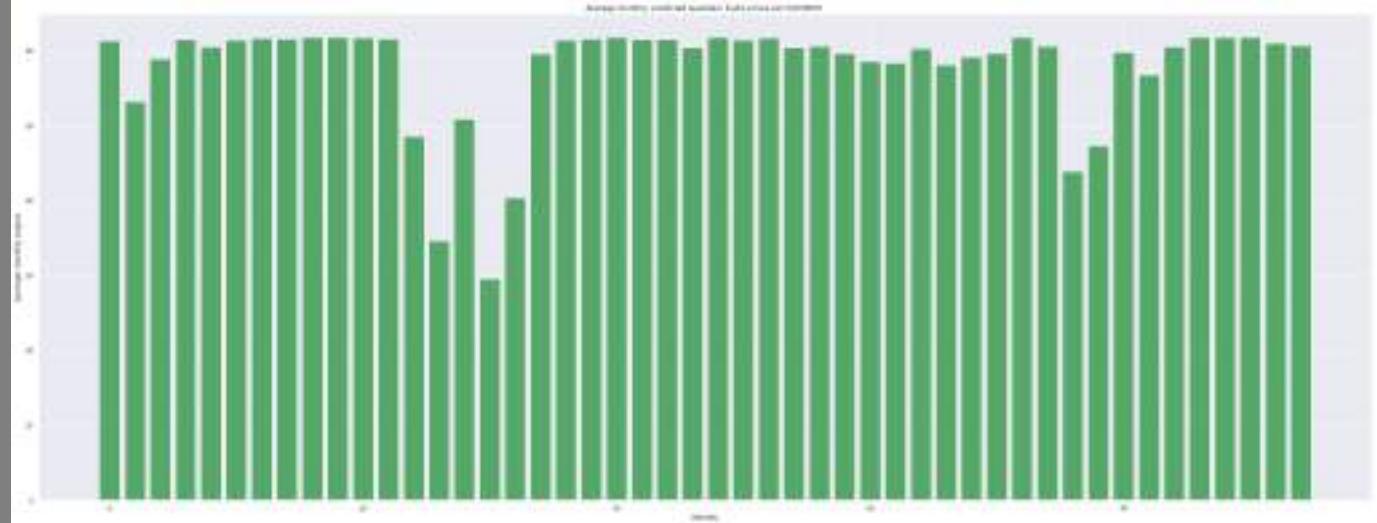
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{61.3376076524368: 0, 53.23032421639376: 1, 58.94131093358947: 2, 61.481266818688226: 3, 60.51882212939074: 4, 61.44499015745893: 5, 61.68219642784088: 6, 61.60803683034199: 7, 61.82912405339957: 8, 61.82196732038972: 9, 61.694532263137944: 10, 61.569706366514396: 11, 48.572059842755415: 12, 34.63696257388266: 13, 50.93132503376881: 14, 29.581510622142275: 15, 40.39897314063682: 16, 59.55496108363039: 17, 61.379927561757256: 18, 61.600112569182286: 19, 61.80563868511031: 20, 61.516607337639925: 21, 61.51481462879916: 22, 60.48672198576486: 23, 61.82881070824977: 24, 61.41278538169082: 25, 61.76569175276494: 26, 60.47678138618451: 27, 60.6022577646758: 28, 59.634147791875286: 29, 58.62550365257845: 30, 58.410010679349355: 31, 60.2840125073846: 32, 58.17921591815906: 33, 59.1632459278975: 34, 59.65907642754619: 35, 61.824453393966806: 36, 60.628502478567874: 37, 43.916671656036094: 38, 47.346121018397795: 39, 59.85474758644083: 40, 56.77391285910282: 41, 60.55868887159784: 42, 61.81075166227316: 43, 61.82640894900889: 44, 61.834021985154315: 45, 61.04556169294959: 46, 60.67183459826377: 47}

```



The green bars represent the observation value for each respective month. With the exception of a few trenches, this histogram is uniform.

```
In [ ]: df_Hydro_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Hydro_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	-0.014165	-0.807450
std	4.174498	7.281292	NaN	6.875286	11.859763
min	17.075344	29.581511	NaN	-21.349814	-31.899756
25%	21.390791	58.862359	NaN	-0.965277	-2.942702
50%	24.615282	60.615380	NaN	0.007613	-0.526435
75%	27.214266	61.602094	NaN	1.297076	1.627376
max	36.035067	61.834022	NaN	19.155988	30.895271

```
In [ ]: print(list(Hydro_Logpred))
```

```

[58.086506216931475, 51.032723442515774, 54.93628620492427, 58.43492496208573, 56.689517103506034, 58.34117714163843, 59.07505676474696, 61.8552626083986, 60.01136618584462, 59.91386581908864, 59.125378435090305, 61.97671846143919, 48.78799583877904, 43.802478920776956, 49.8645395541369, 42.331989559057746, 45.65708831182323, 55.54140987192436, 58.18326502523207, 58.7812728123209, 59.74968845622487, 58.53094196352282, 58.52594837406535, 56.64551934834873, 60.655941335778145, 62.40091809063356, 59.47134867792718, 64.03042476965678, 63.855979878254175, 65.03726417747669, 66.01249343191415, 66.19987464992916, 64.2834512148047, 66.39414223631847, 65.51544738869949, 65.01058839062595, 59.94493528451246, 56.84403570171733, 46.91477046231597, 48.266461425452476, 55.86675409365672, 53.19869014737627, 56.744911467797806, 59.7956571866573, 59.97125412308502, 60.56167034172636, 57.507777244082256, 56.90700336147279]

```

```
In [ ]: print(list(Logpred))
```

```
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: dfHydro_Log = ({ "Price_Log": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.086506216931475, 51.032723442515774, 54.93628620492427, 58.43492496208573, 56.03596300746373], "Hydro_Log": [58.086506216931475, 51.032723442515774, 54.93628620492427, 58.43492496208573, 56.03596300746373], "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08']})
```

```
print(dfHydro_Log) #Dataframes
```

```
df_Hydro_Log= pd.DataFrame.from_dict(dfHydro_Log, orient = "columns")
```

```
print(df_Hydro_Log)
```

```
#ADF Tests
```

```
from statsmodels.tsa.stattools import adfuller
```

```
def adfuller_test(test_result):
```

```
    result=adfuller(test_result)
```

```
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
```

```
    for value,label in zip(result,labels):
```

```
        print(label+' : '+str(value) )
```



```
    if result[1] <= 0.05:
```

```
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
```

```
    else:
```

```
        print("weak evidence against null hypothesis,indicating it is non-stationary ")
```

```
adfuller_test(df_Hydro_Log["Hydro_Log"])
```



```
test_result=adfuller(df_Hydro_Log["Hydro_Log"])
```

```

{'Price_Log': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.00637178859053, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Hydro_Log': [58.086506216931475, 51.032723442515774, 54.93628620492427, 58.43492496208573, 56.689517103506034, 58.34117714163843, 59.07505676474696, 61.8552626083986, 60.01136618584462, 59.91386581908864, 59.125378435090305, 61.97671846143919, 48.78799583877904, 43.802478920776956, 49.8645395541369, 42.331989559057746, 45.65708831182323, 55.54140987192436, 58.18326502523207, 58.7812728123209, 59.74968845622487, 58.53094196352282, 58.52594837406535, 56.64551934834873, 60.655941335778145, 62.40091809063356, 59.47134867792718, 64.03042476965678, 63.855979878254175, 65.03726417747669, 66.01249343191415, 66.19987464992916, 64.2834512148047, 66.39414223631847, 65.51544738869949, 65.01058839062595, 59.94493528451246, 56.84403570171733, 46.91477046231597, 48.266461425452476, 55.86675409365672, 53.19869014737627, 56.744911467797806, 59.7956571866573, 59.97125412308502, 60.56167034172636, 57.507777244082256, 56.90700336147279], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price_Log    Hydro_Log    Dates
0   27.188078  58.086506  2015-01
1   23.225548  51.032723  2015-02
2   22.878236  54.936286  2015-03
3   24.055117  58.434925  2015-04
4   23.602709  56.689517  2015-05
5   27.724722  58.341177  2015-06
6   30.587839  59.075057  2015-07
7   26.702482  61.855263  2015-08
8   24.901804  60.011366  2015-09
9   24.518415  59.913866  2015-10
10  25.119121  59.125378  2015-11
11  25.672300  61.976718  2015-12
12  19.546363  48.787996  2016-01
13  17.633406  43.802479  2016-02
14  17.644052  49.864540  2016-03
15  17.075344  42.331990  2016-04
16  17.328255  45.657088  2016-05
17  19.736629  55.541410  2016-06
18  20.093546  58.183265  2016-07
19  20.123369  58.781273  2016-08
20  21.007992  59.749688  2016-09
21  24.868735  58.530942  2016-10
22  26.000055  58.525948  2016-11
23  28.599305  56.645519  2016-12
24  36.035067  60.655941  2017-01
25  24.712150  62.400918  2017-02
26  21.194644  59.471349  2017-03
27  21.456173  64.030425  2017-04
28  22.201511  63.855980  2017-05
29  23.174311  65.037264  2017-06
30  22.771345  66.012493  2017-07
31  22.319233  66.199875  2017-08
32  22.995762  64.283451  2017-09
33  26.665789  66.394142  2017-10
34  27.438370  65.515447  2017-11
35  27.292830  65.010588  2017-12
36  23.278006  59.944935  2018-01
37  25.188915  56.844036  2018-02
38  20.328049  46.914770  2018-03
39  21.006372  48.266461  2018-04
40  25.544593  55.866754  2018-05
41  26.879918  53.198690  2018-06
42  28.703098  56.744911  2018-07
43  30.170478  59.795657  2018-08
44  34.272280  59.971254  2018-09
45  30.169486  60.561670  2018-10
46  28.070862  57.507777  2018-11
47  28.297338  56.907003  2018-12

ADF Test Statistic : -3.1850778204255867
p-value : 0.020864845170417895
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

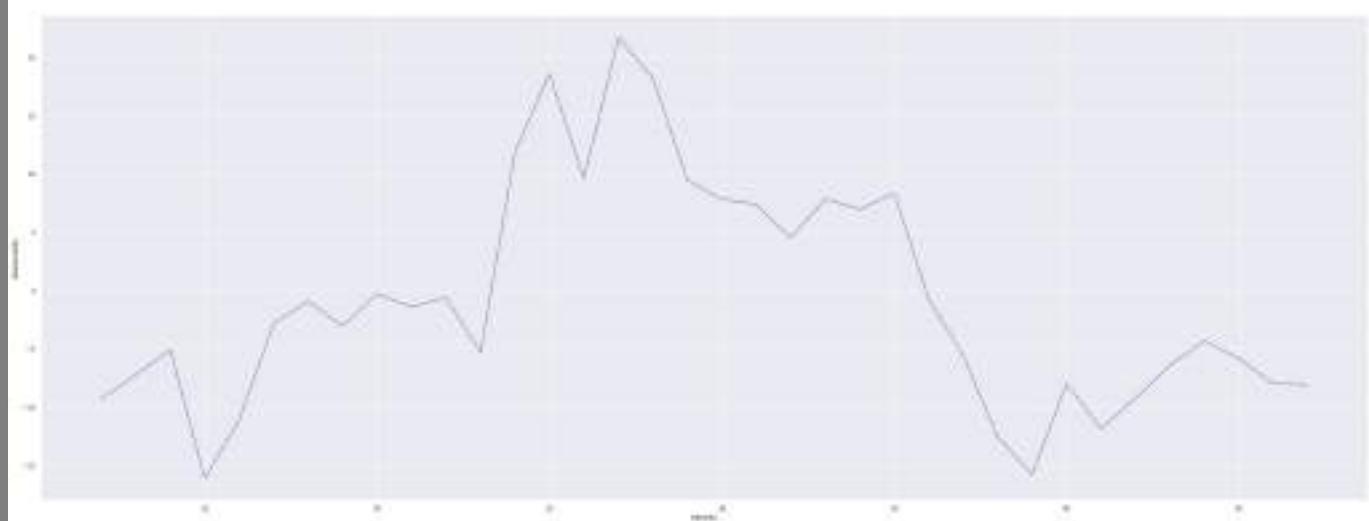
```

In [ ]: df_Hydro_Log['First Difference'] = df_Hydro_Log["Hydro_Log"] - df_Hydro_Log["Hydro_Log"].shift(1) # Seasonality
df_Hydro_Log['Seasonal Difference']=df_Hydro_Log["Hydro_Log"]- df_Hydro_Log["Hydro_Log"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic hydro prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Hydro_Log.head() #Predictive Logarithmic Seasonality Plot

```

```
df_Hydro_Log['Seasonal Difference'].plot()  
# Seasonality Plot
```

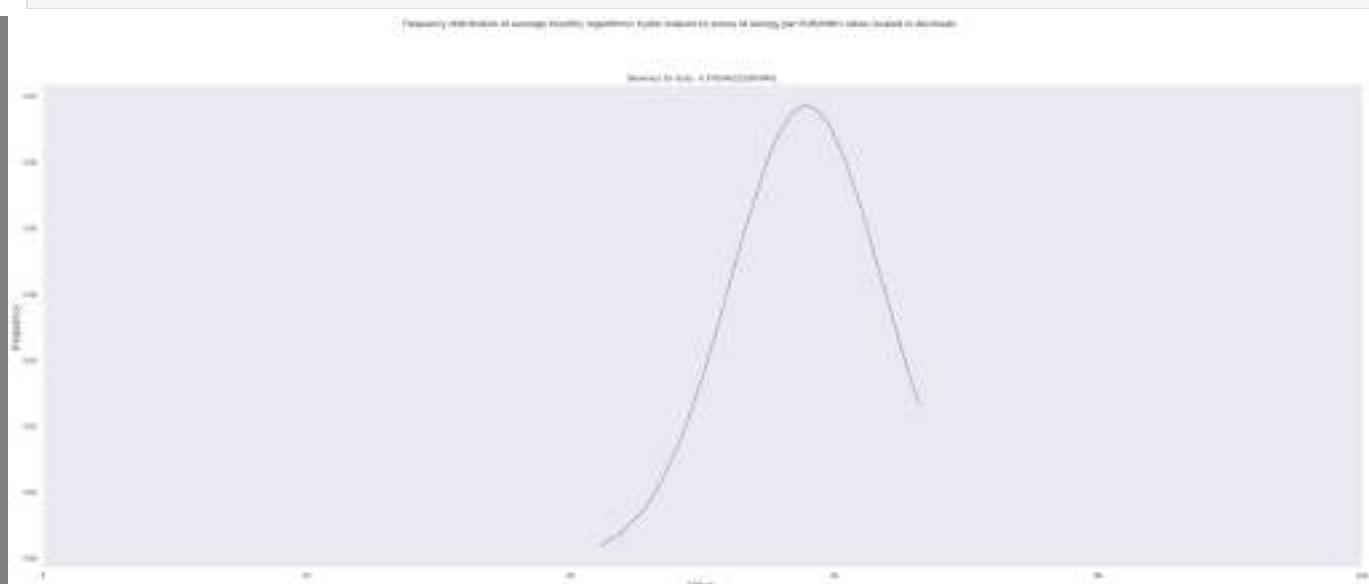
Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a308f50>



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Hydro_LogResults_mean = np.mean(df_Hydro_Log["Hydro_Log"])  
Hydro_LogResults_std = np.std(df_Hydro_Log["Hydro_Log"])  
  
Hydro_LogResultspdf = stats.norm.pdf(df_Hydro_Log["Hydro_Log"].sort_values(), Hydro_LogResults_mean, Hydro_LogRe  
  
plt.plot(df_Hydro_Log["Hydro_Log"].sort_values(), Hydro_LogResultspdf)  
plt.xlim([0,100])  
plt.xlabel("Value ", size=15)  
plt.title(f'Skewness for data: {skew(df_Hydro_Log["Hydro_Log"])}')  
plt.suptitle("Frequency distribution of average monthly logarithmic hydro outputs to prices of energy per EUR/MW  
plt.ylabel("Frequency", size=15)  
plt.grid(True, alpha=0.3, linestyle="--")  
plt.show()
```



This bell shaped curve is slightly skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)  
Hydro_Log_Dict = {key: i for i, key in enumerate(df_Hydro_Log["Hydro_Log"])}
```

```

def Hist_Hydro_Log(Hydro_Log_Dict):
    for k, v in Hydro_Log_Dict.items(): print(f"{v}:{k}")
print(Hydro_Log_Dict)

plt.bar(list(Hydro_Log_Dict.values()), Hydro_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly logarithmic hydro predicted logarithmic prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

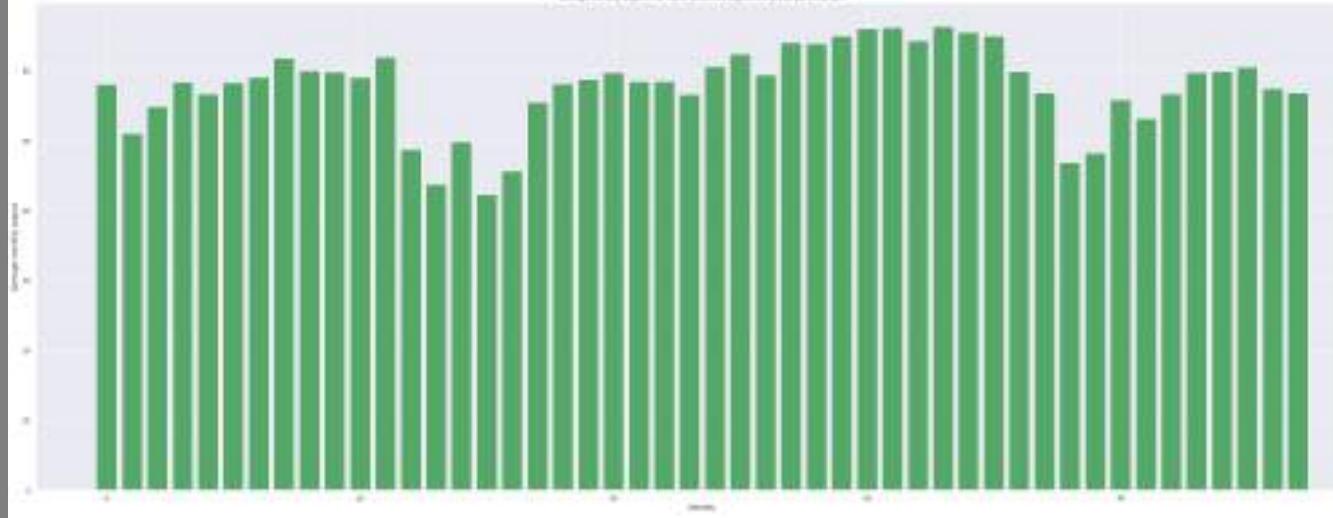
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{58.086506216931475: 0, 51.032723442515774: 1, 54.93628620492427: 2, 58.43492496208573: 3, 56.689517103506034: 4, 58.34117714163843: 5, 59.07505676474696: 6, 61.8552626083986: 7, 60.01136618584462: 8, 59.91386581908864: 9, 59.125378435090305: 10, 61.97671846143919: 11, 48.78799583877904: 12, 43.802478920776956: 13, 49.8645395541369: 14, 42.331989559057746: 15, 45.65708831182323: 16, 55.54140987192436: 17, 58.18326502523207: 18, 58.7812728123209: 19, 59.74968845622487: 20, 58.53094196352282: 21, 58.52594837406535: 22, 56.64551934834873: 23, 60.655941335778145: 24, 62.40091809063356: 25, 59.47134867792718: 26, 64.03042476965678: 27, 63.855979878254175: 28, 65.03726417747669: 29, 66.01249343191415: 30, 66.19987464992916: 31, 64.2834512148047: 32, 66.39414223631847: 33, 65.5154473869949: 34, 65.01058839062595: 35, 59.94493528451246: 36, 56.84403570171733: 37, 46.91477046231597: 38, 48.266461425452476: 39, 55.86675409365672: 40, 53.19869014737627: 41, 56.744911467797806: 42, 59.7956571866573: 43, 59.97125412308502: 44, 60.56167034172636: 45, 57.507777244082256: 46, 56.90700336147279: 47}

```



The green bars represent the observation value for each respective month. With the exception of a few trenches, this histogram is uniform.

```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Hydro_Logpred)
plt.suptitle("Autocorrelations of average monthly Hydro Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Hydro_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Hydro Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Hydro_Log_Autocorrelations = sm.tsa.acf(Hydro_Logpred, fft=False) #Autocorrelations
print(Hydro_Log_Autocorrelations)

```

```

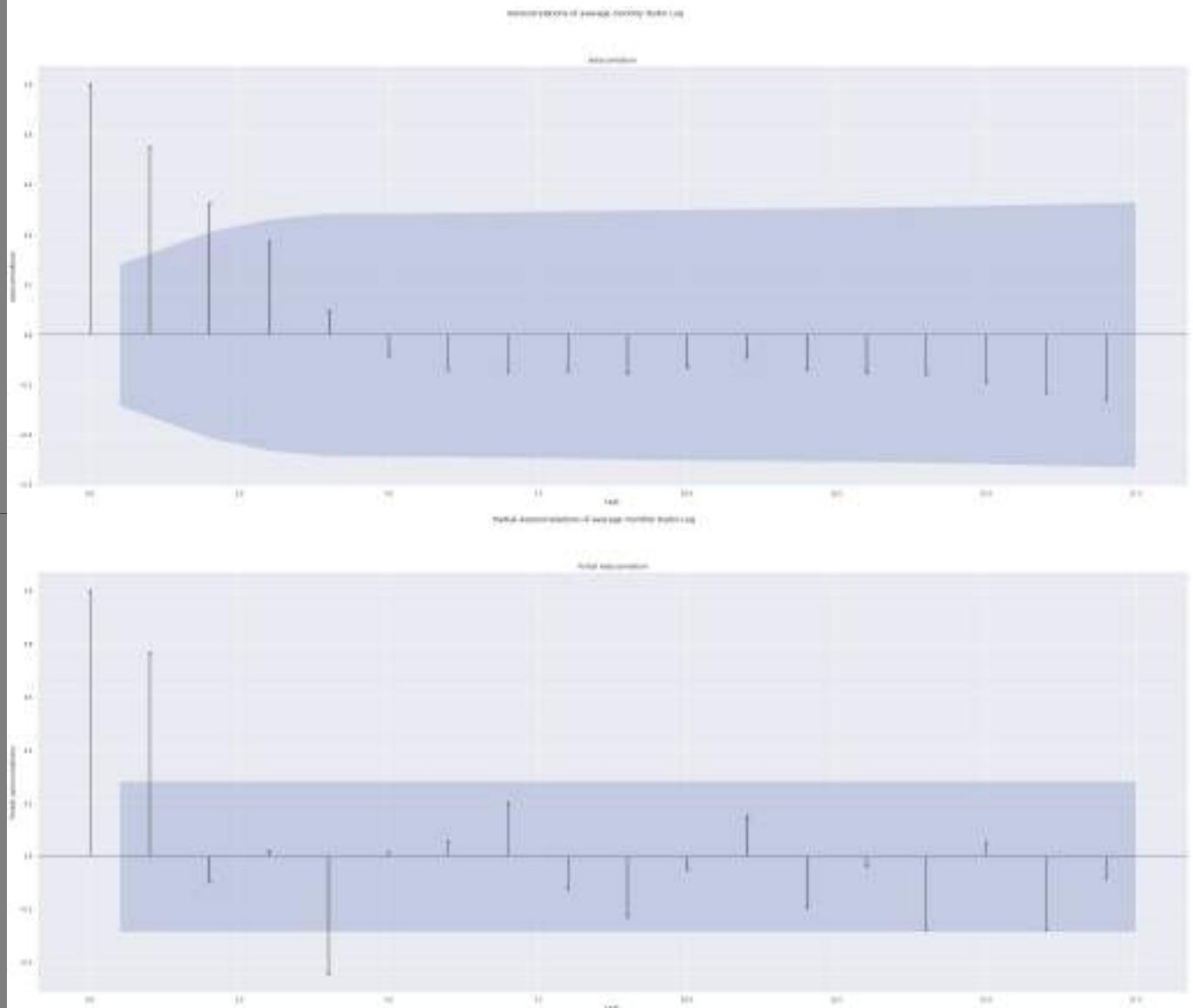
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```

```

[ 1.          0.75039708  0.52451454  0.36950145  0.09603364 -0.08483258
 -0.14248453 -0.14709291 -0.14342582 -0.15346075 -0.12905736 -0.08930078
 -0.13725747 -0.149127   -0.15540326 -0.1898376  -0.23216446 -0.26003238
 -0.26410381 -0.24404753 -0.18040301 -0.09309511  0.055291   0.17621231
 0.21764736  0.24333474  0.21183231  0.05638467 -0.03305661 -0.08108554
 -0.12945711 -0.12417628 -0.10857882 -0.06574946 -0.02132402  0.00204649
 0.02835408  0.07444157  0.0486353   0.01414937  0.01866687]

```



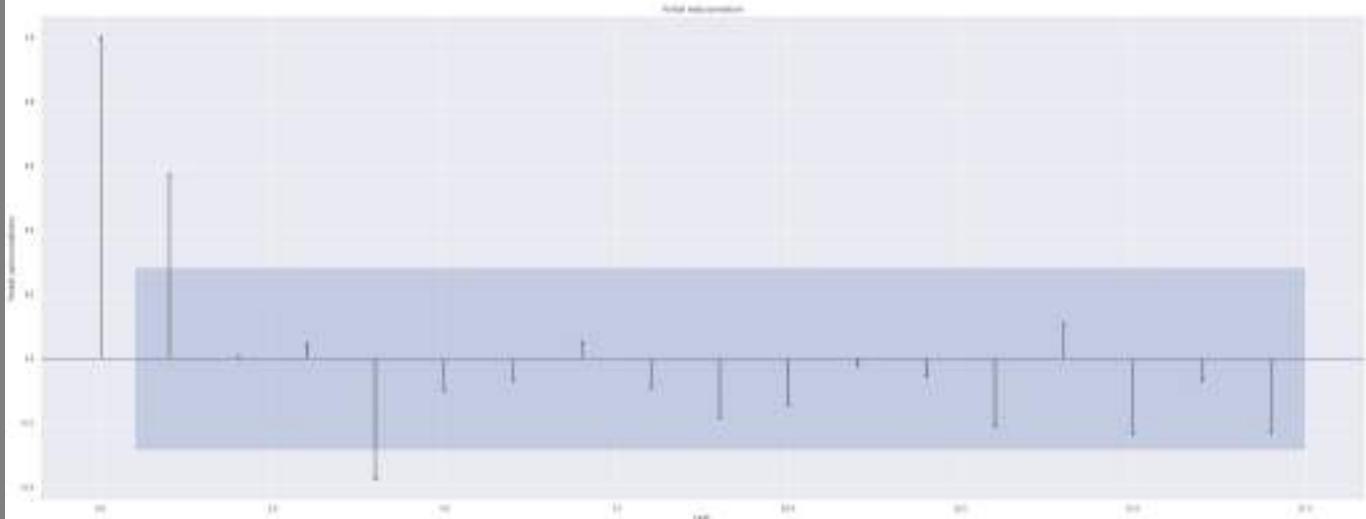
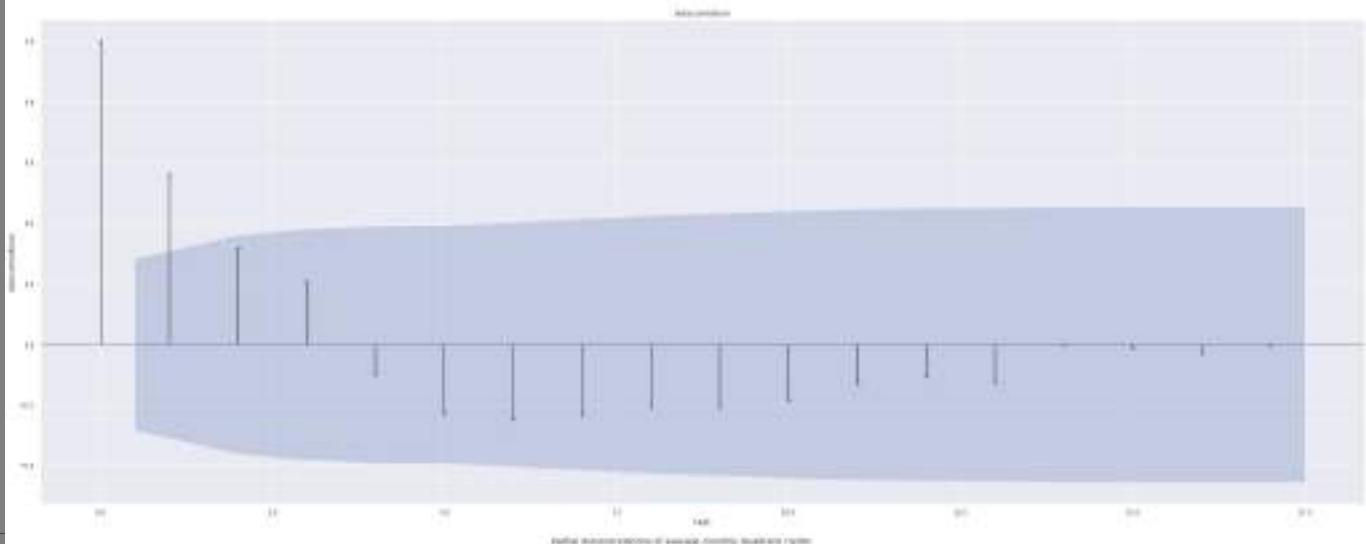
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Hydro_ypred)
plt.suptitle(" Autocorrelations of average monthly Quadratic Hydro")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show()
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Hydro_ypred)
plt.suptitle("Partial Autocorrelations of average monthly Quadratic Hydro")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show()
Hydro_Quad_Autocorrelations = sm.tsa.acf(Hydro_ypred, fft=False) #Autocorrelations
print(Hydro_Quad_Autocorrelations)
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.          0.55967511  0.31652063  0.20724571 -0.0967032 -0.22629804
-0.24458381 -0.23341702 -0.20636625 -0.20708846 -0.18359151 -0.12781473
-0.10535078 -0.12465042  0.00161017 -0.01026306 -0.02939573 -0.00273626
 0.01495387 -0.0180748 -0.03523148 -0.04530762  0.08504885  0.2314571
 0.1532518   0.18385573  0.18657444 -0.01751291 -0.08300414 -0.11402813
-0.14510739 -0.14063537 -0.10412459 -0.06344128 -0.03983677 -0.01689387
 0.011434    0.05858009  0.02953354  0.01036135  0.03078223]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

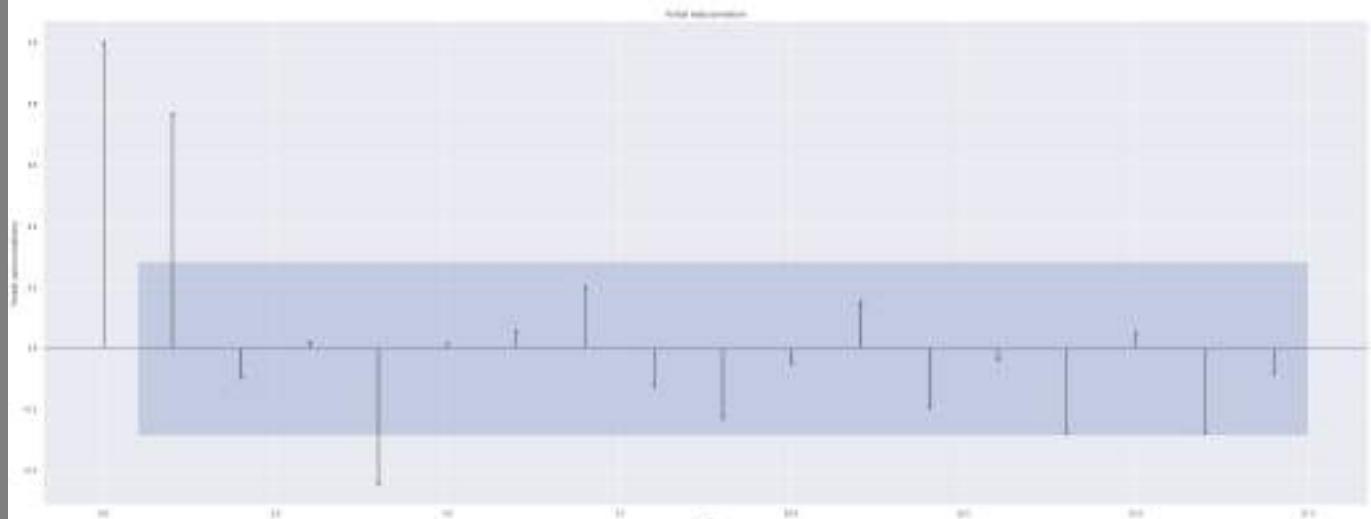
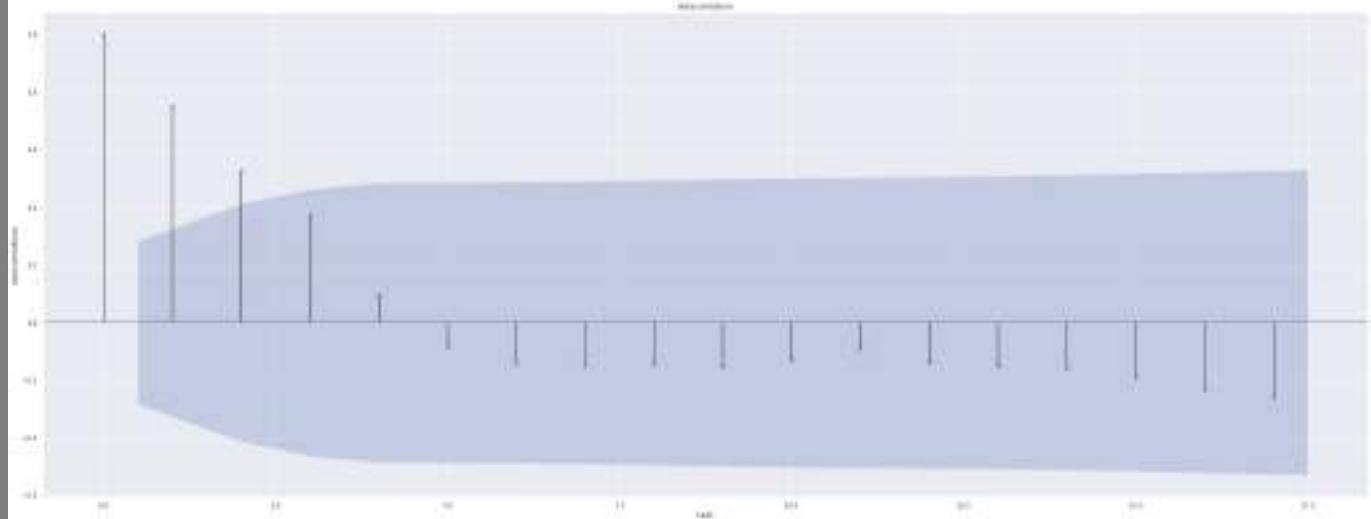
As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(predictionshydropower)
plt.suptitle(" Autocorrelations of average monthly Linear Hydro")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(predictionshydropower)
plt.suptitle("Partial Autocorrelations of average monthly Linear Hydro")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
Hydro_Pred_Autocorrelations = sm.tsa.acf(predictionshydropower, fft=False) #Autocorrelations
print(Hydro_Pred_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statauto.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.75039708  0.52451454  0.36950145  0.09603364 -0.08483258  
-0.14248453 -0.14709291 -0.14342582 -0.15346075 -0.12905736 -0.08930078  
-0.13725747 -0.149127   -0.15540326 -0.1898376  -0.23216446 -0.26003238  
-0.26410381 -0.24404753 -0.18040301 -0.09309511  0.055291    0.17621231  
0.21764736  0.24333474  0.21183231  0.05638467 -0.03305661 -0.08108554  
-0.12945711 -0.12417628 -0.10857882 -0.06574946 -0.02132402  0.00204649  
 0.02835408  0.07444157  0.0486353  0.01414937  0.01866687]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resource analyzed was nuclear energy.

```
In [ ]: nuclear1 = nuclear
```

```
nuclear1 = sm.add_constant(nuclear1)
```

```
In [ ]: #Dataframes analyzed by resource
```

```
dfnuclear = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139,  
"Nuclear": [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 56!]  
print(dfnuclear)  
df_nuclear= pd.DataFrame.from_dict(dfnuclear, orient = "columns")  
print(df_nuclear)  
df_nuclear["Ratio"] = df_nuclear["Nuclear"]/df_nuclear["Price"]
```

```
pdToListNuclear = list(df_nuclear["Ratio"])

print(pdToListNuclear)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.4920833333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.2582222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Nuclear': [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 5659.276773296245, 6483.623655913979, 6437.845430107527, 6466.175, 5854.012096774193, 5973.075, 6626.029609690444, 6223.8494623655915, 6159.264367816092, 6699.888290713325, 6706.406944444445, 5714.009408602151, 6647.463888888885, 6628.9220430107525, 6633.3494623655915, 6675.7555555555555, 6576.6859060402685, 5534.29722222222, 6325.970430107527, 6769.916666666667, 6739.267857142857, 6755.951547779273, 6676.383333333333, 5561.240591397849, 6063.85972222222, 6117.403225806452, 6675.846774193548, 6427.688888888889, 6003.754362416107, 5565.216666666666, 6815.138440860215, 6723.689516129032, 6429.3735119047615, 6091.685060565276, 5504.175, 5465.200268817204, 5603.227777777778, 6121.515477792732, 6655.32123655914, 6621.998611111111, 6539.120805369127, 5821.1518817204305, 5403.49722222222], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price Nuclear Dates
0 64.949019 6665.969986 2015-01
1 56.383854 6681.123512 2015-02
2 55.522463 6687.913863 2015-03
3 58.354083 6068.169916 2015-04
4 57.294059 5403.817204 2015-05
5 65.974903 5659.276773 2015-06
6 71.072043 6483.623656 2015-07
7 63.998065 6437.845430 2015-08
8 60.254792 6466.175000 2015-09
9 59.406765 5854.012097 2015-10
10 60.726792 5973.075000 2015-11
11 61.901761 6626.029610 2015-12
12 45.578723 6223.849462 2016-01
13 36.752083 6159.264368 2016-02
14 36.818008 6699.888291 2016-03
15 32.618667 6706.406944 2016-04
16 34.691371 5714.009409 2016-05
17 46.266319 6647.463889 2016-06
18 47.502016 6628.922043 2016-07
19 47.602339 6633.349462 2016-08
20 50.405597 6675.755556 2016-09
21 60.182430 6576.685906 2016-10
22 62.581056 5534.297222 2016-11
23 67.595134 6325.970430 2016-12
24 79.492083 6769.916667 2017-01
25 59.837798 6739.267857 2017-02
26 50.959892 6755.951548 2017-03
27 51.717917 6676.383333 2017-04
28 53.772621 5561.240591 2017-05
29 56.258222 6063.859722 2017-06
30 55.252581 6117.403226 2017-07
31 54.084328 6675.846774 2017-08
32 55.816556 6427.688889 2017-09
33 63.925289 6003.754362 2017-10
34 65.430653 5565.216667 2017-11
35 65.151277 6815.138441 2017-12
36 56.511976 6723.689516 2018-01
37 60.877098 6429.373512 2018-02
38 48.279717 6091.685061 2018-03
39 50.400736 5504.175000 2018-04
40 61.633763 5465.200269 2018-05
41 64.348139 5603.227778 2018-06
42 67.783441 6121.515478 2018-07
43 70.363911 6655.321237 2018-08
44 76.914042 6621.998611 2018-09
45 70.362215 6539.120805 2018-10
46 67.042608 5821.151882 2018-11
47 66.623514 5403.497222 2018-12
[102.63388281869778, 118.49355831823476, 120.45420002671308, 103.98877970152685, 94.3172343770747, 85.77923627046935, 91.22607682648228, 100.59437701408982, 107.31387199496582, 98.54116929029787, 98.35979863363437, 107.04105229192045, 136.55164156780614, 167.589529876522, 181.97313328298446, 205.60027830008724, 164.7098182979096, 143.678251667955, 139.55033034817424, 139.3492345580292, 132.44075903166618, 109.27916930870813, 88.43406639744545, 93.58618021036858, 85.16466524444257, 112.62559996020988, 132.5738976103446, 129.09227137597384, 103.42141579325329, 107.78619520307156, 110.71705890251806, 123.43403396826045, 115.15739057905957, 93.91829892025801, 85.05519096023419, 104.60483304468198, 118.97813552223086, 105.61235177921166, 126.17482854931013, 109.20822640101433, 88.67218166973136, 87.07676514860786, 90.30989575192407, 94.58429917431938, 86.09609464822834, 92.93511904364472, 86.8276473194506, 81.1049568960575]
```

This is the logarithmic average monthly nuclear outputs versus the predictive average monthly prices of energy per EUR/MWH. The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
Lognuclearvalues = ((np.log(nuclear)))
Log = np.polyfit(np.log(Price_Actual), nuclear1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(nuclear1), Price_Actual)
Nuclear_Log = sm.OLS(Price_Actual, nuclear1).fit()

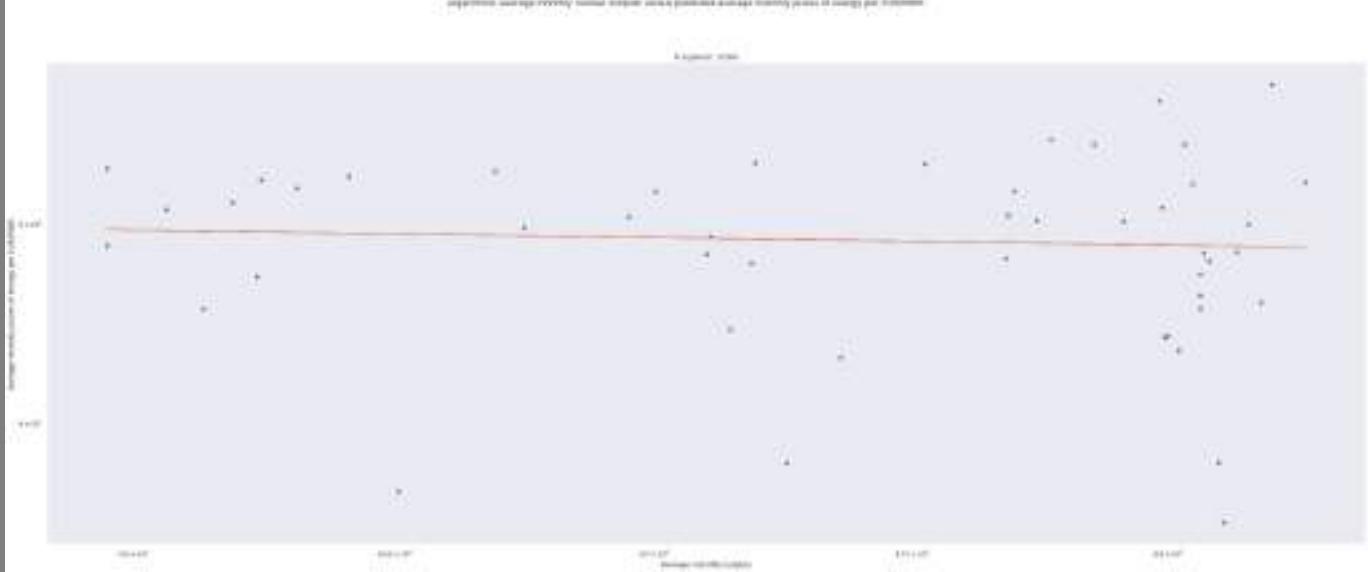
Nuclear_Logpred = Nuclear_Log.predict(nuclear1)
#OLS Logarithmic summary table
Nuclear_Log.summary()
#Log
Log = np.polyfit(np.log(nuclear), Price_Actual, 1)
print(Log)

y = -9.05428682 * Lognuclearvalues + 136.99359892

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly nuclear outputs versus predicted average monthly prices of energy per EUR/MWH")
plt.title("R squared : 0.004")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(Lognuclearvalues, Price_Actual, "o")
plt.plot(Lognuclearvalues, y)
```

```
[ -9.05428682 136.99359892]
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x7ff6085f21d0>]
```



This is a extremely weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH. The blue dots represent the observations and the orange line represents the model of best fit.

```
In [ ]: influenceNuclearLog = Nuclear_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsNuclearLog = influenceNuclearLog.resid_studentized_internal

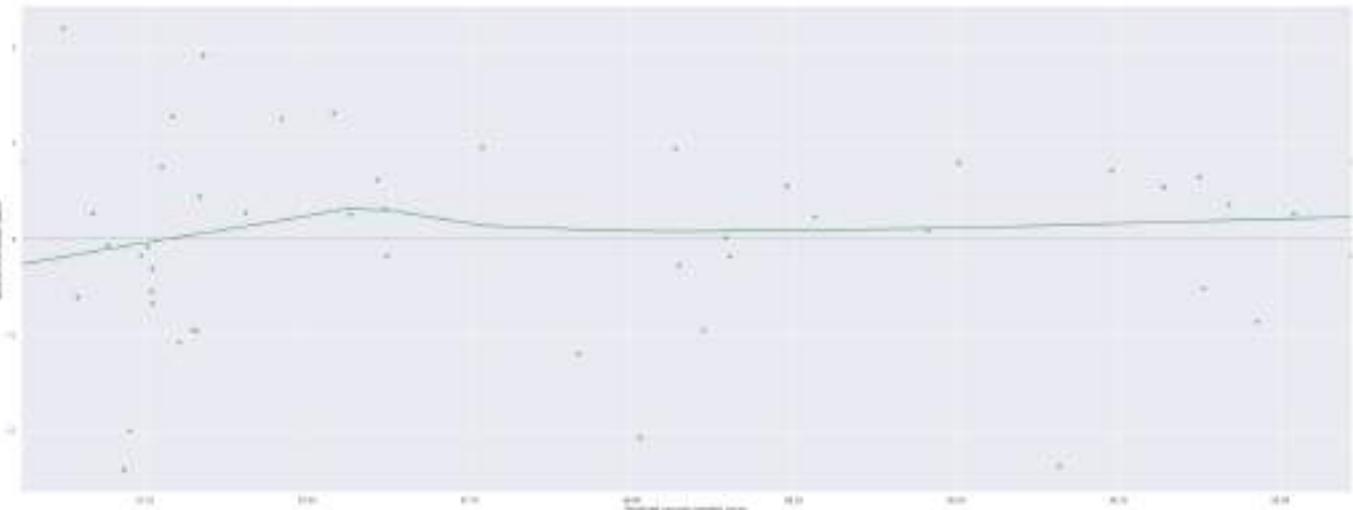
print(standardized_residualsNuclearLog)

print(Nuclear_Logpred) # OLS logarithmic predicted values
```

```
[ 0.75115225 -0.08525951 -0.16872773  0.02036138 -0.18348131  0.71618646
 1.31678601  0.62131206  0.26151717  0.09312165  0.23828751  0.4464321
-1.19796887 -2.06498968 -2.0009957 -2.41252445 -2.36415129 -1.07964728
-0.96070035 -0.95042766 -0.67150818  0.27090227  0.36562134  0.95392725
 2.20067334  0.26209229 -0.60814476 -0.54290636 -0.50916682 -0.18412391
-0.27411574 -0.31120499 -0.17554316  0.55400794  0.6531373  0.79818926
-0.06678203  0.31663023 -0.95579425 -0.85743  0.26251036  0.54922851
 0.94418256  1.27917314  1.91287366  1.25735764  0.8014866  0.79639179]
[57.27620225 57.25418559 57.24431984 58.14475009 59.10999271 58.73883375
57.54113449 57.60764599 57.56648576 58.45590145 58.28291414 57.3342319
57.9185622 58.01239833 57.22692211 57.21745112 58.65931232 57.30308989
57.33002946 57.32359684 57.26198473 57.40592371 58.92041739 57.77018995
57.12517741 57.16970727 57.14546742 57.26107263 58.88127119 58.1510124
58.07321867 57.2618522 57.6224025 58.23833989 58.87549432 57.05947438
57.19234115 57.6199549 58.11058477 58.96418218 59.02080883 58.82026772
58.06724395 57.29167389 57.34008857 57.4605023 59.11045761 58.50364429]
```

```
In [ ]: plt.suptitle("Predicted nuclear residuals from model versus predicted average monthly logarithmic outputs")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend(".#")
sns.residplot(x = Nuclear_Logpred, y = standardized_residualsNuclearLog, lowess = True, color="g")
# OLS Logarithmic average monthly predictions versus residuals
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608515ed0>
```



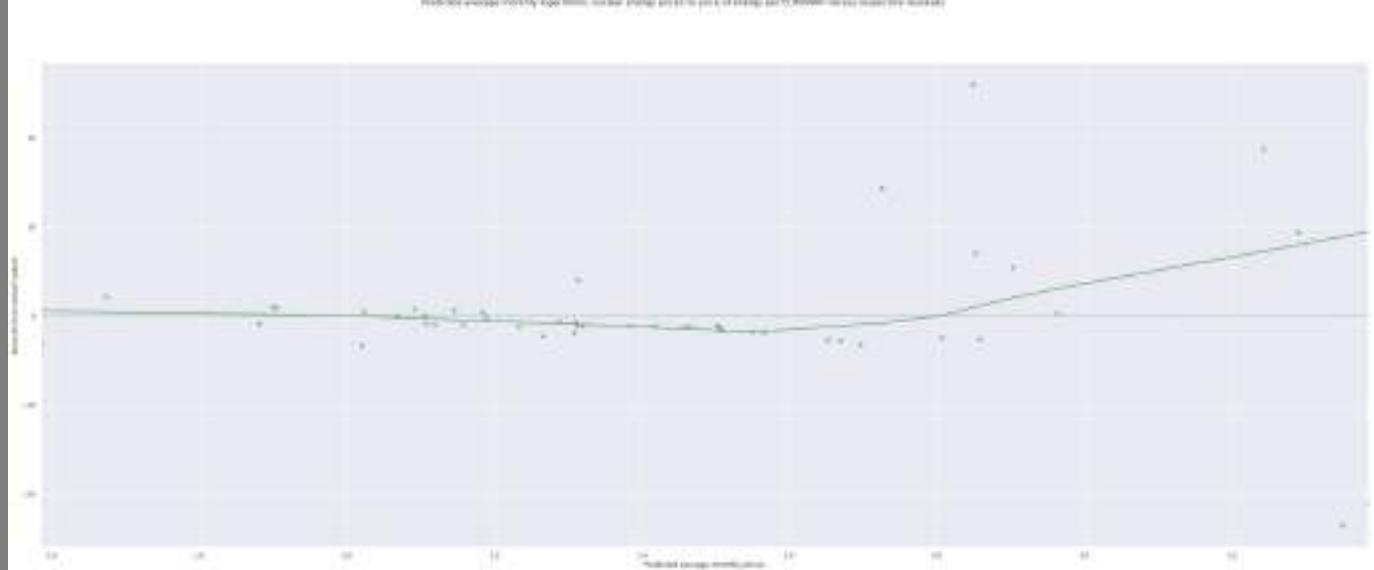
As one can observe this residual plot, one may notice an arching hump in the fitted model, which forms a nonlinear pattern. However, the observations are spread out without a distinct pattern, indicating constant variance, a lack of bias, and homoscedasticity. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: NuclearLogRatioPredict = Nuclear_Logpred/ypred
```

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
```

```
plt.suptitle("Predicted average monthly logarithmic nuclear energy prices to price of energy per EUR/MWh versus
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend(".#")
sns.residplot(x = NuclearLogRatioPredict, y = standardized_residualsNuclearLog/standardized_residualsPriceLog,
```

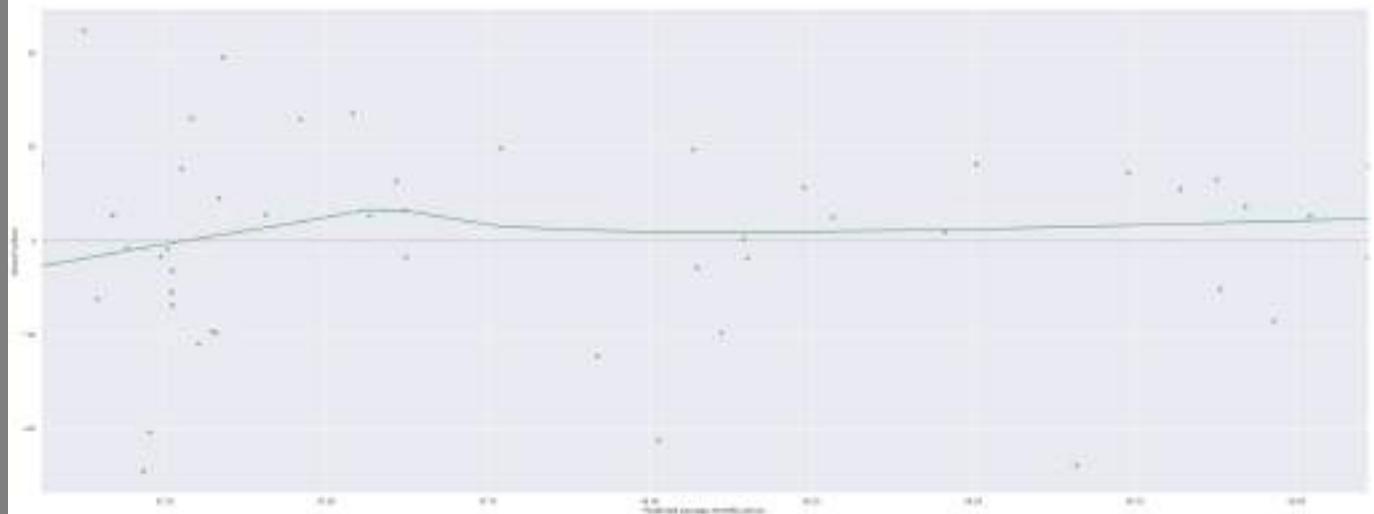
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60847f090>
```



With the exception of few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS predicted logarithmic average monthly values versus actual values
plt.suptitle("Predicted average monthly logarithmic nuclear energy prices per EUR/MWh versus actual values ")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Nuclear_Logpred, y = Price_Actual, lowess = True, color="g")
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608411b90>
```



```
In [ ]:
```

With the exception of few outliers, the predictions seem to be nearly the same as the actual values. This indicates homoscedasticity,

constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: influenceNuclearLog = Nuclear_Log.get_influence()  
#Logarithmic OLS regression residuals  
  
standardized_residualsNuclearLog = influenceNuclearLog.resid_studentized_internal  
  
print(standardized_residualsNuclearLog)  
  
[ 0.75115225 -0.08525951 -0.16872773  0.02036138 -0.18348131  0.71618646  
 1.31678601  0.62131206  0.26151717  0.09312165  0.23828751  0.4464321  
 -1.19796887 -2.06498968 -2.0009957 -2.41252445 -2.36415129 -1.07964728  
 -0.96070035 -0.95042766 -0.67150818  0.27090227  0.36562134  0.95392725  
 2.20067334  0.26209229 -0.60814476 -0.54290636 -0.50916682 -0.18412391  
 -0.27411574 -0.31120499 -0.17554316  0.55400794  0.6531373  0.79818926  
 -0.06678203  0.31663023 -0.95579425 -0.85743  0.26251036  0.54922851  
 0.94418256  1.27917314  1.91287366  1.25735764  0.8014866  0.79639179]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly nuclear output versus the average monthly prices of energy per EUR/MWH.

```
In [ ]:  
  
In [ ]: modelnuclear = stats.linregress( [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5.  
 55.522462987886975,  
 58.35408333333333,  
 57.29405913978498,  
 65.9749027777778,  
 71.07204301075271,  
 63.99806451612899,  
 60.254791666666634,  
 59.40676510067113,  
 60.72679166666668,  
 61.901760752688226,  
 45.57872311827956,  
 36.752083333333374,  
 36.81800807537014,  
 32.61866666666666,  
 34.691370967741896,  
 46.266319444444434,  
 47.50201612903221,  
 47.6023387096774,  
 50.40559722222224,  
 60.182429530201404,  
 62.58105555555558,  
 67.5951344086021,  
 79.4920833333331,  
 59.83779761904767,  
 50.95989232839837,  
 51.71791666666662,  
 53.77262096774189,  
 56.2582222222224,  
 55.252580645161316,  
 54.08432795698931,  
 55.81655555555558,  
 63.92528859060403,  
 65.43065277777781,  
 65.15127688172035,  
 56.51197580645163,  
 60.877098214285674,  
 48.279717362045766,  
 50.4007361111113,  
 61.633763440860214,  
 64.3481388888884,  
 67.78344086021498,  
 70.36391129032262,  
 76.9140416666666,  
 70.36221476510062,  
 67.0426075268817,  
 66.6235138888881] )
```

```
In [ ]: #Linear OLS regression  
nuclear1 = nuclear  
  
nuclear1 = sm.add_constant(nuclear1)
```

```

modelnuclearreg = sm.OLS(Price_Actual, nuclear1).fit()
predictionsnuclear = modelnuclearreg.predict(nuclear1)

modelnuclearreg.summary()
#OLS Linear Summary Table

```

Out[]:

OLS Regression Results					
Dep. Variable:	y	R-squared:	0.004		
Model:	OLS	Adj. R-squared:	-0.018		
Method:	Least Squares	F-statistic:	0.1913		
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.664		
Time:	05:25:04	Log-Likelihood:	-179.54		
No. Observations:	48	AIC:	363.1		
Df Residuals:	46	BIC:	366.8		
Df Model:	1				
Covariance Type:	nonrobust				
coef	std err	t	P> t	[0.025	0.975]
const	66.9612	20.862	3.210	0.002	24.969 108.953
x1	-0.0015	0.003	-0.437	0.664	-0.008 0.005
Omnibus:	2.625	Durbin-Watson:	0.442		
Prob(Omnibus):	0.269	Jarque-Bera (JB):	1.819		
Skew:	-0.461	Prob(JB):	0.403		
Kurtosis:	3.247	Cond. No.	8.72e+04		

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 8.72e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

In []:

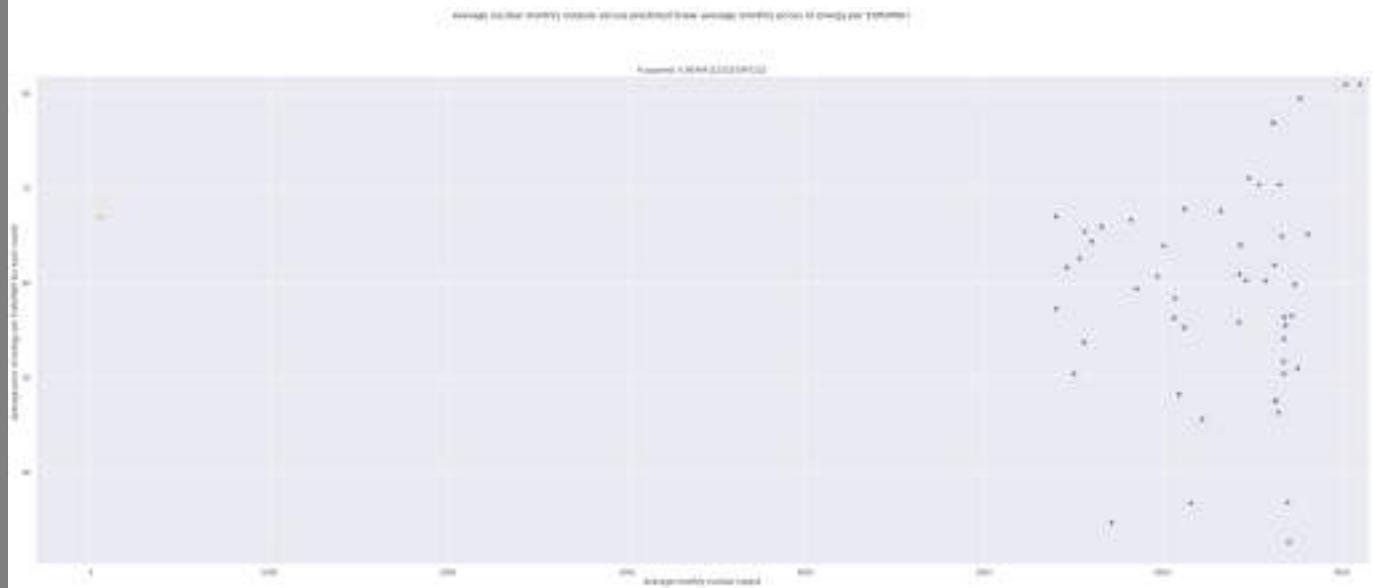
```

#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(nuclear,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(nuclear,Price_Actual, "o")
plt.title(f"R squared: {modelnuclear.rvalue**2}")
f = lambda x: -0.001453 *x + 66.961236
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average nuclear monthly outputs versus predicted linear average monthly prices of energy per EUR/I")
plt.legend("#")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly nuclear output')
plt.show()

```

slope: -0.001453 intercept: 66.961236



There is a very weak yet positive correlation between the output and their average monthly price per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: #Linear OLS regression residuals
influenceNuclearreg = modelnuclearreg.get_influence()

standardized_residualsNuclear = influenceNuclearreg.resid_studentized_internal

print(standardized_residualsNuclear)
```

```
[ 0.75115225 -0.08525951 -0.16872773  0.02036138 -0.18348131  0.71618646
 1.31678601  0.62131206  0.26151717  0.09312165  0.23828751  0.4464321
-1.19796887 -2.06498968 -2.0009957 -2.41252445 -2.36415129 -1.07964728
-0.96070035 -0.95042766 -0.67150818  0.27090227  0.36562134  0.95392725
 2.20067334  0.26209229 -0.60814476 -0.54290636 -0.50916682 -0.18412391
-0.27411574 -0.31120499 -0.17554316  0.55400794  0.6531373  0.79818926
-0.06678203  0.31663023 -0.95579425 -0.85743  0.26251036  0.54922851
 0.94418256  1.27917314  1.91287366  1.25735764  0.8014866  0.79639179]
```

```
In [ ]: print(predictionsnuclear)
#Linear OLS Predicted Values
```

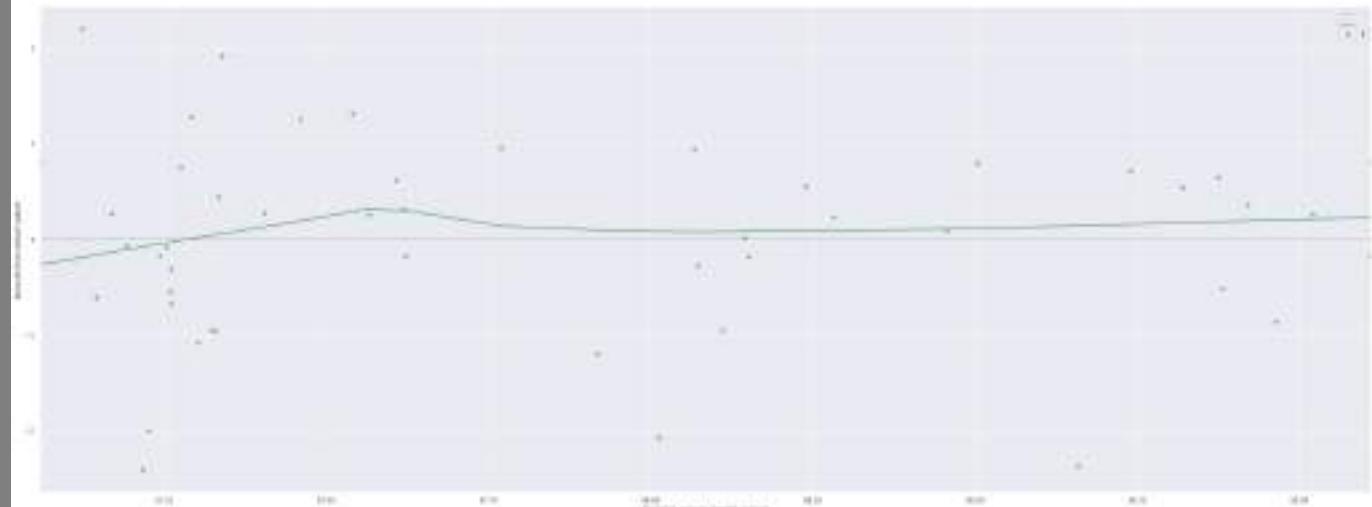
```
[57.27620225 57.25418559 57.24431984 58.14475009 59.10999271 58.73883375
57.54113449 57.60764599 57.56648576 58.45590145 58.28291414 57.3342319
57.9185622 58.01239833 57.22692211 57.21745112 58.65931232 57.30308989
57.33002946 57.32359684 57.26198473 57.40592371 58.92041739 57.77018995
57.12517741 57.16970727 57.14546742 57.26107263 58.88127119 58.1510124
58.07321867 57.2618522 57.6224025 58.23833989 58.87549432 57.05947438
57.19234115 57.6199549 58.11058477 58.96418218 59.02080883 58.82026772
58.06724395 57.29167389 57.34008857 57.4605023 59.11045761 58.50364429]
```

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionsnuclear, y = standardized_residualsNuclear, lowess = True, color ="g")

plt.suptitle("Nuclear residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
```

```
plt.legend(".,#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60836d950>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]:
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly nuclear output versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

```
#Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)

modelNuclearquad = np.poly1d(np.polyfit(nuclear,Price_Actual,2))
print(modelNuclearquad)

nuclear1 = nuclear

nuclear1 = sm.add_constant(nuclear1)
nuclear2 = polynomial_features.fit_transform(nuclear1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(nuclear1)

Nuclear_Q = poly.fit(X_poly, Price_Actual)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
Nuclear_Quad = sm.OLS(Price_Actual, nuclear2).fit()

# OLS Predicted Quadratic values
Nuclear_ypred = Nuclear_Quad.predict(nuclear2)

#OLS Quadratic Summary Table
Nuclear_Quad.summary()
```

$$2 \\ 2.973e-06 x^2 - 0.03791 x + 178.1$$

Out[]:

OLS Regression Results									
Dep. Variable:	y	R-squared:	0.006						
Model:	OLS	Adj. R-squared:	-0.038						
Method:	Least Squares	F-statistic:	0.1378						
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.872						
Time:	05:25:06	Log-Likelihood:	-179.50						
No. Observations:	48	AIC:	365.0						
Df Residuals:	45	BIC:	370.6						
Df Model:	2								
Covariance Type:	nonrobust								
	coef	std err	t	P> t	[0.025	0.975]			
const	59.3574	124.947	0.475	0.637	-192.299	311.013			
x1	59.3574	124.947	0.475	0.637	-192.299	311.013			
x2	-0.0190	0.061	-0.309	0.759	-0.143	0.105			
x3	59.3574	124.947	0.475	0.637	-192.299	311.013			
x4	-0.0190	0.061	-0.309	0.759	-0.143	0.105			
x5	2.973e-06	1e-05	0.297	0.768	-1.72e-05	2.31e-05			
Omnibus:	2.608	Durbin-Watson:	0.445						
Prob(Omnibus):	0.271	Jarque-Bera (JB):	1.838						
Skew:	-0.467	Prob(JB):	0.399						
Kurtosis:	3.214	Cond. No.	8.29e+24						

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 1.11e-33. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
##OLS Quadratic Scatterplot
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelNuclearquad(polyline))
plt.scatter(nuclear,Price_Actual, color = 'blue')
plt.title("R squared : 0.006")
plt.suptitle('Quadratic for average monthly outputs of nuclear power versus average predicted monthly prices of nuclear power')
plt.xlabel('Average monthly outputs of nuclear power ')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()

influenceNuclearquad = Nuclear_Quad.get_influence() #Quadratic OLS residuals

standardized_residualsNuclearQuad = influenceNuclearquad.resid_studentized_internal

print(standardized_residualsNuclearQuad)
```



```
[ 0.72684891 -0.1076632 -0.19284055  0.08449366 -0.28717678  0.70872737
 1.33790549  0.65696554  0.29218058  0.13533639  0.29789044  0.43574267
-1.14954956 -2.02774465 -2.01972351 -2.43309034 -2.32930507 -1.0824437
-0.9585776 -0.9497852 -0.68781518  0.27525474  0.32525038  1.01324839
 2.16064282  0.2192945 -0.659336 -0.56039246 -0.53723017 -0.12273322
-0.2128188 -0.33021616 -0.13642431  0.61985247  0.62060693  0.74071334
-0.10393653  0.35466901 -0.90381765 -0.91286781  0.19913159  0.5278617
 1.02195212  1.25308323  1.8889094   1.26285085  0.7378867  0.83095937]
```

The blue dots represent the observations and the orange line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWh. The blue line represents the quadratic model of fit while the blue dots represent the observations.

```
In [ ]: print(Nuclear_ypred) # OLS quadratic predicted values
```

```
[57.46383591 57.49062459 57.50307165 57.49629998 60.02502679 58.74215575
57.2485375 57.22553089 57.23829895 58.02484349 57.69733745 57.39976991
57.28323765 57.35413914 57.52568934 57.53836035 58.51776784 57.43297221
57.40409094 57.41080144 57.48097887 57.33371711 59.32130337 57.22173702
57.67503201 57.60608244 57.64292255 57.48209808 59.18859728 57.50424883
57.41334049 57.48114135 57.22211535 57.62660436 59.1693791 57.78696521
57.57317714 57.22263945 57.45487853 59.47477637 59.68135856 58.9903984
57.40706335 57.44582764 57.39383093 57.29313654 60.02687698 58.13007237]
```

```
In [ ]: print(standardized_residualsNuclearQuad)#Quadratic OLS residuals
```

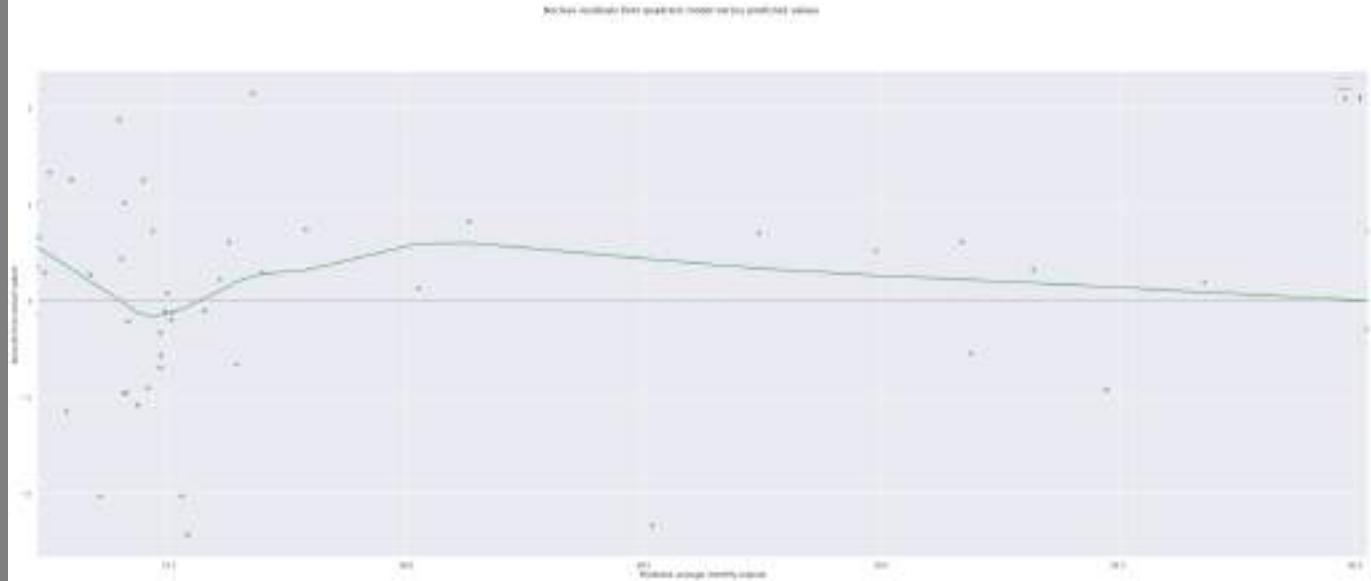
```
[ 0.72684891 -0.1076632 -0.19284055  0.08449366 -0.28717678  0.70872737
 1.33790549  0.65696554  0.29218058  0.13533639  0.29789044  0.43574267
-1.14954956 -2.02774465 -2.01972351 -2.43309034 -2.32930507 -1.0824437
-0.9585776 -0.9497852 -0.68781518  0.27525474  0.32525038  1.01324839
 2.16064282  0.2192945 -0.659336 -0.56039246 -0.53723017 -0.12273322
-0.2128188 -0.33021616 -0.13642431  0.61985247  0.62060693  0.74071334
-0.10393653  0.35466901 -0.90381765 -0.91286781  0.19913159  0.5278617
 1.02195212  1.25308323  1.8889094   1.26285085  0.7378867  0.83095937]
```

```
In [ ]:
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = Nuclear_ypred, y = standardized_residualsNuclearQuad, lowess = True, color="g")

plt.suptitle("Nuclear residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend("...#")
```

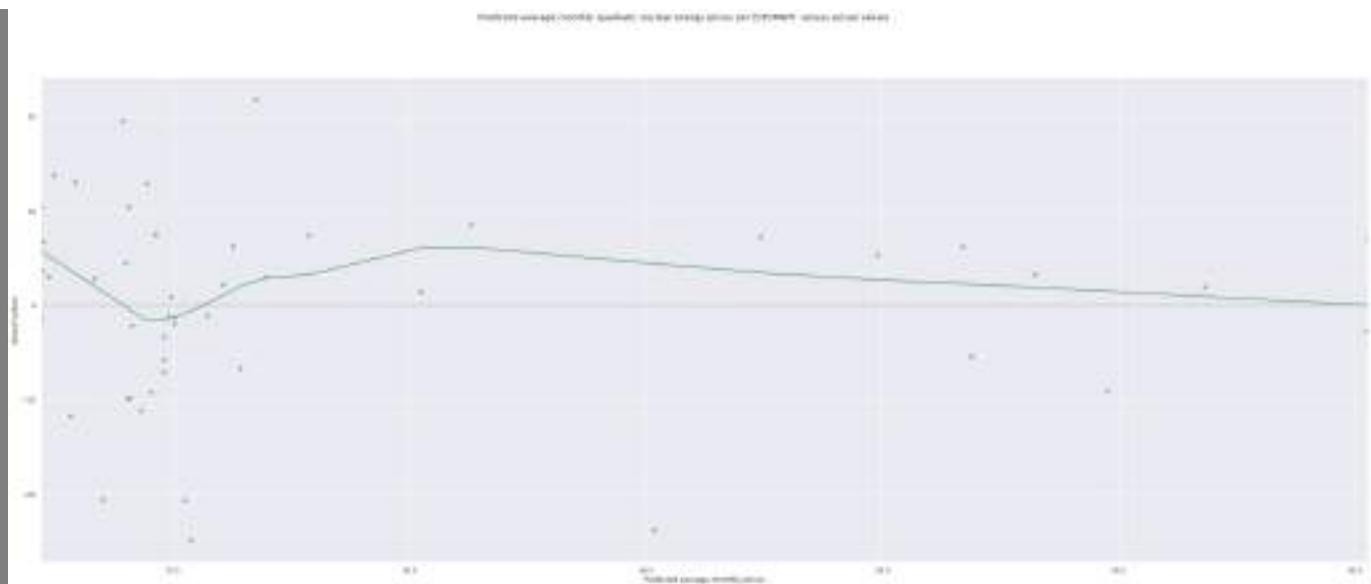
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff608258d10>
```



As one can observe this residual plot, one may notice that the lowess line has a slight hump and that the residuals are clustered on the left side; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly quadratic nuclear energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Nuclear_ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6081a52d0>



As one can observe this residual plot, one may notice that the lowess line has a slight hump and that the observations are clustered on the left side; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

In []:

```

#Dataframes analyzed by resource
dfnuclear = ({ "Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.7520833333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 45.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.581055555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63375344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.62351388888891], 'Nuclear': [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.7520833333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 45.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.581055555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63375344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.62351388888891], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}])

```

```

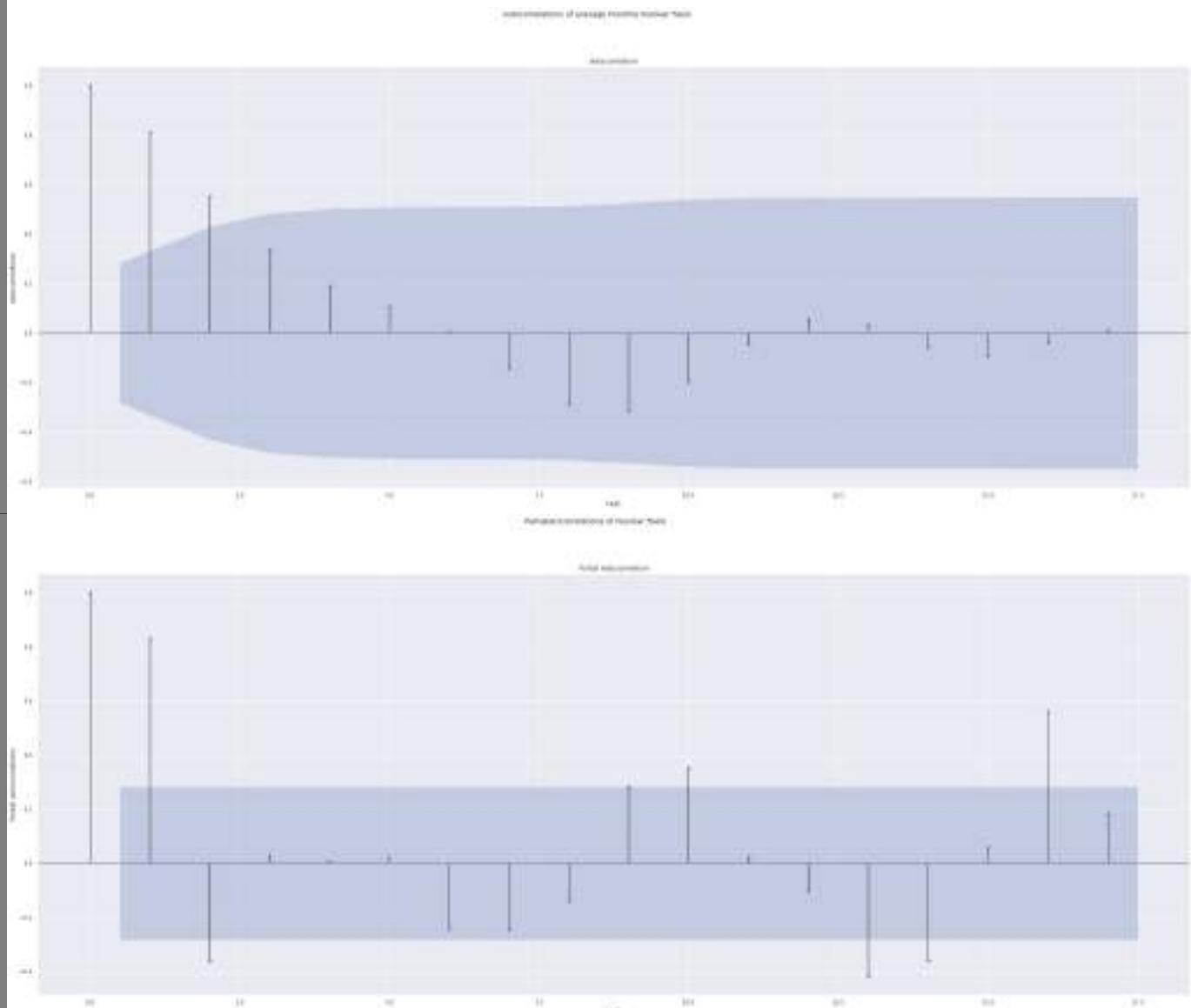
4 57.294059 5403.817204 2015-05
5 65.974903 5659.276773 2015-06
6 71.072043 6483.623656 2015-07
7 63.998065 6437.845430 2015-08
8 60.254792 6466.175000 2015-09
9 59.406765 5854.012097 2015-10
10 60.726792 5973.075000 2015-11
11 61.901761 6626.029610 2015-12
12 45.578723 6223.849462 2016-01
13 36.752083 6159.264368 2016-02
14 36.818008 6699.888291 2016-03
15 32.618667 6706.406944 2016-04
16 34.691371 5714.009409 2016-05
17 46.266319 6647.463889 2016-06
18 47.502016 6628.922043 2016-07
19 47.602339 6633.349462 2016-08
20 50.405597 6675.755556 2016-09
21 60.182430 6576.685906 2016-10
22 62.581056 5534.297222 2016-11
23 67.595134 6325.970430 2016-12
24 79.492083 6769.916667 2017-01
25 59.837798 6739.267857 2017-02
26 50.959892 6755.951548 2017-03
27 51.717917 6676.383333 2017-04
28 53.772621 5561.240591 2017-05
29 56.258222 6063.859722 2017-06
30 55.252581 6117.403226 2017-07
31 54.084328 6675.846774 2017-08
32 55.816556 6427.688889 2017-09
33 63.925289 6003.754362 2017-10
34 65.430653 5565.216667 2017-11
35 65.151277 6815.138441 2017-12
36 56.511976 6723.689516 2018-01
37 60.877098 6429.373512 2018-02
38 48.279717 6091.685061 2018-03
39 50.400736 5504.175000 2018-04
40 61.633763 5465.200269 2018-05
41 64.348139 5603.227778 2018-06
42 67.783441 6121.515478 2018-07
43 70.363911 6655.321237 2018-08
44 76.914042 6621.998611 2018-09
45 70.362215 6539.120805 2018-10
46 67.042608 5821.151882 2018-11
47 66.623514 5403.497222 2018-12
[102.63388281689778, 118.49355831823476, 120.45420002671308, 103.98877970152685, 94.3172343770747, 85.7792362704
6935, 91.22607682648228, 100.59437701408982, 107.31387199496582, 98.54116929029787, 98.35979863363437, 107.04105
229192045, 136.55164156780614, 167.589529876522, 181.97313328298446, 205.60027830008724, 164.7098182979096, 143.
678251667955, 139.55033034817424, 139.3492345580292, 132.44075903166618, 109.27916930870813, 88.43406639744545,
93.58618021036858, 85.16466524444257, 112.62559996020988, 132.5738976103446, 129.09227137597384, 103.42141579325
329, 107.78619520307156, 110.71705890251806, 123.43403396826045, 115.15739057905957, 93.91829892025801, 85.05519
096023419, 104.60483304468198, 118.97813552223086, 105.61235177921166, 126.17482854931013, 109.20822640101433, 8
8.67218166973136, 87.07676514860786, 90.30989575192407, 94.58429917431938, 86.09609464822834, 92.93511904364472,
86.8276473194506, 81.1049568960575]
ADF Test Statistic : -2.6425263674397157
p-value : 0.0845225356075241
#Lags Used : 1
Number of Observations : 46
weak evidence against null hypothesis, indicating it is non-stationary
[ 1.          0.81265419  0.55260299  0.33657777  0.19028745  0.10694041
  0.00590405 -0.14063192 -0.28701972 -0.31525217 -0.197297 -0.0460034
  0.05461874  0.03504196 -0.05475313 -0.09408892 -0.03773365  0.0114265
  0.00632231 -0.04397201 -0.09851278 -0.09167291 -0.04177114  0.01603304
 -0.02322478 -0.11452347 -0.21504476 -0.25560178 -0.22892719 -0.20472481
 -0.18115844 -0.19196898 -0.17731518 -0.09464158 -0.01492975  0.04689275
  0.08250662  0.08862852  0.06315118  0.0534486  0.05902948]

```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partialautocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation plot and the partialautocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly nuclear outputs and the average monthly prices of energy per EUR/MWH.

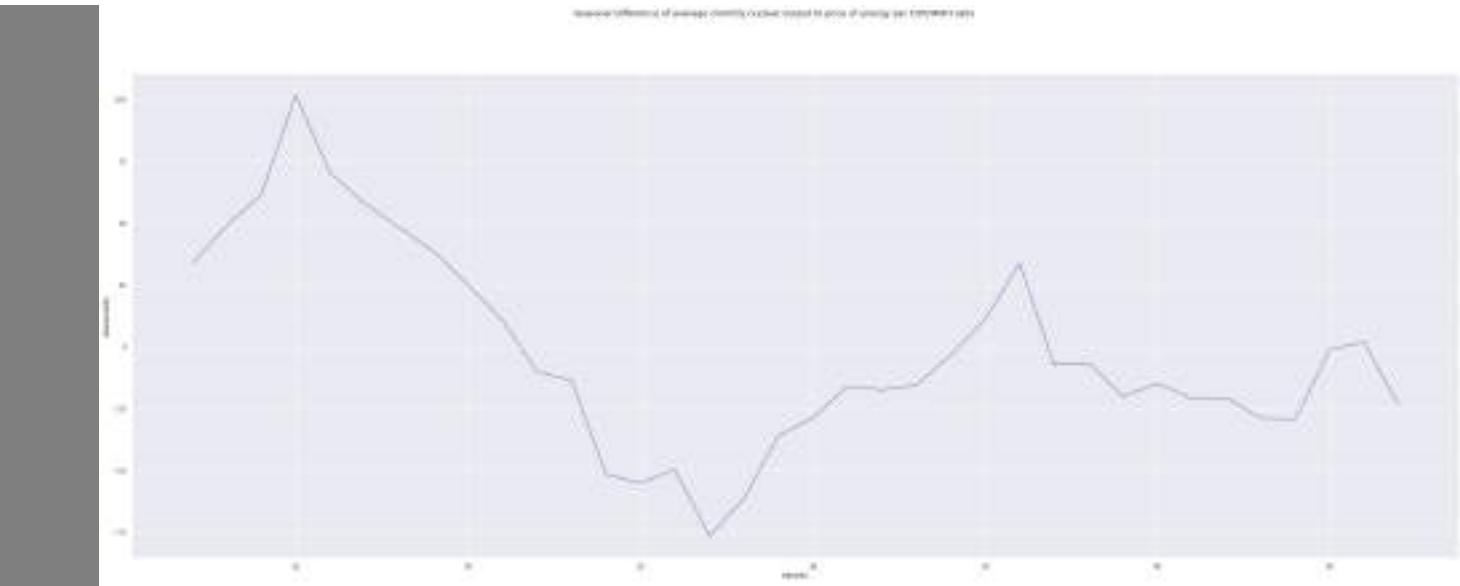
```
In [ ]: df_nuclear.describe(include = 'all') # Description Tables
```

	Price	Nuclear	Dates	Ratio	First Difference	Nuclear Ratio	Seasonal Difference	Nuclear Ratio
count	48.000000	48.000000	48	48.000000		47.000000	36.000000	
unique	NaN	NaN	48	NaN		NaN	NaN	
top	NaN	NaN	2015-01	NaN		NaN	NaN	
freq	NaN	NaN	1	NaN		NaN	NaN	
mean	57.859848	6264.260822	NaN	112.554520		-0.458062	-1.698965	
std	10.320573	457.169177	NaN	26.825833		15.863282	40.147662	
min	32.618667	5403.497222	NaN	81.104957		-40.890460	-76.508007	
25%	51.528411	5943.309274	NaN	93.423415		-8.817905	-24.833225	
50%	59.622281	6433.609471	NaN	106.326702		-0.201096	-11.690302	
75%	64.999583	6668.416379	NaN	124.119233		9.024777	27.298533	
max	79.492083	6815.138441	NaN	205.600278		31.037888	101.611499	

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly nuclear output to price of energy per EUR/MWH ratio")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_nuclear['Seasonal Difference Nuclear Ratio'].plot() # Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608174550>



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly nuclear megawatts and the average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
nuclearResults_mean = np.mean(df_nuclear["Ratio"])
nuclearResults_std = np.std(df_nuclear["Ratio"])

nuclearResultspdf = stats.norm.pdf(df_nuclear["Ratio"].sort_values(), nuclearResults_mean, nuclearResults_std)

plt.plot(df_nuclear["Ratio"].sort_values(), nuclearResultspdf)
plt.xlim([0,300])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_nuclear["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
plt.suptitle("Frequency distribution of average monthly nuclear outputs to energy prices per EUR/MWH ratios (scaled)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

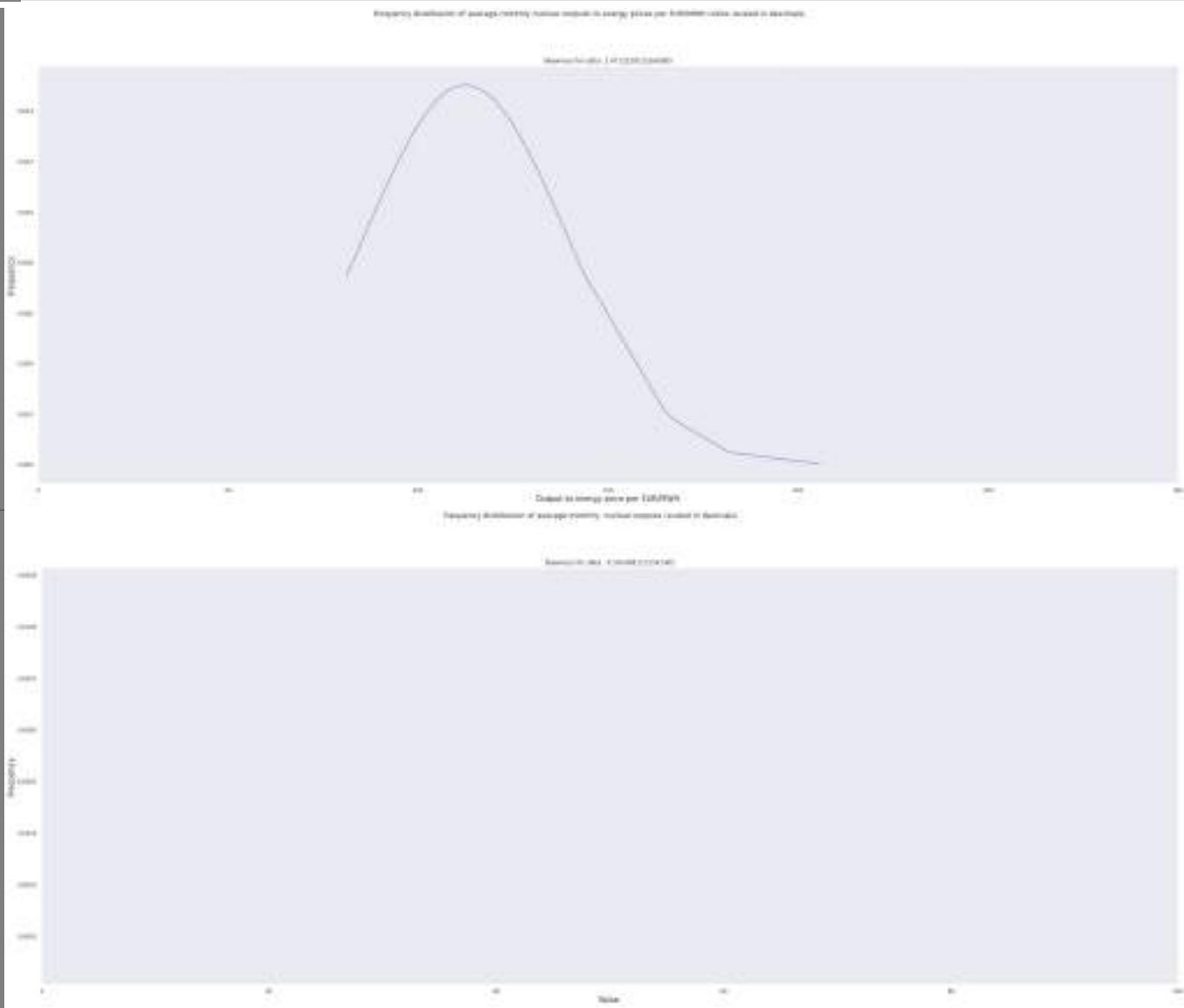
#Bell Curves
nuclearResults_mean = np.mean(df_nuclear["Nuclear"])
nuclearResults_std = np.std(df_nuclear["Nuclear"])

nuclearResultspdf = stats.norm.pdf(df_nuclear["Nuclear"].sort_values(), nuclearResults_mean, nuclearResults_std)
```

```

plt.plot(df_nuclear["Nuclear"].sort_values(), nuclearResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_nuclear["Nuclear"])}')
plt.suptitle("Frequency distribution of average monthly nuclear outputs (scaled in decimals) " )
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

```



These bell shaped curves are respectively skewed to the right and left. Hence they have an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```

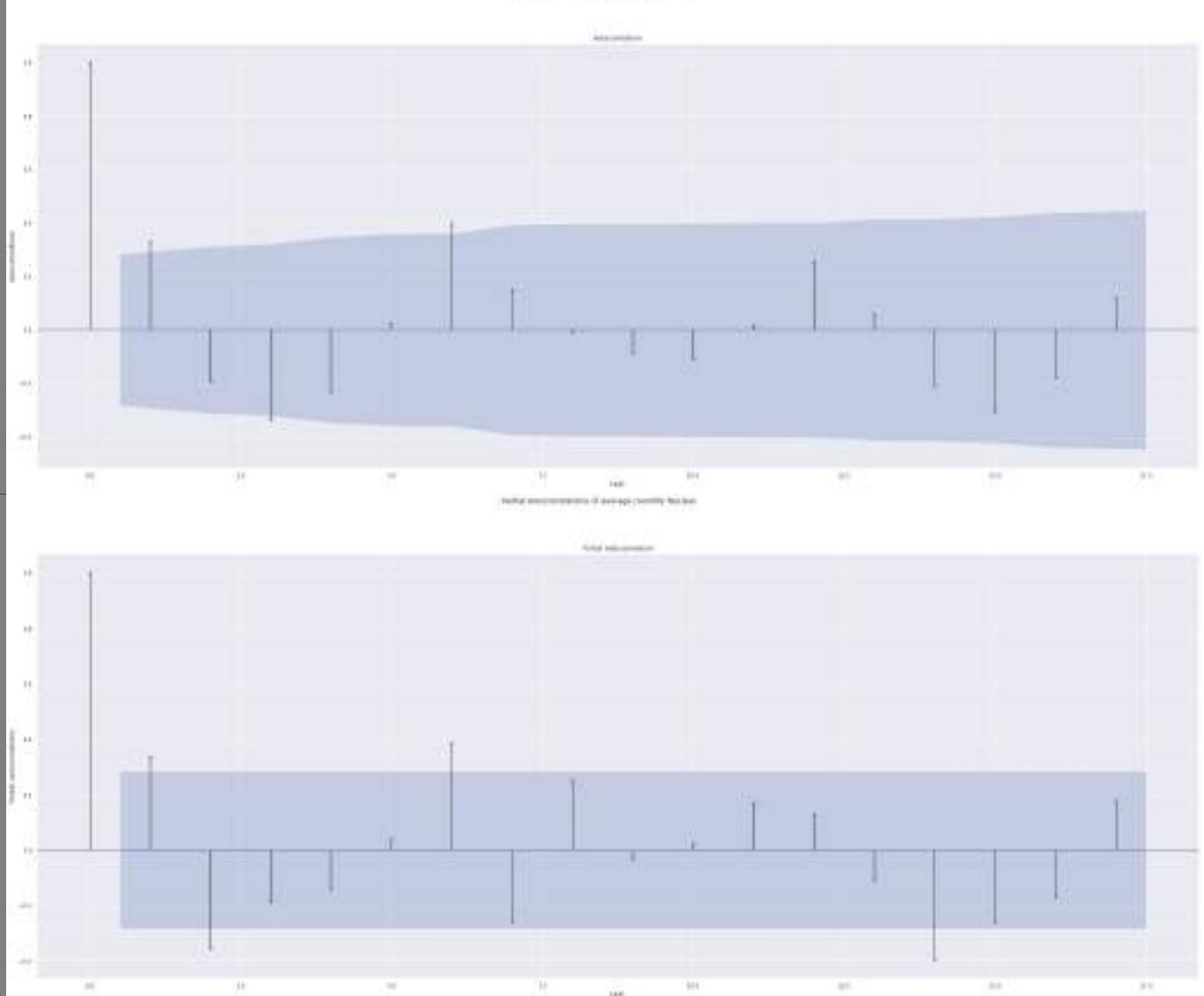
In [ ]:
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_nuclear["Nuclear"])
plt.suptitle(" Autocorrelations of average monthly Nuclear")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_nuclear["Nuclear"])
plt.suptitle("Partial Autocorrelations of average monthly Nuclear")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show

```

Out[]: <function matplotlib.pyplot.show(*args, **kw)>



```
In [ ]: Nuclear_Autocorrelations = sm.tsa.acf(df_nuclear["Nuclear"], fft=False) #Autocorrelations
print(Nuclear_Autocorrelations)
```

```
[ 1.          0.32972859 -0.19245024 -0.33198664 -0.23036835  0.02342246
 0.39790852  0.14999745 -0.00843854 -0.08772573 -0.10680284  0.01429885
 0.25340068  0.05810637 -0.21027822 -0.30252135 -0.17668032  0.1196931
 0.31799238  0.10713671 -0.17085292 -0.23488514 -0.26976576 -0.04748443
 0.22107386  0.18750055 -0.10250962 -0.17661859 -0.11276306  0.06311024
 0.10761525  0.01768954 -0.14620152 -0.10126917  0.03498751  0.12326906
 0.12526003  0.11401715 -0.03247388 -0.16632345 -0.14806052]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the first two lags on the autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly nuclear outputs and the average monthly prices of energy per EUR/MWH.

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective average monthly prices of energy per EUR/MWH.

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_nuclear["Nuclear"])
```

```
ADF Test Statistic : -1.791924835761476
p-value : 0.38447809437018643
#Lags Used : 5
Number of Observations : 42
weak evidence against null hypothesis,indicating it is non-stationary
```

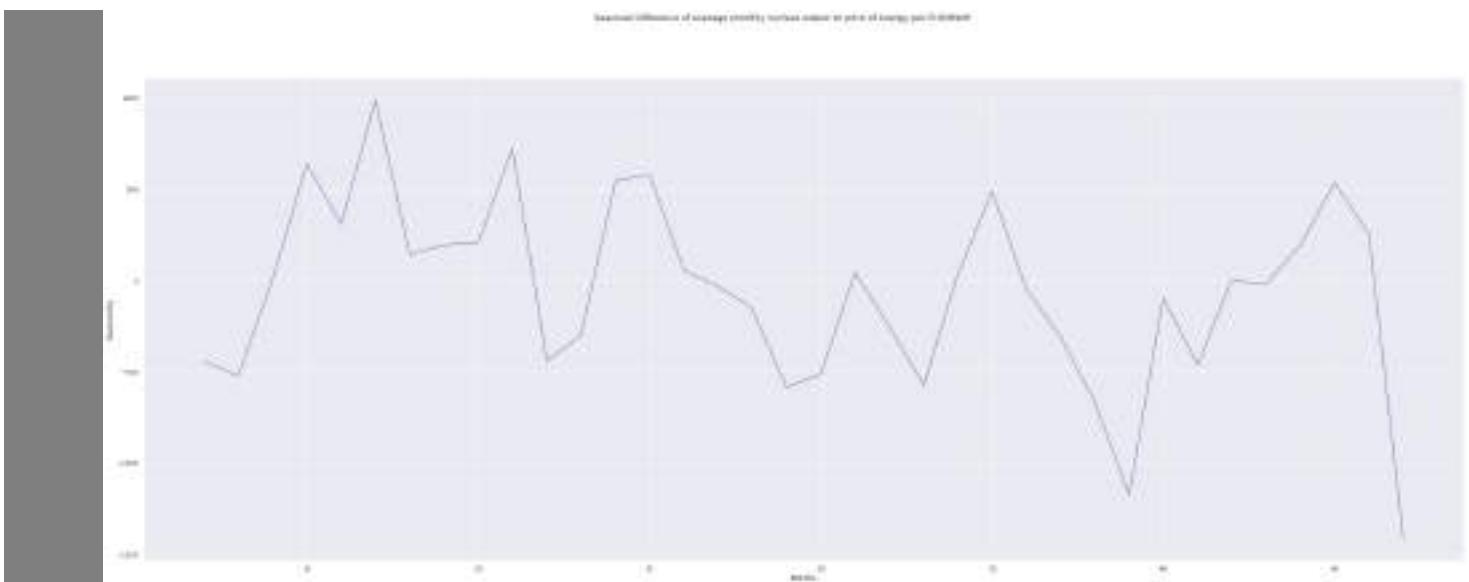
```
In [ ]: df_nuclear['First Difference Nuclear'] = df_nuclear["Nuclear"]- df_nuclear["Nuclear"].shift(1) # Seasonality variable
df_nuclear['Seasonal Difference Nuclear']=df_nuclear["Nuclear"]- df_nuclear["Nuclear"].shift(12)
df_nuclear.head()
```

	Price	Nuclear	Dates	Ratio	First Difference Nuclear Ratio	Seasonal Difference Nuclear Ratio	First Difference Nuclear	Seasonal Difference Nuclear
0	64.949019	6665.969986	2015-01	102.633883	NaN	NaN	NaN	NaN
1	56.383854	6681.123512	2015-02	118.493558	15.859676	NaN	15.153526	NaN
2	55.522463	6687.913863	2015-03	120.454200	1.960642	NaN	6.790351	NaN
3	58.354083	6068.169916	2015-04	103.988780	-16.465420	NaN	-619.743946	NaN
4	57.294059	5403.817204	2015-05	94.317234	-9.671545	NaN	-664.352712	NaN

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly nuclear output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_nuclear['Seasonal Difference Nuclear'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff607ee8450>
```



```
In [ ]:
```

The blue line represents the trend line among the values themselves. As one can observe, there are obvious patterns depicted in the average monthly nuclear output.

```
In [ ]: df_nuclear.describe(include = 'all') # Description Tables
```

Out[]:

	Price	Nuclear	Dates	Ratio	First Difference Nuclear Ratio	Seasonal Difference Nuclear Ratio	First Difference Nuclear	Seasonal Difference Nuclear
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	6264.260822	NaN	112.554520	-0.458062	-1.698965	-26.861123	-56.307658
std	10.320573	457.169177	NaN	26.825833	15.863282	40.147662	515.671510	511.467587
min	32.618667	5403.497222	NaN	81.104957	-40.890460	-76.508007	-1115.142742	-1411.641219
25%	51.528411	5943.309274	NaN	93.423415	-8.817905	-24.833225	-369.934299	-439.613464
50%	59.622281	6433.609471	NaN	106.326702	-0.201096	-11.690302	-30.648810	-8.206643
75%	64.999583	6668.416379	NaN	124.119233	9.024777	27.298533	196.743539	221.169220
max	79.492083	6815.138441	NaN	205.600278	31.037888	101.611499	1249.921774	988.187116

Below is the box and whisker plot for the distribution of the average monthly outputs of nuclear power and its the average monthly prices of energy per EUR/MWH.

In []:

```
# Box and Whisker Plot
sns.set(style="darkgrid")

plt.suptitle('Distribution of average monthly nuclear power outputs by average monthly price of energy per EUR/I')
plt.ylabel('Average monthly nuclear power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')
sns.boxplot(x=Rounded_Y, y=[6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.81])
plt.show()
```



This box and whisker plot depicts the frequencies between the distribution of the monthly average output of nuclear power produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of nuclear power produced, which was inbetween 6500 and 7000 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), nuclear power output was at roughly 6700 units. At approximately 65.15 EUR/MWH, nuclear power produced its highest output, whichughly 6800 units.The lowest amount produced, whcih was roughly 5000 units, was at the prices of approximately 57.29 EUR/MWH and 66.62 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.49 EUR/MWH, the output was slightly below 6600 units.

In []:

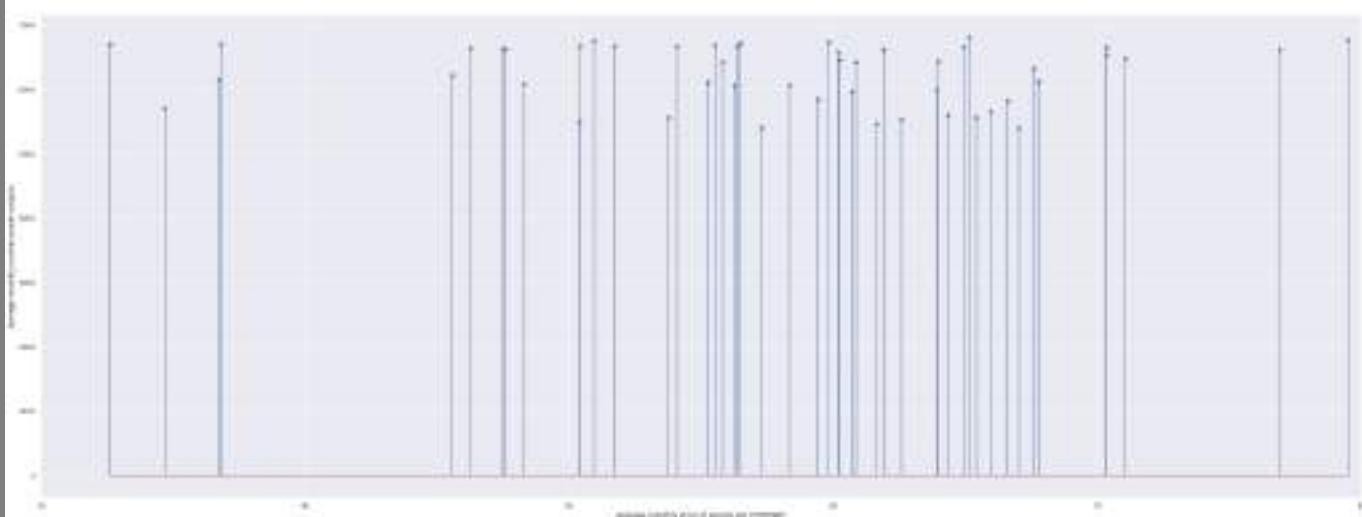
```
plt.suptitle('Distribution of average monthly nuclear power outputs by average monthly price of energy per EUR/I')
plt.ylabel('Average monthly nuclear power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

plt.xlim(30, 80)

# Stem Plot
plt.stem(Rounded_Y, nuclear)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.
```

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of nuclear power produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of nuclear power produced, which was inbetween 6500 and 7000 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), nuclear power output was at roughly 6700 units. At approximately 65.15 EUR/MWH, nuclear power produced its highest output, which was roughly 6800 units.The lowest amount produced, whcih was roughly 5000 units, was at the prices of approximately 57.29 EUR/MWH and 66.62 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.49 EUR/MWH, the output was slightly below 6600 units. Each blue dot at the end of the blue line represents an observation.

```
In [ ]: print(dicDates)
pdToListNuclear_Dict = {key: i for i, key in enumerate(pdToListNuclear)}
```

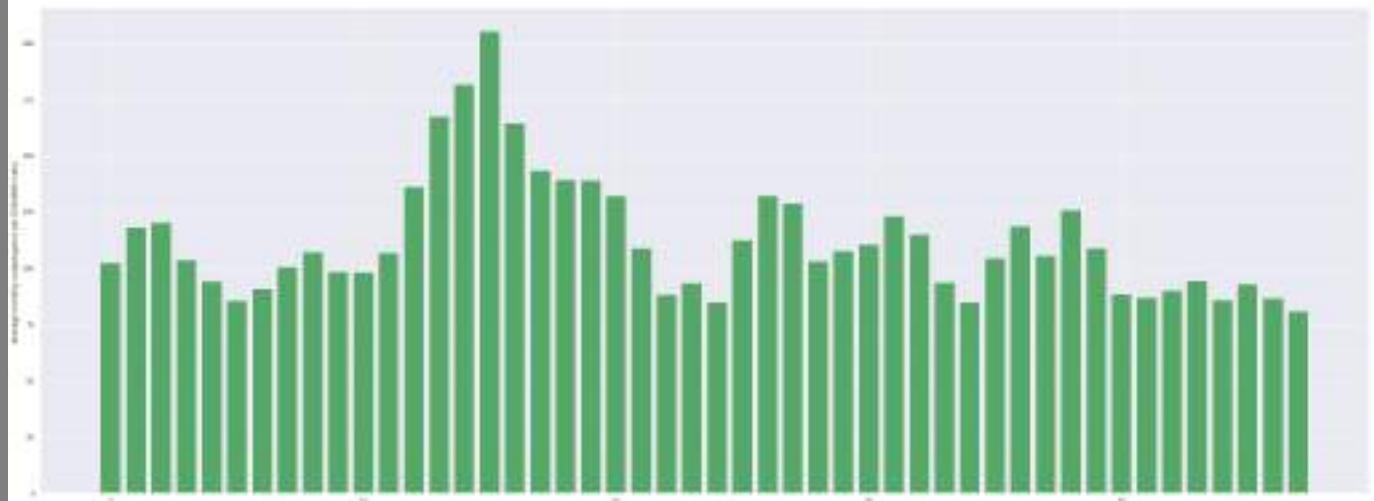
```
def Hist_pdToListNuclear(pdToListNuclear_Dict):
    for k, v in pdToListNuclear_Dict.items(): print(f"{v}:{k}")
```

```
#Output/price per EUR/MWH Histogram
print(pdToListNuclear_Dict)
```

```
plt.bar(list(pdToListNuclear_Dict.values()),pdToListNuclear_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of nuclear power")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')
```

```
plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{102.63388281689778: 0, 118.49355831823476: 1, 120.45420002671308: 2, 103.98877970152685: 3, 94.31723437707474: 4, 85.77923627046935: 5, 91.22607682648228: 6, 100.59437701408982: 7, 107.31387199496582: 8, 98.54116929029787: 9, 98.35979863363437: 10, 107.04105229192045: 11, 136.55164156780614: 12, 167.589529876522: 13, 181.9731332829844: 14, 205.60027830008724: 15, 164.7098182979096: 16, 143.678251667955: 17, 139.55033034817424: 18, 139.3492345580292: 19, 132.44075903166618: 20, 109.27916930870813: 21, 88.43406639744545: 22, 93.58618021036858: 23, 85.1646652444257: 24, 112.62559996020988: 25, 132.5738976103446: 26, 129.09227137597384: 27, 103.42141579325329: 28, 107.78619520307156: 29, 110.71705890251806: 30, 123.43403396826045: 31, 115.15739057905957: 32, 93.91829892025801: 33, 85.05519096023419: 34, 104.60483304468198: 35, 118.97813552223086: 36, 105.61235177921166: 37, 126.17482854931013: 38, 109.20822640101433: 39, 88.67218166973136: 40, 87.07676514860786: 41, 90.30989575192407: 42, 94.58429917431938: 43, 86.09609464822834: 44, 92.93511904364472: 45, 86.8276473194506: 46, 81.1049568960575: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram displays a unimodal distribution between months 10 and 20, meaning that the output amount increased per EUR/MWH. Besides that exception, the histogram is roughly uniform.

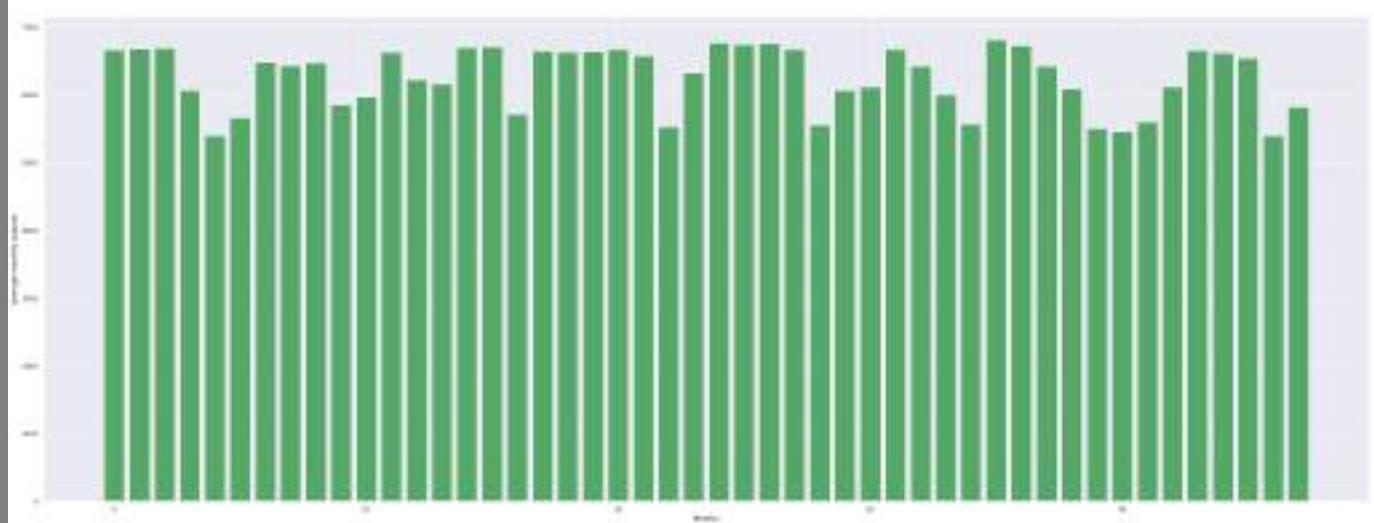
```
In [ ]: Nuclear_Dict = {key: i for i, key in enumerate(nuclear)}

def Hist_Nuclear(Nuclear_Dict):
    for k, v in Nuclear_Dict.items(): print(f"v:{k}") #Outputs in MWH Histogram
print(Nuclear_Dict)

plt.bar(list(Nuclear_Dict.values()), Nuclear_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of nuclear power")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.show()
plt.show()
```

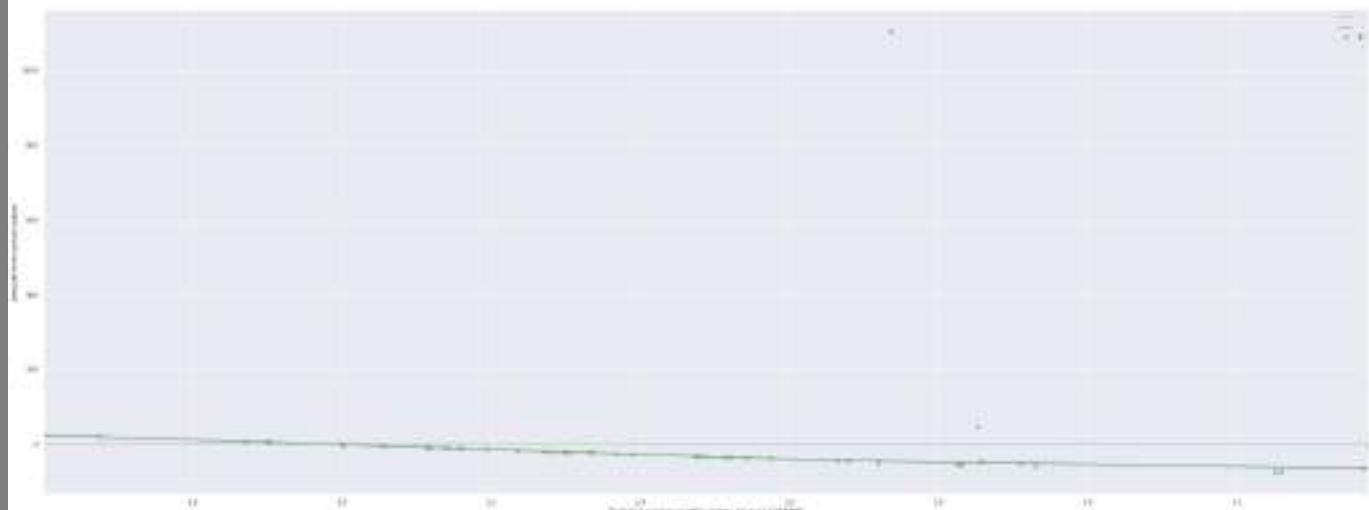
```
{6665.969986357435: 0, 6681.1235119047615: 1, 6687.913862718708: 2, 6068.16991643454: 3, 5403.817204301075: 4, 5659.276773296245: 5, 6483.623655913979: 6, 6437.845430107527: 7, 6466.175: 8, 5854.012096774193: 9, 5973.075: 10, 6626.029609690444: 11, 6223.8494623655915: 12, 6159.264367816092: 13, 6699.888290713325: 14, 6706.406944444445: 15, 5714.009408602151: 16, 6647.4638888888885: 17, 6628.9220430107525: 18, 6633.3494623655915: 19, 6675.755555555555: 20, 6576.6859060402685: 21, 5534.297222222222: 22, 6325.970430107527: 23, 6769.916666666667: 24, 6739.267857142857: 25, 6755.951547779273: 26, 6676.383333333333: 27, 5561.240591397849: 28, 6063.859722222222: 29, 6117.403225806452: 30, 6675.846774193548: 31, 6427.688888888889: 32, 6003.754362416107: 33, 5565.216666666666: 34, 6815.138440860215: 35, 6723.689516129032: 36, 6429.3735119047615: 37, 6091.685060565276: 38, 5504.175: 39, 5465.200268817204: 40, 5603.227777777778: 41, 6121.515477792732: 42, 6655.32123655914: 43, 6621.998611111111: 44, 6539.120805369127: 45, 5403.497222222222: 46, 5821.1518817204305: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. With the exception of the sudden but slight decrease in the average monthly outputs roughly every five months, this histogram is otherwise roughly uniform throughout.

```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals  
NuclearQuadRatioPredict = Nuclear_ypred/ypred  
  
sns.residplot(x = NuclearQuadRatioPredict , y = standardized_residualsNuclearQuad/standardized_residualsPriceQuad  
  
plt.suptitle("Predicted average monthly quadratic nuclear energy prices to EUR/MWH versus respective quadratic ratios")  
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")  
plt.ylabel("Amount from actual values")  
  
plt.legend(..#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60779de90>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly linear nuclear energy prices per EUR/MWH versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend(..#")  
sns.residplot(x = predictionsnuclear, y = Price_Actual, lowess = True, color="g")  
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6077ef990>
```

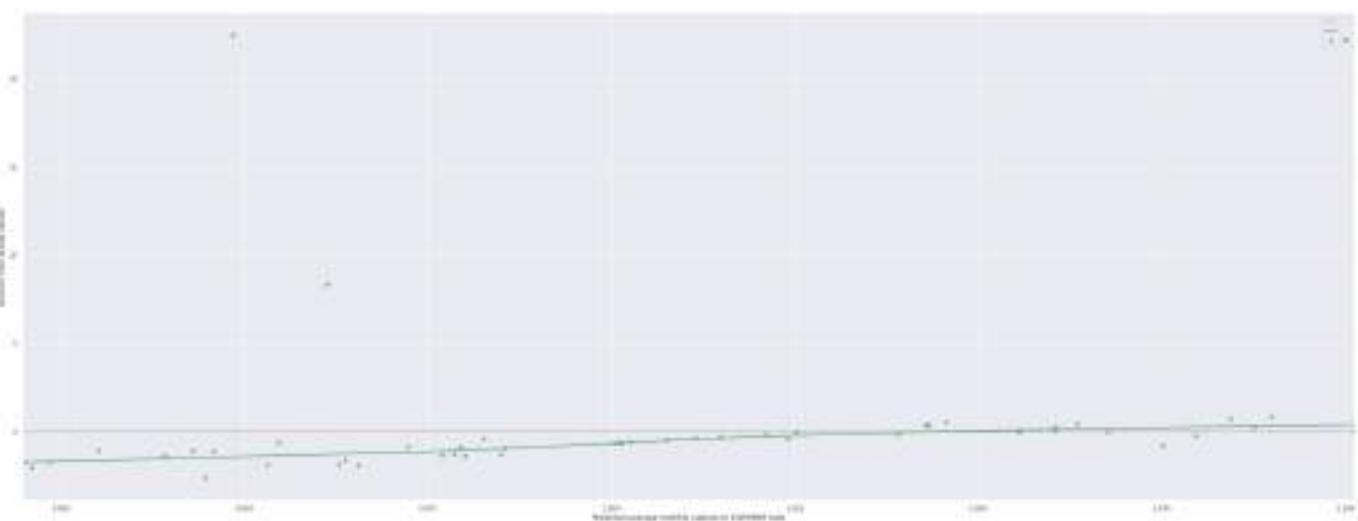


As one can observe, there is a subtle hump along the lowess line in the residual plot. Furthermore, the observations are quite spread out

in no particular pattern which indicates constant variance and a lack of bias. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals  
NuclearRegRatioPredict = predictionsnuclear/predictions  
  
sns.residplot(x = NuclearRegRatioPredict, y = standardized_residualsNuclear/standardized_residualsPricereg, low  
plt.suptitle("Predicted linear nuclear output to EUR/MWH versus respective linear model residuals ")  
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")  
plt.ylabel("Amount from actual values")  
  
plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff6075a5b90>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of nuclear power.

```
In [ ]: modelnuclear = stats.linregress( [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5  
55.522462987886975,  
58.35408333333333,  
57.29405913978498,  
65.9749027777778,  
71.07204301075271,  
63.99806451612899,  
60.254791666666634,  
59.40676510067113,  
60.72679166666668,  
61.901760752688226,  
45.57872311827956,  
36.752083333333374,  
36.81800807537014,  
32.61866666666666,  
34.691370967741896,  
46.266319444444434,  
47.50201612903221,  
47.6023387096774,  
50.4055972222224,  
60.182429530201404,  
62.58105555555558,  
67.5951344086021,  
79.4920833333331,  
59.83779761904767,  
50.95989232839837,  
51.71791666666662,  
53.77262096774189,  
56.2582222222224,  
55.252580645161316,  
54.08432795698931,
```

```
55.816555555555558,
63.92528859060403,
65.43065277777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881] )
```

In []:

```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(nuclear,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(nuclear,Price_Actual, "o")
plt.title(f"R squared: {modelnuclear.rvalue**2}")
f = lambda x: -0.001453 *x + 66.961236
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average nuclear monthly outputs versus predicted linear average monthly prices of energy per EUR/I")
plt.legend("#")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly nuclear output')
plt.show()
```

slope: -0.001453 intercept: 66.961236



There is a very weak yet positive correlation between the output and their average monthly price per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]:
print(Nuclear_Logpred)

print(Nuclear_Logpred/Logpred)

print(Logpred)

Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price
```

```
[57.27620225 57.25418559 57.24431984 58.14475009 59.10999271 58.73883375
57.54113449 57.60764599 57.56648576 58.45590145 58.28291414 57.3342319
57.9185622 58.01239833 57.22692211 57.21745112 58.65931232 57.30308989
57.33002946 57.32359684 57.26198473 57.40592371 58.92041739 57.77018995
57.12517741 57.16970727 57.14546742 57.26107263 58.88127119 58.1510124
58.07321867 57.2618522 57.6224025 58.23833989 58.87549432 57.05947438
57.19234115 57.6199549 58.11058477 58.96418218 59.02080883 58.82026772
58.06724395 57.29167389 57.34008857 57.4605023 59.11045761 58.50364429]
[2.09823319 2.39381174 2.42790295 2.35451294 2.43484035 2.12037043
1.93653426 2.13978966 2.26238609 2.32795534 2.27389788 2.19714178
2.94691948 3.58743639 3.53317956 3.9351846 3.81950174 2.87590229
2.80852572 2.80277516 2.65610957 2.25861048 2.23496948 2.03836098
1.72918154 2.26141627 2.62408236 2.59376541 2.5726723 2.43635811
2.47423449 2.48854212 2.43196863 2.16552617 2.14190856 2.08420004
2.38618219 2.24284276 2.80460116 2.73530974 2.27097736 2.17367836
2.04348469 1.94644304 1.79083227 1.95222326 2.11425197 2.08024295]
[27.29734838 23.91758071 23.57768041 24.69502249 24.27674267 27.70215664
29.71346062 26.92210695 25.44503172 25.11040502 25.63128037 26.09491682
19.65393442 16.17099008 16.19700362 14.53996621 15.35784412 19.92525621
20.41285544 20.45244219 21.55859284 25.41647801 26.36296287 28.34149128
33.03596301 25.28048812 21.77731469 22.076427 22.88720221 23.86800702
23.4711863 23.0102001 23.69372775 26.89338996 27.48739853 27.37715831
23.96813682 25.69059052 20.71973214 21.55667467 25.98916653 27.0602444
28.41579599 29.43403567 32.0186817 29.43336623 27.95809508 28.12346716]
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.
36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.
69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

In []: `print(list(Nuclear_Logpred))`

```
print(list(Logpred))
```

```
[57.27620225255758, 57.25418559106771, 57.244319843667796, 58.14475008566795, 59.1099927103785, 58.7388337455983
8, 57.54113449019134, 57.60764598931672, 57.56648576237625, 58.45590145477641, 58.282914143794756, 57.3342319005
16976, 57.91856220265974, 58.01239833073084, 57.22692211487729, 57.21745111803768, 58.65931232332505, 57.3030898
8884994, 57.33002946425246, 57.323596836221135, 57.26198473176693, 57.40592370692397, 58.920417394931924, 57.770
189946265006, 57.12517740581702, 57.16970727196902, 57.14546742309712, 57.2610726291159, 58.881271188546904, 58.
151012396483466, 58.07321867186711, 57.26185219958042, 57.62240249790744, 58.23833988713541, 58.87549432144239,
57.05947437906716, 57.19234115077194, 57.61995489754126, 58.110584771029124, 58.96418217877171, 59.0208088337656
4, 58.82026771667813, 58.06724395267321, 57.29167389437896, 57.340088566077085, 57.46050230005244, 59.1104576145
41616, 58.50364428723236]
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639
58084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.0949
16817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 1
9.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.36296287420
1227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.88720
2207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 2
7.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.5566746731514
95, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.4333662
30197276, 27.958095077371567, 28.123467161134005]
```

In []: `dfNuclear_Log = {"Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967,
"Nuclear_Log" : [57.27620225255758, 57.25418559106771, 57.244319843667796, 58.14475008566795, 59.1099927103785
"Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2019-01', '2019-02', '2019-03', '2019-04', '2019-05', '2019-06', '2019-07', '2019-08', '2019-09', '2020-01', '2020-02', '2020-03', '2020-04', '2020-05', '2020-06', '2020-07', '2020-08', '2020-09', '2021-01', '2021-02', '2021-03', '2021-04', '2021-05', '2021-06', '2021-07', '2021-08', '2021-09', '2022-01', '2022-02', '2022-03', '2022-04', '2022-05', '2022-06', '2022-07', '2022-08', '2022-09'], "Logpred": [2.09823319, 2.39381174, 2.42790295, 2.35451294, 2.43484035, 2.12037043, 1.93653426, 2.13978966, 2.26238609, 2.32795534, 2.27389788, 2.19714178, 2.94691948, 3.58743639, 3.53317956, 3.9351846, 3.81950174, 2.87590229, 2.80852572, 2.80277516, 2.65610957, 2.25861048, 2.23496948, 2.03836098, 1.72918154, 2.26141627, 2.62408236, 2.59376541, 2.5726723, 2.43635811, 2.47423449, 2.48854212, 2.43196863, 2.16552617, 2.14190856, 2.08420004, 2.38618219, 2.24284276, 2.80460116, 2.73530974, 2.27097736, 2.17367836, 2.04348469, 1.94644304, 1.79083227, 1.95222326, 2.11425197, 2.08024295], "Nuclear_Logpred": [57.27620225255758, 57.25418559106771, 57.244319843667796, 58.14475008566795, 59.1099927103785, 58.7388337455983, 57.54113449019134, 57.60764598931672, 57.56648576237625, 58.45590145477641, 58.282914143794756, 57.334231900516976, 57.91856220265974, 58.01239833073084, 57.22692211487729, 57.21745111803768, 58.65931232332505, 57.3030898884994, 57.33002946425246, 57.323596836221135, 57.26198473176693, 57.40592370692397, 58.920417394931924, 57.770189946265006, 57.12517740581702, 57.16970727196902, 57.14546742309712, 57.2610726291159, 58.881271188546904, 58.151012396483466, 58.07321867186711, 57.26185219958042, 57.62240249790744, 58.23833988713541, 58.87549432144239, 57.05947437906716, 57.19234115077194, 57.61995489754126, 58.110584771029124, 58.96418217877171, 59.02080883376564, 58.82026771667813, 58.06724395267321, 57.29167389437896, 57.340088566077085, 57.46050230005244, 59.110457614541616, 58.50364428723236]} #Dataframes df_Nuclear_Log= pd.DataFrame.from_dict(dfNuclear_Log, orient = "columns") print(df_Nuclear_Log)`

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + '\n')

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_Nuclear_Log["Nuclear_Log"])

test_result=adfuller(df_Nuclear_Log["Nuclear_Log"])

df_Nuclear_Log['First Difference'] = df_Nuclear_Log["Nuclear_Log"]- df_Nuclear_Log["Nuclear_Log"].shift(1) # Seasonal Difference
df_Nuclear_Log['Seasonal Difference']=df_Nuclear_Log["Nuclear_Log"]- df_Nuclear_Log["Nuclear_Log"].shift(12) #
```

```

plt.suptitle("Seasonal Difference of average monthly predicted nuclear logarithmic prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Nuclear_Log.head() #Predictive Logarithmic Seasonality Plot
df_Nuclear_Log['Seasonal Difference'].plot()
# Seasonality Plot

```

```

{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690509519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Nuclear_Log': [57.27620225255758, 57.25418559106771, 57.244319843667796, 58.14475008566795, 59.1099927103785, 58.73883374559838, 57.54113449019134, 57.60764598931672, 57.56648576237625, 58.45590145477641, 58.282914143794756, 57.334231900516976, 57.91856220265974, 58.01239833073084, 57.22692211487729, 57.21745111803768, 58.65931232332505, 57.30308988884994, 57.33002946425246, 57.323596836221135, 57.26198473176693, 57.40592370692397, 58.920417394931924, 57.770189946265006, 57.12517740581702, 57.16970727196902, 57.14546742309712, 57.2610726291159, 58.881271188546904, 58.151012396483466, 58.0732186718671, 57.26185219958042, 57.62240249790744, 58.23833988713541, 58.87549432144239, 57.05947437906716, 57.19234115077194, 57.61995489754126, 58.110584771029124, 58.96418217877171, 59.02080883376564, 58.82026771667813, 58.06724395267321, 57.29167389437896, 57.340088566077085, 57.46050230005244, 59.110457614541616, 58.50364428723236], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

```

	Price_Log	Nuclear_Log	Dates
0	27.297348	57.276202	2015-01
1	23.917581	57.254186	2015-02
2	23.577680	57.244320	2015-03
3	24.695022	58.144750	2015-04
4	24.276743	59.109993	2015-05
5	27.702157	58.738834	2015-06
6	29.713461	57.541134	2015-07
7	26.922107	57.607646	2015-08
8	25.445032	57.566486	2015-09
9	25.110405	58.455901	2015-10
10	25.631280	58.282914	2015-11
11	26.094917	57.334232	2015-12
12	19.653934	57.918562	2016-01
13	16.170990	58.012398	2016-02
14	16.197004	57.226922	2016-03
15	14.539966	57.217451	2016-04
16	15.357844	58.659312	2016-05
17	19.925256	57.303090	2016-06
18	20.412855	57.330029	2016-07
19	20.452442	57.323597	2016-08
20	21.558593	57.261985	2016-09
21	25.416478	57.405924	2016-10
22	26.362963	58.920417	2016-11
23	28.341491	57.770190	2016-12
24	33.035963	57.125177	2017-01
25	25.280488	57.169707	2017-02
26	21.777315	57.145467	2017-03
27	22.076427	57.261073	2017-04
28	22.887202	58.881271	2017-05
29	23.868007	58.151012	2017-06
30	23.471186	58.073219	2017-07
31	23.010200	57.261852	2017-08
32	23.693728	57.622402	2017-09
33	26.893390	58.238340	2017-10
34	27.487399	58.875494	2017-11
35	27.377158	57.059474	2017-12
36	23.968137	57.192341	2018-01
37	25.690591	57.619955	2018-02
38	20.719732	58.110585	2018-03
39	21.556675	58.964182	2018-04
40	25.989167	59.020809	2018-05
41	27.060244	58.820268	2018-06
42	28.415796	58.067244	2018-07
43	29.434036	57.291674	2018-08
44	32.018682	57.340089	2018-09
45	29.433366	57.460502	2018-10
46	27.958095	59.110458	2018-11
47	28.123467	58.503644	2018-12

ADF Test Statistic : -1.8038494997994792

p-value : 0.37855220561202113

#Lags Used : 5

Number of Observations : 42

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60750f7d0>

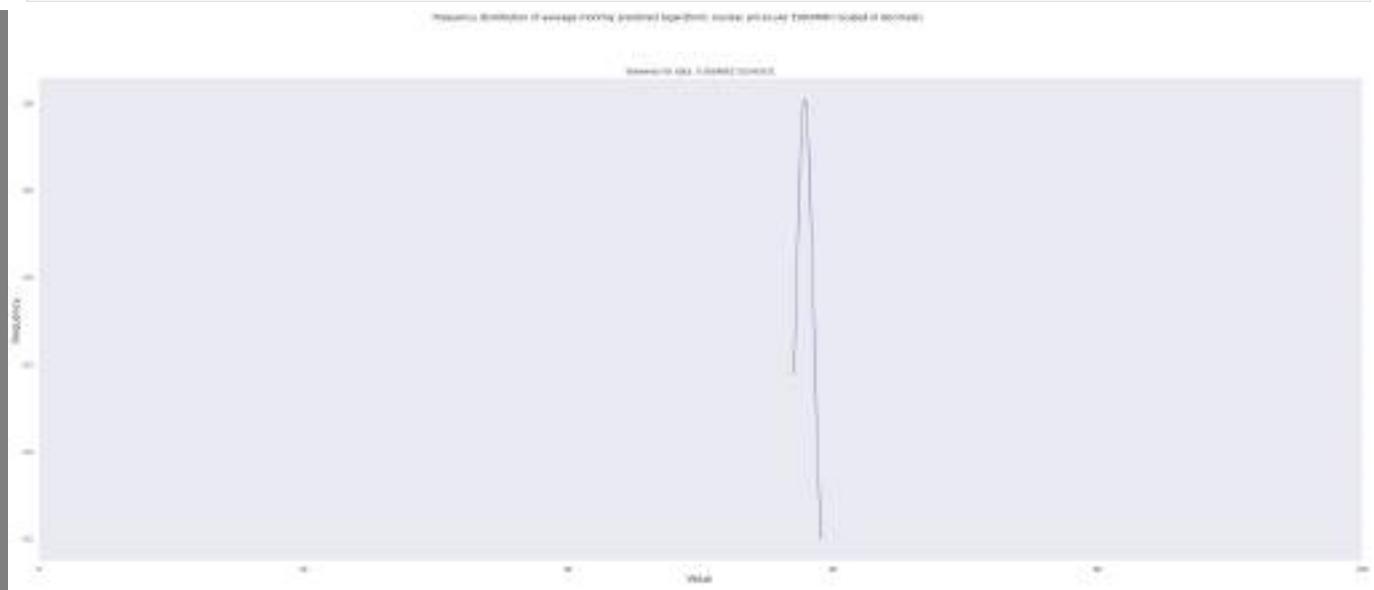
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWh.

In []: #Bell Curves

```
Nuclear_LogResults_mean = np.mean(df_Nuclear_Log["Nuclear_Log"])
Nuclear_LogResults_std = np.std(df_Nuclear_Log["Nuclear_Log"])

Nuclear_LogResultspdf = stats.norm.pdf(df_Nuclear_Log["Nuclear_Log"].sort_values(), Nuclear_LogResults_mean, Nu

plt.plot(df_Nuclear_Log["Nuclear_Log"].sort_values(), Nuclear_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Nuclear_Log["Nuclear_Log"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic nuclear prices per EUR/MWh (scaled")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Nuclear_Log_Dict = {key: i for i, key in enumerate(df_Nuclear_Log["Nuclear_Log"])}
def Hist_Nuclear_Log(Nuclear_Log_Dict):
    for k, v in Nuclear_Log_Dict.items(): print(f"{v}:{k}")
```

```

print(Nuclear_Log_Dict)

plt.bar(list(Nuclear_Log_Dict.values()), Nuclear_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly output of nuclear predicted logarithmic prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

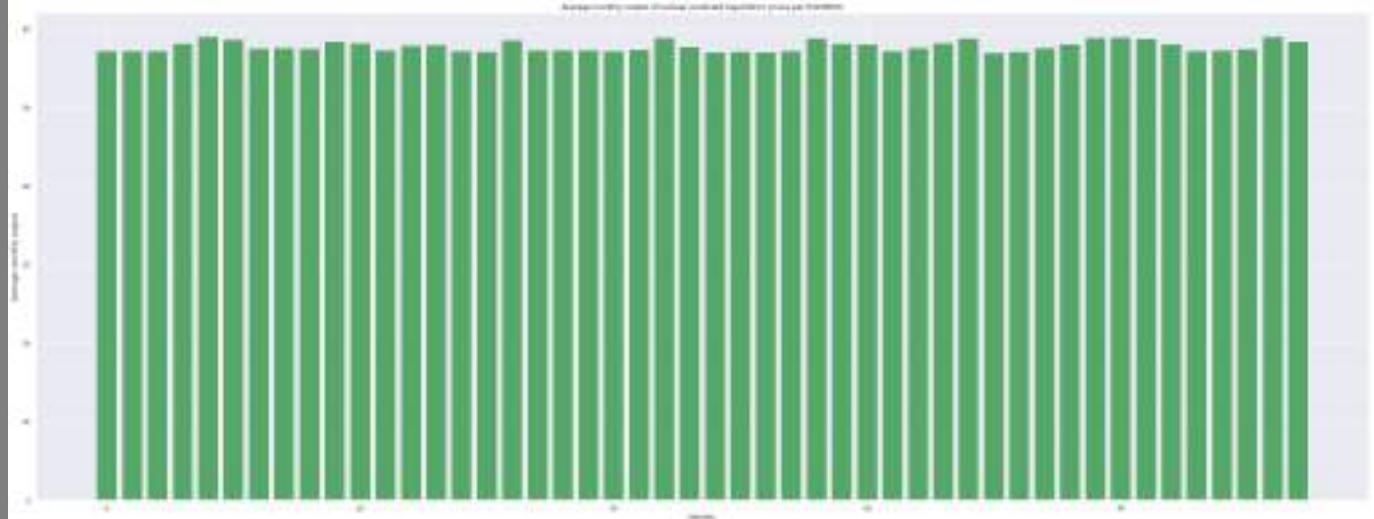
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{57.27620225255758: 0, 57.25418559106771: 1, 57.244319843667796: 2, 58.14475008566795: 3, 59.1099927103785: 4, 58.73883374559838: 5, 57.54113449019134: 6, 57.60764598931672: 7, 57.56648576237625: 8, 58.45590145477641: 9, 58.282914143794756: 10, 57.334231900516976: 11, 57.91856220265974: 12, 58.01239833073084: 13, 57.22692211487729: 14, 57.21745111803768: 15, 58.65931232332505: 16, 57.3030898884994: 17, 57.33002946425246: 18, 57.323596836221135: 19, 57.26198473176693: 20, 57.40592370692397: 21, 58.920417394931924: 22, 57.770189946265006: 23, 57.12517740581702: 24, 57.16970727196902: 25, 57.14546742309712: 26, 57.2610726291159: 27, 58.881271188546904: 28, 58.151012396483466: 29, 58.07321867186711: 30, 57.26185219958042: 31, 57.62240249790744: 32, 58.23833988713541: 33, 58.87549432144239: 34, 57.05947437906716: 35, 57.19234115077194: 36, 57.61995489754126: 37, 58.110584771029124: 38, 58.96418217877171: 39, 59.02080883376564: 40, 58.82026771667813: 41, 58.06724395267321: 42, 57.29167389437896: 43, 57.340088566077085: 44, 57.46050230005244: 45, 59.110457614541616: 46, 58.50364428723236: 47}

```



The green bars represent the observation value for each respective month.

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```

In [ ]: print(list(predictionsnuclear))

print(list(predictions))

```

```

[57.27620225255758, 57.25418559106771, 57.244319843667796, 58.14475008566795, 59.1099927103785, 58.73883374559838, 57.54113449019134, 57.60764598931672, 57.56648576237625, 58.45590145477641, 58.282914143794756, 57.334231900516976, 57.91856220265974, 58.01239833073084, 57.22692211487729, 57.21745111803768, 58.65931232332505, 57.3030898884994, 57.33002946425246, 57.323596836221135, 57.26198473176693, 57.40592370692397, 58.920417394931924, 57.770189946265006, 57.12517740581702, 57.16970727196902, 57.14546742309712, 57.2610726291159, 58.881271188546904, 58.151012396483466, 58.07321867186711, 57.26185219958042, 57.62240249790744, 58.23833988713541, 58.87549432144239, 57.05947437906716, 57.19234115077194, 57.61995489754126, 58.110584771029124, 58.96418217877171, 59.02080883376564, 58.82026771667813, 58.06724395267321, 57.29167389437896, 57.340088566077085, 57.46050230005244, 59.110457614541616, 58.50364428723236]
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895]

```

```

In [ ]: dfNuclearReg = ({"Price_Reg": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895] #Dataframes
df_NuclearReg= pd.DataFrame.from_dict(dfNuclearReg, orient = "columns")
print(df_NuclearReg)

```

```

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_NuclearReg["NuclearReg"])

test_result=adfuller(df_NuclearReg["NuclearReg"])

df_NuclearReg['First Difference'] = df_NuclearReg["NuclearReg"]- df_NuclearReg["NuclearReg"].shift(1) # Seasonal Difference
df_NuclearReg['Seasonal Difference']=df_NuclearReg["NuclearReg"]- df_NuclearReg["NuclearReg"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted nuclear linear prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_NuclearReg.head() #Predictive Linear Seasonality Plot
df_NuclearReg['Seasonal Difference'].plot()
# Seasonality Plot

```

```
{'Price_Reg': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.1109745796539, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.182939493943, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'NuclearReg': [57.27620225255758, 57.25418559106771, 57.244319843667796, 58.14475008566795, 59.1099927103785, 58.73883374559838, 57.54113449019134, 57.60764598931672, 57.56648576237625, 58.45590145477641, 58.282914143794756, 57.334231900516976, 57.91856220265974, 58.01239833073084, 57.22692211487729, 57.21745111803768, 58.65931232332505, 57.30308988884994, 57.33002946425246, 57.323596836221135, 57.26198473176693, 57.40592370692397, 58.920417394931924, 57.770189946265006, 57.12517740581702, 57.16970727196902, 57.14546742309712, 57.2610726291159, 58.881271188546904, 58.151012396483466, 58.07321867186711, 57.26185219958042, 57.62240249790744, 58.23833988713541, 58.87549432144239, 57.05947437906716, 57.19234115077194, 57.61995489754126, 58.110584771029124, 58.96418217877171, 59.02080883376564, 58.82026771667813, 58.06724395267321, 57.29167389437896, 57.340088566077085, 57.46050230005244, 59.110457614541616, 58.50364428723236], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Reg	NuclearReg	Dates
0	52.821613	57.276202	2015-01
1	53.036006	57.254186	2015-02
2	53.250399	57.244320	2015-03
3	53.464792	58.144750	2015-04
4	53.679185	59.109993	2015-05
5	53.893578	58.738834	2015-06
6	54.107971	57.541134	2015-07
7	54.322364	57.607646	2015-08
8	54.536757	57.566486	2015-09
9	54.751150	58.455901	2015-10
10	54.965543	58.282914	2015-11
11	55.179936	57.334232	2015-12
12	55.394329	57.918562	2016-01
13	55.608722	58.012398	2016-02
14	55.823115	57.226922	2016-03
15	56.037508	57.217451	2016-04
16	56.251901	58.659312	2016-05
17	56.466294	57.303090	2016-06
18	56.680687	57.330029	2016-07
19	56.895080	57.323597	2016-08
20	57.109473	57.261985	2016-09
21	57.323866	57.405924	2016-10
22	57.538259	58.920417	2016-11
23	57.752652	57.770190	2016-12
24	57.967045	57.125177	2017-01
25	58.181438	57.169707	2017-02
26	58.395831	57.145467	2017-03
27	58.610224	57.261073	2017-04
28	58.824617	58.881271	2017-05
29	59.039010	58.151012	2017-06
30	59.253403	58.073219	2017-07
31	59.467796	57.261852	2017-08
32	59.682189	57.622402	2017-09
33	59.896582	58.238340	2017-10
34	60.110975	58.875494	2017-11
35	60.325368	57.059474	2017-12
36	60.539761	57.192341	2018-01
37	60.754154	57.619955	2018-02
38	60.968547	58.110585	2018-03
39	61.182939	58.964182	2018-04
40	61.397332	59.020809	2018-05
41	61.611725	58.820268	2018-06
42	61.826118	58.067244	2018-07
43	62.040511	57.291674	2018-08
44	62.254904	57.340089	2018-09
45	62.469297	57.460502	2018-10
46	62.683690	59.110458	2018-11
47	62.898083	58.503644	2018-12

ADF Test Statistic : -1.8038494997994792

p-value : 0.37855220561202113

#Lags Used : 5

Number of Observations : 42

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60722fd10>



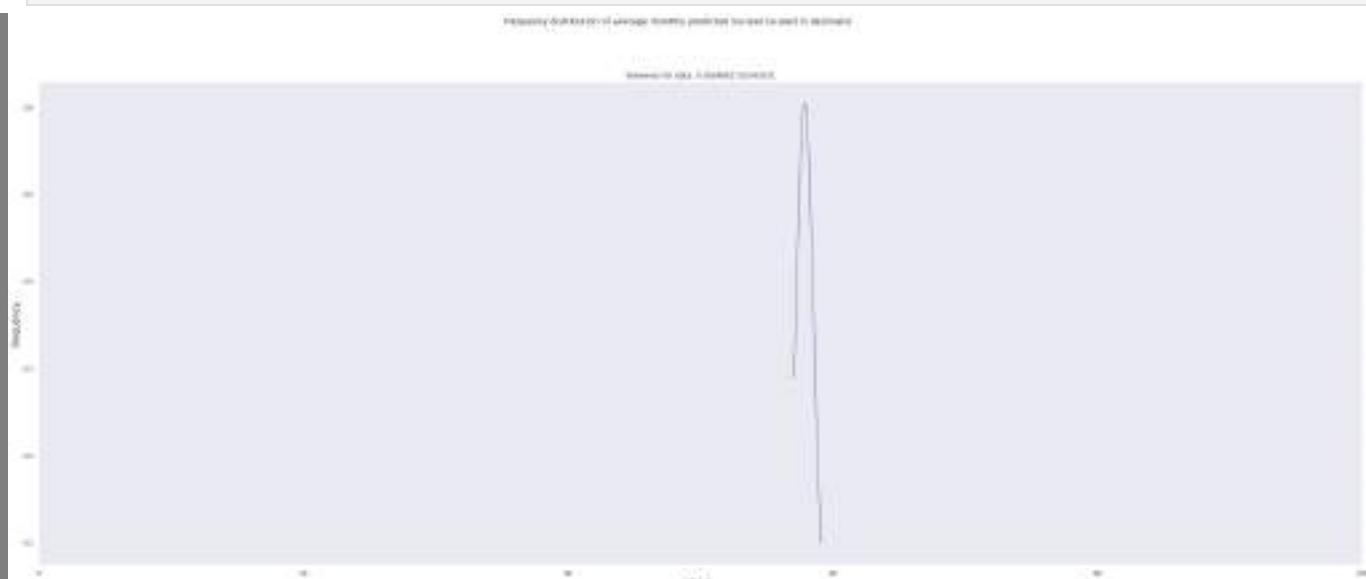
The blue line represents the trend line among the values themselves. As one can observe, there is a consistent up and down pattern depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
NuclearRegResults_mean = np.mean(df_NuclearReg["NuclearReg"])
NuclearRegResults_std = np.std(df_NuclearReg["NuclearReg"])

NuclearRegResultspdf = stats.norm.pdf(df_NuclearReg["NuclearReg"].sort_values(), NuclearRegResults_mean, NuclearRegResults_std)

plt.plot(df_NuclearReg["NuclearReg"].sort_values(), NuclearRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_NuclearReg["NuclearReg"])}')
plt.suptitle("Frequency distribution of average monthly predicted nuclear (scaled in decimals) " )
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
NuclearReg_Dict = {key: i for i, key in enumerate(df_NuclearReg["NuclearReg"])}

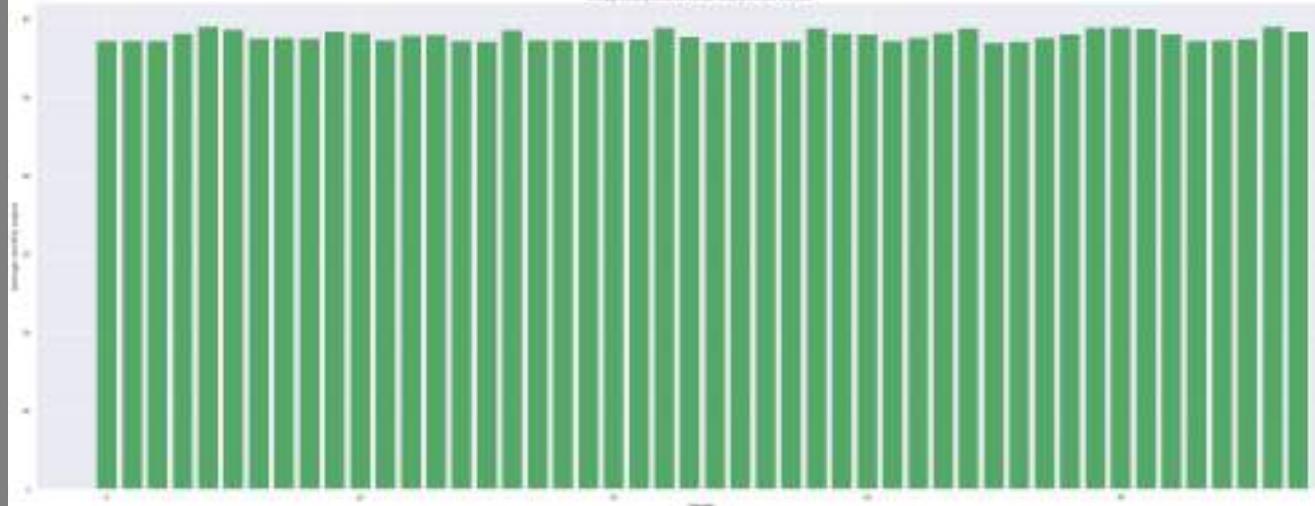
def Hist_NuclearReg(NuclearReg_Dict):
    for k, v in NuclearReg_Dict.items(): print(f"{v}:{k}")
print(NuclearReg_Dict)
```

```

plt.bar(list(NuclearReg_Dict.values()), NuclearReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted nuclear linear prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{57.27620225255758: 0, 57.25418559106771: 1, 57.244319843667796: 2, 58.14475008566795: 3, 59.1099927103785: 4, 58.73883374559838: 5, 57.54113449019134: 6, 57.60764598931672: 7, 57.56648576237625: 8, 58.45590145477641: 9, 58.282914143794756: 10, 57.334231900516976: 11, 57.91856220265974: 12, 58.01239833073084: 13, 57.22692211487729: 14, 57.21745111803768: 15, 58.65931232332505: 16, 57.3030898884994: 17, 57.33002946425246: 18, 57.323596836221135: 19, 57.26198473176693: 20, 57.40592370692397: 21, 58.920417394931924: 22, 57.770189946265006: 23, 57.12517740581702: 24, 57.16970727196902: 25, 57.14546742309712: 26, 57.2610726291159: 27, 58.881271188546904: 28, 58.151012396483466: 29, 58.07321867186711: 30, 57.26185219958042: 31, 57.62240249790744: 32, 58.23833988713541: 33, 58.87549432144239: 34, 57.05947437906716: 35, 57.19234115077194: 36, 57.61995489754126: 37, 58.110584771029124: 38, 58.96418217877171: 39, 59.02080883376564: 40, 58.82026771667813: 41, 58.06724395267321: 42, 57.29167389437896: 43, 57.340088566077085: 44, 57.46050230005244: 45, 59.110457614541616: 46, 58.50364428723236: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform.

In []: df_NuclearReg.describe(include = 'all') # Description Tables

	Price_Reg	NuclearReg	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	57.859848	NaN	0.026116	0.081810
std	3.001502	0.664224	NaN	0.773124	0.699463
min	52.821613	57.059474	NaN	-1.816020	-1.435744
25%	55.340731	57.272648	NaN	-0.488986	-0.289162
50%	57.859848	57.613800	NaN	0.026940	0.036722
75%	60.378966	58.326161	NaN	0.459122	0.638717
max	62.898083	59.110458	NaN	1.649955	1.703110

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

In []: print(list(ypred))

```
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.67299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.73662909849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: print(list(Nuclear_ypred))
```

```
[57.46383590958975, 57.49062459088138, 57.5030716492783, 57.4962999848789, 60.02502679161155, 58.74215574568393, 57.24853750112382, 57.22553088989514, 57.23829894623384, 58.024843487571616, 57.69733744861689, 57.39976990734421, 57.283237653160995, 57.35413913662963, 57.525689342001215, 57.53836034759341, 58.51776784370881, 57.43297220647305, 57.404090938829924, 57.410801438686605, 57.48097886762254, 57.333717111030666, 59.321303365146136, 57.22173702143495, 57.67503200958086, 57.60608243590508, 57.64292255400042, 57.482098080397435, 59.188597283210584, 57.50424882907389, 57.413340491379486, 57.48114134824753, 57.222115350065366, 57.62660435513207, 59.16937909579279, 57.78696520606016, 57.573177141123864, 57.22263944509902, 57.45487852784554, 59.47477636900392, 59.681358561975856, 58.990398399484775, 57.407063353927384, 57.445827637402104, 57.39383092624509, 57.29313653791648, 60.02687698272631, 58.130072369433606]
```

```
In [ ]: dfNuclear_Quad = ({ "Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 57.46383590958975, 57.49062459088138, 57.5030716492783, 57.4962999848789, 60.02502679161155, "Nuclear_Quad" : [57.46383590958975, 57.49062459088138, 57.5030716492783, 57.4962999848789, 60.02502679161155, "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08']}, print(dfNuclear_Quad) #Dataframes df_Nuclear_Quad= pd.DataFrame.from_dict(dfNuclear_Quad, orient = "columns") print(df_Nuclear_Quad)
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + " " )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_Nuclear_Quad["Nuclear_Quad"])

test_result=adfuller(df_Nuclear_Quad["Nuclear_Quad"])

df_Nuclear_Quad['First Difference'] = df_Nuclear_Quad["Nuclear_Quad"]- df_Nuclear_Quad["Nuclear_Quad"].shift(1)
df_Nuclear_Quad['Seasonal Difference']=df_Nuclear_Quad["Nuclear_Quad"]- df_Nuclear_Quad["Nuclear_Quad"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted quadratic nuclear prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Nuclear_Quad.head() #Predictive Quadratic Seasonality Plot
df_Nuclear_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.733629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.886735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Nuclear_Quad': [57.46383590958975, 57.49062459088138, 57.5030716492783, 57.4962999848789, 60.02502679161155, 58.74215574568393, 57.24853750112382, 57.22553088989514, 57.23829894623384, 58.024843487571616, 57.69733744861689, 57.39976990734421, 57.283237653160995, 57.35413913662963, 57.525689342001215, 57.53836034759341, 58.51776784370881, 57.43297220647305, 57.404090938829924, 57.410801438686605, 57.48097886762254, 57.333717111030666, 59.321303365146136, 57.22173702143495, 57.67503200958086, 57.60608243590508, 57.64292255400042, 57.482098080397435, 59.188597283210584, 57.50424882907389, 57.413340491379486, 57.48114134824753, 57.222115350065366, 57.62660435513207, 59.16937909579279, 57.78696520606016, 57.573177141123864, 57.22263944509902, 57.45487852784554, 59.47477636900392, 59.681358561975856, 58.990398399484775, 57.407063353927384, 57.445827637402104, 57.39383092624509, 57.29313653791648, 60.02687698272631, 58.130072369433606], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Quad	Nuclear_Quad	Dates
0	27.188078	57.463836	2015-01
1	23.225548	57.490625	2015-02
2	22.878236	57.503072	2015-03
3	24.055117	57.496300	2015-04
4	23.602709	60.025027	2015-05
5	27.724722	58.742156	2015-06
6	30.587839	57.248538	2015-07
7	26.702482	57.225531	2015-08
8	24.901804	57.238299	2015-09
9	24.518415	58.024843	2015-10
10	25.119121	57.697337	2015-11
11	25.672300	57.399770	2015-12
12	19.546363	57.283238	2016-01
13	17.633406	57.354139	2016-02
14	17.644052	57.525689	2016-03
15	17.075344	57.538360	2016-04
16	17.328255	58.517768	2016-05
17	19.736629	57.432972	2016-06
18	20.093546	57.404091	2016-07
19	20.123369	57.410801	2016-08
20	21.007992	57.480979	2016-09
21	24.868735	57.333717	2016-10
22	26.000055	59.321303	2016-11
23	28.599305	57.221737	2016-12
24	36.035067	57.675032	2017-01
25	24.712150	57.606082	2017-02
26	21.194644	57.642923	2017-03
27	21.456173	57.482098	2017-04
28	22.201511	59.188597	2017-05
29	23.174311	57.504249	2017-06
30	22.771345	57.413340	2017-07
31	22.319233	57.481141	2017-08
32	22.995762	57.222115	2017-09
33	26.665789	57.626604	2017-10
34	27.438370	59.169379	2017-11
35	27.292830	57.786965	2017-12
36	23.278006	57.573177	2018-01
37	25.188915	57.222639	2018-02
38	20.328049	57.454879	2018-03
39	21.006372	59.474776	2018-04
40	25.544593	59.681359	2018-05
41	26.879918	58.990398	2018-06
42	28.703098	57.407063	2018-07
43	30.170478	57.445828	2018-08
44	34.272280	57.393831	2018-09
45	30.169486	57.293137	2018-10
46	28.070862	60.026877	2018-11
47	28.297338	58.130072	2018-12

ADF Test Statistic : -1.814407508268105

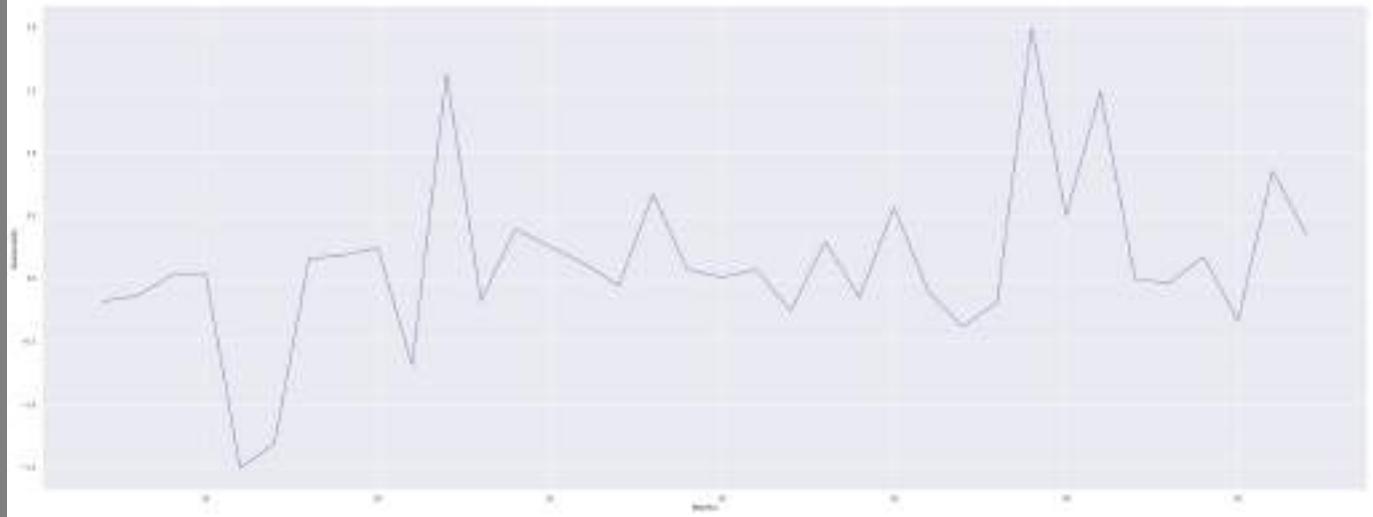
p-value : 0.37333258894953103

#Lags Used : 5

Number of Observations : 42

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff607061b50>



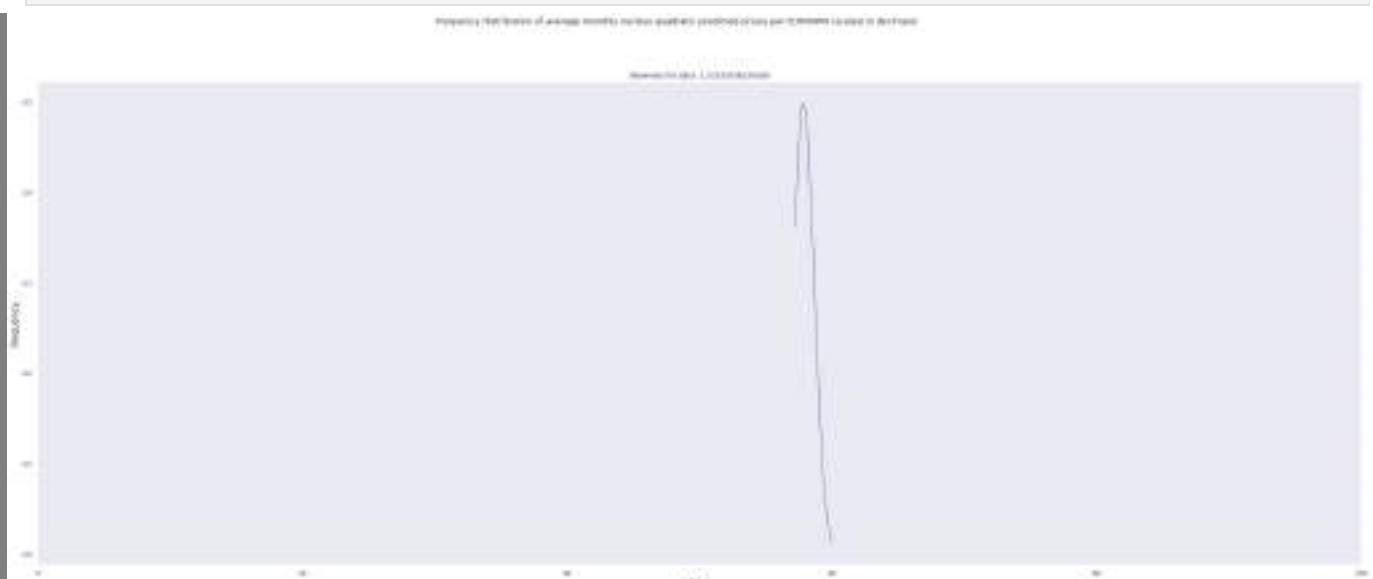
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Nuclear_QuadResults_mean = np.mean(df_Nuclear_Quad["Nuclear_Quad"])
Nuclear_QuadResults_std = np.std(df_Nuclear_Quad["Nuclear_Quad"])

Nuclear_QuadResultspdf = stats.norm.pdf(df_Nuclear_Quad["Nuclear_Quad"].sort_values(), Nuclear_QuadResults_mean)

plt.plot(df_Nuclear_Quad["Nuclear_Quad"].sort_values(), Nuclear_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Nuclear_Quad["Nuclear_Quad"])}')
plt.suptitle("Frequency distribution of average monthly nuclear quadratic predicted prices per EUR/MWH (scaled :")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Nuclear_Quad_Dict = {key: i for i, key in enumerate(df_Nuclear_Quad["Nuclear_Quad"])}

def Hist_Nuclear_Quad(Nuclear_Quad_Dict):
    for k, v in Nuclear_Quad_Dict.items(): print(f"{v}:{k}")
print(Nuclear_Quad_Dict)
```

```

plt.bar(list(Nuclear_Quad_Dict.values()), Nuclear_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic nuclear prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-010': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{57.4638359058975: 0, 57.49062459088138: 1, 57.5030716492783: 2, 57.4962999848789: 3, 60.02502679161155: 4, 58.74215574568393: 5, 57.24853750112382: 6, 57.22553088989514: 7, 57.23829894623384: 8, 58.024843487571616: 9, 57.69733744861689: 10, 57.39976990734421: 11, 57.283237653160995: 12, 57.35413913662963: 13, 57.525689342001215: 14, 57.53836034759341: 15, 58.51776784370881: 16, 57.43297220647305: 17, 57.404090938829924: 18, 57.410801438686605: 19, 57.48097886762254: 20, 57.333717111030666: 21, 59.321303365146136: 22, 57.22173702143495: 23, 57.67503200958086: 24, 57.60608243590508: 25, 57.64292255400042: 26, 57.482098080397435: 27, 59.188597283210584: 28, 57.50424882907389: 29, 57.413340491379486: 30, 57.48114134824753: 31, 57.222115350065366: 32, 57.62660435513207: 33, 59.16937909579279: 34, 57.78696520606016: 35, 57.573177141123864: 36, 57.22263944509902: 37, 57.45487852784554: 38, 59.47477636900392: 39, 59.681358561975856: 40, 58.990398399484775: 41, 57.407063353927384: 42, 57.445827637402104: 43, 57.39383092624509: 44, 57.29313653791648: 45, 60.02687698272631: 46, 58.130072369433606: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform throughout.

```
In [ ]: df_Nuclear_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Nuclear_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	0.014175	0.126075
std	4.174498	0.805332	NaN	1.048650	0.666129
min	17.075344	57.221737	NaN	-2.099566	-1.507259
25%	21.390791	57.403011	NaN	-0.278297	-0.158451
50%	24.615282	57.493462	NaN	-0.006772	0.056200
75%	27.214266	57.846435	NaN	0.189066	0.305442
max	36.035067	60.026877	NaN	2.733740	1.992678

```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Nuclear_Logpred)
plt.suptitle(" Autocorrelations of average monthly Nuclear Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Nuclear_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Nuclear Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Nuclear_Log_Autocorrelations = sm.tsa.acf(Nuclear_Logpred, fft=False) #Autocorrelations

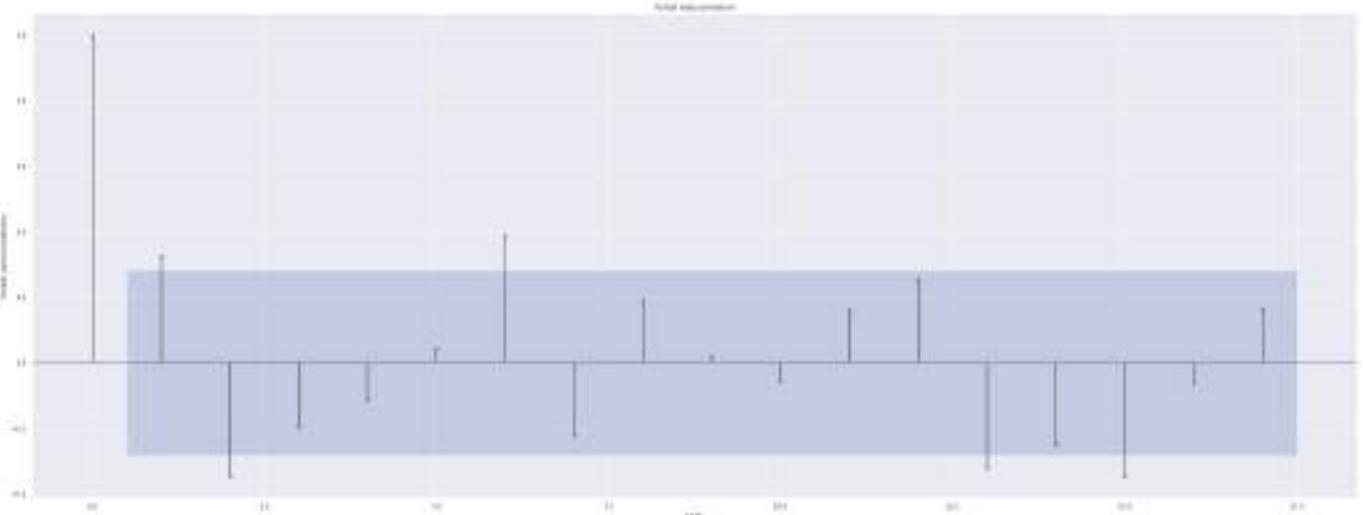
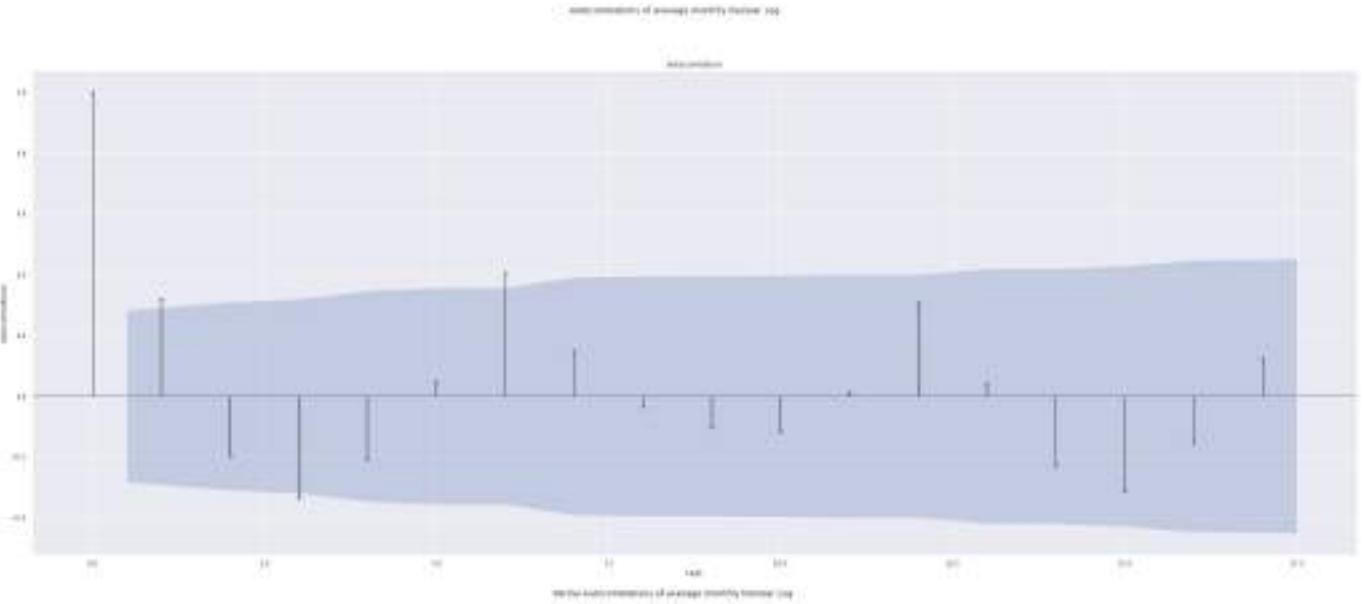
```

```
print(Nuclear_Log_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.31804229 -0.19597398 -0.33340343 -0.20767238  0.04545865  
 0.40377707  0.14834035 -0.03341788 -0.10208333 -0.11931636  0.01041069  
 0.30654397  0.03946094 -0.22830277 -0.31307236 -0.15293682  0.12196962  
 0.33936238  0.05972386 -0.17423594 -0.2341758  -0.27106887 -0.02860904  
 0.25473366  0.14318101 -0.10672179 -0.1748156  -0.11257482  0.06232189  
 0.14730321 -0.02450452 -0.14592436 -0.0782833  0.03224153  0.10616945  
 0.15302189  0.11907939 -0.05850139 -0.16511896 -0.15000688]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
```

```
plot_acf(predictionsnuclear)
```

```
plt.suptitle(" Autocorrelations of average monthly Linear Nuclear")
```

```
plt.ylabel('Autocorrelations')
```

```
plt.xlabel('Lags')
```

```
plt.show
```

```
from statsmodels.graphics.tsaplots import plot_pacf #Partial autocorrelation Plot
```

```
plot_pacf(predictionsnuclear)
```

```
plt.suptitle("Partial Autocorrelations of average monthly Linear Nuclear")
```

```
plt.ylabel('Partial autocorrelations')
```

```
plt.xlabel('Lags')
```

```
plt.show
```

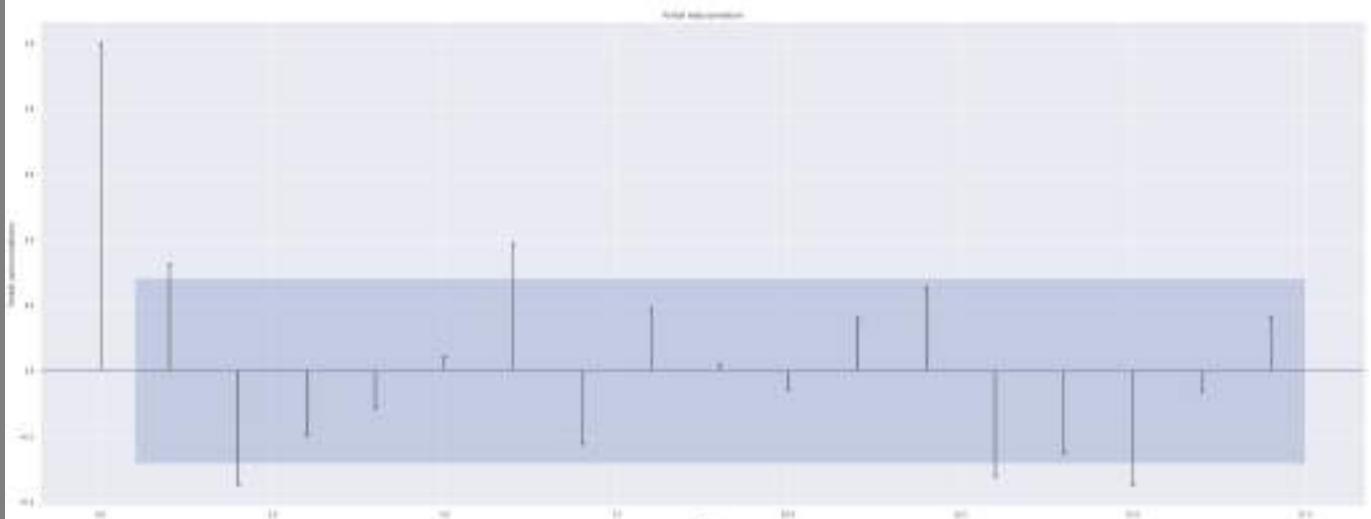
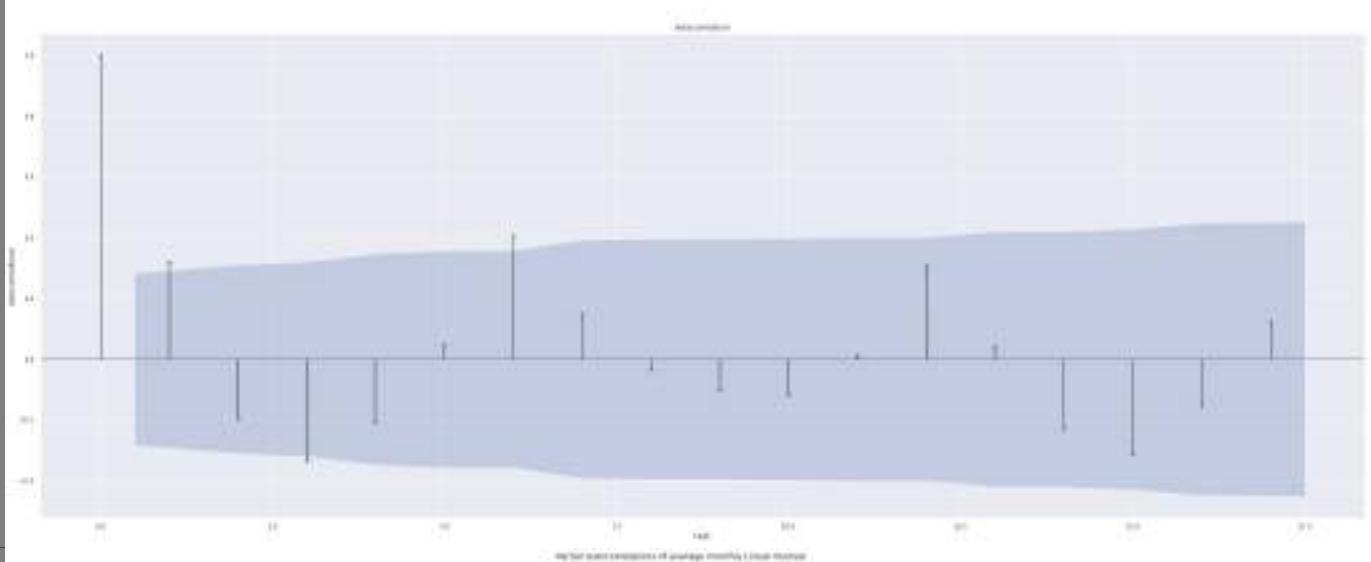
```
Nuclear_Pred_Autocorrelations = sm.tsa.acf(predictionsnuclear, fft=False) #Autocorrelations
```

```
print(Nuclear_Pred_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statauto.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.31804229 -0.19597398 -0.33340343 -0.20767238  0.04545865  
 0.40377707  0.14834035 -0.03341788 -0.10208333 -0.11931636  0.01041069  
 0.30654397  0.03946094 -0.22830277 -0.31307236 -0.15293682  0.12196962  
 0.33936238  0.05972386 -0.17423594 -0.2341758  -0.27106887 -0.02860904  
 0.25473366  0.14318101 -0.10672179 -0.1748156  -0.11257482  0.06232189  
 0.14730321 -0.02450452 -0.14592436 -0.0782833  0.03224153  0.10616945  
 0.15302189  0.11907939 -0.05850139 -0.16511896 -0.15000688]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resources analyzed are other renewables. These renewables are not specified.

```
In [ ]: otherrenewable1 = other_renewable  
otherrenewable1 = sm.add_constant(otherrenewable1)
```

```
In [ ]: other_renewable1 = other_renewable  
other_renewable1 = sm.add_constant(other_renewable1)
```

```
In [ ]: #Dataframes analyzed by resource  
dfotherrenewable = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29,  
"Other_Renewable" : [70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306407, 70.5658602150]})
```

```
print(dfotherrenewable)
df_otherrenewable= pd.DataFrame.from_dict(dfotherrenewable, orient = "columns")
print(df_otherrenewable)
df_otherrenewable[ "Ratio" ] = df_otherrenewable[ "Other_Renewable" ]/df_otherrenewable[ "Price" ]
pdToListOther = list(df_otherrenewable[ "Ratio" ])
print(pdToListOther)
```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.974
90277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.90
1760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 4
6.26631944444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556
, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.08432795698924, 55.81655555555556, 53.772620967741
936, 56.2582222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277
77778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.4007361111111, 61.6337
6344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04
260752688172, 66.6235138888889], 'Other_Renewable': [70.66030013642565, 70.51488095238095, 66.63257065948856, 69
.70055710306407, 70.56586021505376, 65.51182197496523, 68.3252688172043, 68.1760752688172, 67.96666666666667, 70
.66935483870968, 70.3125, 71.62314939434724, 77.95430107526882, 74.45402298850574, 73.40242261103634, 78.275, 79
.28225806451613, 83.48333333333333, 78.6271870794078, 78.43010752688173, 83.79166666666667, 83.78120805369127, 8
5.95694444444445, 90.09005376344086, 99.20430107526882, 94.90922619047619, 95.6850605652759, 95.97083333333333
, 96.74596774193549, 92.10555555555555, 93.39650537634408, 92.24731182795699, 94.8722222222222, 95.6268456375839,
94.9013888888889, 91.81586021505376, 96.98252688172043, 96.75744047619048, 97.64199192462988, 97.1513888888889
, 100.59005376344086, 101.1097222222222, 96.71736204576042, 96.61559139784946, 100.19444444444444, 98.617449664
42953, 95.88172043010752, 96.4625], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06',
'2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2
016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '201
7-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-
01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11
', '2018-12']]}



|    | Price     | Other_Renewable | Dates   |
|----|-----------|-----------------|---------|
| 0  | 64.949019 | 70.660300       | 2015-01 |
| 1  | 56.383854 | 70.514881       | 2015-02 |
| 2  | 55.522463 | 66.632571       | 2015-03 |
| 3  | 58.354083 | 69.700557       | 2015-04 |
| 4  | 57.294059 | 70.565860       | 2015-05 |
| 5  | 65.974903 | 65.511822       | 2015-06 |
| 6  | 71.072043 | 68.325269       | 2015-07 |
| 7  | 63.998065 | 68.176075       | 2015-08 |
| 8  | 60.254792 | 67.966667       | 2015-09 |
| 9  | 59.406765 | 70.669355       | 2015-10 |
| 10 | 60.726792 | 70.312500       | 2015-11 |
| 11 | 61.901761 | 71.623149       | 2015-12 |
| 12 | 45.578723 | 77.954301       | 2016-01 |
| 13 | 36.752083 | 74.454023       | 2016-02 |
| 14 | 36.818008 | 73.402423       | 2016-03 |
| 15 | 32.618667 | 78.275000       | 2016-04 |
| 16 | 34.691371 | 79.282258       | 2016-05 |
| 17 | 46.266319 | 83.483333       | 2016-06 |
| 18 | 47.502016 | 78.627187       | 2016-07 |
| 19 | 47.602339 | 78.430108       | 2016-08 |
| 20 | 50.405597 | 83.791667       | 2016-09 |
| 21 | 60.182430 | 83.781208       | 2016-10 |
| 22 | 62.581056 | 85.956944       | 2016-11 |
| 23 | 67.595134 | 90.090054       | 2016-12 |
| 24 | 79.492083 | 99.204301       | 2017-01 |
| 25 | 59.837798 | 94.909226       | 2017-02 |
| 26 | 50.959892 | 95.685061       | 2017-03 |
| 27 | 51.717917 | 95.970833       | 2017-04 |
| 28 | 53.772621 | 96.745968       | 2017-05 |
| 29 | 56.258222 | 92.105556       | 2017-06 |
| 30 | 55.252581 | 93.396505       | 2017-07 |
| 31 | 54.084328 | 92.247312       | 2017-08 |
| 32 | 55.816556 | 94.872222       | 2017-09 |
| 33 | 63.925289 | 95.626846       | 2017-10 |
| 34 | 65.430653 | 94.901389       | 2017-11 |
| 35 | 65.151277 | 91.815860       | 2017-12 |
| 36 | 56.511976 | 96.982527       | 2018-01 |
| 37 | 60.877098 | 96.757440       | 2018-02 |
| 38 | 48.279717 | 97.641992       | 2018-03 |
| 39 | 50.400736 | 97.151389       | 2018-04 |
| 40 | 61.633763 | 100.590054      | 2018-05 |
| 41 | 64.348139 | 101.109722      | 2018-06 |
| 42 | 67.783441 | 96.717362       | 2018-07 |
| 43 | 70.363911 | 96.615591       | 2018-08 |
| 44 | 76.914042 | 100.194444      | 2018-09 |
| 45 | 70.362215 | 98.617450       | 2018-10 |
| 46 | 67.042608 | 95.881720       | 2018-11 |
| 47 | 66.623514 | 96.462500       | 2018-12 |



[1.0879348360180074, 1.2506218667483064, 1.2001011315730978, 1.1944418131789818, 1.2316435818046778, 0.992980955
1311909, 0.9613522550191385, 1.065283392306891, 1.1279877464793602, 1.1895842959796394, 1.1578497409504178, 1.15
70454301049409, 1.7103221797805228, 2.025844965392141, 1.9936554541672733, 2.3996995585349907, 2.28535961133195
, 1.8044083544094802, 1.6552389453497844, 1.6476103832885234, 1.662348455018912, 1.3921207353659155, 1.373529795
5806647, 1.332789032104891, 1.2479771181650434, 1.586108278829176, 1.877654292294365, 1.8556593056887123, 1.7991
67792099499, 1.637192785647135, 1.690355532461871, 1.7056200069882905, 1.6997147401507715, 1.4959157439242206, 1
.4504117697129297, 1.4092718456115882, 1.7161411452658597, 1.5893898249815874, 2.0224226084925134, 1.92757876937
97464, 1.6320608729330732, 1.571292099011756, 1.426858253555079, 1.3730844352755214, 1.3026807884920595, 1.40156
829903176, 1.430160967287293, 1.4478746972258918]


```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy

per EUR/MWH.This is the logarithmic model for the average monthly other renewable outputs versus the predicted average monthly prices of energy per EUR/MWH.

In []:

```
#Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
Logotherrenewablevalues = ((np.log(other_renewable)))
Log = np.polyfit(np.log(Price_Actual), other_renewable1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(other_renewable1), Price_Actual)
Otherrenewable_Log = sm.OLS(Price_Actual, other_renewable1).fit()

Otherrenewable_Logpred = Otherrenewable_Log.predict(other_renewable1)
#OLS Logarithmic summary table
Otherrenewable_Log.summary()
#Log
Log = np.polyfit(np.log(other_renewable), Price_Actual, 1)
print(Log)

y = 17.05011223 * Logotherrenewablevalues - 17.84401394

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly other renewable outputs versus predicted average monthly prices of energy per EUR/MWH")
plt.title("R squared : 0.069")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(Logotherrenewablevalues, Price_Actual, "o")
plt.plot(Logotherrenewablevalues, y)

plt.xlim([1, 7])
```

[17.05011223 -17.84401394]

Out[]:



The blue dots represent the observations and the orange line is the model of best fit.This is a very weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

In []:

```
Otherrenewable_Log.summary() #OLS Logarithmic summary table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.069			
Model:	OLS	Adj. R-squared:	0.049			
Method:	Least Squares	F-statistic:	3.426			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.0706			
Time:	05:25:30	Log-Likelihood:	-177.92			
No. Observations:	48	AIC:	359.8			
Df Residuals:	46	BIC:	363.6			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	38.4104	10.607	3.621	0.001	17.059	59.762
x1	0.2271	0.123	1.851	0.071	-0.020	0.474
Omnibus:	1.865	Durbin-Watson:	0.436			
Prob(Omnibus):	0.394	Jarque-Bera (JB):	1.772			
Skew:	-0.441	Prob(JB):	0.412			
Kurtosis:	2.671	Cond. No.	631.			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In []:

```
influenceOtherrenewableLog = Otherrenewable_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsOtherrenewableLog = influenceOtherrenewableLog.resid_studentized_internal

print(standardized_residualsOtherrenewableLog)

print(Otherrenewable_Logpred) # OLS logarithmic predicted values
```

[1.07173466 0.20008828 0.20431036 0.42119401 0.29189581 1.3148206
 1.76196653 1.03882106 0.65920866 0.50527028 0.64896446 0.73644449
 -1.06284522 -1.88251627 -1.85515545 -2.37656007 -2.18832498 -1.11552308
 -0.88359491 -0.86915788 -0.70670097 0.27552976 0.46669786 0.87724287
 1.88928248 -0.01302703 -0.92926357 -0.85951039 -0.67017714 -0.30942696
 -0.4408867 -0.53173425 -0.4186111 0.38405904 0.5524565 0.59288256
 -0.39810014 0.0497372 -1.24962417 -1.02204342 0.03849822 0.3041311
 0.75071479 1.01444982 1.60730781 0.97181016 0.68119389 0.65148013]
[54.45914064 54.4261123 53.5443423 54.24115894 54.43769097 53.28979218
 53.92879649 53.8949109 53.84734895 54.46119719 54.38014651 54.67782785
 56.11579113 55.32079022 55.08194541 56.18862989 56.41740345 57.37157297
 56.26862041 56.2238587 57.4416032 57.43922779 57.93339207 58.87212482
 60.94219889 59.9666797 60.14289114 60.2077973 60.38384976 59.32989581
 59.62310288 59.36209221 59.95827517 60.12966907 59.96489965 59.26409873
 60.43757827 60.3864555 60.58735931 60.47593106 61.25693807 61.3749678
 60.37735269 60.35423802 61.16708527 60.80891022 60.31946713 60.18755752]

In []:

OtherrenewableLogRatioPredict = Otherrenewable_Logpred/ypred

In []:

```
# OLS Logarithmic average monthly predicted ratios versus residuals
plt.suptitle("Predicted average monthly logarithmic other renewable energy prices to prices of energy per EUR/MWh")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend("...#")
sns.residplot(x = OtherrenewableLogRatioPredict, y = standardized_residualsOtherrenewableLog/standardized_residuals)
```

Out[]:

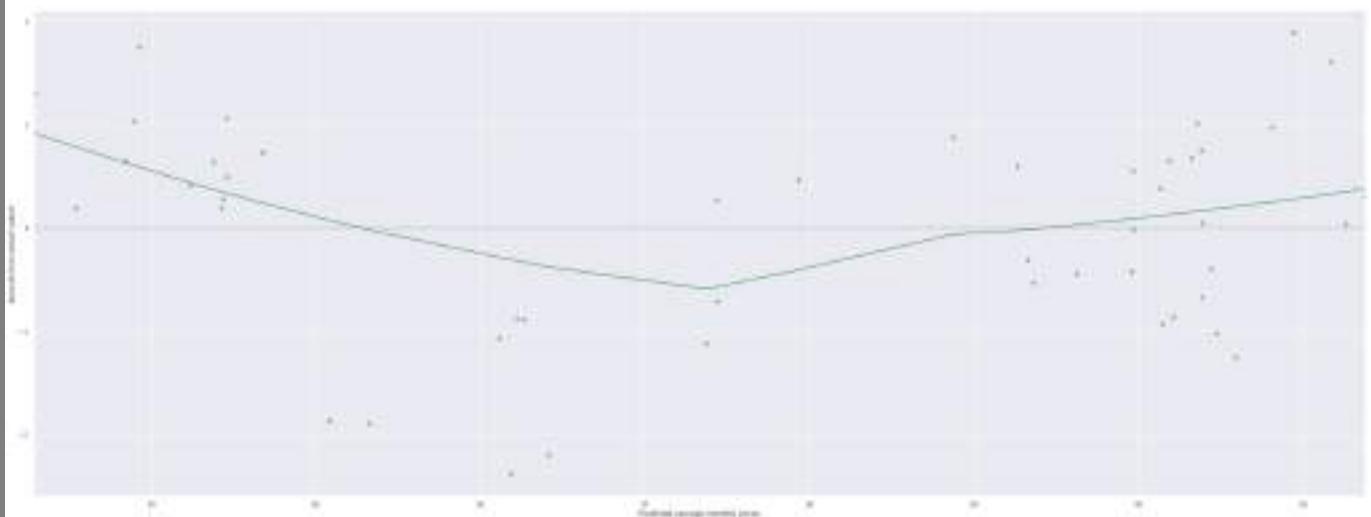
<matplotlib.axes._subplots.AxesSubplot at 0x7ff607ee3e10>



With the exception of few heteroskedastic outliers, the residuals seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS Logarithmic average monthly predictions versus residuals
plt.suptitle("Residuals from logarithmic model versus predicted average monthly logarithmic other renewable outputs")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = Otherrenewable_Logpred, y = standardized_residualsOtherrenewableLog, lowess = True, color="g")
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff607f48490>
```

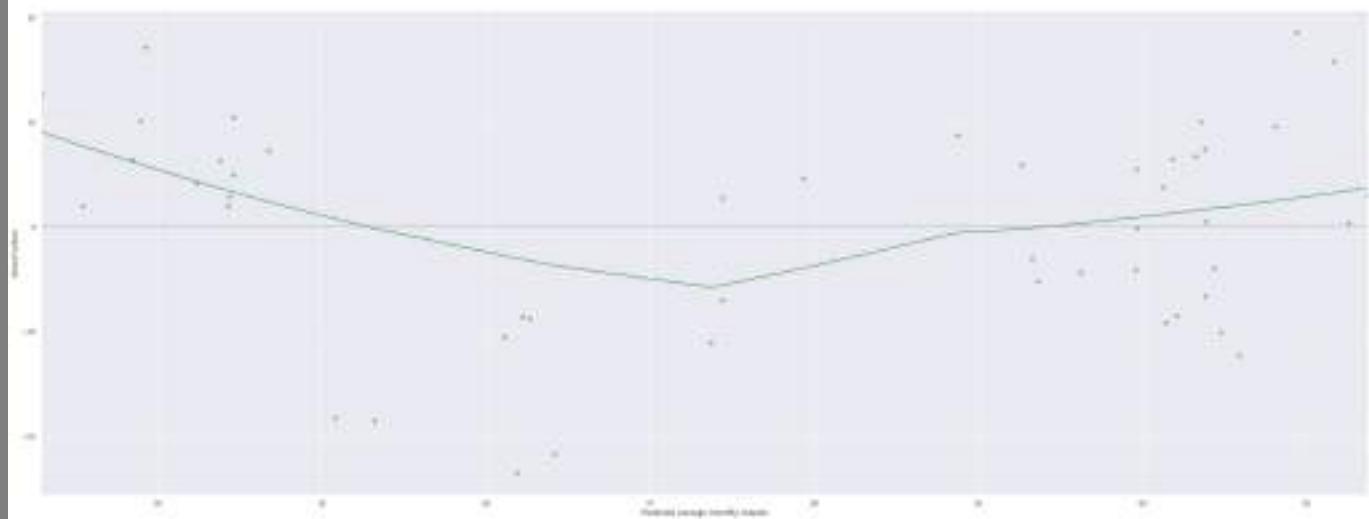


As one can observe this residual plot, one may notice the slight humps in the observations, which form a nonlinear pattern. However, the residuals are spread out. This indicates homoscedasticity, a lack of bias and constant variance. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS predicted logarithmic average monthly values versus actual values
plt.suptitle("Predicted average monthly logarithmic other renewable energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
```

```
sns.residplot(x = Otherrenewable_Logpred, y = Price_Actual, lowess = True, color="g")
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60820dad0>



As one can observe, there is a subtle hump along the lowess line in the residual plot. Furthermore, the observations are quite spread out in no particular pattern which indicates constant variance and a lack of bias. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly other renewable output versus the average monthly prices of energy per EUR/MWH.

In []:

In []:

```
modelotherrenewable = stats.linregress([70.66030013642565, 70.51488095238095, 66.63257065948856, 69.700557103064, 64.9490188172043, 56.383854166666666, 55.522462987886975, 58.35408333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.75208333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.40559722222224, 60.182429530201404, 62.58105555555558, 67.5951344086021, 79.4920833333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.25822222222224, 55.252580645161316, 54.08432795698931, 55.81655555555558, 63.92528859060403, 65.43065277777781, 65.15127688172035, 56.51197580645163, 60.877098214285674, 48.279717362045766, 50.40073611111113, 61.633763440860214,
```

```
64.34813888888884,  
67.78344086021498,  
70.36391129032262,  
76.9140416666666,  
70.36221476510062,  
67.0426075268817,  
66.62351388888881])
```

```
In [ ]: #Linear OLS regression  
otherrenewable1 = other_renewable  
otherrenewable1 = sm.add_constant(otherrenewable1)  
modelotherrenewablereg = sm.OLS(Price_Actual,otherrenewable1).fit()  
predictionsotherrenewable = modelotherrenewablereg.predict(otherrenewable1)  
  
modelotherrenewablereg.summary()  
#OLS Linear Summary Table
```

```
Out[ ]: OLS Regression Results  
Dep. Variable: y R-squared: 0.069  
Model: OLS Adj. R-squared: 0.049  
Method: Least Squares F-statistic: 3.426  
Date: Wed, 30 Nov 2022 Prob (F-statistic): 0.0706  
Time: 05:25:33 Log-Likelihood: -177.92  
No. Observations: 48 AIC: 359.8  
Df Residuals: 46 BIC: 363.6  
Df Model: 1  
Covariance Type: nonrobust  
  
coef std err t P>|t| [0.025 0.975]  
const 38.4104 10.607 3.621 0.001 17.059 59.762  
x1 0.2271 0.123 1.851 0.071 -0.020 0.474  
  
Omnibus: 1.865 Durbin-Watson: 0.436  
Prob(Omnibus): 0.394 Jarque-Bera (JB): 1.772  
Skew: -0.441 Prob(JB): 0.412  
Kurtosis: 2.671 Cond. No. 631.
```

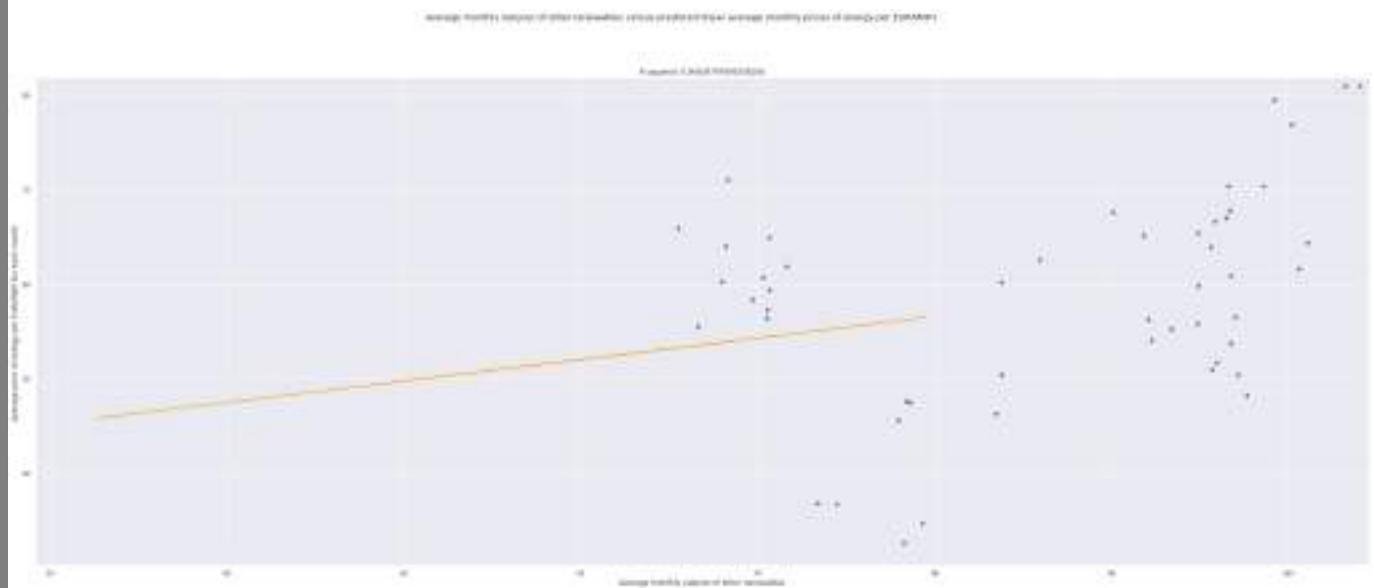
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [ ]:
```

```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(other_renewable,Price_Actual)  
print("slope: %f intercept: %f" % (slope, intercept))  
  
#OLS Linear Scatterplot  
plt.plot(other_renewable,Price_Actual, "o")  
plt.title(f"R squared: {modelotherrenewable.rvalue**2}")  
f = lambda x: 0.227125 *x + 38.410415  
plt.plot(x,f(x), c="orange", label="line of best fit")  
plt.legend("#")  
plt.suptitle('Average monthly outputs of other renewables versus predicted linear average monthly prices of energy')  
plt.ylabel('Average price of energy per EUR/MWh for each month ')  
plt.xlabel('Average monthly outputs of other renewables')  
plt.show()
```

```
slope: 0.227125 intercept: 38.410415
```



There is a very weak and positive correlation between the output and their average monthly price per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: print(predictionsotherrenewable)
#Linear OLS Predicted Values
```

```
[54.45914064 54.4261123 53.5443423 54.24115894 54.43769097 53.28979218
53.92879649 53.8949109 53.84734895 54.46119719 54.38014651 54.67782785
56.11579113 55.32079022 55.08194541 56.18862989 56.41740345 57.37157297
56.26862041 56.2238587 57.4416032 57.43922779 57.93339207 58.87212482
60.94219889 59.9666797 60.14289114 60.2077973 60.38384976 59.32989581
59.62310288 59.36209221 59.95827517 60.12966907 59.96489965 59.26409873
60.43757827 60.3864555 60.58735931 60.47593106 61.25693807 61.3749678
60.37735269 60.35423802 61.16708527 60.80891022 60.31946713 60.18755752]
```

```
In [ ]: #Linear OLS Predicted Values
influenceOtherreg = modelotherrenewablereg.get_influence()

standardized_residualsOther = influenceOtherreg.resid_studentized_internal

print(standardized_residualsOther)
```

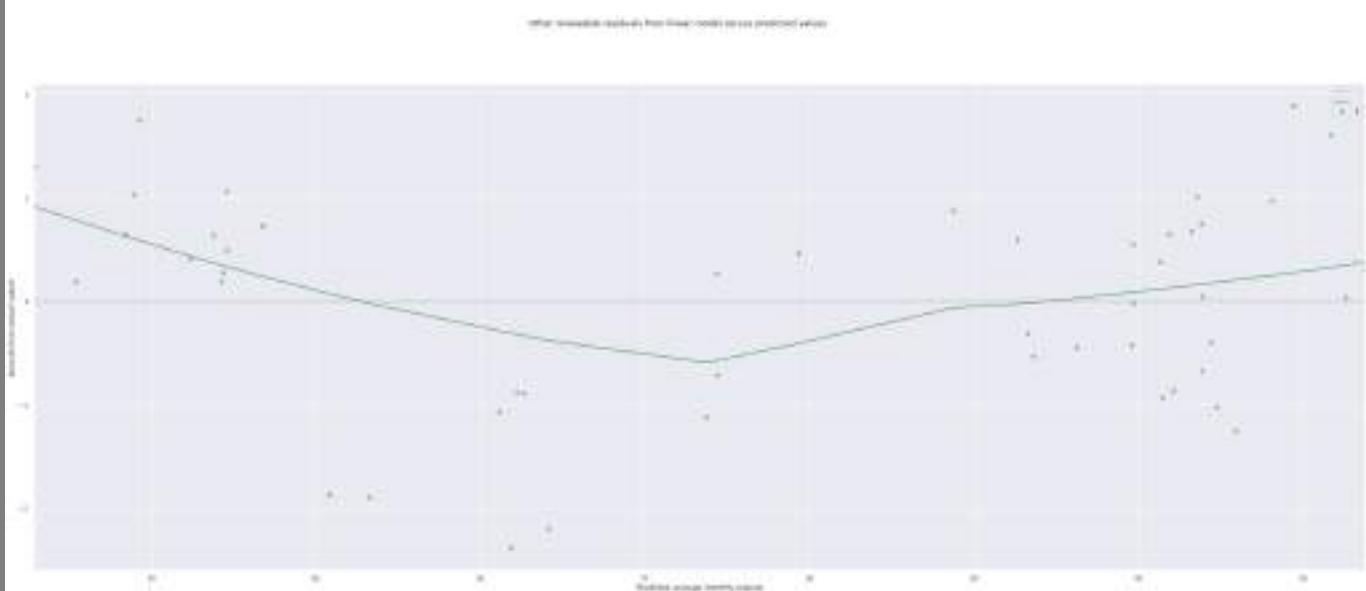
```
[ 1.07173466  0.20008828  0.20431036  0.42119401  0.29189581  1.3148206
 1.76196653  1.03882106  0.65920866  0.50527028  0.64896446  0.7364449
 -1.06284522 -1.88251627 -1.85515545 -2.37656007 -2.18832498 -1.11552308
 -0.88359491 -0.86915788 -0.70670097  0.27552976  0.46669786  0.87724287
 1.88928248 -0.01302703 -0.92926357 -0.85951039 -0.67017714 -0.30942696
 -0.4408867 -0.53173425 -0.4186111  0.38405904  0.5524565  0.59288256
 -0.39810014  0.0497372 -1.24962417 -1.02204342  0.03849822  0.3041311
 0.75071479  1.01444982  1.60730781  0.97181016  0.68119389  0.65148013]
```

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot( x = predictionsotherrenewable, y = standardized_residualsOther, lowess = True , color = 'g' )

plt.suptitle("Other renewable residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

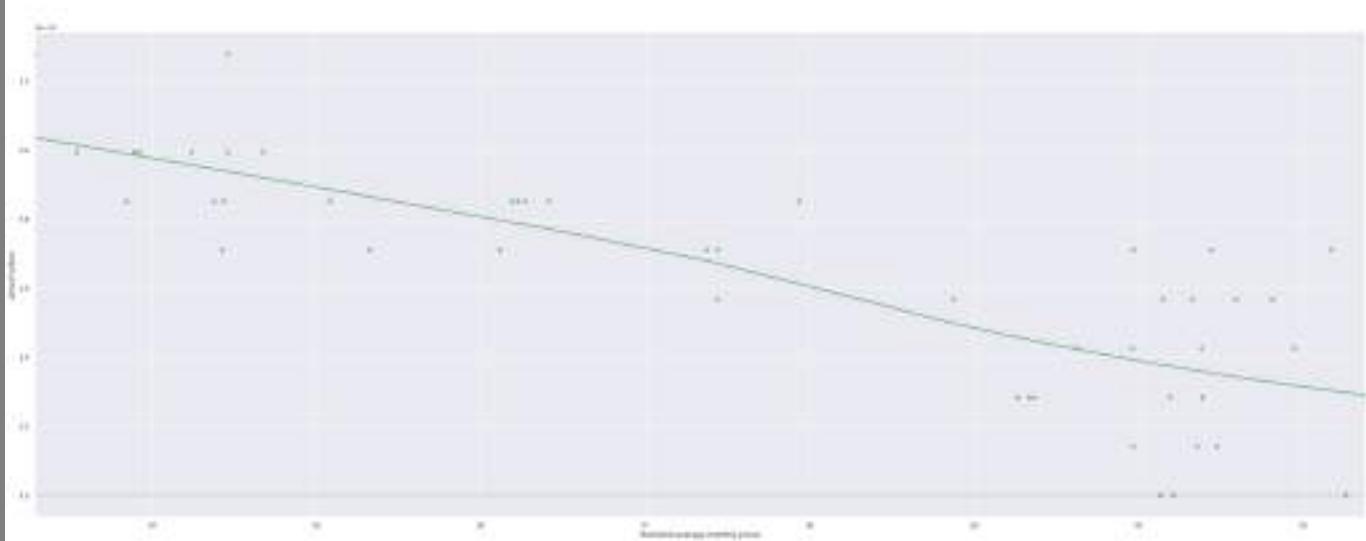
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff608679bd0>
```



As one can observe this residual plot, one may notice the slight humps in the observations, which form a nonlinear pattern. However, the residuals are spread out. This indicates homoscedasticity, a lack of bias and constant variance. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly linear other renewable outputs per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsotherrenewable, y = other_renewable, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6087b9d10>
```



As one can observe this residual plot, one may notice the negative slope in the fitted model. In addition, the observations are spread out in a distinct pattern, indicating bias and homoscedasticity. It would be reasonable to assume that this model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly other renewable output versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)
```

```

modelOtherrenewablequad = np.poly1d(np.polyfit(other_renewable,Price_Actual,2))
print(modelOtherrenewablequad)

otherrenewable1 = other_renewable

otherrenewable1 = sm.add_constant(otherrenewable1)
otherrenewable2 = polynomial_features.fit_transform(otherrenewable1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(otherrenewable1)

Otherrenewable_Q = poly.fit(X_poly, Price_Actual)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
Otherrenewable_Quad = sm.OLS(Price_Actual, otherrenewable2).fit()

# OLS Predicted Quadratic values
Otherrenewable_ypred = Otherrenewable_Quad.predict(otherrenewable2)

#OLS Quadratic Summary Table
Otherrenewable_Quad.summary()

##OLS Quadratic Scatterplot
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelOtherrenewablequad(polyline))
plt.scatter(other_renewable , Price_Actual, color = 'blue')
plt.title("R squared : 0.350")
plt.suptitle('Quadratic for predicted average monthly prices of energy per EUR/MWH versus average monthly outputs of other renewables')
plt.xlabel('Average monthly outputs of other renewables')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()

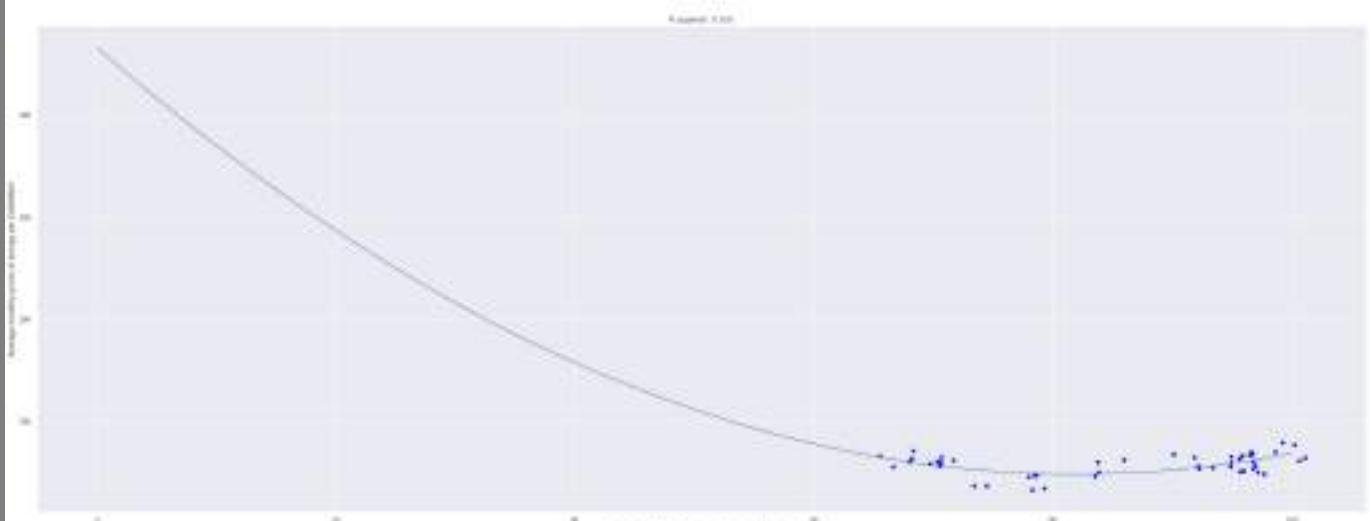
influenceOtherrenewableQuad = Otherrenewable_Quad.get_influence() #Quadratic OLS residuals

standardized_residualsOtherrenewableQuad = influenceOtherrenewableQuad.resid_studentized_internal

print(standardized_residualsOtherrenewableQuad)

```

$$0.06228x^2 - 10.2x + 465.9$$



```
[ 1.07087157  0.00900344 -0.91368909  0.10344507  0.12798552  0.13228305
 1.40050411  0.49821777 -0.00849827  0.40131747  0.50066594  0.85519542
-0.44248681 -1.80249614 -1.91848884 -2.00113833 -1.70564898 -0.26333126
-0.17236007 -0.17009828  0.23566443  1.43400557  1.62495806  1.83112335
 1.53465655  0.1218637 -1.09695937 -1.06614745 -0.98953759  0.179919
-0.15051923 -0.10314731 -0.35240063  0.46829168  0.79298998  1.29338327
-0.71414673 -0.13862756 -1.86261248 -1.48849605 -1.04835312 -0.87146085
 0.69987849  1.03198071  0.96339008  0.5674936   0.66639828  0.73908984]
```

The blue dots represent the observations and the blue line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

In []: Otherrenewable_Quad.summary()

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.350			
Model:	OLS	Adj. R-squared:	0.322			
Method:	Least Squares	F-statistic:	12.14			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	6.08e-05			
Time:	05:25:35	Log-Likelihood:	-169.29			
No. Observations:	48	AIC:	344.6			
Df Residuals:	45	BIC:	350.2			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	155.2962	32.425	4.789	0.000	89.988	220.604
x1	155.2962	32.425	4.789	0.000	89.988	220.604
x2	-5.1002	1.183	-4.313	0.000	-7.482	-2.718
x3	155.2962	32.425	4.789	0.000	89.988	220.604
x4	-5.1002	1.183	-4.313	0.000	-7.482	-2.718
x5	0.0623	0.014	4.413	0.000	0.034	0.091
Omnibus:	1.871	Durbin-Watson:		0.625		
Prob(Omnibus):	0.392	Jarque-Bera (JB):		1.610		
Skew:	-0.309	Prob(JB):		0.447		
Kurtosis:	2.349	Cond. No.		1.50e+22		

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 1.27e-35. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
print(Otherrenewable_ypred) # OLS quadratic predicted values
```

```
[56.10488125 56.30954547 62.74746947 57.5043094 56.23749695 64.95521139
59.70967054 59.96307779 60.32343828 56.0922247 56.59876224 54.81622167
49.21896006 51.69633393 52.73875759 49.06830226 48.67842365 48.41512555
48.91761211 49.00003254 48.48240065 48.47992465 49.28843754 52.44838021
66.93675769 58.81993637 60.11604947 60.61235859 62.00977389 54.76117478
56.5084002 54.94288726 58.75999073 60.01619332 58.80722587 54.39760914
62.45114973 62.03101906 63.71839354 62.77047915 70.04586212 71.27347413
61.95687336 61.76949525 69.13386736 65.69218671 61.49005654 60.45650233]
```

In []:

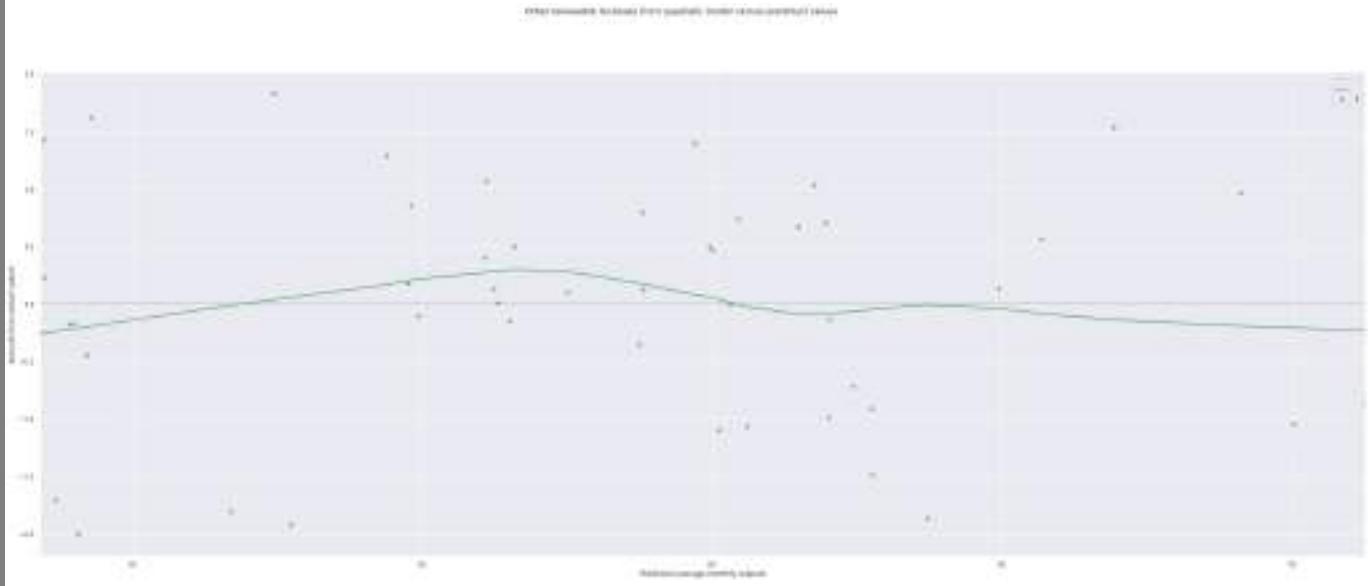
```
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = Otherrenewable_ypred, y = standardized_residualsOtherrenewableQuad, lowess = True, color="g")

plt.suptitle("Other renewable residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

Out[]:

```
<matplotlib.legend.Legend at 0x7ff609774e10>
```



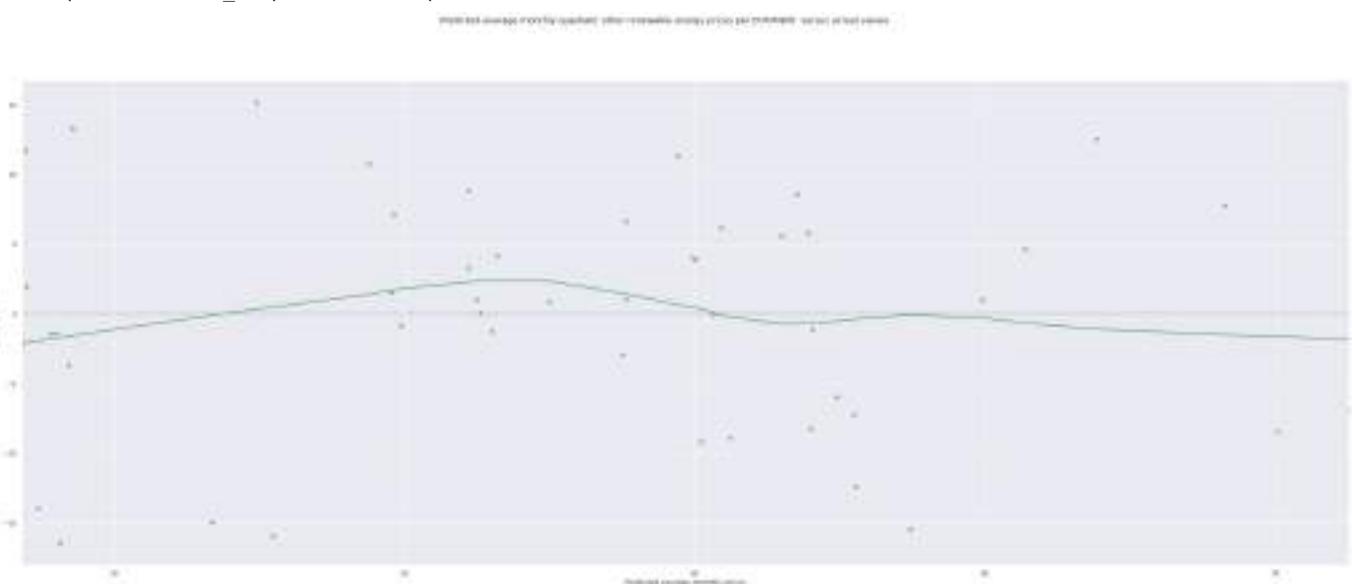
In []:

As one can observe this residual plot, one may notice the slight humps in the observations, which form a nonlinear pattern. However, the residuals are spread out. This indicates homoscedasticity, a lack of bias and constant variance. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []:

```
plt.suptitle("Predicted average monthly quadratic other renewable energy prices per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Otherrenewable_ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

Out[]:



As one can observe this residual plot, one may notice the slight humps in the observations, which form a nonlinear pattern. However, the residuals are spread out. This indicates homoscedasticity, a lack of bias and constant variance. The green dots represent the respective

observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

In []:

```
#Dataframes analyzed by resource
dfotherrenewable = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.4306527777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Other_Renewable': [70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306407, 70.5658602150!print(dfotherrenewable)
df_otherrenewable= pd.DataFrame.from_dict(dfotherrenewable, orient = "columns")
print(df_otherrenewable)
df_otherrenewable["Ratio"] = df_otherrenewable["Other_Renewable"]/df_otherrenewable["Price"]
pdToListOther = list(df_otherrenewable["Ratio"])
print(pdToListOther)
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")
adfuller_test(df_otherrenewable["Ratio"])

test_result=adfuller(df_otherrenewable["Ratio"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_otherrenewable["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Other Renewable Ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_otherrenewable["Ratio"])
plt.suptitle("Partialatocorrelations of Other Renewable Ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show

df_otherrenewable['First Difference Ratio'] = df_otherrenewable["Ratio"]- df_otherrenewable["Ratio"].shift(1) #
df_otherrenewable['Seasonal Difference Ratio']=df_otherrenewable["Ratio"]- df_otherrenewable["Ratio"].shift(12)
df_otherrenewable.head()
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.4306527777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Other_Renewable': [70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306407, 70.5658602150!print(dfotherrenewable)
df_otherrenewable= pd.DataFrame.from_dict(dfotherrenewable, orient = "columns")
print(df_otherrenewable)
df_otherrenewable["Ratio"] = df_otherrenewable["Other_Renewable"]/df_otherrenewable["Price"]
pdToListOther = list(df_otherrenewable["Ratio"])
print(pdToListOther)
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")
adfuller_test(df_otherrenewable["Ratio"])

test_result=adfuller(df_otherrenewable["Ratio"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_otherrenewable["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Other Renewable Ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_otherrenewable["Ratio"])
plt.suptitle("Partialatocorrelations of Other Renewable Ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show

df_otherrenewable['First Difference Ratio'] = df_otherrenewable["Ratio"]- df_otherrenewable["Ratio"].shift(1) #
df_otherrenewable['Seasonal Difference Ratio']=df_otherrenewable["Ratio"]- df_otherrenewable["Ratio"].shift(12)
df_otherrenewable.head()}
```

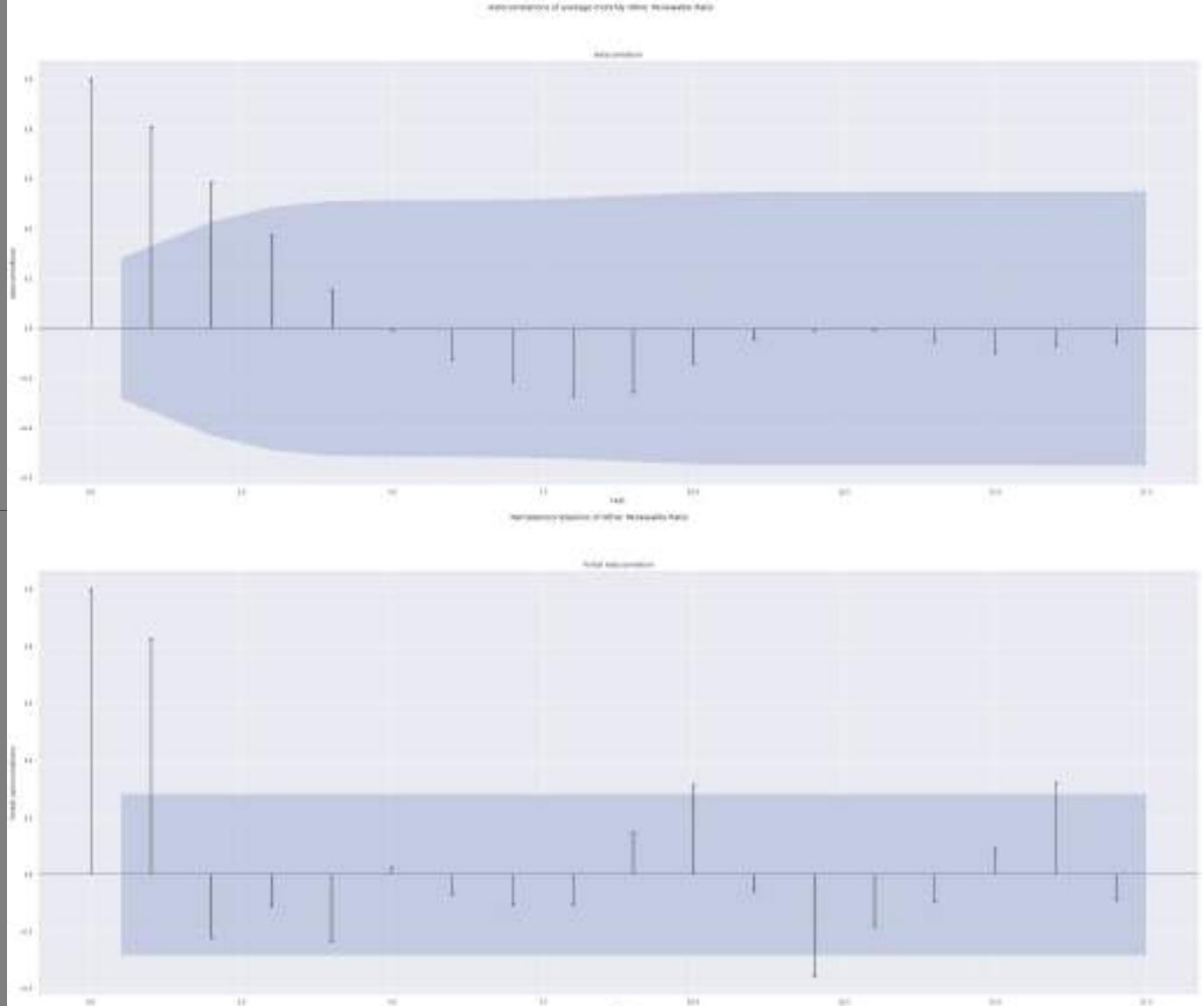
	Price	Other_Renewable	Dates
0	64.949019	70.660300	2015-01
1	56.383854	70.514881	2015-02
2	55.522463	66.632571	2015-03
3	58.354083	69.700557	2015-04
4	57.294059	70.565860	2015-05
5	65.974903	65.511822	2015-06

6	71.072043	68.325269	2015-07
7	63.998065	68.176075	2015-08
8	60.254792	67.966667	2015-09
9	59.406765	70.669355	2015-10
10	60.726792	70.312500	2015-11
11	61.901761	71.623149	2015-12
12	45.578723	77.954301	2016-01
13	36.752083	74.454023	2016-02
14	36.818008	73.402423	2016-03
15	32.618667	78.275000	2016-04
16	34.691371	79.282258	2016-05
17	46.266319	83.483333	2016-06
18	47.502016	78.627187	2016-07
19	47.602339	78.430108	2016-08
20	50.405597	83.791667	2016-09
21	60.182430	83.781208	2016-10
22	62.581056	85.956944	2016-11
23	67.595134	90.090054	2016-12
24	79.492083	99.204301	2017-01
25	59.837798	94.909226	2017-02
26	50.959892	95.685061	2017-03
27	51.717917	95.970833	2017-04
28	53.772621	96.745968	2017-05
29	56.258222	92.105556	2017-06
30	55.252581	93.396505	2017-07
31	54.084328	92.247312	2017-08
32	55.816556	94.872222	2017-09
33	63.925289	95.626846	2017-10
34	65.430653	94.901389	2017-11
35	65.151277	91.815860	2017-12
36	56.511976	96.982527	2018-01
37	60.877098	96.757440	2018-02
38	48.279717	97.641992	2018-03
39	50.400736	97.151389	2018-04
40	61.633763	100.590054	2018-05
41	64.348139	101.109722	2018-06
42	67.783441	96.717362	2018-07
43	70.363911	96.615591	2018-08
44	76.914042	100.194444	2018-09
45	70.362215	98.617450	2018-10
46	67.042608	95.881720	2018-11
47	66.623514	96.462500	2018-12

[1.0879348360180074, 1.2506218667483064, 1.2001011315730978, 1.1944418131789818, 1.2316435818046778, 0.9929809551311909, 0.9613522550191385, 1.065283392306891, 1.1279877464793602, 1.1895842959796394, 1.1578497409504178, 1.1570454301049409, 1.7103221797805228, 2.025844965392141, 1.9936554541672733, 2.3996995585349907, 2.285359611133195, 1.8044083544094802, 1.6552389453497844, 1.6476103832885234, 1.662348455018912, 1.3921207353659155, 1.3735297955806647, 1.332789032104891, 1.2479771181650434, 1.586108278829176, 1.877654292294365, 1.8556593056887123, 1.799167792099499, 1.637192785647135, 1.690355532461871, 1.7056200069882905, 1.6997147401507715, 1.4959157439242206, 1.4504117697129297, 1.4092718456115882, 1.7161411452658597, 1.5893898249815874, 2.0224226084925134, 1.9275787693797464, 1.6320608729330732, 1.571292099011756, 1.426858253555079, 1.3730844352755214, 1.3026807884920595, 1.40156829903176, 1.430160967287293, 1.4478746972258918]

ADF Test Statistic : -2.9200566693127272
p-value : 0.04306489847495319
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

	Price	Other_Renewable	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
0	64.949019	70.660300	2015-01	1.087935	NaN	NaN
1	56.383854	70.514881	2015-02	1.250622	0.162687	NaN
2	55.522463	66.632571	2015-03	1.200101	-0.050521	NaN
3	58.354083	69.700557	2015-04	1.194442	-0.005659	NaN
4	57.294059	70.565860	2015-05	1.231644	0.037202	NaN



```
In [ ]: Otherrenewable_Ratio_Autocorrelations = sm.tsa.acf(df_otherrenewable["Ratio"], fft=False) #Autocorrelations
print(Otherrenewable_Ratio_Autocorrelations)
```

```
[ 1.00000000e+00  8.08338684e-01  5.85664396e-01  3.75155513e-01
 1.51730385e-01 -8.04460448e-03 -1.21698231e-01 -2.12594081e-01
 -2.72535864e-01 -2.56112464e-01 -1.43448245e-01 -3.97944772e-02
 -6.86690054e-03 -6.11325310e-03 -5.14667308e-02 -9.64563667e-02
 -7.05159923e-02 -5.68939251e-02 -4.62779407e-02 -5.01295488e-02
 -5.52441881e-02 -1.67679296e-02  3.82513820e-02  9.59581881e-02
 6.03165821e-02 -2.06085012e-02 -8.01150999e-02 -1.54621471e-01
 -1.96350987e-01 -1.97838553e-01 -1.99878852e-01 -1.85908729e-01
 -1.46742587e-01 -1.00096980e-01 -5.93217303e-02 -2.82051089e-02
 -2.65216720e-02  8.78216787e-04 -4.55815649e-03  2.33842576e-02
 4.78189352e-02]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each

spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation and partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly other renewable outputs and the average monthly prices of energy per EUR/MWH.

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective average monthly prices of energy per EUR/MWH.

```
In [ ]: df_otherrenewable.describe(include = 'all') # Description Tables
```

	Price	Other_Renewable	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN
mean	57.859848	85.633141	NaN	1.524915	0.007658	0.145119
std	10.320573	11.963775	NaN	0.328195	0.194540	0.445335
min	32.618667	65.511822	NaN	0.961352	-0.480951	-0.544040
25%	51.528411	72.957604	NaN	1.249961	-0.077608	-0.128778
50%	59.622281	90.952957	NaN	1.473164	-0.021995	0.064965
75%	64.999583	96.500773	NaN	1.706796	0.057380	0.484713
max	79.492083	101.109722	NaN	2.399700	0.553277	1.205258

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly other renewable output to price of energy per EUR/MWH ratio")
plt.ylabel('Seasonality')
plt.xlabel('Months')
```

```
df_otherrenewable['Seasonal Difference Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60afecdd0>
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly other renewable outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + '\n')

    if result[1] <= 0.05:
```

```

        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_otherrenewable["Other_Renewable"])

test_result=adfuller(df_otherrenewable["Other_Renewable"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_otherrenewable["Other_Renewable"])
plt.suptitle(" Autocorrelations of average monthly Other Renewables")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_otherrenewable["Other_Renewable"])
plt.suptitle("Partial Autocorrelations of average monthly Other Renewables")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
otherrenewable_Autocorrelations = sm.tsa.acf(df_otherrenewable["Other_Renewable"], fft=False) #Autocorrelations
print(otherrenewable_Autocorrelations)

```

ADF Test Statistic : -1.2345999417034008

p-value : 0.6585374708946143

#Lags Used : 2

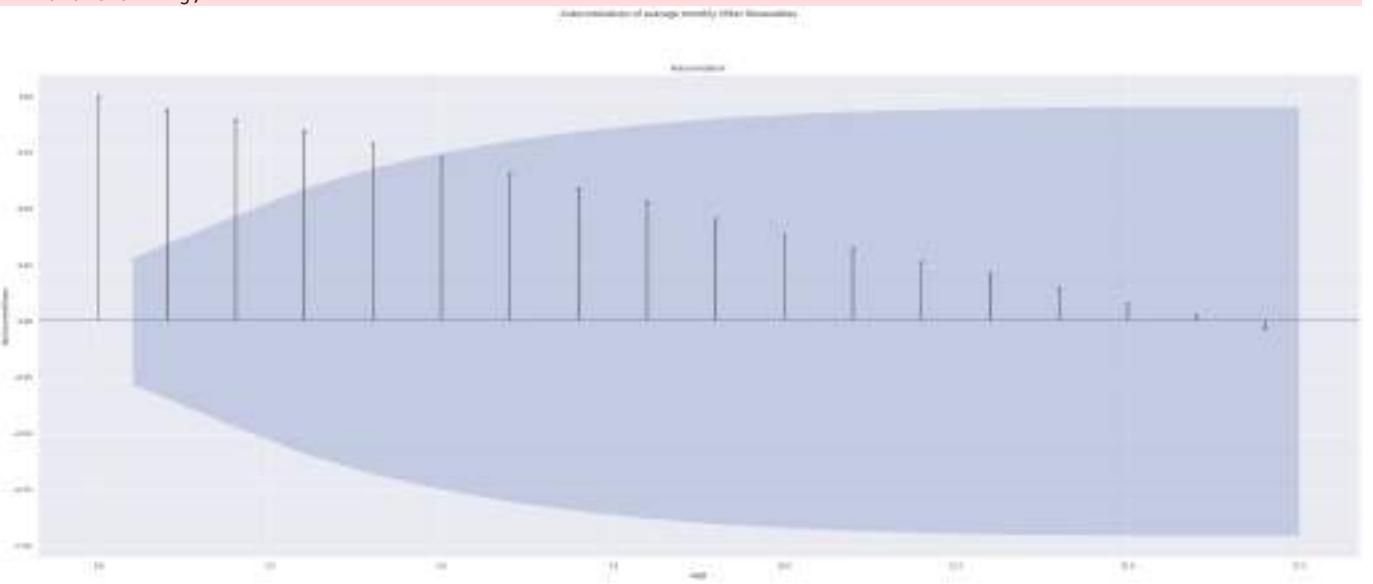
Number of Observations : 45

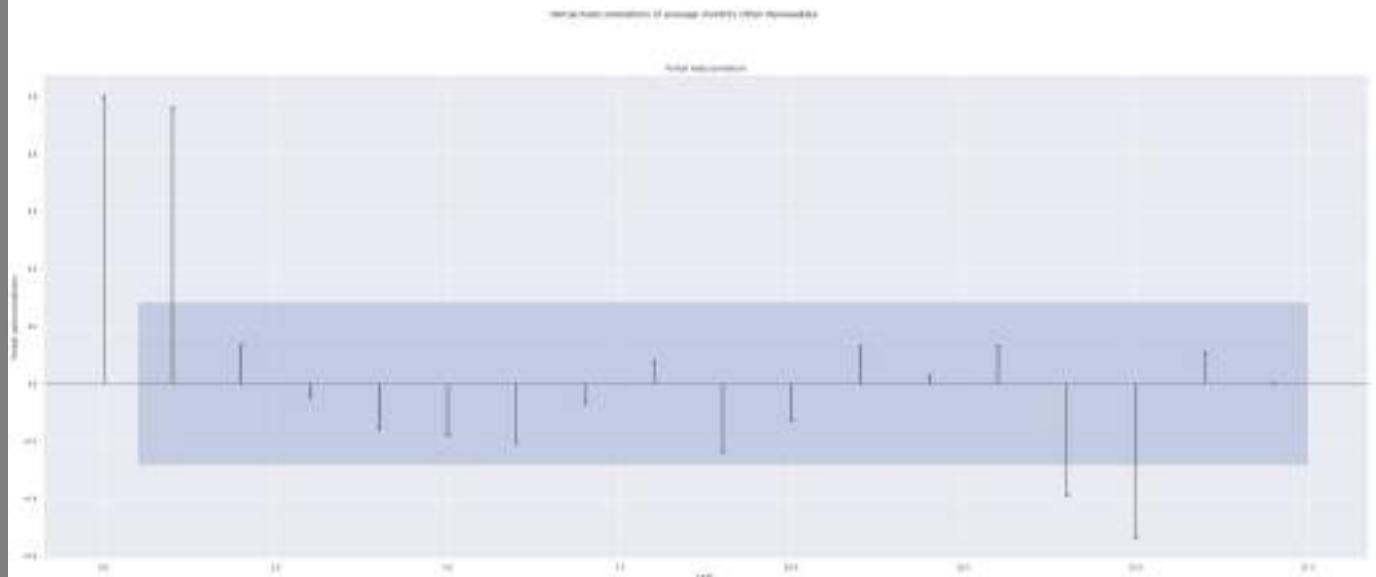
weak evidence against null hypothesis, indicating it is non-stationary

```
[ 1.      0.94005439  0.89334958  0.84400076  0.78725014  0.72448122
 0.65473394  0.58777813  0.52752608  0.45248732  0.38017258  0.32079616
 0.26196695  0.21231213  0.1454036   0.07465342  0.02542058  -0.03187081
-0.09081406 -0.14769177 -0.20026099 -0.2496017   -0.30878078 -0.34426981
-0.37392397 -0.38303187 -0.39210148 -0.38943438 -0.3883535  -0.37997718
-0.38739299 -0.39686237 -0.39381883 -0.3888731  -0.37635204 -0.36275166
-0.36128157 -0.33512127 -0.31134832 -0.28580589 -0.25130849]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,





```
In [ ]: Otherrenewable_Autocorrelations = sm.tsa.acf(df_otherrenewable["Other_Renewable"],fft=False) #Autocorrelations  
print(Otherrenewable_Autocorrelations)
```

```
[ 1.          0.94005439  0.89334958  0.84400076  0.78725014  0.72448122
 0.65473394  0.58777813  0.52752608  0.45248732  0.38017258  0.32079616
 0.26196695  0.21231213  0.1454036   0.07465342  0.02542058  -0.03187081
-0.09081406 -0.14769177 -0.20026099 -0.2496017   -0.30878078 -0.34426981
-0.37392397 -0.38303187 -0.39210148 -0.38943438 -0.3883535   -0.37997718
-0.38739299 -0.39686237 -0.39381883 -0.3888731   -0.37635204 -0.36275166
-0.36128157 -0.33512127 -0.31134832 -0.28580589 -0.25130849]
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the first two lags on the autocorrelation and partial autocorrelation plot, indicating that the lags directly beforehand influenced the average monthly other renewable outputs.

```
In [ ]: df_otherrenewable['First Difference'] = df_otherrenewable["Other_Renewable"]- df_otherrenewable["Other_Renewable"].shift(1)
df_otherrenewable['Seasonal Difference']=df_otherrenewable["Other_Renewable"]- df_otherrenewable["Other_Renewable"].shift(12)
df_otherrenewable.head()

plt.suptitle("Seasonal Difference of average monthly Other Renewable output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_otherrenewable['Seasonal Difference'].plot() # Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6082e4890>



In []:

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted in the average monthly other renewable outputs.

In []:

```
#Bell Curves

otherrenewableResults_mean = np.mean(df_otherrenewable["Ratio"])
otherrenewableResults_std = np.std(df_otherrenewable["Ratio"])

otherrenewableResultspdf = stats.norm.pdf(df_otherrenewable["Ratio"].sort_values(), otherrenewableResults_mean,
                                           otherrenewableResults_std)

plt.plot(df_otherrenewable["Ratio"].sort_values(), otherrenewableResultspdf)
plt.xlim([0,5])
plt.xlabel("Output to energy price per EUR/MWh", size=15)
plt.suptitle("Frequency distribution of average monthly other renewable outputs to energy prices per EUR/MWh raw")
plt.title(f'Skewness for data: {skew(df_otherrenewable["Ratio"])}' # Output/Price per EUR/MWh Bell Curve
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves

otherrenewableResults_mean = np.mean(df_otherrenewable["Other_Renewable"])
otherrenewableResults_std = np.std(df_otherrenewable["Other_Renewable"])

otherrenewableResultspdf = stats.norm.pdf(df_otherrenewable["Other_Renewable"].sort_values(), otherrenewableResults_mean,
                                           otherrenewableResults_std)

plt.plot(df_otherrenewable["Other_Renewable"].sort_values(), otherrenewableResultspdf)
plt.xlim([0,200])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_otherrenewable["Other_Renewable"])}')
plt.suptitle("Frequency distribution of average monthly other renewable outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



These bell shaped curves are roughly symmetrical, hence they have a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: df_otherrenewable.describe(include = 'all') # Description Tables

	Price	Other_Renewable	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	85.633141	NaN	1.524915	0.007658	0.145119	0.548983	9.557311
std	10.320573	11.963775	NaN	0.328195	0.194540	0.445335	3.130864	6.615029
min	32.618667	65.511822	NaN	0.961352	-0.480951	-0.544040	-5.054038	-2.221774
25%	51.528411	72.957604	NaN	1.249961	-0.077608	-0.128778	-0.888529	3.915378
50%	59.622281	90.952957	NaN	1.473164	-0.021995	0.064965	0.519668	8.830421
75%	64.999583	96.500773	NaN	1.706796	0.057380	0.484713	2.663799	14.988100
max	79.492083	101.109722	NaN	2.399700	0.553277	1.205258	9.114247	22.282638

Below is the box and whisker plot for the distribution of the average monthly outputs of other renewable and its the average monthly prices of energy per EUR/MWH.

In []: # Box and Whisker Plot
sns.set(style="darkgrid")

```

plt.suptitle('Distribution of average monthly outputs of other renewables by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly outputs of other renewables')
plt.xlabel('Average monthly price of energy per EUR/MWH')

sns.boxplot(x=Rounded_Y, y = other_renewable)
plt.show()

```



This box and whisker plot depicts the frequencies between the distribution of the monthly average output of other renewables produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of nuclear power produced, which was in between 95 and 100 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), the output of other renewables was slightly below 80 units. At approximately 64.35 EUR/MWH, nuclear power produced its highest output, which was slightly above 100 units. The lowest amount produced, whcih was slightly above 65 units, was at the price of approximately 65.97 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was slightly below 100 units.

```

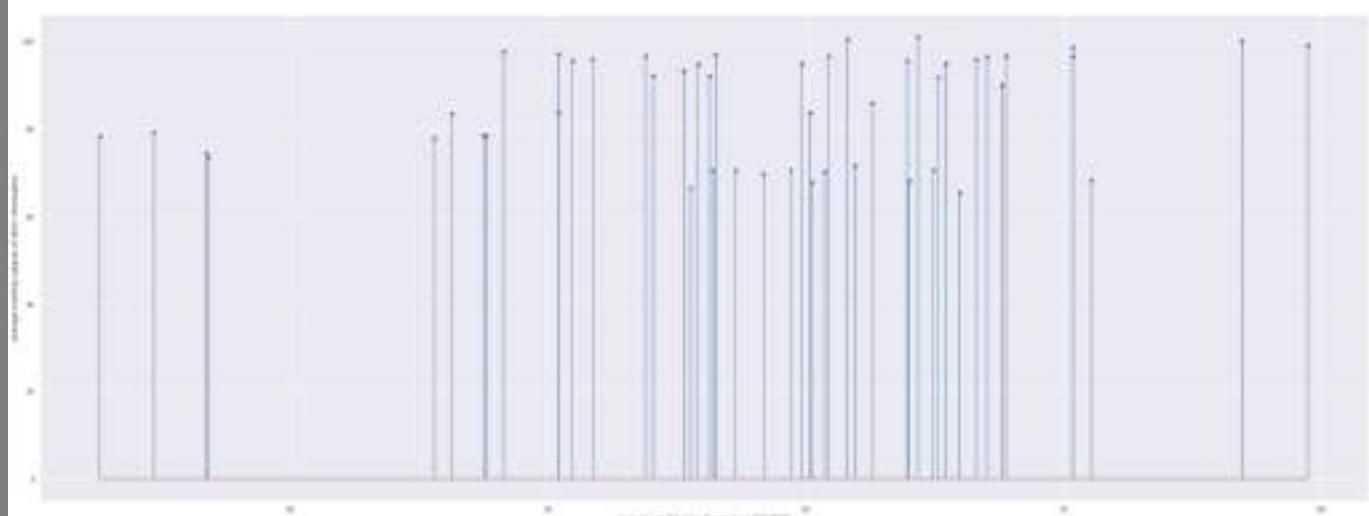
In [ ]: plt.suptitle('Distribution of average monthly outputs of other renewables by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly outputs of other renewables')
plt.xlabel('Average monthly price of energy per EUR/MWH')

# Stem Plot
plt.stem(Rounded_Y, other_renewable)

```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

```
Out[ ]: <StemContainer object of 3 artists>
```

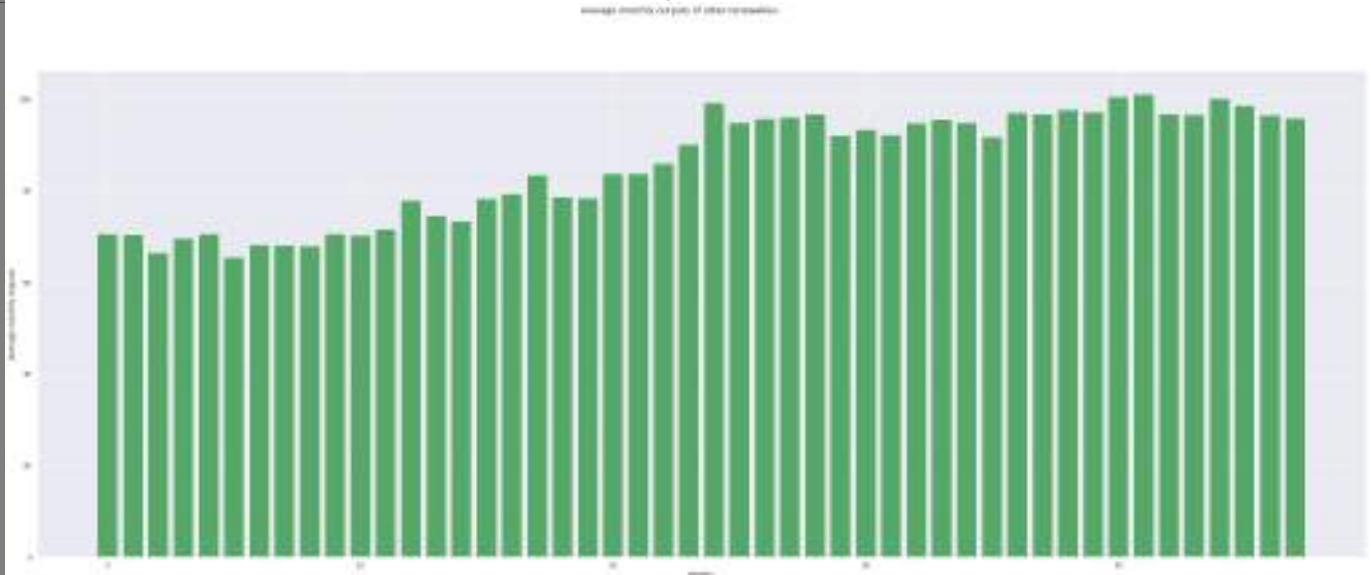


This plot depicts the frequencies between the distribution of the monthly average output of other renewables produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of nuclear power produced, which was in between 95 and 100 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price

of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), the output of other renewables was slightly below 80 units. At approximately 64.35 EUR/MWH, nuclear power produced its highest output, which was slightly above 100 units. The lowest amount produced, whcih was slightly above 65 units, was at the price of approximately 65.97 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was slightly below 100 units. Each blue dot and the end of the blue lines represent an observation.

```
In [ ]: #Outputs in MWH Histogram  
other_renewable_Dict = {key: i for i, key in enumerate(other_renewable)}  
  
def Hist_other_renewable(other_renewable_Dict):  
    for k, v in other_renewable_Dict.items(): print(f"{v}:{k}")  
print(other_renewable_Dict)  
  
plt.bar(list(other_renewable_Dict.values()), other_renewable_Dict.keys(), color='g')  
print(dicDates)  
plt.suptitle("Average monthly outputs of other renewables")  
plt.ylabel('Average monthly outputs')  
plt.xlabel('Months')  
  
plt.show()  
plt.show()
```

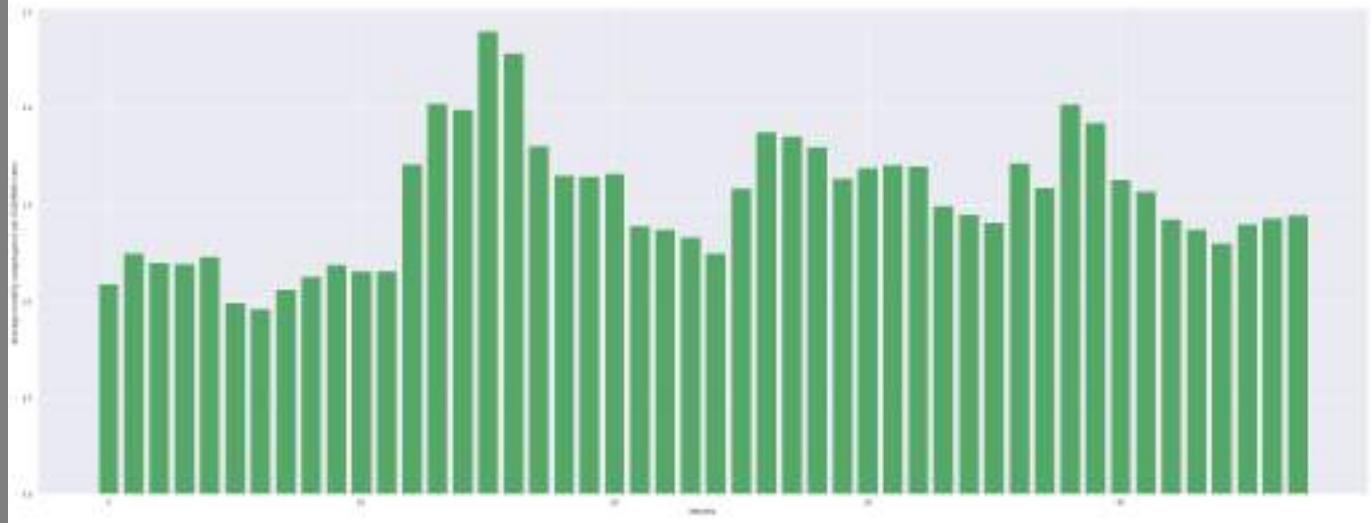
```
{70.66030013642565: 0, 70.51488095238095: 1, 66.63257065948856: 2, 69.70055710306407: 3, 70.56586021505376: 4, 6  
5.51182197496523: 5, 68.3252688172043: 6, 68.1760752688172: 7, 67.96666666666667: 8, 70.66935483870968: 9, 70.31  
25: 10, 71.62314939434724: 11, 77.95430107526882: 12, 74.45402298850574: 13, 73.40242261103634: 14, 78.275: 15,  
79.28225806451613: 16, 83.48333333333333: 17, 78.6271870794078: 18, 78.43010752688173: 19, 83.79166666666667: 20  
, 83.78120805369127: 21, 85.95694444444445: 22, 90.09005376344086: 23, 99.20430107526882: 24, 94.90922619047619:  
25, 95.6850605652759: 26, 95.97083333333333: 27, 96.74596774193549: 28, 92.10555555555555: 29, 93.39650537634408  
: 30, 92.24731182795699: 31, 94.87222222222222: 32, 95.6268456375839: 33, 94.90138888888889: 34, 91.815860215053  
76: 35, 96.98252688172043: 36, 96.75744047619048: 37, 97.64199192462988: 38, 97.15138888888889: 39, 100.59005376  
344086: 40, 101.10972222222222: 41, 96.71736204576042: 42, 96.61559139784946: 43, 100.19444444444444: 44, 98.617  
44966442953: 45, 96.4625: 46, 95.88172043010752: 47}  
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15  
-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18  
, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03  
' : 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is unimodal yet is skewed to the left, with a gradual increase in the average monthly outputs as the months progress.

```
In [ ]: pdToListOther_Dict = {key: i for i, key in enumerate(pdToListOther)}  
  
def Hist_pdToListOther(pdToListOther_Dict):  
    for k, v in pdToListOther_Dict.items(): print(f"{v}:{k}")  
#Output/price per EUR/MWH Histogram  
print(pdToListOther_Dict)  
  
plt.bar(list(pdToListOther_Dict.values()), pdToListOther_Dict.keys(), color='g')  
print(dicDates)  
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of other renewables")  
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')  
plt.xlabel('Months')  
  
plt.show()  
plt.show()
```

```
{1.0879348360180074: 0, 1.2506218667483064: 1, 1.2001011315730978: 2, 1.1944418131789818: 3, 1.2316435818046778: 4, 0.9929809551311909: 5, 0.9613522550191385: 6, 1.065283392306891: 7, 1.1279877464793602: 8, 1.1895842959796394: 9, 1.1578497409504178: 10, 1.1570454301049409: 11, 1.7103221797805228: 12, 2.025844965392141: 13, 1.9936554541: 672733: 14, 2.3996995585349907: 15, 2.285359611133195: 16, 1.8044083544094802: 17, 1.6552389453497844: 18, 1.647: 6103832885234: 19, 1.662348455018912: 20, 1.3921207353659155: 21, 1.3735297955806647: 22, 1.332789032104891: 23, 1.2479771181650434: 24, 1.586108278829176: 25, 1.877654292294365: 26, 1.8556593056887123: 27, 1.799167792099499: 28, 1.637192785647135: 29, 1.690355532461871: 30, 1.7056200069882905: 31, 1.6997147401507715: 32, 1.495915743924: 2206: 33, 1.4504117697129297: 34, 1.4092718456115882: 35, 1.7161411452658597: 36, 1.5893898249815874: 37, 2.0224: 226084925134: 38, 1.9275787693797464: 39, 1.6320608729330732: 40, 1.571292099011756: 41, 1.426858253555079: 42, 1.3730844352755214: 43, 1.3026807884920595: 44, 1.40156829903176: 45, 1.430160967287293: 46, 1.4478746972258918: 47}{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram displays a gradual increase in the output amount per EUR/MWH as the months progress; with a unimodal between months 10 and 20.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of other renewables.

In []:

```
In [ ]: modelotherrenewable = stats.linregress([70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306, 64.9490188172043, 56.38385416666666, 55.522462987886975, 58.35408333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.75208333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.40559722222224, 60.182429530201404, 62.58105555555558, 67.5951344086021, 79.4920833333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.25822222222224, 55.252580645161316, 54.08432795698931, 55.81655555555558,
```

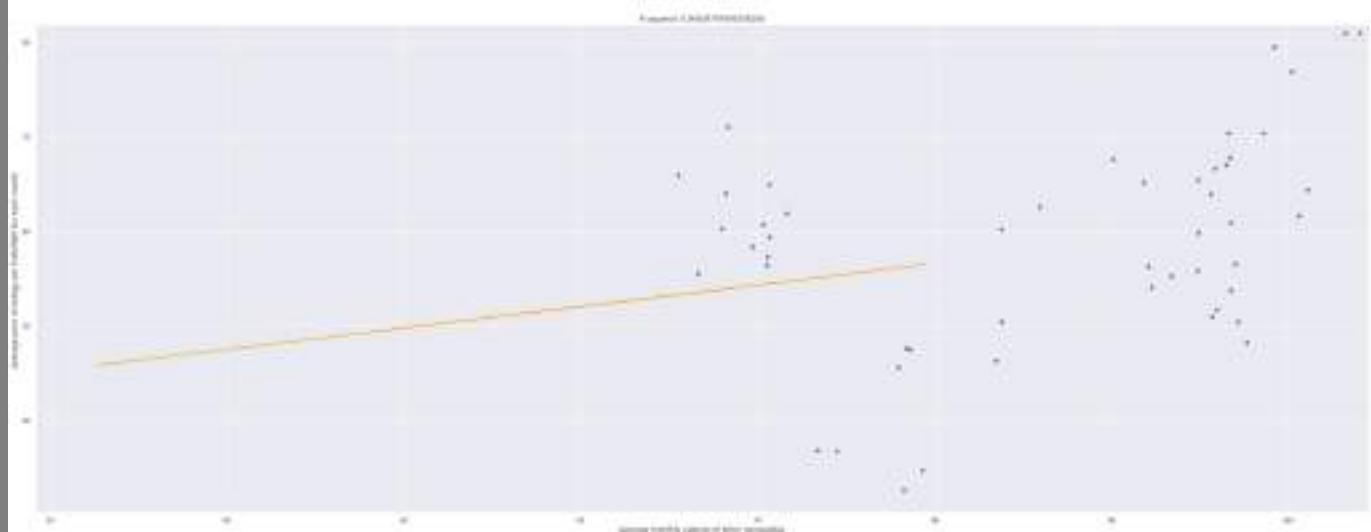
```
63.92528859060403,
65.43065277777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881])
```

In []:

```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(other_renewable,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(other_renewable,Price_Actual, "o")
plt.title(f"R squared: {modelotherrenewable.rvalue**2}")
f = lambda x: 0.227125 *x + 38.410415
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.legend("#")
plt.suptitle('Average monthly outputs of other renewables versus predicted linear average monthly prices of energy from MWh')
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly outputs of other renewables')
plt.show()
```

slope: 0.227125 intercept: 38.410415



There is a very weak and positive correlation between the output and their average monthly price per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

In []:

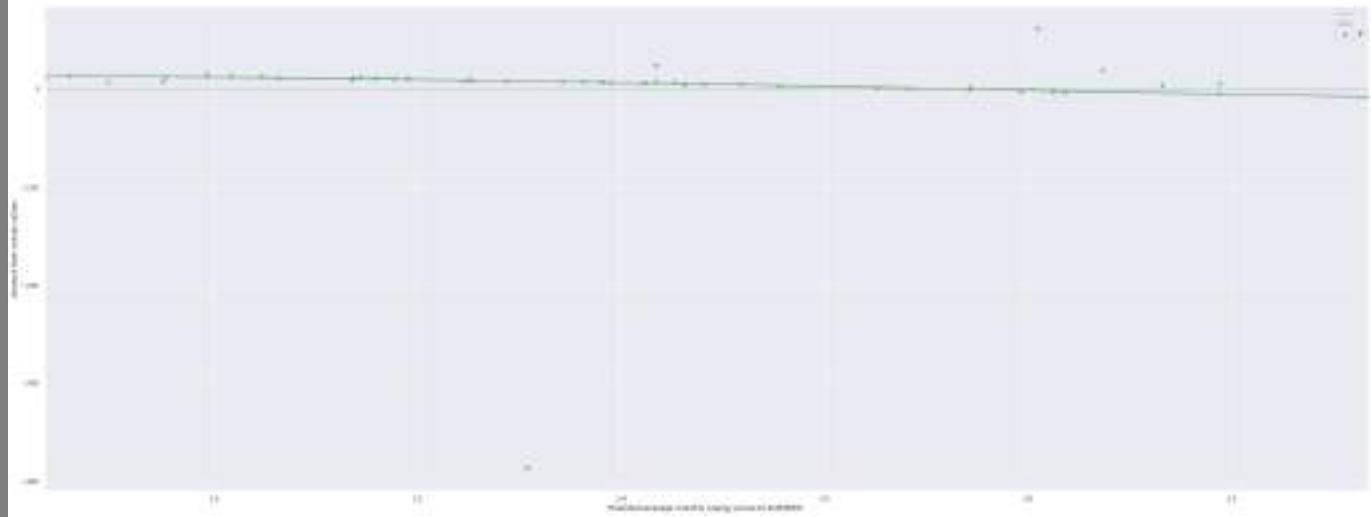
```
#OLS predicted quadratic average monthly ratios versus residuals
OtherQuadRatioPredict = Otherrenewable_ypred/ypred

sns.residplot(x = OtherQuadRatioPredict , y = standardized_residuals0therrenewableQuad/standardized_residualsPr

plt.suptitle("Predicted average monthly quadratic other renewable energy prices to EUR/MWH versus respective qua
plt.xlabel("Predicted average monthly energy prices to EUR/MWH ")
plt.ylabel("Amount from actual values")

plt.legend("..#")
```

Out[]: <matplotlib.legend.Legend at 0x7ff60ae0c9d0>



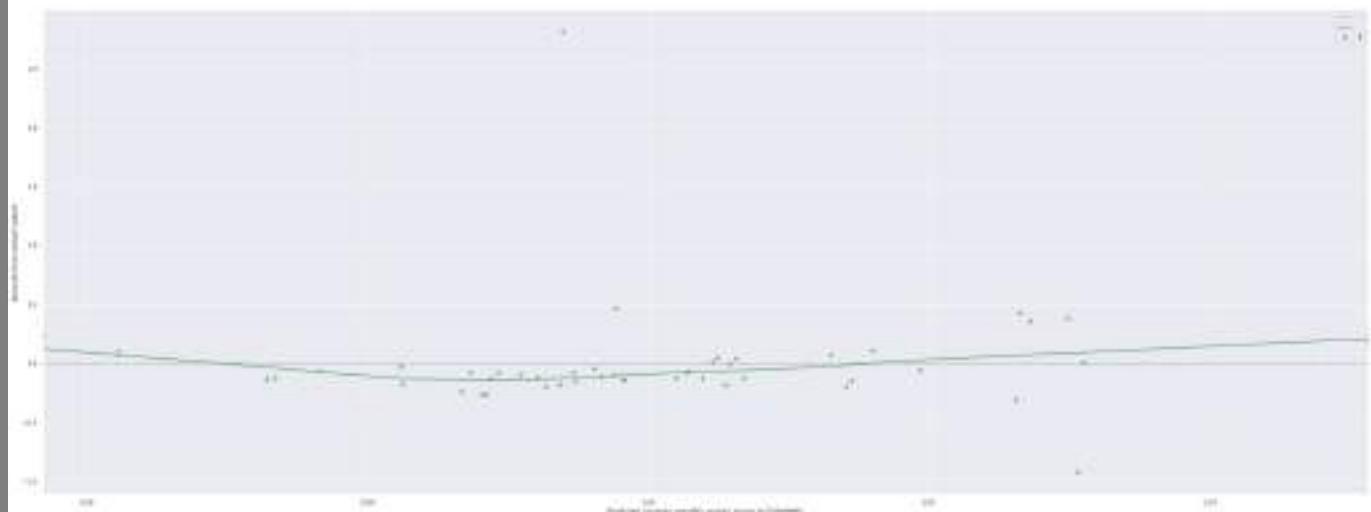
With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals
OtherRegRatioPredict = predictionsotherrenewable/predictions

sns.residplot(x = OtherRegRatioPredict, y = standardized_residualsOther/standardized_residualsPricereg, lowess = True)
plt.suptitle("Predicted average monthly linear other renewable energy prices to EUR/MWH versus respective linear prediction")
plt.xlabel("Predicted average monthly energy prices to EUR/MWH ")
plt.ylabel("Amount from actual values")

plt.legend(..#..)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60a326c10>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(predictions))
```

```
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895]
```

```
In [ ]: print(list(predictionsotherrenewable))
```

```
[54.45914064486219, 54.42611230233754, 53.54434230078673, 54.24115894011155, 54.43769097099995, 53.2897921760583, 53.92879649031573, 53.894910895021226, 53.8473489513556, 54.4611971947628, 54.38014651412594, 54.67782785108706, 56.1157911271614, 55.320790217614345, 55.081945407046355, 56.18862989326293, 56.41740345284586, 57.371572972237935, 56.26862040873618, 56.223858701343886, 57.44160320251326, 57.43922778929695, 57.933392071879126, 58.87212482008148, 60.94219888937942, 59.96667970051962, 60.14289113810686, 60.207797298388414, 60.38384975610322, 59.329895808045094, 59.62310287795981, 59.3620922115021, 59.95827517159661, 60.12966906851206, 59.964899652838874, 59.26409873321798, 60.43757826756118, 60.38645550168732, 60.58735931236163, 60.47593106295608, 61.256938067295096, 61.37496780297708, 60.377352685323785, 60.35423801967468, 61.1670852725652, 60.80891022093032, 60.31946712504365, 60.18755752390169]
```

```
In [ ]: dfOtherrenewableReg = ({ "Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], "OtherrenewableReg" : [54.45914064486219, 54.42611230233754, 53.54434230078673, 54.24115894011155, 54.43769097099995, 53.2897921760583, 53.92879649031573, 53.894910895021226, 53.8473489513556, 54.4611971947628, 54.38014651412594, 54.67782785108706, 56.1157911271614, 55.320790217614345, 55.081945407046355, 56.18862989326293, 56.41740345284586, 57.371572972237935, 56.26862040873618, 56.223858701343886, 57.44160320251326, 57.43922778929695, 57.933392071879126, 58.87212482008148, 60.94219888937942, 59.96667970051962, 60.14289113810686, 60.207797298388414, 60.38384975610322, 59.329895808045094, 59.62310287795981, 59.3620922115021, 59.95827517159661, 60.12966906851206, 59.964899652838874, 59.26409873321798, 60.43757826756118, 60.38645550168732, 60.58735931236163, 60.47593106295608, 61.256938067295096, 61.37496780297708, 60.377352685323785, 60.35423801967468, 61.1670852725652, 60.80891022093032, 60.31946712504365, 60.18755752390169])
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + ' ')
    
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_OtherrenewableReg["OtherrenewableReg"])

test_result=adfuller(df_OtherrenewableReg["OtherrenewableReg"])

df_OtherrenewableReg['First Difference'] = df_OtherrenewableReg["OtherrenewableReg"]- df_OtherrenewableReg["OtherrenewableReg"].shift(1)
df_OtherrenewableReg['Seasonal Difference']=df_OtherrenewableReg["OtherrenewableReg"]- df_OtherrenewableReg["OtherrenewableReg"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_OtherrenewableReg.head() #Predictive Linear Seasonality Plot
df_OtherrenewableReg['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'OtherrenewableReg': [54.45914064486219, 54.42611230233754, 53.54434230078673, 54.24115894011155, 54.43769097099995, 53.2897921760583, 53.92879649031573, 53.894910895021226, 53.8473489513556, 54.4611971947628, 54.38014651412594, 54.67782785108706, 56.1157911271614, 55.320790217614345, 55.081945407046355, 56.18862989326293, 56.41740345284586, 57.371572972237935, 56.26862040873618, 56.223858701343886, 57.44160320251326, 57.43922778929695, 57.933392071879126, 58.87212482008148, 60.94219888937942, 59.96667970051962, 60.14289113810686, 60.207797298388414, 60.38384975610322, 59.329895808045094, 59.62310287795981, 59.3620922115021, 59.95827517159661, 60.12966906851206, 59.964899652838874, 59.26409873321798, 60.43757826756118, 60.38645550168732, 60.58735931236163, 60.47593106295608, 61.256938067295096, 61.37496780297708, 60.377352685323785, 60.35423801967468, 61.1670852725652, 60.80891022093032, 60.31946712504365, 60.18755752390169], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	OtherrenewableReg	Dates
0	52.821613	54.459141	2015-01
1	53.036006	54.426112	2015-02
2	53.250399	53.544342	2015-03
3	53.464792	54.241159	2015-04
4	53.679185	54.437691	2015-05
5	53.893578	53.289792	2015-06
6	54.107971	53.928796	2015-07
7	54.322364	53.894911	2015-08
8	54.536757	53.847349	2015-09
9	54.751150	54.461197	2015-10
10	54.965543	54.380147	2015-11
11	55.179936	54.677828	2015-12
12	55.394329	56.115791	2016-01
13	55.608722	55.320790	2016-02
14	55.823115	55.081945	2016-03
15	56.037508	56.188630	2016-04
16	56.251901	56.417403	2016-05
17	56.466294	57.371573	2016-06
18	56.680687	56.268620	2016-07
19	56.895080	56.223859	2016-08
20	57.109473	57.441603	2016-09
21	57.323866	57.439228	2016-10
22	57.538259	57.933392	2016-11
23	57.752652	58.872125	2016-12
24	57.967045	60.942199	2017-01
25	58.181438	59.966680	2017-02
26	58.395831	60.142891	2017-03
27	58.610224	60.207797	2017-04
28	58.824617	60.383850	2017-05
29	59.039010	59.329896	2017-06
30	59.253403	59.623103	2017-07
31	59.467796	59.362092	2017-08
32	59.682189	59.958275	2017-09
33	59.896582	60.129669	2017-10
34	60.110975	59.964900	2017-11
35	60.325368	59.264099	2017-12
36	60.539761	60.437578	2018-01
37	60.754154	60.386456	2018-02
38	60.968547	60.587359	2018-03
39	61.182939	60.475931	2018-04
40	61.397332	61.256938	2018-05
41	61.611725	61.374968	2018-06
42	61.826118	60.377353	2018-07
43	62.040511	60.354238	2018-08
44	62.254904	61.167085	2018-09
45	62.469297	60.808910	2018-10
46	62.683690	60.319467	2018-11
47	62.898083	60.187558	2018-12

ADF Test Statistic : -1.2573280794556998

p-value : 0.6484617951433853

#Lags Used : 2

Number of Observations : 45

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a80acd0>



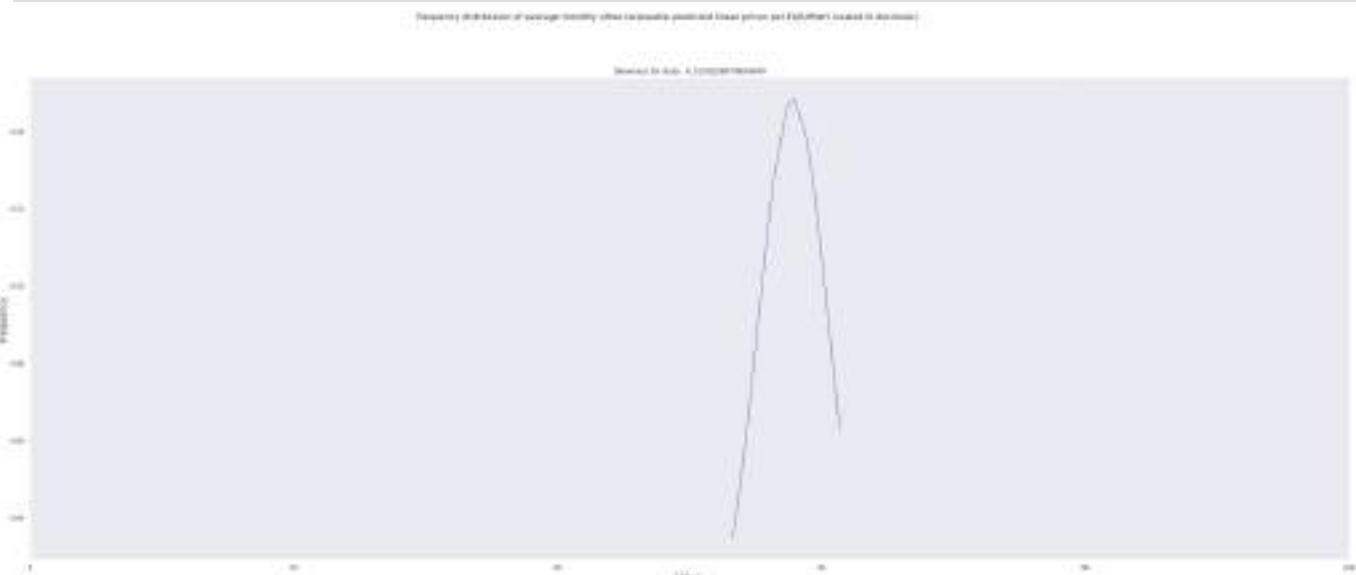
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
OtherrenewableRegResults_mean = np.mean(df_OtherrenewableReg["OtherrenewableReg"])
OtherrenewableRegResults_std = np.std(df_OtherrenewableReg["OtherrenewableReg"])

OtherrenewableRegResultspdf = stats.norm.pdf(df_OtherrenewableReg["OtherrenewableReg"].sort_values(), OtherrenewableRegResults_mean, OtherrenewableRegResults_std)

plt.plot(df_OtherrenewableReg["OtherrenewableReg"].sort_values(), OtherrenewableRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_OtherrenewableReg["OtherrenewableReg"])}')
plt.suptitle("Frequency distribution of average monthly other renewable predicted linear prices per EUR/MWH (scaled)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
OtherrenewableReg_Dict = {key: i for i, key in enumerate(df_OtherrenewableReg["OtherrenewableReg"])}

def Hist_OtherrenewableReg(OtherrenewableReg_Dict):
    for k, v in OtherrenewableReg_Dict.items(): print(f"{v}:{k}")
print(OtherrenewableReg_Dict)
```

```

plt.bar(list(OtherrenewableReg_Dict.values()), OtherrenewableReg_Dict.keys(), color='g') #Predictive Linear His
plt.title("Average monthly predicted other renewable prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

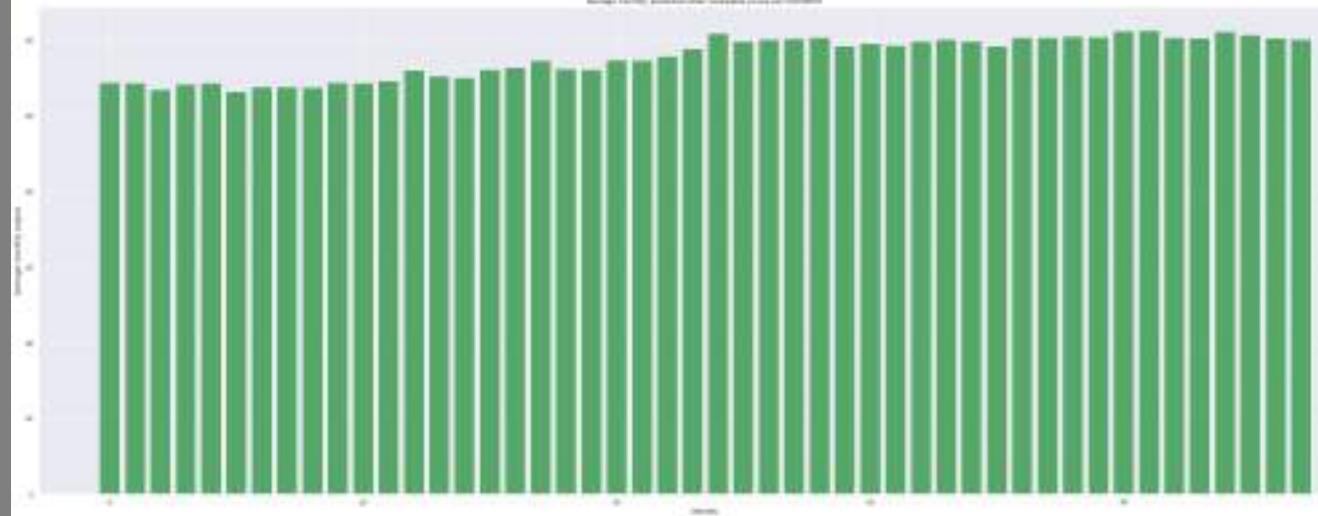
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{54.45914064486219: 0, 54.42611230233754: 1, 53.54434230078673: 2, 54.24115894011155: 3, 54.43769097099995: 4, 53.2897921760583: 5, 53.92879649031573: 6, 53.894910895021226: 7, 53.8473489513556: 8, 54.4611971947628: 9, 54.38014651412594: 10, 54.67782785108706: 11, 56.1157911271614: 12, 55.320790217614345: 13, 55.081945407046355: 14, 56.18862989326293: 15, 56.41740345284586: 16, 57.371572972237935: 17, 56.26862040873618: 18, 56.223858701343886: 19, 57.44160320251326: 20, 57.43922778929695: 21, 57.933392071879126: 22, 58.87212482008148: 23, 60.94219888937942: 24, 59.96667970051962: 25, 60.14289113810686: 26, 60.207797298388414: 27, 60.38384975610322: 28, 59.329895808045094: 29, 59.62310287795981: 30, 59.3620922115021: 31, 59.95827517159661: 32, 60.12966906851206: 33, 59.964899652838874: 34, 59.26409873321798: 35, 60.43757826756118: 36, 60.38645550168732: 37, 60.58735931236163: 38, 60.47593106295608: 39, 61.256938067295096: 40, 61.37496780297708: 41, 60.377352685323785: 42, 60.35423801967468: 43, 61.1670852725652: 44, 60.80891022093032: 45, 60.31946712504365: 46, 60.18755752390169: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform yet slightly increasing in value each month.

```
In [ ]: df_OtherrenewableReg.describe(include = 'all') # Description Tables
```

	Price	OtherrenewableReg	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	57.859848	NaN	0.121881	2.170705
std	3.001502	2.717273	NaN	0.709353	1.500680
min	52.821613	53.289792	NaN	-1.147899	-0.504621
25%	55.340731	54.980916	NaN	-0.201807	0.889281
50%	57.859848	59.068112	NaN	0.064906	2.005610
75%	60.378966	60.328160	NaN	0.605016	3.404173
max	62.898083	61.374968	NaN	2.070074	5.060946

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the predicted average monthly prices of energy per EUR/MWH for each respective resource; given all other variables constant.

```
In [ ]: print(list(Otherrenewable_Logpred))
```

```
print(list(Logpred))
```

```
[54.45914064486219, 54.42611230233754, 53.54434230078673, 54.24115894011155, 54.43769097099995, 53.2897921760583, 53.92879649031573, 53.894910895021226, 53.8473489513556, 54.4611971947628, 54.38014651412594, 54.67782785108706, 56.1157911271614, 55.320790217614345, 55.081945407046355, 56.18862989326293, 56.41740345284586, 57.371572972237935, 56.26862040873618, 56.223858701343886, 57.44160320251326, 57.43922778929695, 57.933392071879126, 58.87212482008148, 60.94219888937942, 59.96667970051962, 60.14289113810686, 60.207797298388414, 60.38384975610322, 59.329895808045094, 59.62310287795981, 59.3620922115021, 59.95827517159661, 60.12966906851206, 59.964899652838874, 59.26409873321798, 60.43757826756118, 60.38645550168732, 60.58735931236163, 60.47593106295608, 61.256938067295096, 61.37496780297708, 60.377352685323785, 60.35423801967468, 61.1670852725652, 60.80891022093032, 60.31946712504365, 60.18755752390169]
```

```
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: dfOtherrenewable_Log = ({ "Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005] })
```

```
#print(df0therrenewable_Log) #Dataframes
```

```
df_Otherrenewable_Log= pd.DataFrame.from_dict(df0therrenewable_Log, orient = "columns")
```

```
print(df_Otherrenewable_Log)
```



```
#ADF Tests
```

```
from statsmodels.tsa.stattools import adfuller
```

```
def adfuller_test(test_result):
```

```
    result=adfuller(test_result)
```

```
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
```

```
    for value,label in zip(result,labels):
```

```
        print(label+' : '+str(value) )
```



```
    if result[1] <= 0.05:
```

```
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
```

```
    else:
```

```
        print("weak evidence against null hypothesis,indicating it is non-stationary ")
```

```
adfuller_test(df_Otherrenewable_Log["Otherrenewable_Log"])
```



```
test_result=adfuller(df_Otherrenewable_Log["Otherrenewable_Log"])
```



```
df_Otherrenewable_Log['First Difference'] = df_Otherrenewable_Log["Otherrenewable_Log"]- df_Otherrenewable_Log.shift(1)
```

```
df_Otherrenewable_Log['Seasonal Difference']=df_Otherrenewable_Log["Otherrenewable_Log"]- df_Otherrenewable_Log.shift(12)
```

```
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic prices of energy per EUR/MWH")
```

```
plt.ylabel('Seasonality')
```

```
plt.xlabel('Months')
```

```
df_Otherrenewable_Log.head() #Predictive Logarithmic Seasonality Plot
```

```
df_Otherrenewable_Log['Seasonal Difference'].plot()
```

```
# Seasonality Plot
```

```
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Otherrenewable_Log': [54.45914064486219, 54.42611230233754, 53.54434230078673, 54.24115894011155, 54.43769097099995, 53.2897921760583, 53.92879649031573, 53.894910895021226, 53.8473489513556, 54.4611971947628, 54.38014651412594, 54.67782785108706, 56.1157911271614, 55.320790217614345, 55.081945407046355, 56.18862989326293, 56.41740345284586, 57.371572972237935, 56.26862040873618, 56.223858701343886, 57.44160320251326, 57.43922778929695, 57.933392071879126, 58.87212482008148, 60.94219888937942, 59.96667970051962, 60.14289113810686, 60.207797298388414, 60.38384975610322, 59.329895808045094, 59.62310287795981, 59.3620922115021, 59.95827517159661, 60.12966906851206, 59.964899652838874, 59.26409873321798, 60.43757826756118, 60.38645550168732, 60.58735931236163, 60.47593106295608, 61.256938067295096, 61.37496780297708, 60.377352685323785, 60.35423801967468, 61.1670852725652, 60.80891022093032, 60.31946712504365, 60.18755752390169], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Log	Otherrenewable_Log	Dates
0	27.297348	54.459141	2015-01
1	23.917581	54.426112	2015-02
2	23.577680	53.544342	2015-03
3	24.695022	54.241159	2015-04
4	24.276743	54.437691	2015-05
5	27.702157	53.289792	2015-06
6	29.713461	53.928796	2015-07
7	26.922107	53.894911	2015-08
8	25.445032	53.847349	2015-09
9	25.110405	54.461197	2015-10
10	25.631280	54.380147	2015-11
11	26.094917	54.677828	2015-12
12	19.653934	56.115791	2016-01
13	16.170990	55.320790	2016-02
14	16.197004	55.081945	2016-03
15	14.539966	56.188630	2016-04
16	15.357844	56.417403	2016-05
17	19.925256	57.371573	2016-06
18	20.412855	56.268620	2016-07
19	20.452442	56.223859	2016-08
20	21.558593	57.441603	2016-09
21	25.416478	57.439228	2016-10
22	26.362963	57.933392	2016-11
23	28.341491	58.872125	2016-12
24	33.035963	60.942199	2017-01
25	25.280488	59.966680	2017-02
26	21.777315	60.142891	2017-03
27	22.076427	60.207797	2017-04
28	22.887202	60.383850	2017-05
29	23.868007	59.329896	2017-06
30	23.471186	59.623103	2017-07
31	23.010200	59.362092	2017-08
32	23.693728	59.958275	2017-09
33	26.893390	60.129669	2017-10
34	27.487399	59.964900	2017-11
35	27.377158	59.264099	2017-12
36	23.968137	60.437578	2018-01
37	25.690591	60.386456	2018-02
38	20.719732	60.587359	2018-03
39	21.556675	60.475931	2018-04
40	25.989167	61.256938	2018-05
41	27.060244	61.374968	2018-06
42	28.415796	60.377353	2018-07
43	29.434036	60.354238	2018-08
44	32.018682	61.167085	2018-09
45	29.433366	60.808910	2018-10
46	27.958095	60.319467	2018-11
47	28.123467	60.187558	2018-12

ADF Test Statistic : -1.2573280794556998

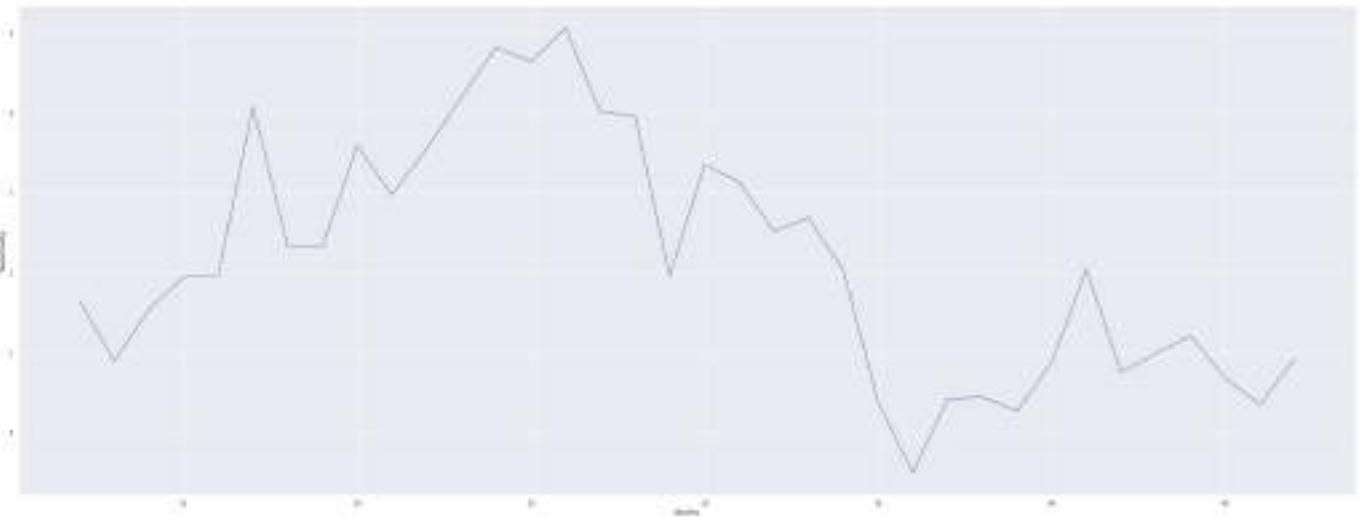
p-value : 0.6484617951433853

#Lags Used : 2

Number of Observations : 45

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609552690>



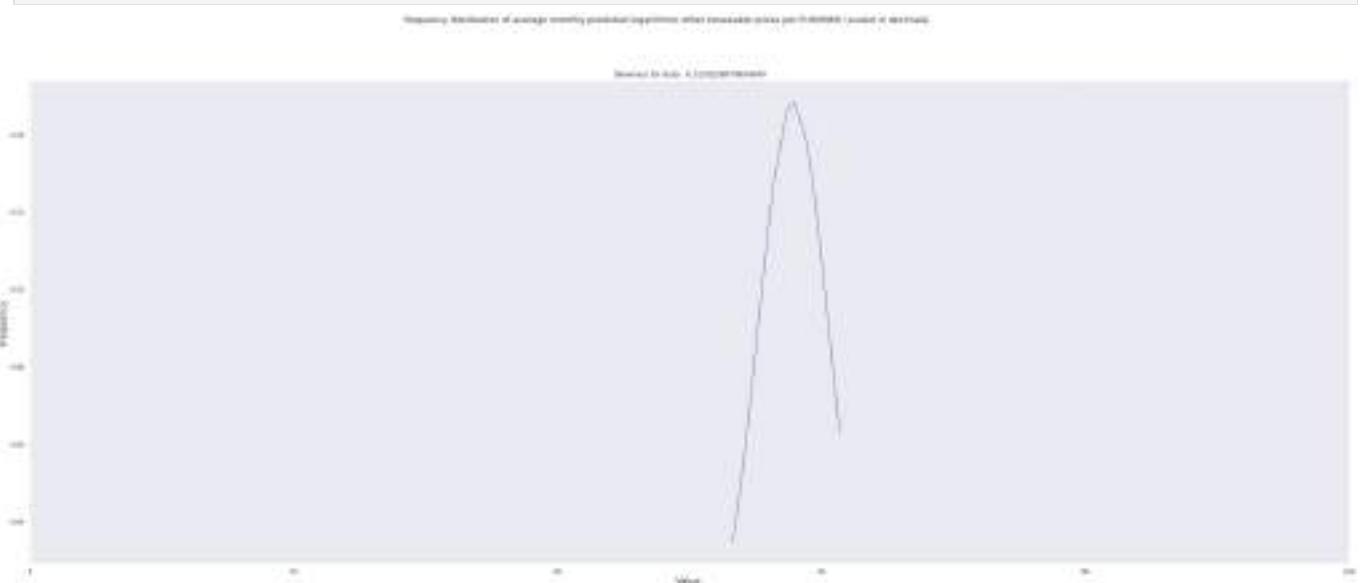
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWh for this particular resource and the predicted average monthly prices of energy per EUR/MWh.

In []: #Bell Curves

```
Otherrenewable_LogResults_mean = np.mean(df_Otherrenewable_Log["Otherrenewable_Log"])
Otherrenewable_LogResults_std = np.std(df_Otherrenewable_Log["Otherrenewable_Log"])

Otherrenewable_LogResultspdf = stats.norm.pdf(df_Otherrenewable_Log["Otherrenewable_Log"].sort_values(), Otherrenewable_LogResults_mean, Otherrenewable_LogResults_std)

plt.plot(df_Otherrenewable_Log["Otherrenewable_Log"].sort_values(), Otherrenewable_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Otherrenewable_Log["Otherrenewable_Log"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic other renewable prices per EUR/MWh")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetrical, hence it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)

```
Otherrenewable_Log_Dict = {key: i for i, key in enumerate(df_Otherrenewable_Log["Otherrenewable_Log"])}

def Hist_Otherrenewable_Log(Otherrenewable_Log_Dict):
    for k, v in Otherrenewable_Log_Dict.items(): print(f"{v}:{k}")
print(Otherrenewable_Log_Dict)
```

```

plt.bar(list(Otherrenewable_Log_Dict.values()), Otherrenewable_Log_Dict.keys(), color='g') #Predictive Logarithmic
plt.title("Average monthly output of other renewable predicted logarithmic prices per EUR/MWh")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

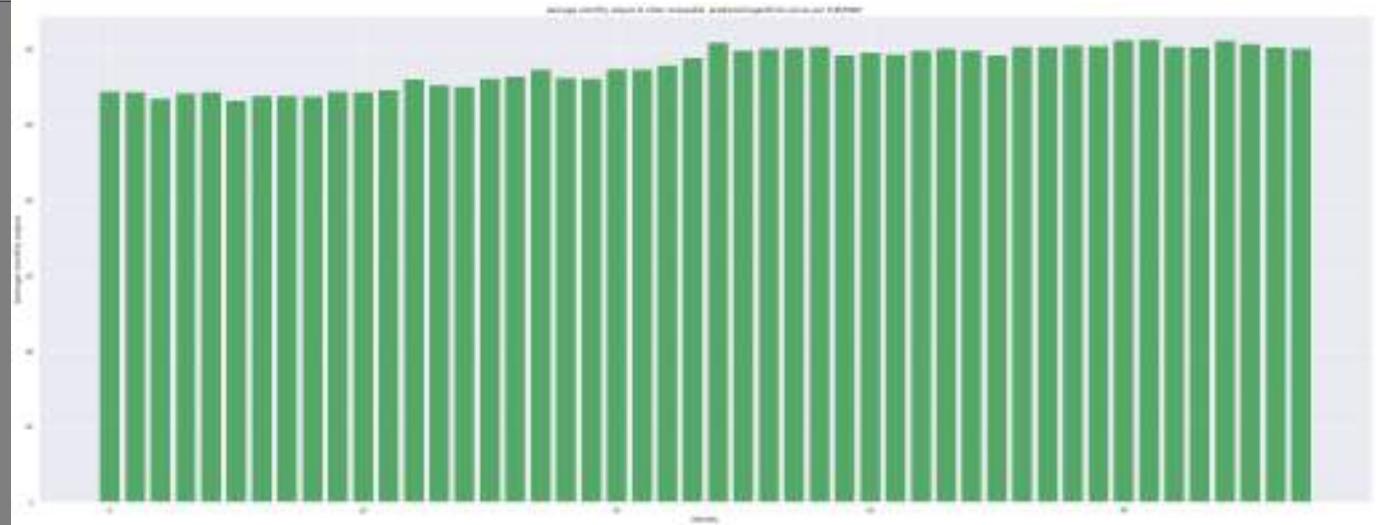
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-010': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{54.45914064486219: 0, 54.42611230233754: 1, 53.54434230078673: 2, 54.24115894011155: 3, 54.43769097099995: 4, 53.2897921760583: 5, 53.92879649031573: 6, 53.894910895021226: 7, 53.8473489513556: 8, 54.4611971947628: 9, 54.38014651412594: 10, 54.67782785108706: 11, 56.1157911271614: 12, 55.320790217614345: 13, 55.081945407046355: 14, 56.18862989326293: 15, 56.41740345284586: 16, 57.371572972237935: 17, 56.26862040873618: 18, 56.223858701343886: 19, 57.44160320251326: 20, 57.43922778929695: 21, 57.933392071879126: 22, 58.87212482008148: 23, 60.94219888937942: 24, 59.96667970051962: 25, 60.14289113810686: 26, 60.207797298388414: 27, 60.38384975610322: 28, 59.329895808045094: 29, 59.62310287795981: 30, 59.3620922115021: 31, 59.95827517159661: 32, 60.12966906851206: 33, 59.964899652838874: 34, 59.26409873321798: 35, 60.43757826756118: 36, 60.38645550168732: 37, 60.58735931236163: 38, 60.47593106295608: 39, 61.256938067295096: 40, 61.37496780297708: 41, 60.377352685323785: 42, 60.35423801967468: 43, 61.1670852725652: 44, 60.80891022093032: 45, 60.31946712504365: 46, 60.18755752390169: 47}

```



The green bars represent the observation value for each respective month. This histogram is mostly uniform throughout but is gradually increasing in value overtime.

```
In [ ]: df_Otherrenewable_Log.describe(include = 'all') # Description Tables
```

	Price_Log	Otherrenewable_Log	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	0.121881	2.170705
std	4.072442	2.717273	NaN	0.709353	1.500680
min	14.539966	53.289792	NaN	-1.147899	-0.504621
25%	22.001649	54.980916	NaN	-0.201807	0.889281
50%	25.195447	59.068112	NaN	0.064906	2.005610
75%	27.317301	60.328160	NaN	0.605016	3.404173
max	33.035963	61.374968	NaN	2.070074	5.060946

```

In [ ]: print(Otherrenewable_Logpred)
print(Otherrenewable_Logpred/Logpred)

print(Logpred)

Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price

```

```
print(Otherrenewable_ypred/ypred)

print(Otherrenewable_ypred)

print(ypred)

Rounded_ypred = [round(item, 2) for item in ypred]
print(Rounded_ypred)
#Rounded Price

print(Otherrenewable_ypred/ypred)

print(predictions)

print(predictionsotherrenewable)

Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price
```

[54.45914064 54.4261123 53.5443423 54.24115894 54.43769097 53.28979218
 53.92879649 53.8949109 53.84734895 54.46119719 54.38014651 54.67782785
 56.11579113 55.32079022 55.08194541 56.18862989 56.41740345 57.37157297
 56.26862041 56.2238587 57.4416032 57.43922779 57.93339207 58.87212482
 60.94219889 59.9666797 60.14289114 60.2077973 60.38384976 59.32989581
 59.62310288 59.36209221 59.95827517 60.12966907 59.96489965 59.26409873
 60.43757827 60.3864555 60.58735931 60.47593106 61.25693807 61.3749678
 60.37735269 60.35423802 61.16708527 60.80891022 60.31946713 60.18755752]
 [1.99503409 2.2755693 2.27097583 2.19644096 2.24238036 1.92366944
 1.81496182 2.0018831 2.11622251 2.16886972 2.12163207 2.09534402
 2.85519377 3.42098968 3.40074909 3.86442644 3.67352364 2.87933929
 2.75652863 2.74900465 2.66444121 2.25992082 2.19752963 2.0772416
 1.84472294 2.37205387 2.76172209 2.72724374 2.63832378 2.48574989
 2.54026797 2.57981643 2.53055474 2.23585309 2.18154146 2.16472791
 2.52158016 2.35052812 2.92413815 2.80543878 2.35701818 2.26808623
 2.12478133 2.05049143 1.91035614 2.06598558 2.1574956 2.14011868]
 [27.29734838 23.91758071 23.57768041 24.69502249 24.27674267 27.70215664
 29.71346062 26.92210695 25.44503172 25.11040502 25.63128037 26.09491682
 19.65393442 16.17099008 16.19700362 14.53996621 15.35784412 19.92525621
 20.41285544 20.45244219 21.55859284 25.41647801 26.36296287 28.34149128
 33.03596301 25.28048812 21.77731469 22.076427 22.88720221 23.86800702
 23.4711863 23.0102001 23.69372775 26.89338996 27.48739853 27.37715831
 23.96813682 25.69059052 20.71973214 21.55667467 25.98916653 27.0602444
 28.41579599 29.43403567 32.0186817 29.43336623 27.95809508 28.12346716]
 [27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.
 36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.
 69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
 [2.06358397 2.42446572 2.74267075 2.39052292 2.38267132 2.34286246
 1.95207222 2.24559944 2.42245255 2.28775902 2.25321425 2.13522833
 2.51806237 2.93172705 2.98903884 2.8736347 2.80919369 2.4530595
 2.4344937 2.43498162 2.30780751 1.94943264 1.89570512 1.83390403
 1.85754496 2.38020316 2.83637926 2.82493805 2.79304294 2.36301202
 2.48155742 2.46168351 2.55525302 2.25068132 2.14324779 1.99310992
 2.68283932 2.46263164 3.13450606 2.98816377 2.74210128 2.65155106
 2.1585431 2.0473489 2.01719487 2.17743806 2.1905297 2.13647316]
 [56.10488125 56.30954547 62.74746947 57.5043094 56.23749695 64.95521139
 59.70967054 59.96307779 60.32343828 56.0922247 56.59876224 54.81622167
 49.21896006 51.69633393 52.73875759 49.06830226 48.67842365 48.41512555
 48.91761211 49.00003254 48.48240065 48.47992465 49.28843754 52.44838021
 66.93675769 58.81993637 60.11604947 60.61235859 62.00977389 54.76117478
 56.5084002 54.94288726 58.75999073 60.01619332 58.80722587 54.39760914
 62.45114973 62.03101906 63.71839354 62.77047915 70.04586212 71.27347413
 61.95687336 61.76949525 69.13386736 65.69218671 61.49005654 60.45650233]
 [27.18807769 23.22554817 22.87823628 24.05511741 23.60270863 27.72472243
 30.58783893 26.70248166 24.90180391 24.51841487 25.11912136 25.67229968
 19.54636257 17.6334062 17.64405232 17.07534443 17.32825464 19.7366291
 20.09354637 20.12336851 21.00799158 24.86873542 26.00005506 28.59930479
 36.03506727 24.71214951 21.19464431 21.45617264 22.20151115 23.17431072
 22.77134503 22.31923277 22.99576214 26.66578905 27.43836997 27.29282952
 23.278006 25.18891499 20.32804924 21.00637179 25.54459336 26.87991763
 28.70309766 30.1704781 34.27227994 30.16948579 28.0708618 28.29733761]
 [27.19, 23.23, 22.88, 24.06, 23.6, 27.72, 30.59, 26.7, 24.9, 24.52, 25.12, 25.67, 19.55, 17.63, 17.64, 17.08, 17.
 .33, 19.74, 20.09, 20.12, 21.01, 24.87, 26.0, 28.6, 36.04, 24.71, 21.19, 21.46, 22.2, 23.17, 22.77, 22.32, 23.0,
 26.67, 27.44, 27.29, 23.28, 25.19, 20.33, 21.01, 25.54, 26.88, 28.7, 30.17, 34.27, 30.17, 28.07, 28.3]
 [2.06358397 2.42446572 2.74267075 2.39052292 2.38267132 2.34286246
 1.95207222 2.24559944 2.42245255 2.28775902 2.25321425 2.13522833
 2.51806237 2.93172705 2.98903884 2.8736347 2.80919369 2.4530595
 2.4344937 2.43498162 2.30780751 1.94943264 1.89570512 1.83390403
 1.85754496 2.38020316 2.83637926 2.82493805 2.79304294 2.36301202
 2.48155742 2.46168351 2.55525302 2.25068132 2.14324779 1.99310992
 2.68283932 2.46263164 3.13450606 2.98816377 2.74210128 2.65155106
 2.1585431 2.0473489 2.01719487 2.17743806 2.1905297 2.13647316]
 [52.82161316 53.03600615 53.25039913 53.46479211 53.67918509 53.89357808
 54.10797106 54.32236404 54.53675703 54.75115001 54.96554299 55.17993597
 55.39432896 55.60872194 55.82311492 56.03750791 56.25190089 56.46629387
 56.68068685 56.89507984 57.10947282 57.3238658 57.53825879 57.75265177
 57.96704475 58.18143773 58.39583072 58.6102237 58.82461668 59.03900967
 59.25340265 59.46779563 59.68218861 59.8965816 60.11097458 60.32536756
 60.53976055 60.75415353 60.96854651 61.18293949 61.39733248 61.61172546
 61.82611844 62.04051143 62.25490441 62.46929739 62.68369037 62.89808336]
 [54.45914064 54.4261123 53.5443423 54.24115894 54.43769097 53.28979218
 53.92879649 53.8949109 53.84734895 54.46119719 54.38014651 54.67782785
 56.11579113 55.32079022 55.08194541 56.18862989 56.41740345 57.37157297
 56.26862041 56.2238587 57.4416032 57.43922779 57.93339207 58.87212482
 60.94219889 59.9666797 60.14289114 60.2077973 60.38384976 59.32989581
 59.62310288 59.36209221 59.95827517 60.12966907 59.96489965 59.26409873
 60.43757827 60.3864555 60.58735931 60.47593106 61.25693807 61.3749678
 60.37735269 60.35423802 61.16708527 60.80891022 60.31946713 60.18755752]
 [52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04,
 56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59.
 .68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource;

given all other variables constant.

```
In [ ]: print(list(ypred))
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]

In [ ]: print(list(Otherrenewable_ypred))
[56.1048812487428, 56.309545474732715, 62.747469467313124, 57.50430940491407, 56.23749695211211, 64.95521138600515, 59.70967053614663, 59.963077785588894, 60.32343828384302, 56.09222469931012, 56.59876223558814, 54.81622167317096, 49.218960058849746, 51.696333933634435, 52.738757591906335, 49.06830225817225, 48.678423648153625, 48.41512554579555, 48.91761211149054, 49.00003253541041, 48.482400650324905, 48.47992465397948, 49.288437535342894, 52.44838020644494, 66.9367576939049, 58.81993636829844, 60.116049470018254, 60.61235858907003, 62.009773888742984, 54.76117478119096, 56.50840020050134, 54.942887260857276, 58.75999072888152, 60.01619331503406, 58.807225869519584, 54.397609142724036, 62.45114973178613, 62.03101906140358, 63.718393542498575, 62.77047914663564, 70.04586212318839, 71.27347412887764, 61.956873358603104, 61.76949525462885, 69.13386735791408, 65.6921867103315, 61.4900565431131, 60.45650233395156]

In [ ]:
dfOtherrenewable_Quad = {"Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]
#Dataframes
df_Otherrenewable_Quad= pd.DataFrame.from_dict(dfOtherrenewable_Quad, orient = "columns")
print(df_Otherrenewable_Quad)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + ' ')
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")
adfuller_test(df_Otherrenewable_Quad["Otherrenewable_Quad"])

test_result=adfuller(df_Otherrenewable_Quad["Otherrenewable_Quad"])

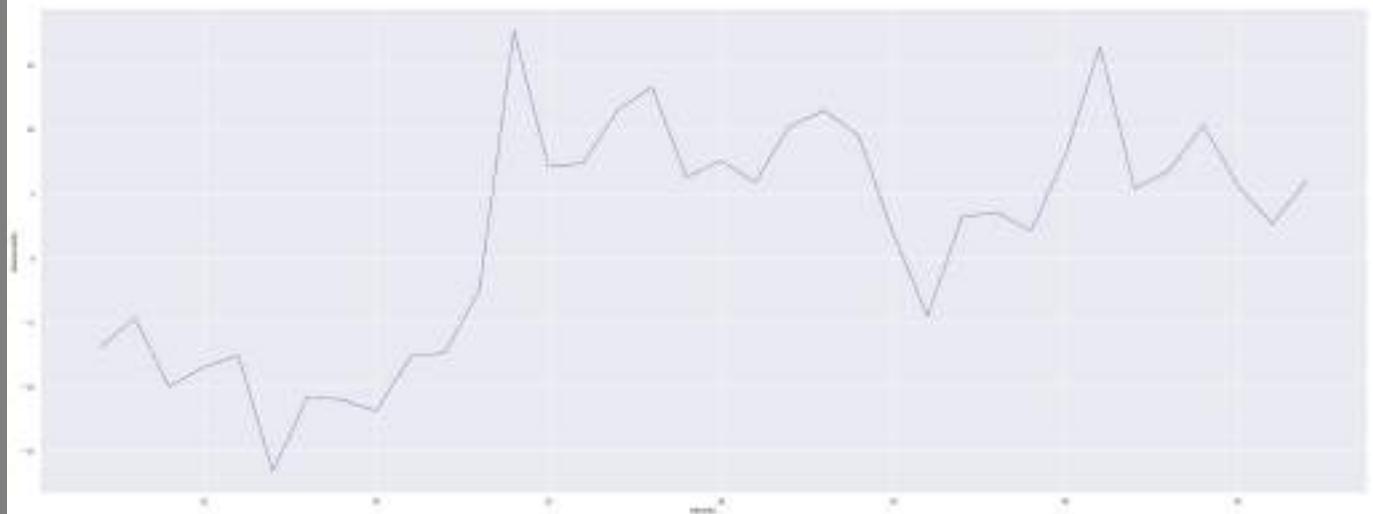
df_Otherrenewable_Quad['First Difference'] = df_Otherrenewable_Quad["Otherrenewable_Quad"]- df_Otherrenewable_Quad.shift(1)
df_Otherrenewable_Quad['Seasonal Difference']=df_Otherrenewable_Quad["Otherrenewable_Quad"]- df_Otherrenewable_Quad.shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Otherrenewable_Quad.head() #Predictive Quadratic Seasonality Plot
df_Otherrenewable_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Otherrenewable_Quad': [56.1048812487428, 56.309545474732715, 62.747469467313124, 57.50430940491407, 56.23749695211211, 64.95521138600515, 59.70967053614663, 59.963077785588894, 60.32343828384302, 56.09222469931012, 56.59876223558814, 54.81622167317096, 49.218960058849746, 51.696333933634435, 52.738757591906335, 49.06830225817225, 48.678423648153625, 48.41512554579555, 48.9176121149054, 49.00003253541041, 48.482400650324905, 48.47992465397948, 49.288437535342894, 52.44838020644494, 66.9367576939049, 58.81993636829844, 60.116049470018254, 60.61235858907003, 62.009773888742984, 54.76117478119096, 56.50840020050134, 54.942887260857276, 58.75999072888152, 60.01619331503406, 58.807225869519584, 54.397609142724036, 62.45114973178613, 62.03101906140358, 63.718393542498575, 62.77047914663564, 70.04586212318839, 71.27347412887764, 61.956873358603104, 61.76949525462885, 69.13386735791408, 65.6921867103315, 61.4900565431131, 60.45650233395156], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price_Quad Otherrenewable_Quad Dates
0    27.188078      56.104881 2015-01
1    23.225548      56.309545 2015-02
2    22.878236      62.747469 2015-03
3    24.055117      57.504309 2015-04
4    23.602709      56.237497 2015-05
5    27.724722      64.955211 2015-06
6    30.587839      59.709671 2015-07
7    26.702482      59.963078 2015-08
8    24.901804      60.323438 2015-09
9    24.518415      56.092225 2015-10
10   25.119121      56.598762 2015-11
11   25.672300      54.816222 2015-12
12   19.546363      49.218960 2016-01
13   17.633406      51.696334 2016-02
14   17.644052      52.738758 2016-03
15   17.075344      49.068302 2016-04
16   17.328255      48.678424 2016-05
17   19.736629      48.415126 2016-06
18   20.093546      48.917612 2016-07
19   20.123369      49.000033 2016-08
20   21.007992      48.482401 2016-09
21   24.868735      48.479925 2016-10
22   26.000055      49.288438 2016-11
23   28.599305      52.448380 2016-12
24   36.035067      66.936758 2017-01
25   24.712150      58.819936 2017-02
26   21.194644      60.116049 2017-03
27   21.456173      60.612359 2017-04
28   22.201511      62.009774 2017-05
29   23.174311      54.761175 2017-06
30   22.771345      56.508400 2017-07
31   22.319233      54.942887 2017-08
32   22.995762      58.759991 2017-09
33   26.665789      60.016193 2017-10
34   27.438370      58.807226 2017-11
35   27.292830      54.397609 2017-12
36   23.278006      62.451150 2018-01
37   25.188915      62.031019 2018-02
38   20.328049      63.718394 2018-03
39   21.006372      62.770479 2018-04
40   25.544593      70.045862 2018-05
41   26.879918      71.273474 2018-06
42   28.703098      61.956873 2018-07
43   30.170478      61.769495 2018-08
44   34.272280      69.133867 2018-09
45   30.169486      65.692187 2018-10
46   28.070862      61.490057 2018-11
47   28.297338      60.456502 2018-12

ADF Test Statistic : -2.61287380332136
p-value : 0.09036559836915842
#Lags Used : 0
Number of Observations : 47
weak evidence against null hypothesis, indicating it is non-stationary
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60778e250>



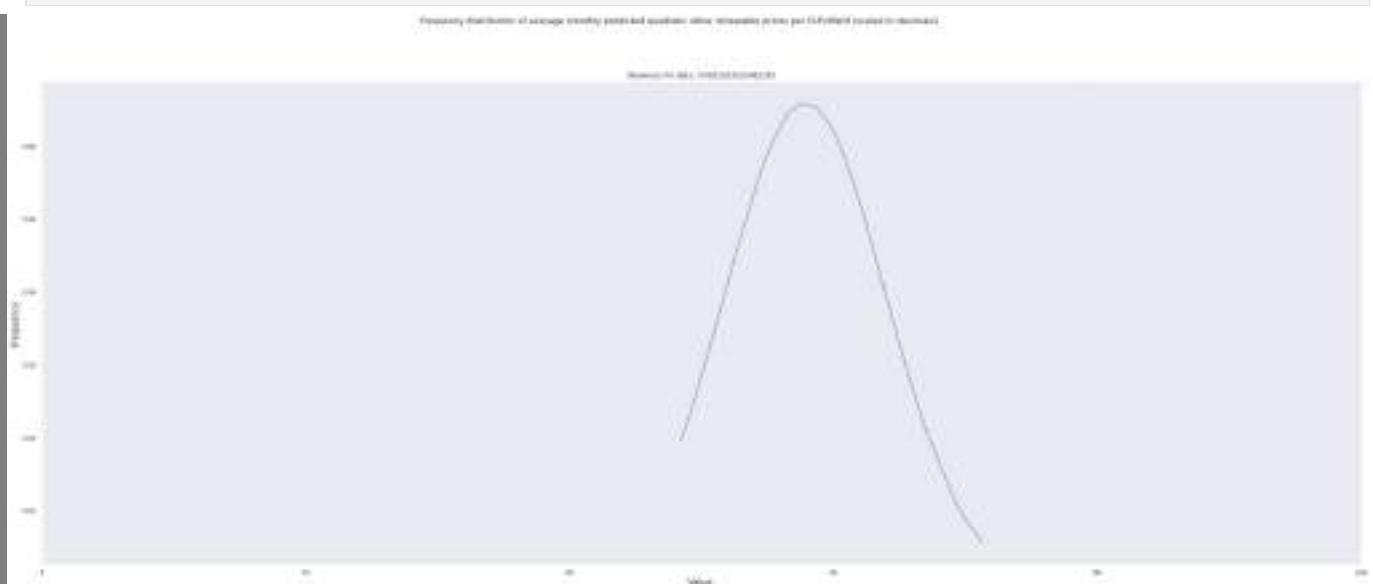
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Otherrenewable_QuadResults_mean = np.mean(df_Otherrenewable_Quad["Otherrenewable_Quad"])
Otherrenewable_QuadResults_std = np.std(df_Otherrenewable_Quad["Otherrenewable_Quad"])

Otherrenewable_QuadResultspdf = stats.norm.pdf(df_Otherrenewable_Quad["Otherrenewable_Quad"].sort_values(), Otherrenewable_QuadResults_mean, Otherrenewable_QuadResults_std)

plt.plot(df_Otherrenewable_Quad["Otherrenewable_Quad"].sort_values(), Otherrenewable_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Otherrenewable_Quad["Otherrenewable_Quad"])}')
plt.suptitle("Frequency distribution of average monthly predicted quadratic other renewable prices per EUR/MWH")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetric. Hence, it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)

```
Otherrenewable_Quad_Dict = {key: i for i, key in enumerate(df_Otherrenewable_Quad["Otherrenewable_Quad"])}

def Hist_Otherrenewable_Quad(Otherrenewable_Quad_Dict):
    for k, v in Otherrenewable_Quad_Dict.items(): print(f"{v}:{k}")
print(Otherrenewable_Quad_Dict)
```

```

plt.bar(list(Otherrenewable_Quad_Dict.values()), Otherrenewable_Quad_Dict.keys(), color='g') #Predictive Quadra
plt.title("Average monthly predicted quadratic other renewable prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

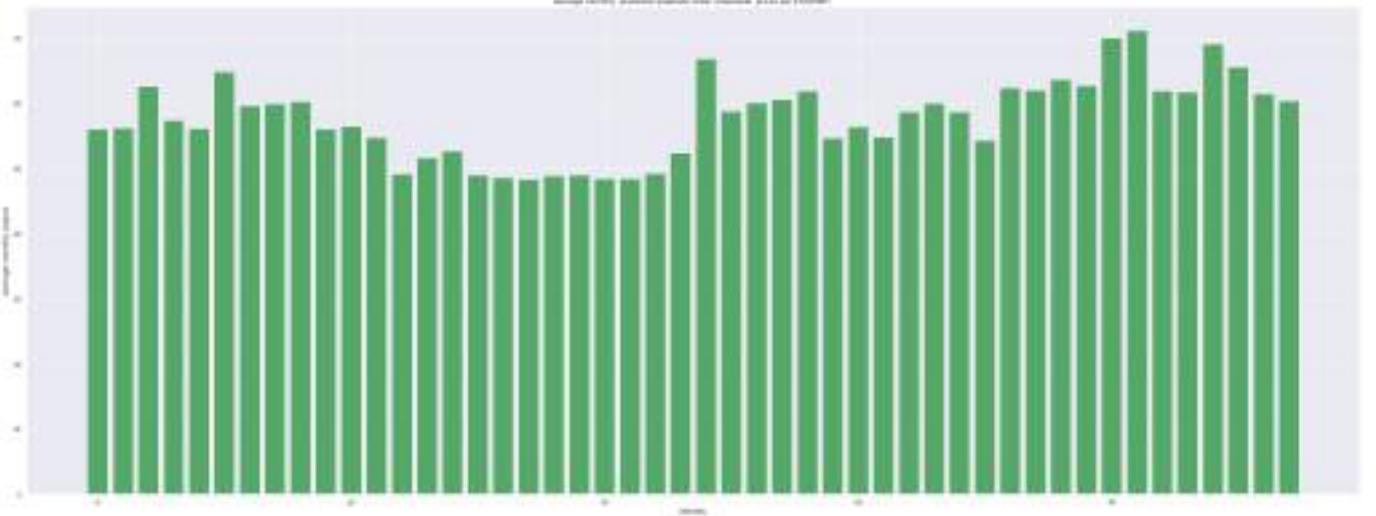
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15
-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18
, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03
': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '1
7-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{56.1048812487428: 0, 56.309545474732715: 1, 62.747469467313124: 2, 57.50430940491407: 3, 56.23749695211211: 4,
64.95521138600515: 5, 59.70967053614663: 6, 59.96307778558894: 7, 60.32343828384302: 8, 56.09222469931012: 9, 5
6.59876223558814: 10, 54.81622167317096: 11, 49.218960058849746: 12, 51.696333933634435: 13, 52.738757591906335:
14, 49.06830225817225: 15, 48.678423648153625: 16, 48.4151254579555: 17, 48.91761211149054: 18, 49.000032535410
41: 19, 48.482400650324905: 20, 48.47992465397948: 21, 49.288437535342894: 22, 52.44838020644494: 23, 66.9367576
939049: 24, 58.81993636829844: 25, 60.116049470018254: 26, 60.61235858907003: 27, 62.009773888742984: 28, 54.761
17478119096: 29, 56.50840020050134: 30, 54.942887260857276: 31, 58.75999072888152: 32, 60.01619331503406: 33, 58
.807225869519584: 34, 54.397609142724036: 35, 62.45114973178613: 36, 62.03101906140358: 37, 63.718393542498575:
38, 62.77047914663564: 39, 70.04586212318839: 40, 71.27347412887764: 41, 61.956873358603104: 42, 61.769495254628
85: 43, 69.13386735791408: 44, 65.6921867103315: 45, 61.4900565431131: 46, 60.45650233395156: 47}

```



The green bars represent the observation value for each respective month. This histogram is multimodal.

```
In [ ]: df_Otherrenewable_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Otherrenewable_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	0.092588	1.984085
std	4.174498	6.109699	NaN	4.492203	8.887615
min	17.075344	48.415126	NaN	-9.316601	-16.540086
25%	21.390791	53.982896	NaN	-1.674027	-6.992022
50%	24.615282	58.783608	NaN	0.082420	4.525409
75%	27.214266	61.970098	NaN	1.346764	7.702113
max	36.035067	71.273474	NaN	14.488377	17.717798

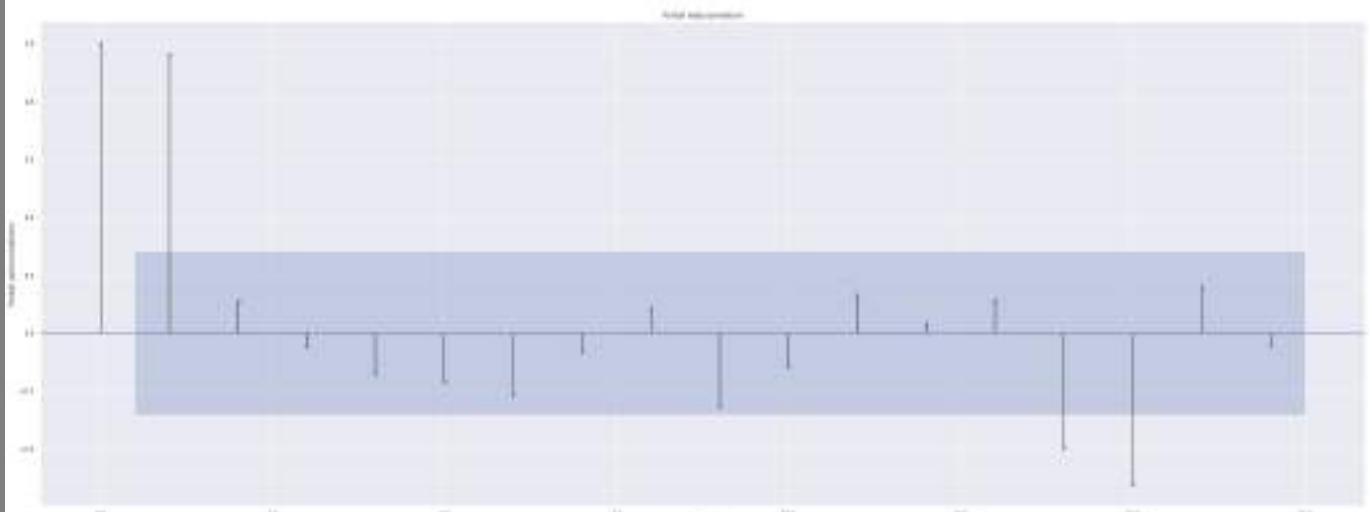
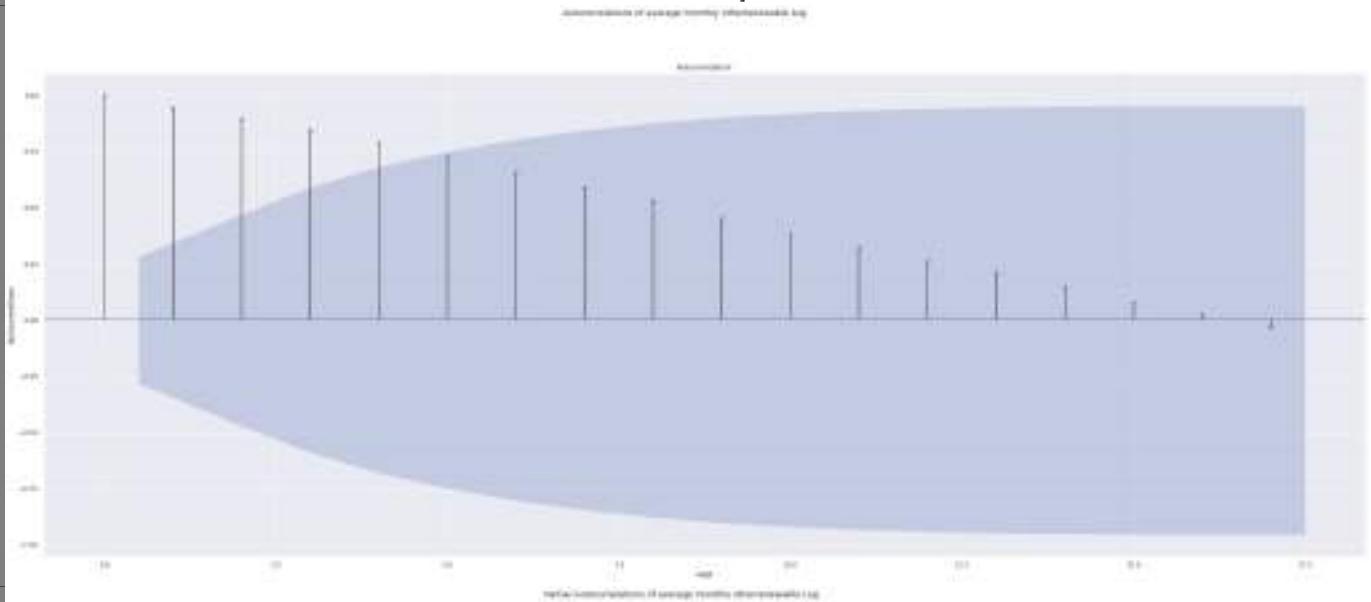
```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Otherrenewable_Logpred)
plt.suptitle(" Autocorrelations of average monthly otherrenewable Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Otherrenewable_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly otherrenewable Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
otherrenewable_Log_Autocorrelations = sm.tsa.acf(Otherrenewable_Logpred, fft=False) #Autocorrelations

```

```
print(otherrenewable_Log_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,[ 1. 0.94117536 0.89348572 0.84369179 0.78725893 0.72486043  
0.65468907 0.58748126 0.52756843 0.45241096 0.38019201 0.3203501  
0.26223333 0.21237476 0.14533845 0.0744268 0.02551979 -0.03198226  
-0.09041343 -0.14775868 -0.20028566 -0.24966868 -0.30840998 -0.34505667  
-0.37428079 -0.38321971 -0.39210057 -0.38989726 -0.38833649 -0.37955793  
-0.38775568 -0.39694933 -0.39423949 -0.38878231 -0.37604985 -0.36329824  
-0.36139473 -0.33509046 -0.31158165 -0.28578781 -0.25129561]
```



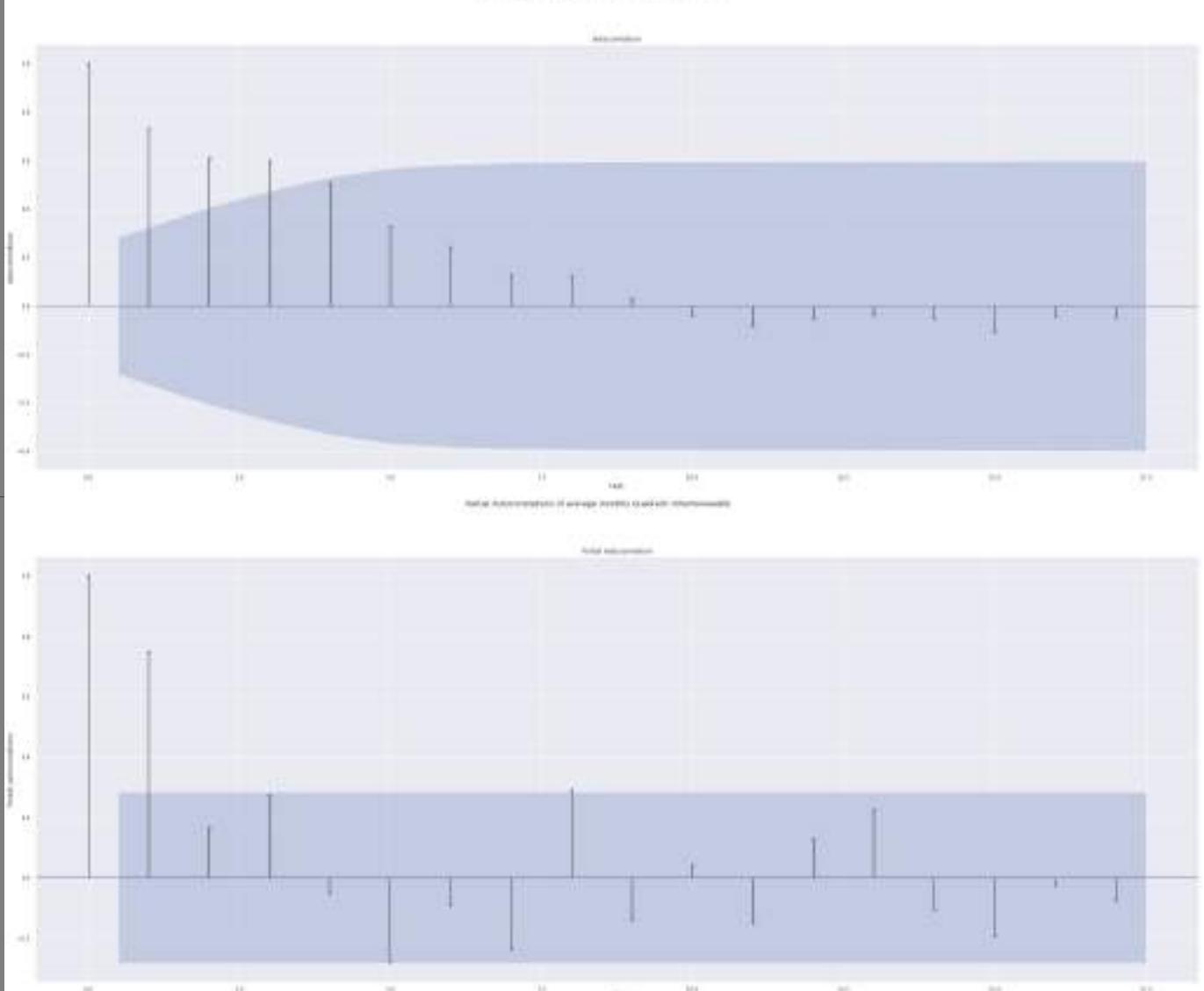
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(Otherrenewable_ypred)  
plt.suptitle(" Autocorrelations of average monthly Quadratic otherrenewable")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(Otherrenewable_ypred)  
plt.suptitle("Partial Autocorrelations of average monthly Quadratic otherrenewable")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
otherrenewable_Quad_Autocorrelations = sm.tsa.acf(Otherrenewable_ypred, fft=False) #Autocorrelations
```

```
print(otherrenewable_Quad_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,[ 1.          0.73253501  0.60637543  0.59593054  0.50362887  0.32763345  
 0.24308862  0.12642582  0.122127  0.02845757 -0.03958689 -0.08083627  
-0.04948442 -0.03501756 -0.05031444 -0.10665062 -0.04274767 -0.04498679  
-0.13956825 -0.136102  -0.18509792 -0.25471084 -0.27077981 -0.33740531  
-0.40453721 -0.36372203 -0.32691309 -0.30764599 -0.28180761 -0.24717  
-0.16683352 -0.09450717 -0.08755819 -0.04051667  0.0105493   0.0430506  
 0.05985923  0.02913895  0.05681152  0.07909392  0.01476994]
```



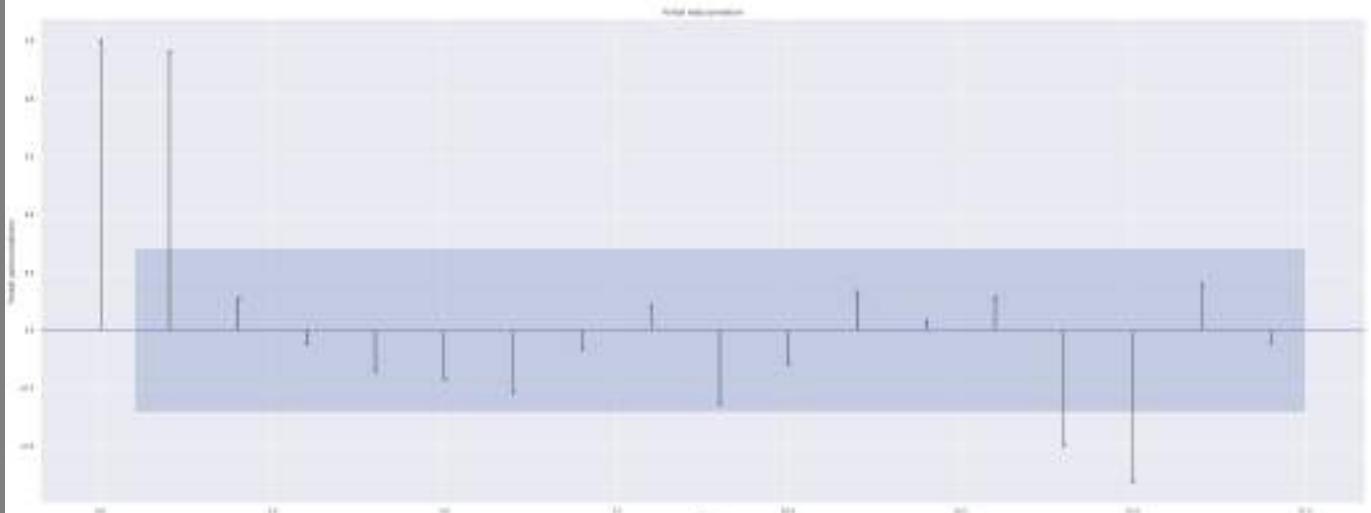
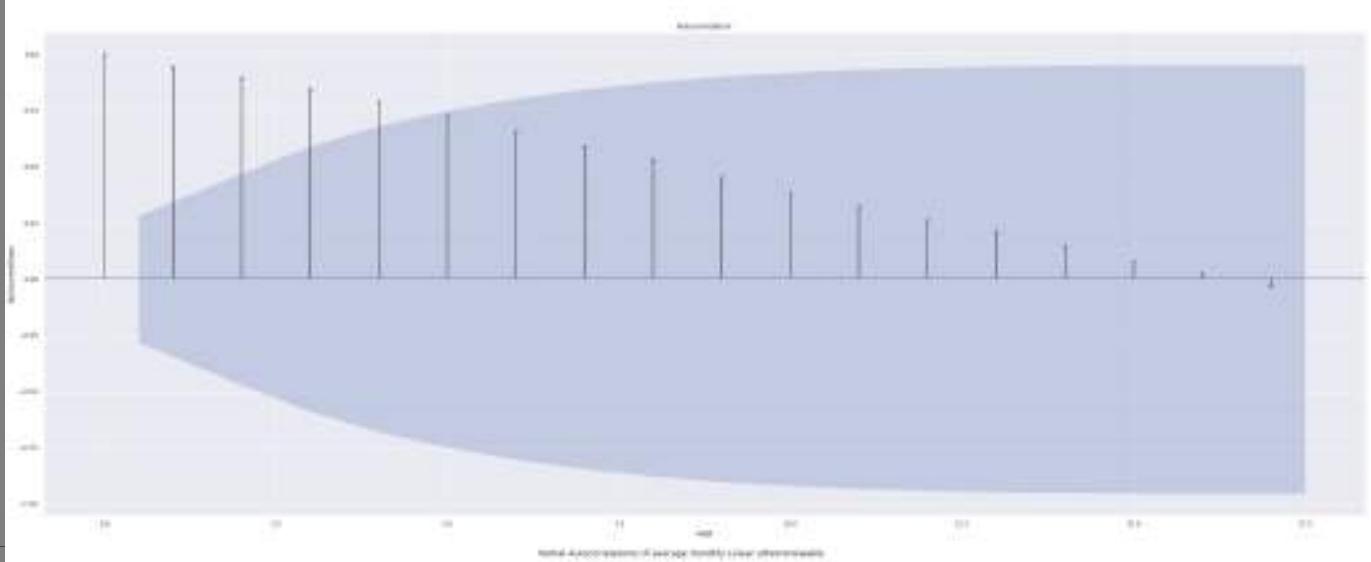
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(predictionsotherrenewable)  
plt.suptitle(" Autocorrelations of average monthly Linear otherrenewable")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partial autocorrelation Plot  
plot_pacf(predictionsotherrenewable)  
plt.suptitle("Partial Autocorrelations of average monthly Linear otherrenewable")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
otherrenewable_Pred_Autocorrelations = sm.tsa.acf(predictionsotherrenewable, fft=False) #Autocorrelations
```

```
print(otherrenewable_Pred_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,[ 1.  0.94117536  0.89348572  0.84369179  0.78725893  0.72486043  
 0.65468907  0.58748126  0.52756843  0.45241096  0.38019201  0.3203501  
 0.26223333  0.21237476  0.14533845  0.0744268  0.02551979 -0.03198226  
-0.09041343 -0.14775868 -0.20028566 -0.24966868 -0.30840998 -0.34505667  
-0.37428079 -0.38321971 -0.39210057 -0.38989726 -0.38833649 -0.37955793  
-0.38775568 -0.39694933 -0.39423949 -0.38878231 -0.37604985 -0.36329824  
-0.36139473 -0.33509046 -0.31158165 -0.28578781 -0.25129561]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resource analyzed was fossil hard coal.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly fossil hard coal outputs versus the average monthly prices of energy per EUR/MWH.

In []:

```
In [ ]: modelFossilHardcoal = stats.linregress([5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.51671309],[64.9490188172043,
```

56. 3838541666666666 ,
55. 522462987886975 ,
58. 35408333333333 ,
57. 29405913978498 ,
65. 9749027777778 ,
71. 07204301075271 ,
63. 99806451612899 ,
60. 254791666666634 ,
59. 40676510067113 ,
60. 72679166666668 ,
61. 901760752688226 ,
45. 57872311827956 ,
36. 752083333333374 ,
36. 81800807537014 ,
32. 61866666666666 ,
34. 691370967741896 ,
46. 2663194444444434 ,
47. 50201612903221 ,
47. 6023387096774 ,
50. 40559722222224 ,
60. 182429530201404 ,
62. 58105555555558 ,
67. 5951344086021 ,
79. 49208333333331 ,
59. 83779761904767 ,
50. 95989232839837 ,
51. 71791666666662 ,
53. 77262096774189 ,
56. 25822222222224 ,
55. 252580645161316 ,
54. 08432795698931 ,
55. 81655555555558 ,
63. 92528859060403 ,
65. 43065277777781 ,
65. 15127688172035 ,
56. 51197580645163 ,
60. 877098214285674 ,
48. 279717362045766 ,
50. 40073611111113 ,
61. 633763440860214 ,
64. 34813888888884 ,
67. 78344086021498 ,
70. 36391129032262 ,
76. 9140416666666 ,
70. 36221476510062 ,
67. 0426075268817 ,
66. 623513888888811)

```
In [ ]: #Dataframes analyzed by resource
dfFossilHard = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059,
    "Fossil_Hard_Coal": [5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.612903],
    print(dfFossilHard)
    df_FossilHard = pd.DataFrame.from_dict(dfFossilHard, orient = "columns")
    print(df_FossilHard)
    df_FossilHard["Ratio"] = df_FossilHard["Fossil_Hard_Coal"] / df_FossilHard["Price"]
    pdToListFossilHard = list(df_FossilHard["Ratio"])
    print(pdToListFossilHard)
```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.095989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Hard_Coal': [5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.6129032258063, 6207.095966620306, 6748.469086021505, 5859.370967741936, 5653.873611111111, 5788.694892473119, 5982.631944444444, 5323.676985195155, 4204.615591397849, 3027.655172413793, 2985.886944818304, 2226.186111111111, 2226.65188172043, 2877.080555555557, 4003.9165545087485, 4172.501344086021, 4208.0138888889, 4271.040268456376, 4581.631944444444, 4984.295698924731, 5533.782258064516, 4382.5327380952385, 2967.8492597577388, 3119.322222222222, 4428.919354838709, 5091.152777777777, 4561.283602150537, 3786.2325268817203, 4010.465277777778, 4321.130201342282, 5491.216666666666, 4330.653225806452, 3783.6223118279568, 4144.389880952381, 2157.1830417227457, 2775.8791666666666, 3674.15188172043, 3059.616666666667, 4014.707940780619, 4121.565860215053, 4651.018055555555, 3782.8161073825504, 3365.7580645161293, 4838.777777777777], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

	Price	Fossil_Hard_Coal	Dates
0	64.949019	5411.263302	2015-01
1	56.383854	4045.974702	2015-02
2	55.522463	4233.818304	2015-03
3	58.354083	4819.516713	2015-04
4	57.294059	4019.612903	2015-05
5	65.974903	6207.095967	2015-06
6	71.072043	6748.469086	2015-07
7	63.998065	5859.370968	2015-08
8	60.254792	5653.873611	2015-09
9	59.406765	5788.694892	2015-10
10	60.726792	5982.631944	2015-11
11	61.901761	5323.676985	2015-12
12	45.578723	4204.615591	2016-01
13	36.752083	3027.655172	2016-02
14	36.818008	2985.886945	2016-03
15	32.618667	2226.186111	2016-04
16	34.691371	2226.651882	2016-05
17	46.266319	2877.080556	2016-06
18	47.502016	4003.916555	2016-07
19	47.602339	4172.501344	2016-08
20	50.405597	4208.013889	2016-09
21	60.182430	4271.040268	2016-10
22	62.581056	4581.631944	2016-11
23	67.595134	4984.295699	2016-12
24	79.492083	5533.782258	2017-01
25	59.837798	4382.532738	2017-02
26	50.959892	2967.849260	2017-03
27	51.717917	3119.322222	2017-04
28	53.772621	4428.919355	2017-05
29	56.258222	5091.152778	2017-06
30	55.252581	4561.283602	2017-07
31	54.084328	3786.232527	2017-08
32	55.816556	4010.465278	2017-09
33	63.925289	4321.130201	2017-10
34	65.430653	5491.216667	2017-11
35	65.151277	4330.653226	2017-12
36	56.511976	3783.622312	2018-01
37	60.877098	4144.389881	2018-02
38	48.279717	2157.183042	2018-03
39	50.400736	2775.879167	2018-04
40	61.633763	3674.151882	2018-05
41	64.348139	3059.616667	2018-06
42	67.783441	4014.707941	2018-07
43	70.363911	4121.565860	2018-08
44	76.914042	4651.018056	2018-09
45	70.362215	3782.816107	2018-10
46	67.042608	3365.758065	2018-11
47	66.623514	4838.777778	2018-12

[83.31555118223427, 71.75768244613677, 76.2541514971327, 82.59090774439244, 70.1575863811435, 94.08268455548271, 94.95251297335733, 91.55544018468301, 93.8327634155421, 97.44167827794621, 98.51717471397966, 86.0020283827543, 92.24952573784516, 82.38050466292277, 81.0985466325581, 68.24883842925661, 64.18460324877044, 62.185204920185825, 84.28940244626031, 87.65328463237383, 83.48306776997595, 70.96822613837882, 73.21116436550287, 73.73749224012832, 69.61425623806792, 73.24020790331005, 58.23892328170889, 60.31415074831686, 82.36383637493897, 90.49615463616182, 82.55331332745612, 70.00609363016835, 71.85081984835243, 67.59656931728608, 83.92422257067331, 66.47073446723971, 66.95257523443384, 68.0779801028663, 44.68093766055425, 55.07616318434503, 59.61264859715907, 47.54786571138853, 59.22844709314572, 58.574996537776435, 60.47033746727723, 53.7620386170506, 50.20326906537128, 72.62867860508874]

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy

per EUR/MWH. This is the linear model for the average monthly fossil hard coal outputs versus the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Linear OLS regression
Fossil_Hard1 = Fossil_Hard

Fossil_Hard1 = sm.add_constant(Fossil_Hard1)
modelFossilHardreg = sm.OLS(Price_Actual, Fossil_Hard1).fit()
predictionsFossilHard = modelFossilHardreg.predict(Fossil_Hard1)

modelFossilHardreg.summary()
#OLS Linear Summary Table
```

```
Out[ ]: OLS Regression Results
Dep. Variable: y R-squared: 0.408
Model: OLS Adj. R-squared: 0.395
Method: Least Squares F-statistic: 31.70
Date: Wed, 30 Nov 2022 Prob (F-statistic): 1.03e-06
Time: 05:25:57 Log-Likelihood: -167.06
No. Observations: 48 AIC: 338.1
Df Residuals: 46 BIC: 341.9
Df Model: 1

Covariance Type: nonrobust

            coef  std err      t  P>|t|  [0.025  0.975]
const    31.7292   4.783  6.633  0.000  22.101  41.357
x1       0.0061   0.001  5.631  0.000    0.004   0.008

Omnibus: 1.401 Durbin-Watson: 0.365
Prob(Omnibus): 0.496 Jarque-Bera (JB): 1.393
Skew: 0.336 Prob(JB): 0.498
Kurtosis: 2.506 Cond. No. 1.81e+04
```

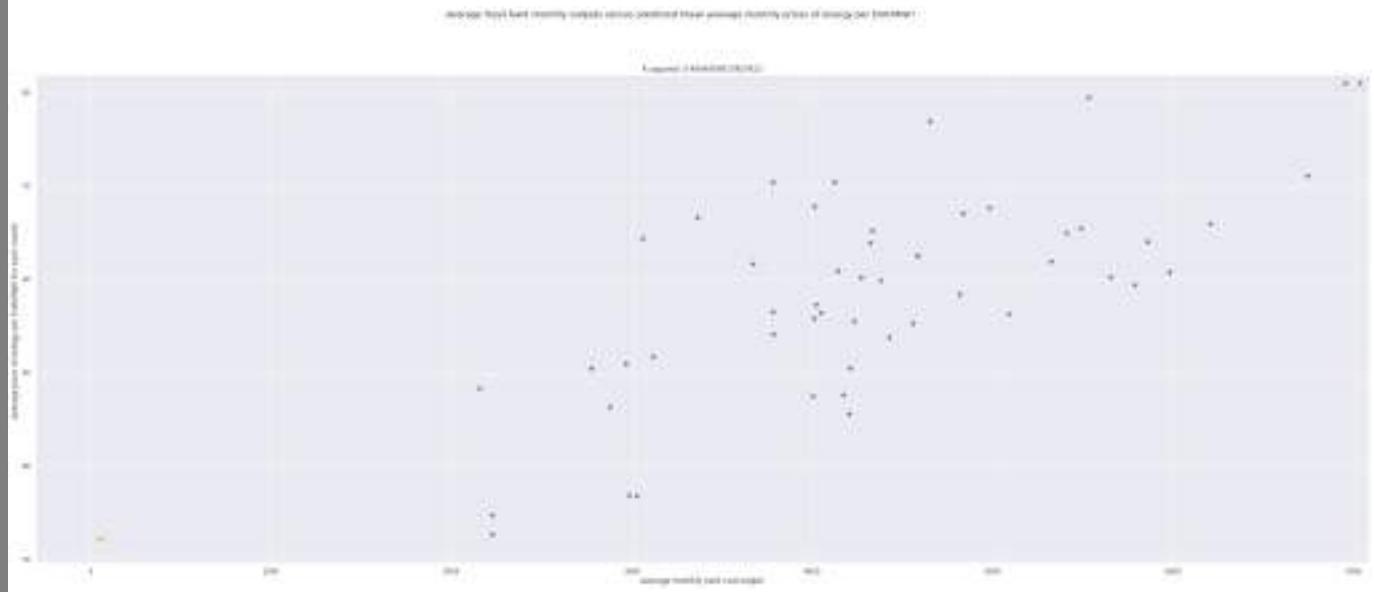
Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.81e+04. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [ ]:
```

```
In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(Fossil_Hard,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))
#OLS Linear Scatterplot
plt.plot(Fossil_Hard,Price_Actual, "o")
plt.title(f"R squared: {modelFossilHard.coef**2}")
f = lambda x: 0.006141 *x + 31.729162
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average fossil hard monthly outputs versus predicted linear average monthly prices of energy per MWh")
plt.legend("#")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly hard coal output')
plt.show()
```

```
slope: 0.006141      intercept: 31.729162
```



The blue dots represent the observations and the orange line is the model of best fit. There is a moderately positive correlation the average monthly outputs of hard fossil coal and their respective the average monthly prices of energy per EUR/MWH. It appears that as the average monthly price of energy per EUR/MWH increases, the average monthly outputs of fossil hard coal increases as well.

```
In [ ]: #Linear OLS regression residuals
influenceFossilHardreg = modelFossilHardreg.get_influence()

standardized_residualsFossilHard = influenceFossilHardreg.resid_studentized_internal

print(standardized_residualsFossilHard)
```

[-1.12137088e-03 -2.39573832e-02 -2.77628550e-01 -3.75063870e-01
 1.11090009e-01 -5.05725448e-01 -2.81007936e-01 -4.79044473e-01
 -7.94482878e-01 -1.01341675e+00 -1.00306269e+00 -3.20519797e-01
 -1.50704004e+00 -1.73317069e+00 -1.69370381e+00 -1.67548898e+00
 -1.40411545e+00 -4.01327916e-01 -1.11034930e+00 -1.22747373e+00
 -9.01928545e-01 2.80302909e-01 3.42514289e-01 6.65491825e-01
 1.76256357e+00 1.50736638e-01 1.28714180e-01 1.06318038e-01
 -6.48961687e-01 -8.53462306e-01 -5.65278692e-01 -1.12886314e-01
 -6.79575032e-02 7.12845086e-01 -2.31558911e-03 8.59855891e-01
 1.95426424e-01 4.65734902e-01 4.34412210e-01 2.09064171e-01
 9.27461100e-01 1.76532539e+00 1.43626026e+00 1.67803481e+00
 2.09619752e+00 1.94353115e+00 7.07362470e-01 1.80467377e+00]

```
In [ ]: print(predictionsFossilHard)
#Linear OLS Predicted Values
```

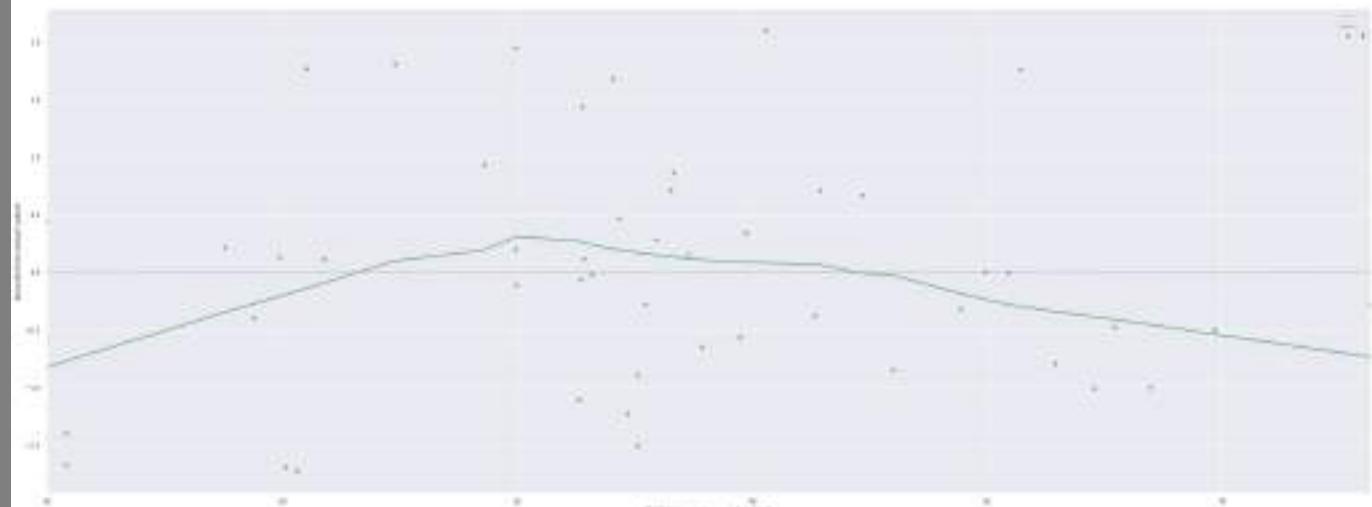
[64.9578125 56.5740589 57.72753991 61.32410634 56.41218043 69.84473895
 73.16911948 67.70948292 66.44759646 67.27548618 68.46638491 64.41997596
 57.5482164 50.32091944 50.06443555 45.39938187 45.40224201 49.39629406
 56.31579472 57.35101418 57.56908414 57.95610681 59.86334025 62.33595571
 65.710158 58.64074256 49.95367251 50.88381431 58.92558634 62.99212718
 59.73838829 54.97907422 56.35600811 58.26369134 65.44877779 58.32216886
 54.96304582 57.17839165 44.97565847 48.77485218 54.29082669 50.51718365
 56.3820608 57.03823742 60.28941578 54.9580952 61.44238171 52.39708951]

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionsFossilHard, y = standardized_residualsFossilHard, lowess = True, color ="g")

plt.suptitle("Fossil hard residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend("#")
```

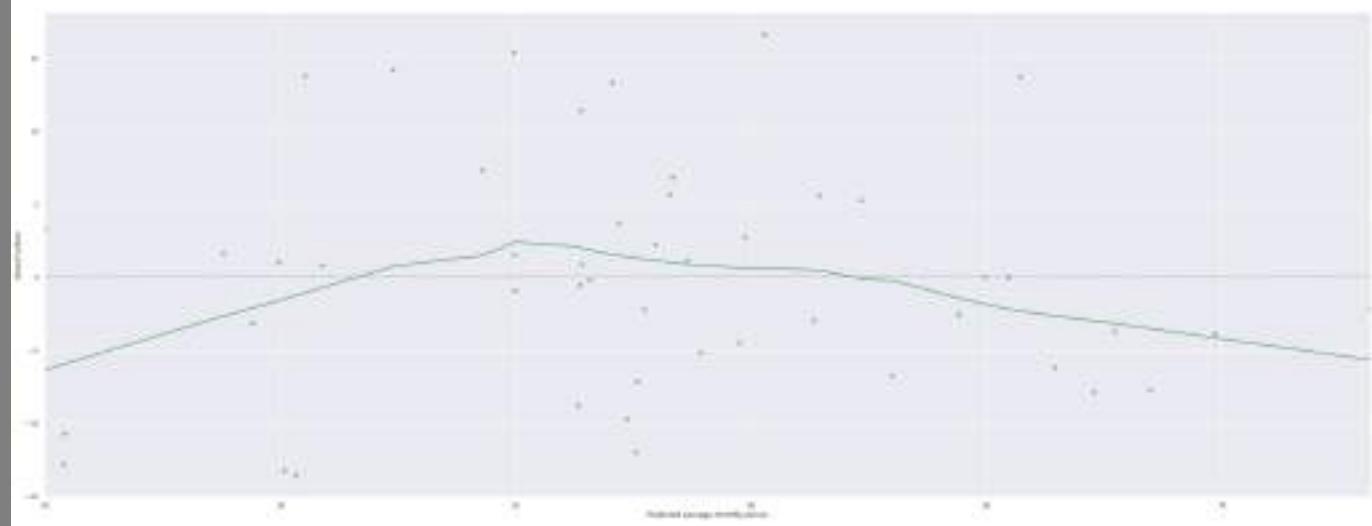
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff607bf2310>
```



As one can observe this residual plot, one may notice an arching hump in the fitted model, which form a nonlinear pattern. However, the residuals are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly linear fossil hard energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsFossilHard, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff607a64bd0>
```



As one can observe this residual plot, one may notice an arching hump in the fitted model, which form a nonlinear pattern. However, the observations are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]:
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the average monthly fossil hard coal outputs versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
LogFossilHardvalues = ((np.log(Fossil_Hard)))
```

```

Log = np.polyfit(np.log(Price_Actual), Fossil_Hard1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(Fossil_Hard1), Price_Actual)
FossilHard_Log = sm.OLS(Price_Actual, Fossil_Hard1).fit()

FossilHard_Logpred = FossilHard_Log.predict(Fossil_Hard1)
#OLS Logarithmic summary table
FossilHard_Log.summary()
#Log
Log = np.polyfit(np.log(Fossil_Hard), Price_Actual, 1)
print(Log)

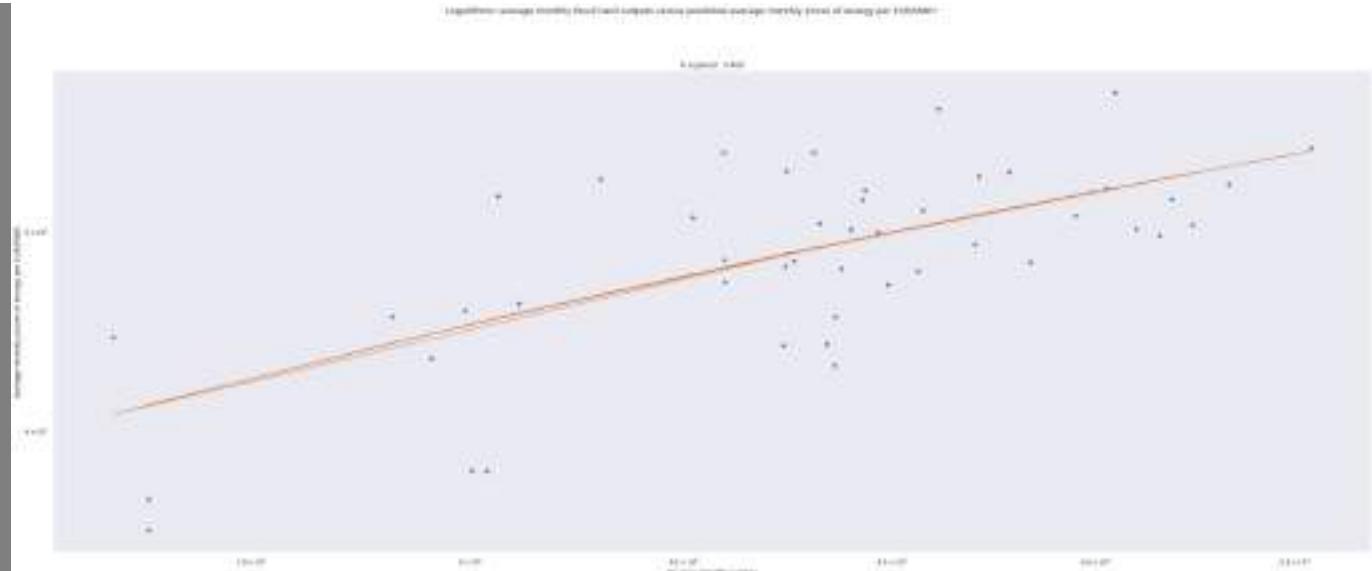
y = 25.67246221 * LogFossilHardvalues -155.79490271

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly fossil hard outputs versus predicted average monthly prices of energy")
plt.title("R squared : 0.408")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(LogFossilHardvalues, Price_Actual, "o")
plt.plot(LogFossilHardvalues, y)

```

[25.67246221 -155.79490271]

Out[]: [`<matplotlib.lines.Line2D at 0x7ff606f29950>`]



The blue dots represent the observations and the orange line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

In []: `FossilHard_Log.summary() #OLS Logarithmic summary table`

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.408			
Model:	OLS	Adj. R-squared:	0.395			
Method:	Least Squares	F-statistic:	31.70			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	1.03e-06			
Time:	05:26:00	Log-Likelihood:	-167.06			
No. Observations:	48	AIC:	338.1			
Df Residuals:	46	BIC:	341.9			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	31.7292	4.783	6.633	0.000	22.101	41.357
x1	0.0061	0.001	5.631	0.000	0.004	0.008
Omnibus:	1.401	Durbin-Watson:	0.365			
Prob(Omnibus):	0.496	Jarque-Bera (JB):	1.393			
Skew:	0.336	Prob(JB):	0.498			
Kurtosis:	2.506	Cond. No.	1.81e+04			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.81e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
In [ ]: influenceFossilHardLog = FossilHard_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsFossilHardLog = influenceFossilHardLog.resid_studentized_internal

print(standardized_residualsFossilHardLog)

print(FossilHard_Logpred) # OLS logarithmic predicted values
```

-1.12137088e-03	-2.39573832e-02	-2.77628550e-01	-3.75063870e-01
1.11090009e-01	-5.05725448e-01	-2.81007936e-01	-4.79044473e-01
-7.94482878e-01	-1.01341675e+00	-1.00306269e+00	-3.20519797e-01
-1.50704004e+00	-1.73317069e+00	-1.69370381e+00	-1.67548898e+00
-1.40411545e+00	-4.01327916e-01	-1.11034930e+00	-1.22747373e+00
-9.01928545e-01	2.80302909e-01	3.42514289e-01	6.65491825e-01
1.76256357e+00	1.50736638e-01	1.28714180e-01	1.06318038e-01
-6.48961687e-01	-8.53462306e-01	-5.65278692e-01	-1.12886314e-01
-6.79575032e-02	7.12845086e-01	-2.31558911e-03	8.59855891e-01
1.95426424e-01	4.65734902e-01	4.34412210e-01	2.09064171e-01
9.27461100e-01	1.76532539e+00	1.43626026e+00	1.67803481e+00
2.09619752e+00	1.94353115e+00	7.07362470e-01	1.80467377e+00
[64.9578125	56.5740589	57.72753991	61.32410634
73.16911948	67.70948292	66.44759646	67.27548618
57.5482164	50.32091944	50.06443555	45.39938187
56.31579472	57.35101418	57.56908414	57.95610681
65.710158	58.64074256	49.95367251	50.88381431
59.73838829	54.97907422	56.35600811	58.26369134
54.96304582	57.17839165	44.97565847	48.77485218
56.3820608	57.03823742	60.28941578	54.9580952
			61.44238171
			52.39708951]

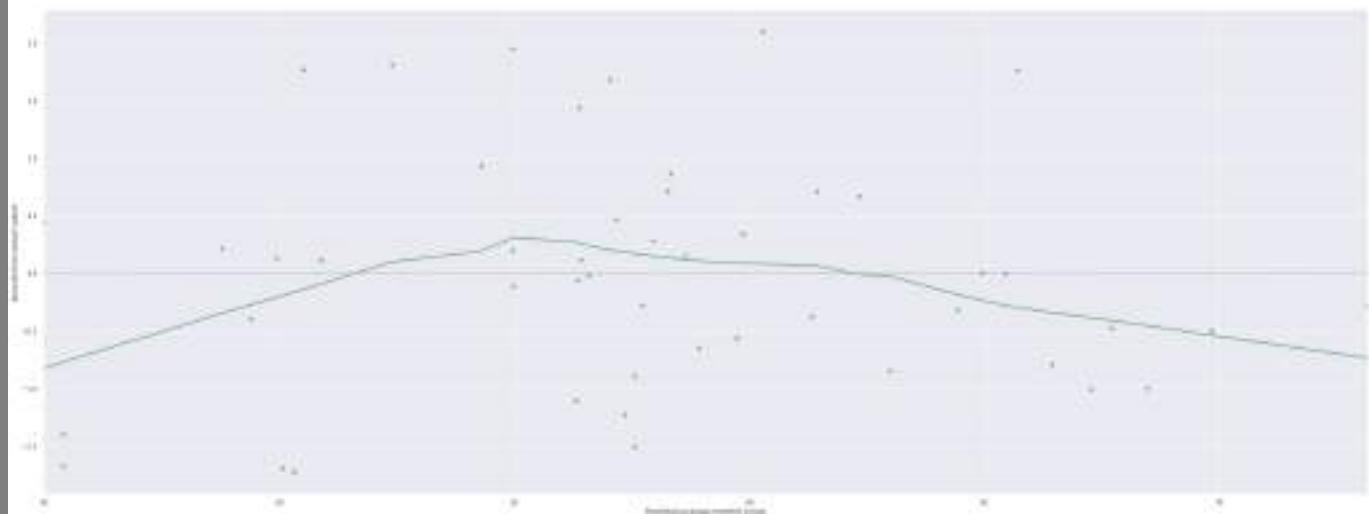
In []:

FossilHardLogRatioPredict = FossilHard_Logpred/Logpred

In []:

```
# OLS Logarithmic average monthly predictions versus residuals
plt.suptitle("Residuals from logarithmic model versus predicted average monthly logarithmic fossil hard output")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend(".#")
sns.residplot(x = FossilHard_Logpred, y = standardized_residualsFossilHardLog, lowess = True, color="g")
```

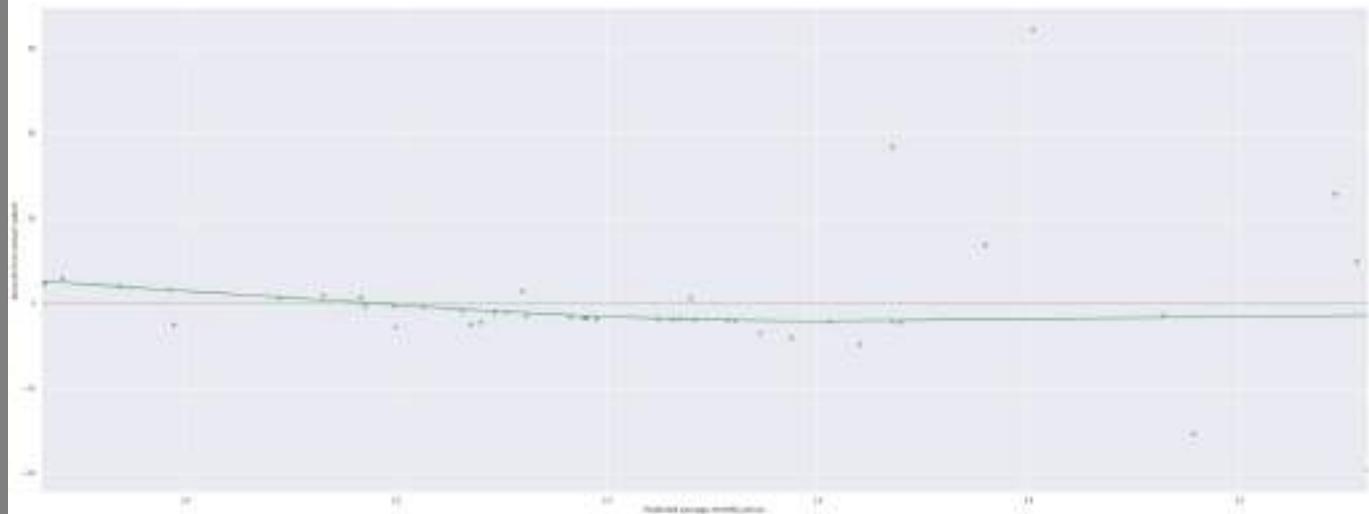
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606e32550>
```



As one can observe this residual plot, one may notice an arching hump in the fitted model, which form a nonlinear pattern. However, the residuals are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
plt.suptitle("Predicted average monthly logarithmic output to price of energy per EUR/MWH ratio versus respective predicted average monthly prices")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Amount from actual values")  
plt.legend(..#")  
sns.residplot(x = FossilHardLogRatioPredict, y = standardized_residualsFossilHardLog/standardized_residualsPrice
```

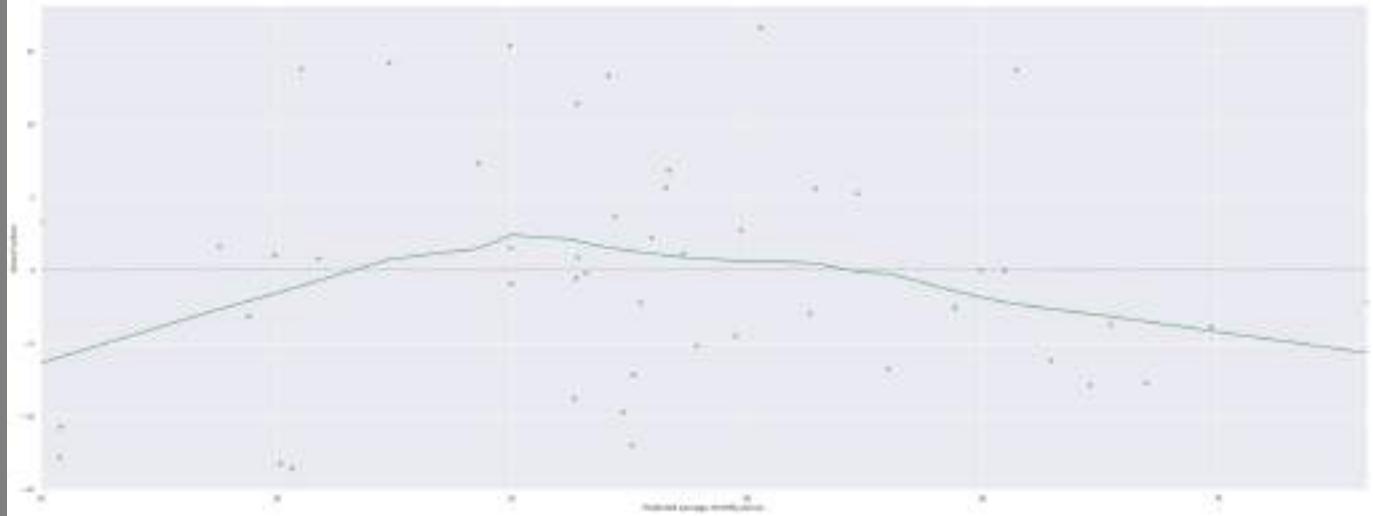
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606d9f2d0>
```



With the exception of few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly logarithmic fossil hard energy prices per EUR/MWH versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend(..#")  
sns.residplot(x = FossilHard_Logpred, y = Price_Actual, lowess = True, color="g")  
# OLS predicted logarithmic average monthly values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606d38fd0>
```



As one can observe, there is a hump along the lowess line in the residual plot. Furthermore, the observations are quite spread out in no particular pattern which indicates constant variance and a lack of bias. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: influenceFossilHardLog = FossilHard_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsFossilHardLog = influenceFossilHardLog.resid_studentized_internal

print(standardized_residualsFossilHardLog)

[-1.12137088e-03 -2.39573832e-02 -2.77628550e-01 -3.75063870e-01
 1.11090009e-01 -5.05725448e-01 -2.81007936e-01 -4.79044473e-01
 -7.94482878e-01 -1.01341675e+00 -1.00306269e+00 -3.20519797e-01
 -1.50704004e+00 -1.73317069e+00 -1.69370381e+00 -1.67548898e+00
 -1.40411545e+00 -4.01327916e-01 -1.11034930e+00 -1.22747373e+00
 -9.01928545e-01  2.80302909e-01  3.42514289e-01  6.65491825e-01
 1.76256357e+00  1.50736638e-01  1.28714180e-01  1.06318038e-01
 -6.48961687e-01 -8.53462306e-01 -5.65278692e-01 -1.12886314e-01
 -6.79575032e-02  7.12845086e-01 -2.31558911e-03  8.59855891e-01
 1.95426424e-01  4.65734902e-01  4.34412210e-01  2.09064171e-01
 9.27461100e-01  1.76532539e+00  1.43626026e+00  1.67803481e+00
 2.09619752e+00  1.94353115e+00  7.07362470e-01  1.80467377e+00]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly fossil hard coal outputs versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)

modelFossilHardquad = np.poly1d(np.polyfit(Fossil_Hard, Price_Actual,2))
print(modelFossilHardquad)

Fossil_Hard1 = Fossil_Hard

Fossil_Hard1 = sm.add_constant(Fossil_Hard1)
Fossil_Hard2 = polynomial_features.fit_transform(Fossil_Hard1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(Fossil_Hard2)

FossilHard_Q = poly.fit(X_poly, Price_Actual)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
FossilHard_Quad = sm.OLS(Price_Actual, Fossil_Hard2).fit()
```

```
# OLS Predicted Quadratic values
FossilHard_ypred = FossilHard_Quad.predict(Fossil_Hard2)

#OLS Quadratic Summary Table
FossilHard_Quad.summary()

##OLS Quadratic Scatterplot
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelFossilHardquad(polyline))
plt.scatter(Fossil_Hard, Price_Actual, color = 'blue')
plt.title("R squared : 0.457")
plt.suptitle('Quadratic for average monthly outputs of fossil oil versus predicted average monthly prices of energy per EUR/MWH')
plt.xlabel('Average monthly outputs of fossil oil')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()
```

$$-1.569e-06 x^2 + 0.01962 x + 4.536$$



In []: influenceFossilHardQuad = FossilHard_Quad.get_influence() #Quadratic OLS residuals

```
standardized_residualsFossilHardQuad = influenceFossilHardQuad.resid_studentized_internal
```

```
print(standardized_residualsFossilHardQuad)
```

```
[ 0.02196864 -0.24398851 -0.51984551 -0.56480219 -0.10085589  0.01157126
 0.90483068 -0.22248841 -0.67436385 -0.82322543 -0.68739714 -0.34660445
-1.79676366 -1.69246357 -1.63020441 -1.12380444 -0.82698687 -0.23150269
-1.36751502 -1.50463948 -1.16793453  0.05943352  0.14028995  0.553684
 1.90467973 -0.07390585  0.26525851  0.16274566 -0.9029873 -0.98513313
-0.80489013 -0.29548747 -0.28575337  0.50904413  0.05901784  0.66197058
 0.02469374  0.25673052  1.27051117  0.46582894  0.80811325  1.90741892
 1.2760139   1.51799573  1.96971249  1.83712112  0.56134005  1.81016537]
```

The blue dots represent the observations and the blue line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

In []: FossilHard_Quad.summary()#OLS Quadratic Summary Table

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.457			
Model:	OLS	Adj. R-squared:	0.433			
Method:	Least Squares	F-statistic:	18.93			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	1.08e-06			
Time:	05:26:03	Log-Likelihood:	-164.99			
No. Observations:	48	AIC:	336.0			
Df Residuals:	45	BIC:	341.6			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	1.5121	4.758	0.318	0.752	-8.071	11.095
x1	1.5121	4.758	0.318	0.752	-8.071	11.095
x2	0.0098	0.003	2.895	0.006	0.003	0.017
x3	1.5121	4.758	0.318	0.752	-8.071	11.095
x4	0.0098	0.003	2.895	0.006	0.003	0.017
x5	-1.569e-06	7.79e-07	-2.014	0.050	-3.14e-06	7.28e-11
Omnibus:	1.268	Durbin-Watson:	0.500			
Prob(Omnibus):	0.531	Jarque-Bera (JB):	1.282			
Skew:	0.318	Prob(JB):	0.527			
Kurtosis:	2.513	Cond. No.	6.23e+24			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 5.64e-34. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
print(FossilHard_ypred) # OLS quadratic predicted values
```

```
[64.78220667 58.248637 59.49457413 62.66858586 58.06492369 65.89222334
 65.51151477 65.65169283 65.33117725 65.55640889 65.78036496 64.538644
 59.30814568 49.5672009 49.14164589 40.44614492 40.45203094 48.00736856
 57.95450131 59.10004066 59.3299778 59.72831785 61.50925735 63.36753331
 65.08252326 60.40244586 48.95617666 50.48195852 60.67142768 63.77525103
 61.40184571 56.3434062 58.00066497 60.03600715 64.9835267 60.0936139
 56.32318563 58.91521798 39.56662128 46.91902622 55.45590039 49.88914288
 58.03050078 58.76333432 61.86575309 56.31693589 62.75468354 52.81044814]
```

In []:

```
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = FossilHard_ypred, y = standardized_residualsFossilHardQuad, lowess = True, color="g")

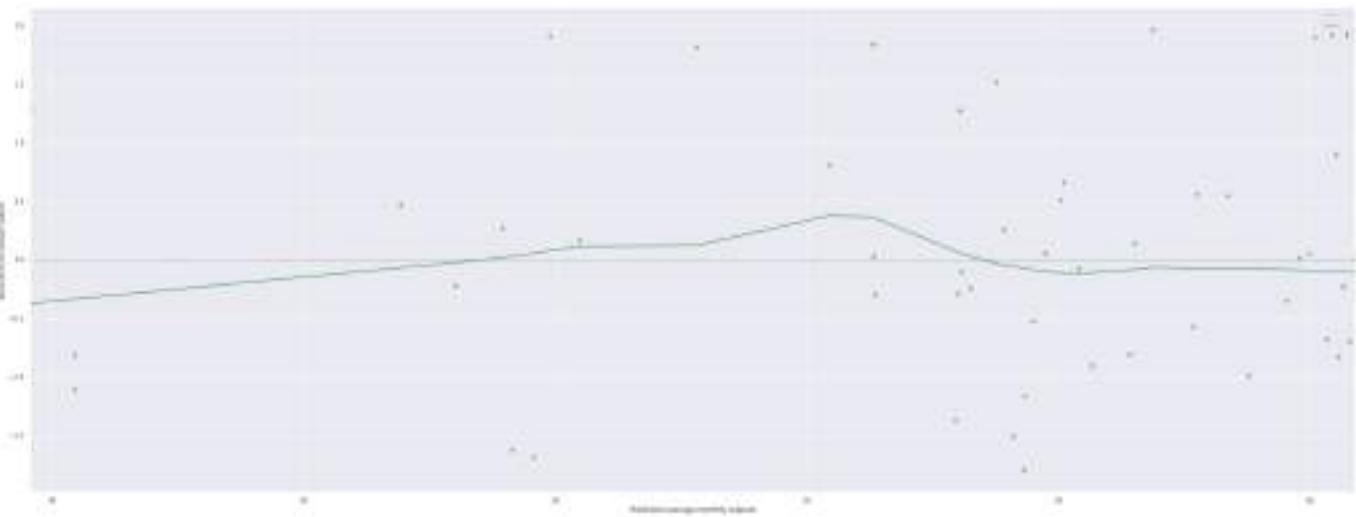
plt.suptitle("Fossil hard residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

Out[]:

```
<matplotlib.legend.Legend at 0x7ff606c71250>
```

Residuals replaced from quadratic model versus predicted values



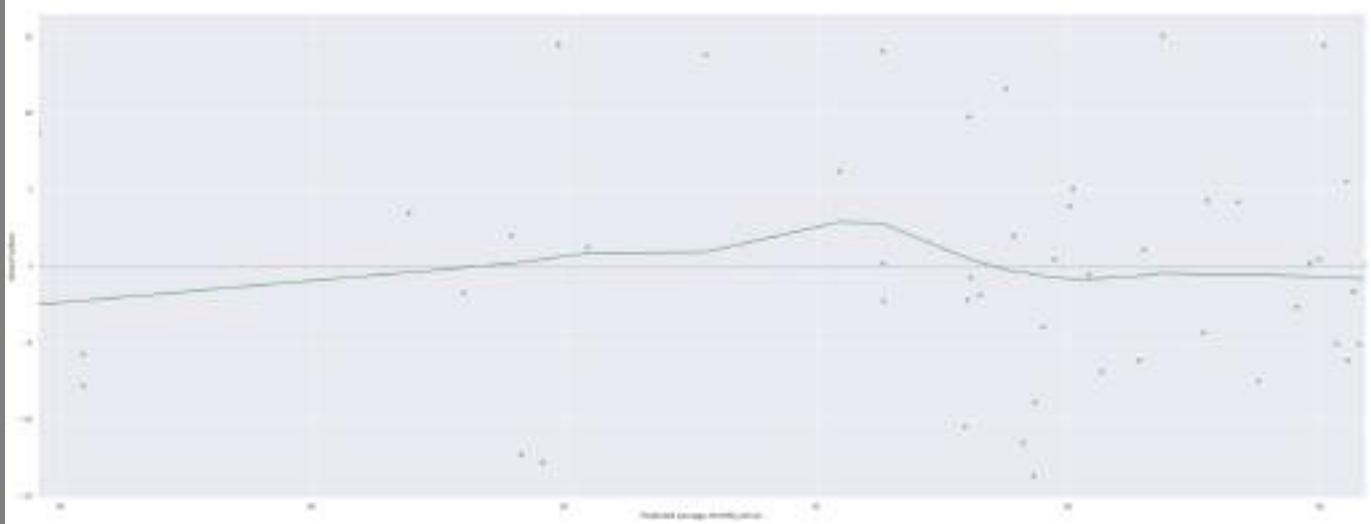
As one can observe this residual plot, one may notice that the lowess line is subtle and that the residuals are clustered on the right side; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []: `print(FossilHard_ypred)`

```
[64.78220667 58.248637 59.49457413 62.66858586 58.06492369 65.89222334
 65.51151477 65.65169283 65.33117725 65.55640889 65.78036496 64.538644
 59.30814568 49.5672009 49.14164589 40.44614492 40.45203094 48.00736856
 57.95450131 59.10004066 59.3299778 59.72831785 61.50925735 63.36753331
 65.08252326 60.40244586 48.95617666 50.48195852 60.67142768 63.77525103
 61.40184571 56.3434062 58.00066497 60.03600715 64.9835267 60.0936139
 56.32318563 58.91521798 39.56662128 46.91902622 55.45590039 49.88914288
 58.03050078 58.76333432 61.86575309 56.31693589 62.75468354 52.81044814]
```

In []: `plt.suptitle("Predicted average monthly quadratic fossil hard energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = FossilHard_ypred , y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values`

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606b98b90>



As one can observe this residual plot, one may notice that the lowess line is subtle and that the residuals are clustered on the right side; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

```
In [ ]: df_FossilHard['First Difference Ratio'] = df_FossilHard["Ratio"] - df_FossilHard["Ratio"].shift(1) # Seasonality
df_FossilHard['Seasonal Difference Ratio'] = df_FossilHard["Ratio"] - df_FossilHard["Ratio"].shift(12)
df_FossilHard.head()
```

	Price	Fossil_Hard_Coal	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
0	64.949019	5411.263302	2015-01	83.315551	NaN	NaN
1	56.383854	4045.974702	2015-02	71.757682	-11.557869	NaN
2	55.522463	4233.818304	2015-03	76.254151	4.496469	NaN
3	58.354083	4819.516713	2015-04	82.590908	6.336756	NaN
4	57.294059	4019.612903	2015-05	70.157586	-12.433321	NaN

```
In [ ]: #Dataframes analyzed by resource
dfFossilHard = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059],
                'Fossil_Hard_Coal' : [5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.612903],
                'Ratio' : [83.315551, 71.757682, 76.254151, 82.590908, 70.157586]})

print(dfFossilHard)
df_FossilHard= pd.DataFrame.from_dict(dfFossilHard, orient = "columns")
print(df_FossilHard)
df_FossilHard["Ratio"] = df_FossilHard["Fossil_Hard_Coal"]/df_FossilHard["Price"]
pdToListFossilHard = list(df_FossilHard["Ratio"])

print(pdToListFossilHard)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
```

```

if result[1] <= 0.05:
    print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
else:
    print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_FossilHard["Ratio"])

test_result=adfuller(df_FossilHard["Ratio"])

```

```

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilHard["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Fossil Hard Ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show

from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_FossilHard["Ratio"])
plt.suptitle("Partialatocorrelations of Fossil Hard Rato")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show

```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.726791666666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555555, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.4306527777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Hard_Coal': [5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.6129032258063, 6207.095966620306, 6748.469086021505, 5859.370967741936, 5653.873611111111, 5788.694892473119, 5982.631944444444, 5323.676985195155, 4204.615591397849, 3027.655172413793, 2985.886944818304, 2226.186111111111, 2226.65188172043, 2877.0805555555557, 4003.9165545087485, 4172.501344086021, 4208.0138888889, 4271.040268456376, 4581.631944444444, 4984.295698924731, 5533.782258064516, 4382.5327380952385, 2967.8492597577388, 3119.322222222222, 4428.919354838709, 5091.152777777777, 4561.283602150537, 3786.2325268817203, 4010.465277777778, 4321.130201342282, 5491.216666666666, 4330.653225806452, 3783.6223118279568, 4144.389880952381, 2157.1830417227457, 2775.8791666666666, 3674.15188172043, 3059.616666666667, 4014.707940780619, 4121.565860215053, 4651.018055555555, 3782.8161073825504, 3365.7580645161293, 4838.777777777777], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

```

	Price	Fossil_Hard_Coal	Dates
0	64.949019	5411.263302	2015-01
1	56.383854	4045.974702	2015-02
2	55.522463	4233.818304	2015-03
3	58.354083	4819.516713	2015-04
4	57.294059	4019.612903	2015-05
5	65.974903	6207.095967	2015-06
6	71.072043	6748.469086	2015-07
7	63.998065	5859.370968	2015-08
8	60.254792	5653.873611	2015-09
9	59.406765	5788.694892	2015-10
10	60.726792	5982.631944	2015-11
11	61.901761	5323.676985	2015-12
12	45.578723	4204.615591	2016-01
13	36.752083	3027.655172	2016-02
14	36.818008	2985.886945	2016-03
15	32.618667	2226.186111	2016-04
16	34.691371	2226.651882	2016-05
17	46.266319	2877.080556	2016-06
18	47.502016	4003.916555	2016-07
19	47.602339	4172.501344	2016-08
20	50.405597	4208.013889	2016-09
21	60.182430	4271.040268	2016-10
22	62.581056	4581.631944	2016-11
23	67.595134	4984.295699	2016-12
24	79.492083	5533.782258	2017-01
25	59.837798	4382.532738	2017-02
26	50.959892	2967.849260	2017-03
27	51.717917	3119.322222	2017-04
28	53.772621	4428.919355	2017-05
29	56.258222	5091.152778	2017-06
30	55.252581	4561.283602	2017-07
31	54.084328	3786.232527	2017-08
32	55.816556	4010.465278	2017-09
33	63.925289	4321.130201	2017-10
34	65.430653	5491.216667	2017-11

```
35 65.151277 4330.653226 2017-12  
36 56.511976 3783.622312 2018-01  
37 60.877098 4144.389881 2018-02  
38 48.279717 2157.183042 2018-03  
39 50.400736 2775.879167 2018-04  
40 61.633763 3674.151882 2018-05  
41 64.348139 3059.616667 2018-06  
42 67.783441 4014.707941 2018-07  
43 70.363911 4121.565860 2018-08  
44 76.914042 4651.018056 2018-09  
45 70.362215 3782.816107 2018-10  
46 67.042608 3365.758065 2018-11  
47 66.623514 4838.777778 2018-12
```

```
[83.31555118223427, 71.75768244613677, 76.2541514971327, 82.59090774439244, 70.1575863811435, 94.08268455548271,  
94.95251297335733, 91.55544018468301, 93.8327634155421, 97.44167827794621, 98.51717471397966, 86.0020283827543,  
92.24952573784516, 82.38050466292277, 81.0985466325581, 68.24883842925661, 64.18460324877044, 62.18520492018525  
, 84.28940244626031, 87.65328463237383, 83.48306776997595, 70.96822613837882, 73.21116436550287, 73.737492240128  
32, 69.61425623806792, 73.24020790331005, 58.23892328170889, 60.31415074831686, 82.36383637493897, 90.4961546361  
6182, 82.55331332745612, 70.00609363016835, 71.85081984835243, 67.59656931728608, 83.92422257067331, 66.47073446  
723971, 66.95257523443384, 68.0779801028663, 44.68093766055425, 55.07616318434503, 59.61264859715907, 47.5478657  
1138853, 59.22844709314572, 58.574996537776435, 60.47033746727723, 53.7620386170506, 50.20326906537128, 72.62867  
860508874]
```

ADF Test Statistic : -2.823796312617435

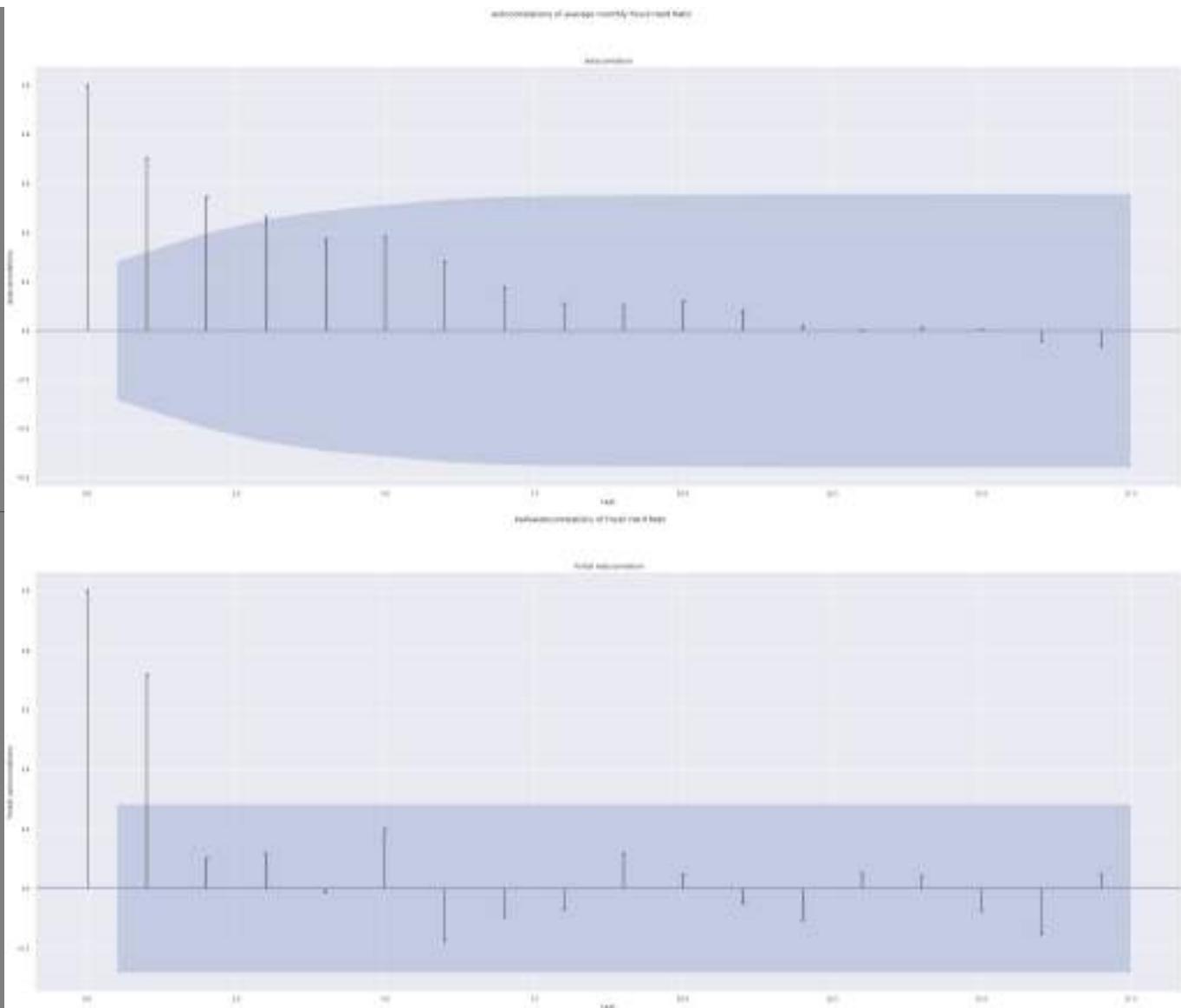
p-value : 0.0549542086667497

#Lags Used : 0

Number of Observations : 47

weak evidence against null hypothesis, indicating it is non-stationary

```
Out[ ]: <function matplotlib.pyplot.show(*args, **kw)>
```



```
In [ ]: Fossil_Hard_Ratio_Autocorrelations = sm.tsa.acf(df_FossilHard["Ratio"], fft=False) #Autocorrelations  
print(Fossil_Hard_Ratio_Autocorrelations)
```

```
[ 1.          0.70408106  0.54281724  0.46429807  0.37396149  0.3847749  
 0.28075634  0.1750466   0.11064319  0.10131795  0.11906783  0.08185152  
 0.01903642  0.00109449  0.01250249  0.00469951 -0.0405789  -0.06209652  
 -0.01635036  0.02686034  0.02807037 -0.0241816   0.03341151  0.0166502  
 -0.00664579 -0.05984499 -0.15512133 -0.18053875 -0.18004537 -0.17295415  
 -0.24809351 -0.29257653 -0.34485231 -0.37199736 -0.2947842  -0.30796218  
 -0.29734834 -0.21179432 -0.21043175 -0.15783117 -0.12533972]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial and autocorrelation plots, indicating that the lags directly beforehand influenced the relationship between average monthly fossil hard coal outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: df_FossilHard['First Difference'] = df_FossilHard["Ratio"] - df_FossilHard["Ratio"].shift(1) # Seasonality value  
df_FossilHard['Seasonal Difference']=df_FossilHard["Ratio"] - df_FossilHard["Ratio"].shift(12)  
df_FossilHard.head()
```

```
Out[ ]:
```

	Price	Fossil_Hard_Coal	Dates	Ratio	First Difference	Seasonal Difference
0	64.949019	5411.263302	2015-01	83.315551	NaN	NaN
1	56.383854	4045.974702	2015-02	71.757682	-11.557869	NaN
2	55.522463	4233.818304	2015-03	76.254151	4.496469	NaN
3	58.354083	4819.516713	2015-04	82.590908	6.336756	NaN
4	57.294059	4019.612903	2015-05	70.157586	-12.433321	NaN

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly Fossil Hard output to price of energy per EUR/MWH ratio")  
plt.ylabel('Seasonality')  
plt.xlabel('Months')
```

```
Out[ ]:
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly fossil hard outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

```
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value))
```

```

if result[1] <= 0.05:
    print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
else:
    print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_FossilHard["Fossil_Hard_Coal"])

test_result=adfuller(df_FossilHard["Fossil_Hard_Coal"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilHard["Fossil_Hard_Coal"])
plt.suptitle(" Autocorrelations of average monthly Fossil Hard Coal")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_FossilHard["Fossil_Hard_Coal"])
plt.suptitle("Partialatocorrelations of Fossil Hard Coal")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show
Fossilhard_Autocorrelations = sm.tsa.acf(df_FossilHard["Fossil_Hard_Coal"], fft=False) #Autocorrelations
print(Fossilhard_Autocorrelations)

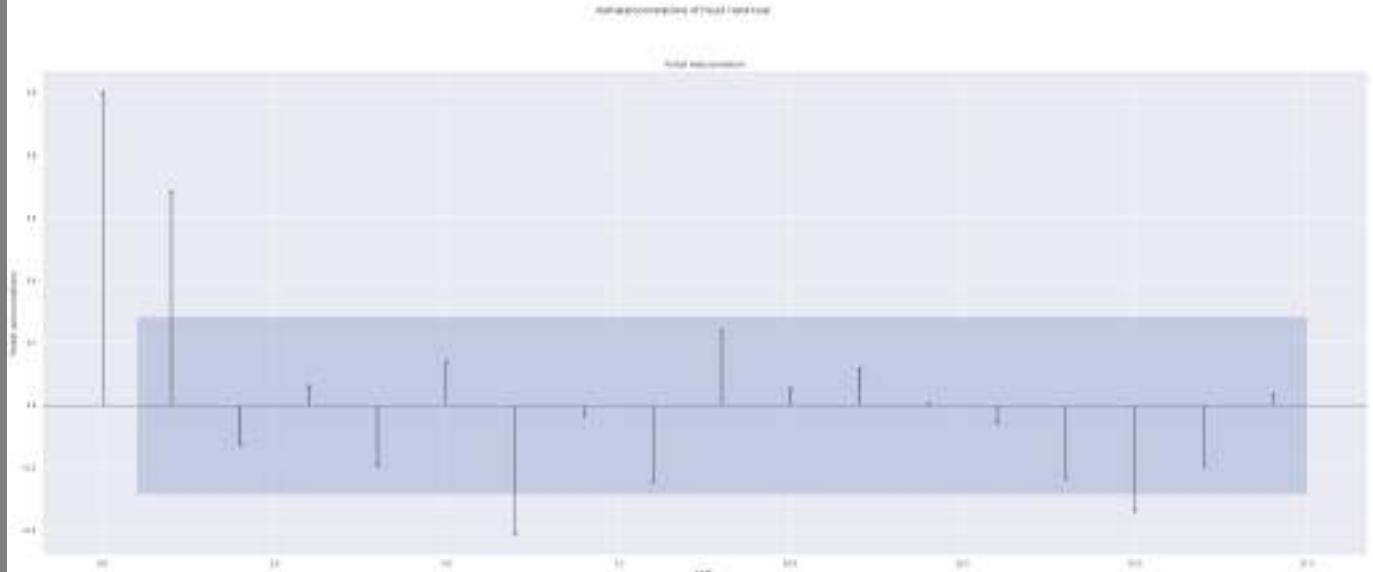
```

ADF Test Statistic : -3.3044420062197006
p-value : 0.014681080539392712
#Lags Used : 5
Number of Observations : 42
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```
[ 1.         0.66809054  0.38141791  0.23530336  0.06006703  0.01961799
 -0.14519694 -0.28635112 -0.33955909 -0.29873817 -0.13309716 -0.04176869
  0.02819511  0.05771468  0.0177532   -0.03097759 -0.0915284  -0.09594781
 -0.01053633  0.01196637 -0.04534828 -0.04002335  0.10014705  0.23042636
  0.24452614  0.17696681  0.08096435  0.00403787 -0.01465317 -0.04035599
 -0.12654668 -0.16460015 -0.1973464   -0.19595344 -0.10031535 -0.12045646
 -0.09982701 -0.01641022 -0.05728641 -0.04954641 -0.05149528]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,





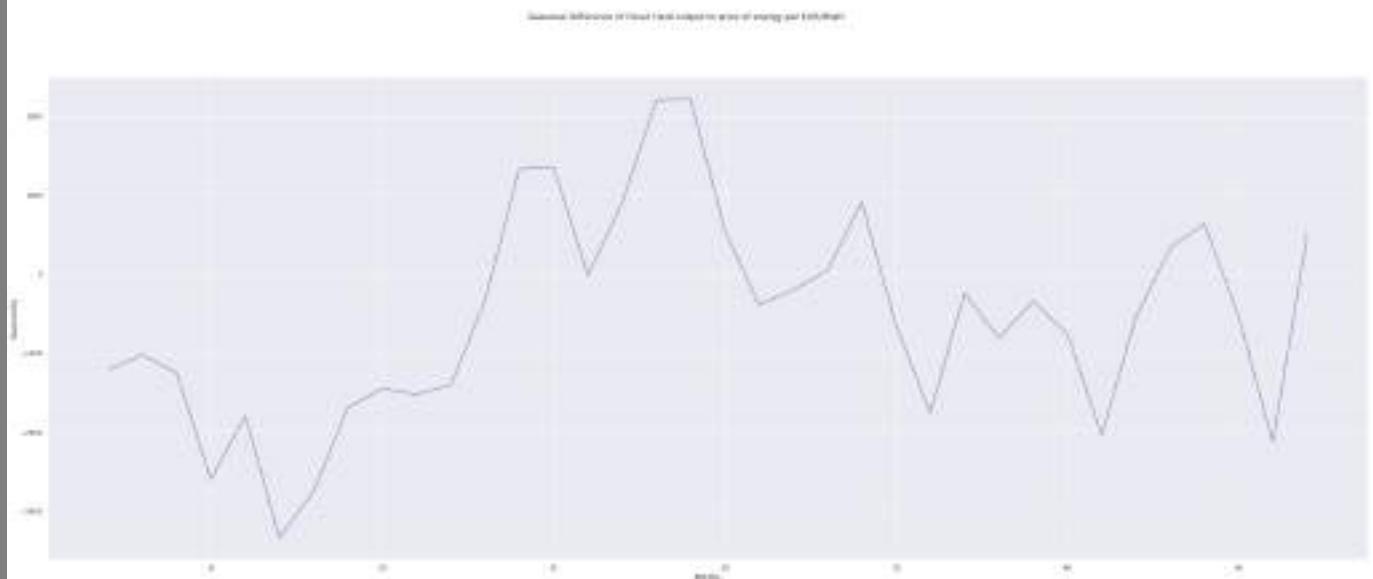
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial and autocorrelation plot, indicating that the lags directly beforehand influenced the average monthly fossil hard coal outputs.

In []:

```
df_FossilHard['First Difference Fossil Hard '] = df_FossilHard["Fossil_Hard_Coal"]- df_FossilHard["Fossil_Hard_Coal"].shift(1)
df_FossilHard['Seasonal Difference Fossil Hard']=df_FossilHard["Fossil_Hard_Coal"]- df_FossilHard["Fossil_Hard_Coal"].shift(12)
df_FossilHard.head()
plt.suptitle("Seasonal Difference of Fossil Hard output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilHard['Seasonal Difference Fossil Hard'].plot() # Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606a29ad0>



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted in the average monthly fossil hard outputs.

```
In [ ]: #Bell Curves

FossilHardResults_mean = np.mean(df_FossilHard["Ratio"])
FossilHardResults_std = np.std(df_FossilHard["Ratio"])

FossilHardResultspdf = stats.norm.pdf(df_FossilHard["Ratio"].sort_values(), FossilHardResults_mean, FossilHardResults_std)

plt.plot(df_FossilHard["Ratio"].sort_values(), FossilHardResultspdf)
plt.xlim([0,200])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_FossilHard["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
plt.suptitle("Frequency distribution of average monthly fossil hard outputs to energy prices per EUR/MWH ratios")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

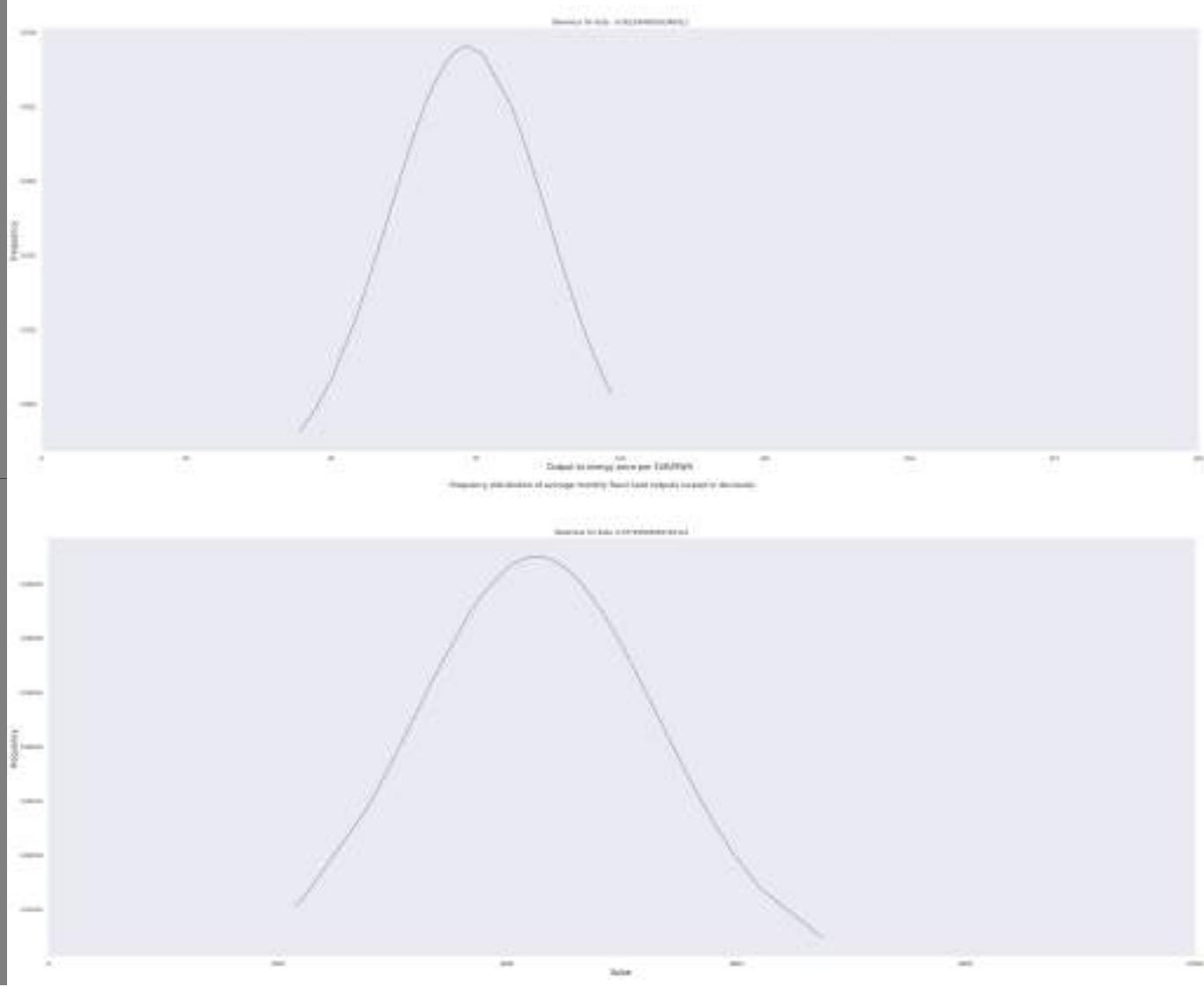
#Bell Curves

FossilHardResults_mean = np.mean(df_FossilHard["Fossil_Hard_Coal"])
FossilHardResults_std = np.std(df_FossilHard["Fossil_Hard_Coal"])

FossilHardResultspdf = stats.norm.pdf(df_FossilHard["Fossil_Hard_Coal"].sort_values(), FossilHardResults_mean, FossilHardResults_std)

plt.plot(df_FossilHard["Fossil_Hard_Coal"].sort_values(), FossilHardResultspdf)
plt.xlim([0,10000])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilHard["Fossil_Hard_Coal"])}')
plt.suptitle("Frequency distribution of average monthly fossil hard outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
```

```
plt.show()
```



These bell shaped curves are roughly symmetrical, hence they have a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: df_FossilHard.describe(include = 'all') # Description Tables
```

	Price	Fossil_Hard_Coal	Dates	Ratio	First Difference	Seasonal Difference	First Difference Fossil Hard	Seasonal Difference Fossil Hard
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	4255.364629	NaN	73.700734	-0.227380	-9.545673	-12.180543	-547.903128
std	10.320573	1073.549370	NaN	13.874850	10.692285	14.776675	863.187000	1325.673639
min	32.618667	2157.183042	NaN	44.680938	-23.397042	-42.948289	-1987.206839	-3330.015411
25%	51.528411	3755.650051	NaN	63.684754	-7.325570	-18.894211	-709.327896	-1463.808448
50%	59.622281	4206.314740	NaN	72.239749	0.526328	-10.506403	134.821281	-542.444878
75%	64.999583	4875.157258	NaN	83.593356	4.061210	-3.194163	545.429839	378.531138
max	79.492083	6748.469086	NaN	98.517175	23.925098	28.310950	2187.483063	2214.072222

Below is the box and whisker plot for the distribution of the average monthly outputs of fossil hard coal and its the average monthly prices of energy per EUR/MWH.

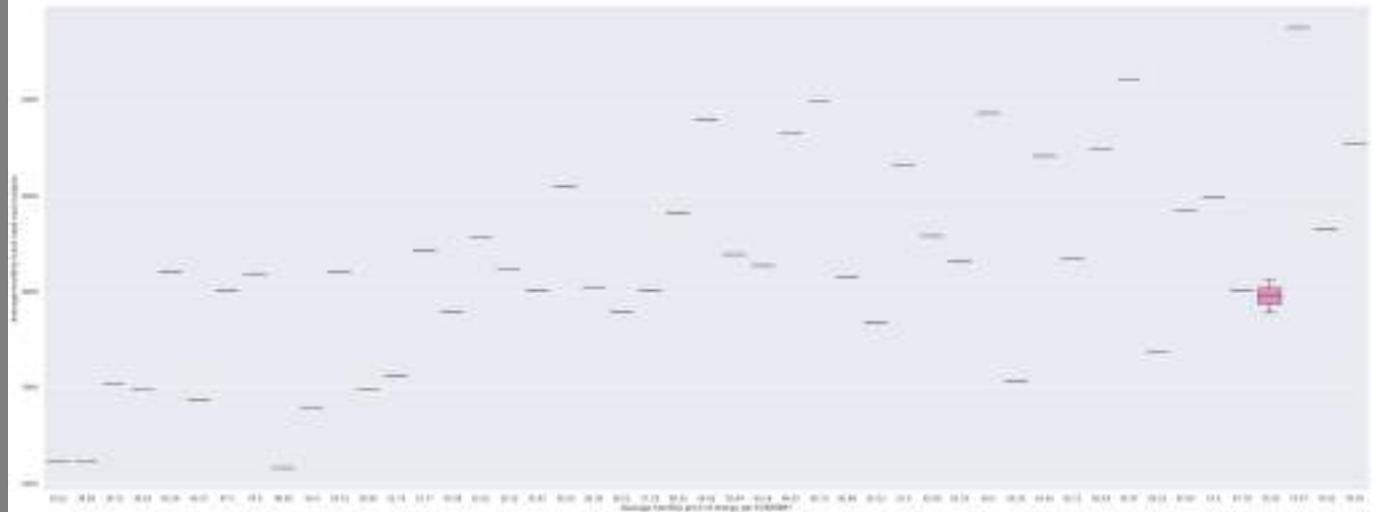
```
In [ ]: # Box and Whisker Plot
```

```

sns.set(style="darkgrid")

plt.suptitle('Distribution of average monthly fossil hard coal outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly fossil hard coal outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')
sns.boxplot(x=Rounded_Y, y= Fossil_Hard)
plt.show()

```



This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil hard coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil hard coal produced, which was slightly above and below 4000 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), fossil hard coal output was at roughly 2250 units. At approximately 71.07 EUR/MWH, fossil hard coal produced its highest output, which was slightly below 7000 units. The lowest amount produced, whcih was slightly above 2000 units, was at the price of approximently 48.28 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 5500 units.

```

In [ ]: plt.suptitle('Distribution of average monthly fossil hard coal outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly fossil hard coal outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

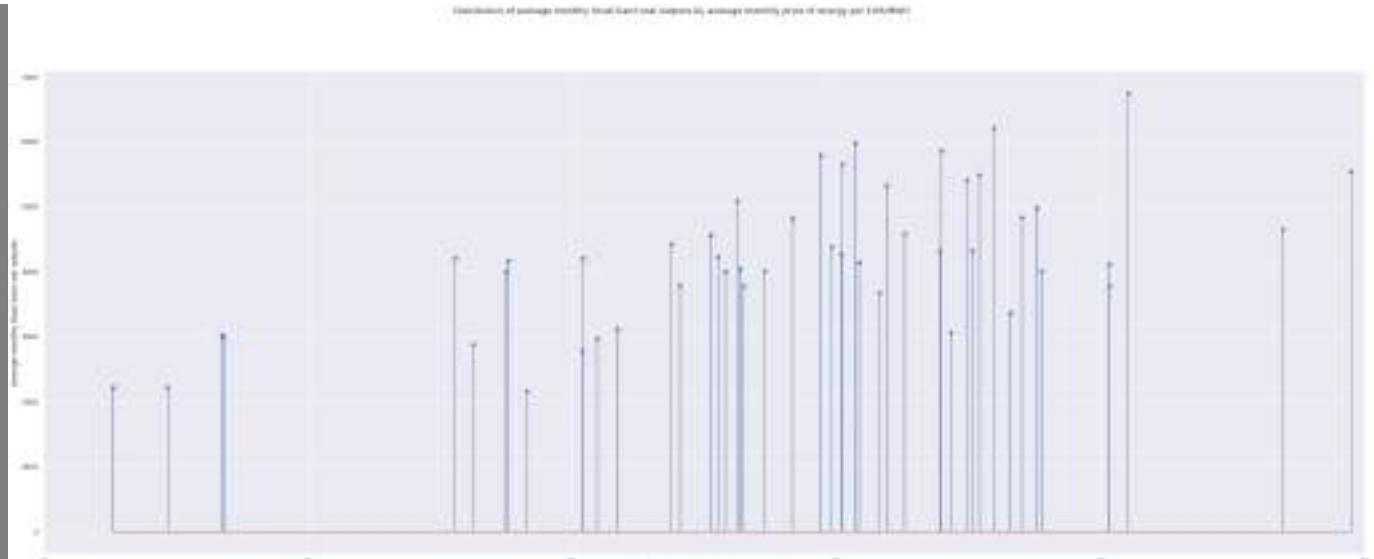
plt.xlim(30, 80)

# Stem Plot
plt.stem(Rounded_Y, Fossil_Hard)

```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of fossil hard coal produced and its the average

monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil hard coal produced, which was slightly above and below 4000 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), fossil hard coal output was at roughly 2250 units. At approximately 71.07 EUR/MWH, fossil hard coal produced its highest output, which was slightly below 7000 units. The lowest amount produced, whcih was slightly above 2000 units, was at the price of approximately 48.28 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 5500 units. Each dot at the end of the blue lines represent an observation.

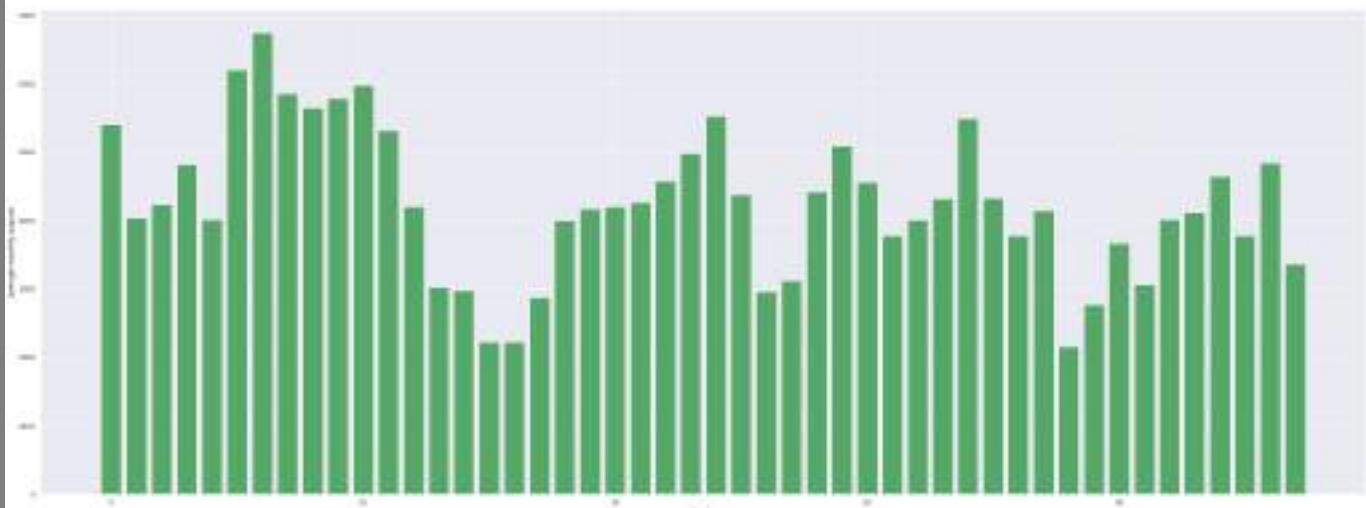
```
In [ ]: Fossil_Hard_Dict = {key: i for i, key in enumerate(Fossil_Hard)}

def Hist_Fossil_Hard(Fossil_Hard_Dict):
    for k, v in Fossil_Hard_Dict.items(): print(f"{v}:{k}") #Outputs in MWH Histogram
print(Fossil_Hard_Dict)

plt.bar(list(Fossil_Hard_Dict.values()), Fossil_Hard_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of fossil hard coal")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{5411.263301500682: 0, 4045.9747023809523: 1, 4233.818304172274: 2, 4819.516713091922: 3, 4019.6129032258063: 4,
6207.095966620306: 5, 6748.469086021505: 6, 5859.370967741936: 7, 5653.873611111111: 8, 5788.694892473119: 9, 59
82.631944444444: 10, 5323.676985195155: 11, 4204.615591397849: 12, 3027.655172413793: 13, 2985.886944818304: 14,
2226.186111111111: 15, 2226.65188172043: 16, 2877.0805555555557: 17, 4003.9165545087485: 18, 4172.501344086021:
19, 4208.013888888889: 20, 4271.040268456376: 21, 4581.631944444444: 22, 4984.295698924731: 23, 5533.78225806451
6: 24, 4382.5327380952385: 25, 2967.8492597577388: 26, 3119.322222222222: 27, 4428.919354838709: 28, 5091.152777
777777: 29, 4561.283602150537: 30, 3786.2325268817203: 31, 4010.465277777778: 32, 4321.130201342282: 33, 5491.21
6666666666: 34, 4330.653225806452: 35, 3783.6223118279568: 36, 4144.389880952381: 37, 2157.1830417227457: 38, 27
75.8791666666666: 39, 3674.15188172043: 40, 3059.616666666667: 41, 4014.707940780619: 42, 4121.565860215053: 43,
4651.018055555555: 44, 3782.8161073825504: 45, 4838.777777777777: 46, 3365.7580645161293: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15
-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18
, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03
': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '
17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08
': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is multimodal yet is slightly skewed to the right, indicating a gradual decrease in the average monthly outputs. In addition, there seems to be recurring troughs occurring roughly every ten months. It would be fair to assume that this fluctuating pattern is seasonal.

```
In [ ]: pdToListFossilHard_Dict = {key: i for i, key in enumerate(pdToListFossilHard)}

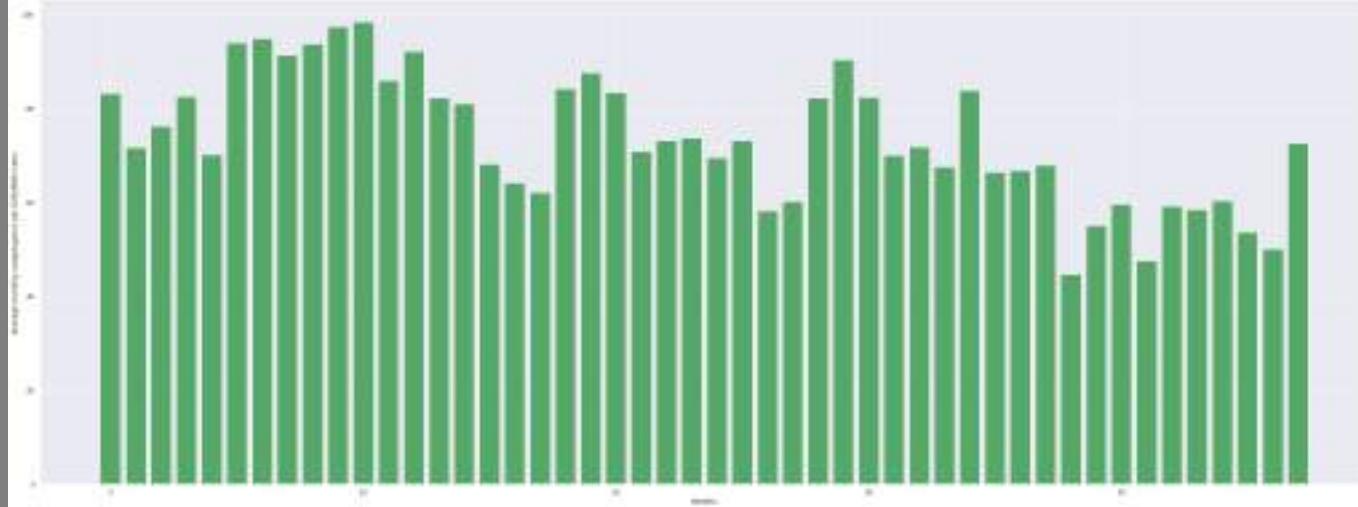
def Hist_pdToListFossilHard(pdToListFossilHard_Dict):
    for k, v in pdToListFossilHard_Dict.items(): print(f"{v}:{k}")
#Output/price per EUR/MWH Histogram
print(pdToListFossilHard_Dict)

plt.bar(list(pdToListFossilHard_Dict.values()), pdToListFossilHard_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of fossil hard coal")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')

plt.show()
```

```
plt.show()
```

```
{83.31555118223427: 0, 71.75768244613677: 1, 76.2541514971327: 2, 82.59090774439244: 3, 70.1575863811435: 4, 94.08268455548271: 5, 94.95251297335733: 6, 91.55544018468301: 7, 93.8327634155421: 8, 97.44167827794621: 9, 98.51717471397966: 10, 86.0020283827543: 11, 92.24952573784516: 12, 82.38050466292277: 13, 81.0985466325581: 14, 68.24883842925661: 15, 64.18460324877044: 16, 62.185204920185825: 17, 84.28940244626031: 18, 87.65328463237383: 19, 83.48306776997595: 20, 70.96822613837882: 21, 73.21116436550287: 22, 73.73749224012832: 23, 69.61425623806792: 24, 73.24020790331005: 25, 58.23892328170889: 26, 60.31415074831686: 27, 82.36383637493897: 28, 90.49615463616182: 29, 82.55331332745612: 30, 70.00609363016835: 31, 71.85081984835243: 32, 67.59656931728608: 33, 83.92422257067331: 34, 66.47073446723971: 35, 66.95257523443384: 36, 68.0779801028663: 37, 44.68093766055425: 38, 55.07616318434503: 39, 59.61264859715907: 40, 47.54786571138853: 41, 59.22844709314572: 42, 58.574996537776435: 43, 60.4703374672723: 44, 53.7620386170506: 45, 50.20326906537128: 46, 72.62867860508874: 47}  
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is roughly uniform that gradually declines in value as the months progress.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of fossil hard coal.

```
In [ ]:
```

```
In [ ]: modelFossilHardcoal = stats.linregress([5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.51671309, 64.9490188172043, 56.383854166666666, 55.522462987886975, 58.35408333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.752083333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.4055972222224, 60.182429530201404, 62.5810555555558, 67.5951344086021, 79.4920833333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.2582222222224, 55.252580645161316, 54.08432795698931,
```

```
55.81655555555558,
63.92528859060403,
65.43065277777781,
65.15127688172035,
56.51197580645163,
60.877098214285674,
48.279717362045766,
50.40073611111113,
61.633763440860214,
64.34813888888884,
67.78344086021498,
70.36391129032262,
76.9140416666666,
70.36221476510062,
67.0426075268817,
66.62351388888881])
```

In []:

[| | |]

In []:

```
#OLS predicted quadratic average monthly ratios versus residuals
FossilHardQuadRatioPredict = FossilHard_ypred/ypred

sns.residplot(x = FossilHardQuadRatioPredict , y = standardized_residualsFossilHardQuad/standardized_residualsPriceread)

plt.suptitle("Predicted average monthly quadratic fossil hard coal energy prices to EUR/MWH versus respective quadratic ratios")
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")
plt.ylabel("Amount from actual values")

plt.legend(..#")
```

Out[]:



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []:

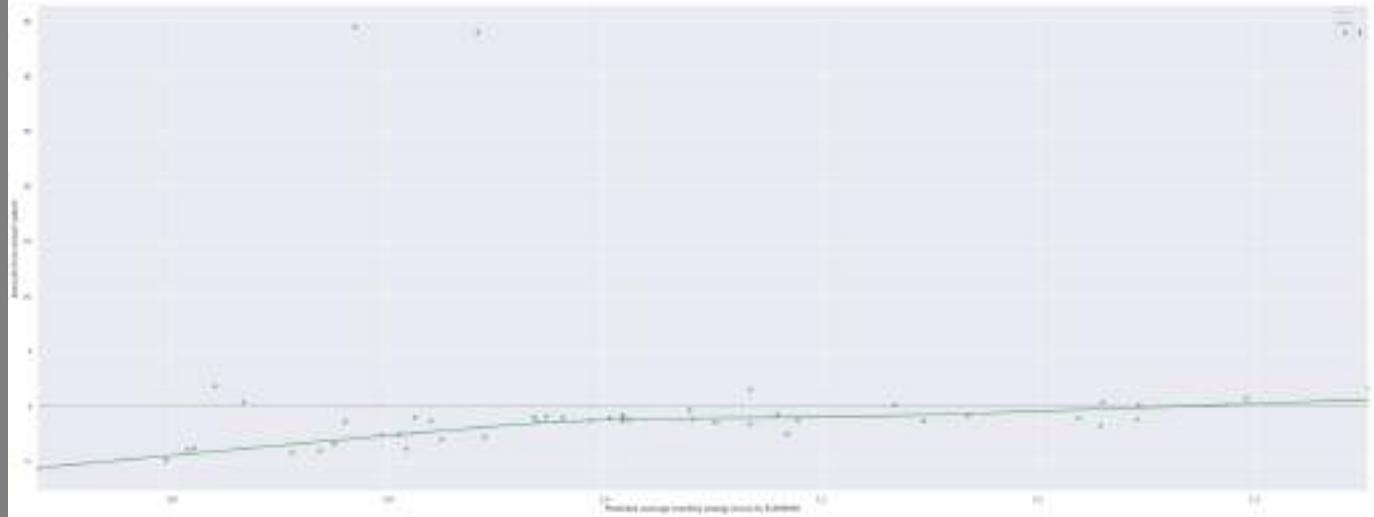
```
#Predicted OLS linear average monthly ratios versus residuals
FossilHardRegRatioPredict = predictionsFossilHard/predictions

sns.residplot(x = FossilHardRegRatioPredict, y = standardized_residualsFossilHard/standardized_residualsPriceread)

plt.suptitle("Predicted average monthly linear fossil hard coal energy prices to EUR/MWH versus respective linear ratios")
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")
plt.ylabel("Amount from actual values")

plt.legend(..#")
```

Out[]: <matplotlib.legend.Legend at 0x7ff605f8aa10>



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price
```

```
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

```
In [ ]: print(list(FossilHard_Logpred))

print(list(Logpred))
```

```
[64.95781249637768, 56.57405890141766, 57.72753991109107, 61.32410633998268, 56.41218043222142, 69.8447389465693, 73.16911947601992, 67.7094829199068, 66.44759645610768, 67.2754861806865, 68.46638490913766, 64.41997595788376, 57.54821639783313, 50.32091943648355, 50.06443554839969, 45.39938187418508, 45.40224200648446, 49.396294059688564, 56.31579471562698, 57.35101418247473, 57.569084138782266, 57.956106806822504, 59.863340253149175, 62.335955714349666, 65.71015800281864, 58.640742563503146, 49.953672514804616, 50.88381431370313, 58.92558634314933, 62.99212717843335, 59.73838829210005, 54.97907422271423, 56.356008105151005, 58.26369133889453, 65.44877778611324, 58.322168858327316, 54.96304581681183, 57.17839164665469, 44.97565847136683, 48.774852183986084, 54.29082668852824, 50.51718364942401, 56.38206079556768, 57.03823742136387, 60.289415779483406, 54.9580952009334, 61.442381714439705, 52.39708951433883]
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: dfFossilHard_Log = ({"Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], "Fossil_Hard_Log" : [64.95781249637768, 56.57405890141766, 57.72753991109107, 61.32410633998268, 56.41218043222142, 69.8447389465693, 73.16911947601992, 67.7094829199068, 66.44759645610768, 67.2754861806865, 68.46638490913766, 64.41997595788376, 57.54821639783313, 50.32091943648355, 50.06443554839969, 45.39938187418508, 45.40224200648446, 49.396294059688564, 56.31579471562698, 57.35101418247473, 57.569084138782266, 57.956106806822504, 59.863340253149175, 62.335955714349666, 65.71015800281864, 58.640742563503146, 49.953672514804616, 50.88381431370313, 58.92558634314933, 62.99212717843335, 59.73838829210005, 54.97907422271423, 56.356008105151005, 58.26369133889453, 65.44877778611324, 58.322168858327316, 54.96304581681183, 57.17839164665469, 44.97565847136683, 48.774852183986084, 54.29082668852824, 50.51718364942401, 56.38206079556768, 57.03823742136387, 60.289415779483406, 54.9580952009334, 61.442381714439705, 52.39708951433883], "Date": ["2010-01-01", "2010-02-01", "2010-03-01", "2010-04-01", "2010-05-01", "2010-06-01", "2010-07-01", "2010-08-01", "2010-09-01", "2010-10-01", "2010-11-01", "2010-12-01", "2011-01-01", "2011-02-01", "2011-03-01", "2011-04-01", "2011-05-01", "2011-06-01", "2011-07-01", "2011-08-01", "2011-09-01", "2011-10-01", "2011-11-01", "2011-12-01", "2012-01-01", "2012-02-01", "2012-03-01", "2012-04-01", "2012-05-01", "2012-06-01", "2012-07-01", "2012-08-01", "2012-09-01", "2012-10-01", "2012-11-01", "2012-12-01", "2013-01-01", "2013-02-01", "2013-03-01", "2013-04-01", "2013-05-01", "2013-06-01", "2013-07-01", "2013-08-01", "2013-09-01", "2013-10-01", "2013-11-01", "2013-12-01", "2014-01-01", "2014-02-01", "2014-03-01", "2014-04-01", "2014-05-01", "2014-06-01", "2014-07-01", "2014-08-01", "2014-09-01", "2014-10-01", "2014-11-01", "2014-12-01", "2015-01-01", "2015-02-01", "2015-03-01", "2015-04-01", "2015-05-01", "2015-06-01", "2015-07-01", "2015-08-01", "2015-09-01", "2015-10-01", "2015-11-01", "2015-12-01", "2016-01-01", "2016-02-01", "2016-03-01", "2016-04-01", "2016-05-01", "2016-06-01", "2016-07-01", "2016-08-01", "2016-09-01", "2016-10-01", "2016-11-01", "2016-12-01", "2017-01-01", "2017-02-01", "2017-03-01", "2017-04-01", "2017-05-01", "2017-06-01", "2017-07-01", "2017-08-01", "2017-09-01", "2017-10-01", "2017-11-01", "2017-12-01"], "Logpred": [27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12], "FossilHard_Logpred": [64.95781249637768, 56.57405890141766, 57.72753991109107, 61.32410633998268, 56.41218043222142, 69.8447389465693, 73.16911947601992, 67.7094829199068, 66.44759645610768, 67.2754861806865, 68.46638490913766, 64.41997595788376, 57.54821639783313, 50.32091943648355, 50.06443554839969, 45.39938187418508, 45.40224200648446, 49.396294059688564, 56.31579471562698, 57.35101418247473, 57.569084138782266, 57.956106806822504, 59.863340253149175, 62.335955714349666, 65.71015800281864, 58.640742563503146, 49.953672514804616, 50.88381431370313, 58.92558634314933, 62.99212717843335, 59.73838829210005, 54.97907422271423, 56.356008105151005, 58.26369133889453, 65.44877778611324, 58.322168858327316, 54.96304581681183, 57.17839164665469, 44.97565847136683, 48.774852183986084, 54.29082668852824, 50.51718364942401, 56.38206079556768, 57.03823742136387, 60.289415779483406, 54.9580952009334, 61.442381714439705, 52.39708951433883]}]

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']

In [ ]:
```

```
for value,label in zip(result,labels):
    print(label+' : '+str(value) )

if result[1] <= 0.05:
    print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
else:
    print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_FossilHard_Log["Fossil_Hard_Log"])

test_result=adfuller(df_FossilHard_Log["Fossil_Hard_Log"])

df_FossilHard_Log['First Difference'] = df_FossilHard_Log["Fossil_Hard_Log"]- df_FossilHard_Log["Fossil_Hard_Log"].shift(1)
df_FossilHard_Log['Seasonal Difference']=df_FossilHard_Log["Fossil_Hard_Log"]- df_FossilHard_Log["Fossil_Hard_Log"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilHard_Log.head() #Predictive Logarithmic Seasonality Plot
df_FossilHard_Log['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.0120010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Fossil_Hard_Log': [64.95781249637768, 56.57405890141766, 57.72753991109107, 61.32410633998268, 56.41218043222142, 69.8447389465693, 73.16911947601992, 67.7094829199068, 66.44759645610768, 67.2754861806865, 68.46638490913766, 64.41997595788376, 57.54821639783313, 50.32091943648355, 50.06443554839969, 45.39938187418508, 45.40224200648446, 49.396294059688564, 56.31579471562698, 57.35101418247473, 57.569084138782266, 57.956106806822504, 59.863340253149175, 62.335955714349666, 65.71015800281864, 58.640742563503146, 49.953672514804616, 50.88381431370313, 58.92558634314933, 62.99212717843335, 59.73838829210005, 54.97907422271423, 56.356008105151005, 58.26369133889453, 65.4487778611324, 58.322168858327316, 54.96304581681183, 57.17839164665469, 44.97565847136683, 48.774852183986084, 54.29082668852824, 50.51718364942401, 56.38206079556768, 57.03823742136387, 60.289415779483406, 54.9580952009334, 61.442381714439705, 52.39708951433883], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}
```

	Price_Log	Fossil_Hard_Log	Dates
0	27.297348	64.957812	2015-01
1	23.917581	56.574059	2015-02
2	23.577680	57.727540	2015-03
3	24.695022	61.324106	2015-04
4	24.276743	56.412180	2015-05
5	27.702157	69.844739	2015-06
6	29.713461	73.169119	2015-07
7	26.922107	67.709483	2015-08
8	25.445032	66.447596	2015-09
9	25.110405	67.275486	2015-10
10	25.631280	68.466385	2015-11
11	26.094917	64.419976	2015-12
12	19.653934	57.548216	2016-01
13	16.170990	50.320919	2016-02
14	16.197004	50.064436	2016-03
15	14.539966	45.399382	2016-04
16	15.357844	45.402242	2016-05
17	19.925256	49.396294	2016-06
18	20.412855	56.315795	2016-07
19	20.452442	57.351014	2016-08
20	21.558593	57.569084	2016-09
21	25.416478	57.956107	2016-10
22	26.362963	59.863340	2016-11
23	28.341491	62.335956	2016-12
24	33.035963	65.710158	2017-01
25	25.280488	58.640743	2017-02
26	21.777315	49.953673	2017-03
27	22.076427	50.883814	2017-04
28	22.887202	58.925586	2017-05
29	23.868007	62.992127	2017-06
30	23.471186	59.738388	2017-07
31	23.010200	54.979074	2017-08
32	23.693728	56.356008	2017-09
33	26.893390	58.263691	2017-10
34	27.487399	65.448778	2017-11
35	27.377158	58.322169	2017-12
36	23.968137	54.963046	2018-01
37	25.690591	57.178392	2018-02
38	20.719732	44.975658	2018-03
39	21.556675	48.774852	2018-04
40	25.989167	54.290827	2018-05
41	27.060244	50.517184	2018-06
42	28.415796	56.382061	2018-07
43	29.434036	57.038237	2018-08
44	32.018682	60.289416	2018-09
45	29.433366	54.958095	2018-10
46	27.958095	61.442382	2018-11
47	28.123467	52.397090	2018-12

ADF Test Statistic : -3.034423210792851

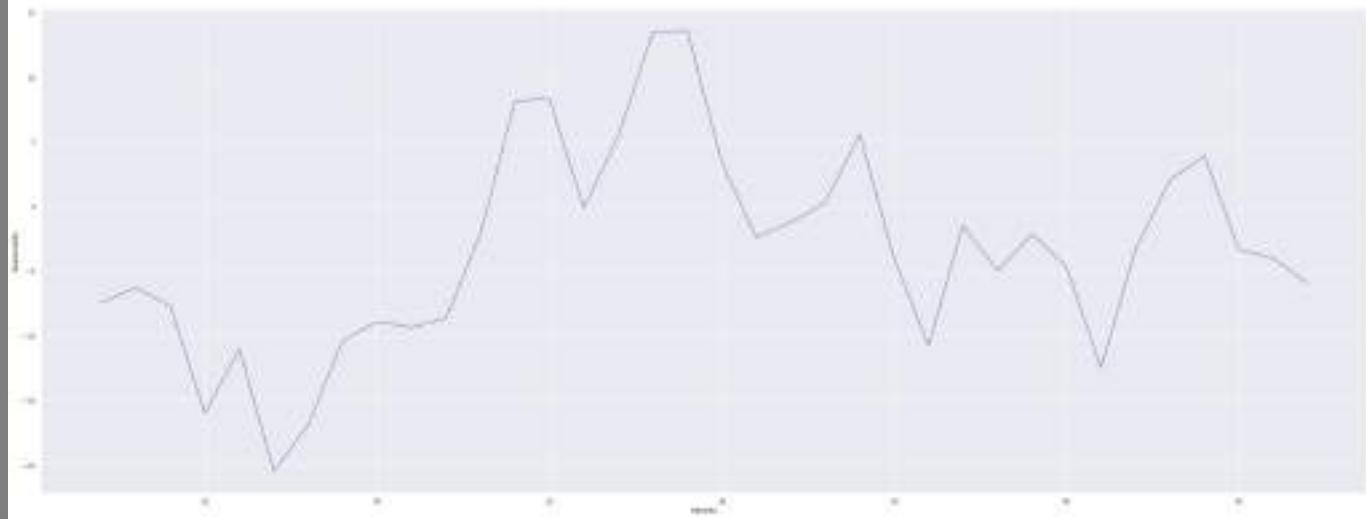
p-value : 0.03180237376567887

#Lags Used : 0

Number of Observations : 47

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff605f49a50>



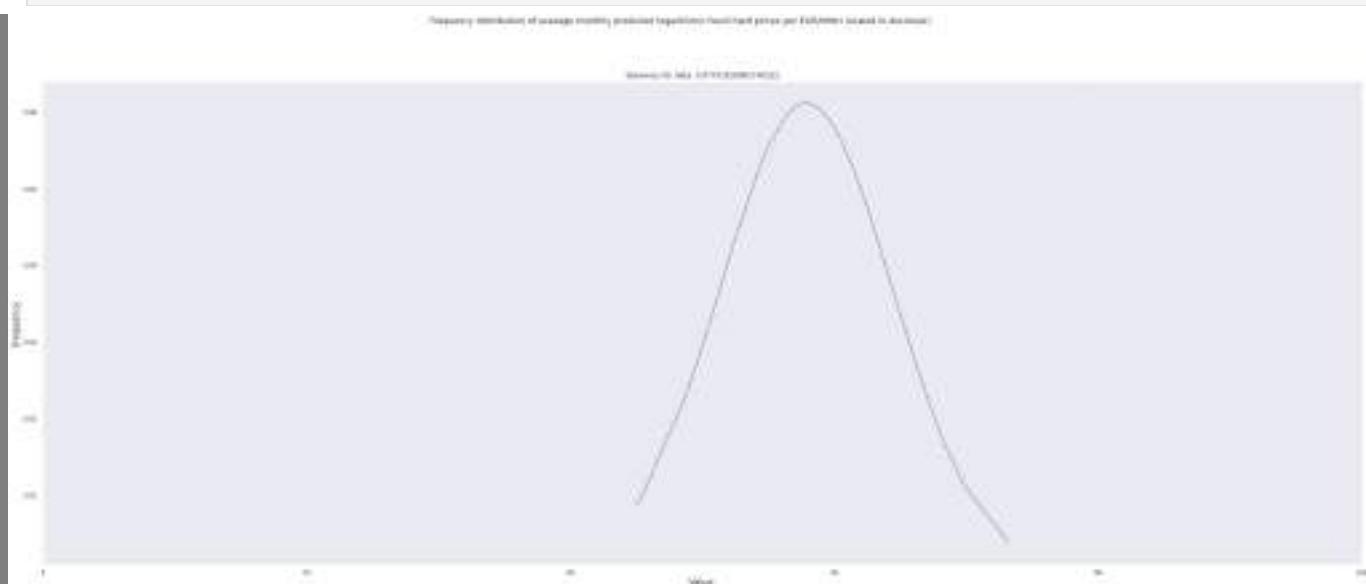
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
FossilHard_LogResults_mean = np.mean(df_FossilHard_Log["Fossil_Hard_Log"])
FossilHard_LogResults_std = np.std(df_FossilHard_Log["Fossil_Hard_Log"])

FossilHard_LogResultspdf = stats.norm.pdf(df_FossilHard_Log["Fossil_Hard_Log"].sort_values(), FossilHard_LogResults_mean, FossilHard_LogResults_std)

plt.plot(df_FossilHard_Log["Fossil_Hard_Log"].sort_values(), FossilHard_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilHard_Log["Fossil_Hard_Log"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic fossil hard prices per EUR/MWH (skewness: -0.1111111111111111)", size=10)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetrical, hence it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
Fossil_Hard_Log_Dict = {key: i for i, key in enumerate(df_FossilHard_Log["Fossil_Hard_Log"])}

def Hist_Fossil_Hard_Log(Fossil_Hard_Log_Dict):
    for k, v in Fossil_Hard_Log_Dict.items(): print(f"{v}:{k}")
print(Fossil_Hard_Log_Dict)
```

```

plt.bar(list(Fossil_Hard_Log_Dict.values()), Fossil_Hard_Log_Dict.keys(), color='g') #Predictive Logarithmic History
plt.title("Average monthly output of fossil hard predicted logarithmic prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

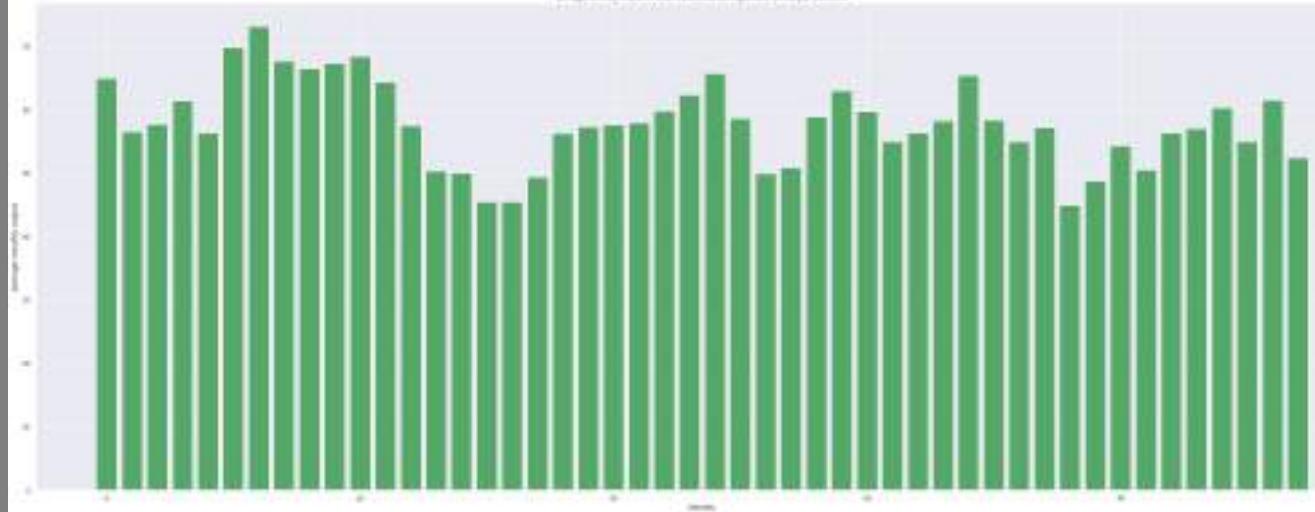
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{64.95781249637768: 0, 56.57405890141766: 1, 57.72753991109107: 2, 61.32410633998268: 3, 56.41218043222142: 4, 69.8447389465693: 5, 73.16911947601992: 6, 67.7094829199068: 7, 66.44759645610768: 8, 67.2754861806865: 9, 68.46638490913766: 10, 64.41997595788376: 11, 57.54821639783313: 12, 50.32091943648355: 13, 50.06443554839969: 14, 45.39938187418508: 15, 45.40224200648446: 16, 49.396294059688564: 17, 56.31579471562698: 18, 57.35101418247473: 19, 57.569084138782266: 20, 57.956106806822504: 21, 59.863340253149175: 22, 62.335955714349666: 23, 65.71015800281864: 24, 58.640742563503146: 25, 49.953672514804616: 26, 50.88381431370313: 27, 58.92558634314933: 28, 62.99212717843335: 29, 59.73838829210005: 30, 54.97907422271423: 31, 56.356008105151005: 32, 58.26369133889453: 33, 65.4487778611324: 34, 58.322168858327316: 35, 54.96304581681183: 36, 57.17839164665469: 37, 44.97565847136683: 38, 48.774852183986084: 39, 54.29082668852824: 40, 50.51718364942401: 41, 56.38206079556768: 42, 57.03823742136387: 43, 60.289415779483406: 44, 54.9580952009334: 45, 61.442381714439705: 46, 52.39708951433883: 47}

```



The green bars represent the observation value for each respective month. This histogram has trenches every ten months but is decreasing in value overtime. It is multimodal.

```
In [ ]: df_FossilHard_Log.describe(include = 'all') # Description Tables
```

	Price_Log	Fossil_Hard_Log	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859848	NaN	-0.267249	-3.364479
std	4.072442	6.592286	NaN	5.366542	7.911008
min	14.539966	44.975658	NaN	-12.202733	-20.448445
25%	22.001649	54.791278	NaN	-4.712184	-8.671912
50%	25.195447	57.558650	NaN	0.827890	-3.681362
75%	27.317301	61.665775	NaN	3.349291	0.745479
max	33.035963	73.169119	NaN	13.432559	13.595833

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price
```

```
[52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04, 56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59.68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]
```

```
In [ ]: print(list(predictions))
print(list(predictionsFossilHard))
```

```
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.
03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373
92337, 62.89808335677895]
[64.95781249637768, 56.57405890141766, 57.72753991109107, 61.32410633998268, 56.41218043222142, 69.8447389465693
, 73.16911947601992, 67.7094829199068, 66.44759645610768, 67.2754861806865, 68.46638490913766, 64.41997595788376
, 57.54821639783313, 50.32091943648355, 50.06443554839969, 45.39938187418508, 45.40224200648446, 49.396294059688
564, 56.31579471562698, 57.35101418247473, 57.569084138782266, 57.956106806822504, 59.863340253149175, 62.335955
714349666, 65.71015800281864, 58.640742563503146, 49.953672514804616, 50.88381431370313, 58.92558634314933, 62.9
9212717843335, 59.73838829210005, 54.97907422271423, 56.356008105151005, 58.26369133889453, 65.44877778611324, 5
8.322168858327316, 54.96304581681183, 57.17839164665469, 44.97565847136683, 48.774852183986084, 54.2908266885282
4, 50.51718364942401, 56.38206079556768, 57.03823742136387, 60.289415779483406, 54.9580952009334, 61.44238171443
9705, 52.39708951433883]
```

```
In [ ]: dfFossilHardReg = ({ "Price_Reg": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53
        "Fossil_HardReg" : [64.95781249637768, 56.57405890141766, 57.72753991109107, 61.32410633998268, 56.41218043222
        "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10']
print(dfFossilHardReg) #Dataframes
df_FossilHardReg= pd.DataFrame.from_dict(dfFossilHardReg, orient = "columns")
print(df_FossilHardReg)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_FossilHardReg["Fossil_HardReg"])

test_result=adfuller(df_FossilHardReg["Fossil_HardReg"])

df_FossilHardReg['First Difference'] = df_FossilHardReg["Fossil_HardReg"]- df_FossilHardReg["Fossil_HardReg"].shift(1)
df_FossilHardReg['Seasonal Difference']=df_FossilHardReg["Fossil_HardReg"]- df_FossilHardReg["Fossil_HardReg"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilHardReg.head() #Predictive Linear Seasonality Plot
df_FossilHardReg['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Reg': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.251900888255 93, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878 538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.6102236996732, 58.824616 68252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.110 974579653693, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.182939493943, 61.3 9733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'Fossil_HardReg': [64.95781249637768, 56.57405890141766, 57.7275399110910 7, 61.32410633998268, 56.41218043222142, 69.8447389465693, 73.16911947601992, 67.7094829199068, 66.4475964561076 8, 67.2754861806865, 68.46638490913766, 64.41997595788376, 57.54821639783313, 50.32091943648355, 50.064435548399 69, 45.39938187418508, 45.40224200648446, 49.396294059688564, 56.31579471562698, 57.35101418247473, 57.569084138 782266, 57.956106806822504, 59.863340253149175, 62.335955714349666, 65.71015800281864, 58.640742563503146, 49.95 3672514804616, 50.88381431370313, 58.92558634314933, 62.99212717843335, 59.73838829210005, 54.97907422271423, 56 .356008105151005, 58.26369133889453, 65.44877778611324, 58.322168858327316, 54.96304581681183, 57.17839164665469 , 44.97565847136683, 48.774852183986084, 54.29082668852824, 50.51718364942401, 56.38206079556768, 57.03823742136 387, 60.289415779483406, 54.9580952009334, 61.442381714439705, 52.39708951433883], 'Dates': ['2015-01', '2015-02 ', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Reg	Fossil_HardReg	Dates
0	52.821613	64.957812	2015-01
1	53.036006	56.574059	2015-02
2	53.250399	57.727540	2015-03
3	53.464792	61.324106	2015-04
4	53.679185	56.412180	2015-05
5	53.893578	69.844739	2015-06
6	54.107971	73.169119	2015-07
7	54.322364	67.709483	2015-08
8	54.536757	66.447596	2015-09
9	54.751150	67.275486	2015-10
10	54.965543	68.466385	2015-11
11	55.179936	64.419976	2015-12
12	55.394329	57.548216	2016-01
13	55.608722	50.320919	2016-02
14	55.823115	50.064436	2016-03
15	56.037508	45.399382	2016-04
16	56.251901	45.402242	2016-05
17	56.466294	49.396294	2016-06
18	56.680687	56.315795	2016-07
19	56.895080	57.351014	2016-08
20	57.109473	57.569084	2016-09
21	57.323866	57.956107	2016-10
22	57.538259	59.863340	2016-11
23	57.752652	62.335956	2016-12
24	57.967045	65.710158	2017-01
25	58.181438	58.640743	2017-02
26	58.395831	49.953673	2017-03
27	58.610224	50.883814	2017-04
28	58.824617	58.925586	2017-05
29	59.039010	62.992127	2017-06
30	59.253403	59.738388	2017-07
31	59.467796	54.979074	2017-08
32	59.682189	56.356008	2017-09
33	59.896582	58.263691	2017-10
34	60.110975	65.448778	2017-11
35	60.325368	58.322169	2017-12
36	60.539761	54.963046	2018-01
37	60.754154	57.178392	2018-02
38	60.968547	44.975658	2018-03
39	61.182939	48.774852	2018-04
40	61.397332	54.290827	2018-05
41	61.611725	50.517184	2018-06
42	61.826118	56.382061	2018-07
43	62.040511	57.038237	2018-08
44	62.254904	60.289416	2018-09
45	62.469297	54.958095	2018-10
46	62.683690	61.442382	2018-11
47	62.898083	52.397090	2018-12

ADF Test Statistic : -3.034423210792851

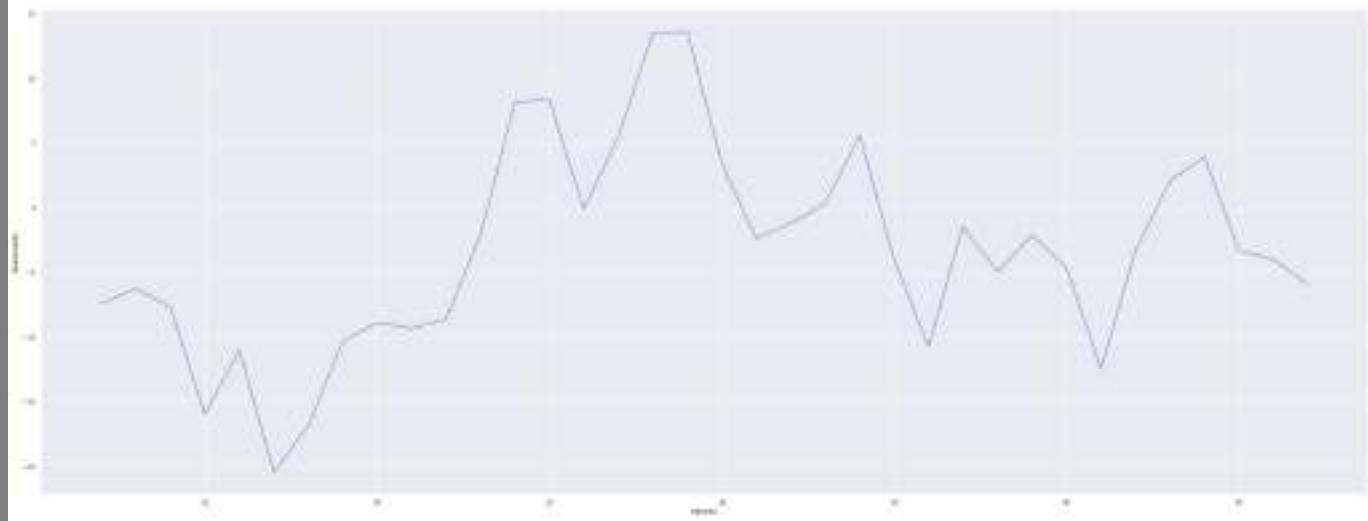
p-value : 0.03180237376567887

#Lags Used : 0

Number of Observations : 47

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6063ea550>



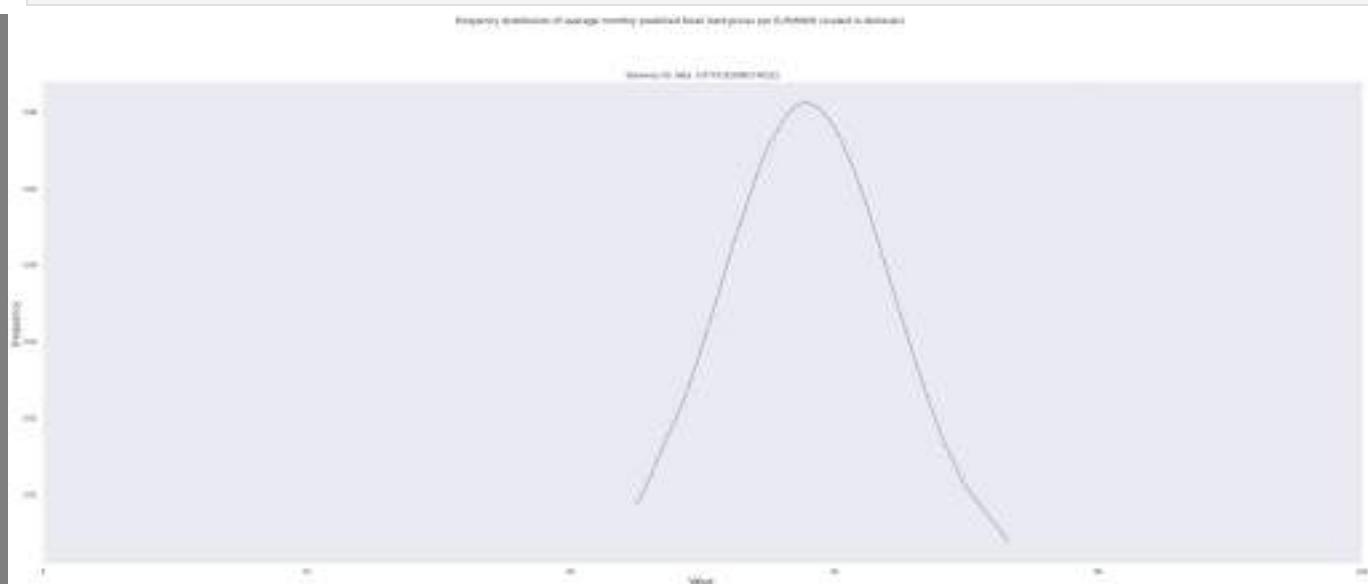
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
FossilHardRegResults_mean = np.mean(df_FossilHardReg["Fossil_HardReg"])
FossilHardRegResults_std = np.std(df_FossilHardReg["Fossil_HardReg"])

FossilHardRegResultspdf = stats.norm.pdf(df_FossilHardReg["Fossil_HardReg"].sort_values(), FossilHardRegResults_mean, FossilHardRegResults_std)

plt.plot(df_FossilHardReg["Fossil_HardReg"].sort_values(), FossilHardRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilHardReg["Fossil_HardReg"])}')
plt.suptitle("Frequency distribution of average monthly predicted fossil hard prices per EUR/MWH (scaled in deciles)", size=10)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetrical, hence it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
Fossil_HardReg_Dict = {key: i for i, key in enumerate(df_FossilHardReg["Fossil_HardReg"])}

def Hist_Fossil_HardReg(Fossil_HardReg_Dict):
    for k, v in Fossil_HardReg_Dict.items(): print(f"{v}:{k}")
print(Fossil_HardReg_Dict)
```

```

plt.bar(list(Fossil_HardReg_Dict.values()), Fossil_HardReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted linear fossil hard prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

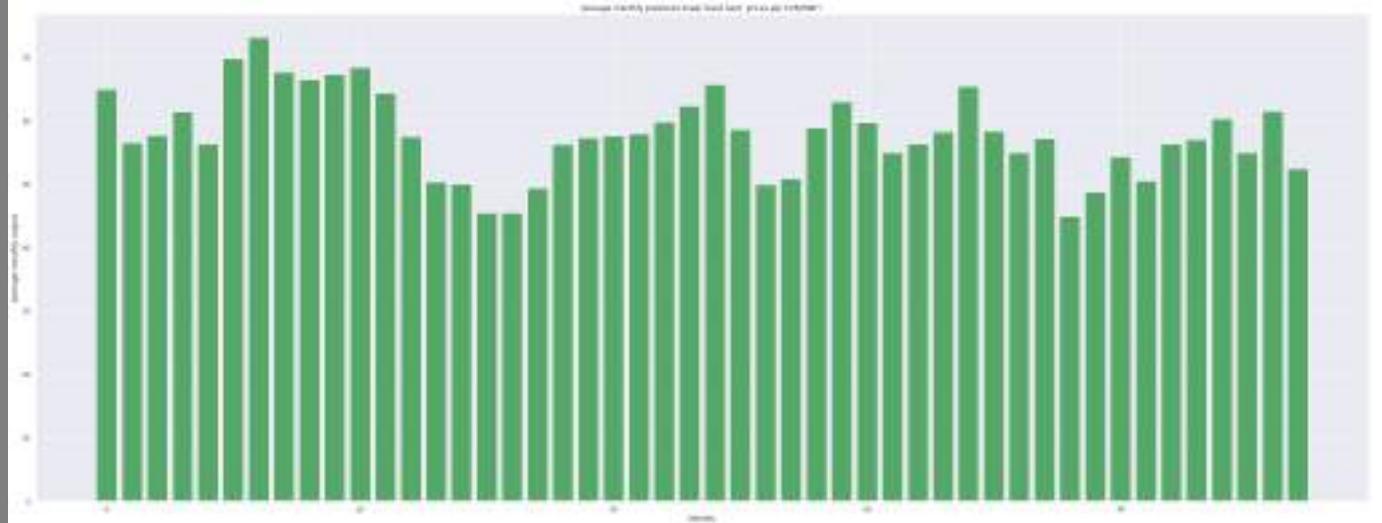
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{64.95781249637768: 0, 56.57405890141766: 1, 57.72753991109107: 2, 61.32410633998268: 3, 56.41218043222142: 4, 69.8447389465693: 5, 73.16911947601992: 6, 67.7094829199068: 7, 66.44759645610768: 8, 67.2754861806865: 9, 68.46638490913766: 10, 64.41997595788376: 11, 57.54821639783313: 12, 50.32091943648355: 13, 50.06443554839969: 14, 45.39938187418508: 15, 45.40224200648446: 16, 49.396294059688564: 17, 56.31579471562698: 18, 57.35101418247473: 19, 57.569084138782266: 20, 57.956106806822504: 21, 59.863340253149175: 22, 62.335955714349666: 23, 65.71015800281864: 24, 58.640742563503146: 25, 49.953672514804616: 26, 50.88381431370313: 27, 58.92558634314933: 28, 62.99212717843335: 29, 59.73838829210005: 30, 54.97907422271423: 31, 56.356008105151005: 32, 58.26369133889453: 33, 65.4487778611324: 34, 58.322168858327316: 35, 54.96304581681183: 36, 57.17839164665469: 37, 44.97565847136683: 38, 48.774852183986084: 39, 54.29082668852824: 40, 50.51718364942401: 41, 56.38206079556768: 42, 57.03823742136387: 43, 60.289415779483406: 44, 54.9580952009334: 45, 61.442381714439705: 46, 52.39708951433883: 47}

```



The green bars represent the observation value for each respective month. This histogram is multimodal with troughs occurring roughly every ten months. The values are decreasing over time.

```
In [ ]: df_FossilHardReg.describe(include = 'all') # Description Tables
```

	Price_Reg	Fossil_HardReg	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	57.859848	NaN	-0.267249	-3.364479
std	3.001502	6.592286	NaN	5.366542	7.911008
min	52.821613	44.975658	NaN	-12.202733	-20.448445
25%	55.340731	54.791278	NaN	-4.712184	-8.671912
50%	57.859848	57.558650	NaN	0.827890	-3.681362
75%	60.378966	61.665775	NaN	3.349291	0.745479
max	62.898083	73.169119	NaN	13.432559	13.595833

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```

In [ ]: Rounded_ypred = [round(item, 2) for item in ypred]
print(Rounded_ypred)
#Rounded Price

print(FossilHard_ypred)

print(ypred)

```

```
[27.19, 23.23, 22.88, 24.06, 23.6, 27.72, 30.59, 26.7, 24.9, 24.52, 25.12, 25.67, 19.55, 17.63, 17.64, 17.08, 17.33, 19.74, 20.09, 20.12, 21.01, 24.87, 26.0, 28.6, 36.04, 24.71, 21.19, 21.46, 22.2, 23.17, 22.77, 22.32, 23.0, 26.67, 27.44, 27.29, 23.28, 25.19, 20.33, 21.01, 25.54, 26.88, 28.7, 30.17, 34.27, 30.17, 28.07, 28.3]
[64.78220667 58.248637 59.49457413 62.66858586 58.06492369 65.89222334
65.51151477 65.65169283 65.33117725 65.55640889 65.78036496 64.538644
59.30814568 49.5672009 49.14164589 40.44614492 40.45203094 48.00736856
57.95450131 59.10004066 59.3299778 59.72831785 61.50925735 63.36753331
65.08252326 60.40244586 48.95617666 50.48195852 60.67142768 63.77525103
61.40184571 56.3434062 58.00066497 60.03600715 64.9835267 60.0936139
56.32318563 58.91521798 39.56662128 46.91902622 55.45590039 49.88914288
58.03050078 58.76333432 61.86575309 56.31693589 62.75468354 52.81044814]
[27.18807769 23.22554817 22.87823628 24.05511741 23.60270863 27.72472243
30.58783893 26.70248162 24.90180391 24.51841487 25.11912136 25.67229968
19.54636257 17.6334062 17.64405232 17.07534443 17.32825464 19.7366291
20.09354637 20.12336851 21.00799158 24.86873542 26.00005506 28.59930479
36.03506727 24.71214951 21.19464431 21.45617264 22.20151115 23.17431072
22.77134503 22.31923277 22.99576214 26.66578905 27.43836997 27.29282952
23.278006 25.18891499 20.32804924 21.00637179 25.54459336 26.87991763
28.70309766 30.1704781 34.27227994 30.16948579 28.0708618 28.29733761]
```

```
In [ ]: print(list(FossilHard_ypred))
```

```
print(list(ypred))
```

```
[64.7822066720878, 58.24863700224269, 59.49457413175417, 62.66858585663662, 58.06492369458065, 65.89222334426583
, 65.51151476834167, 65.65169283458859, 65.3311772471827, 65.5564088746805, 65.78036496226927, 64.5386440012904
4, 59.30814568338462, 49.56720089685194, 49.1416458858873, 40.44614492479379, 40.452030942053156, 48.00736856375
364, 57.954501313687196, 59.100040662572155, 59.329977803493826, 59.72831785220323, 61.50925735320212, 63.367533
309117825, 65.08252326182557, 60.40244586133788, 48.95617665530424, 50.48195852490237, 60.6714276789064, 63.7752
5102556739, 61.401845706696484, 56.34340619599832, 58.00066497290037, 60.03600714715631, 64.9835266977521, 60.09
361389565158, 56.32318563440776, 58.915217978366215, 39.56662128405752, 46.9190262227999, 55.45590038582421, 49.
88914288156278, 58.03050077820774, 58.76333432073122, 61.865753087005984, 56.316935886343074, 62.754683543524116
, 52.810448135530706]
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338
71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672
299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486
, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506
2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20
1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,
27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905
53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857
92181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: import pandas as pd
dffossilHard_Quad = {"Fossil_Hard_Quad": [64.7822066720878, 58.24863700224269, 59.49457413175417, 62.6685858566
"Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398
"Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10']
print(dffossilHard_Quad) #Dataframes
df_FossilHard_Quad= pd.DataFrame.from_dict(dffossilHard_Quad, orient = "columns")
print(df_FossilHard_Quad)
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")
```

```
adfuller_test(df_FossilHard_Quad["Fossil_Hard_Quad"])
```

```
test_result=adfuller(df_FossilHard_Quad["Fossil_Hard_Quad"])
```

```
df_FossilHard_Quad['First Difference'] = df_FossilHard_Quad["Fossil_Hard_Quad"]- df_FossilHard_Quad["Fossil_Hard_Quad"].shift(1)
df_FossilHard_Quad['Seasonal Difference']=df_FossilHard_Quad["Fossil_Hard_Quad"]- df_FossilHard_Quad["Fossil_Hard_Quad"].shift(12)
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilHard_Quad.head() #Predictive Quadratic Seasonality Plot
df_FossilHard_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Fossil_Hard_Quad': [64.7822066720878, 58.24863700224269, 59.49457413175417, 62.66858585663662, 58.064923694580
65, 65.89222334426583, 65.51151476834167, 65.65169283458859, 65.3311772471827, 65.55640888746805, 65.78036496226
927, 64.53864400129044, 59.30814568338462, 49.56720089685194, 49.1416458858873, 40.44614492479379, 40.4520309420
53156, 48.00736856375364, 57.954501313687196, 59.100040662572155, 59.329977803493826, 59.72831785220323, 61.5092
5735320212, 63.367533309117825, 65.08252326182557, 60.40244586133788, 48.95617665530424, 50.48195852490237, 60.6
714276789064, 63.77525102556739, 61.401845706696484, 56.34340619599832, 58.00066497290037, 60.03600714715631, 64
.9835266977521, 60.09361389565158, 56.32318563440776, 58.915217978366215, 39.56662128405752, 46.9190262227999, 5
5.45590038582421, 49.88914288156278, 58.03050077820774, 58.76333432073122, 61.865753087005984, 56.31693588634307
4, 62.754683543524116, 52.810448135530706], 'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204
505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.90180
3909409455, 24.518414867253377, 25.119121355777786, 25.672299682598407, 19.546362569195352, 17.633406195018726,
17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.1233685109
62656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.712
14950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817,
22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.27800599945
75, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.70309766
3581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Da
tes': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-
10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08
', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06',
'2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2
018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Fossil_Hard_Quad  Price_Quad   Dates
0      64.782207  27.188078  2015-01
1      58.248637  23.225548  2015-02
2      59.494574  22.878236  2015-03
3      62.668586  24.055117  2015-04
4      58.064924  23.602709  2015-05
5      65.892223  27.724722  2015-06
6      65.511515  30.587839  2015-07
7      65.651693  26.702482  2015-08
8      65.331177  24.901804  2015-09
9      65.556409  24.518415  2015-10
10     65.780365  25.119121  2015-11
11     64.538644  25.672300  2015-12
12     59.308146  19.546363  2016-01
13     49.567201  17.633406  2016-02
14     49.141646  17.644052  2016-03
15     40.446145  17.075344  2016-04
16     40.452031  17.328255  2016-05
17     48.007369  19.736629  2016-06
18     57.954501  20.093546  2016-07
19     59.100041  20.123369  2016-08
20     59.329978  21.007992  2016-09
21     59.728318  24.868735  2016-10
22     61.509257  26.000055  2016-11
23     63.367533  28.599305  2016-12
24     65.082523  36.035067  2017-01
25     60.402446  24.712150  2017-02
26     48.956177  21.194644  2017-03
27     50.481959  21.456173  2017-04
28     60.671428  22.201511  2017-05
29     63.775251  23.174311  2017-06
30     61.401846  22.771345  2017-07
31     56.343406  22.319233  2017-08
32     58.000665  22.995762  2017-09
33     60.036007  26.665789  2017-10
34     64.983527  27.438370  2017-11
35     60.093614  27.292830  2017-12
36     56.323186  23.278006  2018-01
37     58.915218  25.188915  2018-02
38     39.566621  20.328049  2018-03
39     46.919026  21.006372  2018-04
40     55.455900  25.544593  2018-05
41     49.889143  26.879918  2018-06
42     58.030501  28.703098  2018-07
43     58.763334  30.170478  2018-08
44     61.865753  34.272280  2018-09
45     56.316936  30.169486  2018-10
46     62.754684  28.070862  2018-11
47     52.810448  28.297338  2018-12

ADF Test Statistic : -3.2304013713463733
p-value : 0.01829100544580024
#Lags Used : 0
Number of Observations : 47
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606a3f550>

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

```
In [1]: #Bell Curves
```



This bell shaped curve is skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Fossil_Hard_Quad_Dict = {key: i for i, key in enumerate(df_FossilHard_Quad["Fossil_Hard_Quad"])}
def Hist_Fossil_Hard_Quad(Fossil_Hard_Quad_Dict):
    for k, v in Fossil_Hard_Quad_Dict.items(): print(f"{v}:{k}")
print(Fossil_Hard_Quad_Dict)
```

```

plt.bar(list(Fossil_Hard_Quad_Dict.values()), Fossil_Hard_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic fossil hard prices per EUR/MWh ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

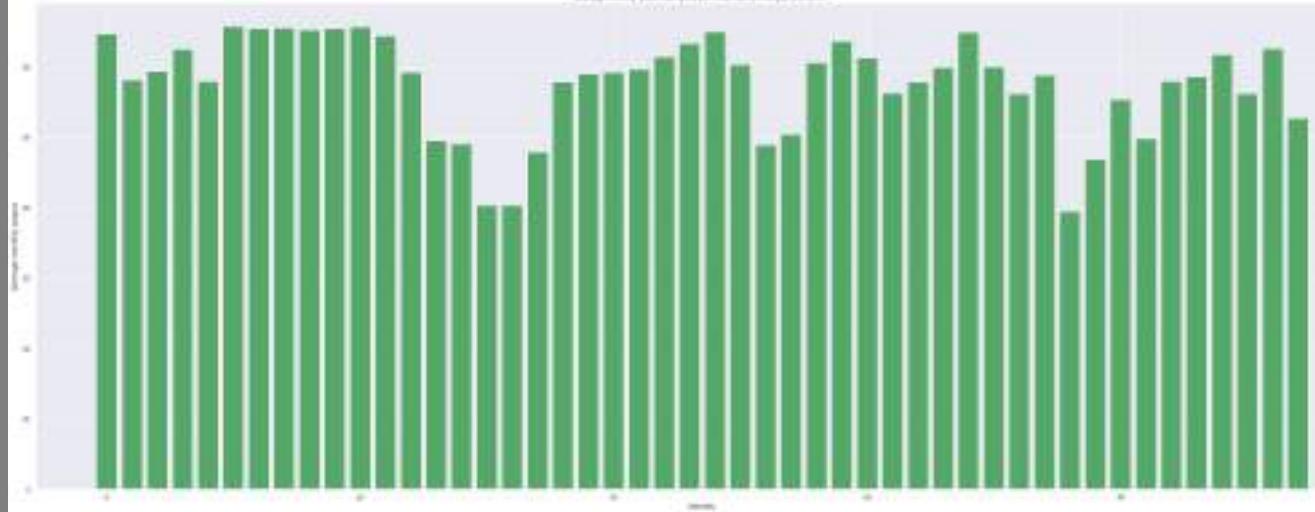
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-010': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{64.7822066720878: 0, 58.24863700224269: 1, 59.49457413175417: 2, 62.66858585663662: 3, 58.06492369458065: 4, 65.8922334426583: 5, 65.51151476834167: 6, 65.65169283458859: 7, 65.3311772471827: 8, 65.55640888746805: 9, 65.78036496226927: 10, 64.53864400129044: 11, 59.30814568338462: 12, 49.56720089685194: 13, 49.1416458858873: 14, 40.44614492479379: 15, 40.452030942053156: 16, 48.00736856375364: 17, 57.954501313687196: 18, 59.100040662572155: 19, 59.329977803493826: 20, 59.72831785220323: 21, 61.50925735320212: 22, 63.367533309117825: 23, 65.08252326182557: 24, 60.40244586133788: 25, 48.95617665530424: 26, 50.48195852490237: 27, 60.6714276789064: 28, 63.77525102556739: 29, 61.401845706696484: 30, 56.34340619599832: 31, 58.00066497290037: 32, 60.03600714715631: 33, 64.9835266977521: 34, 60.09361389565158: 35, 56.32318563440776: 36, 58.915217978366215: 37, 39.56662128405752: 38, 46.9190262227999: 39, 55.45590038582421: 40, 49.88914288156278: 41, 58.03050077820774: 42, 58.76333432073122: 43, 61.865753087005984: 44, 56.316935886343074: 45, 62.754683543524116: 46, 52.810448135530706: 47}

```



The green bars represent the observation value for each respective month. This histogram is roughly uniform with trenches every ten months.

```
In [ ]: df_FossilHard_Quad.describe(include = 'all') # Description Tables
```

	Fossil_Hard_Quad	Price_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	24.500000	NaN	-0.254718	-2.886395
std	6.976536	4.174498	NaN	6.001432	8.755480
min	39.566621	17.075344	NaN	-19.348597	-22.222441
25%	56.101677	21.390791	NaN	-4.641870	-7.351628
50%	59.319062	24.615282	NaN	0.229937	-3.467139
75%	62.907896	27.214266	NaN	2.847226	0.835749
max	65.892223	36.035067	NaN	10.189469	20.219397

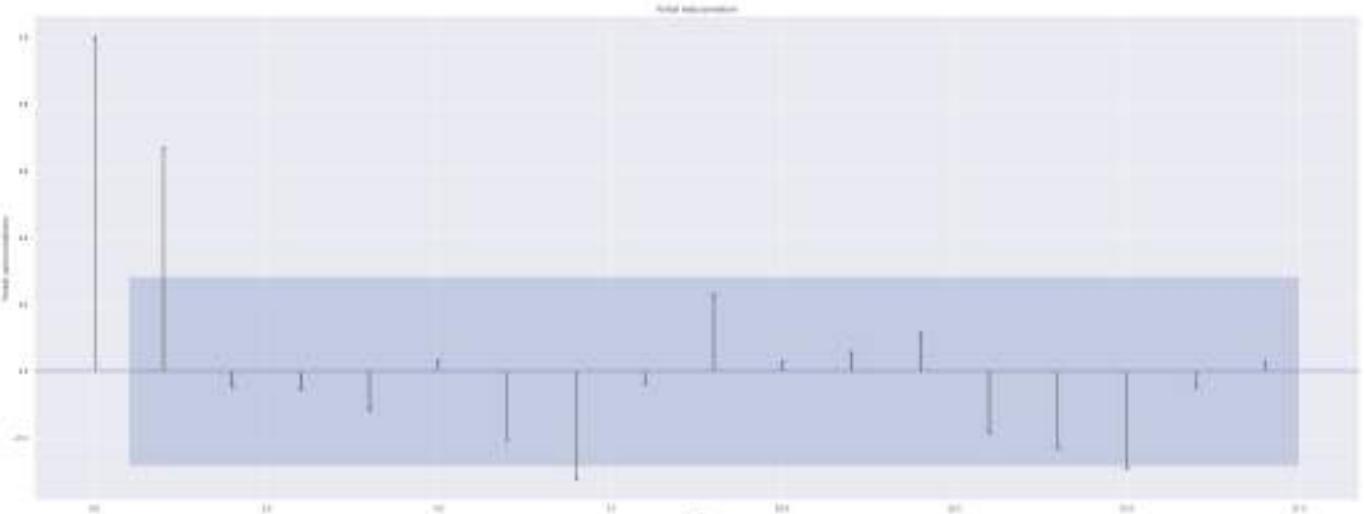
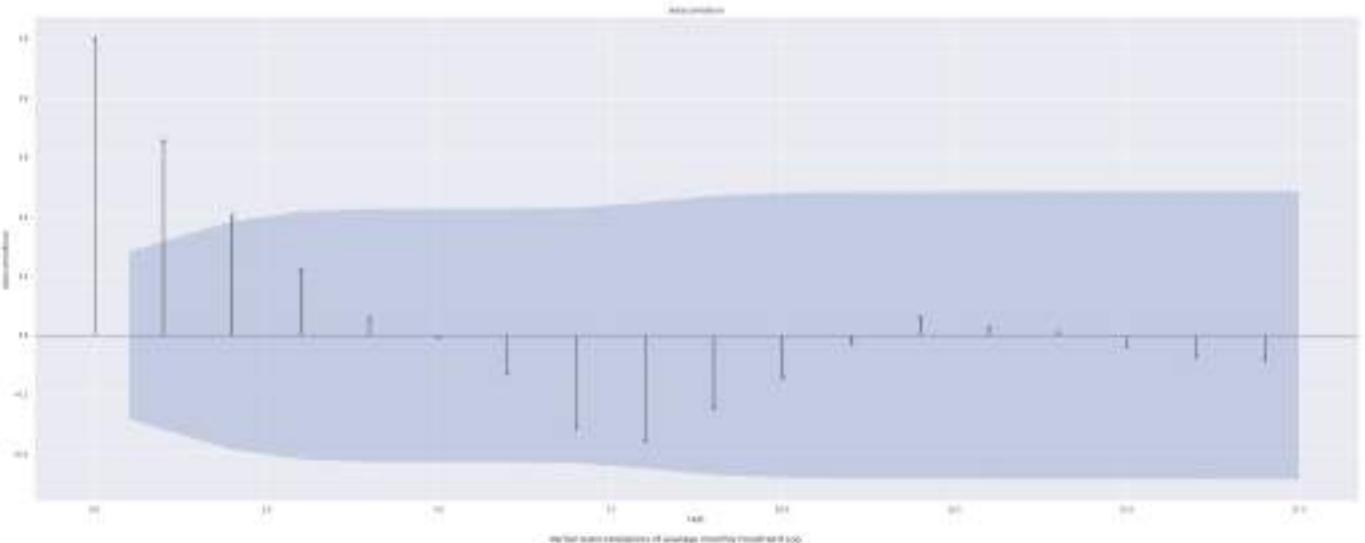
```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilHard_Logpred)
plt.suptitle(" Autocorrelations of average monthly FossilHard Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilHard_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly FossilHard Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

```

```
plt.show  
FossilHard_LogRatio_Autocorrelations = sm.tsa.acf(FossilHard_Logpred, fft=False) #Autocorrelations  
print(FossilHard_LogRatio_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,  
[ 1.          0.65524025  0.40502744  0.22090565  0.05716118 -0.00635438  
-0.12848552 -0.31077839 -0.35638366 -0.24469887 -0.14290773 -0.02689295  
0.059755   0.02589582  0.0093051  -0.03707528 -0.07045197 -0.08153876  
-0.02854484 -0.02364628 -0.04946738 -0.00155302  0.13145367  0.21548385  
0.23357626  0.16852071  0.07925043  0.00307215 -0.01923759 -0.07099871  
-0.14423417 -0.16461282 -0.1766874  -0.19481761 -0.06830956 -0.09002516  
-0.08190765 -0.02168407 -0.06095269 -0.0439582  -0.0273175 ]
```



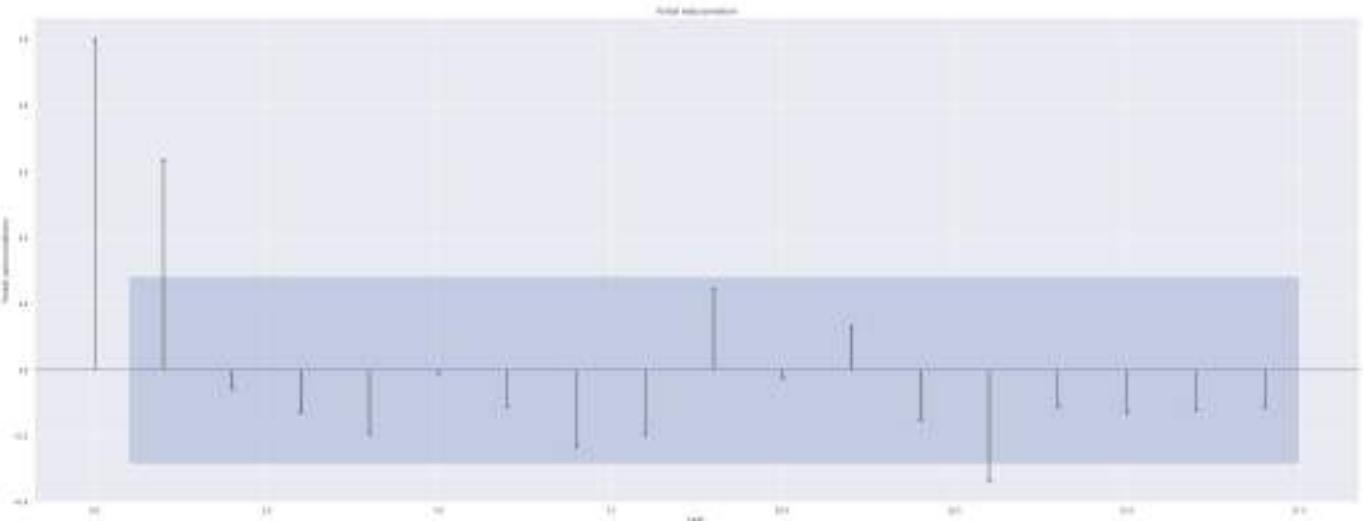
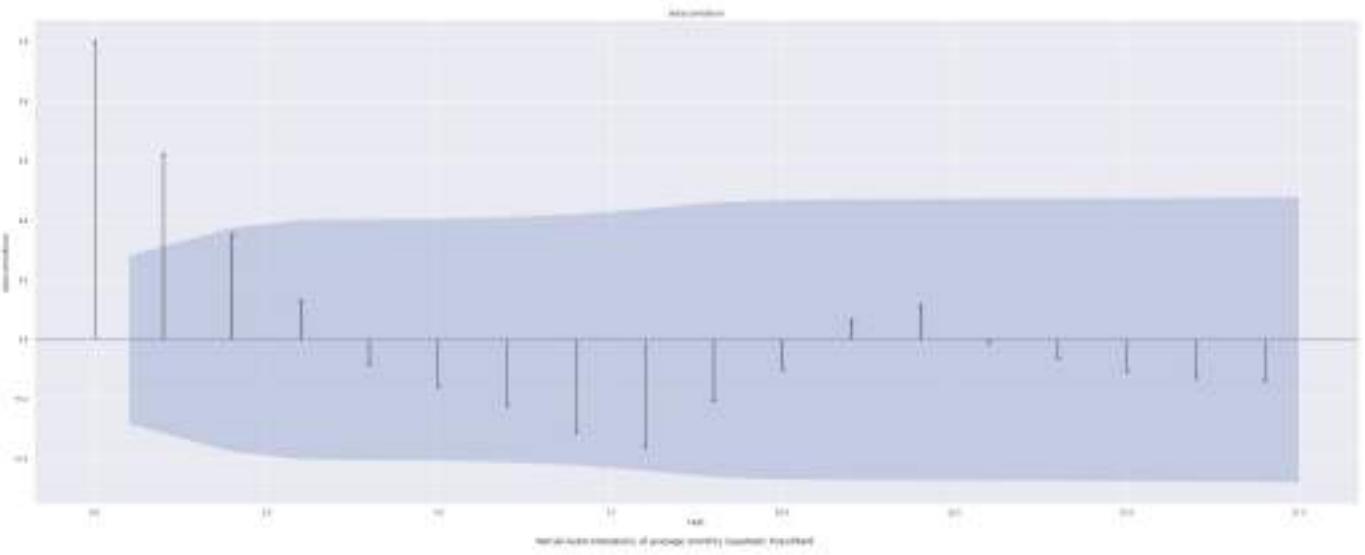
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(FossilHard_ypred)  
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilHard")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(FossilHard_ypred)  
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilHard")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')
```

```
plt.show  
FossilHard_Quad_Autocorrelations = sm.tsa.acf(FossilHard_ypred, fft=False) #Autocorrelations  
print(FossilHard_Quad_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,  
[ 1.          0.62116068  0.35424647  0.12970092 -0.08292558 -0.16069038  
-0.22176301 -0.31138039 -0.3602324 -0.20313361 -0.10060111  0.06371252  
0.11283855 -0.00948166 -0.06093299 -0.10632664 -0.12861721 -0.13714899  
-0.10145144 -0.08099994 -0.03839695  0.07971788  0.22814918  0.2987317  
0.26537802  0.23357415  0.10874268 -0.01982034 -0.05533115 -0.10699164  
-0.12826828 -0.12983625 -0.08783224 -0.11287183 -0.01782404 -0.04452262  
-0.05499859 -0.00228936 -0.06535928 -0.02968622 -0.01249288]
```



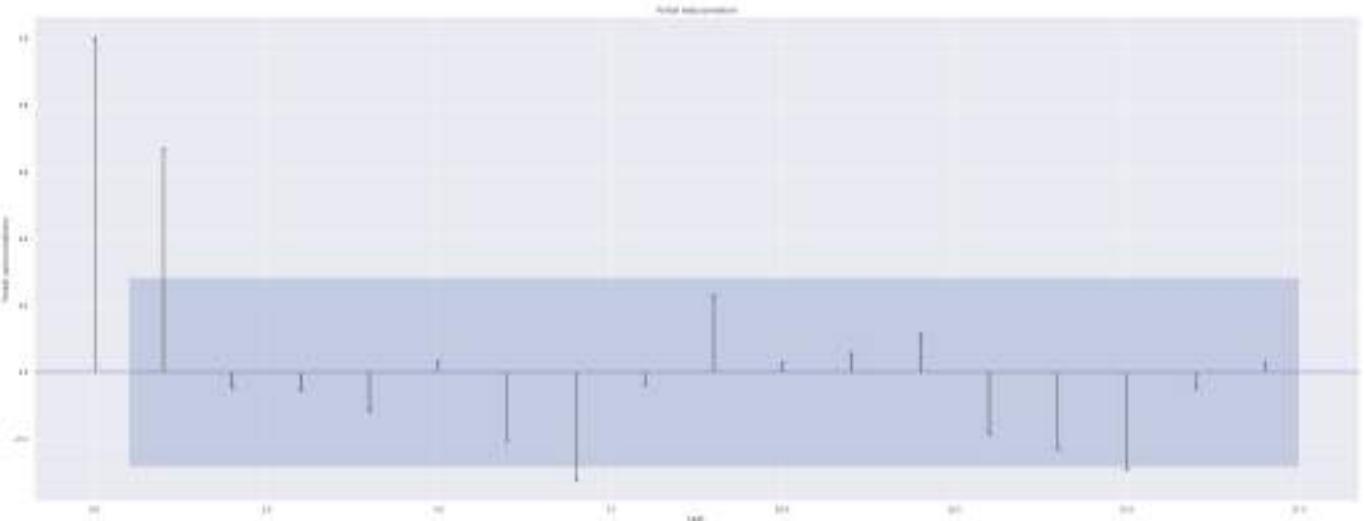
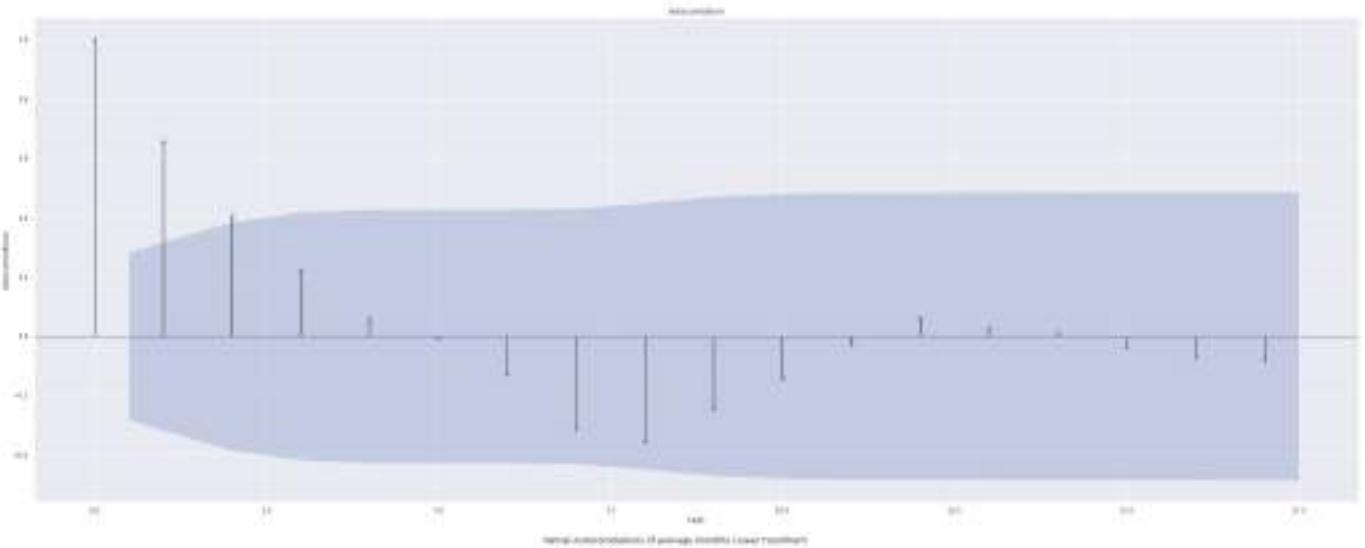
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(predictionsFossilHard)  
plt.suptitle(" Autocorrelations of average monthly Linear FossilHard")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(predictionsFossilHard)  
plt.suptitle("Partial Autocorrelations of average monthly Linear FossilHard")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')
```

```
plt.show()
FossilHard_Pred_Autocorrelations = sm.tsa.acf(predictionsFossilHard, fft=False) #Autocorrelations
print(FossilHard_Pred_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
  FutureWarning,
[ 1.          0.65524025  0.40502744  0.22090565  0.05716118 -0.00635438
 -0.12848552 -0.31077839 -0.35638366 -0.24469887 -0.14290773 -0.02689295
  0.059755   0.02589582  0.0093051  -0.03707528 -0.07045197 -0.08153876
 -0.02854484 -0.02364628 -0.04946738 -0.00155302  0.13145367  0.21548385
  0.23357626  0.16852071  0.07925043  0.00307215 -0.01923759 -0.07099871
 -0.14423417 -0.16461282 -0.1766874  -0.19481761 -0.06830956 -0.09002516
 -0.08190765 -0.02168407 -0.06095269 -0.0439582  -0.0273175 ]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resource analyzed was solar power.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly solar outputs versus the average monthly prices of energy per EUR/MWH.

In []:

```
In [ ]: modelsolar = stats.linregress([1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 19])
```

```
In [ ]: #Dataframes analyzed by resource
dfsolar = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139784, "Solar": [1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.272849, "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12'], "Ratio": df_solar["Solar"] / df_solar["Price"], "pdToListSolar": list(df_solar["Ratio"])}, {"df_solar": df_solar, "pdToListSolar": pdToListSolar})
```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.08432795698924, 55.81655555555556, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Solar': [1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657, 1998.6606397774688, 1973.7405913978494, 1738.9502688172042, 1591.2902777777779, 1082.279569892473, 1171.719444444444, 874.3135935397039, 1025.741935483871, 1258.058908045977, 1382.9057873485867, 1390.4166666666667, 1689.1814516129032, 1942.829166666667, 1915.7752355316286, 1835.8521505376343, 1548.21388888889, 1013.2832214765101, 951.381944444445, 914.9758064516129, 1088.4180107526881, 1157.4613095238096, 1415.0915208613728, 1468.827777777778, 1800.0524193548388, 1919.023611111112, 2025.119623655914, 1754.159946236559, 1731.3666666666666, 1344.3261744966444, 1180.645833333333, 1035.4260752688172, 1051.4193548387098, 1246.110119047619, 1352.3189771197847, 1501.647222222221, 1526.899193548387, 1855.197222222223, 2010.3458950201884, 1706.9301075268818, 1510.652777777778, 999.786577181208, 831.668055555556, 875.5900537634409], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

  Price      Solar      Dates
0   64.949019  1130.392906  2015-01
1   56.383854  1244.525298  2015-02
2   55.522463  1283.479139  2015-03
3   58.354083  1461.519499  2015-04
4   57.294059  1920.272849  2015-05
5   65.974903  1998.660640  2015-06
6   71.072043  1973.740591  2015-07
7   63.998065  1738.950269  2015-08
8   60.254792  1591.290278  2015-09
9   59.406765  1082.279570  2015-10
10  60.726792  1171.719444  2015-11
11  61.901761  874.313594  2015-12
12  45.578723  1025.741935  2016-01
13  36.752083  1258.058908  2016-02
14  36.818008  1382.905787  2016-03
15  32.618667  1390.416667  2016-04
16  34.691371  1689.181452  2016-05
17  46.266319  1942.829167  2016-06
18  47.502016  1915.775236  2016-07
19  47.602339  1835.852151  2016-08
20  50.405597  1548.213889  2016-09
21  60.182430  1013.283221  2016-10
22  62.581056  951.381944  2016-11
23  67.595134  914.975806  2016-12
24  79.492083  1088.418011  2017-01
25  59.837798  1157.461310  2017-02
26  50.959892  1415.091521  2017-03
27  51.717917  1468.827778  2017-04
28  53.772621  1800.052419  2017-05
29  56.258222  1919.023611  2017-06
30  55.252581  2025.119624  2017-07
31  54.084328  1754.159946  2017-08
32  55.816556  1731.366667  2017-09
33  63.925289  1344.326174  2017-10
34  65.430653  1180.645833  2017-11
35  65.151277  1035.426075  2017-12
36  56.511976  1051.419355  2018-01
37  60.877098  1246.110119  2018-02
38  48.279717  1352.318977  2018-03
39  50.400736  1501.647222  2018-04
40  61.633763  1526.899194  2018-05
41  64.348139  1855.197222  2018-06
42  67.783441  2010.345895  2018-07
43  70.363911  1706.930108  2018-08
44  76.914042  1510.652778  2018-09
45  70.362215  999.786577  2018-10
46  67.042608  831.668056  2018-11
47  66.623514  875.590054  2018-12

[17.40431074174863, 22.072370113974813, 23.116394150367523, 25.04571085897575, 33.516090119873006, 30.294256688934063, 27.770984310936953, 27.171919681710804, 26.409356563389956, 18.218119907024636, 19.294934118635627, 14.124212024158545, 22.50484141080495, 34.23095492670875, 37.56058134697677, 42.62641023544801, 48.69168915761797, 41.992300014259236, 40.33039840514782, 38.56642762311195, 30.71511844336071, 16.836861346184342, 15.202395293570383, 13.536119344341051, 13.692156062744463, 19.34331401855214, 27.76873058801158, 28.40075301649708, 33.47525909950876, 34.110989208491006, 36.652036882430444, 32.43379390110128, 31.01887333308118, 21.029645765169665, 18.044231307658848, 15.892644393579458, 18.605248530678267, 20.469275895203573, 28.010084793555276, 29.79415258760826, 24.77374588708186, 28.830627493758996, 29.658363008835465, 24.258601834739714, 19.64079308593233, 14.209140239813733, 12.405067258490595, 13.142357745100368]

```

In []: #Linear OLS regression
solar1 = solar

```

solar1 = sm.add_constant(solar1)
modelsolarreg = sm.OLS(Price_Actual, solar1).fit()
predictionssolar = modelsolarreg.predict(solar1)

modelsolarreg.summary()
#OLS Linear Summary Table

```

Out[]: OLS Regression Results

Dep. Variable:	y	R-squared:	0.032		
Model:	OLS	Adj. R-squared:	0.011		
Method:	Least Squares	F-statistic:	1.524		
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.223		
Time:	05:26:25	Log-Likelihood:	-178.86		
No. Observations:	48	AIC:	361.7		
Df Residuals:	46	BIC:	365.5		
Df Model:	1				
Covariance Type:	nonrobust				
coef	std err	t	P> t	[0.025	0.975]
const	65.1302	6.073	10.725	0.000	52.906 77.354
x1	-0.0051	0.004	-1.234	0.223	-0.013 0.003
Omnibus:	3.295	Durbin-Watson:	0.431		
Prob(Omnibus):	0.193	Jarque-Bera (JB):	2.342		
Skew:	-0.512	Prob(JB):	0.310		
Kurtosis:	3.348	Cond. No.	6.05e+03		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 6.05e+03. This might indicate that there are strong multicollinearity or other numerical problems.

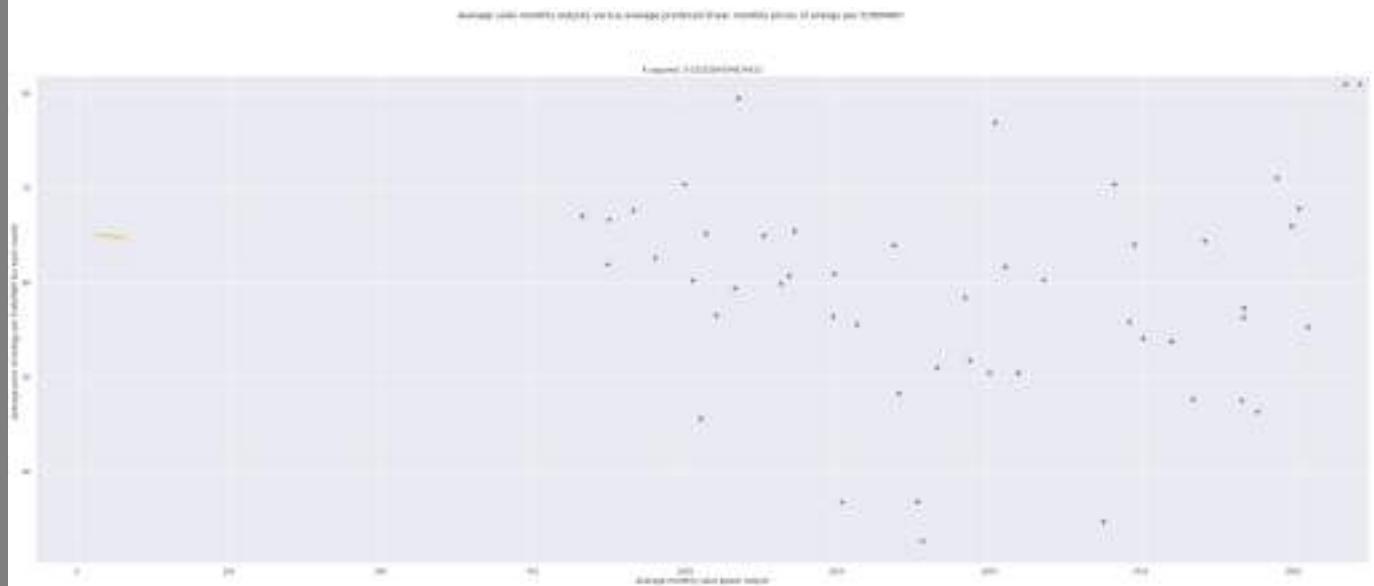
In []:

```

In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(solar,Price_Actual)
print("slope: %f    intercept: %f" % (slope, intercept))
#OLS Linear Scatterplot
plt.plot(solar,Price_Actual, "o")
plt.title(f'R squared: {modelsolar.rvalue**2}')
f = lambda x: -0.005078 *x + 65.130153
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.legend("#")
plt.suptitle("Average solar monthly outputs versus average predicted linear monthly prices of energy per EUR/MWh")
plt.ylabel('Average price of energy per EUR/MWh for each month ')
plt.xlabel('Average monthly solar power output')
plt.show()

```

slope: -0.005078 intercept: 65.130153



There is a very weak and negative correlation between the outputs and their respective the average monthly prices of energy per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: print(predictionssolar)
#Linear OLS Predicted Values
```

```
[59.39046844 58.81094967 58.61315759 57.70913964 55.37977306 54.98175112
 55.10828519 56.30045682 57.05021537 59.63476877 59.18062876 60.69073707
 59.92184434 58.74223139 58.10830872 58.07017147 56.55316303 55.26524108
 55.40261015 55.80842771 57.2689401 59.98510472 60.29941472 60.48427057
 59.60360022 59.25302588 57.9448824 57.67203111 55.99020447 55.3861162
 54.84740295 56.22322814 56.33896333 58.30420061 59.13530411 59.87267214
 59.79146464 58.80290257 58.26361634 57.50538706 57.37716762 55.71020115
 54.92241805 56.46304243 57.45966044 60.05363529 60.90727411 60.68425572]
```

```
In [ ]: #Linear OLS regression residuals
influencesolarreg = modelsolarreg.get_influence()
```

```
standardized_residualsSolar = influencesolarreg.resid_studentized_internal
```

```
print(standardized_residualsSolar)
```

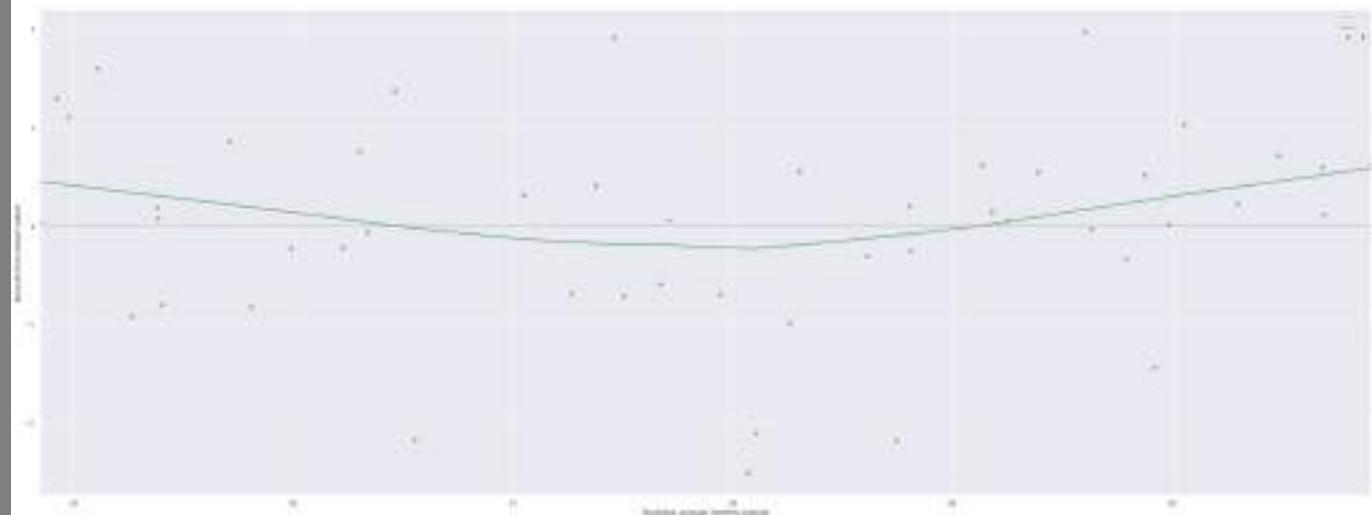
```
[ 0.55143891 -0.23967078 -0.3048712   0.06350795  0.19228693  1.11212416
 1.61112714  0.76386454  0.31619356 -0.02267844  0.1530924   0.1224027
 -1.43177046 -2.17060795 -2.09672766 -2.50639499 -2.16437445 -0.90567141
 -0.79330579 -0.81903826 -0.6765401   0.0197146   0.22903607  0.71602347
 1.97750884  0.05793745 -0.68778185 -0.58632828 -0.22082007  0.08759253
 0.04109924 -0.21242113 -0.05182077  0.55381952  0.62309438  0.52658024
 -0.32681186  0.20481265 -0.98355907 -0.69982801  0.41942996  0.86330568
 1.30259785  1.37730187  1.91652157  1.03092342  0.62278398  0.60022934]
```

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionssolar, y = standardized_residualsSolar, lowess = True, color ="g")
```

```
plt.suptitle("Solar power residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
```

```
plt.legend(..#")
```

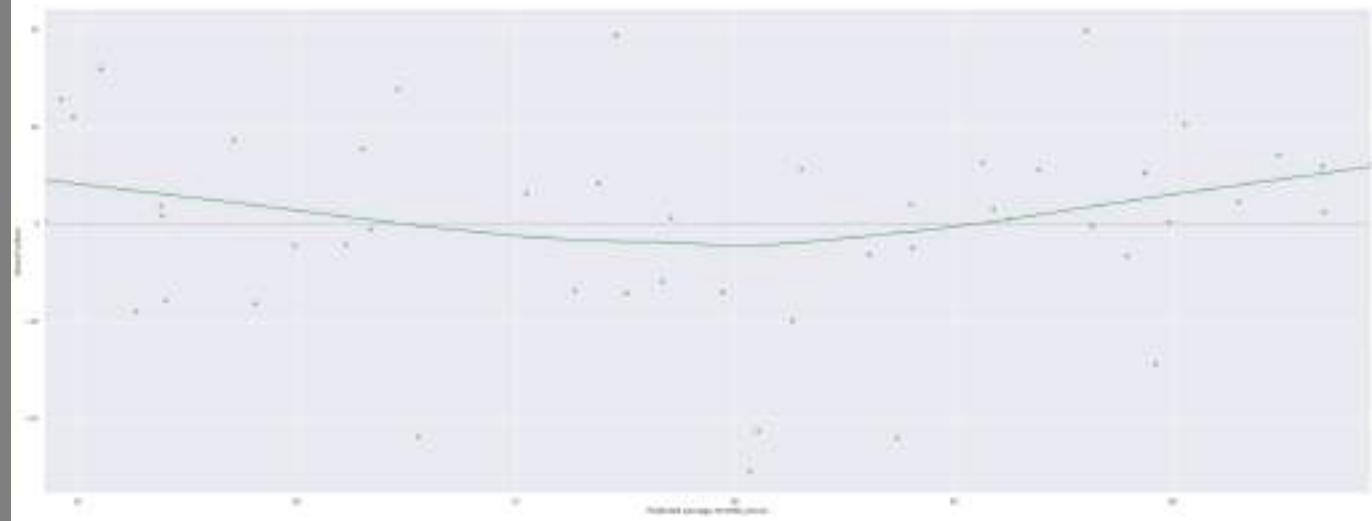
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60bbe5790>
```



As one can observe this residual plot, one may notice dull hump within the lowess line, which form a roughly linear pattern. However, the residuals are quite spread out, which indicates homoscedasticity and constant variance. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly linear solar energy prices per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#")
sns.residplot(x = predictionssolar, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60ad6d690>
```



As one can observe this residual plot, one may notice the negative slope in the fitted model. In addition, the residuals are spread out in a distinct pattern, indicating bias and homoscedasticity. It would be reasonable to assume that this model does not fit the data. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: print(solar)
```

```
[1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657, 1998.66063977  
74688, 1973.7405913978494, 1738.9502688172042, 1591.2902777777779, 1082.279569892473, 1171.719444444444, 874.31  
35935397039, 1025.741935483871, 1258.058908045977, 1382.9057873485867, 1390.41666666666667, 1689.1814516129032, 1  
942.82916666666667, 1915.7752355316286, 1835.8521505376343, 1548.213888888889, 1013.2832214765101, 951.3819444444  
445, 914.9758064516129, 1088.4180107526881, 1157.4613095238096, 1415.0915208613728, 1468.8277777777778, 1800.052  
4193548388, 1919.0236111111112, 2025.119623655914, 1754.159946236559, 1731.3666666666666, 1344.3261744966444, 11  
80.6458333333333, 1035.4260752688172, 1051.4193548387098, 1246.110119047619, 1352.3189771197847, 1501.647222222  
221, 1526.899193548387, 1855.197222222223, 2010.3458950201884, 1706.9301075268818, 1510.6527777777778, 999.7865  
77181208, 831.668055555556, 875.5900537634409]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly solar output versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=3)

modelSolarquad = np.poly1d(np.polyfit(solar, Price_Actual, 2))
print(modelSolarquad)

solar1 = solar
solar1 = sm.add_constant(solar1)
solar2 = polynomial_features.fit_transform(solar1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(solar1)

Solar_Q = poly.fit(X_poly, solar)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
Solar_Quad = sm.OLS(Price_Actual, solar2).fit()

# OLS Predicted Quadratic values
Solar_ypred = Solar_Quad.predict(solar2)

#OLS Quadratic Summary Table
Solar_Quad.summary()
```

2
3.069e-05 x² - 0.09402 x + 125.6

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.146			
Model:	OLS	Adj. R-squared:	0.088			
Method:	Least Squares	F-statistic:	2.514			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.0707			
Time:	05:26:27	Log-Likelihood:	-175.85			
No. Observations:	48	AIC:	359.7			
Df Residuals:	44	BIC:	367.2			
Df Model:	3					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	25.9545	28.237	0.919	0.363	-30.953	82.862
x1	25.9545	28.237	0.919	0.363	-30.953	82.862
x2	-0.0149	0.084	-0.178	0.859	-0.184	0.154
x3	25.9545	28.237	0.919	0.363	-30.953	82.862
x4	-0.0149	0.084	-0.178	0.859	-0.184	0.154
x5	-2.454e-06	9.01e-05	-0.027	0.978	-0.000	0.000
x6	25.9545	28.237	0.919	0.363	-30.953	82.862
x7	-0.0149	0.084	-0.178	0.859	-0.184	0.154
x8	-2.43e-06	9.02e-05	-0.027	0.979	-0.000	0.000
x9	8.264e-09	4.18e-08	0.198	0.844	-7.59e-08	9.25e-08
Omnibus:	1.251	Durbin-Watson:	0.517			
Prob(Omnibus):	0.535	Jarque-Bera (JB):	0.542			
Skew:	-0.185	Prob(JB):	0.763			
Kurtosis:	3.367	Cond. No.	2.89e+35			

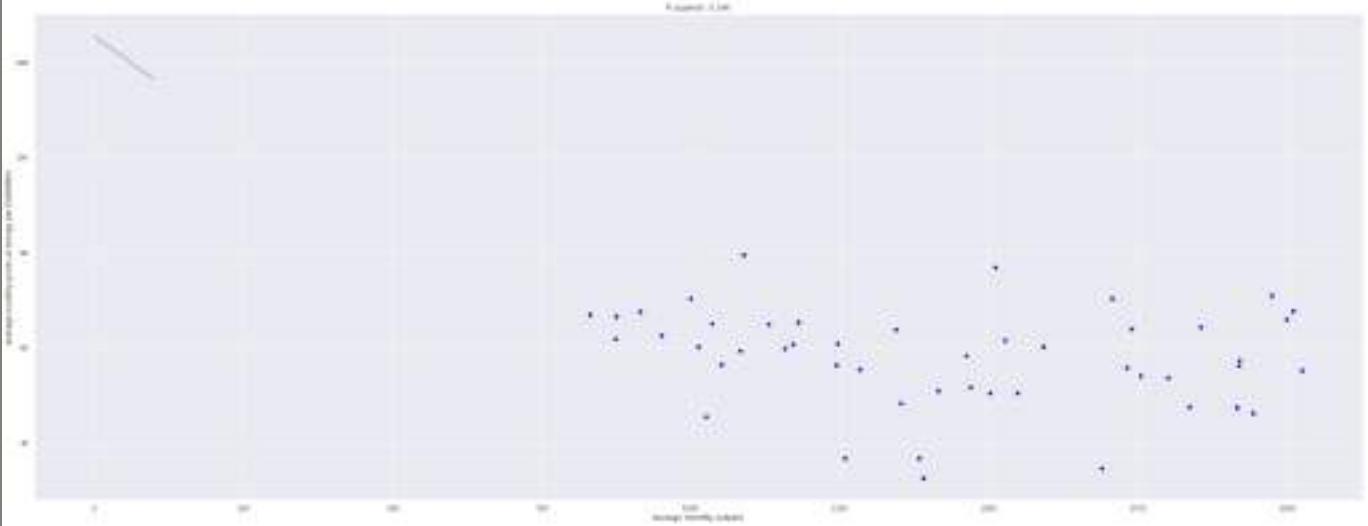
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 1.03e-50. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
##OLS Quadratic Scatterplot
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelSolarquad(polyline))
plt.scatter(solar, Price_Actual, color = 'blue')
plt.title("R squared : 0.146")
plt.suptitle('Quadratic for predicted average monthly prices of energy per EUR/MWH versus average monthly solar outputs')
plt.xlabel('Average monthly outputs')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()
```



The blue dots represent the observations and the blue line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

```
In [ ]: influenceSolarQuad = Solar_Quad.get_influence() #Quadratic OLS residuals

standardized_residualsSolarQuad = influenceSolarQuad.resid_studentized_internal

print(standardized_residualsSolarQuad)
```

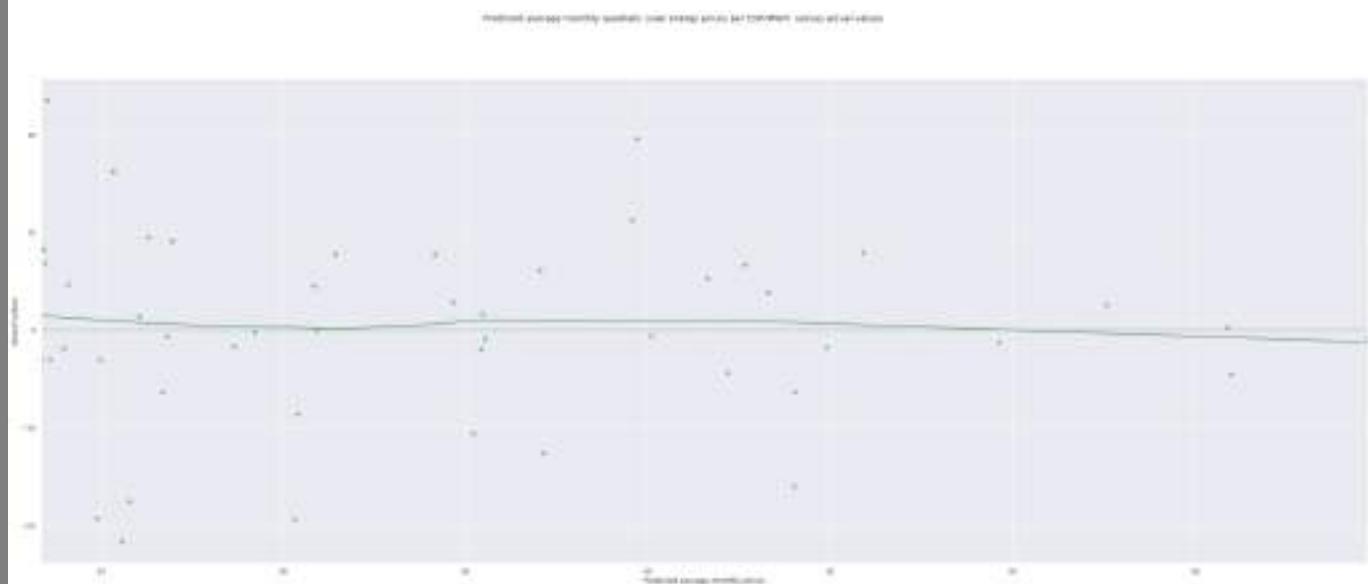
```
[ 6.41888474e-01  1.30803621e-03 -1.71286352e-02  4.90743624e-01
-9.64415637e-02  5.89295491e-01  1.21651486e+00  1.00029910e+00
 7.21643003e-01 -6.60412606e-02  3.00309795e-01 -5.06042425e-01
-1.67556556e+00 -2.02443421e+00 -1.82015736e+00 -2.24844037e+00
-2.03513384e+00 -1.33693897e+00 -1.11429531e+00 -8.97838543e-01
-3.08174239e-01 -1.86014561e-01 -1.34184737e-01  2.77177038e-01
 2.05247881e+00  1.73427530e-01 -3.15677293e-01 -1.95899931e-01
-1.78301742e-01 -2.01903516e-01 -7.32900817e-01 -6.80444128e-02
 1.47089786e-01  9.51957503e-01  8.13646225e-01  4.00513705e-01
-4.55976105e-01  4.74181621e-01 -6.65979328e-01 -3.17760963e-01
 8.63463522e-01  8.14599002e-01  7.55882780e-01  1.71370187e+00
 2.45392940e+00  8.38590883e-01 -1.04898136e-01  3.01967385e-02]
```

```
In [ ]: print(Solar_ypred) # OLS quadratic predicted values
```

```
[58.8206058 56.37134467 55.68656518 53.64179117 58.20906644 60.65601725
59.81973627 54.52075647 53.38772545 60.03800825 57.86025427 66.3986232
61.59801586 56.1243502 54.3161987 54.23568319 53.96057245 58.85832918
58.08482346 56.16031686 53.34830407 61.95975601 63.84833701 65.02676263
59.87691787 58.18243754 53.99478258 53.59841119 55.46811457 58.17438405
61.60468186 54.72945106 54.42331939 54.78158928 57.66354151 61.32123665
60.87269557 56.34192682 54.67812308 53.44540705 53.37502179 56.57756046
61.06721676 54.13900492 53.41553198 62.35871886 67.8969952 66.3546964 ]
```

```
In [ ]: plt.suptitle("Predicted average monthly quadratic solar energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend("#")
sns.residplot(x = Solar_ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609123e90>
```



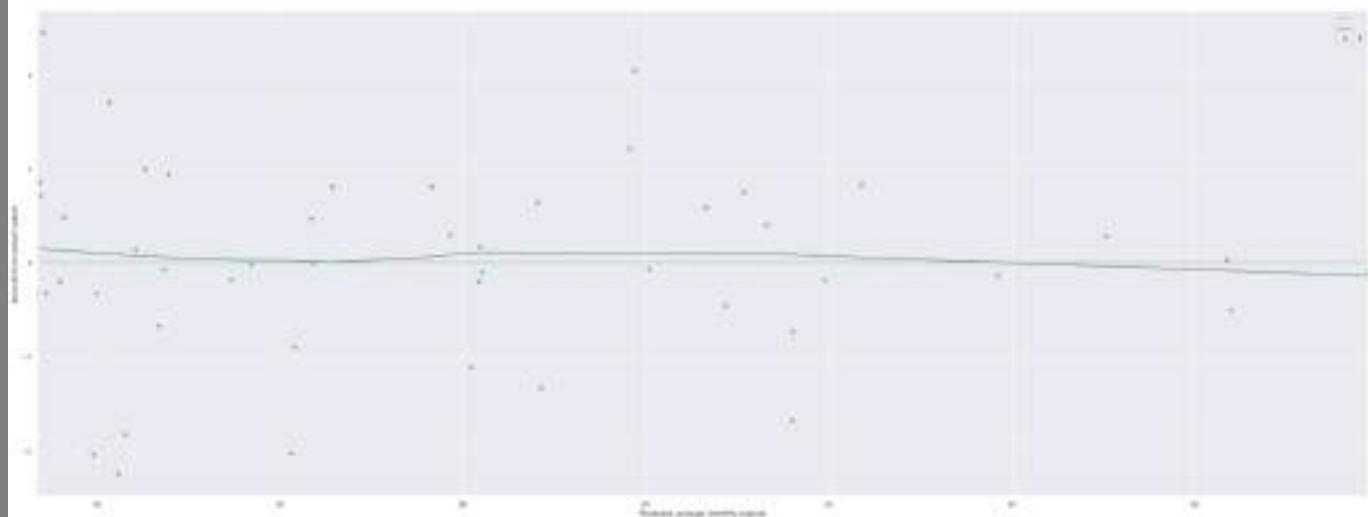
With the exception of a few outliers, the predictions seem to be nearly the same as the observations. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = Solar_ypred, y = standardized_residualsSolarQuad, lowess = True, color="g")

plt.suptitle("Solar power residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff6099e1f90>
```



With the exception of a few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the average monthly solar outputs versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
Logsolarvalues = ((np.log(solar)))
Log = np.polyfit(np.log(Price_Actual), solar1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(solar1), Price_Actual)
Solar_Log = sm.OLS(Price_Actual, solar1).fit()

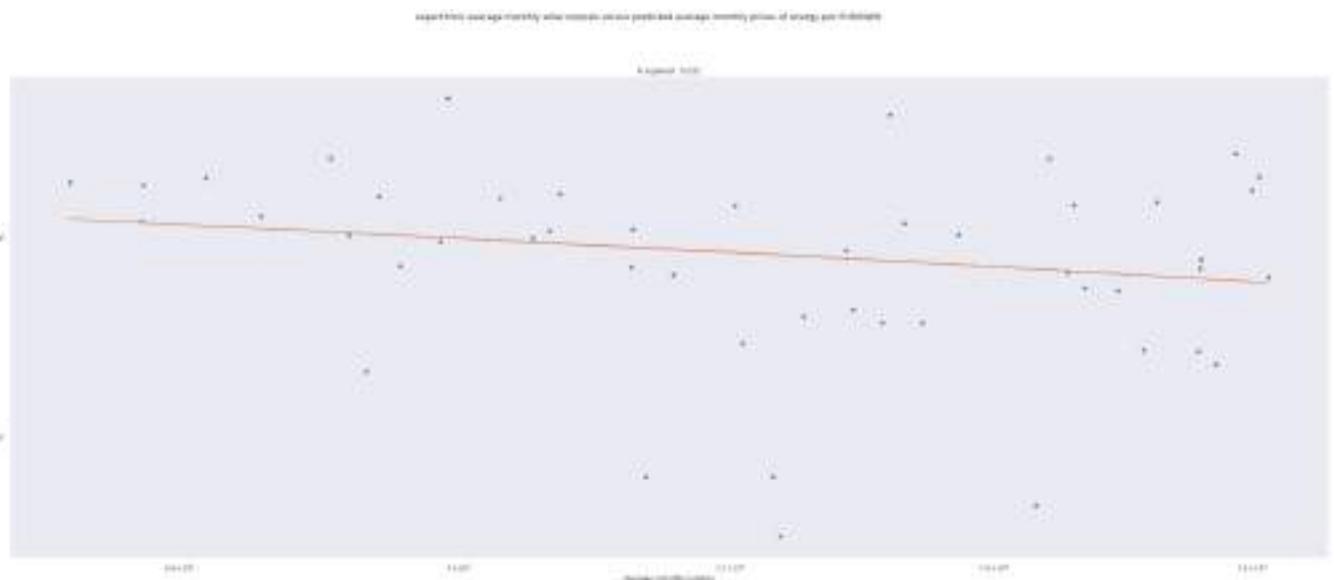
Solar_Logpred = Solar_Log.predict(solar1)
#OLS Logarithmic summary table
Solar_Log.summary()
#Log
Log = np.polyfit(np.log(solar), Price_Actual, 1)
print(Log)

y = -8.46222795 * Logsolarvalues + 119.07321213

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly solar outputs versus predicted average monthly prices of energy per EUR/MWH")
plt.title("R squared : 0.032")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(Logsolarvalues, Price_Actual, "o")
plt.plot(Logsolarvalues, y)
```

```
[ -8.46222795 119.07321213]
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x7ff60a6680d0>]
```



The blue dots represent the observations and the orange line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: Solar_Log.summary() #OLS Logarithmic summary table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.032			
Model:	OLS	Adj. R-squared:	0.011			
Method:	Least Squares	F-statistic:	1.524			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.223			
Time:	05:26:30	Log-Likelihood:	-178.86			
No. Observations:	48	AIC:	361.7			
Df Residuals:	46	BIC:	365.5			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	65.1302	6.073	10.725	0.000	52.906	77.354
x1	-0.0051	0.004	-1.234	0.223	-0.013	0.003
Omnibus:	3.295	Durbin-Watson:	0.431			
Prob(Omnibus):	0.193	Jarque-Bera (JB):	2.342			
Skew:	-0.512	Prob(JB):	0.310			
Kurtosis:	3.348	Cond. No.	6.05e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 6.05e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceSolarLog = Solar_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsSolarLog = influenceSolarLog.resid_studentized_internal

print(standardized_residualsSolarLog)
```

[0.55143891 -0.23967078 -0.3048712 0.06350795 0.19228693 1.11212416
 1.61112714 0.76386454 0.31619356 -0.02267844 0.1530924 0.1224027
 -1.43177046 -2.17060795 -2.09672766 -2.50639499 -2.16437445 -0.90567141
 -0.79330579 -0.81903826 -0.6765401 0.0197146 0.22903607 0.71602347
 1.97750884 0.05793745 -0.68778185 -0.58632828 -0.22082007 0.08759253
 0.04109924 -0.21242113 -0.05182077 0.55381952 0.62309438 0.52658024
 -0.32681186 0.20481265 -0.98355907 -0.69982801 0.41942996 0.86330568
 1.30259785 1.37730187 1.91652157 1.03092342 0.62278398 0.60022934]

In []:

```
print(Solar_Logpred) # OLS logarithmic predicted values
```

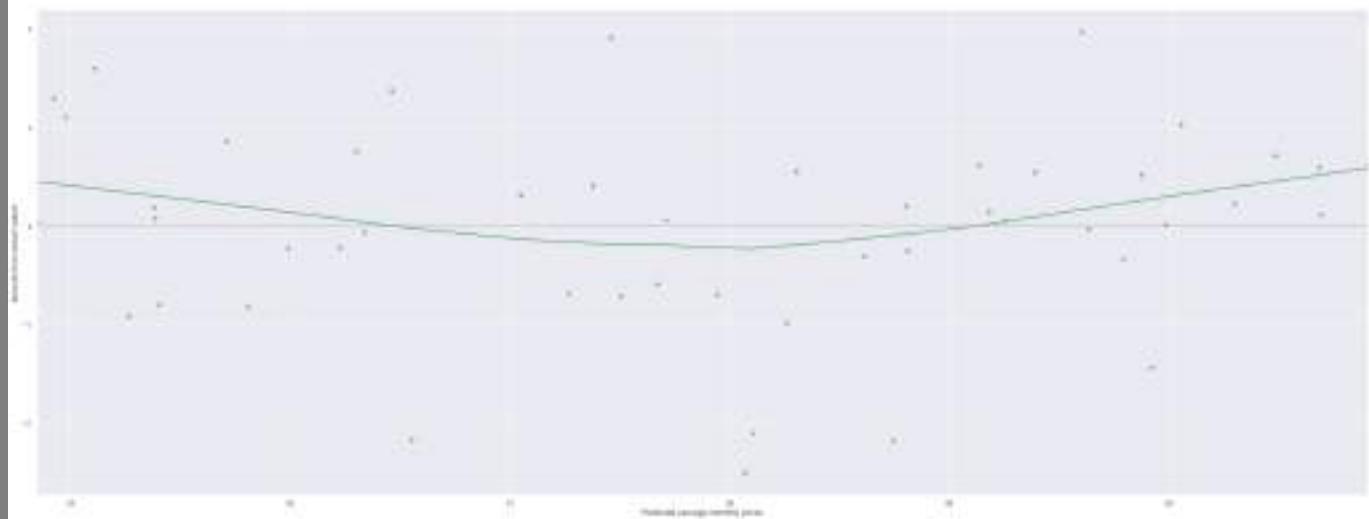
[59.39046844 58.81094967 58.61315759 57.70913964 55.37977306 54.98175112
 55.10828519 56.30045682 57.05021537 59.63476877 59.18062876 60.69073707
 59.92184434 58.74223139 58.10830872 58.07017147 56.55316303 55.26524108
 55.40261015 55.80842771 57.2689401 59.98510472 60.29941472 60.48427057
 59.60360022 59.25302588 57.9448824 57.67203111 55.99020447 55.3861162
 54.84740295 56.22322814 56.33896333 58.30420061 59.13530411 59.87267214
 59.79146464 58.80290257 58.26361634 57.50538706 57.37716762 55.71020115
 54.92241805 56.46304243 57.45966044 60.05363529 60.90727411 60.68425572]

In []:

```
SolarLogRatioPredict = Solar_Logpred/Logpred

In [ ]:
# OLS Logarithmic average monthly predictions versus residuals
plt.suptitle("Residuals from logarithmic solar model versus predicted average monthly logarithmic solar output")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = Solar_Logpred, y = standardized_residualsSolarLog, lowess = True, color="g")
```

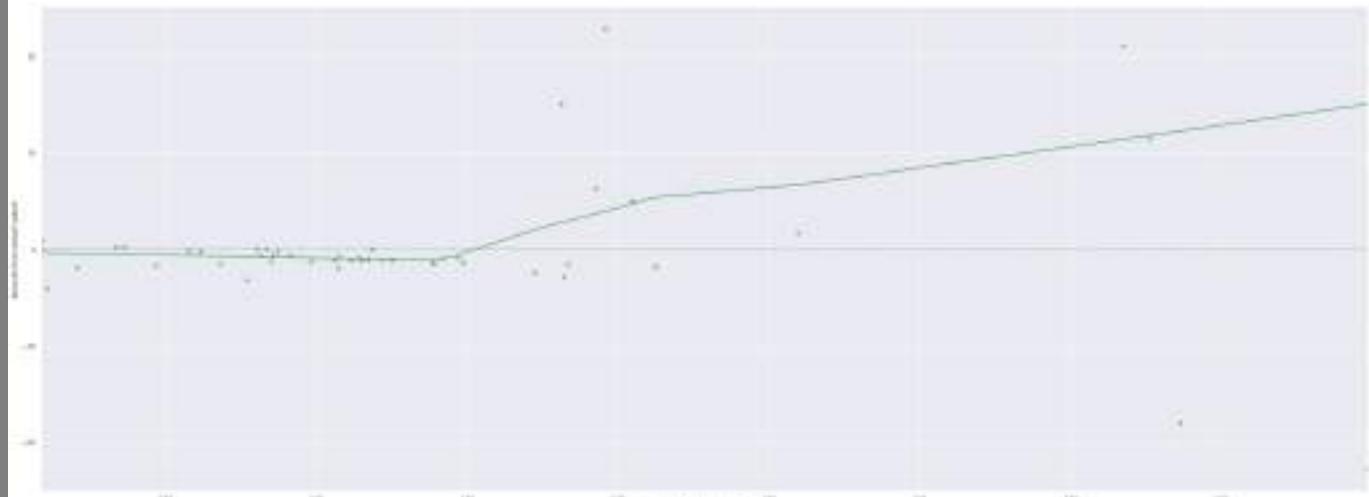
Out[]:



As one can observe, there is a subtle hump along the lowess line in the residual plot. Furthermore, there residuals are quite spread out in no particular pattern which indicates constant variance and a lack of bias. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
plt.suptitle("Predicted average monthly logarithmic output to price of energy per EUR/MWH ratio versus respective monthly average monthly logarithmic output to price of energy per EUR/MWH ratio")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Amount from actual values")  
plt.legend(..#)  
sns.residplot(x = SolarLogRatioPredict, y = standardized_residualsSolarLog/standardized_residualsPriceLog, lowess=True)
```

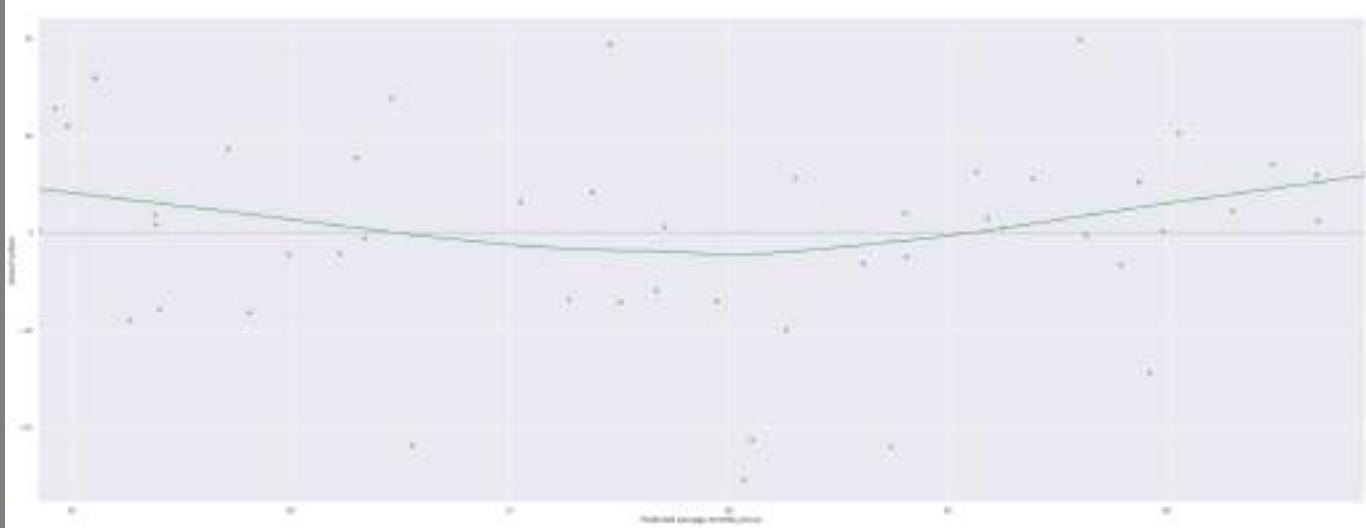
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a398310>
```



With the exception of few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS predicted logarithmic average monthly values versus actual values  
plt.suptitle("Predicted average monthly logarithmic solar energy prices per EUR/MWH versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend(..#)  
sns.residplot(x = Solar_Logpred, y = Price_Actual, lowess = True, color="g")
```

```
influenceSolarLog = Solar_Log.get_influence()  
#Logarithmic OLS regression residuals
```



As one can observe, there is a subtle hump along the lowess line in the residual plot. Furthermore, the observations are quite spread out in no particular pattern which indicates constant variance and a lack of bias. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: standardized_residualsSolarLog = influenceSolarLog.resid_studentized_internal  
  
print(standardized_residualsSolarLog)
```

```
[ 0.55143891 -0.23967078 -0.3048712   0.06350795  0.19228693  1.11212416  
 1.61112714  0.76386454  0.31619356 -0.02267844  0.1530924   0.1224027  
-1.43177046 -2.17060795 -2.09672766 -2.50639499 -2.16437445 -0.90567141  
-0.79330579 -0.81903826 -0.6765401   0.0197146   0.22903607  0.71602347  
 1.97750884  0.05793745 -0.68778185 -0.58632828 -0.22082007  0.08759253  
 0.04109924 -0.21242113 -0.05182077  0.55381952  0.62309438  0.52658024  
-0.32681186  0.20481265 -0.98355907 -0.69982801  0.41942996  0.86330568  
 1.30259785  1.37730187  1.91652157  1.03092342  0.62278398  0.60022934]
```

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

```
In [ ]: #Dataframes analyzed by resource  
dfsolar = ({'Price':[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139784,  
           "Solar" : [1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.272849],  
           "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12']})  
print(dfsolar)  
df_solar = pd.DataFrame.from_dict(dfsolar, orient = "columns")  
print(df_solar)  
df_solar["Ratio"] = df_solar["Solar"]/df_solar["Price"]  
pdToListSolar = list(df_solar["Ratio"])  
  
print(pdToListSolar)  
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis, indicating it is non-stationary ")  
  
adfuller_test(df_solar["Ratio"])  
  
test_result=adfuller(df_solar["Ratio"])  
  
df_solar['First Difference Solar Ratio'] = df_solar["Ratio"]- df_solar["Ratio"].shift(1) # Seasonality values  
df_solar['Seasonal Difference Solar Ratio']=df_solar["Ratio"]- df_solar["Ratio"].shift(12)
```

```

df_solar.head()

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_solar["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Solar Ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_solar["Ratio"])
plt.suptitle("Partialatocorrelations of Solar Ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

Solar_Ratio_Autocorrelations = sm.tsa.acf(df_solar["Ratio"],fft=False) #Autocorrelations
print(Solar_Ratio_Autocorrelations)

```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555555, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.4306527777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Solar': [1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657, 1998.6606397774688, 1973.7405913978494, 1738.9502688172042, 1591.2902777777779, 1082.279569892473, 1171.719444444444, 874.3135935397039, 1025.741935483871, 1258.058908045977, 1382.9057873485867, 1390.41666666666667, 1689.1814516129032, 1942.8291666666667, 1915.7752355316286, 1835.8521505376343, 1548.21388888889, 1013.2832214765101, 951.3819444444445, 914.9758064516129, 1088.4180107526881, 1157.4613095238096, 1415.0915208613728, 1468.827777777778, 1800.0524193548388, 1919.023611111112, 2025.119623655914, 1754.159946236559, 1731.366666666666, 1344.3261744966444, 1180.645833333333, 1035.4260752688172, 1051.4193548387098, 1246.110119047619, 1352.3189771197847, 1501.647222222221, 1526.899193548387, 1855.197222222223, 2010.3458950201884, 1706.9301075268818, 1510.652777777778, 999.786577181208, 831.6680555555556, 875.5900537634409], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

      Price      Solar      Dates
0   64.949019  1130.392906  2015-01
1   56.383854  1244.525298  2015-02
2   55.522463  1283.479139  2015-03
3   58.354083  1461.519499  2015-04
4   57.294059  1920.272849  2015-05
5   65.974903  1998.660640  2015-06
6   71.072043  1973.740591  2015-07
7   63.998065  1738.950269  2015-08
8   60.254792  1591.290278  2015-09
9   59.406765  1082.279570  2015-10
10  60.726792  1171.719444  2015-11
11  61.901761  874.313594  2015-12
12  45.578723  1025.741935  2016-01
13  36.752083  1258.058908  2016-02
14  36.818008  1382.905787  2016-03
15  32.618667  1390.416667  2016-04
16  34.691371  1689.181452  2016-05
17  46.266319  1942.829167  2016-06
18  47.502016  1915.775236  2016-07
19  47.602339  1835.852151  2016-08
20  50.405597  1548.213889  2016-09
21  60.182430  1013.283221  2016-10
22  62.581056  951.381944  2016-11
23  67.595134  914.975806  2016-12
24  79.492083  1088.418011  2017-01
25  59.837798  1157.461310  2017-02
26  50.959892  1415.091521  2017-03
27  51.717917  1468.827778  2017-04
28  53.772621  1800.052419  2017-05
29  56.258222  1919.023611  2017-06
30  55.252581  2025.119624  2017-07
31  54.084328  1754.159946  2017-08
32  55.816556  1731.366667  2017-09
33  63.925289  1344.326174  2017-10
34  65.430653  1180.645833  2017-11
35  65.151277  1035.426075  2017-12
36  56.511976  1051.419355  2018-01
37  60.877098  1246.110119  2018-02
38  48.279717  1352.318977  2018-03
39  50.400736  1501.647222  2018-04
40  61.633763  1526.899194  2018-05
41  64.348139  1855.197222  2018-06
42  67.783441  2010.345895  2018-07

```

```

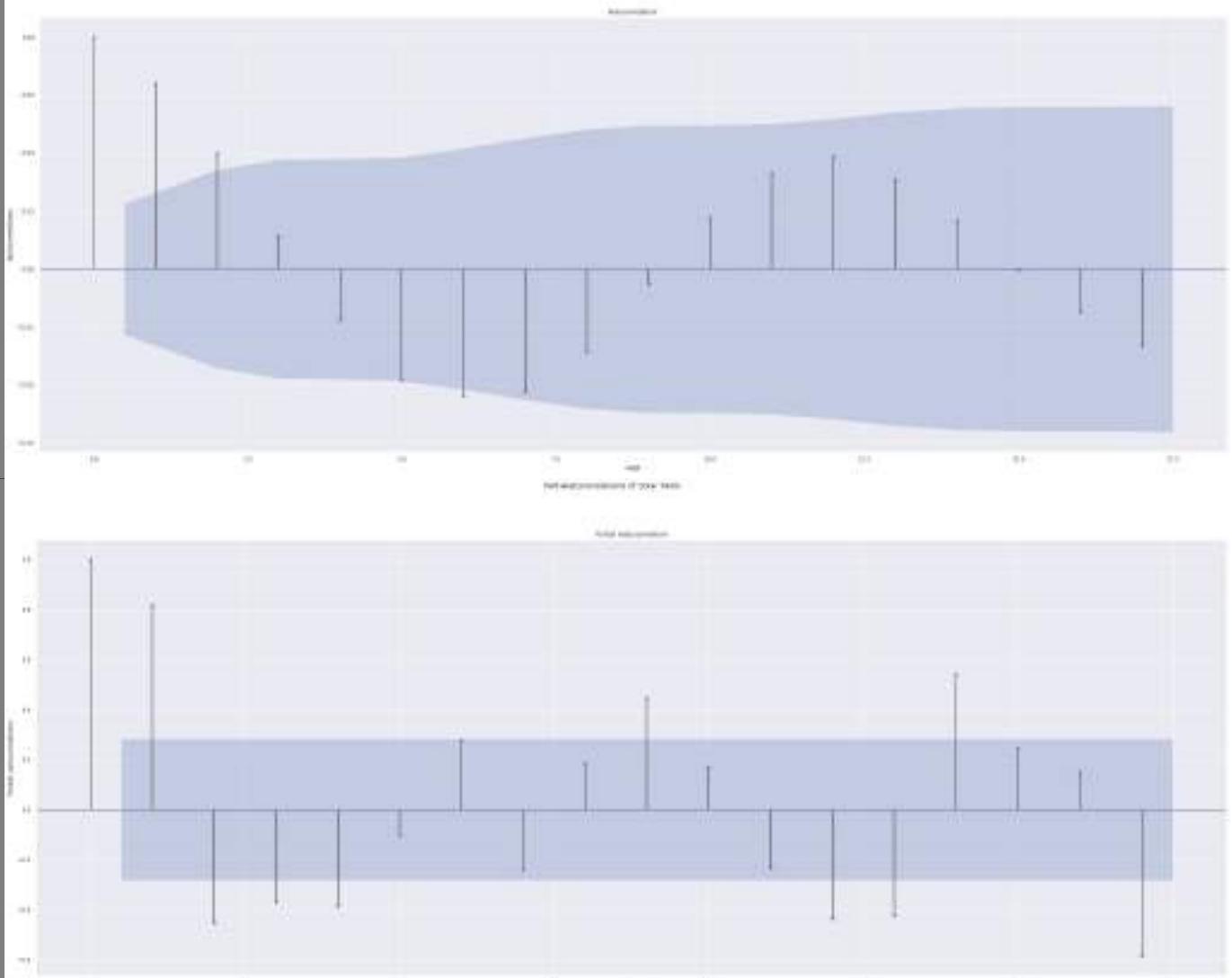
43 70.363911 1706.930108 2018-08
44 76.914042 1510.652778 2018-09
45 70.362215 999.786577 2018-10
46 67.042608 831.668056 2018-11
47 66.623514 875.590054 2018-12
[17.40431074174863, 22.072370113974813, 23.116394150367523, 25.04571085897575, 33.516090119873006, 30.2942566889
34063, 27.770984310936953, 27.171919681710804, 26.409356563389956, 18.218119907024636, 19.294934118635627, 14.12
4212024158545, 22.50484141080495, 34.23095492670875, 37.56058134697677, 42.62641023544801, 48.69168915761797, 41
.992300014259236, 40.33039840514782, 38.56642762311195, 30.71511844336071, 16.836861346184342, 15.20239529357038
3, 13.536119344341051, 13.692156062744463, 19.34331401855214, 27.76873058801158, 28.40075301649708, 33.475259099
50876, 34.110989208491006, 36.652036882430444, 32.43379390110128, 31.01887333308118, 21.029645765169665, 18.0442
31307658848, 15.892644393579458, 18.605248530678267, 20.469275895203573, 28.010084793555276, 29.79415258760826,
24.77374588708186, 28.830627493758996, 29.658363008835465, 24.258601834739714, 19.64079308593233, 14.20914023981
3733, 12.405067258490595, 13.142357745100368]
ADF Test Statistic : -4.799625333645088
p-value : 5.4475760780583485e-05
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
[ 1.          0.80123263  0.49857742  0.14210673 -0.22089172 -0.48037084
-0.54817347 -0.52643109 -0.35832974 -0.0629625  0.22211735  0.41193346
0.48604285  0.38428837  0.20876778 -0.00442883 -0.18452782 -0.33015212
-0.35933673 -0.31520721 -0.19439195 -0.02642322  0.15813641  0.248672
0.25264935  0.17531177  0.04715843 -0.11412346 -0.21289272 -0.27354676
-0.29469558 -0.24407921 -0.14298848 -0.02261562  0.07861004  0.12886182
0.10796204  0.06282674  0.01414269 -0.03329311 -0.04005472]

```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```



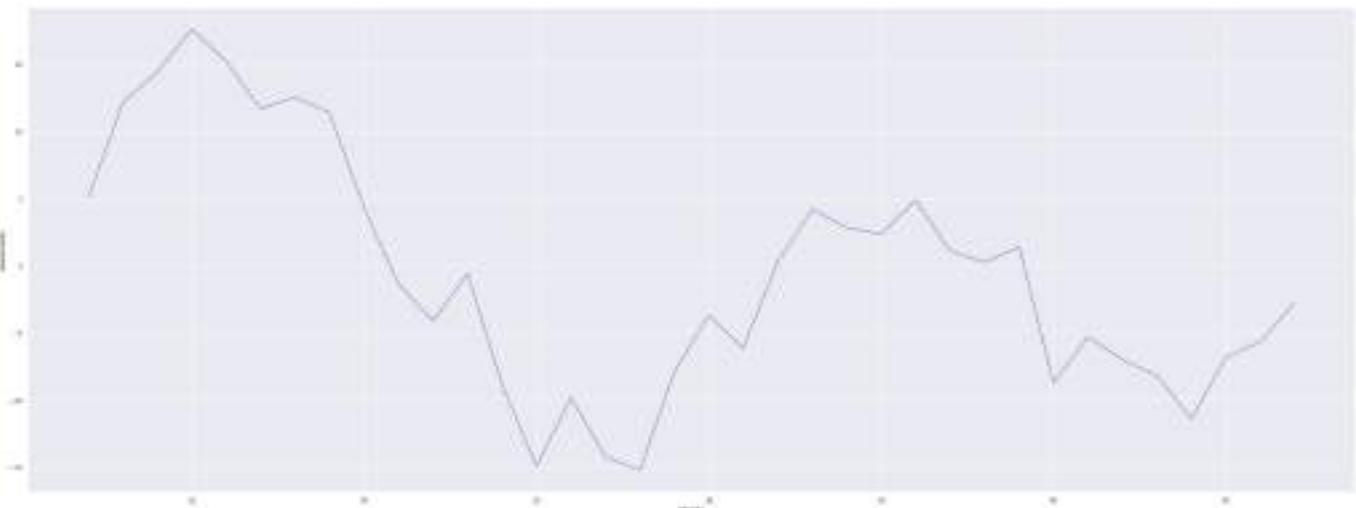
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the auto and partial autocorrelation plots, indicating that the lags directly beforehand influenced the relationship between average monthly solar outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly Solar outputs to prices of energy per EUR/MWH ratio")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_solar['Seasonal Difference Solar Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a695b50>
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective the average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_solar["Solar"])

test_result=adfuller(df_solar["Solar"])
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_solar["Solar"])
plt.suptitle(" Autocorrelations of average monthly Solar")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_solar["Solar"])
plt.suptitle("Partialatocorrelations of Solar")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show
```

```

df_solar['First Difference'] = df_solar["Solar"] - df_solar["Solar"].shift(1) # Seasonality values
df_solar['Seasonal Difference']=df_solar["Solar"]- df_solar["Solar"].shift(12)
df_solar.head()

Solar_Ratio_Autocorrelations = sm.tsa.acf(df_solar["Solar"],fft=False) #Autocorrelations
print(Solar_Ratio_Autocorrelations)

Solar_Autocorrelations = sm.tsa.acf(df_solar["Solar"], fft=False) #Autocorrelations
print(Solar_Autocorrelations)

```

ADF Test Statistic : -5.220109587451363
p-value : 7.997896033360128e-06
#Lags Used : 7
Number of Observations : 40
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

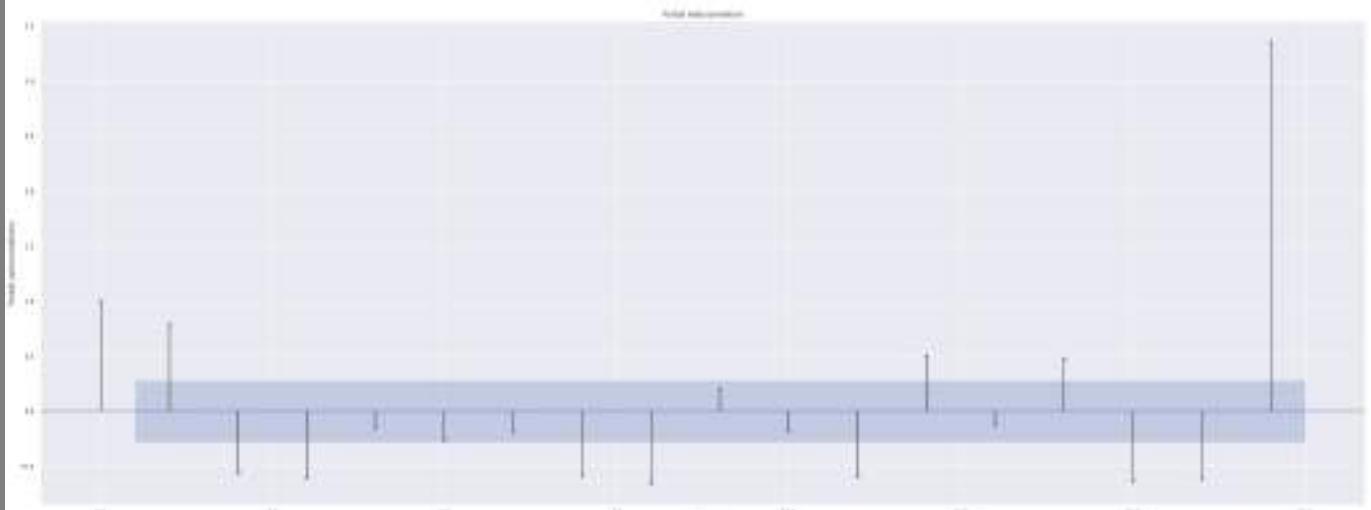
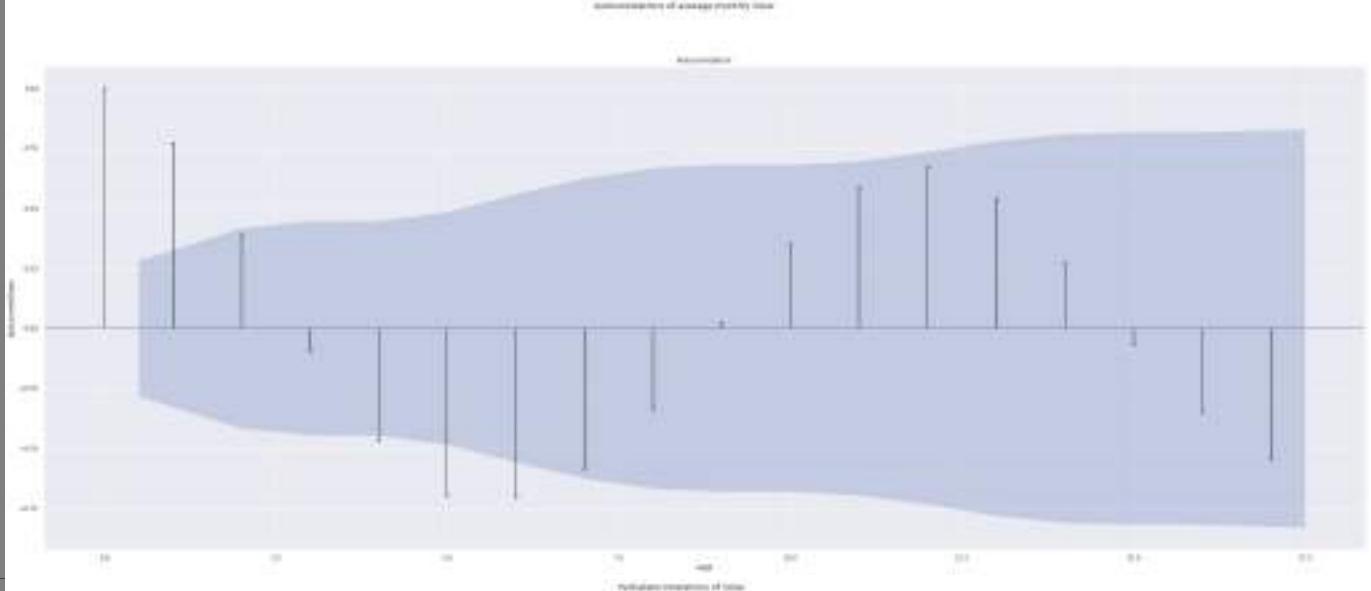
[ 1.    0.77030015  0.38989331 -0.09095154 -0.46743581 -0.69629739
 -0.7028239 -0.58499881 -0.33691503  0.02290571  0.35150157  0.58539428
  0.67078862  0.5369131   0.27239882 -0.06330717 -0.34742861 -0.54609372
 -0.53851649 -0.43904541 -0.23343633 -0.00860393  0.24493205  0.40076597
  0.47945209  0.39202109  0.18965779 -0.04248235 -0.2125484   -0.29919937
 -0.3054867 -0.21832683 -0.11280112  0.01749715  0.15097483  0.22488667
  0.23730704  0.16974959  0.07333613 -0.06626634 -0.14263412]
[ 1.    0.77030015  0.38989331 -0.09095154 -0.46743581 -0.69629739
 -0.7028239 -0.58499881 -0.33691503  0.02290571  0.35150157  0.58539428
  0.67078862  0.5369131   0.27239882 -0.06330717 -0.34742861 -0.54609372
 -0.53851649 -0.43904541 -0.23343633 -0.00860393  0.24493205  0.40076597
  0.47945209  0.39202109  0.18965779 -0.04248235 -0.2125484   -0.29919937
 -0.3054867 -0.21832683 -0.11280112  0.01749715  0.15097483  0.22488667
  0.23730704  0.16974959  0.07333613 -0.06626634 -0.14263412]

```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/regression/linear_model.py:1434: RuntimeWarning: invalid value encountered in sqrt
      return rho, np.sqrt(sigmasq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
  FutureWarning,

```



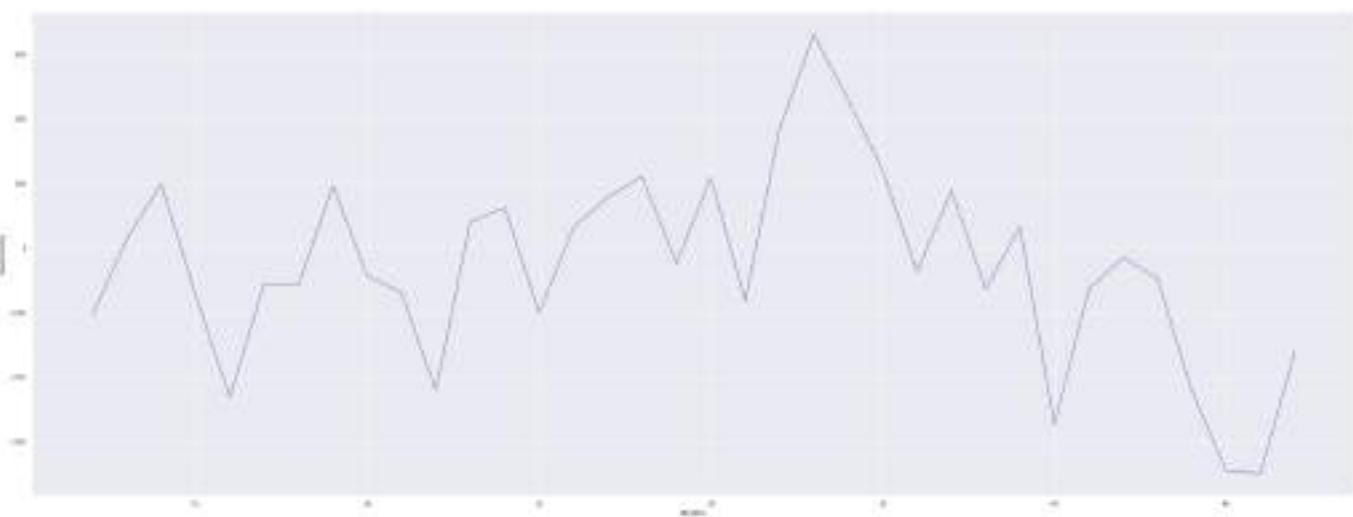
The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the of the autocorrelation plot, indicating that the lags beforehand influenced the average monthly solar outputs.

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly Solar output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_solar['Seasonal Difference'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608b64e50>
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted in the average monthly solar outputs.

```
In [ ]: #Bell Curves

solarResults_mean = np.mean(df_solar["Ratio"])
solarResults_std = np.std(df_solar["Ratio"])

solarResultspdf = stats.norm.pdf(df_solar["Ratio"].sort_values(), solarResults_mean, solarResults_std)

plt.plot(df_solar["Ratio"].sort_values(), solarResultspdf)
plt.xlim([0,100])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_solar["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
plt.suptitle("Frequency distribution of average monthly solar outputs to energy prices per EUR/MWH ratios (scaled)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves

solarResults_mean = np.mean(df_solar["Solar"])
solarResults_std = np.std(df_solar["Solar"])

solarResultspdf = stats.norm.pdf(df_solar["Solar"].sort_values(), solarResults_mean, solarResults_std)

plt.plot(df_solar["Solar"].sort_values(), solarResultspdf)
plt.xlim([0,3000])
plt.xlabel("Output in MWH ", size=15)
```

```
plt.title(f'Skewness for data: {skew(df_solar["Solar"])}')
plt.suptitle("Frequency distribution of average monthly solar outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



These bell shaped curves are roughly symmetrical, hence they have a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
In [ ]: df_solar.describe(include = 'all') # Description Tables
```

	Price	Solar	Dates	Ratio	First Difference Solar Ratio	Seasonal Difference Solar Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	1431.838433	NaN	25.893597	-0.090680	-0.573367	-5.421337	-27.849403
std	10.320573	363.970512	NaN	8.970498	5.251212	8.956610	231.192504	150.930918
min	32.618667	831.668056	NaN	12.405067	-13.878257	-15.216430	-534.930667	-348.977778
25%	51.528411	1119.899182	NaN	18.508466	-3.103624	-7.215583	-155.670166	-86.418553
50%	59.622281	1402.754094	NaN	25.727534	0.156037	-0.984676	38.953841	-40.037522
75%	64.999583	1742.752688	NaN	31.372603	3.021115	4.457595	150.378294	80.970536
max	79.492083	2025.119624	NaN	48.691689	11.726114	17.580699	458.753351	331.042953

Below is the box and whisker plot for the distribution of the average monthly outputs of solar power and its the average monthly prices of energy per EUR/MWH.

```
In [ ]: # Box and Whisker Plot
sns.set(style="darkgrid")

plt.suptitle('Distribution of average monthly solar power outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly solar power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')
sns.boxplot(x=Rounded_Y, y=solar)
plt.show()
```



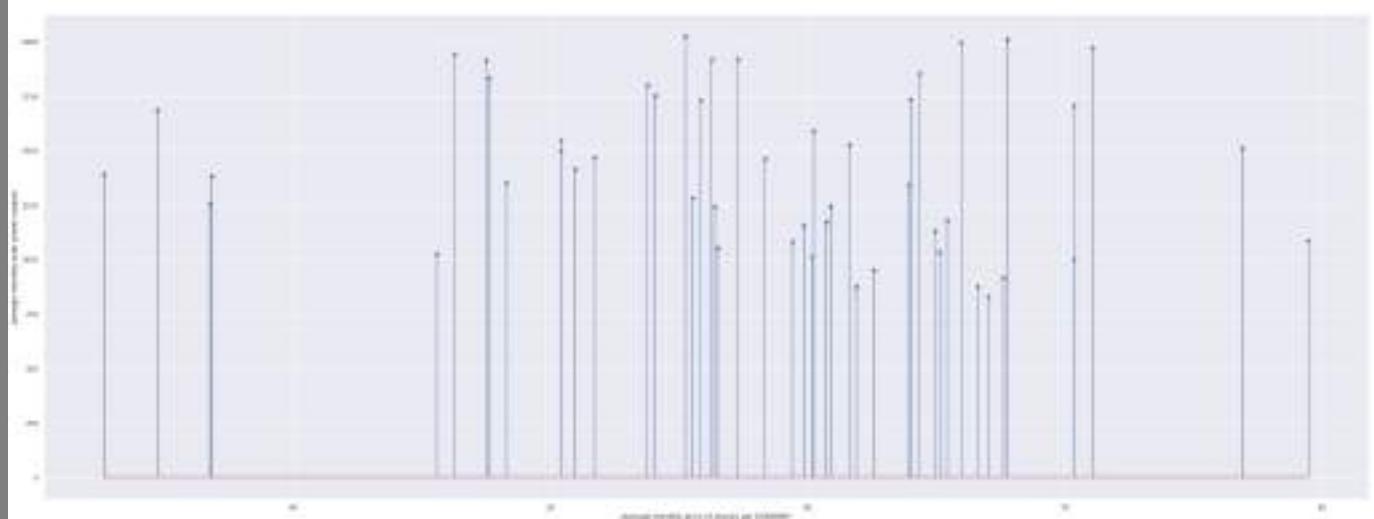
This box and whisker plot depicts the frequencies between the distribution of the monthly average output of solar power produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of solar power produced, which was inbetween 1000 and 1800 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), solar power output was at roughly 1400 units. At approximately 71.07 EUR/MWH, solar power produced its highest output, which was slightly below 7000 units. The lowest amount produced, whcih was slightly above 800 units, was at the price of approximately 66.62 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 110 units. At approximately 66.62 EUR/MWH, solar power produced slightly above 800 units.

```
In [ ]: plt.suptitle('Distribution of average monthly solar power outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly solar power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

# Stem Plot
plt.stem(Rounded_Y, solar)
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of solar power produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of solar power produced, which was inbetween 1000 and 1800 units, was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), solar power output was at roughly 1400 units. At approximately 71.07 EUR/MWH, solar power produced its highest output, which was slightly below 7000 units. The lowest amount produced, whcih was slightly above 800 units, was at the price of approximately 66.62 EUR/MWH. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 110 units. At approximately 66.62 EUR/MWH, solar power produced slightly above 800 units. Each blue dot at the end of the blue lines represent an observation.

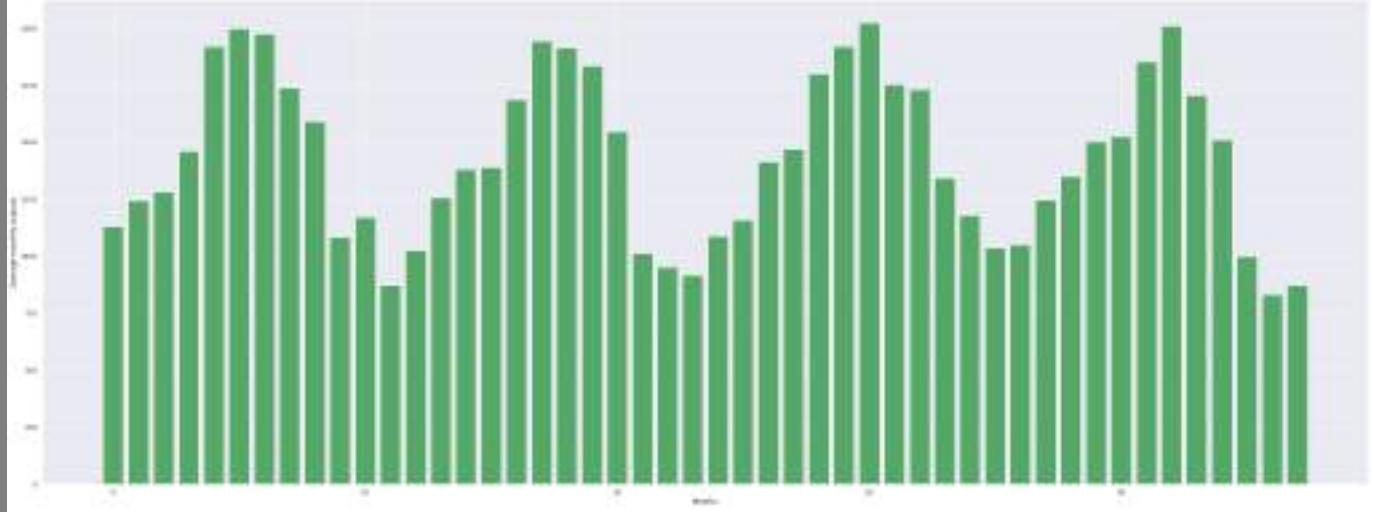
```
In [ ]: #Outputs in MWH Histogram
solar_Dict = {key: i for i, key in enumerate(solar)}

def Hist_solar(solar_Dict):
    for k, v in solar_Dict.items(): print(f"{v}:{k}")
print(solar_Dict)

plt.bar(list(solar_Dict.values()), solar_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of solar power")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{1130.392905866303: 0, 1244.5252976190477: 1, 1283.479138627187: 2, 1461.5194986072424: 3, 1920.2728494623657: 4, 1998.6606397774688: 5, 1973.7405913978494: 6, 1738.9502688172042: 7, 1591.2902777777779: 8, 1082.279569892473: 9, 1171.719444444444: 10, 874.3135935397039: 11, 1025.741935483871: 12, 1258.058908045977: 13, 1382.9057873485867: 14, 1390.4166666666667: 15, 1689.1814516129032: 16, 1942.8291666666667: 17, 1915.7752355316286: 18, 1835.8521505376343: 19, 1548.213888888889: 20, 1013.2832214765101: 21, 951.3819444444445: 22, 914.9758064516129: 23, 1088.4180107526881: 24, 1157.4613095238096: 25, 1415.0915208613728: 26, 1468.8277777777778: 27, 1800.0524193548388: 28, 1919.023611111112: 29, 2025.119623655914: 30, 1754.159946236559: 31, 1731.3666666666666: 32, 1344.3261744966444: 33, 1180.645833333333: 34, 1035.4260752688172: 35, 1051.4193548387098: 36, 1246.110119047619: 37, 1352.3189771197847: 38, 1501.6472222222221: 39, 1526.899193548387: 40, 1855.1972222222223: 41, 2010.3458950201884: 42, 1706.9301075268818: 43, 1510.6527777777778: 44, 999.786577181208: 45, 831.6680555555556: 46, 875.5900537634409: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is multimodal, with sudden troughs in the average monthly outputs occurring roughly every ten months. It would be fair to assume that this fluctuating pattern is seasonal.

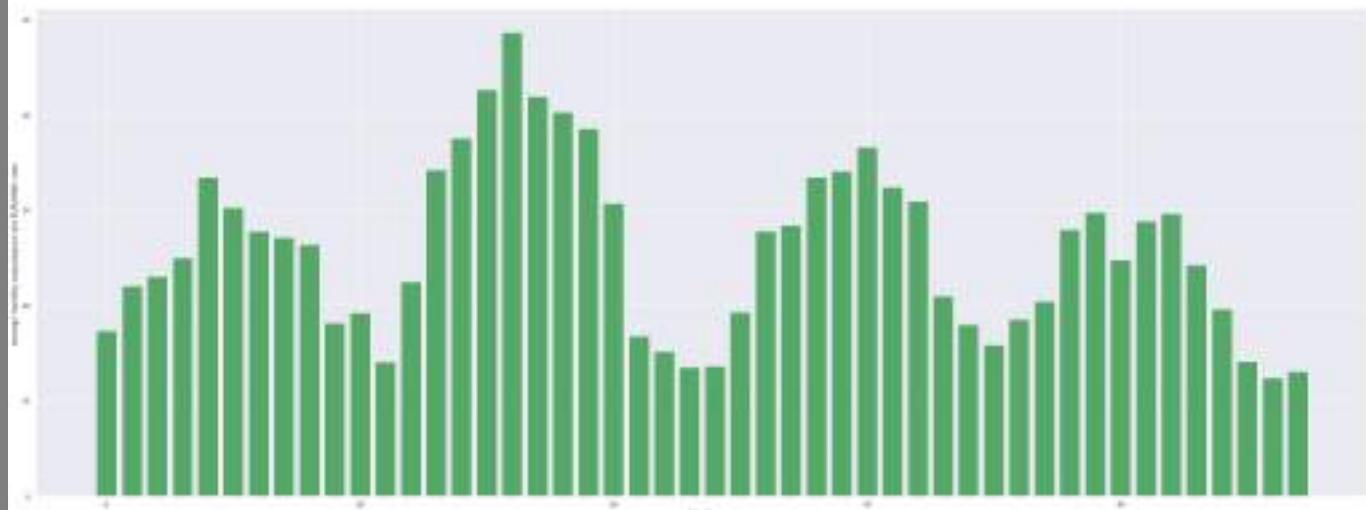
```
In [ ]: pdToListSolar_Dict = {key: i for i, key in enumerate(pdToListSolar)}
```

```
def Hist_pdToListSolar(pdToListSolar_Dict):
    for k, v in pdToListSolar_Dict.items(): print(f"{v}:{k}")
#Output/price per EUR/MWH Histogram
print(pdToListSolar_Dict)

plt.bar(list(pdToListSolar_Dict.values()), pdToListSolar_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of solar power")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{17.40431074174863: 0, 22.072370113974813: 1, 23.116394150367523: 2, 25.04571085897575: 3, 33.516090119873006: 4
, 30.294256688934063: 5, 27.770984310936953: 6, 27.171919681710804: 7, 26.409356563389956: 8, 18.218119907024636
: 9, 19.294934118635627: 10, 14.124212024158545: 11, 22.50484141080495: 12, 34.23095492670875: 13, 37.5605813469
7677: 14, 42.62641023544801: 15, 48.69168915761797: 16, 41.992300014259236: 17, 40.33039840514782: 18, 38.566427
62311195: 19, 30.71511844336071: 20, 16.836861346184342: 21, 15.202395293570383: 22, 13.536119344341051: 23, 13.
692156062744463: 24, 19.34331401855214: 25, 27.76873058801158: 26, 28.40075301649708: 27, 33.47525909950876: 28,
34.110989208491006: 29, 36.652036882430444: 30, 32.43379390110128: 31, 31.01887333308118: 32, 21.029645765169665
: 33, 18.044231307658848: 34, 15.892644393579458: 35, 18.605248530678267: 36, 20.469275895203573: 37, 28.0100847
93555276: 38, 29.79415258760826: 39, 24.77374588708186: 40, 28.830627493758996: 41, 29.658363008835465: 42, 24.2
58601834739714: 43, 19.64079308593233: 44, 14.209140239813733: 45, 12.405067258490595: 46, 13.142357745100368: 4
7}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15
-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18
, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03
': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35
, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08
': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



This histogram is multimodal, with the modals occurring as a result of sharp increases in output amount followed by deep trenches occurring roughly every ten months. It would be fair to assume that this fluctuating pattern is seasonal.

```
In [ ]: standardized_residualsSolarQuad/standardized_residualsPriceQuad
```

```
Out[ ]: array([-3.29480787e-01, -8.27448694e-04,  1.15642925e-02, -3.28919969e-01,
 6.96452004e-02, -3.64098193e-01, -6.77398664e-01, -7.19551697e-01,
-6.10574524e-01,  6.11779704e-02, -2.86219317e-01,  4.98187729e-01,
3.40476883e+00,  6.95778737e+00,  8.60513314e+00,  2.37093789e+01,
7.44811021e+01,  1.02551385e+01,  1.36004139e+01,  9.71494784e+01,
5.16109647e+02,  8.72381630e-01,  6.02033703e-01, -8.05795318e-01,
-2.06096195e+00,  1.81150221e+00, -7.28021435e-01, -4.00977475e-01,
-3.51657858e-01, -3.97143213e-01, -1.19516372e+00, -9.42602815e-02,
1.97352333e-01,  1.74628468e+00,  1.44554655e+00,  6.18175210e-01,
-4.46206860e-01,  4.98099950e-01, -4.76505719e-01, -2.23907841e-01,
7.51964192e-01,  7.24623450e-01,  7.06500472e-01,  1.63570501e+00,
2.75592206e+00,  6.99256894e-01, -7.42873547e-02,  2.05245791e-02])
```

```
In [ ]: SolarRatioQuad = list(standardized_residualsSolarQuad/standardized_residualsPriceQuad)
```

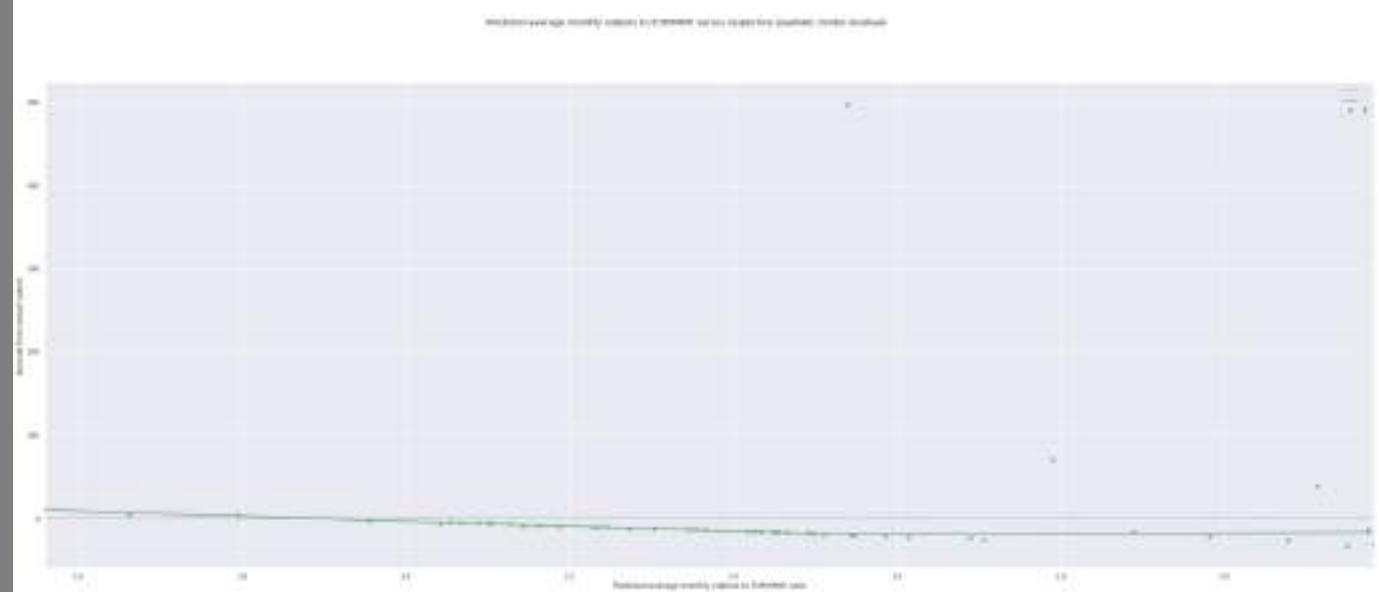
```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
SolarQuadRatioPredict = Solar_ypred/ypred

sns.residplot(x = SolarQuadRatioPredict , y = standardized_residualsSolarQuad/standardized_residualsPriceQuad ,

plt.suptitle("Predicted average monthly outputs to EUR/MWH versus respective quadratic model residuals ")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#..)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff606137b50>
```



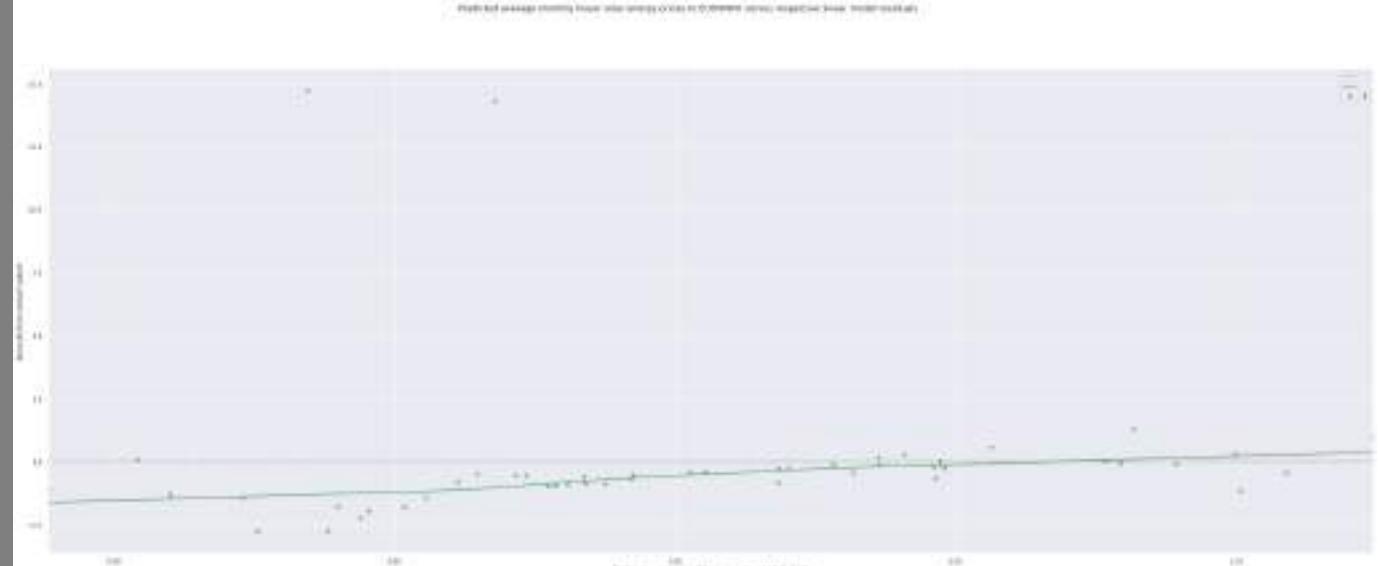
With the exception of few outliers, the residuals seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals
SolarRegRatioPredict = predictionsSolar/predictions

sns.residplot(x = SolarRegRatioPredict, y = standardized_residualsSolar/standardized_residualsPricereg, lowess =
plt.suptitle("Predicted average monthly linear solar energy prices to EUR/MWh versus respective linear model re-
plt.xlabel("Predicted average monthly energy prices to EUR/MWh")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

Out[]: <matplotlib.legend.Legend at 0x7ff607b849d0>



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the

regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of solar power.

```
In [ ]: modelsolar = stats.linregress([1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 19])
```

```
In [ ]:
```

There is a very weak and negative correlation between the outputs and their respective the average monthly prices of energy per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price
```

```
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

```
In [ ]: print(list(Logpred))
```

```
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: print(list(Solar_Logpred))
```

```
[59.39046843867489, 58.81094966525369, 58.61315759400729, 57.709139642130545, 55.37977306085032, 54.98175111986603, 55.10828518831734, 56.300456820432736, 57.05021537342911, 59.634768772879404, 59.1806287553146, 60.690737071435635, 59.921844335227995, 58.74223138822358, 58.10830871957703, 58.070171469468775, 56.553163028179114, 55.26524107673616, 55.402610150903676, 55.80842770604699, 57.268940098889914, 59.985104716869635, 60.29941471763211, 60.484270568899255, 59.60360021798101, 59.253025878809396, 57.94488239931381, 57.67203111478185, 55.99020446534603, 55.3861161950466, 54.84740295062389, 56.22322814357042, 56.338963328084965, 58.30420060947502, 59.135304112060155, 59.87267213508722, 59.79146463897687, 58.80290257400928, 58.26361634499737, 57.505387063108834, 57.37716762229391, 55.71020114615254, 54.92241805340366, 56.46304243097454, 57.45966044308988, 60.05363529462411, 60.907274107810025, 60.684255715426]
```

```
In [ ]: dfSolar_Log = ({"Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], "Solar_Log" : [59.39046843867489, 58.81094966525369, 58.61315759400729, 57.709139642130545, 55.37977306085032, 54.98175111986603, 55.10828518831734, 56.300456820432736, 57.05021537342911, 59.634768772879404, 59.1806287553146, 60.690737071435635, 59.921844335227995, 58.74223138822358, 58.10830871957703, 58.070171469468775, 56.553163028179114, 55.26524107673616, 55.402610150903676, 55.80842770604699, 57.268940098889914, 59.985104716869635, 60.29941471763211, 60.484270568899255, 59.60360021798101, 59.253025878809396, 57.94488239931381, 57.67203111478185, 55.99020446534603, 55.3861161950466, 54.84740295062389, 56.22322814357042, 56.338963328084965, 58.30420060947502, 59.135304112060155, 59.87267213508722, 59.79146463897687, 58.80290257400928, 58.26361634499737, 57.505387063108834, 57.37716762229391, 55.71020114615254, 54.92241805340366, 56.46304243097454, 57.45966044308988, 60.05363529462411, 60.907274107810025, 60.684255715426]}
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) + '\n')
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary")
adfuller_test(df_Solar_Log["Solar_Log"])

test_result=adfuller(df_Solar_Log["Solar_Log"])
```

```

df_Solar_Log['First Difference'] = df_Solar_Log["Solar_Log"] - df_Solar_Log["Solar_Log"].shift(1) # Seasonality
df_Solar_Log['Seasonal Difference']=df_Solar_Log["Solar_Log"] - df_Solar_Log["Solar_Log"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Solar_Log.head() #Predictive Logarithmic Seasonality Plot
df_Solar_Log[['Seasonal Difference']].plot()
# Seasonality Plot

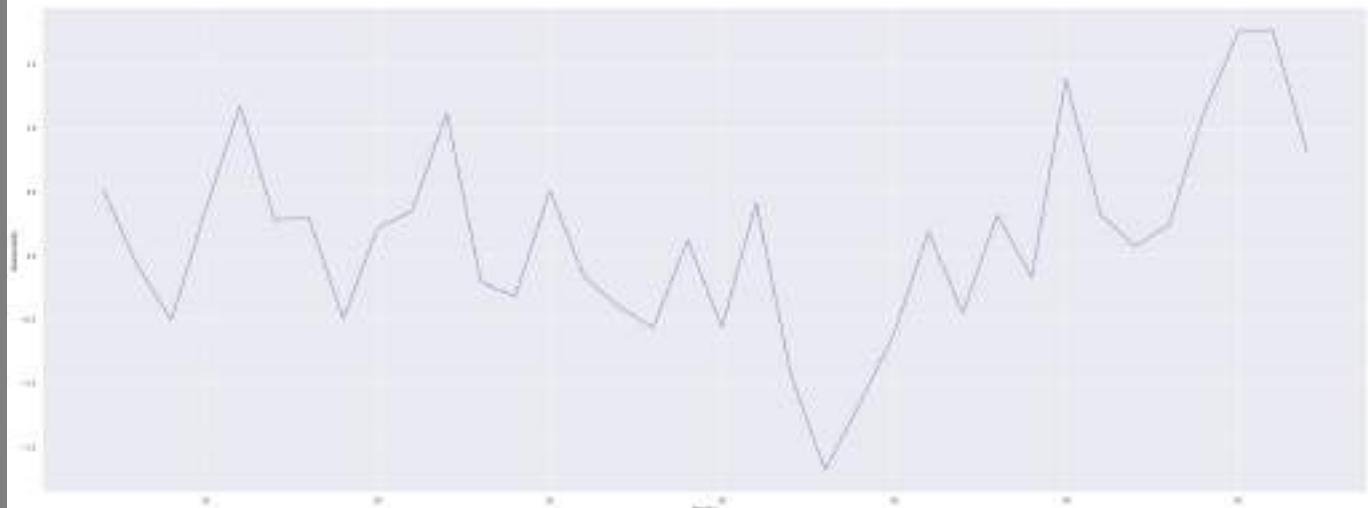
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834
, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.6312803685
45422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.3578
44118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848,
26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.07642699916
3463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.8933
89962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21
.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.018681703731
89, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Solar_Log': [59.39046843867489, 58.81094966525
369, 58.61315759400729, 57.709139642130545, 55.37977306085032, 54.98175111986603, 55.10828518831734, 56.30045682
0432736, 57.05021537342911, 59.634768772879404, 59.1806287553146, 60.690737071435635, 59.921844335227995, 58.742
23138822358, 58.10830871957703, 58.070171469468775, 56.553163028179114, 55.26524107673616, 55.402610150903676, 5
5.80842770604699, 57.268940098889914, 59.985104716869635, 60.29941471763211, 60.484270568899255, 59.603600217981
01, 59.253025878809396, 57.94488239931381, 57.67203111478185, 55.99020446534603, 55.3861161950466, 54.8474029506
2389, 56.22322814357042, 56.338963328084965, 58.30420060947502, 59.135304112060155, 59.87267213508722, 59.791464
63897687, 58.80290257400928, 58.26361634499737, 57.505387063108834, 57.37716762229391, 55.71020114615254, 54.922
41805340366, 56.46304243097454, 57.45966044308988, 60.05363529462411, 60.907274107810025, 60.684255715426]}

    Price_Log Solar_Log
0    27.297348   59.390468
1    23.917581   58.810950
2    23.577680   58.613158
3    24.695022   57.709140
4    24.276743   55.379773
5    27.702157   54.981751
6    29.713461   55.108285
7    26.922107   56.300457
8    25.445032   57.050215
9    25.110405   59.634769
10   25.631280   59.180629
11   26.094917   60.690737
12   19.653934   59.921844
13   16.170990   58.742231
14   16.197004   58.108309
15   14.539966   58.070171
16   15.357844   56.553163
17   19.925256   55.265241
18   20.412855   55.402610
19   20.452442   55.808428
20   21.558593   57.268940
21   25.416478   59.985105
22   26.362963   60.299415
23   28.341491   60.484271
24   33.035963   59.603600
25   25.280488   59.253026
26   21.777315   57.944882
27   22.076427   57.672031
28   22.887202   55.990204
29   23.868007   55.386116
30   23.471186   54.847403
31   23.010200   56.223228
32   23.693728   56.338963
33   26.893390   58.304201
34   27.487399   59.135304
35   27.377158   59.872672
36   23.968137   59.791465
37   25.690591   58.802903
38   20.719732   58.263616
39   21.556675   57.505387
40   25.989167   57.377168
41   27.060244   55.710201
42   28.415796   54.922418
43   29.434036   56.463042
44   32.018682   57.459660
45   29.433366   60.053635
46   27.958095   60.907274
47   28.123467   60.684256

ADF Test Statistic : -5.220109587451467
p-value : 7.997896033356194e-06
#Lags Used : 7
Number of Observations : 40
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609760ed0>



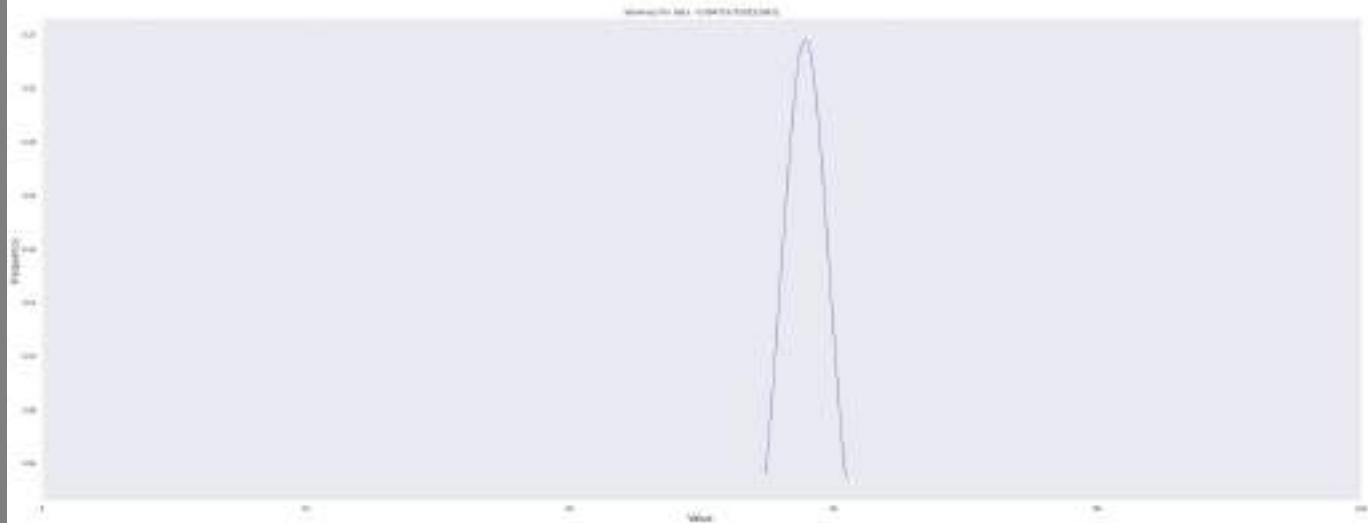
The blue line represents the trend line among the values themselves. As one can observe, there is a consistent upwards and downwards pattern depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Solar_LogResults_mean = np.mean(df_Solar_Log["Solar_Log"])
Solar_LogResults_std = np.std(df_Solar_Log["Solar_Log"])

Solar_LogResultspdf = stats.norm.pdf(df_Solar_Log["Solar_Log"].sort_values(), Solar_LogResults_mean, Solar_LogResults_std)

plt.plot(df_Solar_Log["Solar_Log"].sort_values(), Solar_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Solar_Log["Solar_Log"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic solar prices per EUR/MWH (scaled : 0-100) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetrical, hence it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and -0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
Solar_Log_Dict = {key: i for i, key in enumerate(df_Solar_Log["Solar_Log"])}

def Hist_Solar_Log(Solar_Log_Dict):
```

```

for k, v in Solar_Log_Dict.items(): print(f"{v}:{k}")
print(Solar_Log_Dict)

plt.bar(list(Solar_Log_Dict.values()), Solar_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly output of predicted logarithmic solar prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{59.39046843867489: 0, 58.81094966525369: 1, 58.61315759400729: 2, 57.709139642130545: 3, 55.37977306085032: 4, 54.98175111986603: 5, 55.10828518831734: 6, 56.300456820432736: 7, 57.05021537342911: 8, 59.634768772879404: 9, 59.1806287553146: 10, 60.690737071435635: 11, 59.921844335227995: 12, 58.74223138822358: 13, 58.10830871957703: 14, 58.070171469468775: 15, 56.553163028179114: 16, 55.26524107673616: 17, 55.402610150903676: 18, 55.80842770604699: 19, 57.268940098889914: 20, 59.985104716869635: 21, 60.29941471763211: 22, 60.484270568899255: 23, 59.60360021798101: 24, 59.253025878809396: 25, 57.94488239931381: 26, 57.67203111478185: 27, 55.99020446534603: 28, 55.3861161950466: 29, 54.84740295062389: 30, 56.22322814357042: 31, 56.338963328084965: 32, 58.30420060947502: 33, 59.135304112060155: 34, 59.87267213508722: 35, 59.79146463897687: 36, 58.80290257400928: 37, 58.26361634499737: 38, 57.505387063108834: 39, 57.37716762229391: 40, 55.71020114615254: 41, 54.92241805340366: 42, 56.46304243097454: 43, 57.45966044308988: 44, 60.05363529462411: 45, 60.907274107810025: 46, 60.684255715426: 47}
```



The green bars represent the observation value for each respective month. This histogram is uniform with slightly trenches roughly every ten months.

In []: df_Solar_Log.describe(include = 'all') # Description Tables

	Price_Log	Solar_Log	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	0.027527	0.141408
std	4.072442	1.848097	1.173903	0.766367
min	14.539966	54.847403	-2.329367	-1.680904
25%	22.001649	56.281150	-0.763561	-0.411136
50%	25.195447	58.007527	-0.197792	0.203295
75%	27.317301	59.443751	0.790431	0.438799
max	33.035963	60.907274	2.716165	1.771970

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

In []: print(list(predictions))

```
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895]
```

```
In [ ]: print(list(predictionssolar))

[59.39046843867489, 58.81094966525369, 58.61315759400729, 57.709139642130545, 55.37977306085032, 54.981751119866
03, 55.10828518831734, 56.300456820432736, 57.05021537342911, 59.634768772879404, 59.1806287553146, 60.690737071
435635, 59.921844335227995, 58.74223138822358, 58.10830871957703, 58.070171469468775, 56.553163028179114, 55.265
24107673616, 55.402610150903676, 55.80842770604699, 57.26894009889914, 59.985104716869635, 60.29941471763211, 6
0.484270568899255, 59.60360021798101, 59.253025878809396, 57.94488239931381, 57.67203111478185, 55.9902044653460
3, 55.3861161950466, 54.84740295062389, 56.22322814357042, 56.338963328084965, 58.30420060947502, 59.13530411206
0155, 59.87267213508722, 59.79146463897687, 58.80290257400928, 58.26361634499737, 57.505387063108834, 57.3771676
2229391, 55.71020114615254, 54.92241805340366, 56.46304243097454, 57.45966044308988, 60.05363529462411, 60.90727
4107810025, 60.684255715426]
```

```
In [ ]: dfSolarReg = {"Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509,
 "Solar_Reg" : [59.39046843867489, 58.81094966525369, 58.61315759400729, 57.709139642130545, 55.37977306085032,
 "Dates": ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', ''],
 print(dfSolarReg) #Dataframes
 df_SolarReg= pd.DataFrame.from_dict(dfSolarReg, orient = "columns")
 print(df_SolarReg)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_SolarReg["Solar_Reg"])

test_result=adfuller(df_SolarReg["Solar_Reg"])

df_SolarReg['First Difference'] = df_SolarReg["Solar_Reg"]- df_SolarReg["Solar_Reg"].shift(1) # Seasonality value
df_SolarReg['Seasonal Difference']=df_SolarReg["Solar_Reg"]- df_SolarReg["Solar_Reg"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_SolarReg.head() #Predictive Linear Seasonality Plot
df_SolarReg['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'Solar_Reg': [59.39046843867489, 58.81094966525369, 58.61315759400729, 57.709139642130545, 55.37977306085032, 54.98175111986603, 55.10828518831734, 56.300456820432736, 57.05021537342911, 59.634768772879404, 59.1806287553146, 60.690737071435635, 59.921844335227995, 58.74223138822358, 58.10830871957703, 58.070171469468775, 56.553163028179114, 55.26524107673616, 55.402610150903676, 55.80842770604699, 57.26894009888914, 59.985104716869635, 60.29941471763211, 60.484270568899255, 59.60360021798101, 59.253025878809396, 57.94488239931381, 57.67203111478185, 55.99020446534603, 55.3861161950466, 54.84740295062389, 56.22322814357042, 56.338963328084965, 58.30420060947502, 59.135304112060155, 59.87267213508722, 59.79146463897687, 58.80290257400928, 58.26361634499737, 57.505387063108834, 57.37716762229391, 55.71020114615254, 54.92241805340366, 56.46304243097454, 57.45966044308988, 60.05363529462411, 60.907274107810025, 60.684255715426], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	Solar_Reg	Dates
0	52.821613	59.390468	2015-01
1	53.036006	58.810950	2015-02
2	53.250399	58.613158	2015-03
3	53.464792	57.709140	2015-04
4	53.679185	55.379773	2015-05
5	53.893578	54.981751	2015-06
6	54.107971	55.108285	2015-07
7	54.322364	56.300457	2015-08
8	54.536757	57.050215	2015-09
9	54.751150	59.634769	2015-10
10	54.965543	59.180629	2015-11
11	55.179936	60.690737	2015-12
12	55.394329	59.921844	2016-01
13	55.608722	58.742231	2016-02
14	55.823115	58.108309	2016-03
15	56.037508	58.070171	2016-04
16	56.251901	56.553163	2016-05
17	56.466294	55.265241	2016-06
18	56.680687	55.402610	2016-07
19	56.895080	55.808428	2016-08
20	57.109473	57.268940	2016-09
21	57.323866	59.985105	2016-10
22	57.538259	60.299415	2016-11
23	57.752652	60.484271	2016-12
24	57.967045	59.603600	2017-01
25	58.181438	59.253026	2017-02
26	58.395831	57.944882	2017-03
27	58.610224	57.672031	2017-04
28	58.824617	55.990204	2017-05
29	59.039010	55.386116	2017-06
30	59.253403	54.847403	2017-07
31	59.467796	56.223228	2017-08
32	59.682189	56.338963	2017-09
33	59.896582	58.304201	2017-10
34	60.110975	59.135304	2017-11
35	60.325368	59.872672	2017-12
36	60.539761	59.791465	2018-01
37	60.754154	58.802903	2018-02
38	60.968547	58.263616	2018-03
39	61.182939	57.505387	2018-04
40	61.397332	57.377168	2018-05
41	61.611725	55.710201	2018-06
42	61.826118	54.922418	2018-07
43	62.040511	56.463042	2018-08
44	62.254904	57.459660	2018-09
45	62.469297	60.053635	2018-10
46	62.683690	60.907274	2018-11
47	62.898083	60.684256	2018-12

ADF Test Statistic : -5.220109587451467

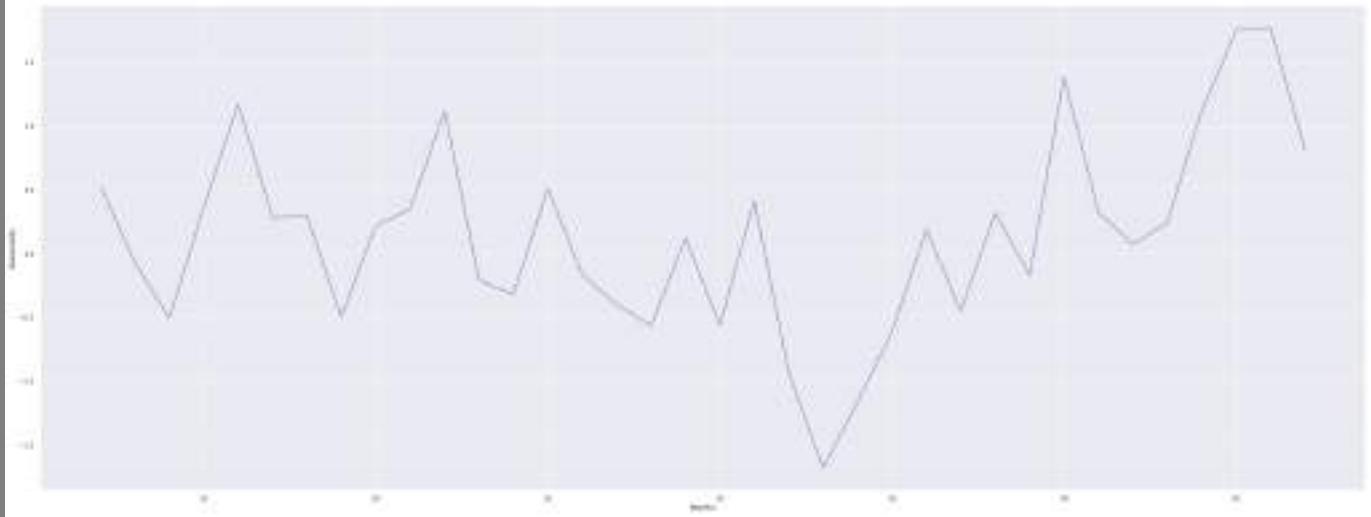
p-value : 7.997896033356194e-06

#Lags Used : 7

Number of Observations : 40

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606ac9950>



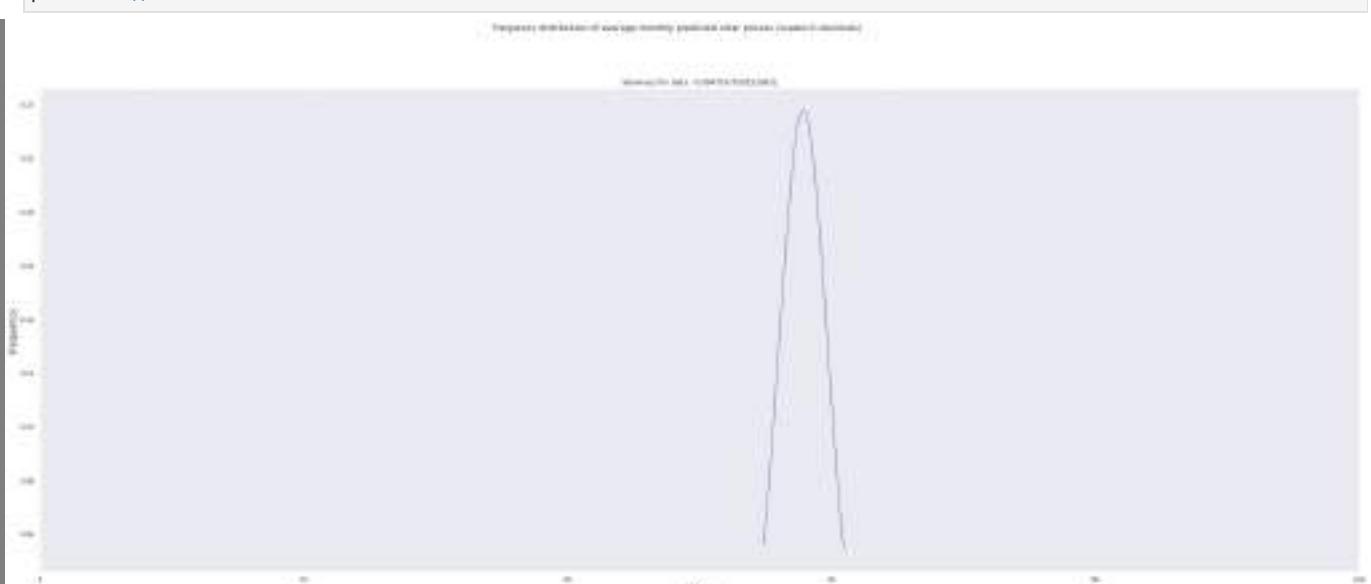
The blue line represents the trend line among the values themselves. As one can observe, there is an upwards and downwards pattern depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
SolarRegResults_mean = np.mean(df_SolarReg["Solar_Reg"])
SolarRegResults_std = np.std(df_SolarReg["Solar_Reg"])

SolarRegResultspdf = stats.norm.pdf(df_SolarReg["Solar_Reg"].sort_values(), SolarRegResults_mean, SolarRegResults_std)

plt.plot(df_SolarReg["Solar_Reg"].sort_values(), SolarRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_SolarReg["Solar_Reg"])}')
plt.suptitle("Frequency distribution of average monthly predicted solar prices (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetrical, hence it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and -0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
SolarReg_Dict = {key: i for i, key in enumerate(df_SolarReg["Solar_Reg"])}

def Hist_SolarReg(SolarReg_Dict):
    for k, v in SolarReg_Dict.items(): print(f"{v}:{k}")
```

```

print(SolarReg_Dict)

plt.bar(list(SolarReg_Dict.values()), SolarReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted solar prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

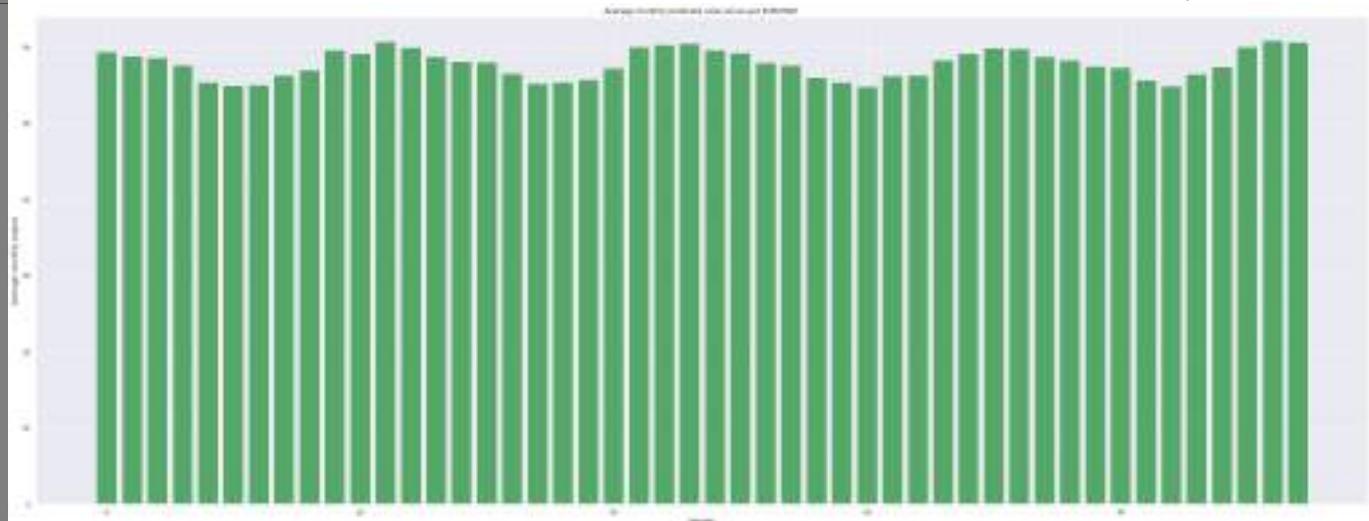
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{59.39046843867489: 0, 58.81094966525369: 1, 58.61315759400729: 2, 57.709139642130545: 3, 55.37977306085032: 4, 54.98175111986603: 5, 55.10828518831734: 6, 56.300456820432736: 7, 57.05021537342911: 8, 59.634768772879404: 9, 59.1806287553146: 10, 60.690737071435635: 11, 59.921844335227995: 12, 58.74223138822358: 13, 58.10830871957703: 14, 58.070171469468775: 15, 56.553163028179114: 16, 55.26524107673616: 17, 55.402610150903676: 18, 55.80842770604699: 19, 57.268940098889914: 20, 59.985104716869635: 21, 60.29941471763211: 22, 60.484270568899255: 23, 59.60360021798101: 24, 59.253025878809396: 25, 57.94488239931381: 26, 57.67203111478185: 27, 55.99020446534603: 28, 55.3861161950466: 29, 54.84740295062389: 30, 56.22322814357042: 31, 56.338963328084965: 32, 58.30420060947502: 33, 59.135304112060155: 34, 59.87267213508722: 35, 59.79146463897687: 36, 58.80290257400928: 37, 58.26361634499737: 38, 57.505387063108834: 39, 57.37716762229391: 40, 55.71020114615254: 41, 54.92241805340366: 42, 56.46304243097454: 43, 57.45966044308988: 44, 60.05363529462411: 45, 60.907274107810025: 46, 60.684255715426: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform with slight trenches roughly every ten months.

In []: df_SolarReg.describe(include = 'all') # Description Tables

	Price	Solar_Reg	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	57.859848	57.859848	NaN	0.027527	0.141408
std	3.001502	1.848097	NaN	1.173903	0.766367
min	52.821613	54.847403	NaN	-2.329367	-1.680904
25%	55.340731	56.281150	NaN	-0.763561	-0.411136
50%	57.859848	58.007527	NaN	-0.197792	0.203295
75%	60.378966	59.443751	NaN	0.790431	0.438799
max	62.898083	60.907274	NaN	2.716165	1.771970

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

In []: print(Solar_ypred/ypred)

```

Rounded_ypred = [round(item, 2) for item in ypred]
print(Rounded_ypred)
#Rounded Price

```

```
[2.16347056 2.42712655 2.434041 2.22995341 2.46620281 2.18779529  
1.95567057 2.04178612 2.14393004 2.44869045 2.30343464 2.58639171  
3.15137999 3.18284225 3.0784424 3.17625705 3.11402236 2.98218753  
2.89072035 2.790801 2.53942905 2.49147192 2.45570007 2.2737183  
1.66162914 2.35440618 2.54756729 2.49804157 2.49839365 2.5102962  
2.70535982 2.45212063 2.36666735 2.05437721 2.10156586 2.24678927  
2.6150305 2.23677466 2.68978702 2.54424741 2.0894841 2.10482641  
2.12754796 1.79443643 1.55856372 2.06694669 2.41877131 2.34490952]  
[27.19, 23.23, 22.88, 24.06, 23.6, 27.72, 30.59, 26.7, 24.9, 24.52, 25.12, 25.67, 19.55, 17.63, 17.64, 17.08, 17  
.33, 19.74, 20.09, 20.12, 21.01, 24.87, 26.0, 28.6, 36.04, 24.71, 21.19, 21.46, 22.2, 23.17, 22.77, 22.32, 23.0,  
26.67, 27.44, 27.29, 23.28, 25.19, 20.33, 21.01, 25.54, 26.88, 28.7, 30.17, 34.27, 30.17, 28.07, 28.3]
```

```
In [ ]: print(list(ypred))
```

```
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338  
71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672  
299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486  
, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506  
2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20  
1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,  
27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905  
53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857  
92181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: print(list(Solar_ypred))
```

```
[58.820605800457024, 56.37134467421464, 55.68656518274346, 53.64179116651381, 58.209066442039045, 60.65601724560  
4886, 59.81973627354879, 54.52075647413949, 53.38772545166902, 60.03800824589917, 57.86025427272131, 66.39862320  
07896, 61.59801585699268, 56.12435020145253, 54.316198698371416, 54.235683191249336, 53.96057244939353, 58.85832  
918480547, 58.084823460679914, 56.16031686438524, 53.34830406948494, 61.959756006901756, 63.848337009836094, 65.  
02676263249238, 59.87691787495525, 58.18243754268894, 53.994782577225536, 53.598411189067235, 55.46811456712309,  
58.174384054311005, 61.60468186248904, 54.72945106492728, 54.423319392248146, 54.781589280407886, 57.66354151191  
148, 61.3212366455536, 60.872695566715336, 56.34192681512813, 54.67812308484664, 53.445407050213575, 53.37502178  
6273464, 56.577560456017295, 61.06721675830047, 54.13900492476701, 53.41553197688826, 62.35871886423703, 67.8969  
9520379065, 66.35469640128008]
```

```
In [ ]: dfSolar_Quad = ({"Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23  
"Solar_Quad" : [58.820605800457024, 56.37134467421464, 55.68656518274346, 53.64179116651381, 58.209066442039045  
print(dfSolar_Quad) #Dataframes  
df_Solar_Quad= pd.DataFrame.from_dict(dfSolar_Quad, orient = "columns")  
print(df_Solar_Quad)  
  
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis, indicating it is non-stationary ")  
  
adfuller_test(df_Solar_Quad["Solar_Quad"])  
  
test_result=adfuller(df_Solar_Quad["Solar_Quad"])  
  
df_Solar_Quad['First Difference'] = df_Solar_Quad["Solar_Quad"]- df_Solar_Quad["Solar_Quad"].shift(1) # Seasonal  
df_Solar_Quad['Seasonal Difference']=df_Solar_Quad["Solar_Quad"]- df_Solar_Quad["Solar_Quad"].shift(12) #  
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")  
plt.ylabel('Seasonality')  
plt.xlabel('Months')  
df_Solar_Quad.head() #Predictive Quadratic Seasonality Plot  
df_Solar_Quad['Seasonal Difference'].plot()  
# Seasonality Plot
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.886735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Solar_Quad': [58.820605800457024, 56.37134467421464, 55.68656518274346, 53.64179116651381, 58.209066442039045, 60.656017245604886, 59.81973627354879, 54.52075647413949, 53.38772545166902, 60.03800824589917, 57.86025427272131, 66.3986232007896, 61.59801585699268, 56.12435020145253, 54.316198698371416, 54.235683191249336, 53.96057244939353, 58.85832918480547, 58.084823460679914, 56.16031686438524, 53.34830406948494, 61.959756006901756, 63.848337009836094, 65.02676263249238, 59.87691787495525, 58.18243754268894, 53.994782577225536, 53.598411189067235, 55.46811456712309, 58.174384054311005, 61.60468186248904, 54.72945106492728, 54.423319392248146, 54.781589280407886, 57.66354151191148, 61.3212366455536, 60.872695566715336, 56.34192681512813, 54.67812308484664, 53.445407050213575, 53.375021786273464, 56.577560456017295, 61.06721675830047, 54.13900492476701, 53.41553197688826, 62.35871886423703, 67.89699520379065, 66.35469640128008]}, 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price_Quad	Solar_Quad	Dates
0	27.188078	58.820606	2015-01
1	23.225548	56.371345	2015-02
2	22.878236	55.686565	2015-03
3	24.055117	53.641791	2015-04
4	23.602709	58.209066	2015-05
5	27.724722	60.656017	2015-06
6	30.587839	59.819736	2015-07
7	26.702482	54.520756	2015-08
8	24.901804	53.387725	2015-09
9	24.518415	60.038008	2015-10
10	25.119121	57.860254	2015-11
11	25.672300	66.398623	2015-12
12	19.546363	61.598016	2016-01
13	17.633406	56.124350	2016-02
14	17.644052	54.316199	2016-03
15	17.075344	54.235683	2016-04
16	17.328255	53.960572	2016-05
17	19.736629	58.858329	2016-06
18	20.093546	58.084823	2016-07
19	20.123369	56.160317	2016-08
20	21.007992	53.348304	2016-09
21	24.868735	61.959756	2016-10
22	26.000055	63.848337	2016-11
23	28.599305	65.026763	2016-12
24	36.035067	59.876918	2017-01
25	24.712150	58.182438	2017-02
26	21.194644	53.994783	2017-03
27	21.456173	53.598411	2017-04
28	22.201511	55.468115	2017-05
29	23.174311	58.174384	2017-06
30	22.771345	61.604682	2017-07
31	22.319233	54.729451	2017-08
32	22.995762	54.423319	2017-09
33	26.665789	54.781589	2017-10
34	27.438370	57.663542	2017-11
35	27.292830	61.321237	2017-12
36	23.278006	60.872696	2018-01
37	25.188915	56.341927	2018-02
38	20.328049	54.678123	2018-03
39	21.006372	53.445407	2018-04
40	25.544593	53.375022	2018-05
41	26.879918	56.577560	2018-06
42	28.703098	61.067217	2018-07
43	30.170478	54.139005	2018-08
44	34.272280	53.415532	2018-09
45	30.169486	62.358719	2018-10
46	28.070862	67.896995	2018-11
47	28.297338	66.354696	2018-12

ADF Test Statistic : -4.0679058211404895

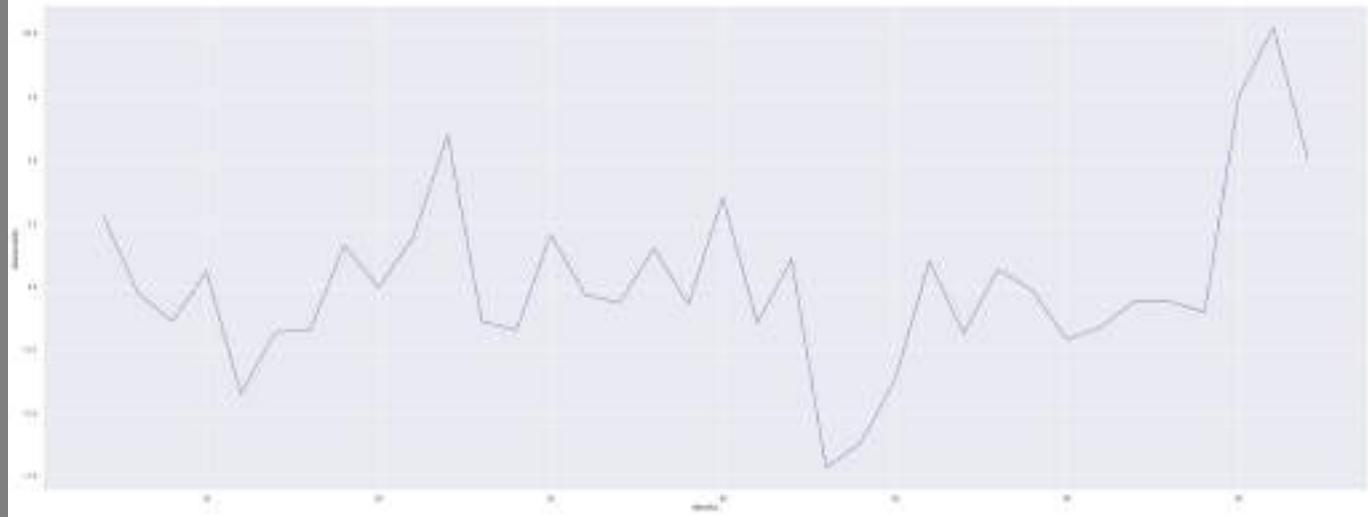
p-value : 0.0010938360120598184

#Lags Used : 7

Number of Observations : 40

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6063cfa90>



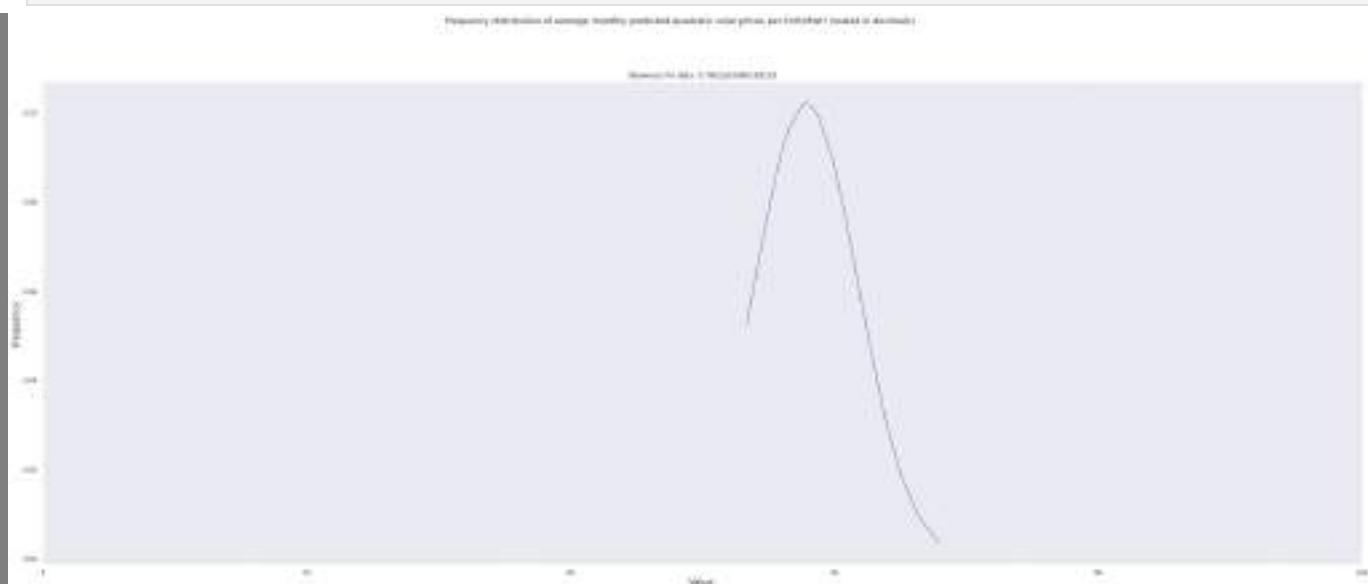
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Solar_QuadResults_mean = np.mean(df_Solar_Quad["Solar_Quad"])
Solar_QuadResults_std = np.std(df_Solar_Quad["Solar_Quad"])

Solar_QuadResultspdf = stats.norm.pdf(df_Solar_Quad["Solar_Quad"].sort_values(), Solar_QuadResults_mean, Solar_QuadResults_std)

plt.plot(df_Solar_Quad["Solar_Quad"].sort_values(), Solar_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Solar_Quad["Solar_Quad"])}')
plt.suptitle("Frequency distribution of average monthly predicted quadratic solar prices per EUR/MWH (scaled in percent) measured in December")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Solar_Quad_Dict = {key: i for i, key in enumerate(df_Solar_Quad["Solar_Quad"])}

def Hist_Solar_Quad(Solar_Quad_Dict):
    for k, v in Solar_Quad_Dict.items(): print(f"{v}:{k}")
print(Solar_Quad_Dict)
```

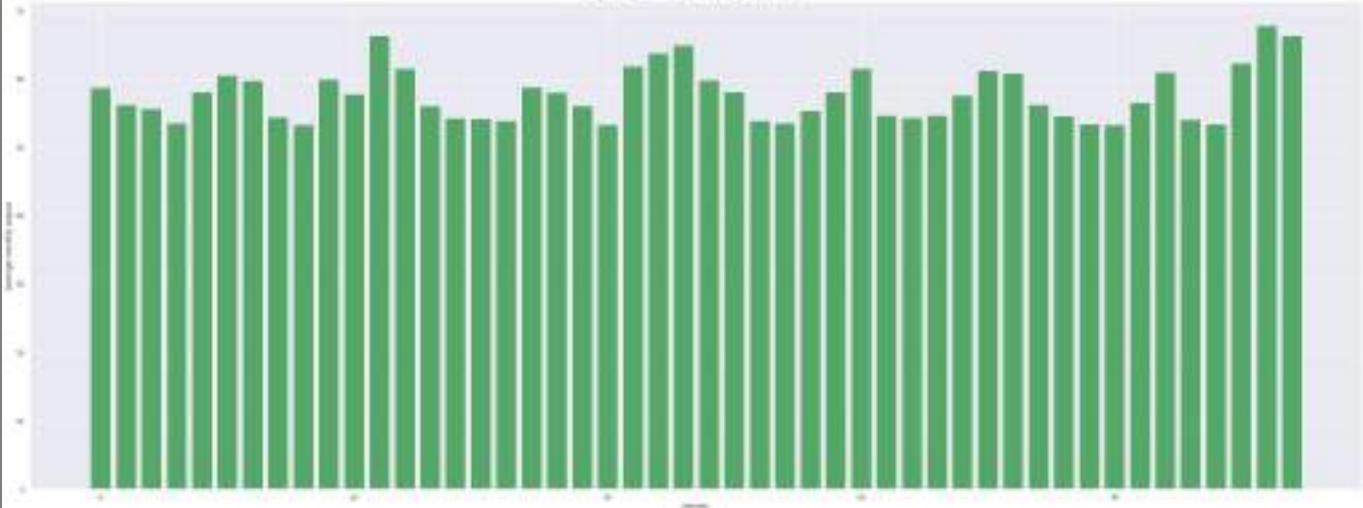
```

plt.bar(list(Solar_Quad_Dict.values()), Solar_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic solar prices per EUR/MWh ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{58.820605800457024: 0, 56.37134467421464: 1, 55.68656518274346: 2, 53.64179116651381: 3, 58.209066442039045: 4, 60.656017245604886: 5, 59.81973627354879: 6, 54.52075647413949: 7, 53.38772545166902: 8, 60.03800824589917: 9, 57.86025427272131: 10, 66.3986232007896: 11, 61.59801585699268: 12, 56.12435020145253: 13, 54.316198698371416: 14, 54.235683191249336: 15, 53.96057244939353: 16, 58.85832918480547: 17, 58.084823460679914: 18, 56.1603168643852: 19, 53.34830406948494: 20, 61.959756006901756: 21, 63.848337009836094: 22, 65.02676263249238: 23, 59.87691787495525: 24, 58.18243754268894: 25, 53.994782577225536: 26, 53.598411189067235: 27, 55.46811456712309: 28, 58.174384054311005: 29, 61.60468186248904: 30, 54.72945106492728: 31, 54.423319392248146: 32, 54.781589280407886: 33, 57.66354151191148: 34, 61.3212366455536: 35, 60.872695566715336: 36, 56.34192681512813: 37, 54.67812308484664: 38, 53.445407050213575: 39, 53.375021786273464: 40, 56.577560456017295: 41, 61.06721675830047: 42, 54.13900492476701: 43, 53.41553197688826: 44, 62.35871886423703: 45, 67.89699520379065: 46, 66.35469640128008: 47}



The green bars represent the observation value for each respective month. This histogram is uniform with slight troughs roughly every five months.

In []: df_Solar_Quad.describe(include = 'all') # Description Tables

	Price_Quad	Solar_Quad	Dates	First Difference	Seasonal Difference
count	48.000000	48.000000	48	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN
freq	NaN	NaN	1	NaN	NaN
mean	24.500000	57.859869	NaN	0.160300	0.142011
std	4.174498	3.947656	NaN	3.944534	3.414956
min	17.075344	53.348304	NaN	-6.928212	-7.178167
25%	21.390791	54.396539	NaN	-1.984640	-1.627892
50%	24.615282	57.120551	NaN	-0.448541	-0.429441
75%	27.214266	60.710187	NaN	2.794111	1.540547
max	36.035067	67.896995	NaN	8.943187	10.233454

In []: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot

```

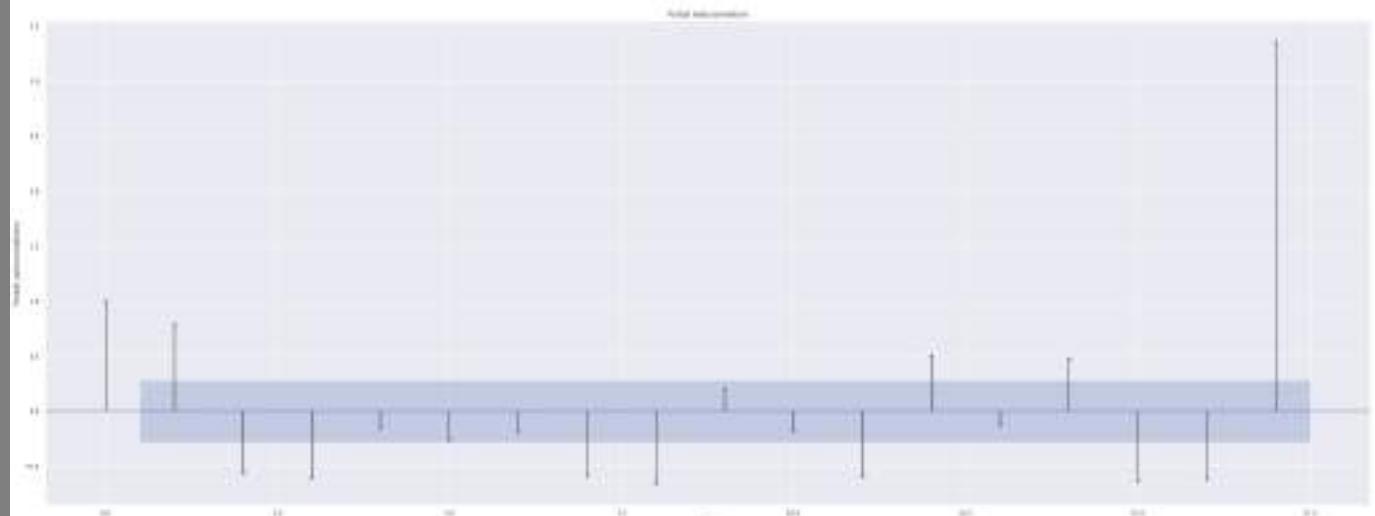
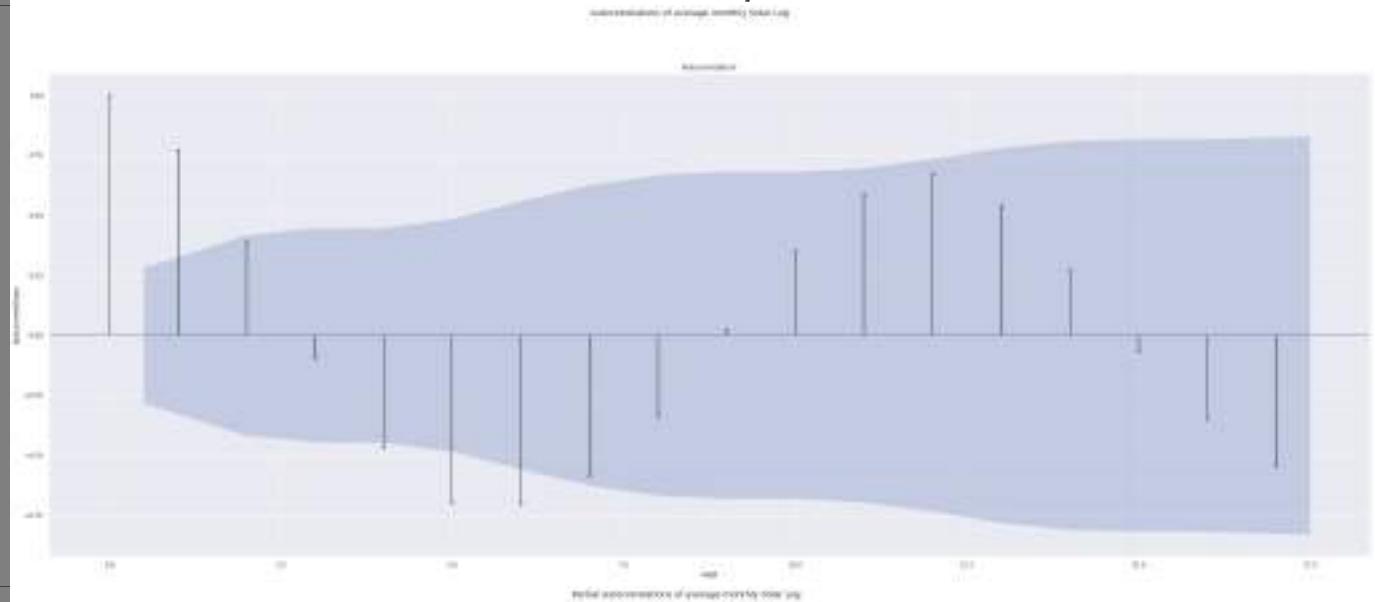
plot_acf(Solar_Logpred)
plt.suptitle(" Autocorrelations of average monthly Solar Log")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show()

from statsmodels.graphics.tsaplots import plot_pacf #Partial autocorrelation Plot
plot_pacf(Solar_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Solar Log")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')

```

```
plt.show  
Solar_Log_Autocorrelations = sm.tsa.acf(Solar_Logpred, fft=False) #Autocorrelations  
print(Solar_Log_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/regression/linear_model.py:1434: RuntimeWarning: invalid value encountered in sqrt  
    return rho, np.sqrt(sigmasq)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
  FutureWarning,  
[ 1.          0.77030015  0.38989331 -0.09095154 -0.46743581 -0.69629739  
 -0.7028239 -0.58499881 -0.33691503  0.02290571  0.35150157  0.58539428  
  0.67078862  0.5369131   0.27239882 -0.06330717 -0.34742861 -0.54609372  
 -0.53851649 -0.43904541 -0.23343633 -0.00860393  0.24493205  0.40076597  
  0.47945209  0.39202109  0.18965779 -0.04248235 -0.2125484  -0.29919937  
 -0.3054867 -0.21832683 -0.11280112  0.01749715  0.15097483  0.22488667  
  0.23730704  0.16974959  0.07333613 -0.06626634 -0.14263412]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(Solar_ypred)  
plt.suptitle(" Autocorrelations of average monthly Quadratic Solar")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(Solar_ypred)
```

```

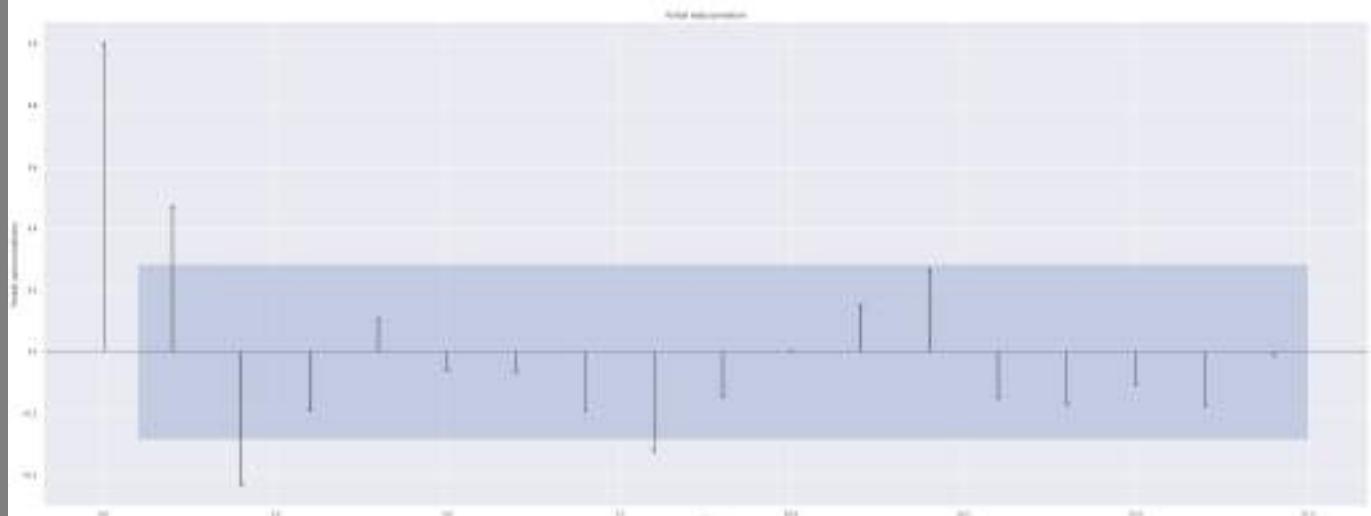
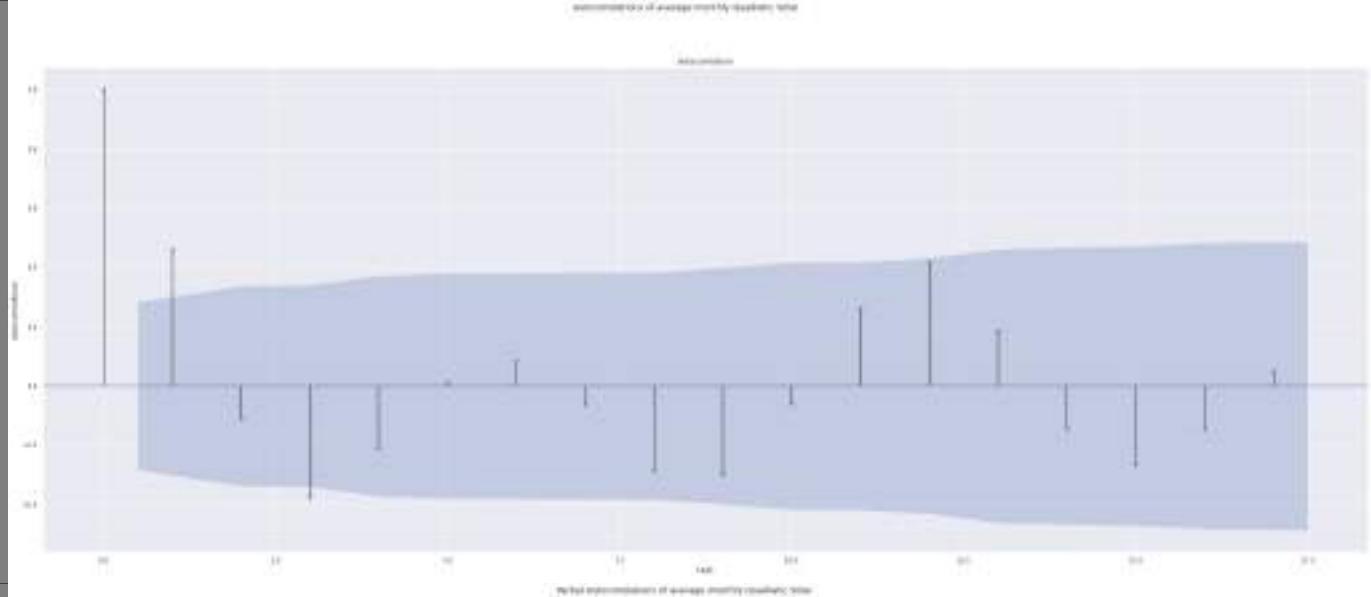
plt.suptitle("Partial Autocorrelations of average monthly Quadratic Solar")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
Solar_Quad_Autocorrelations = sm.tsa.acf(Solar_ypred, fft=False) #Autocorrelations
print(Solar_Quad_Autocorrelations)

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,

```
[ 1.          0.46069643 -0.10929335 -0.37478597 -0.20765765  0.01129959
 0.08543004 -0.06046509 -0.28482783 -0.3000041  -0.05918749  0.26103222
 0.41639937  0.18489838 -0.14385657 -0.26644941  -0.14478295  0.04838848
 0.03187391 -0.04601525 -0.22061637 -0.19580216  -0.11382745  0.20540882
 0.42023007  0.27140384 -0.02861416 -0.17650116  -0.1047224  -0.01909878
 0.04122136 -0.03736865 -0.15409147 -0.11186347  0.02087997  0.15383629
 0.1999627   0.06772278 -0.05560958 -0.11566934  0.01182077]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

In []:

```

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(predictionssolar)
plt.suptitle(" Autocorrelations of average monthly Linear Solar")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(predictionssolar)

```

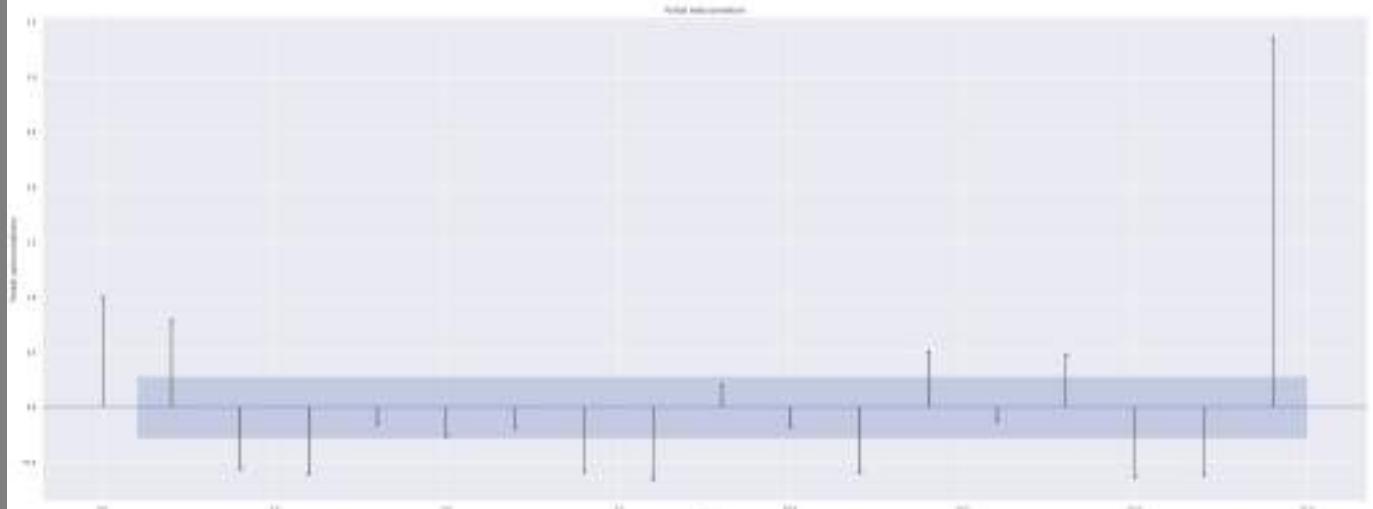
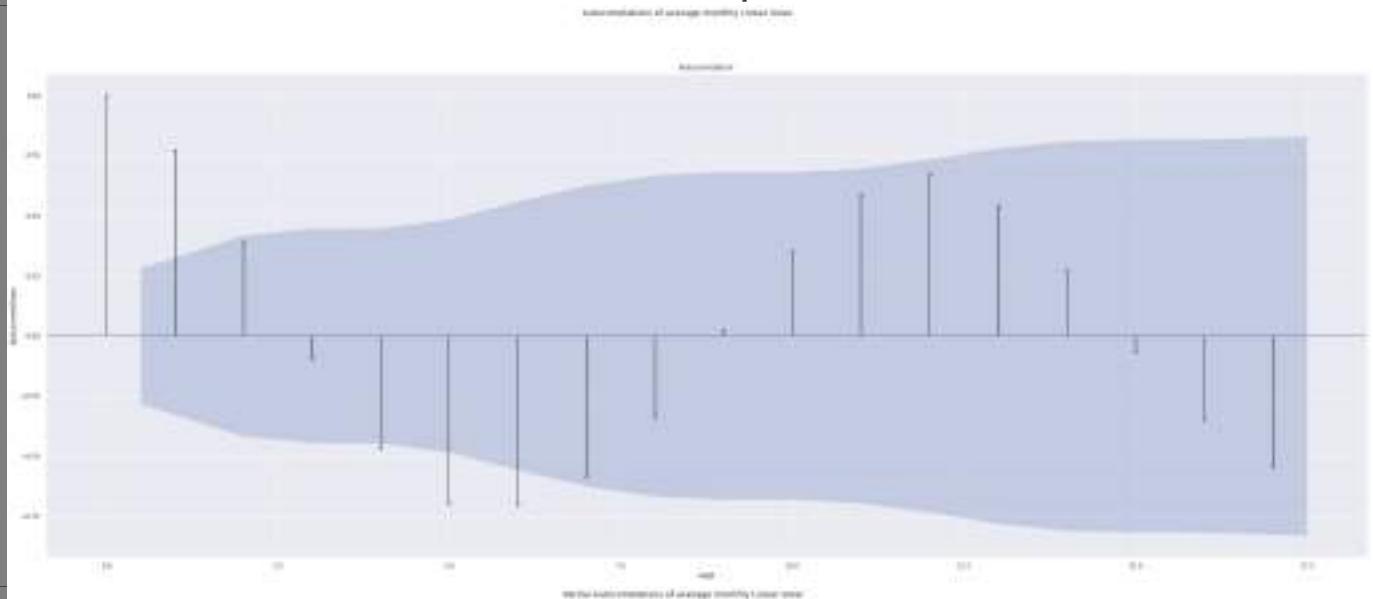
```

plt.suptitle("Partial Autocorrelations of average monthly Linear Solar")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')
plt.show
Solar_Pred_Autocorrelations = sm.tsa.acf(predictionssolar, fft=False) #Autocorrelations
print(Solar_Pred_Autocorrelations)

```

/usr/local/lib/python3.7/dist-packages/statsmodels/regression/linear_model.py:1434: RuntimeWarning: invalid value encountered in sqrt
 return rho, np.sqrt(sigmasq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
[ 1.          0.77030015  0.38989331 -0.09095154 -0.46743581 -0.69629739
 -0.7028239 -0.58499881 -0.33691503  0.02290571  0.35150157  0.58539428
 0.67078862  0.5369131   0.27239882 -0.06330717 -0.34742861 -0.54609372
-0.53851649 -0.43904541 -0.23343633 -0.00860393  0.24493205  0.40076597
 0.47945209  0.39202109  0.18965779 -0.04248235 -0.2125484  -0.29919937
-0.3054867 -0.21832683 -0.11280112  0.01749715  0.15097483  0.22488667
 0.23730704  0.16974959  0.07333613 -0.06626634 -0.14263412]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resource analyzed was fossil gas.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy

per EUR/MWh. This is the linear model used for the average monthly fossil gas outputs versus the average monthly prices of energy per EUR/MWh.

In []:

```
In [ ]: modelfossilfuelgas = stats.linregress([4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431, 64.9490188172043, 56.38385416666666, 55.522462987886975, 58.35408333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.726791666666668, 61.901760752688226, 45.57872311827956, 36.75208333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.40559722222224, 60.182429530201404, 62.58105555555558, 67.5951344086021, 79.49208333333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.2582222222224, 55.252580645161316, 54.08432795698931, 55.81655555555558, 63.92528859060403, 65.43065277777781, 65.15127688172035, 56.51197580645163, 60.877098214285674, 48.279717362045766, 50.40073611111113, 61.633763440860214, 64.34813888888884, 67.78344086021498, 70.36391129032262, 76.9140416666666, 70.36221476510062, 67.0426075268817, 66.62351388888881]))
```

In []:

```
#Dataframes analyzed by resource
dfFossilGas = {"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405915
    "Fossil_Gas": [4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915
print(dfFossilGas)
df_FossilGas= pd.DataFrame.from_dict(dfFossilGas, orient = "columns")
print(df_FossilGas)
df_FossilGas["Ratio"] = df_FossilGas["Fossil_Gas"]/df_FossilGas["Price"]
pdToListFossilGas = list(df_FossilGas["Ratio"])
print(pdToListFossilGas)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.08432795698924, 55.81655555555556, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Fossil_Gas': [4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915, 4934.930458970793, 6529.490591397849, 4919.491935483871, 5076.581944444444, 5231.5, 5387.741666666667, 5062.588156123822, 5271.1196236559135, 4450.360632183908, 4336.594885598924, 4263.4819444445, 4508.034946236559, 4994.327777777778, 5475.64333781965, 4869.987903225807, 4643.55, 6222.820134228188, 6439.015277777778, 6176.176075268817, 6412.591397849463, 5561.144345238095, 5337.418573351279, 5169.840277777777, 5493.271505376344, 7194.280555555555, 7366.073924731183, 6701.935483870968, 7092.451388888889, 6961.8, 8534.818055555555, 5835.915322580645, 5687.715053763441, 5468.040178571428, 4936.013458950202, 4745.273611111111, 5777.162634408603, 5807.555555555556, 5865.691790040377, 6119.346774193548, 5842.0625, 6127.481879194631, 6463.5362903225805, 6945.804166666667], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	Fossil_Gas	Dates
0	64.949019	4850.009550	2015-01
1	56.383854	4674.135417	2015-02
2	55.522463	4614.752355	2015-03
3	58.354083	4952.123955	2015-04
4	57.294059	4415.349462	2015-05
5	65.974903	4934.930459	2015-06
6	71.072043	6529.490591	2015-07
7	63.998065	4919.491935	2015-08
8	60.254792	5076.581944	2015-09
9	59.406765	5231.500000	2015-10
10	60.726792	5387.741667	2015-11
11	61.901761	5062.588156	2015-12
12	45.578723	5271.119624	2016-01
13	36.752083	4450.360632	2016-02
14	36.818008	4336.594886	2016-03
15	32.618667	4263.481944	2016-04
16	34.691371	4508.034946	2016-05
17	46.266319	4994.327778	2016-06
18	47.502016	5475.643338	2016-07
19	47.602339	4869.987903	2016-08
20	50.405597	4643.550000	2016-09
21	60.182430	6222.820134	2016-10
22	62.581056	6439.015278	2016-11
23	67.595134	6176.176075	2016-12
24	79.492083	6412.591398	2017-01
25	59.837798	5561.144345	2017-02
26	50.959892	5337.418573	2017-03
27	51.717917	5169.840278	2017-04
28	53.772621	5493.271505	2017-05
29	56.258222	7194.280556	2017-06
30	55.252581	7366.073925	2017-07
31	54.084328	6701.935484	2017-08
32	55.816556	7092.451389	2017-09
33	63.925289	6961.800000	2017-10
34	65.430653	8534.818056	2017-11
35	65.151277	5835.915323	2017-12
36	56.511976	5687.715054	2018-01
37	60.877098	5468.040179	2018-02
38	48.279717	4936.013459	2018-03
39	50.400736	4745.273611	2018-04
40	61.633763	5777.162634	2018-05
41	64.348139	5807.555556	2018-06
42	67.783441	5865.691790	2018-07
43	70.363911	6119.346774	2018-08
44	76.914042	5842.062500	2018-09
45	70.362215	6127.481879	2018-10
46	67.042608	6463.536290	2018-11
47	66.623514	6945.804167	2018-12

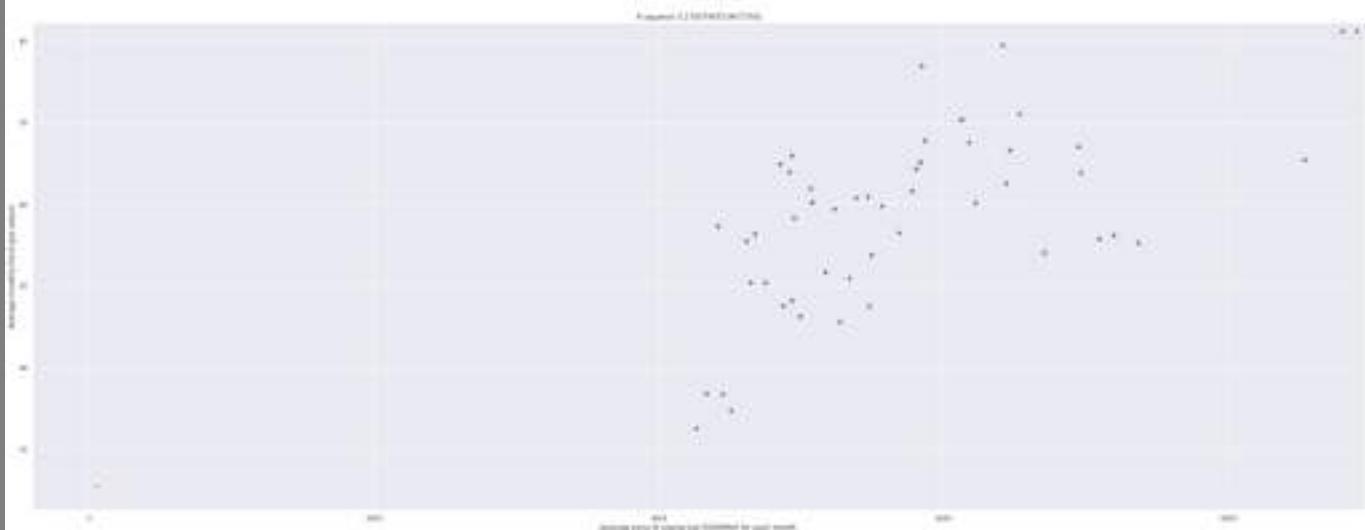
[74.67410036547997, 82.89847307795338, 83.11505122391748, 84.8633664099214, 77.06469970286285, 74.80011718384856, 91.87143516347186, 76.86938617095277, 84.2519209514227, 88.06236109868401, 88.72099972348832, 81.78423512620311, 115.6486900692026, 121.09138390387596, 117.78461443980032, 130.7068123909963, 129.94686633827166, 107.94737592591203, 115.27180915744435, 102.30564369804277, 92.12369768238372, 103.39928418983801, 102.89080650072484, 91.3701278842762, 80.66956014927435, 92.93698241774639, 104.73763443132118, 99.9622686099004, 102.15740662281915, 12.7.87962845924743, 133.31637796317597, 123.91640493713274, 127.06716346603655, 108.90525726970706, 130.4406679930639, 89.57484184347635, 100.64618999065522, 89.82097272974603, 102.23782840184066, 94.15087907942261, 93.73373151149524, 90.25211382699923, 86.53576324248239, 86.96712081488805, 75.95573413393849, 87.0848352294026, 96.40938097061529, 104.25454559857808]

In []:

```
In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(fossil_gas,Price_Actual)
print("slope: %f      intercept: %f" % (slope, intercept))

#OLS Linear Scatterplot
plt.plot(fossil_gas,Price_Actual, "o")
plt.title(f"R squared: {modelFossilFuelGas.rvalue**2}")
f = lambda x: 0.005835 *x + 25.066272
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.legend("#")
plt.suptitle("Average fossil gas monthly outputs versus predicted linear average monthly prices of energy per EUR/MWH for each month ")
plt.xlabel('Average price of energy per EUR/MWH for each month ')
plt.ylabel('Average monthly fossil gas output')
plt.show()
```

slope: 0.005835 intercept: 25.066272



There is a slightly moderate yet positive correlation between the outputs and their respective the average monthly prices of energy per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

```
In [ ]: #Linear OLS regression
fossil_gas1 = fossil_gas
fossil_gas1 = sm.add_constant(fossil_gas1)
modelFossilGasreg = sm.OLS(Price_Actual,fossil_gas1).fit()
predictionsFossilGas = modelFossilGasreg.predict(fossil_gas1)
modelFossilGasreg.summary()
#OLS Linear Summary Table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.279			
Model:	OLS	Adj. R-squared:	0.263			
Method:	Least Squares	F-statistic:	17.80			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.000114			
Time:	05:26:53	Log-Likelihood:	-171.79			
No. Observations:	48	AIC:	347.6			
Df Residuals:	46	BIC:	351.3			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	25.0663	7.876	3.183	0.003	9.212	40.920
x1	0.0058	0.001	4.220	0.000	0.003	0.009
Omnibus:	1.853	Durbin-Watson:	0.531			
Prob(Omnibus):	0.396	Jarque-Bera (JB):	1.320			
Skew:	-0.160	Prob(JB):	0.517			
Kurtosis:	2.254	Cond. No.	3.51e+04			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.51e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
#Linear OLS regression residuals
influenceFossilGasreg = modelFossilGasreg.get_influence()

standardized_residualsFossilGas = influenceFossilGasreg.resid_studentized_internal

print(standardized_residualsFossilGas)
```

```
[ 1.33110265  0.46636491  0.40756078  0.50365537  0.75096313  1.38989208
 0.91112347  1.17368259  0.63718895  0.43577377  0.48185974  0.83529676
-1.17082789 -1.65811851 -1.57908943 -2.02369081 -1.93312455 -0.91077545
-1.08607276 -0.67586965 -0.20297015 -0.13715476 -0.00686655  0.74310232
 1.95551448  0.26467844 -0.5998555 -0.40222494 -0.38219446 -1.27084604
-1.51890711 -1.16839671 -1.24775252 -0.20617141 -1.21291296  0.6883476
-0.19905142  0.44535313 -0.6415435 -0.27142232  0.32585172  0.61546865
 0.96915522  1.0973707  2.02710947  1.09185567  0.16853664  0.44200252]
```

In []:

```
print(predictionsFossilGas)
#Linear OLS Predicted Values
```

```
[53.36819441 52.34189209 51.99536601 53.96407653 50.83176311 53.86374497
63.16869921 53.77365445 54.69034321 55.59435767 56.50609598 54.60868335
55.82555546 51.03606857 50.37219579 49.94554987 51.37262336 54.21035425
57.01904059 53.48477682 52.16341284 61.37914326 62.64073627 61.10695482
62.4865414 57.51797628 56.21243877 55.23454628 57.12190864 67.04803876
68.05052804 64.17499041 66.45382114 65.69141327 74.87066008 59.12138484
58.2565716 56.97467284 53.87006475 52.7570145 58.77853727 58.95589322
59.29514352 60.77533105 59.1572563 60.82280294 65.5980706 62.78382708]
```

In []:

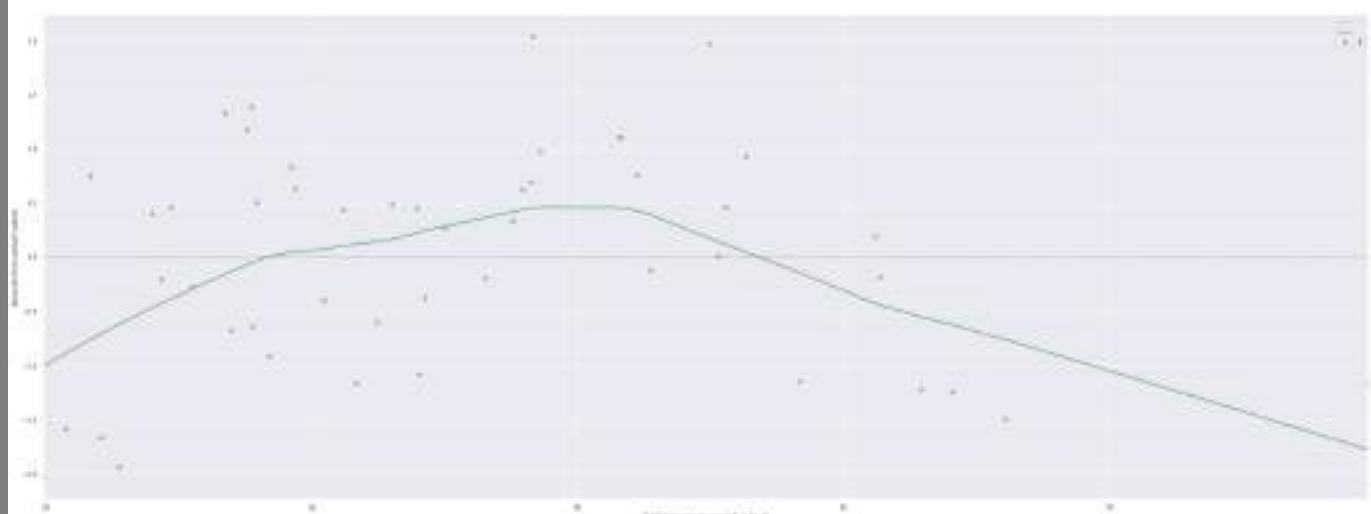
```
#Predicted OLS linear values versus residual values
sns.residplot(x = predictionsFossilGas, y = standardized_residualsFossilGas, lowess = True, color="g")

plt.suptitle("Fossil gas residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#..)
```

Out[]:

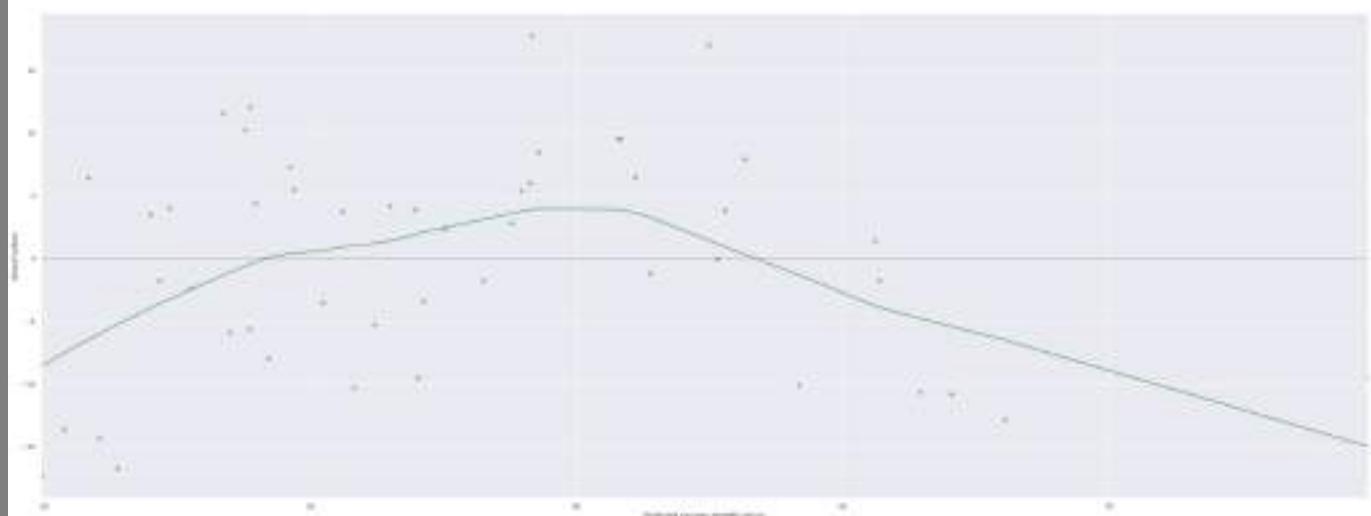
<matplotlib.legend.Legend at 0x7ff605c56f10>



As one can observe this residual plot, one may notice that the lowess line has a arching hump and that the residuals are spread out in a pattern; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly linear fossil gas energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsFossilGas, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff605b91490>
```



As one can observe this residual plot, one may notice the positive slope in the fitted model. In addition, the observations are spread out in a distinct pattern, indicating bias and homoscedasticity. It would be reasonable to assume that this model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

This is the average monthly logarithmic outputs of fossil gas versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: print(fossil_gas)
```

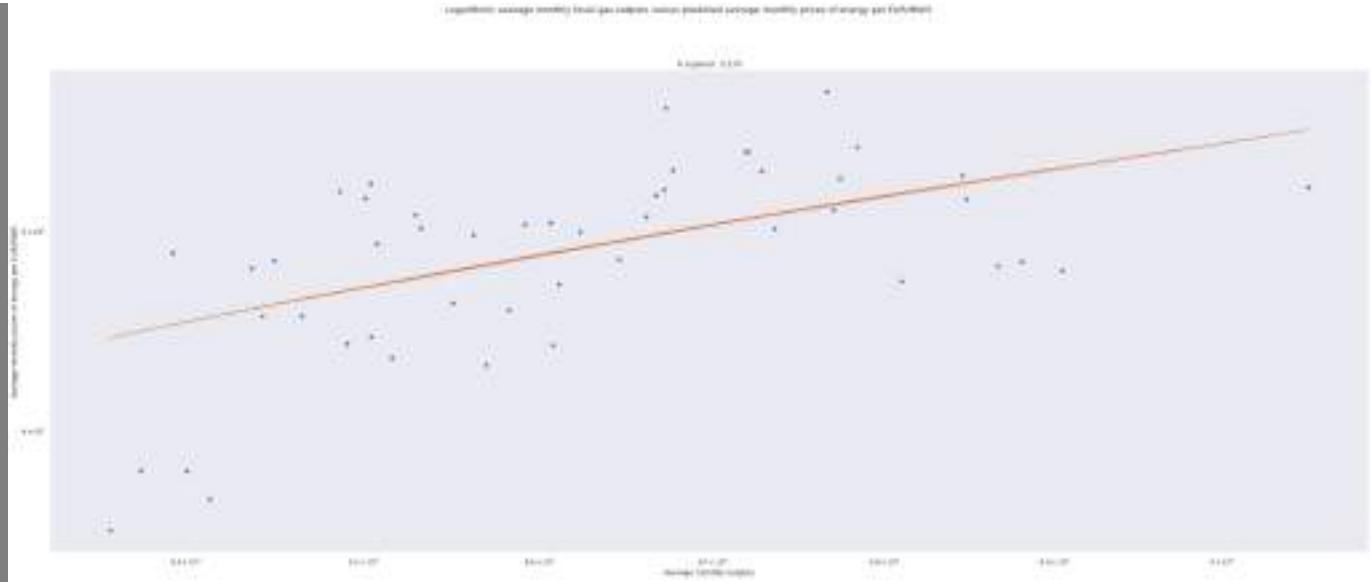
```
[4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915, 4934.930458970  
793, 6529.490591397849, 4919.491935483871, 5076.581944444444, 5231.5, 5387.741666666667, 5062.588156123822, 5271  
.1196236559135, 4450.360632183908, 4336.594885598924, 4263.481944444445, 4508.034946236559, 4994.327777777778, 5  
475.64333781965, 4869.987903225807, 4643.55, 6222.820134228188, 6439.015277777778, 6176.176075268817, 6412.59139  
7849463, 5561.144345238095, 5337.418573351279, 5169.840277777778, 5493.271505376344, 7194.280555555555, 7366.073  
924731183, 6701.935483870968, 7092.451388888889, 6961.8, 8534.818055555555, 5835.915322580645, 5687.715053763441  
, 5468.040178571428, 4936.013458950202, 4745.273611111111, 5777.162634408603, 5807.555555555556, 5865.6917900403  
77, 6119.346774193548, 5842.0625, 6127.481879194631, 6945.804166666667, 6463.5362903225805]
```

```
In [ ]: fossil_gas1 = fossil_gas  
fossil_gas1 = sm.add_constant(fossil_gas1)
```

```
In [ ]: #Logarithmic OLS regressions  
Logpricevalues = ((np.log(Price_Actual)))  
LogFossilGasvalues = ((np.log(fossil_gas)))  
Log = np.polyfit(np.log(Price_Actual), fossil_gas1, 1)  
lin2 = LinearRegression()  
lin2.fit(np.log(fossil_gas1), Price_Actual)  
FossilGas_Log = sm.OLS(Price_Actual, fossil_gas1).fit()  
  
FossilGas_Logpred = FossilGas_Log.predict(fossil_gas1)  
#OLS Logarithmic summary table  
FossilGas_Log.summary()  
#Log  
Log = np.polyfit(np.log(fossil_gas), Price_Actual, 1)  
print(Log)  
  
y = 36.53286955 * LogFossilGasvalues -257.09729841  
  
#OLS Logarithmic Scatterplots  
plt.suptitle("Logarithmic average monthly fossil gas outputs versus predicted average monthly prices of energy per EUR/MWH")  
plt.title("R squared : 0.279")  
plt.ylabel("Average monthly prices of energy per EUR/MWH")  
plt.xlabel("Average monthly outputs")  
plt.yscale("log")  
plt.xscale("log")  
plt.plot(LogFossilGasvalues, Price_Actual, "o")  
plt.plot(LogFossilGasvalues, y)
```

```
[ 36.53286955 -257.09729841]
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x7ff605ad2650>]
```



The blue dots represent the observations and the orange line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: FossilGas_Log.summary() #OLS Logarithmic summary table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.279			
Model:	OLS	Adj. R-squared:	0.263			
Method:	Least Squares	F-statistic:	17.80			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.000114			
Time:	05:26:56	Log-Likelihood:	-171.79			
No. Observations:	48	AIC:	347.6			
Df Residuals:	46	BIC:	351.3			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	25.0663	7.876	3.183	0.003	9.212	40.920
x1	0.0058	0.001	4.220	0.000	0.003	0.009
Omnibus:	1.853	Durbin-Watson:	0.531			
Prob(Omnibus):	0.396	Jarque-Bera (JB):	1.320			
Skew:	-0.160	Prob(JB):	0.517			
Kurtosis:	2.254	Cond. No.	3.51e+04			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 3.51e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceFossilGasLog = FossilGas_Log.get_influence()
#Logarithmic OLS regression residuals
```

```
standardized_residualsFossilGasLog = influenceFossilGasLog.resid_studentized_internal
```

```
print(standardized_residualsFossilGasLog)
```

```
print(FossilGas_Logpred) # OLS logarithmic predicted values
```

```
[ 1.33110265  0.46636491  0.40756078  0.50365537  0.75096313  1.38989208
 0.91112347  1.17368259  0.63718895  0.43577377  0.48185974  0.83529676
-1.17082789 -1.65811851 -1.57908943 -2.02369081 -1.93312455 -0.91077545
-1.08607276 -0.67586965 -0.20297015 -0.13715476 -0.00686655  0.74310232
 1.95551448  0.26467844 -0.5998555  -0.40222494 -0.38219446 -1.27084604
-1.51890711 -1.16839671 -1.24775252 -0.20617141 -1.21291296  0.6883476
-0.19905142  0.44535313 -0.6415435  -0.27142232  0.32585172  0.61546865
 0.96915522  1.0973707   2.02710947  1.09185567  0.16853664  0.44200252]
[53.36819441 52.34189209 51.99536601 53.96407653 50.83176311 53.86374497
63.16869921 53.77365445 54.69034321 55.59435767 56.50609598 54.60868335
55.82555546 51.03606857 50.37219579 49.94554987 51.37262336 54.21035425
57.01904059 53.48477682 52.16341284 61.37914326 62.64073627 61.10695482
62.4865414  57.51797628 56.21243877 55.23454628 57.12190864 67.04803876
68.05052804 64.17499041 66.45382114 65.69141327 74.87066008 59.12138484
58.2565716  56.97467284 53.87006475 52.7570145  58.77853727 58.95589322
59.29514352 60.77533105 59.1572563  60.82280294 65.5980706  62.78382708]
```

In []:

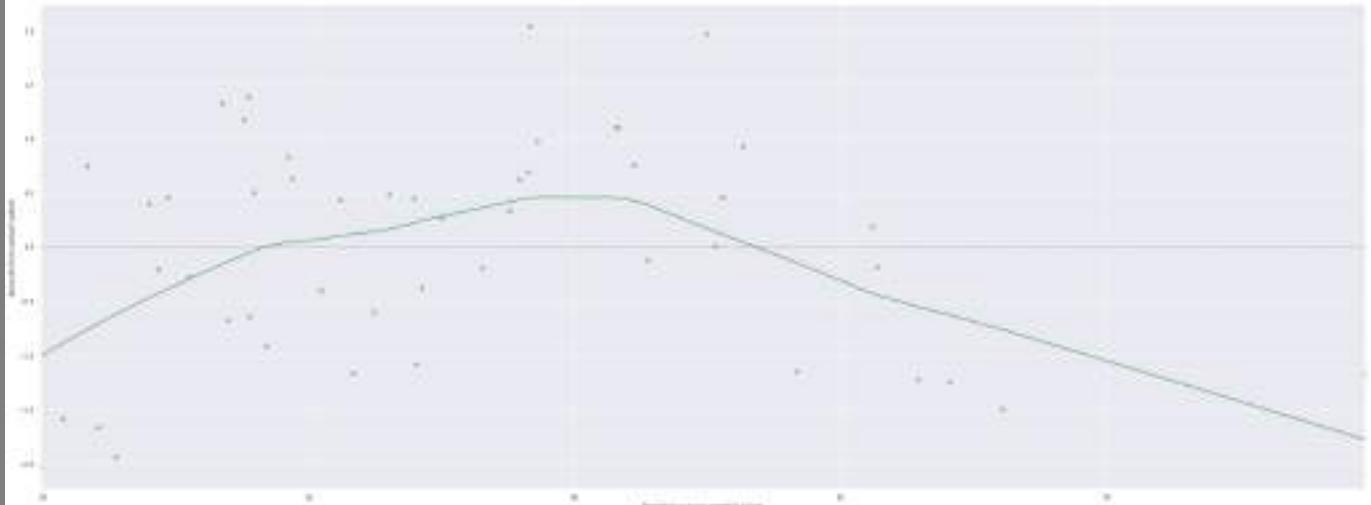
```
FossilGasLogRatioPredict = FossilGas_Logpred/Logpred
```

In []:

```
# OLS Logarithmic average monthly predictions versus residuals
```

```
plt.suptitle("Residuals from logarithmic fossil gas model versus predicted average monthly logarithmic fossil ga")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = FossilGas_Logpred, y =standardized_residualsFossilGasLog, lowess = True, color="g")
```

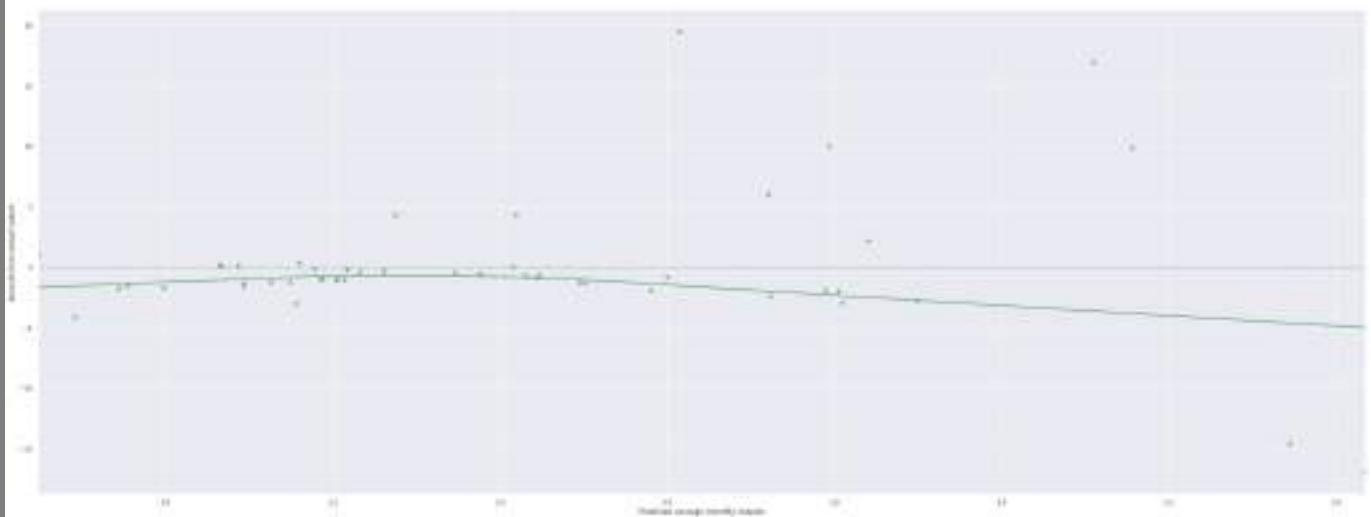
Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff609f62d50>



As one can observe this residual plot, one may notice that the lowess line has a arching hump and that the residuals are spread out in a pattern; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
  
plt.suptitle("Predicted average monthly logarithmic fossil gas output to price of energy per EUR/MWH ratio versus Predicted average monthly outputs")  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Amount from actual values")  
plt.legend(..#)  
sns.residplot(x = FossilGasLogRatioPredict, y = standardized_residualsFossilGasLog/standardized_residualsPriceLog)
```

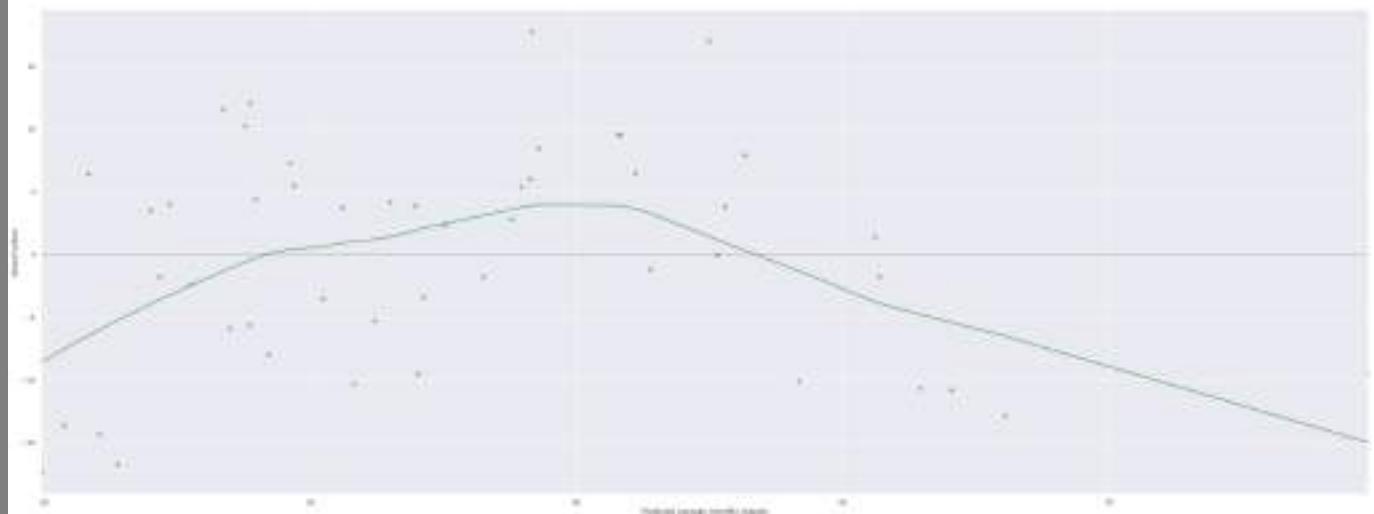
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60595b750>
```



With the exception of few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly logarithmic energy prices per EUR/MWH versus actual values")  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Actual values")  
plt.legend(..#)  
sns.residplot(x = FossilGas_Logpred, y = Price_Actual, lowess = True, color="g")  
# OLS predicted logarithmic average monthly values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff605b18c90>
```



As one can observe this residual plot, one may notice that the lowess line has a arching hump and that the residuals are spread out in a pattern; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: influenceFossilGasLog = FossilGas_Log.get_influence()  
#Logarithmic OLS regression residuals  
  
standardized_residualsFossilGasLog = influenceFossilGasLog.resid_studentized_internal  
  
print(standardized_residualsFossilGasLog)  
  
[ 1.33110265  0.46636491  0.40756078  0.50365537  0.75096313  1.38989208  
 0.91112347  1.17368259  0.63718895  0.43577377  0.48185974  0.83529676  
 -1.17082789 -1.65811851 -1.57908943 -2.02369081 -1.93312455 -0.91077545  
 -1.08607276 -0.67586965 -0.20297015 -0.13715476 -0.00686655  0.74310232  
  1.95551448  0.26467844 -0.5998555 -0.40222494 -0.38219446 -1.27084604  
 -1.51890711 -1.16839671 -1.24775252 -0.20617141 -1.21291296  0.6883476  
 -0.19905142  0.44535313 -0.6415435 -0.27142232  0.32585172  0.61546865  
  0.96915522  1.0973707  2.02710947  1.09185567  0.16853664  0.44200252]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly fossil gas outputs versus the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Quadratic OLS regression  
from sklearn.preprocessing import PolynomialFeatures  
polynomial_features = PolynomialFeatures(degree=2)  
  
modelFossilGasquad = np.poly1d(np.polyfit(fossil_gas,Price_Actual,2))  
print(modelFossilGasquad)  
  
fossil_gas1 = sm.add_constant(fossil_gas1)  
fossil_gas2 = polynomial_features.fit_transform(fossil_gas1)  
from sklearn.preprocessing import PolynomialFeatures  
  
poly = PolynomialFeatures(degree = 3)  
X_poly = poly.fit_transform(fossil_gas2)  
  
FossilGas_Quad = poly.fit(X_poly, fossil_gas)  
lin2 = LinearRegression()  
lin2.fit(X_poly, fossil_gas)  
FossilGas_Quad = sm.OLS(Price_Actual, fossil_gas2).fit()  
  
# OLS Predicted Quadratic values  
FossilGas_ypred = FossilGas_Quad.predict(fossil_gas2)  
#OLS Quadratic Summary Table  
FossilGas_Quad.summary()  
  
##OLS Quadratic Scatterplot  
plt.scatter(fossil_gas,Price_Actual, color = 'blue')  
polyline = np.linspace(start = 2000, stop =10000 , num = 100)
```

```

plt.plot(polyline, modelFossilGasquad(polyline))
plt.title("R squared : 0.439")
plt.suptitle('Quadratic for predicted average monthly prices of energy per EUR/MWH versus average monthly fossil gas outputs')
plt.xlabel('Average monthly outputs')
plt.ylabel('Average monthly prices of energy per EUR/MWH')

plt.show()

```



The blue dots represent the observations and the blue line is the model of best fit. This is a moderate and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

In []: `print(FossilGas_ypred) # OLS quadratic predicted values`

```
[52.24719288 49.66127153 48.73778675 53.64627702 45.45078952 53.41596985
 65.70819512 53.20735448 55.24984163 57.08984623 58.77031803 55.0751127
 57.53263506 46.04866852 44.07361488 42.75506869 47.01429171 54.20256132
 59.63839043 52.52683452 49.18880314 64.7678809 65.50129133 64.56545272
 65.42972866 60.42930924 58.24827769 56.37822554 59.8057712 65.41850443
 64.82555648 65.93911026 65.66952866 65.88211071 55.14373649 62.61419951
 61.50337779 59.56550632 53.43053955 50.73409292 62.19276855 62.41388305
 62.81828238 64.29762447 62.65685484 64.33739204 65.89968138 65.56319729]
```

In []: `FossilGas_Quad.summary()`

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.439			
Model:	OLS	Adj. R-squared:	0.414			
Method:	Least Squares	F-statistic:	17.59			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	2.27e-06			
Time:	05:26:58	Log-Likelihood:	-165.78			
No. Observations:	48	AIC:	337.6			
Df Residuals:	45	BIC:	343.2			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x1	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x2	0.0245	0.006	4.041	0.000	0.012	0.037
x3	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x4	0.0245	0.006	4.041	0.000	0.012	0.037
x5	-3.605e-06	1.01e-06	-3.579	0.001	-5.63e-06	-1.58e-06
Omnibus:	5.062	Durbin-Watson:	0.652			
Prob(Omnibus):	0.080	Jarque-Bera (JB):	2.070			
Skew:	0.098	Prob(JB):	0.355			
Kurtosis:	2.002	Cond. No.	6.00e+24			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 1.57e-33. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

FossilGas_Quad.summary()

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.439			
Model:	OLS	Adj. R-squared:	0.414			
Method:	Least Squares	F-statistic:	17.59			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	2.27e-06			
Time:	05:26:58	Log-Likelihood:	-165.78			
No. Observations:	48	AIC:	337.6			
Df Residuals:	45	BIC:	343.2			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x1	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x2	0.0245	0.006	4.041	0.000	0.012	0.037
x3	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x4	0.0245	0.006	4.041	0.000	0.012	0.037
x5	-3.605e-06	1.01e-06	-3.579	0.001	-5.63e-06	-1.58e-06
Omnibus:	5.062	Durbin-Watson:	0.652			
Prob(Omnibus):	0.080	Jarque-Bera (JB):	2.070			
Skew:	0.098	Prob(JB):	0.355			
Kurtosis:	2.002	Cond. No.	6.00e+24			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 1.57e-33. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
influenceFossilGasQuad = FossilGas_Quad.get_influence() #Quadratic OLS residuals

standardized_residualsFossilGasQuad = influenceFossilGasQuad.resid_studentized_internal

print(standardized_residualsFossilGasQuad)
```

```
[ 1.63799933  0.87365282  0.885045   0.60553145  1.57332645  1.61594265
 0.69614225  1.38890965  0.6426209   0.29731549  0.25122355  0.87663704
-1.53407663 -1.23018905 -0.97367455 -1.37562293 -1.62132483 -1.02001512
-1.5594812  -0.63466873  0.1584256  -0.593556  -0.37867275  0.3920242
 1.82307357 -0.07606979 -0.93558714 -0.59805819 -0.77536299 -1.21170095
-1.28293013 -1.54197198 -1.29618698 -0.25608876  2.4368582  0.32724055
-0.64277348  0.16852355 -0.66273453 -0.04316281 -0.0720569  0.24941129
 0.64062254  0.7845728   1.83905808  0.77926381  0.14949675  0.13752378]
```

In []:

FossilGas_Quad.summary()

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.439			
Model:	OLS	Adj. R-squared:	0.414			
Method:	Least Squares	F-statistic:	17.59			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	2.27e-06			
Time:	05:26:58	Log-Likelihood:	-165.78			
No. Observations:	48	AIC:	337.6			
Df Residuals:	45	BIC:	343.2			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x1	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x2	0.0245	0.006	4.041	0.000	0.012	0.037
x3	-33.5947	11.953	-2.811	0.007	-57.669	-9.521
x4	0.0245	0.006	4.041	0.000	0.012	0.037
x5	-3.605e-06	1.01e-06	-3.579	0.001	-5.63e-06	-1.58e-06
Omnibus:	5.062	Durbin-Watson:	0.652			
Prob(Omnibus):	0.080	Jarque-Bera (JB):	2.070			
Skew:	0.098	Prob(JB):	0.355			
Kurtosis:	2.002	Cond. No.	6.00e+24			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 1.57e-33. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

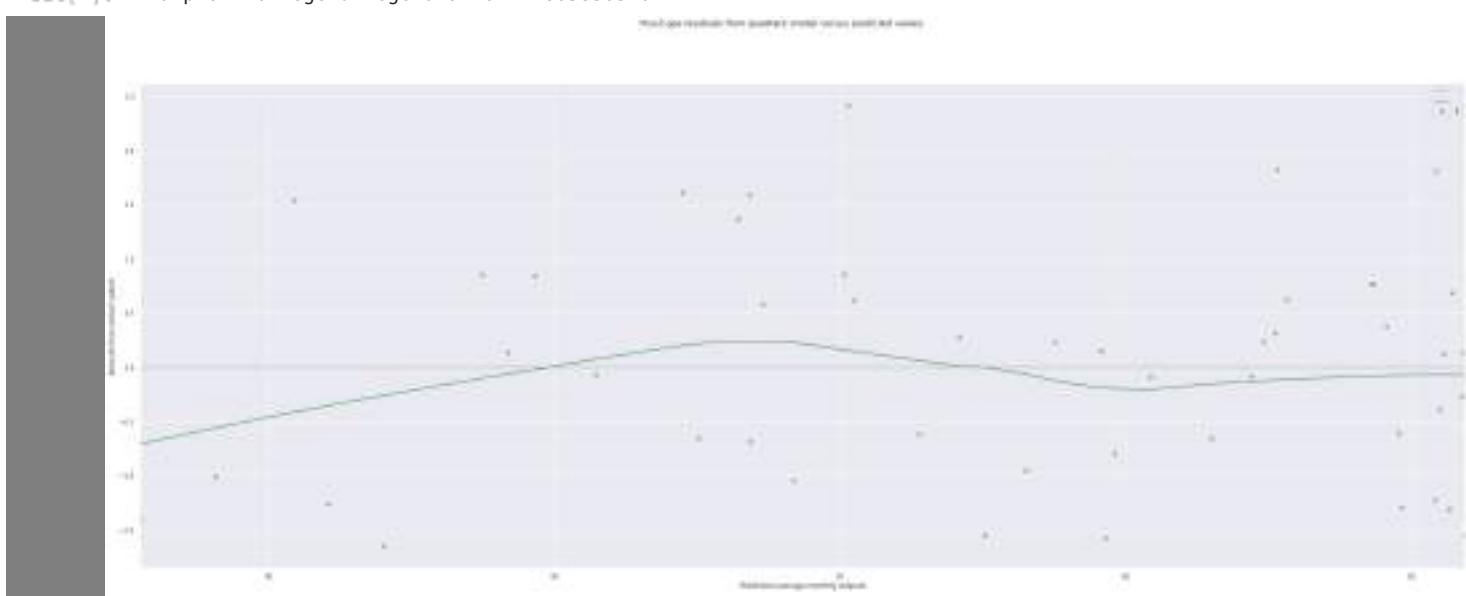
In []:

```
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = FossilGas_ypred, y =standardized_residualsFossilGasQuad, lowess = True, color="g")

plt.suptitle("Fossil gas residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#..)
```

Out[]:

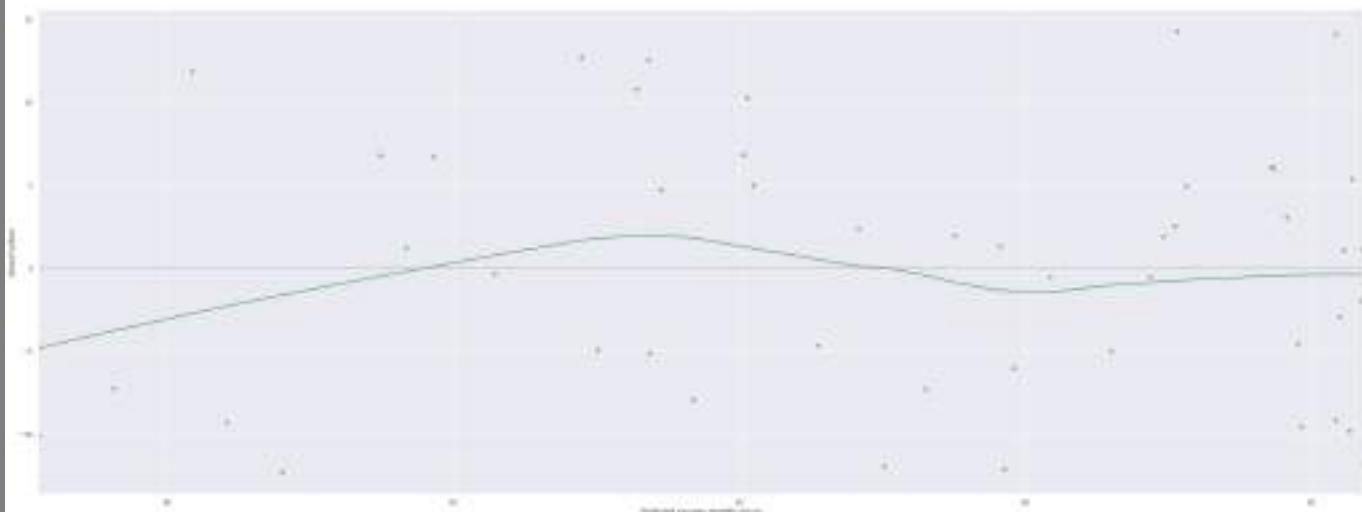


As one can observe this residual plot, one may notice an arching hump in the fitted model, which form a nonlinear pattern. However, the residuals are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent respective observations while the dotted horizontal line represents the regression model. The green line represents the

expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly quadratic fossil gas energy prices per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = FossilGas_ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60572b450>
```



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+': '+str(value))

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_FossilGas["Fossil_Gas"])

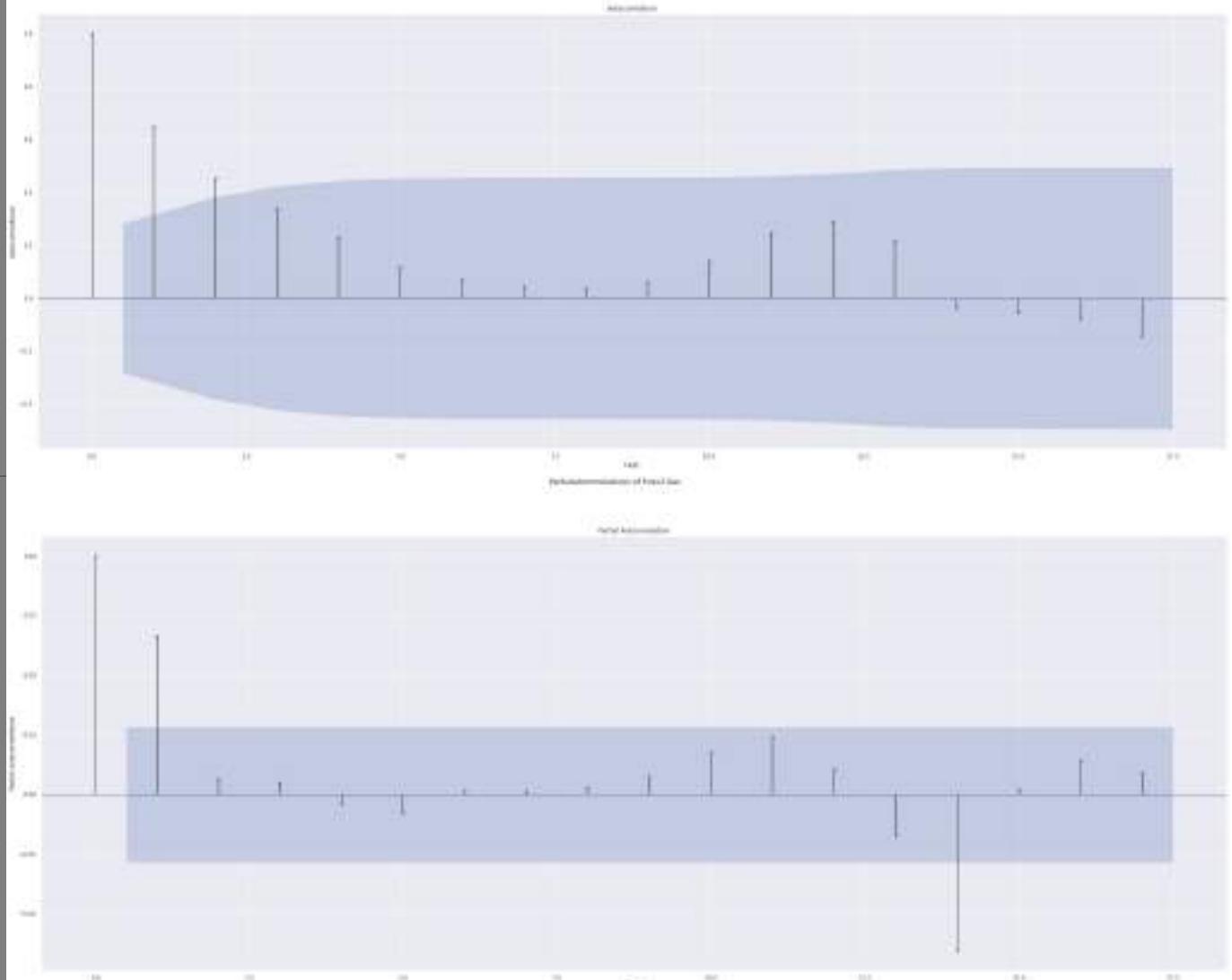
test_result=adfuller(df_FossilGas["Fossil_Gas"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilGas["Fossil_Gas"])
plt.suptitle(" Autocorrelations of average monthly Fossil Gas")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_FossilGas["Fossil_Gas"])
plt.suptitle("Partialatocorrelations of Fossil Gas")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
```

ADF Test Statistic : -2.862608705796581
p-value : 0.0498720749433969
#Lags Used : 0
Number of Observations : 47
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```
Out[ ]: Text(0.5, 0, 'Lags')
```



```
In [ ]: Fossil_Gas_Autocorrelations = sm.tsa.acf(df_FossilGas["Fossil_Gas"], fft=False) #Autocorrelations
print(Fossil_Gas_Autocorrelations)
```

```
[ 1.0000000e+00  6.47223813e-01  4.53119414e-01  3.38363318e-01
 2.26625682e-01  1.17599966e-01  6.92871354e-02  4.35895201e-02
 3.53665336e-02  5.90033215e-02  1.37459901e-01  2.46135482e-01
 2.90423026e-01  2.15319798e-01 -3.56935443e-02 -5.26843971e-02
 -7.67894566e-02 -1.41159479e-01 -1.54880171e-01 -2.03734901e-01
 -2.19703128e-01 -1.98768593e-01 -9.34050127e-02  6.06951596e-04
 -2.07043651e-02 -4.82585213e-02 -1.10729374e-01 -2.02096549e-01
 -2.07043651e-02 -4.82585213e-02 -1.10729374e-01 -2.02096549e-01
 -1.54582485e-01 -2.56277304e-01 -2.69986011e-01 -2.18350273e-01
 -2.31827168e-01 -1.85489475e-01 -8.50263961e-02 -2.08792203e-03
 -8.15967392e-03  1.29233404e-02 -1.69469707e-02 -3.49040373e-02
 -4.03639728e-02]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/ststools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation plot, indicating that the lags beforehand influenced the average monthly fossil gas outputs.

```
In [ ]: df_FossilGas['First Difference'] = df_FossilGas["Fossil_Gas"] - df_FossilGas["Fossil_Gas"].shift(1) # Seasonality
df_FossilGas['Seasonal Difference']=df_FossilGas["Fossil_Gas"] - df_FossilGas["Fossil_Gas"].shift(12)
df_FossilGas.head()
```

```
plt.suptitle("Seasonal Difference of average monthly Fossil Gas output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilGas['Seasonal Difference'].plot() # Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff605717550>



As one can observe, there are no obvious patterns depicted in the average monthly fossil gas outputs. The blue line represents the trend of the outputs.

In []: #Bell Curves

```
FossilGasResults_mean = np.mean(df_FossilGas["Ratio"])
FossilGasResults_std = np.std(df_FossilGas["Ratio"])

FossilGasResultspdf = stats.norm.pdf(df_FossilGas["Ratio"].sort_values(), FossilGasResults_mean, FossilGasResults_std)

plt.plot(df_FossilGas["Ratio"].sort_values(), FossilGasResultspdf)
plt.xlim([0,200])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.suptitle("Frequency distribution of average monthly fossil gas outputs to energy prices per EUR/MWH ratios")
plt.title(f'Skewness for data: {skew(df_FossilGas["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves

FossilGasResults_mean = np.mean(df_FossilGas["Fossil_Gas"])
FossilGasResults_std = np.std(df_FossilGas["Fossil_Gas"])

FossilGasResultspdf = stats.norm.pdf(df_FossilGas["Fossil_Gas"].sort_values(), FossilGasResults_mean, FossilGasResults_std)

plt.plot(df_FossilGas["Fossil_Gas"].sort_values(), FossilGasResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilGas["Fossil_Gas"])}')
plt.suptitle("Frequency distribution of average monthly fossil gas outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



These bell shaped curves are skewed to the right. Hence, they have an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

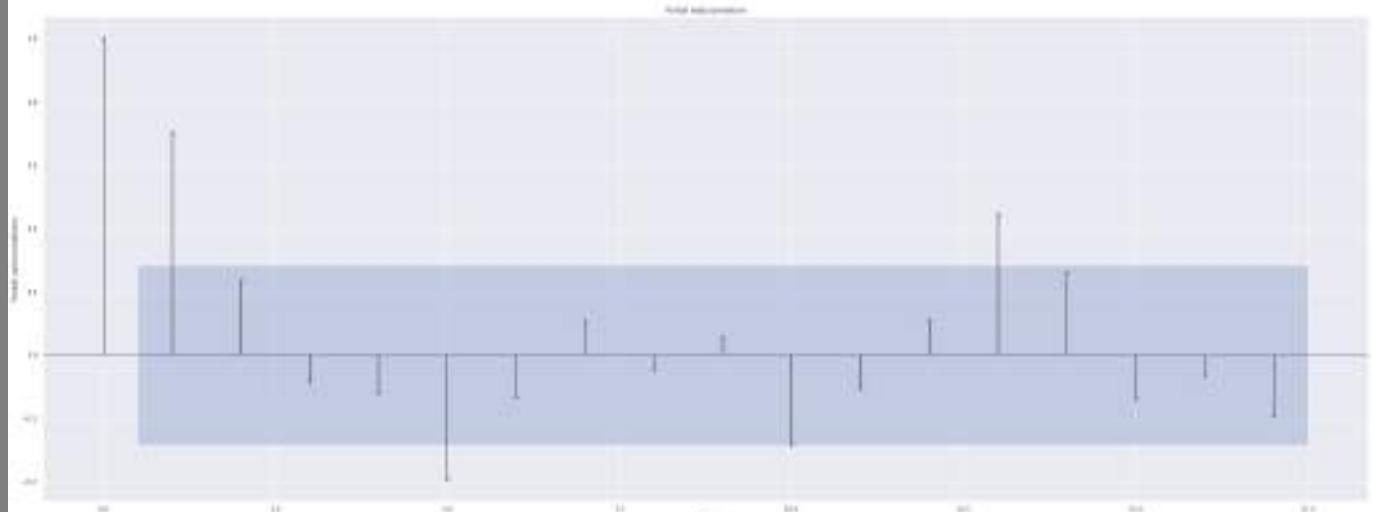
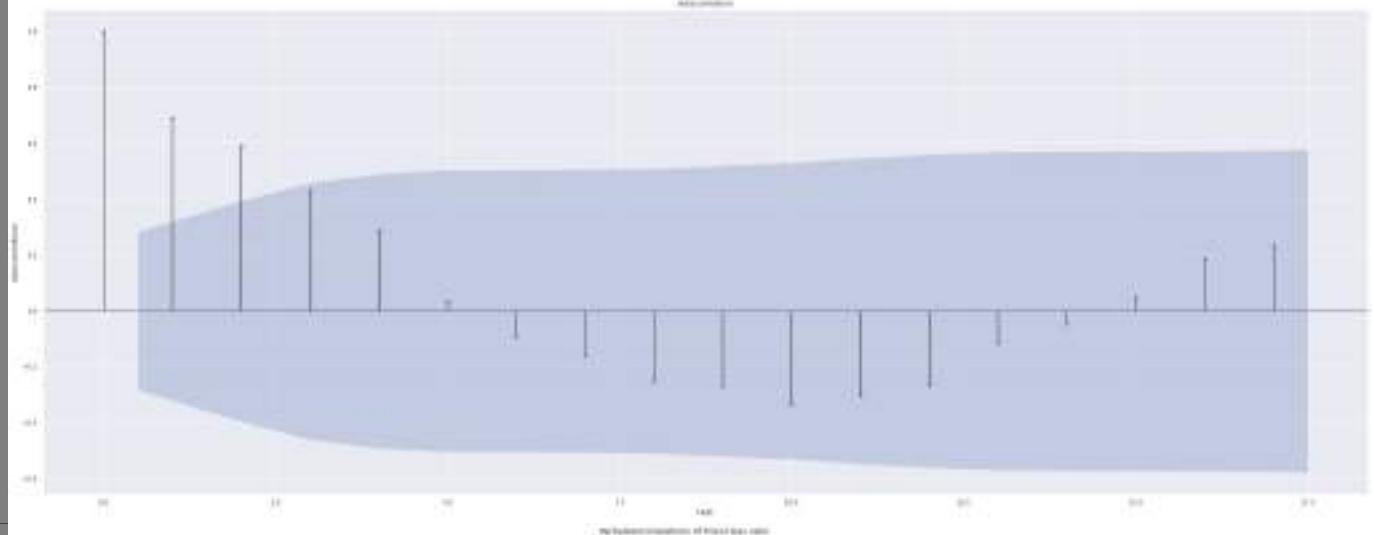
adfuller_test(df_FossilGas["Ratio"])
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilGas["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Fossil Gas ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_FossilGas["Ratio"])
plt.suptitle("Partialatocorrelations of Fossil Gas ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')

plt.show
```

```
ADF Test Statistic : -3.187221811408375
p-value : 0.02073634807351454
#Lags Used : 4
Number of Observations : 43
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
```

```
Out[ ]: <function matplotlib.pyplot.show(*args, **kw)>
```



```
In [ ]:
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly fossil gas outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: df_FossilGas['First Difference in Fossil Gas Ratio'] = df_FossilGas["Ratio"]- df_FossilGas["Ratio"].shift(1) # :
df_FossilGas['Seasonal Difference Fossil Gas Ratio']=df_FossilGas["Ratio"]- df_FossilGas["Ratio"].shift(12)
df_FossilGas.head()
```

	Price	Fossil_Gas	Dates	Ratio	First Difference	Seasonal Difference	First Difference in Fossil Gas Ratio	Seasonal Difference Fossil Gas Ratio
0	64.949019	4850.009550	2015-01	74.674100	NaN	NaN	NaN	NaN
1	56.383854	4674.135417	2015-02	82.898473	-175.874133	NaN	8.224373	NaN
2	55.522463	4614.752355	2015-03	83.115051	-59.383061	NaN	0.216578	NaN
3	58.354083	4952.123955	2015-04	84.863366	337.371600	NaN	1.748315	NaN
4	57.294059	4415.349462	2015-05	77.064700	-536.774493	NaN	-7.798667	NaN

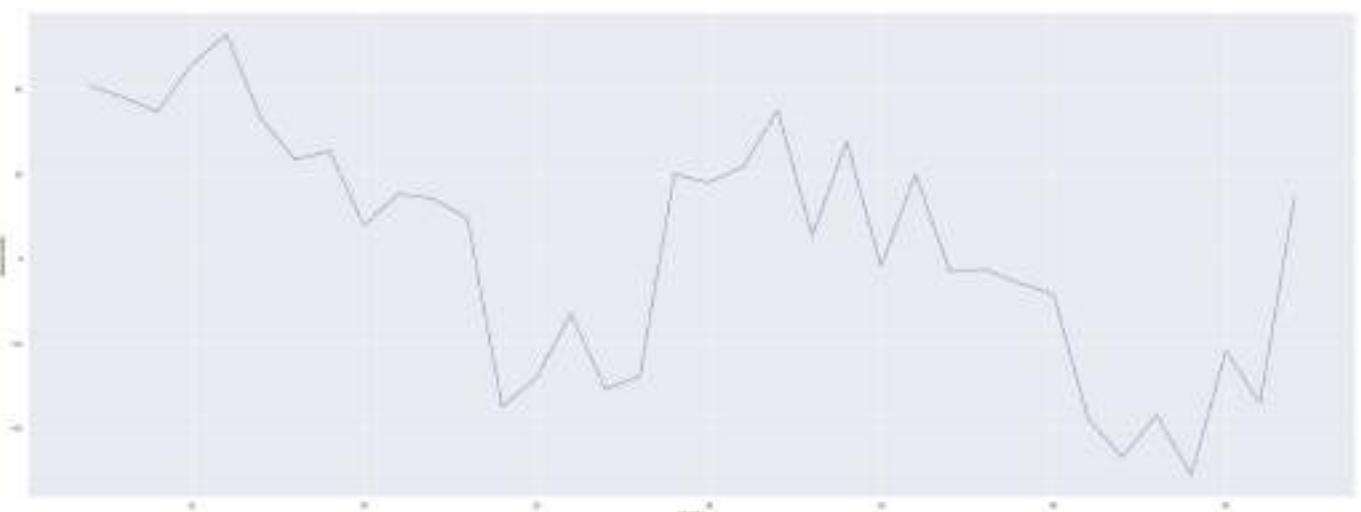
```
In [ ]: Fossil_Gas_Ratio_Autocorrelations = sm.tsa.acf(df_FossilGas["Fossil_Gas"], fft=False) #Autocorrelations
print(Fossil_Gas_Ratio_Autocorrelations)
```

```
[ 1.00000000e+00  6.47223813e-01  4.53119414e-01  3.38363318e-01
 2.26625682e-01  1.17599966e-01  6.92871354e-02  4.35895201e-02
 3.53665336e-02  5.90033215e-02  1.37459901e-01  2.46135482e-01
 2.90423026e-01  2.15319798e-01  -3.56935443e-02  -5.26843971e-02
 -7.67894566e-02  -1.41159479e-01  -1.54880171e-01  -2.03734901e-01
 -2.19703128e-01  -1.98768593e-01  -9.34050127e-02  6.06951596e-04
 2.07043651e-02  -4.82585213e-02  -1.10729374e-01  -2.02096549e-01
 -1.54582485e-01  -2.56277304e-01  -2.69986011e-01  -2.18350273e-01
 -2.31827168e-01  -1.85489475e-01  -8.50263961e-02  -2.08792203e-03
 -8.15967392e-03  1.29233404e-02  -1.69469707e-02  -3.49040373e-02
 -4.03639728e-02]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,
```

```
In [ ]: plt.suptitle("Seasonal Difference of Fossil Gas output to price of energy per EUR/MWH ratio")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilGas['Seasonal Difference Fossil Gas Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff605397ad0>
```



The blue line represents the trend of the outputs. As one can observe, there are no obvious patterns depicted between the average monthly fossil gas outputs and the average monthly prices of energy per EUR/MWH.

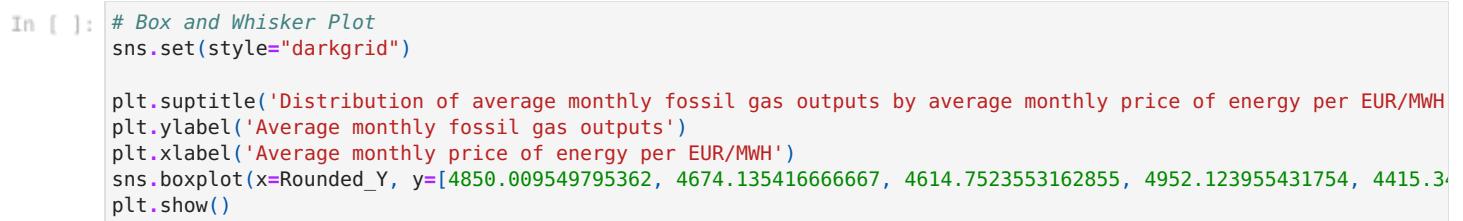
```
In [ ]: print(fossil_gas)
```

```
[4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915, 4934.930458970793, 6529.490591397849, 4919.491935483871, 5076.581944444444, 5231.5, 5387.741666666667, 5062.588156123822, 5271.1196236559135, 4450.360632183908, 4336.594885598924, 4263.481944444445, 4508.034946236559, 4994.327777777778, 5475.64333781965, 4869.987903225807, 4643.55, 6222.820134228188, 6439.015277777778, 6176.176075268817, 6412.591397849463, 5561.144345238095, 5337.418573351279, 5169.840277777777, 5493.271505376344, 7194.280555555555, 7366.073924731183, 6701.935483870968, 7092.451388888889, 6961.8, 8534.818055555555, 5835.915322580645, 5687.715053763441, 5468.040178571428, 4936.013458950202, 4745.273611111111, 5777.162634408603, 5807.555555555556, 5865.691790040377, 6119.346774193548, 5842.0625, 6127.481879194631, 6945.804166666667, 6463.5362903225805]
```

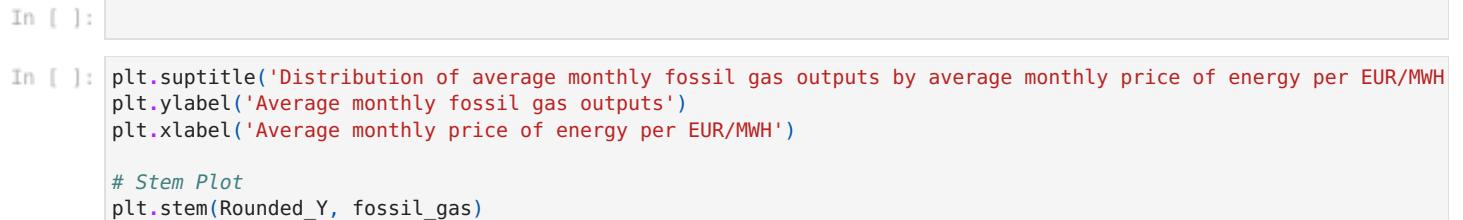
```
In [ ]: df_FossilGas.describe(include = 'all') # Description Tables
```

	Price	Fossil_Gas	Dates	Ratio	First Difference	Seasonal Difference	First Difference in Fossil Gas Ratio	Seasonal Difference Fossil Gas Ratio
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	5619.729849	NaN	98.939095	44.591375	253.805233	0.629371	3.307582
std	10.320573	934.269176	NaN	16.930010	759.010275	1127.620344	12.972232	28.678458
min	32.618667	4263.481944	NaN	74.674100	-2698.902733	-2071.281765	-40.865826	-51.111429
25%	51.528411	4931.070828	NaN	86.859281	-225.081838	-470.421136	-7.942808	-23.312681
50%	59.622281	5471.841758	NaN	93.942305	58.136234	76.041401	0.431358	8.728835
75%	64.999583	6187.837090	NaN	108.186846	329.742819	1065.927419	8.774459	23.909345
max	79.492083	8534.818056	NaN	133.316378	1701.009050	2448.901389	33.864455	52.882167

Below is the box and whisker plot for the distribution of the average monthly outputs of fossil gas and its the average monthly prices of energy per EUR/MWH.

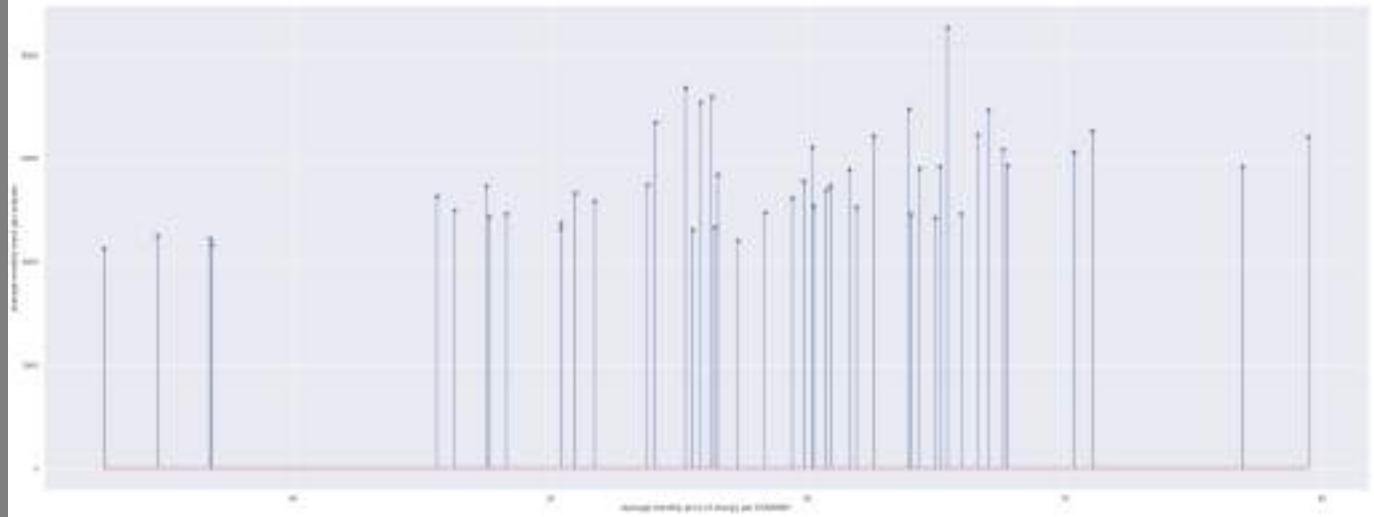


This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil gas produced and its the average monthly prices of energy per EUR/MWH. As one can observe, there is no area in which the values are concentrated. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), fossil gas output was slightly above 4000 units whcih was also the lowest amount produced. At approximately 66.62 EUR/MWH, fossil gas produced its highest output, which was slightly below 9000 units. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 6500 units.



/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

Out[]: <StemContainer object of 3 artists>



This plot depicts the frequencies between the distribution of the monthly average output of fossil gas produced and its the average monthly prices of energy per EUR/MWH. As one can observe, there is no area in which the values are concentrated. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), fossil gas output was slightly above 4000 units whcih was also the lowest amount produced. At approximately 66.62 EUR/MWH, fossil gas produced its highest output, which was slightly below 9000 units. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 6500 units.Each blue dot at the end of the blue lines represent an observation.

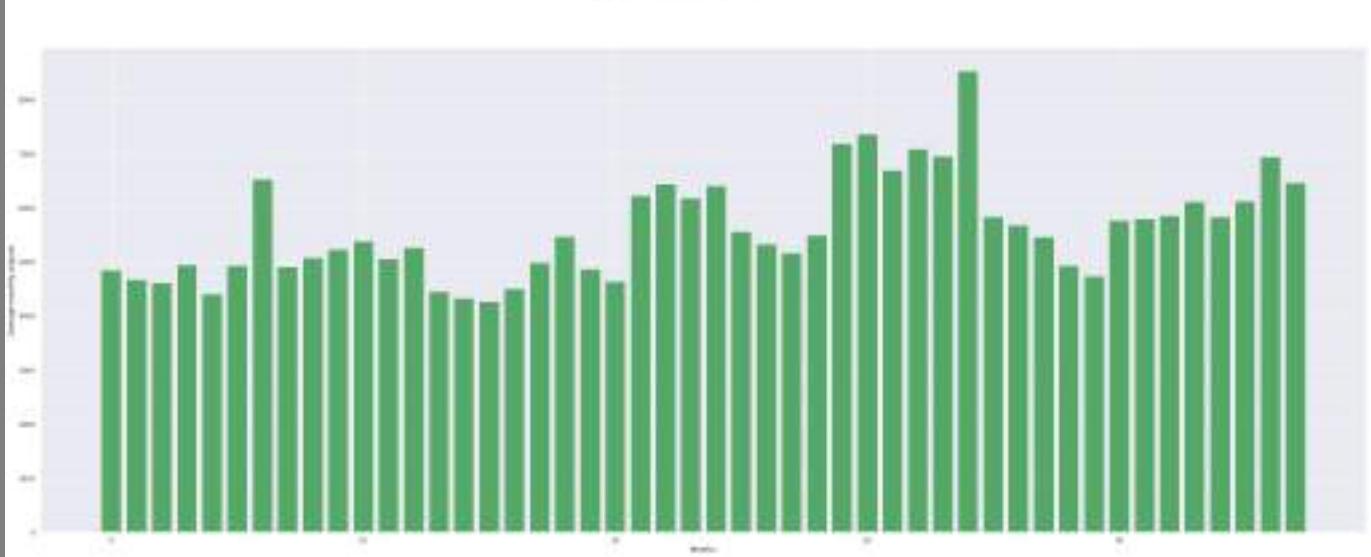
```
In [ ]: #Outputs in MWH Histogram
fossil_gas_Dict = {key: i for i, key in enumerate(fossil_gas)}

def Hist_fossil_gas(fossil_gas_Dict):
    for k, v in fossil_gas_Dict.items(): print(f"{v}:{k}")
print(fossil_gas_Dict)

plt.bar(list(fossil_gas_Dict.values()), fossil_gas_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of fossil gas")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{4850.009549795362: 0, 4674.135416666667: 1, 4614.7523553162855: 2, 4952.123955431754: 3, 4415.3494623655915: 4, 4934.930458970793: 5, 6529.490591397849: 6, 4919.491935483871: 7, 5076.581944444444: 8, 5231.5: 9, 5387.741666666667: 10, 5062.588156123822: 11, 5271.1196236559135: 12, 4450.360632183908: 13, 4336.594885598924: 14, 4263.481944444445: 15, 4508.034946236559: 16, 4994.327777777778: 17, 5475.64333781965: 18, 4869.987903225807: 19, 4643.55: 20, 6222.820134228188: 21, 6439.015277777778: 22, 6176.176075268817: 23, 6412.591397849463: 24, 5561.144345238095: 25, 5337.418573351279: 26, 5169.840277777777: 27, 5493.271505376344: 28, 7194.280555555555: 29, 7366.073924731183: 30, 6701.935483870968: 31, 7092.451388888889: 32, 6961.8: 33, 8534.818055555555: 34, 5835.915322580645: 35, 5687.715053763441: 36, 5468.040178571428: 37, 4936.013458950202: 38, 4745.273611111111: 39, 5777.162634408603: 40, 5807.555555555556: 41, 5865.691790040377: 42, 6119.346774193548: 43, 5842.0625: 44, 6127.481879194631: 45, 6945.804166666667: 46, 6463.5362903225805: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is bimodal, but indicates a gradual increase in the average monthly outputs. There are no sudden changes in the average monthly outputs.

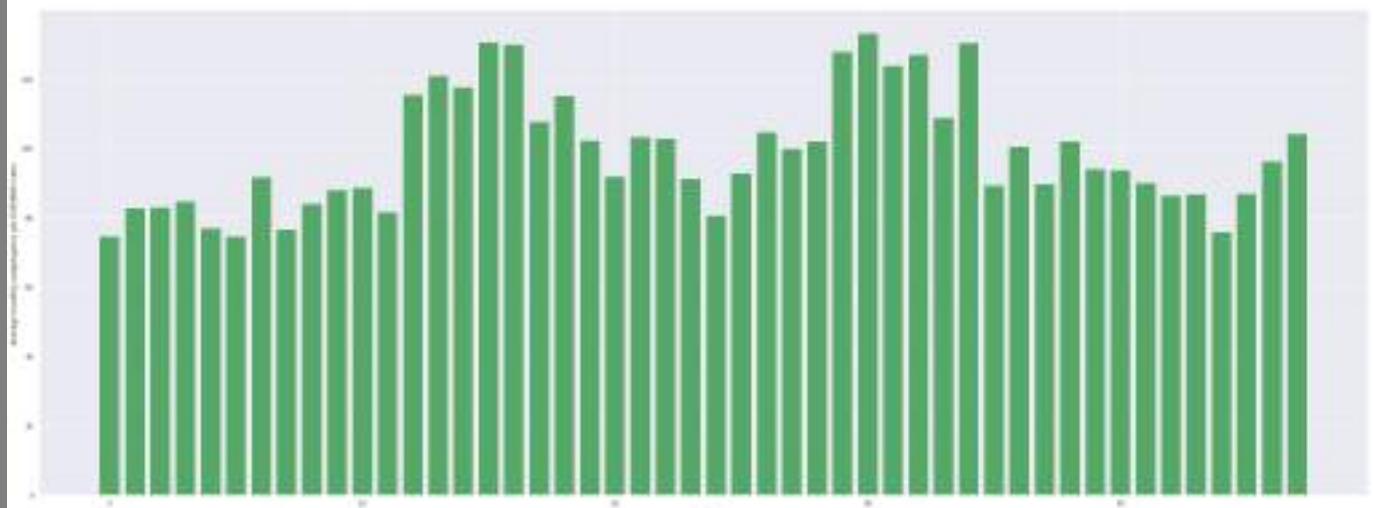
```
In [ ]: pdToListFossilGas_Dict = {key: i for i, key in enumerate(pdToListFossilGas)}

def Hist_pdToListFossilGas(pdToListFossilGas_Dict):
    for k, v in pdToListFossilGas_Dict.items(): print(f'{v}:{k}')
#Output/price per EUR/MWH Histogram
print(pdToListFossilGas_Dict)

plt.bar(list(pdToListFossilGas_Dict.values()), pdToListFossilGas_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of fossil gas")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')

plt.show()
plt.show()

{74.67410036547997: 0, 82.89847307795338: 1, 83.11505122391748: 2, 84.8633664099214: 3, 77.06469970286285: 4, 74
.80011718384856: 5, 91.87143516347186: 6, 76.86938617095277: 7, 84.2519209514227: 8, 88.06236109868401: 9, 88.72
099972348832: 10, 81.78423512620311: 11, 115.6486900692026: 12, 121.09138390387596: 13, 117.78461443980032: 14,
130.7068123909963: 15, 129.94686633827166: 16, 107.94737592591203: 17, 115.27180915744435: 18, 102.3056436980427
7: 19, 92.12369768238372: 20, 103.39928418983801: 21, 102.89080650072484: 22, 91.3701278842762: 23, 80.669560149
27435: 24, 92.93698241774639: 25, 104.73763443132118: 26, 99.9622686099004: 27, 102.15740662281915: 28, 127.8796
2845924743: 29, 133.31637796317597: 30, 123.91640493713274: 31, 127.06716346603655: 32, 108.90525726970706: 33,
130.4406679930639: 34, 89.57484184347635: 35, 100.64618999065522: 36, 89.82097272974603: 37, 102.23782840184066:
38, 94.15087907942261: 39, 93.73373151149524: 40, 90.25211382699923: 41, 86.53576324248239: 42, 86.9671208148880
5: 43, 75.95573413393849: 44, 87.0848352294026: 45, 96.40938097061529: 46, 104.25454559857808: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15
-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18
, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03
': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '
17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08':
44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is bimodal with gradual declines in output amount occurring after every modal and gradual increases before every modal.

In []:

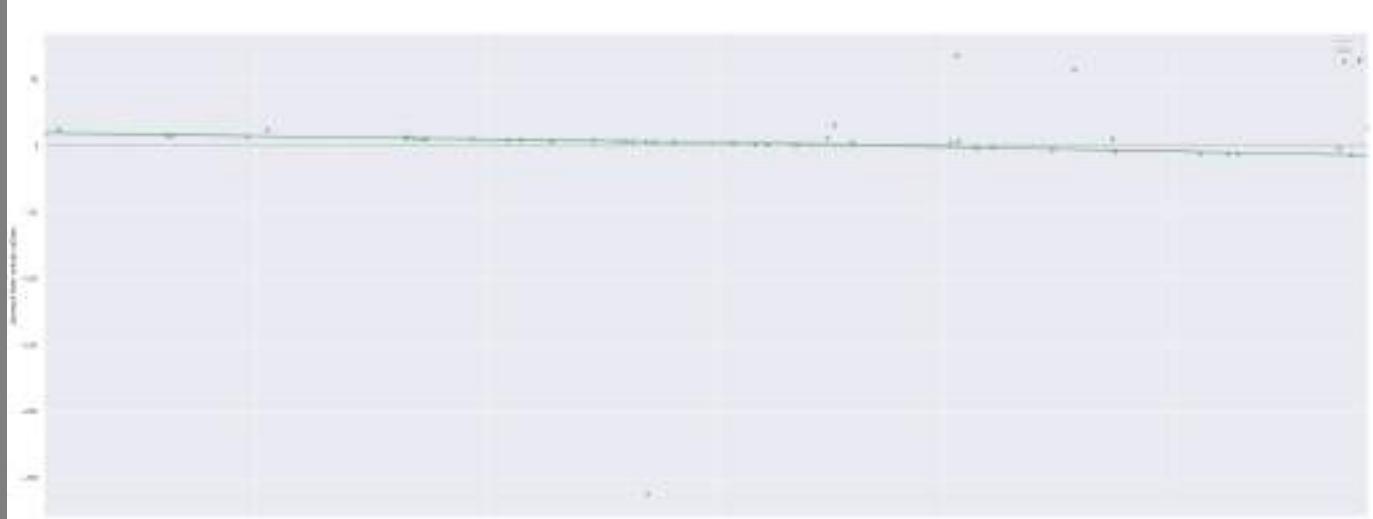
```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
FossilGasQuadRatioPredict = FossilGas_ypred/ypred

sns.residplot(x = FossilGasQuadRatioPredict , y = standardized_residualsFossilGasQuad/standardized_residualsPri

plt.suptitle("Predicted average monthly outputs to EUR/MWH versus respective quadratic model residuals ")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

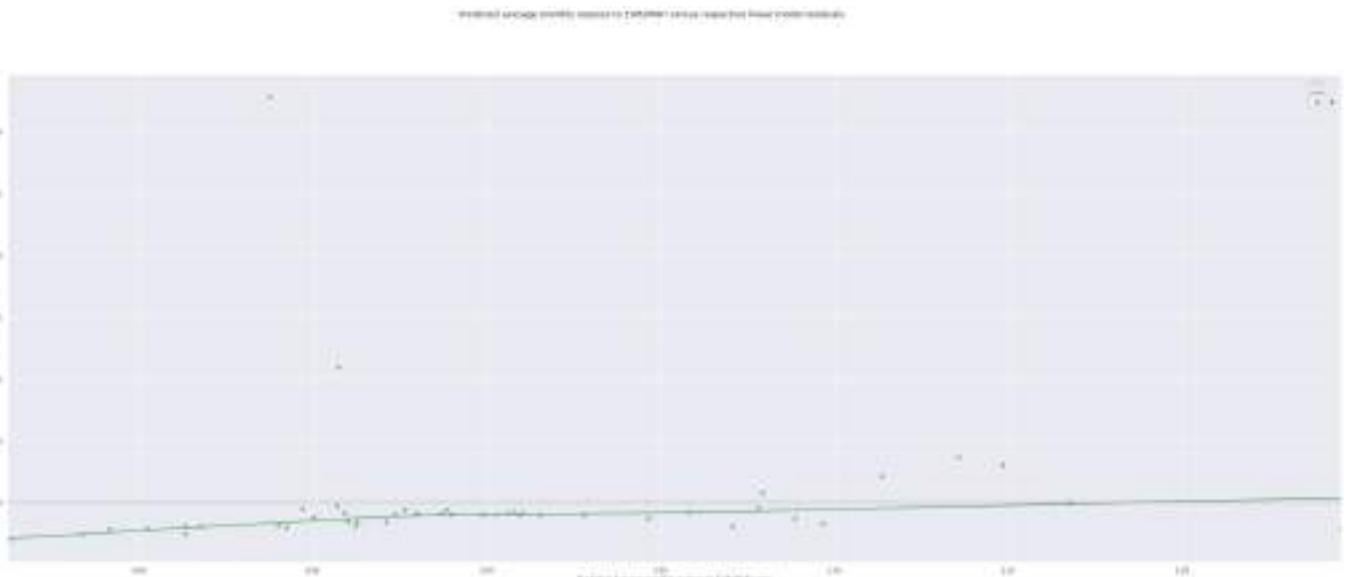
Out[]: <matplotlib.legend.Legend at 0x7ff604bb6ad0>



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals  
  
FossilGasRegRatioPredict = predictionsFossilGas/predictions  
  
sns.residplot(x = FossilGasRegRatioPredict, y = standardized_residualsFossilGas/standardized_residualsPricereg,  
plt.suptitle("Predicted average monthly outputs to EUR/MWH versus respective linear model residuals ")  
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")  
plt.ylabel("Amount from actual values")  
  
plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff604b6c0d0>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of fossil fuel gas.

```
In [ ]:
```

```
In [ ]: modelfossilfuelgas = stats.linregress([4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431,  
[64.9490188172043,  
56.38385416666666,  
55.522462987886975,  
58.35408333333333,  
57.29405913978498,  
65.9749027777778,  
71.07204301075271,  
63.99806451612899,  
60.25479166666634,  
59.40676510067113,  
60.72679166666668,  
61.901760752688226,  
45.57872311827956,  
36.75208333333374,  
36.81800807537014,  
32.61866666666666,  
34.691370967741896,  
46.266319444444434,  
47.50201612903221,  
47.6023387096774,  
50.4055972222224,  
60.182429530201404,  
62.58105555555558,  
67.5951344086021,  
79.49208333333331,  
59.83779761904767,  
50.95989232839837,  
51.71791666666662,  
53.77262096774189,  
56.2582222222224,  
55.252580645161316,  
54.08432795698931,  
55.81655555555558,
```

```
63.92528859060403,  
65.43065277777781,  
65.15127688172035,  
56.51197580645163,  
60.877098214285674,  
48.279717362045766,  
50.40073611111113,  
61.633763440860214,  
64.34813888888884,  
67.78344086021498,  
70.36391129032262,  
76.9140416666666,  
70.36221476510062,  
67.0426075268817,  
66.62351388888881])
```

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: Rounded_Logpred = [round(item, 2) for item in Logpred]  
print(Rounded_Logpred)  
#Rounded Price
```

```
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.  
36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.  
69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

```
In [ ]: print(list(FossilGas_Logpred))  
  
print(list(Logpred))
```

```
[53.368194409384955, 52.34189209151994, 51.995366013911806, 53.96407652876036, 50.83176311458983, 53.86374497370  
913, 63.16869920872184, 53.77365445182751, 54.69034320782382, 55.59435766786649, 56.50609597636326, 54.608683346  
1166, 55.82555546180956, 51.03606856927209, 50.37219579084379, 49.94554987289142, 51.37262336070465, 54.21035424  
979846, 57.01904058762399, 53.484776820158174, 52.16341283786586, 61.37914326086883, 62.64073627053412, 61.10695  
4821483, 62.48654139983319, 57.517976278349664, 56.21243876598811, 55.23454628054169, 57.12190863820826, 67.0480  
3876236276, 68.05052804172581, 64.17499041084899, 66.45382113685332, 65.69141326672707, 74.87066008228277, 59.12  
138484055332, 58.25657159775531, 56.974672835409756, 53.870064751207636, 52.75701450064443, 58.77853726553417, 5  
8.95589322382612, 59.295143522561396, 60.77533104953514, 59.157256303409056, 60.822802937357196, 65.598070598682  
94, 62.78382707964539]  
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639  
58084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.0949  
16817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 1  
9.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.36296287420  
1227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.88720  
2207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 2  
7.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.5566746731514  
95, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.4333662  
30197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: dfFossilGas_Log = ({"Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967  
"Fossil_Gas_Log" : [53.368194409384955, 52.34189209151994, 51.995366013911806, 53.96407652876036, 50.83176311458983]  
print(dfFossilGas_Log) #Dataframes  
df_FossilGas_Log= pd.DataFrame.from_dict(dfFossilGas_Log, orient = "columns")  
print(df_FossilGas_Log)  
  
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis, indicating it is non-stationary ")  
  
adfuller_test(df_FossilGas_Log["Fossil_Gas_Log"])  
  
test_result=adfuller(df_FossilGas_Log["Fossil_Gas_Log"])  
  
df_FossilGas_Log['First Difference'] = df_FossilGas_Log["Fossil_Gas_Log"]- df_FossilGas_Log["Fossil_Gas_Log"].shift(1)  
df_FossilGas_Log['Seasonal Difference']=df_FossilGas_Log["Fossil_Gas_Log"]- df_FossilGas_Log["Fossil_Gas_Log"].shift(12)  
df_FossilGas_Log.head() #Predictive Logarithmic Seasonality Plot
```

```
df_FossilGas_Log['Seasonal Difference'].plot()  
# Seasonality Plot
```

```
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834  
, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.6312803685  
45422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.3578  
44118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848,  
26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.07642699916  
3463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.8933  
89962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21  
.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.018681703731  
89, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Fossil_Gas_Log': [53.368194409384955, 52.34189  
209151994, 51.995366013911806, 53.96407652876036, 50.83176311458983, 53.86374497370913, 63.16869920872184, 53.77  
365445182751, 54.69034320782382, 55.59435766786649, 56.50609597636326, 54.6086833461166, 55.82555546180956, 51.0  
3606856927209, 50.37219579084379, 49.94554987289142, 51.37262336070465, 54.21035424979846, 57.01904058762399, 53  
.484776820158174, 52.16341283786586, 61.37914326086883, 62.64073627053412, 61.106954821483, 62.48654139983319, 5  
7.517976278349664, 56.21243876598811, 55.23454628054169, 57.12190863820826, 67.04803876236276, 68.05052804172581  
, 64.17499041084899, 66.45382113685332, 65.69141326672707, 74.87066008228277, 59.12138484055332, 58.256571597755  
31, 56.974672835409756, 53.870064751207636, 52.75701450064443, 58.77853726553417, 58.95589322382612, 59.29514352  
2561396, 60.77533104953514, 59.157256303409056, 60.822802937357196, 65.59807059868294, 62.78382707964539]}  
Price_Log Fossil_Gas_Log  
0 27.297348 53.368194  
1 23.917581 52.341892  
2 23.577680 51.995366  
3 24.695022 53.964077  
4 24.276743 50.831763  
5 27.702157 53.863745  
6 29.713461 63.168699  
7 26.922107 53.773654  
8 25.445032 54.608683  
9 25.110405 55.594358  
10 25.631280 56.506096  
11 26.094917 54.608683  
12 19.653934 55.825555  
13 16.170990 51.036069  
14 16.197004 50.372196  
15 14.539966 49.945550  
16 15.357844 51.372623  
17 19.925256 54.210354  
18 20.412855 57.019041  
19 20.452442 53.484777  
20 21.558593 52.163413  
21 25.416478 61.379143  
22 26.362963 62.640736  
23 28.341491 61.106955  
24 33.035963 62.486541  
25 25.280488 57.517976  
26 21.777315 56.212439  
27 22.076427 55.234546  
28 22.887202 57.121909  
29 23.868007 67.048039  
30 23.471186 68.050528  
31 23.010200 64.174990  
32 23.693728 66.453821  
33 26.893390 65.691413  
34 27.487399 74.870660  
35 27.377158 59.121385  
36 23.968136 58.256572  
37 25.690591 56.974673  
38 20.719732 53.870065  
39 21.556675 52.757015  
40 25.989167 58.778537  
41 27.060244 58.955893  
42 28.415796 59.295144  
43 29.434036 60.775331  
44 32.018682 59.157256  
45 29.433366 60.822803  
46 27.958095 65.598071  
47 28.123467 62.783827  
ADF Test Statistic : -2.9991155703001486  
p-value : 0.03497707747176592  
#Lags Used : 0  
Number of Observations : 47  
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff604a54290>
```



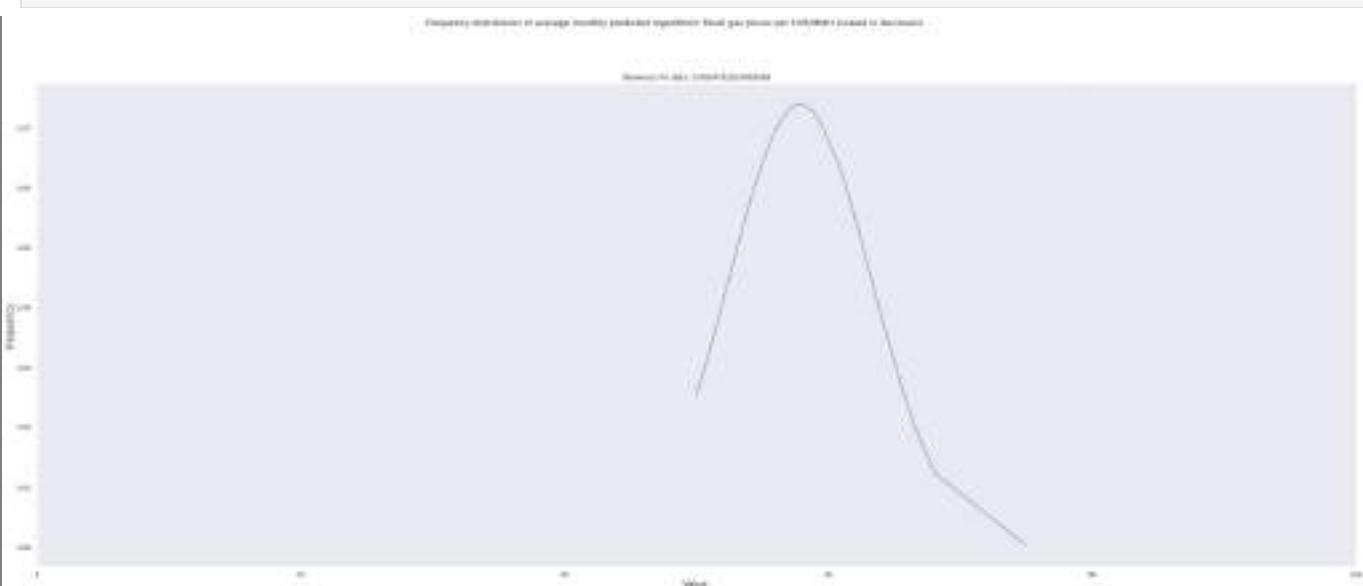
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
FossilGas_LogResults_mean = np.mean(df_FossilGas_Log["Fossil_Gas_Log"])
FossilGas_LogResults_std = np.std(df_FossilGas_Log["Fossil_Gas_Log"])

FossilGas_LogResultspdf = stats.norm.pdf(df_FossilGas_Log["Fossil_Gas_Log"].sort_values(), FossilGas_LogResults_mean, FossilGas_LogResults_std)

plt.plot(df_FossilGas_Log["Fossil_Gas_Log"].sort_values(), FossilGas_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilGas_Log["Fossil_Gas_Log"])}')
plt.suptitle("Frequency distribution of average monthly predicted logarithmic fossil gas prices per EUR/MWH (scale: log10)", size=10)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)

```

Fossil_Gas_Log_Dict = {key: i for i, key in enumerate(df_FossilGas_Log["Fossil_Gas_Log"])}

def Hist_Fossil_Gas_Log(Fossil_Gas_Log_Dict):
    for k, v in Fossil_Gas_Log_Dict.items(): print(f"{v}:{k}")
print(Fossil_Gas_Log_Dict)

plt.bar(list(Fossil_Gas_Log_Dict.values()), Fossil_Gas_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly output of fossil gas predicted logarithmic prices per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

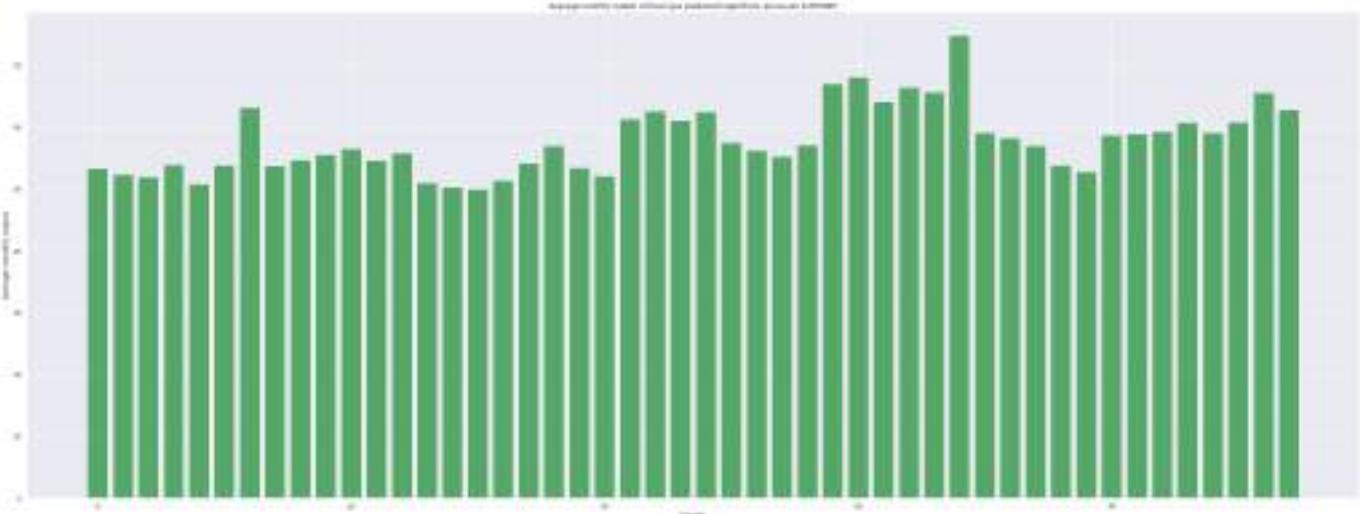
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{53.368194409384955: 0, 52.34189209151994: 1, 51.995366013911806: 2, 53.96407652876036: 3, 50.83176311458983: 4, 53.86374497370913: 5, 63.16869920872184: 6, 53.77365445182751: 7, 54.69034320782382: 8, 55.59435766786649: 9, 56.50609597636326: 10, 54.6086833461166: 11, 55.82555546180956: 12, 51.03606856927209: 13, 50.37219579084379: 14, 49.94554987289142: 15, 51.37262336070465: 16, 54.21035424979846: 17, 57.01904058762399: 18, 53.484776820158174: 19, 52.16341283786586: 20, 61.37914326086883: 21, 62.64073627053412: 22, 61.106954821483: 23, 62.48654139983319: 24, 57.517976278349664: 25, 56.21243876598811: 26, 55.23454628054169: 27, 57.12190863820826: 28, 67.048038762362: 29, 68.05052804172581: 30, 64.17499041084899: 31, 66.45382113685332: 32, 65.69141326672707: 33, 74.870660082: 28277: 34, 59.12138484055332: 35, 58.25657159775531: 36, 56.974672835409756: 37, 53.87064751207636: 38, 52.7570: 1450064443: 39, 58.77853726553417: 40, 58.95589322382612: 41, 59.295143522561396: 42, 60.77533104953514: 43, 59.157256303409056: 44, 60.822802937357196: 45, 65.59807059868294: 46, 62.78382707964539: 47}

```



The green bars represent the observation value for each respective month. This histogram is multimodal

```
In [ ]: df_FossilGas_Log.describe(include = 'all') # Description Tables
```

	Price_Log	Fossil_Gas_Log	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	0.200333	1.481064
std	4.072442	5.451868	4.478585	6.384807
min	14.539966	49.945550	-15.749275	-9.272589
25%	22.001649	53.841222	-1.427573	-2.745113
50%	25.195447	56.996857	0.177356	0.443735
75%	27.317301	61.175002	1.776454	5.913842
max	33.035963	74.870660	9.926130	14.290408

```
In [ ]: print(predictionsFossilGas/predictions)
```

```

Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price

```

```
[1.01034768 0.9869124 0.97643148 1.00933856 0.94695482 0.99944644
1.16745644 0.98989901 1.0028162 1.01540073 1.02802761 0.98964746
1.00778467 0.91777093 0.90235373 0.89128785 0.91326022 0.96004803
1.00596947 0.94005979 0.91339335 1.07074327 1.08867973 1.0580805
1.07796666 0.98859668 0.96261048 0.94240463 0.9710545 1.13565656
1.14846616 1.07915536 1.11346153 1.09674729 1.24554061 0.98004185
0.96228613 0.93779058 0.88357141 0.8622831 0.95734676 0.95689405
0.95906301 0.97960719 0.9502425 0.97364314 1.04649344 0.99818347]
[52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04,
56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59
.68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]
```

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(predictionsFossilGas))
print(list(predictions))
```

```
[53.368194409384955, 52.34189209151994, 51.995366013911806, 53.96407652876036, 50.83176311458983, 53.86374497370
913, 63.16869920872184, 53.77365445182751, 54.69034320782382, 55.59435766786649, 56.50609597636326, 54.608683346
1166, 55.82555546180956, 51.03606856927209, 50.37219579084379, 49.94554987289142, 51.37262336070465, 54.21035424
979846, 57.01904058762399, 53.484776820158174, 52.16341283786586, 61.37914326086883, 62.64073627053412, 61.10695
4821483, 62.48654139983319, 57.517976278349664, 56.21243876598811, 55.23454628054169, 57.12190863820826, 67.0480
3876236276, 68.05052804172581, 64.17499041084899, 66.45382113685332, 65.69141326672707, 74.87066008228277, 59.12
138484055332, 58.25657159775531, 56.974672835409756, 53.870064751207636, 52.75701450064443, 58.77853726553417, 5
8.95589322382612, 59.295143522561396, 60.77533104953514, 59.157256303409056, 60.822802937357196, 65.598070598682
94, 62.78382707964539]
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.
03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373
92337, 62.89808335677895]
```

```
In [ ]: dfFossilGasReg = ({ "Price_Reg" : [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 5
" Fossil_GasReg" : [53.368194409384955, 52.34189209151994, 51.995366013911806, 53.96407652876036, 50.83176311458983
print(dfFossilGasReg) #Dataframes
df_FossilGasReg= pd.DataFrame.from_dict(dfFossilGasReg, orient = "columns")
print(df_FossilGasReg)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+ ' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_FossilGasReg["Fossil_GasReg"])

test_result=adfuller(df_FossilGasReg[ "Fossil_GasReg"])
test_result=adfuller(df_FossilGasReg[ "Fossil_GasReg"])

df_FossilGasReg['First Difference'] = df_FossilGasReg[ "Fossil_GasReg"]- df_FossilGasReg[ "Fossil_GasReg"].shift()
df_FossilGasReg['Seasonal Difference']=df_FossilGasReg[ "Fossil_GasReg"]- df_FossilGasReg[ "Fossil_GasReg"].shift(4)
plt.suptitle("Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilGasReg.head() #Predictive Linear Seasonality Plot
df_FossilGasReg['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Reg': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.251900888255 93, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878 538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.824616 68252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.110 974579653693, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.182939493943, 61.3 9733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'Fossil_GasReg': [53.368194409384955, 52.34189209151994, 51.9953660139118 06, 53.96407652876036, 50.83176311458983, 53.86374497370913, 63.16869920872184, 53.77365445182751, 54.6903432078 2382, 55.59435766786649, 56.50609597636326, 54.6086833461166, 55.82555546180956, 51.03606856927209, 50.372195790 84379, 49.94554987289142, 51.37262336070465, 54.21035424979846, 57.01904058762399, 53.484776820158174, 52.163412 83786586, 61.37914326086883, 62.64073627053412, 61.106954821483, 62.48654139983319, 57.517976278349664, 56.21243 876598811, 55.23454628054169, 57.12190863820826, 67.04803876236276, 68.05052804172581, 64.17499041084899, 66.453 82113685332, 65.69141326672707, 74.87066008228277, 59.12138484055332, 58.25657159775531, 56.974672835409756, 53. 870064751207636, 52.75701450064443, 58.77853726553417, 58.95589322382612, 59.295143522561396, 60.77533104953514, 59.157256303409056, 60.822802937357196, 65.59807059868294, 62.78382707964539]}
```

```
Price_Reg Fossil_GasReg
```

0	52.821613	53.368194
1	53.036006	52.341892
2	53.250399	51.995366
3	53.464792	53.964077
4	53.679185	50.831763
5	53.893578	53.863745
6	54.107971	63.168699
7	54.322364	53.773654
8	54.536757	54.690343
9	54.751150	55.594358
10	54.965543	56.506096
11	55.179936	54.608683
12	55.394329	55.825555
13	55.608722	51.036069
14	55.823115	50.372196
15	56.037508	49.945550
16	56.251901	51.372623
17	56.466294	54.210354
18	56.680687	57.019041
19	56.895080	53.484777
20	57.109473	52.163413
21	57.323866	61.379143
22	57.538259	62.640736
23	57.752652	61.106955
24	57.967045	62.486541
25	58.181438	57.517976
26	58.395831	56.212439
27	58.610224	55.234546
28	58.824617	57.121909
29	59.039010	67.048039
30	59.253403	68.050528
31	59.467796	64.174990
32	59.682189	66.453821
33	59.896582	65.691413
34	60.110975	74.870660
35	60.325368	59.121385
36	60.539761	58.256572
37	60.754154	56.974673
38	60.968547	53.870065
39	61.182939	52.757015
40	61.397332	58.778537
41	61.611725	58.955893
42	61.826118	59.295144
43	62.040511	60.775331
44	62.254904	59.157256
45	62.469297	60.822803
46	62.683690	65.598071
47	62.898083	62.783827

```
ADF Test Statistic : -2.9991155703001486
```

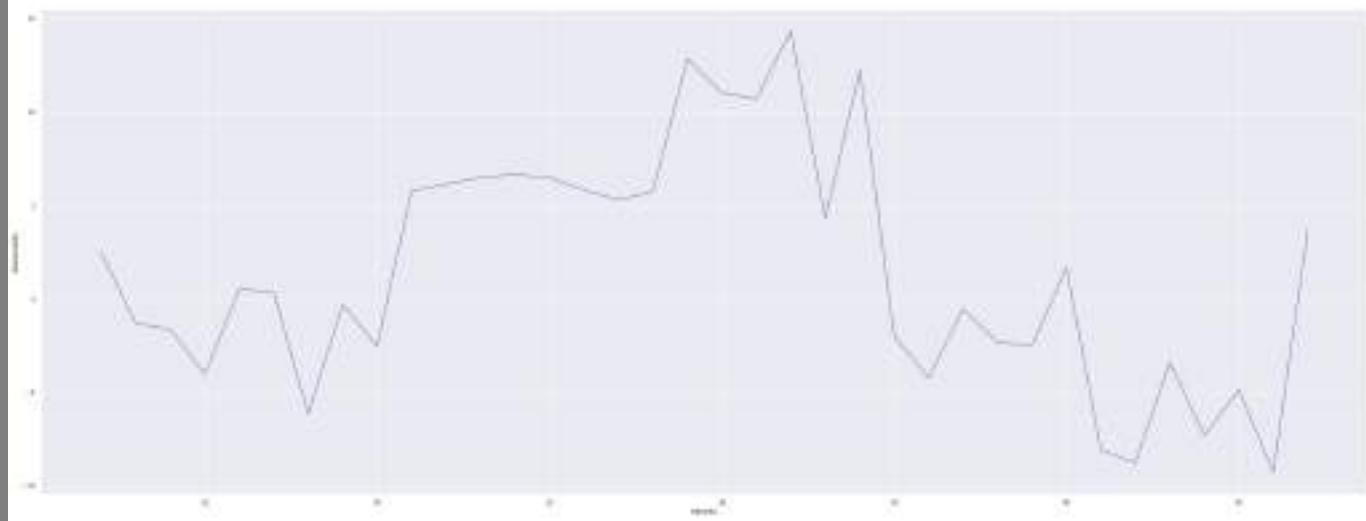
```
p-value : 0.03497707747176592
```

```
#Lags Used : 0
```

```
Number of Observations : 47
```

```
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606da5410>
```



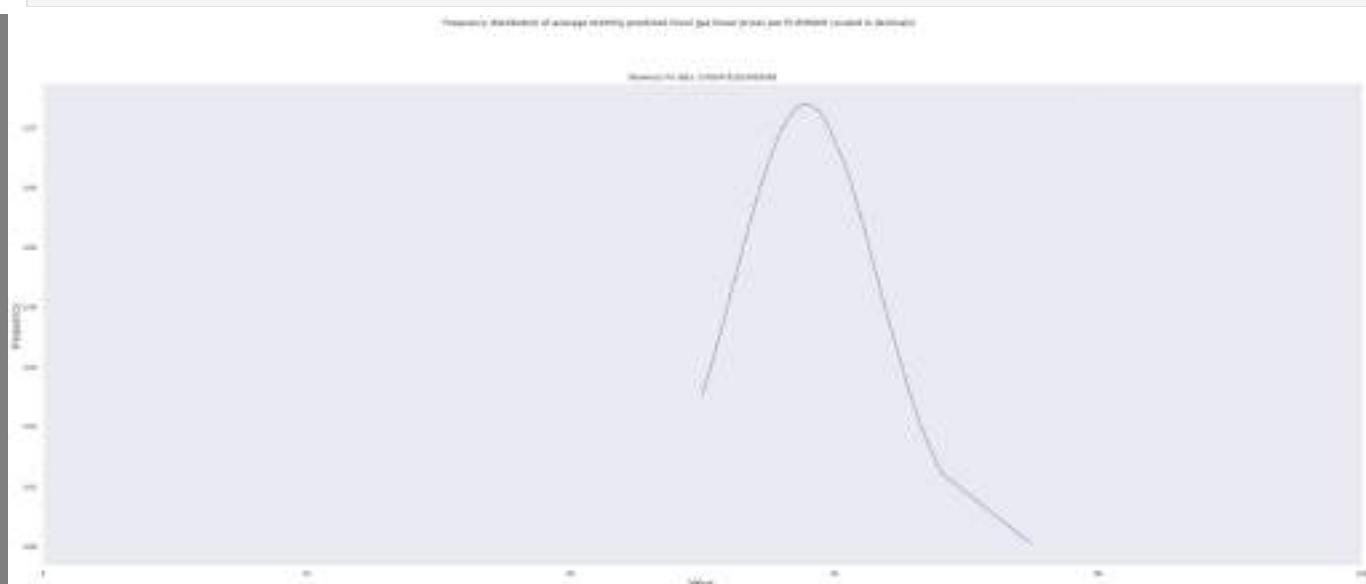
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
FossilGasRegResults_mean = np.mean(df_FossilGasReg["Fossil_GasReg"])
FossilGasRegResults_std = np.std(df_FossilGasReg["Fossil_GasReg"])

FossilGasRegResultspdf = stats.norm.pdf(df_FossilGasReg["Fossil_GasReg"].sort_values(), FossilGasRegResults_mean, FossilGasRegResults_std)

plt.plot(df_FossilGasReg["Fossil_GasReg"].sort_values(), FossilGasRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilGasReg["Fossil_GasReg"])}')
plt.suptitle("Frequency distribution of average monthly predicted fossil gas linear prices per EUR/MWH (scaled :")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Fossil_GasReg_Dict = {key: i for i, key in enumerate(df_FossilGasReg["Fossil_GasReg"])}

def Hist_Fossil_GasReg(Fossil_GasReg_Dict):
    for k, v in Fossil_GasReg_Dict.items(): print(f"{v}:{k}")
print(Fossil_GasReg_Dict)
```

```

plt.bar(list(Fossil_GasReg_Dict.values()), Fossil_GasReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted linear fossil gas prices per EUR/MWh ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

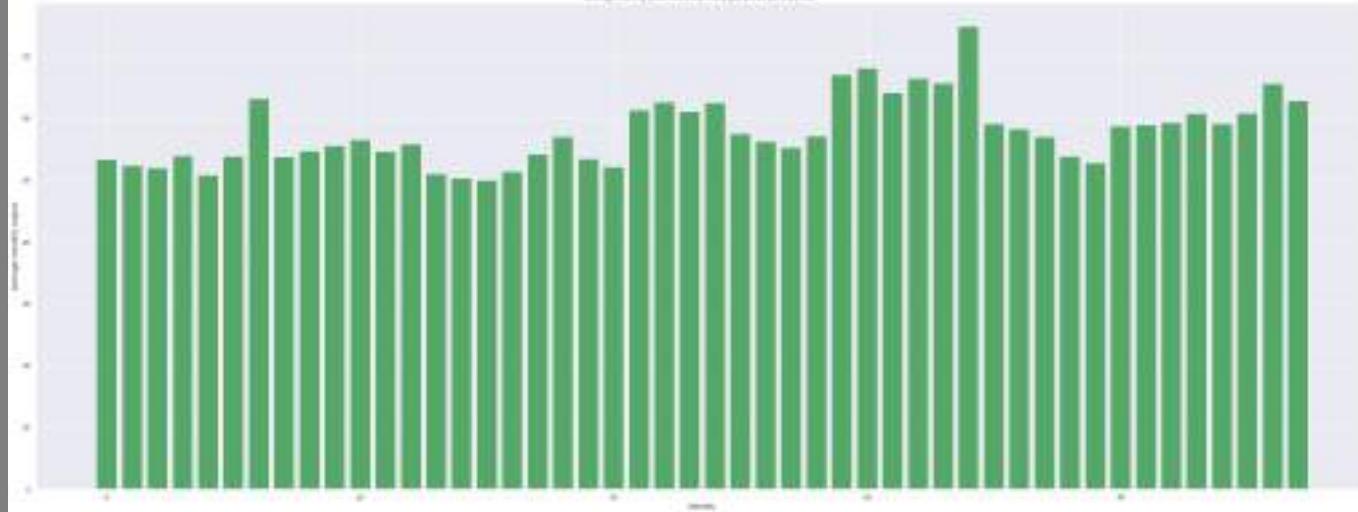
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{53.368194409384955: 0, 52.34189209151994: 1, 51.995366013911806: 2, 53.96407652876036: 3, 50.83176311458983: 4, 53.86374497370913: 5, 63.16869920872184: 6, 53.77365445182751: 7, 54.69034320782382: 8, 55.59435766786649: 9, 56.50609597636326: 10, 54.6086833461166: 11, 55.82555546180956: 12, 51.03606856927209: 13, 50.37219579084379: 14, 49.94554987289142: 15, 51.37262336070465: 16, 54.21035424979846: 17, 57.01904058762399: 18, 53.484776820158174: 19, 52.16341283786586: 20, 61.37914326086883: 21, 62.64073627053412: 22, 61.106954821483: 23, 62.48654139983319: 24, 57.517976278349664: 25, 56.21243876598811: 26, 55.23454628054169: 27, 57.12190863820826: 28, 67.048038762362 76: 29, 68.05052804172581: 30, 64.17499041084899: 31, 66.45382113685332: 32, 65.69141326672707: 33, 74.870660082 28277: 34, 59.12138484055332: 35, 58.25657159775531: 36, 56.974672835409756: 37, 53.870064751207636: 38, 52.7570 1450064443: 39, 58.77853726553417: 40, 58.95589322382612: 41, 59.295143522561396: 42, 60.77533104953514: 43, 59. 157256303409056: 44, 60.822802937357196: 45, 65.59807059868294: 46, 62.78382707964539: 47}

```



The green bars represent the observation value for each respective month. This histogram is multimodal.

```
In [ ]: df_FossilGasReg.describe(include = 'all') # Description Tables
```

	Price_Reg	Fossil_GasReg	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	57.859848	57.859848	0.200333	1.481064
std	3.001502	5.451868	4.478585	6.384807
min	52.821613	49.945550	-15.749275	-9.272589
25%	55.340731	53.841222	-1.427573	-2.745113
50%	57.859848	56.996857	0.177356	0.443735
75%	60.378966	61.175002	1.776454	5.913842
max	62.898083	74.870660	9.926130	14.290408

```
In [ ]: print(list(FossilGas_ypred))
```

```
print(list(ypred))
```

[52.24719287654996, 49.661271525709566, 48.73778675153166, 53.646277019315136, 45.45078951927634, 53.41596984657
 46, 65.70819512077176, 53.20735447921798, 55.24984162863376, 57.089846226197835, 58.7703180340357, 55.0751126956
 49295, 57.53263505715947, 46.04866852473859, 44.07361487943113, 42.755068688058756, 47.01429170858647, 54.202561
 315529806, 59.6383904296284, 52.52683451714151, 49.18880314121975, 64.76788090065577, 65.50129132790323, 64.5654
 5272141838, 65.4297286609673, 60.42930923998331, 58.248277693520336, 56.37822554388498, 59.805771196656366, 65.4
 1850442637292, 64.82555647959936, 65.93911025787733, 65.6695286648021, 65.8821107133308, 55.14373648777055, 62.6
 14199513390545, 61.50337778881625, 59.56550631889708, 53.43053954560075, 50.73409291525142, 62.19276855262842, 6
 2.413883053014786, 62.81828237780809, 64.29762447450886, 62.65685484273281, 64.33739204048922, 65.89968138083438
 , 65.56319729117945]
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338
 71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672
 299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486
 , 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506
 2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20
 1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,
 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905
 53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857
 92181256, 28.070861800194585, 28.297337605159356]

```
In [ ]: dfFossilGas_Quad = [{"Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, "Fossil_Gas_Quad": [52.24719287654996, 49.661271525709566, 48.73778675153166, 53.646277019315136, 45.4507895195], print(dfFossilGas_Quad) #Dataframes df_FossilGas_Quad= pd.DataFrame.from_dict(dfFossilGas_Quad, orient = "columns") print(df_FossilGas_Quad)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_FossilGas_Quad["Fossil_Gas_Quad"])

test_result=adfuller(df_FossilGas_Quad["Fossil_Gas_Quad"])

df_FossilGas_Quad['First Difference'] = df_FossilGas_Quad["Fossil_Gas_Quad"]- df_FossilGas_Quad["Fossil_Gas_Quad"]
df_FossilGas_Quad['Seasonal Difference']=df_FossilGas_Quad["Fossil_Gas_Quad"]- df_FossilGas_Quad["Fossil_Gas_Quad"]
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_FossilGas_Quad.head() #Predictive Quadratic Seasonality Plot
df_FossilGas_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

```

{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398
, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135
5777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.
328254644736486, 19.733629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.8867354176134
72, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.4561726
37404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.66
5789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 2
1.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527
835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Fossil_Gas_Quad': [52.24719287654996, 49.6612
71525709566, 48.73778675153166, 53.646277019315136, 45.45078951927634, 53.4159698465746, 65.70819512077176, 53.2
0735447921798, 55.24984162863376, 57.089846226197835, 58.7703180340357, 55.075112695649295, 57.53263505715947, 4
6.04866852473859, 44.07361487943113, 42.755068688058756, 47.01429170858647, 54.202561315529806, 59.6383904296284
, 52.52683451714151, 49.18880314121975, 64.76788090065577, 65.50129132790323, 64.56545272141838, 65.429728660967
3, 60.42930923998331, 58.248277693520336, 56.37822554388498, 59.805771196656366, 65.41850442637292, 64.825556479
59936, 65.93911025787733, 65.6695286648021, 65.8821107133308, 55.14373648777055, 62.614199513390545, 61.50337778
881625, 59.56550631889708, 53.43053954560075, 50.73409291525142, 62.19276855262842, 62.413883053014786, 62.81828
237780809, 64.29762447450886, 62.65685484273281, 64.33739204048922, 65.89968138083438, 65.56319729117945]}

    Price_Quad   Fossil_Gas_Quad
0      27.188078      52.247193
1      23.225548      49.661272
2      22.878236      48.737787
3      24.055117      53.646277
4      23.602709      45.450790
5      27.724722      53.415970
6      30.587839      65.708195
7      26.702482      53.207354
8      24.901804      55.249842
9      24.518415      57.089846
10     25.119121      58.770318
11     25.672300      55.075113
12     19.546363      57.532635
13     17.633406      46.048669
14     17.644052      44.073615
15     17.075344      42.755069
16     17.328255      47.014292
17     19.736629      54.202561
18     20.093546      59.638390
19     20.123369      52.526835
20     21.007992      49.188803
21     24.868735      64.767881
22     26.000055      65.501291
23     28.599305      64.565453
24     36.035067      65.429729
25     24.712150      60.429309
26     21.194644      58.248278
27     21.456173      56.378226
28     22.201511      59.805771
29     23.174311      65.418504
30     22.771345      64.825556
31     22.319233      65.939110
32     22.995762      65.669529
33     26.665789      65.882111
34     27.438370      55.143736
35     27.292830      62.614200
36     23.278006      61.503378
37     25.188915      59.565506
38     20.328049      53.430540
39     21.006372      50.734093
40     25.544593      62.192769
41     26.879918      62.413883
42     28.703098      62.818282
43     30.170478      64.297624
44     34.272280      62.656855
45     30.169486      64.337392
46     28.070862      65.899681
47     28.297338      65.563197

ADF Test Statistic : -2.999999105516227
p-value : 0.03489448380895966
#Lags Used : 0
Number of Observations : 47
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6052fb310>

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: #Bell Curves
```

```
FossilGas_QuadResults_mean = np.mean(df_FossilGas_Quad["Fossil_Gas_Quad"])
FossilGas_QuadResults_std = np.std(df_FossilGas_Quad["Fossil_Gas_Quad"])

FossilGas_QuadResultspdf = stats.norm.pdf(df_FossilGas_Quad["Fossil_Gas_Quad"].sort_values(), FossilGas_QuadResults_mean, FossilGas_QuadResults_std)

plt.plot(df_FossilGas_Quad["Fossil_Gas_Quad"].sort_values(), FossilGas_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilGas_Quad["Fossil_Gas_Quad"])}')
plt.suptitle("Frequency distribution of average monthly quadratic fossil gas predicted prices per EUR/MWh (scale: log10)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

This bell shaped curve is slightly skewed to the left, hence it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and -0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Fossil_Gas_Quad_Dict = {key: i for i, key in enumerate(df_FossilGas_Quad["Fossil_Gas_Quad"])}
def Hist_Fossil_Gas_Quad(Fossil_Gas_Quad_Dict):
    for k, v in Fossil_Gas_Quad_Dict.items(): print(f"{v}:{k}")
print(Fossil Gas Quad Dict)
```

```

plt.bar(list(Fossil_Gas_Quad_Dict.values()), Fossil_Gas_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic fossil gas prices per EUR/MWh ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

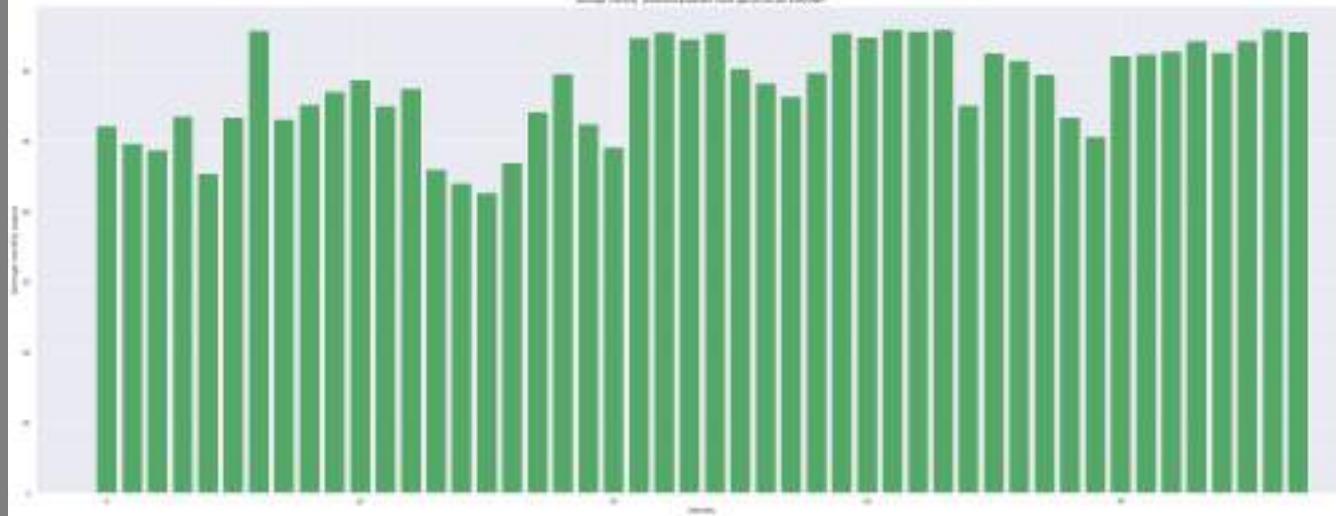
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-010': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{52.24719287654996: 0, 49.661271525709566: 1, 48.73778675153166: 2, 53.646277019315136: 3, 45.45078951927634: 4, 53.4159698465746: 5, 65.70819512077176: 6, 53.20735447921798: 7, 55.24984162863376: 8, 57.089846226197835: 9, 58.7703180340357: 10, 55.075112695649295: 11, 57.53263505715947: 12, 46.04866852473859: 13, 44.07361487943113: 14, 42.755068688058756: 15, 47.01429170858647: 16, 54.202561315529806: 17, 59.6383904296284: 18, 52.52683451714151: 19, 49.18880314121975: 20, 64.76788090065577: 21, 65.50129132790323: 22, 64.56545272141838: 23, 65.4297286609673: 24, 60.42930923998331: 25, 58.248277693520336: 26, 56.37822554388498: 27, 59.805771196656366: 28, 65.41850442637292: 29, 64.82555647959936: 30, 65.93911025787733: 31, 65.6695286648021: 32, 65.8821107133308: 33, 55.1437364877055: 34, 62.614199513390545: 35, 61.50337778881625: 36, 59.56550631889708: 37, 53.43053954560075: 38, 50.73409291525142: 39, 62.19276855262842: 40, 62.413883053014786: 41, 62.81828237780809: 42, 64.29762447450886: 43, 62.65685484273281: 44, 64.33739204048922: 45, 65.89968138083438: 46, 65.56319729117945: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform with increasing values overtime.

```
In [ ]: df_FossilGas_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Fossil_Gas_Quad	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	0.283319	2.420923
std	4.174498	6.836565	5.672955	7.570925
min	17.075344	42.755069	-12.500841	-10.891208
25%	21.390791	53.363816	-2.078043	-3.162656
50%	24.615282	59.167912	0.212582	0.950411
75%	27.214266	64.394407	2.250005	8.295405
max	36.035067	65.939110	15.579078	16.480726

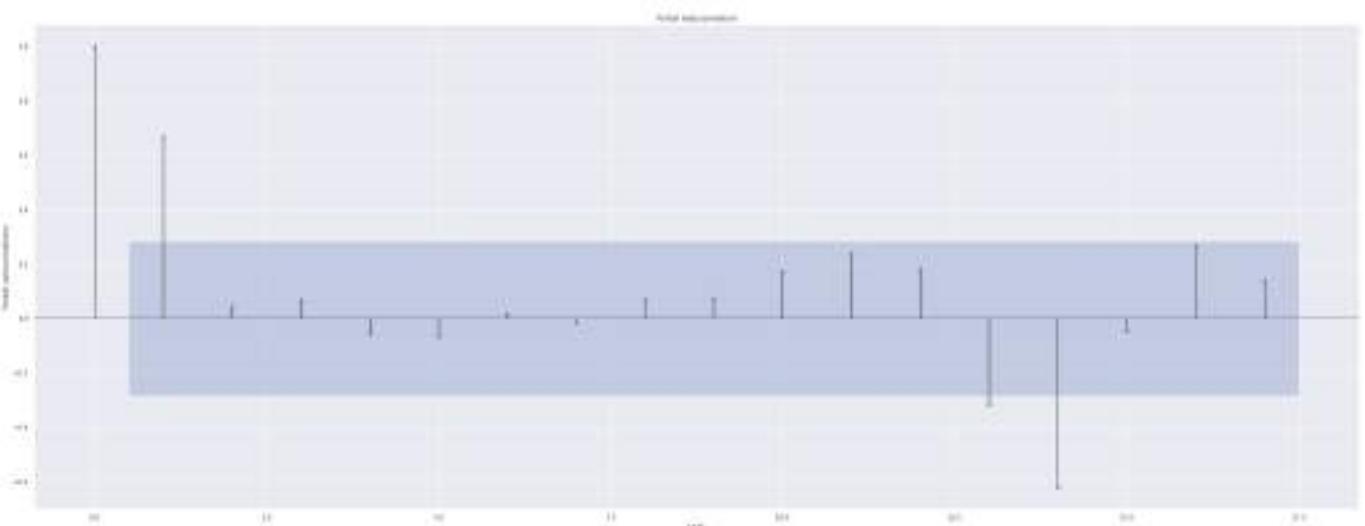
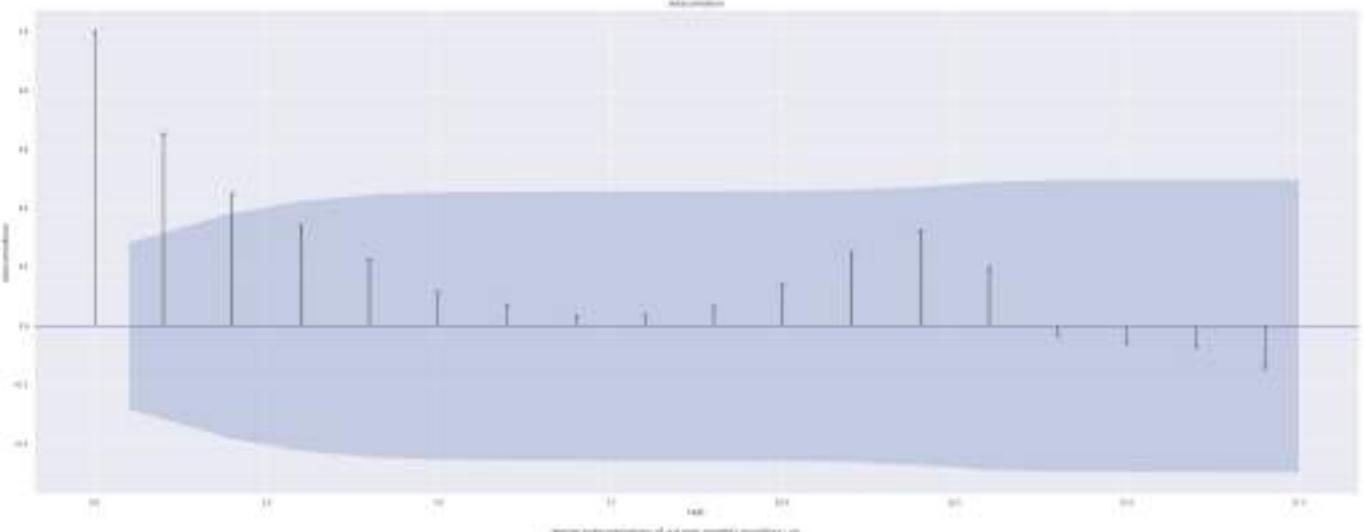
```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(FossilGas_Logpred)
plt.suptitle(" Autocorrelations of average monthly FossilGas Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(FossilGas_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly FossilGas Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
FossilGas_Log_Autocorrelations = sm.tsa.acf(FossilGas_Logpred, fft=False) #Autocorrelations
print(FossilGas_Log_Autocorrelations)

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,
```

```
[ 1.          0.65319276  0.44976412  0.34162297  0.2236438   0.11691654  
 0.06892985  0.031459    0.0376088   0.06525764  0.14004232  0.24787767  
 0.32215037  0.19682796  -0.03415765  -0.05727516  -0.06898208  -0.14317902  
-0.17487663  -0.20753704  -0.21773314  -0.19613855  -0.0833957   -0.00217226  
 0.02379421  -0.05080003  -0.12929471  -0.19943463  -0.14746262  -0.26193548  
-0.2757027   -0.22122515  -0.23096768  -0.18415209  -0.07537785  -0.00453934  
-0.00433728  0.01108662  -0.01876813  -0.03675073  -0.0214374 ]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

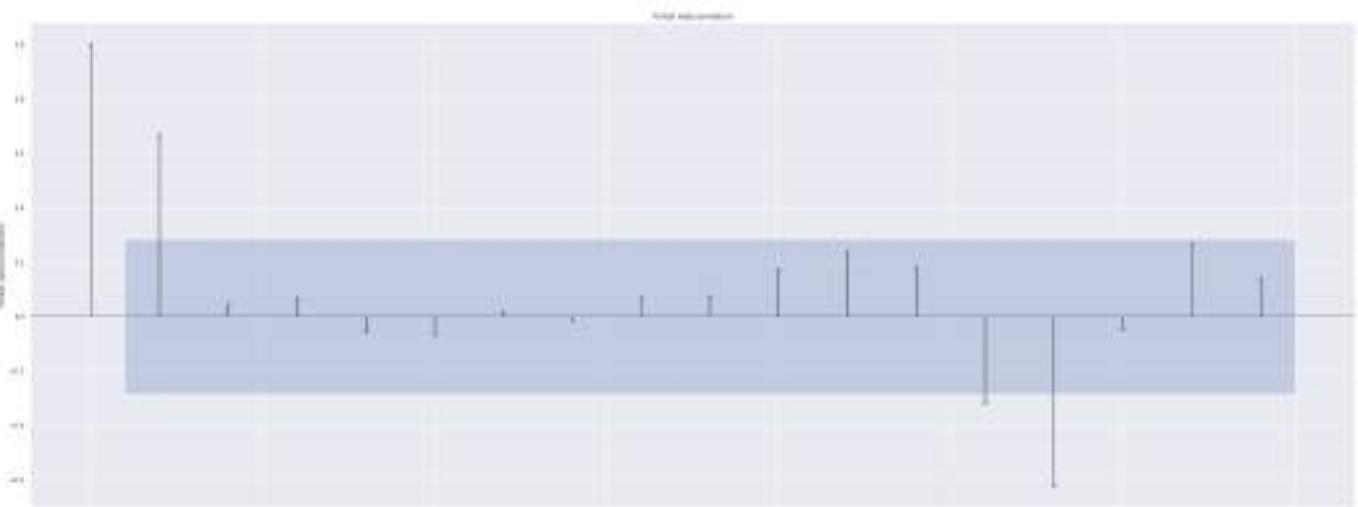
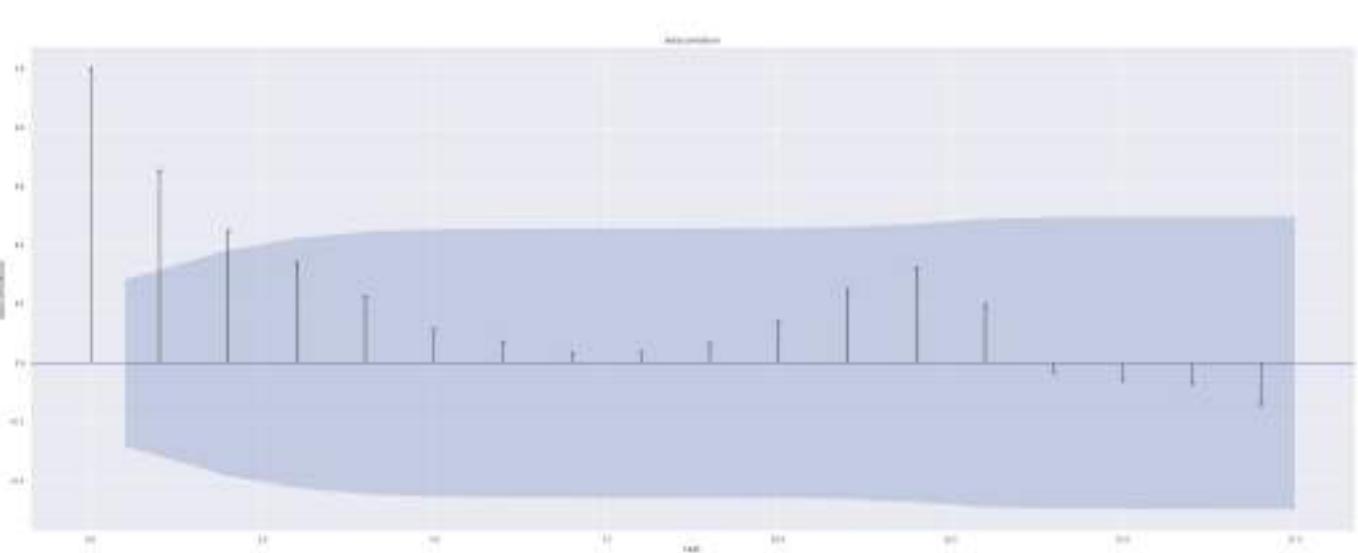
As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(predictionsFossilGas)  
plt.suptitle(" Autocorrelations of average monthly Linear FossilGas")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(predictionsFossilGas)  
plt.suptitle("Partial Autocorrelations of average monthly Linear FossilGas")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
FossilGas_Pred_Autocorrelations = sm.tsa.acf(predictionsFossilGas, fft=False) #Autocorrelations  
print(FossilGas_Pred_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.65319276  0.44976412  0.34162297  0.2236438   0.11691654  
 0.06892985  0.031459    0.0376088   0.06525764  0.14004232  0.24787767  
 0.32215037  0.19682796  -0.03415765 -0.05727516 -0.06898208 -0.14317902  
 -0.17487663 -0.20753704 -0.21773314 -0.19613855 -0.0833957  -0.00217226  
 0.02379421 -0.05080003 -0.12929471 -0.19943463 -0.14746262 -0.26193548  
 -0.2757027 -0.22122515 -0.23096768 -0.18415209 -0.07537785 -0.00453934  
 -0.00433728  0.01108662 -0.01876813 -0.03675073 -0.0214374 ]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

In []:

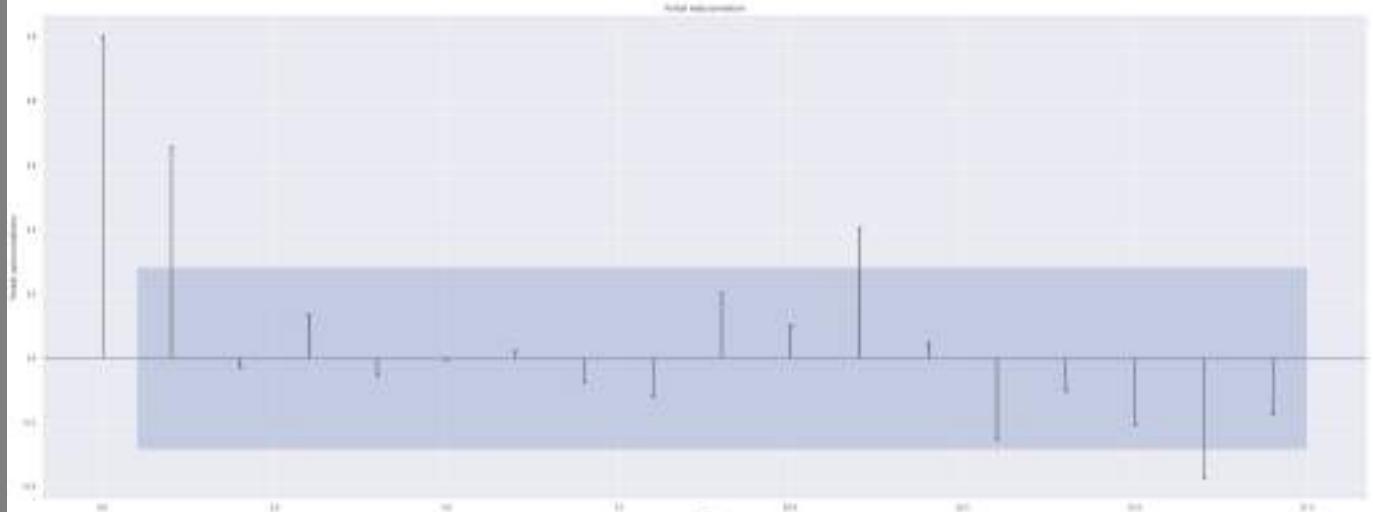
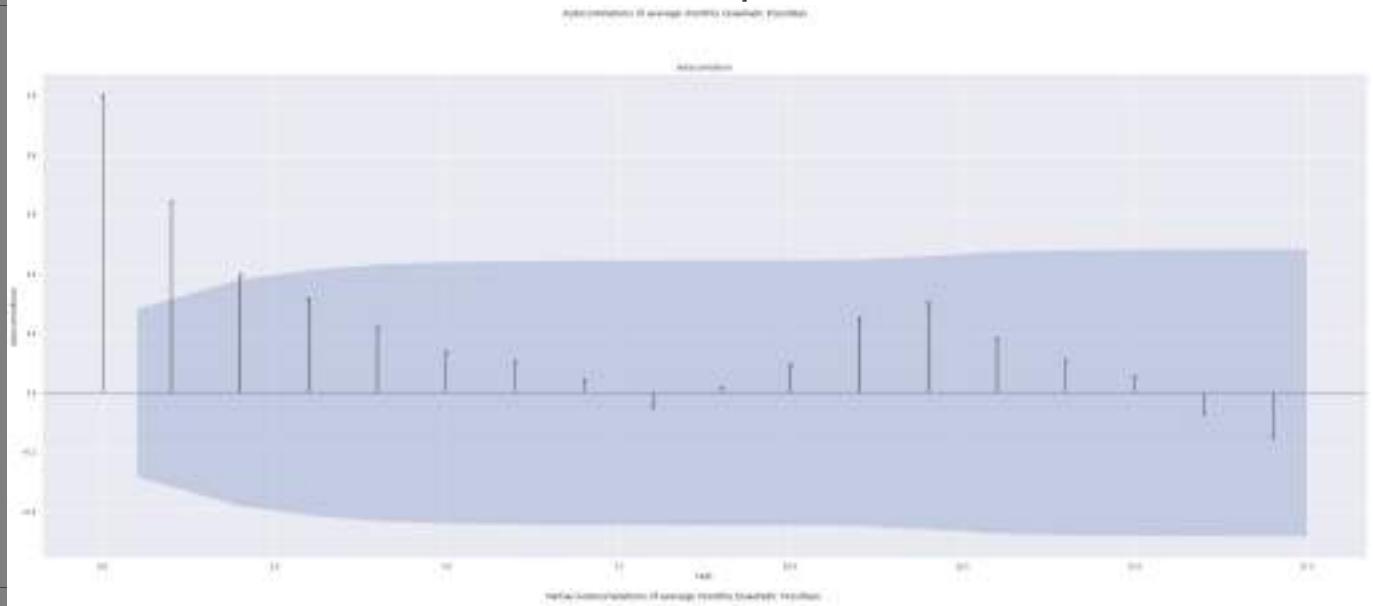
```
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(FossilGas_ypred)  
plt.suptitle(" Autocorrelations of average monthly Quadratic FossilGas")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(FossilGas_ypred)  
plt.suptitle("Partial Autocorrelations of average monthly Quadratic FossilGas")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
FossilGas_Quad_Autocorrelations = sm.tsa.acf(FossilGas_ypred, fft=False) #Autocorrelations
```

```
print(FossilGas_Quad_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statauto.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.6415085   0.39589386  0.31564628  0.2184889   0.13652071  
 0.10405536  0.04306162 -0.04835062  0.01602131  0.09417421  0.24966471  
 0.30103002  0.18192514  0.1099033   0.05436706 -0.07042142 -0.14940331  
-0.07130323 -0.03600684 -0.11657823 -0.12229943 -0.0669288   0.03668046  
 0.01395583 -0.03346119 -0.13599928 -0.24521648 -0.30262952 -0.27316782  
-0.2393731  -0.25481519 -0.22431912 -0.17550157 -0.04092259 -0.00506159  
-0.02762556  0.00321558 -0.0386826  -0.07549517 -0.08879854]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The next resource analyzed was biomass.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model for the average monthly biomass outputs versus the average monthly prices of energy per EUR/MWH.

In []:

```
In [ ]: modelbiomass = stats.linregress([483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 64.9490188172043,
```

```
56.383854166666666,  
55.522462987886975,  
58.354083333333333,  
57.29405913978498,  
65.9749027777778,  
71.07204301075271,  
63.99806451612899,  
60.254791666666634,  
59.40676510067113,  
60.72679166666668,  
61.901760752688226,  
45.57872311827956,  
36.75208333333374,  
36.81800807537014,  
32.618666666666666,  
34.691370967741896,  
46.266319444444434,  
47.50201612903221,  
47.6023387096774,  
50.4055972222224,  
60.182429530201404,  
62.58105555555558,  
67.5951344086021,  
79.4920833333331,  
59.83779761904767,  
50.95989232839837,  
51.71791666666662,  
53.77262096774189,  
56.2582222222224,  
55.252580645161316,  
54.08432795698931,  
55.81655555555558,  
63.92528859060403,  
65.43065277777781,  
65.15127688172035,  
56.51197580645163,  
60.877098214285674,  
48.279717362045766,  
50.40073611111113,  
61.633763440860214,  
64.3481388888884,  
67.78344086021498,  
70.36391129032262,  
76.9140416666666,  
70.36221476510062,  
67.0426075268817,  
66.62351388888881])
```

```
In [ ]: biomass1 = biomass  
biomass1 = sm.add_constant(biomass1)
```

```
In [ ]: #Dataframes analyzed by resource  
dfbiomass = {"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978498],  
"Biomass" : [483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 448.1063257065949],  
"Dates" : ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12']}  
  
print(dfbiomass)  
df_biomass= pd.DataFrame.from_dict (dfbiomass, orient = "columns")  
print(df_biomass)  
df_biomass[ "Ratio" ] = df_biomass[ "Biomass" ]/df_biomass[ "Price" ]  
pdToListBiomass = list(df_biomass[ "Ratio" ])  
print(pdToListBiomass)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.08432795698924, 55.81655555555556, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Biomass': [483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 485.99860917941584, 512.4879032258065, 518.9475806451613, 517.1069444444445, 519.5564516129032, 503.6166666666667, 483.92462987886944, 459.9959677419355, 430.3448275862069, 429.7442799461642, 286.8902777777778, 337.2029569892473, 340.3375, 351.8896366083446, 369.64516129032256, 349.3916666666665, 336.2187919463087, 355.57301808066757, 338.7768817204301, 347.28360215053766, 351.3645833333333, 284.40242261103634, 280.7611111111111, 353.5040322580645, 339.9791666666667, 367.7069892473118, 362.8790322580645, 351.96805555555557, 343.938255033557, 354.5930555555557, 347.1733870967742, 343.8481182795699, 360.51190476190476, 307.2543741588156, 299.3930555555556, 318.30913978494624, 351.73333333333335, 367.2207267833109, 361.3252688172043, 355.4680555555557, 317.9275167785235, 320.16129032258067, 342.10277777777776], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	Biomass	Dates
0	64.949019	483.735334	2015-01
1	56.383854	470.150298	2015-02
2	55.522463	468.106326	2015-03
3	58.354083	426.320334	2015-04
4	57.294059	503.569892	2015-05
5	65.974903	485.998609	2015-06
6	71.072043	512.487903	2015-07
7	63.998065	518.947581	2015-08
8	60.254792	517.106944	2015-09
9	59.406765	519.556452	2015-10
10	60.726792	503.616667	2015-11
11	61.901761	483.924630	2015-12
12	45.578723	459.995968	2016-01
13	36.752083	430.344828	2016-02
14	36.818008	429.744280	2016-03
15	32.618667	286.890278	2016-04
16	34.691371	337.202957	2016-05
17	46.266319	340.337500	2016-06
18	47.502016	351.889637	2016-07
19	47.602339	369.645161	2016-08
20	50.405597	349.391667	2016-09
21	60.182430	336.218792	2016-10
22	62.581056	355.573018	2016-11
23	67.595134	338.776882	2016-12
24	79.492083	347.283602	2017-01
25	59.837798	351.364583	2017-02
26	50.959892	284.402423	2017-03
27	51.717917	280.761111	2017-04
28	53.772621	353.504032	2017-05
29	56.258222	339.979167	2017-06
30	55.252581	367.706989	2017-07
31	54.084328	362.879032	2017-08
32	55.816556	351.968056	2017-09
33	63.925289	343.938255	2017-10
34	65.430653	354.593056	2017-11
35	65.151277	347.173387	2017-12
36	56.511976	343.848118	2018-01
37	60.877098	360.511905	2018-02
38	48.279717	307.254374	2018-03
39	50.400736	299.393056	2018-04
40	61.633763	318.309140	2018-05
41	64.348139	351.733333	2018-06
42	67.783441	367.220727	2018-07
43	70.363911	361.325269	2018-08
44	76.914042	355.468056	2018-09
45	70.362215	317.927517	2018-10
46	67.042608	320.161290	2018-11
47	66.623514	342.102778	2018-12

[7.447923664625113, 8.33838524463611, 8.430935886412662, 7.3057498277642114, 8.7892165441537, 7.366416451061661, 7.210822730229814, 8.108801173422584, 8.582005349966405, 8.74574555157917, 8.293154517878229, 7.817623020648154, 10.0923399400246, 11.709399537519378, 11.672121942785036, 8.795279117615, 9.720081610576829, 7.356053044346214, 7.407888449460503, 7.765273121237944, 6.931604542374731, 5.586660335432024, 5.681799626486198, 5.011853067301799, 4.368782243311914, 5.871950461316555, 5.580907054871221, 5.428701100252707, 6.574052480538948, 6.04319072372631, 6.655019276090836, 6.7095043234455165, 6.3058003499559065, 5.38031603167624, 5.419372121501836, 5.328727289981649, 6.084517721645735, 5.921962697579848, 6.3640466627991925, 5.9402516442523305, 5.164525448626468, 5.466099554808847, 5.417558066144856, 5.135093575545718, 4.621627570893987, 4.518441010418763, 4.775489828527437, 5.134865422263952]

In []: biomass1 = biomass
biomass1 = sm.add_constant(biomass1)

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the average monthly biomass outputs versus the predicted average monthly prices of energy per EUR/MWH/

In []:

```
#Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
Logbiomassvalues = ((np.log(biomass)))
Log = np.polyfit(np.log(Price_Actual), biomass1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(biomass1), Price_Actual)
Biomass_Log = sm.OLS(Price_Actual, biomass1).fit()

Biomass_Logpred = Biomass_Log.predict(biomass1)
#OLS Logarithmic summary table
Biomass_Log.summary()
#Log
Log = np.polyfit(np.log(biomass), Price_Actual, 1)
print(Log)

y = 7.82705763 * Logbiomassvalues + 11.41916835

#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly biomass outputs versus predicted average monthly prices of energy per EUR/MWH")
plt.title("R squared : 0.016")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(Logbiomassvalues, Price_Actual, "o")
plt.plot(Logbiomassvalues, y)

plt.xlim([1, 7])
```

[7.82705763 11.41916835]

Out[]:



The blue dots represent the observations and the orange line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

In []:

```
Biomass_Log.summary() #OLS Logarithmic summary table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.016			
Model:	OLS	Adj. R-squared:	-0.005			
Method:	Least Squares	F-statistic:	0.7590			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.388			
Time:	05:27:22	Log-Likelihood:	-179.25			
No. Observations:	48	AIC:	362.5			
Df Residuals:	46	BIC:	366.2			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	50.8159	8.222	6.181	0.000	34.266	67.366
x1	0.0184	0.021	0.871	0.388	-0.024	0.061
Omnibus:	2.246	Durbin-Watson:	0.406			
Prob(Omnibus):	0.325	Jarque-Bera (JB):	1.516			
Skew:	-0.422	Prob(JB):	0.469			
Kurtosis:	3.210	Cond. No.	2.15e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 [2] The condition number is large, 2.15e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceBiomassLog = Biomass_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsBiomassLog = influenceBiomassLog.resid_studentized_internal

print(standardized_residualsBiomassLog)

print(Biomass_Logpred) # OLS logarithmic predicted values
```

```
[ 0.52394805 -0.30436985 -0.38584876 -0.02855869 -0.27919725  0.62282967
 1.09854486  0.37136448 -0.00587868 -0.09674067  0.06671493  0.21945972
-1.35346218 -2.15551487 -2.14770726 -2.33865239 -2.18967662 -1.05900695
-0.95687564 -0.97729083 -0.66842858  0.31322718  0.51214007  1.03553984
 2.18391813  0.25146297 -0.50673602 -0.42512913 -0.3459625 -0.07859415
-0.22635153 -0.33197275 -0.14322127  0.66566452  0.79277165  0.77957338
-0.0606363  0.33638777 -0.80886367 -0.58643607  0.49005004  0.69225096
 0.9990681  1.26242915  1.91455248  1.35117106  0.97475661  0.97798938]
[ 59.69978787 59.45029685 59.41275903 58.64535369 60.06405219 59.74135322
 60.22783266 60.3464655 60.31266197 60.3576475 60.06491121 59.70326431
 59.26381122 58.71926405 58.70823491 56.08470199 57.0087011 57.0662674
 57.27842394 57.60450653 57.23254838 56.99062677 57.34606973 57.03760643
 57.1938335 57.26878126 56.0390122 55.97213902 57.30807253 57.05968656
 57.56891174 57.48024566 57.27986411 57.13239575 57.32807259 57.19180938
 57.13074037 57.43677305 56.45869135 56.31431718 56.66171361 57.27555341
 57.55998147 57.45171059 57.34414208 56.65470505 57.09868696 56.6957286 ]
```

In []:

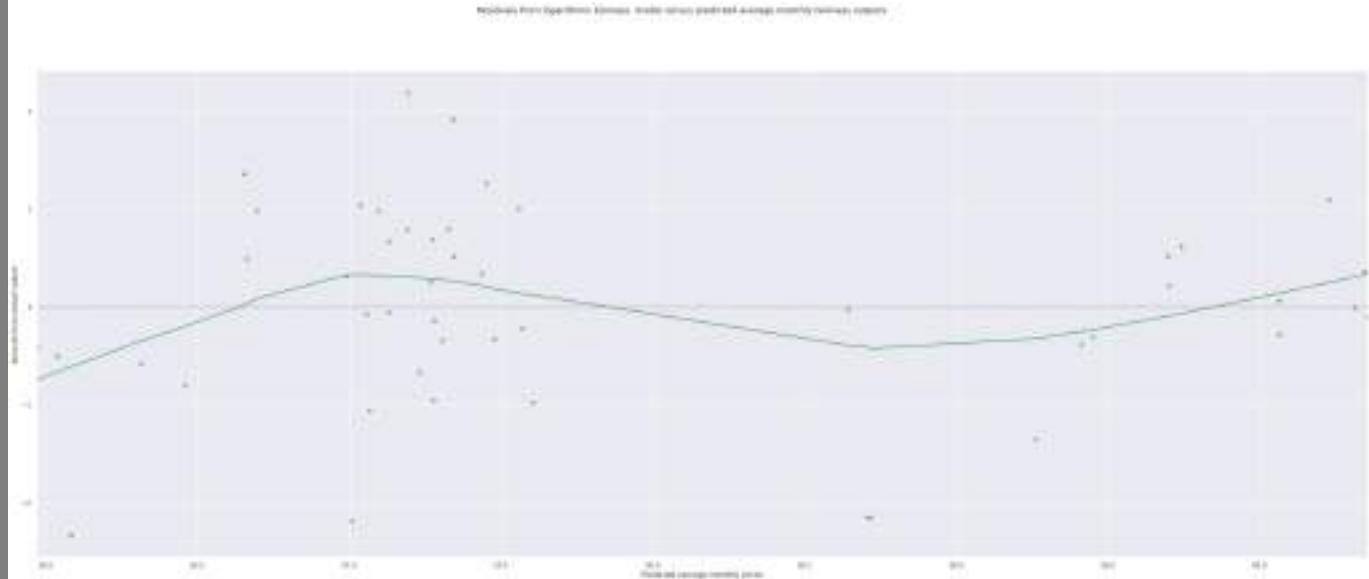
```
BiomassLogRatioPredict = Biomass_Logpred/Logpred
```

In []:

```
plt.suptitle("Residuals from logarithmic biomass model versus predicted average monthly biomass outputs")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = Biomass_Logpred, y = standardized_residualsBiomassLog, lowess = True, color="g")
# OLS Logarithmic average monthly predictions versus residuals
```

Out[]:

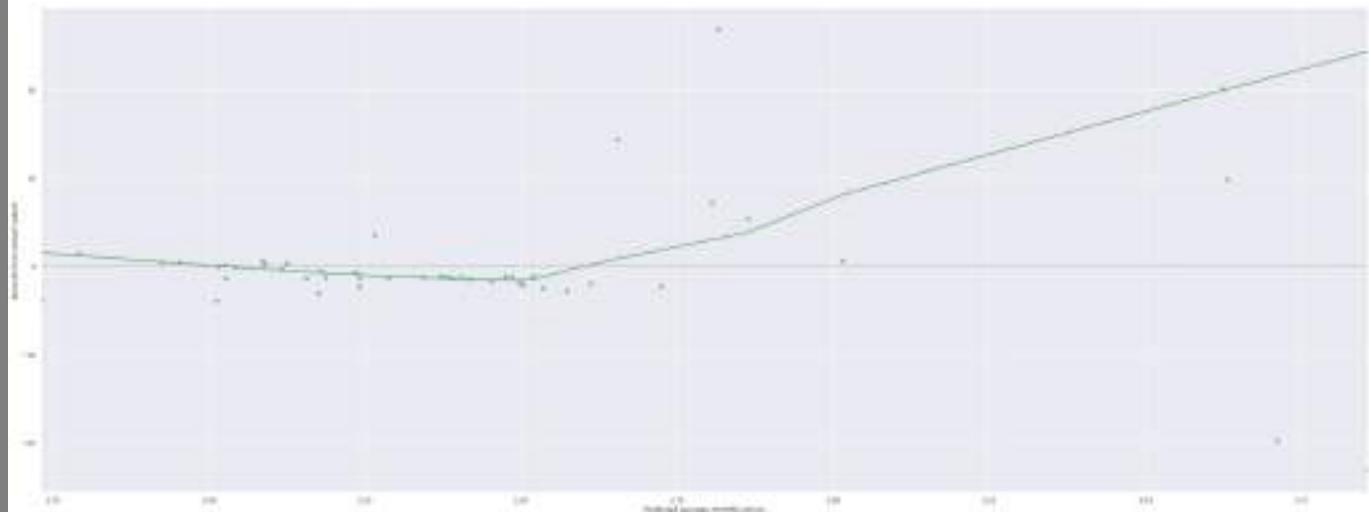
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff607abcc90>
```



As one can observe this residual plot, one may notice a clustured hump in the fitted model, which form a nonlinear pattern. However, the residuals are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
plt.suptitle("Predicted average monthly logarithmic biomass output to price of energy per EUR/MWh ratio versus  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Amount from actual values")  
plt.legend(..#)  
sns.residplot(x = BiomassLogRatioPredict, y = standardized_residualsBiomassLog/standardized_residualsPriceLog,
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6080d0610>
```

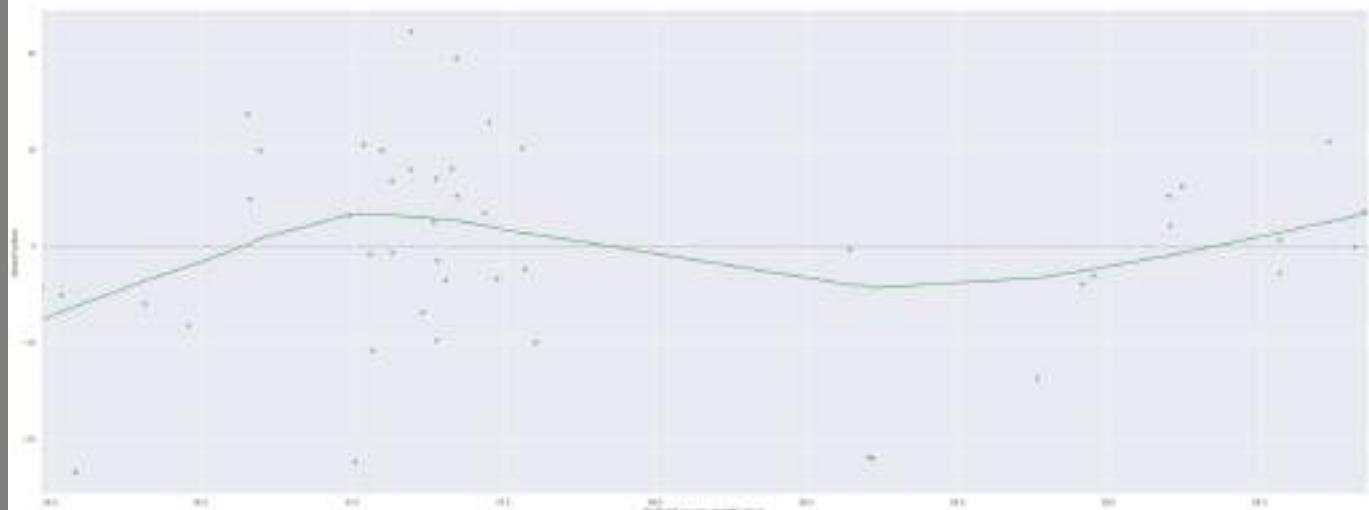


```
In [ ]:
```

With the exception of a few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly logarithmic biomass energy prices per EUR/MWh versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend(..#)  
sns.residplot(x = Biomass_Logpred, y = Price_Actual, lowess = True, color="g")  
# OLS predicted logarithmic average monthly values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608781e90>
```



As one can observe this residual plot, one may notice a clustured hump in the fitted model, which form a nonlinear pattern. However, the residuals are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: influenceBiomassLog = Biomass_Log.get_influence()  
#Logarithmic OLS regression residuals
```



```
standardized_residualsBiomassLog = influenceBiomassLog.resid_studentized_internal
```



```
print(standardized_residualsBiomassLog)
```

```
[ 0.52394805 -0.30436985 -0.38584876 -0.02855869 -0.27919725  0.62282967  
 1.09854486  0.37136448 -0.00587868 -0.09674067  0.06671493  0.21945972  
 -1.35346218 -2.15551487 -2.14770726 -2.33865239 -2.18967662 -1.05900695  
 -0.95687564 -0.97729083 -0.66842858  0.31322718  0.51214007  1.03553984  
  2.18391813  0.25146297 -0.50673602 -0.42512913 -0.3459625 -0.07859415  
 -0.22635153 -0.33197275 -0.14322127  0.66566452  0.79277165  0.77957338  
 -0.0606363  0.33638777 -0.80886367 -0.58643607  0.49005004  0.69225096  
  0.9990681   1.26242915  1.91455248  1.35117106  0.97475661  0.97798938]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model used for the average monthly biomass output versus the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Linear OLS regression  
biomass1 = biomass  
biomass1 = sm.add_constant(biomass1)  
modelbiomassreg = sm.OLS(Price_Actual, biomass1).fit()  
predictionsBiomass = modelbiomassreg.predict(biomass1)
```



```
modelbiomassreg.summary()  
#OLS Linear Summary Table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.016			
Model:	OLS	Adj. R-squared:	-0.005			
Method:	Least Squares	F-statistic:	0.7590			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.388			
Time:	05:27:24	Log-Likelihood:	-179.25			
No. Observations:	48	AIC:	362.5			
Df Residuals:	46	BIC:	366.2			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	50.8159	8.222	6.181	0.000	34.266	67.366
x1	0.0184	0.021	0.871	0.388	-0.024	0.061
Omnibus:	2.246	Durbin-Watson:	0.406			
Prob(Omnibus):	0.325	Jarque-Bera (JB):	1.516			
Skew:	-0.422	Prob(JB):	0.469			
Kurtosis:	3.210	Cond. No.	2.15e+03			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.15e+03. This might indicate that there are strong multicollinearity or other numerical problems.

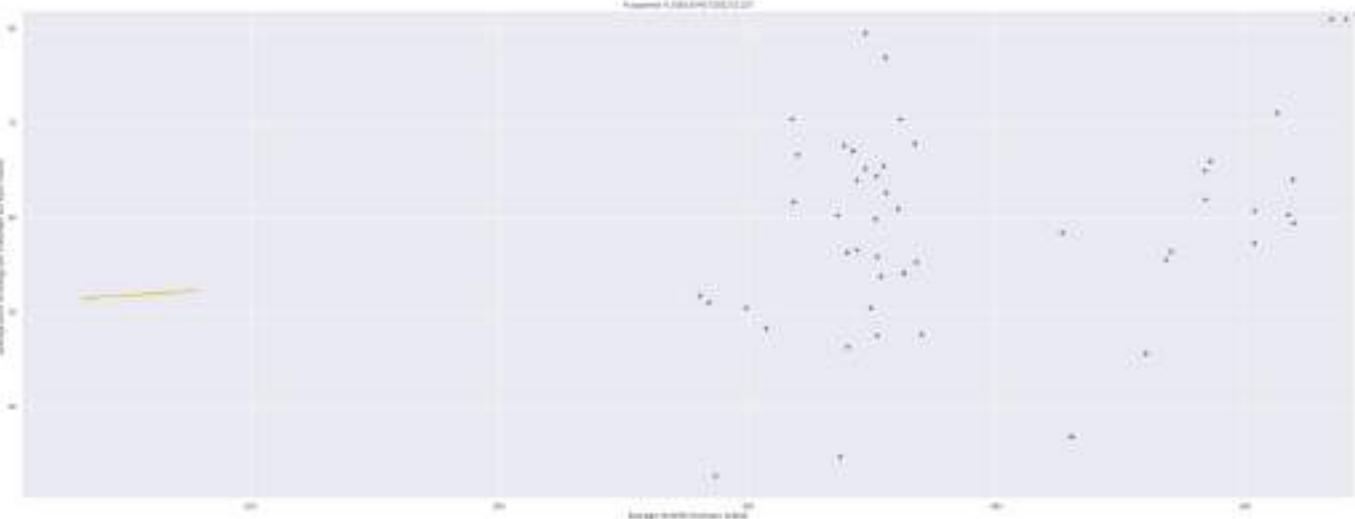
In []:

```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(biomass,Price_Actual)
print("slope: %f      intercept: %f" %(slope, intercept))

#OLS Linear Scatterplot
plt.plot(biomass,Price_Actual, "o")
plt.title(f"R squared: {modelbiomass.rvalue**2}")
f = lambda x: 0.018365 *x + 50.815924
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average biomass monthly outputs versus predicted linear average monthly prices of energy per EUR/I")
plt.legend("#")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly biomass output')
plt.show()
```

slope: 0.018365 intercept: 50.815924

Average monthly biomass output versus predicted linear average monthly prices of energy per EUR/I



There is a very weak correlation between the outputs and their respective the average monthly prices of energy per EUR/MWH. The blue dots are the observations and orange line is the model of best fit.

In []:

```
#Linear OLS regression residuals
influenceBiomass = modelbiomassreg.get_influence()
```

```
standardized_residualsBiomass = influenceBiomass.resid_studentized_internal
```

```
print(standardized_residualsBiomass)
```

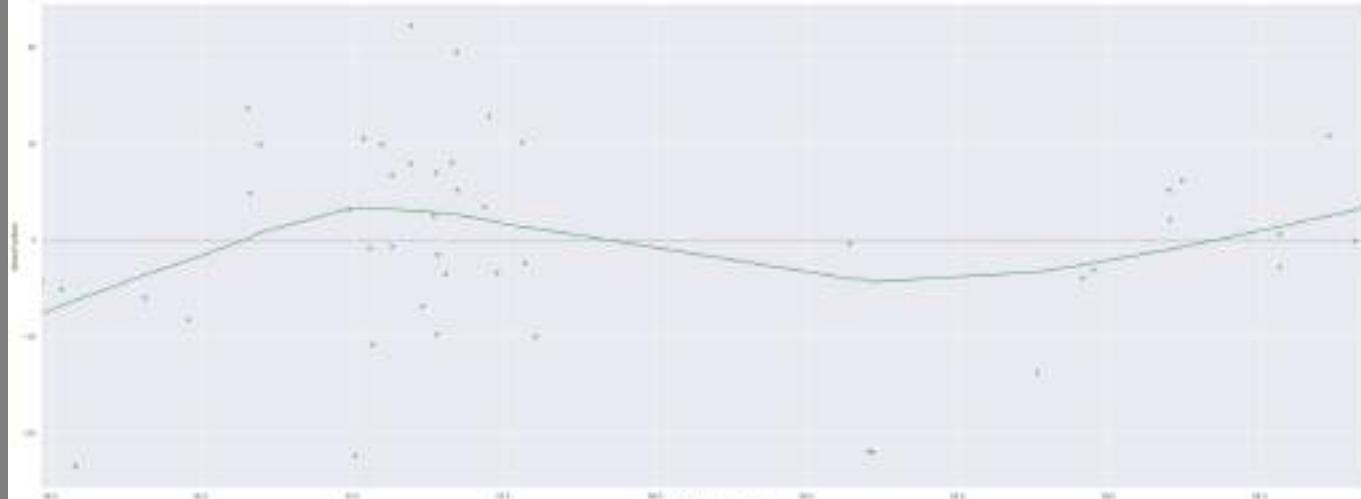
```
[ 0.52394805 -0.30436985 -0.38584876 -0.02855869 -0.27919725  0.62282967  
 1.09854486  0.37136448 -0.00587868 -0.09674067  0.06671493  0.21945972  
 -1.35346218 -2.15551487 -2.14770726 -2.33865239 -2.18967662 -1.05900695  
 -0.95687564 -0.97729083 -0.66842858  0.31322718  0.51214007  1.03553984  
 2.18391813  0.25146297 -0.50673602 -0.42512913 -0.3459625 -0.07859415  
 -0.22635153 -0.33197275 -0.14322127  0.66566452  0.79277165  0.77957338  
 -0.0606363  0.33638777 -0.80886367 -0.58643607  0.49005004  0.69225096  
 0.9990681  1.26242915  1.91455248  1.35117106  0.97475661  0.97798938]
```

```
In [ ]: print(predictionsBiomass)  
#Linear OLS Predicted Values
```

```
[59.69978787 59.45029685 59.41275903 58.64535369 60.06405219 59.74135322  
60.22783266 60.3464655 60.31266197 60.3576475 60.06491121 59.70326431  
59.26381122 58.71926405 58.70823491 56.08470199 57.0087011 57.0662674  
57.27842394 57.60450653 57.23254838 56.99062677 57.34606973 57.03760643  
57.1938335 57.26878126 56.0390122 55.97213902 57.30807253 57.05968656  
57.56891174 57.48024566 57.27986411 57.13239575 57.32807259 57.19180938  
57.13074037 57.43677305 56.45869135 56.31431718 56.66171361 57.27555341  
57.55998147 57.45171059 57.34414208 56.65470505 57.09868696 56.6957286 ]
```

```
In [ ]: plt.suptitle("Predicted average monthly linear biomass energy prices per EUR/MWh versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend(..#)  
sns.residplot(x = predictionsBiomass, y = Price_Actual, lowess = True, color="g")  
#Predicted OLS average monthly linear values versus actual values
```

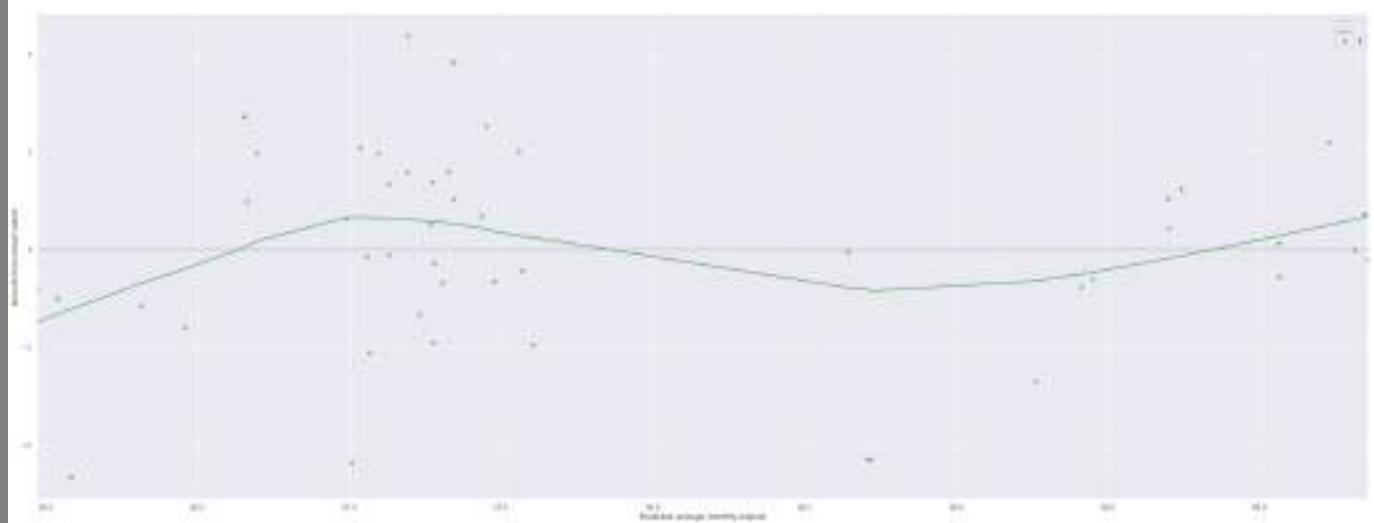
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608b0f350>
```



As one can observe this residual plot, one may notice a clustured hump in the fitted model, which form a nonlinear pattern. However, the observations are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #Predicted OLS linear values versus residual values  
sns.residplot(x = predictionsBiomass, y = standardized_residualsBiomass, lowess = True, color ="g")  
  
plt.suptitle("Biomass residuals from linear model versus predicted values")  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Amount from actual values")  
  
plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff606c66190>
```



As one can observe this residual plot, one may notice a clustered hump in the fitted model, which form a nonlinear pattern. However, the residuals are spread out without a distinct pattern, indicating constant variance, a lack of bias and homoscedasticity. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []:

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly biomass output versus the predicted average monthly prices of energy per EUR/MWH.

In []:

```
#Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)

modelBiomassquad = np.poly1d(np.polyfit(biomass,Price_Actual,2))
print(modelBiomassquad)

biomass1 = sm.add_constant(biomass)
biomass2 = polynomial_features.fit_transform(biomass1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(biomass1)

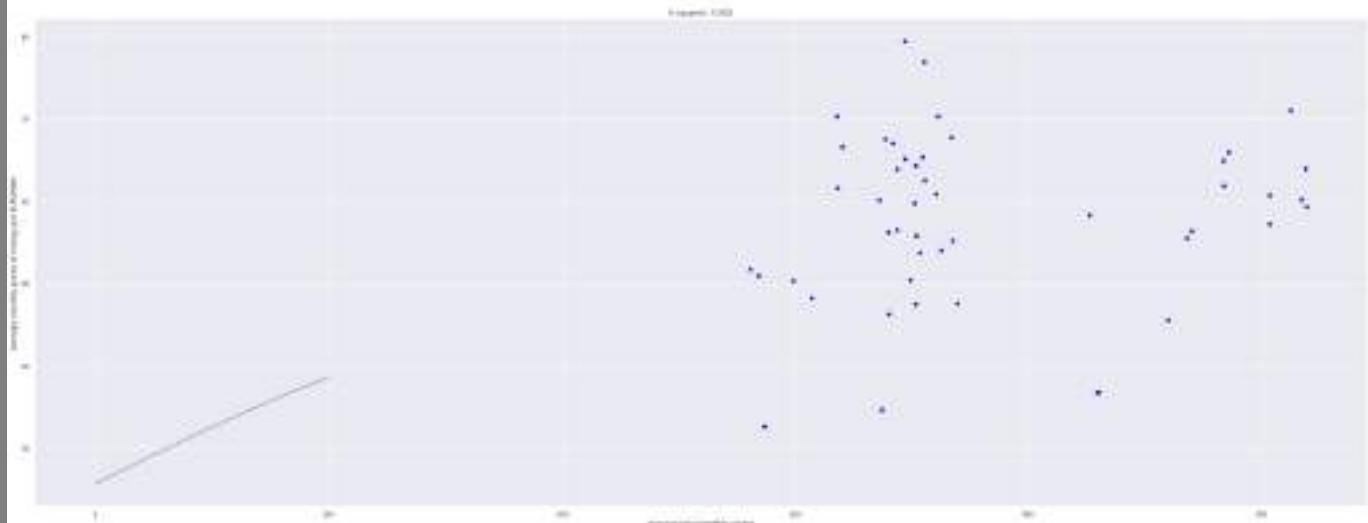
Biomass_Q = poly.fit(X_poly, Price_Actual)
lin2 = LinearRegression()
lin2.fit(X_poly, Price_Actual)
Biomass_Quad = sm.OLS(Price_Actual, biomass2).fit()

# OLS Predicted Quadratic values
Biomass_ypred = Biomass_Quad.predict(biomass2)

#OLS Quadratic Summary Table
Biomass_Quad.summary()

##OLS Quadratic Scatterplot
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelBiomassquad(polyline))
plt.scatter(biomass,Price_Actual, color = 'blue')
plt.title("R squared : 0.002")
plt.suptitle('Quadratic for average total monthly biomass outputs versus predicted average monthly prices of energy')
plt.xlabel('Average total monthly output')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()
```

```
2  
-0.0001558 x + 0.1458 x + 25.66
```



The blue dots represent the observations and the blue line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

```
In [ ]: Biomass_Quad.summary()
```

```
Out[ ]: OLS Regression Results
```

Dep. Variable:	y	R-squared:	0.020			
Model:	OLS	Adj. R-squared:	-0.023			
Method:	Least Squares	F-statistic:	0.4622			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.633			
Time:	05:27:27	Log-Likelihood:	-179.15			
No. Observations:	48	AIC:	364.3			
Df Residuals:	45	BIC:	369.9			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	8.5523	20.021	0.427	0.671	-31.772	48.876
x1	8.5523	20.021	0.427	0.671	-31.772	48.876
x2	0.0729	0.151	0.483	0.632	-0.231	0.377
x3	8.5523	20.021	0.427	0.671	-31.772	48.876
x4	0.0729	0.151	0.483	0.632	-0.231	0.377
x5	-0.0002	0.000	-0.423	0.674	-0.001	0.001
Omnibus:	2.556	Durbin-Watson:	0.397			
Prob(Omnibus):	0.279	Jarque-Bera (JB):	1.785			
Skew:	-0.460	Prob(JB):	0.410			
Kurtosis:	3.219	Cond. No.	8.32e+22			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 1.84e-34. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

```
In [ ]: influenceBiomassquad = Biomass_Quad.get_influence() #Quadratic OLS residuals
```

```
standardized_residualsBiomassQuad = influenceBiomassquad.resid_studentized_internal
```

```
print(standardized_residualsBiomassQuad)
```

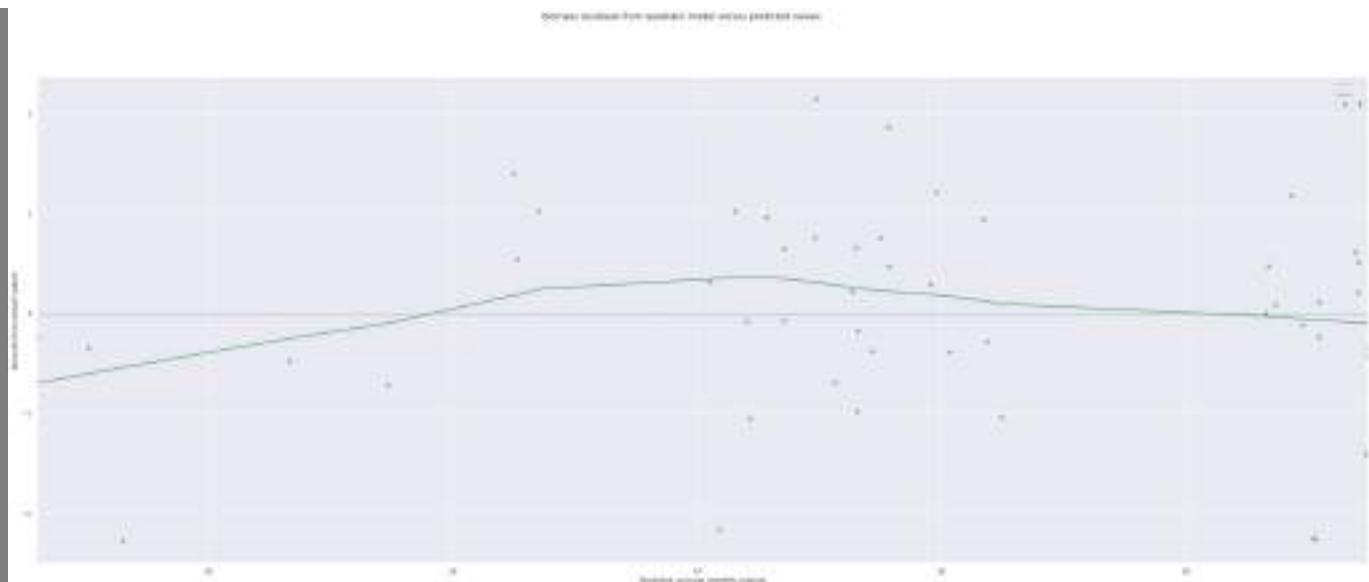
```
[ 0.51852817 -0.33151027 -0.41648415 -0.11139889 -0.22675004  0.6218687  
 1.18951974  0.48376965  0.09167913  0.00810062  0.11879037  0.21720662  
-1.39618827 -2.25429351 -2.24718855 -2.30948082 -2.17844716 -1.06480774  
-0.98851572 -1.04207302 -0.69659441  0.30458637  0.46728994  1.01523804  
2.14083808  0.2143496  -0.37574905 -0.27779057 -0.38377365 -0.09185082  
-0.28708201 -0.38564642 -0.17927209  0.63864124  0.74814372  0.74613943  
-0.08193724  0.28519733 -0.74103723 -0.49764237  0.52642344  0.65218846  
 0.9401223   1.20869461  1.86434049  1.38478091  0.94863865  1.00588786]
```

```
In [ ]: print(Biomass_ypred) # OLS quadratic predicted values
```

```
[59.70709903 59.74596551 59.74683606 59.47928122 59.547076  59.6950353  
59.43517583 59.33864519 59.36747551 59.32887593 59.54655374 59.70615121  
59.73745907 59.52872767 59.52166949 54.64981479 57.09108953 57.21708172  
57.65498291 58.24695663 57.56381742 57.0508997 57.7858636 57.15473575  
57.48536901 57.63598205 54.50863249 54.29851588 57.71286725 57.20283359  
58.18711393 58.03295709 57.6578134 57.35803461 57.7514556 57.48122944  
57.35455547 57.95472101 55.73294386 55.33012842 56.26681146 57.64933552  
58.17191649 57.98180046 57.78219251 56.24901636 57.286689 56.35253275]
```

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals  
sns.residplot(x = Biomass_ypred, y = standardized_residualsBiomassQuad, lowess = True, color="g")  
plt.suptitle("Biomass residuals from quadratic model versus predicted values")  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Amount from actual values")  
plt.legend("..#")
```

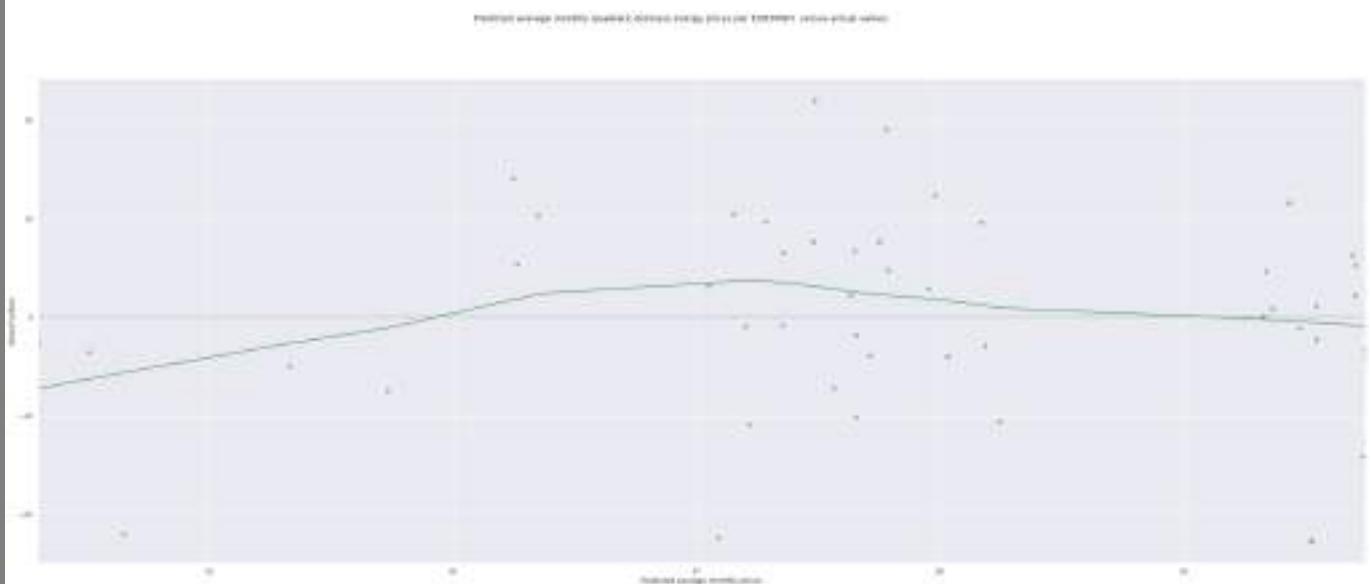
```
Out[ ]: <matplotlib.legend.Legend at 0x7ff607687e90>
```



As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. Given these circumstances, it would be reasonable to assume that the quadratic model of fit is suitable. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly quadratic biomass energy prices per EUR/MWh versus actual values")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Actual values")  
plt.legend("..#")  
sns.residplot(x = Biomass_ypred, y = Price_Actual, lowess = True, color="g")  
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff607634a50>
```



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective average monthly prices of energy per EUR/MWH.

```
In [ ]: dfbiomass = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139, "Biomass" : [483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, "Dates" : ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10']})
```

```
In [ ]: #Dataframes analyzed by resource
dfbiomass = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139, "Biomass" : [483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, "Dates" : ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10']]})
print(dfbiomass)
df_biomass= pd.DataFrame.from_dict(dfbiomass, orient = "columns")
print(df_biomass)
df_biomass["Ratio"] = df_biomass["Biomass"]/df_biomass["Price"]
pdToListBiomass = list(df_biomass["Ratio"])
print(pdToListBiomass)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.095989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Biomass': [483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 485.99860917941584, 512.4879032258065, 518.9475806451613, 517.1069444444445, 519.5564516129032, 503.6166666666667, 483.92462987886944, 459.9959677419355, 430.3448275862069, 429.7442799461642, 286.8902777777778, 337.2029569892473, 340.3375, 351.8896366083446, 369.64516129032256, 349.39166666666665, 336.2187919463087, 355.57301808066757, 338.7768817204301, 347.28360215053766, 351.3645833333333, 284.40242261103634, 280.7611111111111, 353.5040322580645, 339.9791666666667, 367.7069892473118, 362.8790322580645, 351.96805555555557, 343.938255033557, 354.59305555555557, 347.1733870967742, 343.8481182795699, 360.51190476190476, 307.2543741588156, 299.3930555555556, 318.30913978494624, 351.73333333333335, 367.2207267833109, 361.3252688172043, 355.4680555555557, 317.9275167785235, 320.16129032258067, 342.10277777777776], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}
```

	Price	Biomass	Dates
0	64.949019	483.735334	2015-01
1	56.383854	470.150298	2015-02
2	55.522463	468.106326	2015-03
3	58.354083	426.320334	2015-04
4	57.294059	503.569892	2015-05
5	65.974903	485.998609	2015-06
6	71.072043	512.487903	2015-07
7	63.998065	518.947581	2015-08
8	60.254792	517.106944	2015-09
9	59.406765	519.556452	2015-10
10	60.726792	503.616667	2015-11
11	61.901761	483.924630	2015-12
12	45.578723	459.995968	2016-01
13	36.752083	430.344828	2016-02
14	36.818008	429.744280	2016-03
15	32.618667	286.890278	2016-04
16	34.691371	337.202957	2016-05
17	46.266319	340.337500	2016-06
18	47.502016	351.889637	2016-07
19	47.602339	369.645161	2016-08
20	50.405597	349.391667	2016-09
21	60.182430	336.218792	2016-10
22	62.581056	355.573018	2016-11
23	67.595134	338.776882	2016-12
24	79.492083	347.283602	2017-01
25	59.837798	351.364583	2017-02
26	50.0959892	284.402423	2017-03
27	51.717917	280.761111	2017-04
28	53.772621	353.504032	2017-05
29	56.258222	339.979167	2017-06
30	55.252581	367.706989	2017-07
31	54.084328	362.879032	2017-08
32	55.816556	351.968056	2017-09
33	63.925289	343.938255	2017-10
34	65.430653	354.593056	2017-11
35	65.151277	347.173387	2017-12
36	56.511976	343.848118	2018-01
37	60.877098	360.511905	2018-02
38	48.279717	307.254374	2018-03
39	50.400736	299.393056	2018-04
40	61.633763	318.309140	2018-05
41	64.348139	351.733333	2018-06
42	67.783441	367.220727	2018-07
43	70.363911	361.325269	2018-08
44	76.914042	355.468056	2018-09
45	70.362215	317.927517	2018-10
46	67.042608	320.161290	2018-11
47	66.623514	342.102778	2018-12

[7.447923664625113, 8.33838524463611, 8.430935886412662, 7.3057498277642114, 8.7892165441537, 7.366416451061661, 7.210822730229814, 8.108801173422584, 8.582005349966405, 8.74574555157917, 8.293154517878229, 7.817623020648154, 10.0923399400246, 11.709399537519378, 11.672121942785036, 8.795279117615, 9.720081610576829, 7.356053044346214, 7.407888449460503, 7.765273121237944, 6.931604542374731, 5.586660335432024, 5.681799626486198, 5.011853067301799, 4.368782243311914, 5.871950461316555, 5.580907054871221, 5.428701100252707, 6.574052480538948, 6.04319072372631, 6.655019276090836, 6.7095043234455165, 6.3058003499559065, 5.38031603167624, 5.419372121501836, 5.328727289981649, 6.084517721645735, 5.921962697579848, 6.3640466627991925, 5.9402516442523305, 5.164525448626468, 5.466099554808847, 5.417558066144856, 5.135093575545718, 4.621627570893987, 4.518441010418763, 4.775489828527437, 5.134865422263952]

In []:

In []:

#Bell Curves

```
biomassResults_mean = np.mean(df_biomass["Ratio"])
biomassResults_std = np.std(df_biomass["Ratio"])

biomassResultspdf = stats.norm.pdf(df_biomass["Ratio"].sort_values(), biomassResults_mean, biomassResults_std)

plt.plot(df_biomass["Ratio"].sort_values(), biomassResultspdf)
plt.xlim([0,20])
plt.xlabel("Output to energy price per EUR/MWh", size=15)
plt.suptitle("Frequency distribution of average monthly biomass outputs to energy prices per EUR/MWh ratios (scaled in decimals) ")
plt.title(f'Skewness for data: {skew(df_biomass["Ratio"])}') # Output/Price per EUR/MWh Bell Curve
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves

biomassResults_mean = np.mean(df_biomass["Biomass"])
biomassResults_std = np.std(df_biomass["Biomass"])

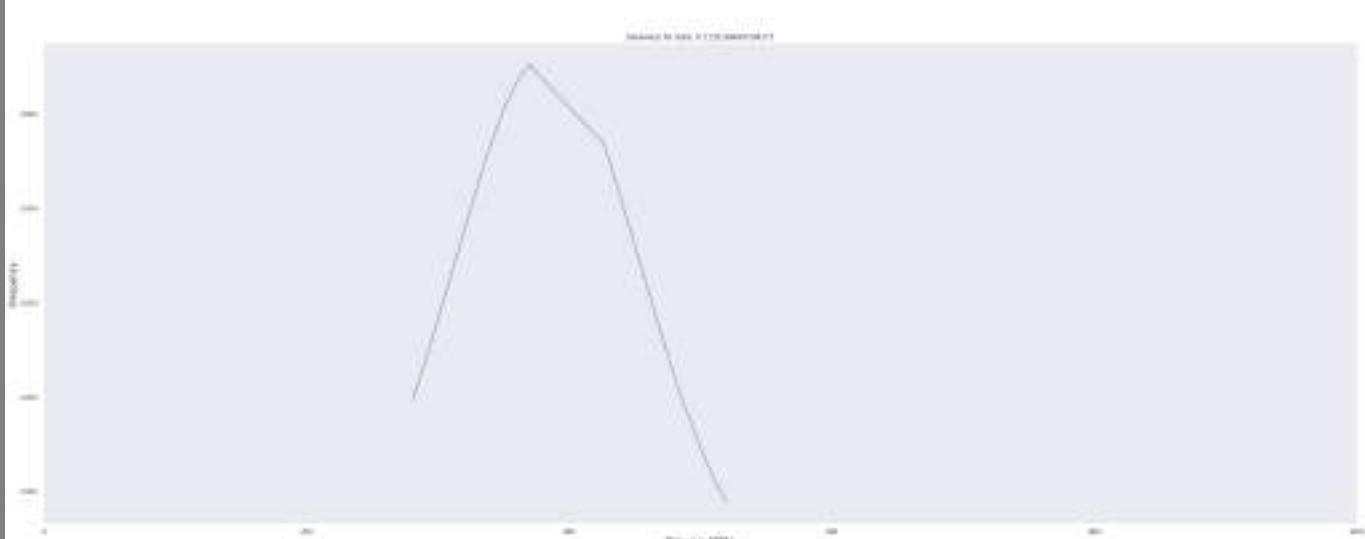
biomassResultspdf = stats.norm.pdf(df_biomass["Biomass"].sort_values(), biomassResults_mean, biomassResults_std)

plt.plot(df_biomass["Biomass"].sort_values(), biomassResultspdf)
plt.xlim([0,1000])
plt.xlabel("Output in MWh ", size=15)
plt.title(f'Skewness for data: {skew(df_biomass["Biomass"])}')
plt.suptitle("Frequency distribution of average monthly biomass outputs (scaled in decimals) ")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

Frequency distribution of average monthly biomass outputs to energy prices per EUR/MWh ratios (scaled in decimals)



Frequency distribution of average monthly biomass outputs (scaled in decimals)



These bell shaped curves are respectively skewed to the right, hence they have an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

```
In [ ]: #ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

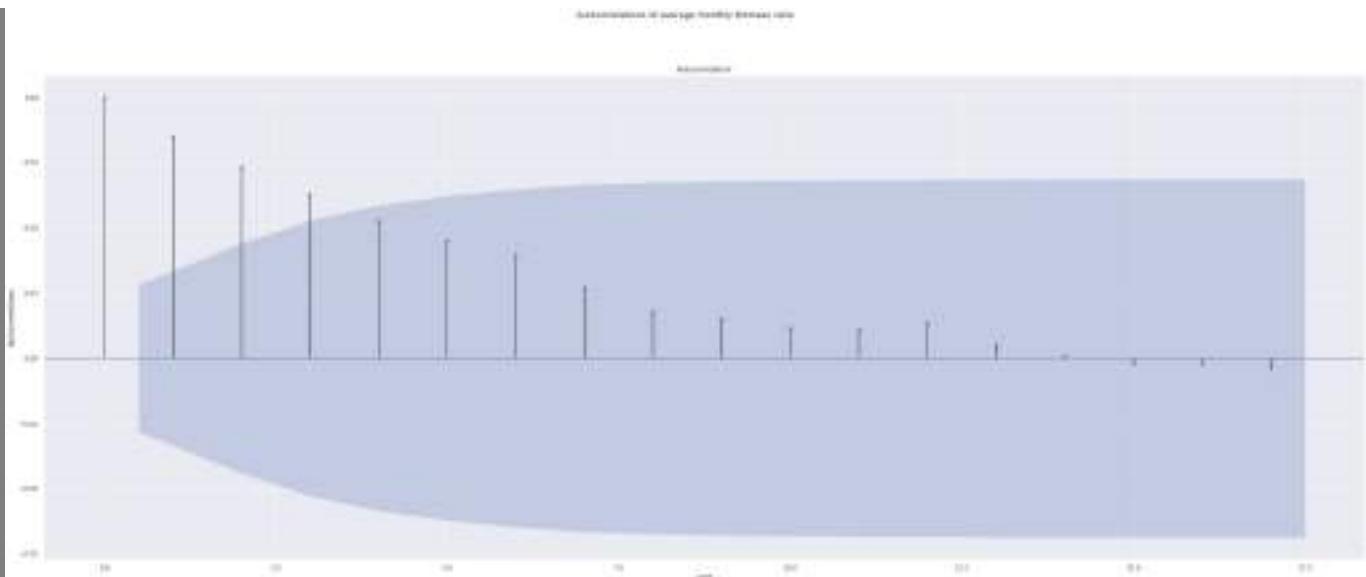
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_biomass["Ratio"])
```

```
ADF Test Statistic : -1.7447944180577406
p-value : 0.4081932381126042
#Lags Used : 0
Number of Observations : 47
weak evidence against null hypothesis,indicating it is non-stationary
```

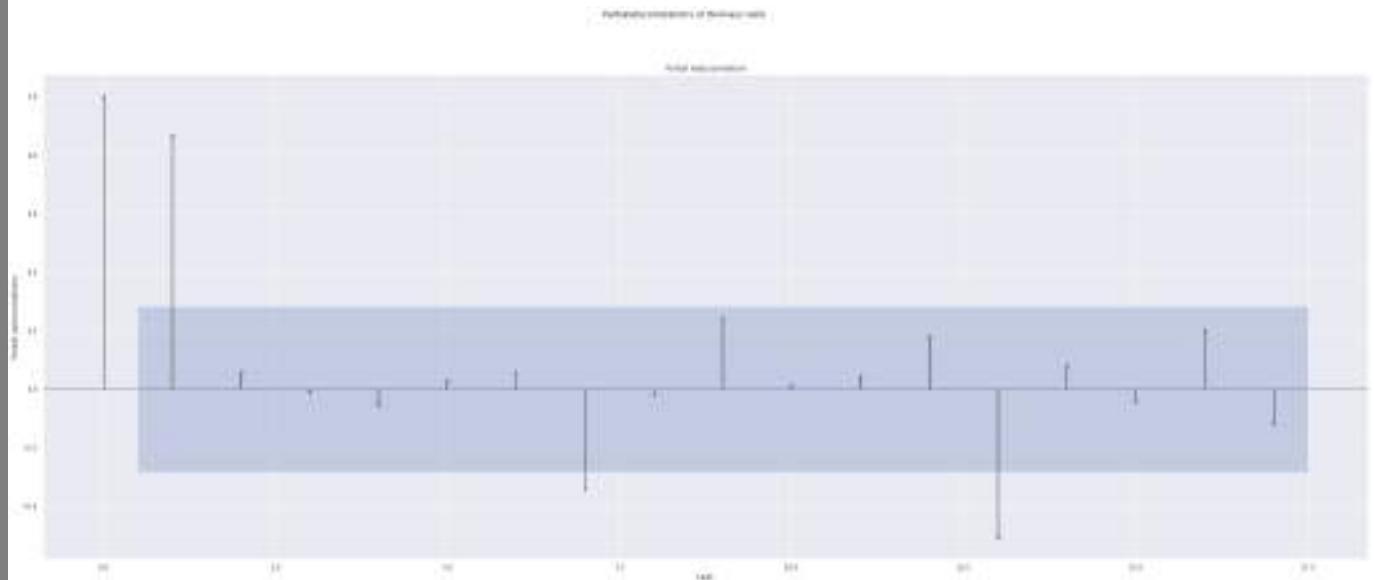
```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_biomass["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Biomass ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
```

```
Out[ ]: <function matplotlib.pyplot.show(*args, **kw)>
```



```
In [ ]: from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_biomass["Ratio"])
plt.suptitle("Partialatocorrelations of Biomass ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
```

```
Out[ ]: <function matplotlib.pyplot.show(*args, **kw)>
```



```
In [ ]: Biomass_Ratio_Autocorrelations = sm.tsa.acf(df_biomass["Ratio"],fft=False) #Autocorrelations
print(Biomass_Ratio_Autocorrelations)
```

```
[ 1.          0.84700432  0.73086405  0.62752773  0.5259759   0.44819678
 0.39235177  0.26891938  0.17809225  0.14828163  0.1156693   0.10748695
 0.13259865  0.04902368  0.00740504 -0.01886405 -0.02139365 -0.03416107
-0.02910695 -0.08660819 -0.12199358 -0.13856272 -0.1424951  -0.14601922
-0.16934916 -0.21707157 -0.25103337 -0.28292488 -0.29648583 -0.30461142
-0.33001364 -0.36171523 -0.34141242 -0.31411144 -0.23804922 -0.18362971
-0.15504984 -0.14249325 -0.13162904 -0.11532126 -0.0999894 ]
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
FutureWarning,

```
In [ ]:
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partialautocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial and autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly biomass outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: df_biomass['First Difference Ratio'] = df_biomass["Ratio"]- df_biomass["Ratio"].shift(1) # Seasonality values
df_biomass['Seasonal Difference Ratio']=df_biomass["Ratio"]- df_biomass["Ratio"].shift(12)
df_biomass.head()
```

```
Out[ ]:
```

	Price	Biomass	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
0	64.949019	483.735334	2015-01	7.447924	NaN	NaN
1	56.383854	470.150298	2015-02	8.338385	0.890462	NaN
2	55.522463	468.106326	2015-03	8.430936	0.092551	NaN
3	58.354083	426.320334	2015-04	7.305750	-1.125186	NaN
4	57.294059	503.569892	2015-05	8.789217	1.483467	NaN

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly Biomass output to price of energy per EUR/MWH ratio")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_biomass['Seasonal Difference Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff608501150>
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly biomass outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

```
In [ ]:
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_biomass["Biomass"])

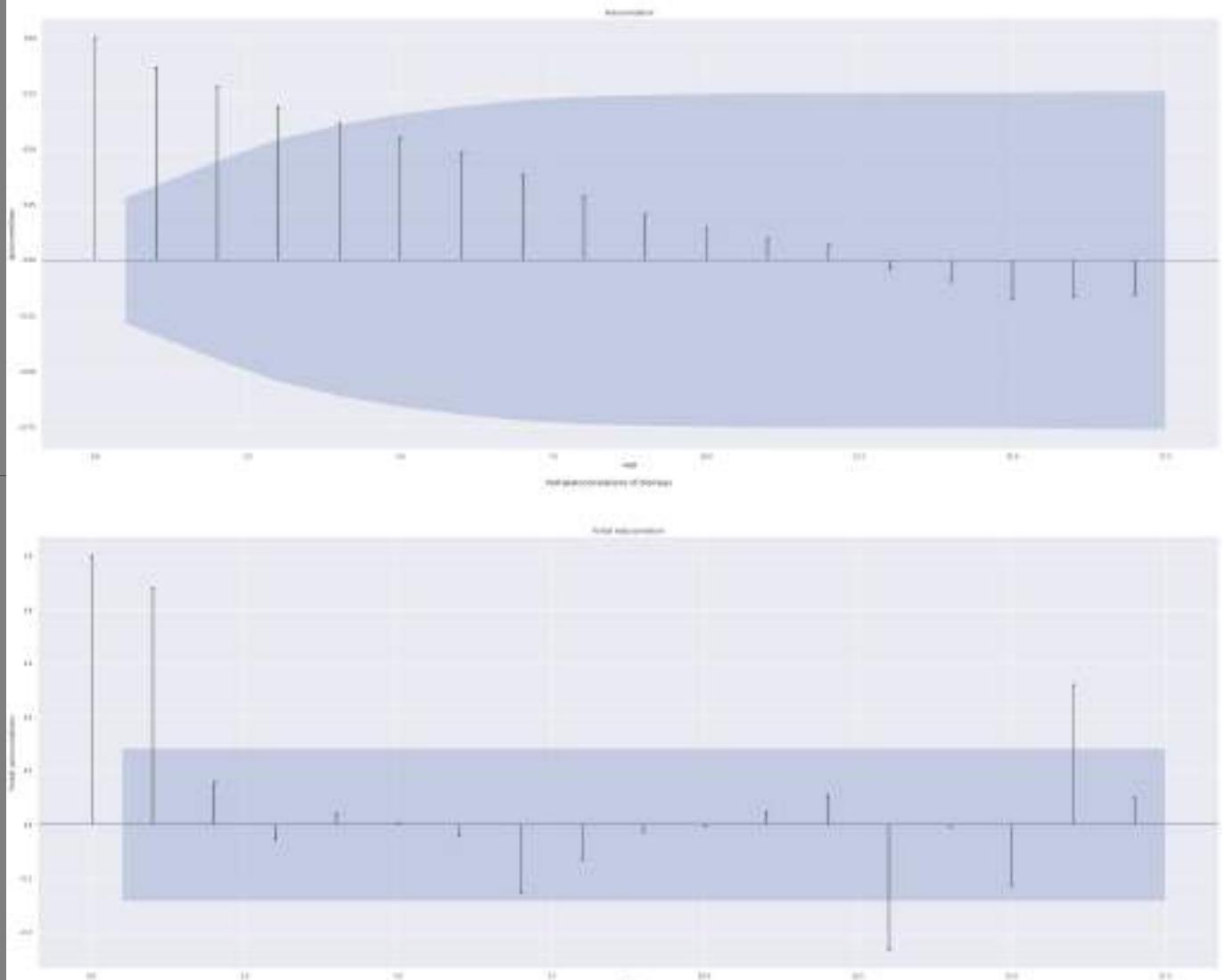
test_result=adfuller(df_biomass["Biomass"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_biomass["Biomass"])
plt.suptitle(" Autocorrelations of average monthly Biomass")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_biomass["Biomass"])
plt.suptitle("Partialatocorrelations of Biomass")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Biomass_Autocorrelations = sm.tsa.acf(df_biomass["Biomass"], fft=False) #Autocorrelations
print(Biomass_Autocorrelations)
```

```
ADF Test Statistic : -1.909143367076302
p-value : 0.32777208337117325
#Lags Used : 0
Number of Observations : 47
weak evidence against null hypothesis, indicating it is non-stationary
[ 1.         0.8640293   0.77963191   0.68649065   0.61616632   0.54976886
 0.48405604   0.38133082   0.28725742   0.20628506   0.14213191   0.09651022
 0.06847825   -0.03470059   -0.09286495   -0.172417   -0.16179856   -0.15534682
 -0.13447794   -0.14188615   -0.15304689   -0.1639042   -0.16974473   -0.14357734
 -0.14704562   -0.19257578   -0.22682116   -0.23012931   -0.20718581   -0.21226837
 -0.20070417   -0.22700787   -0.25037915   -0.2630094   -0.26080536   -0.24731425
 -0.23521725   -0.22139096   -0.21853487   -0.18922881   -0.14734125]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
  FutureWarning,
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partialautocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial and autocorrelation plot, indicating that the lags directly beforehand influenced the average monthly biomass outputs.

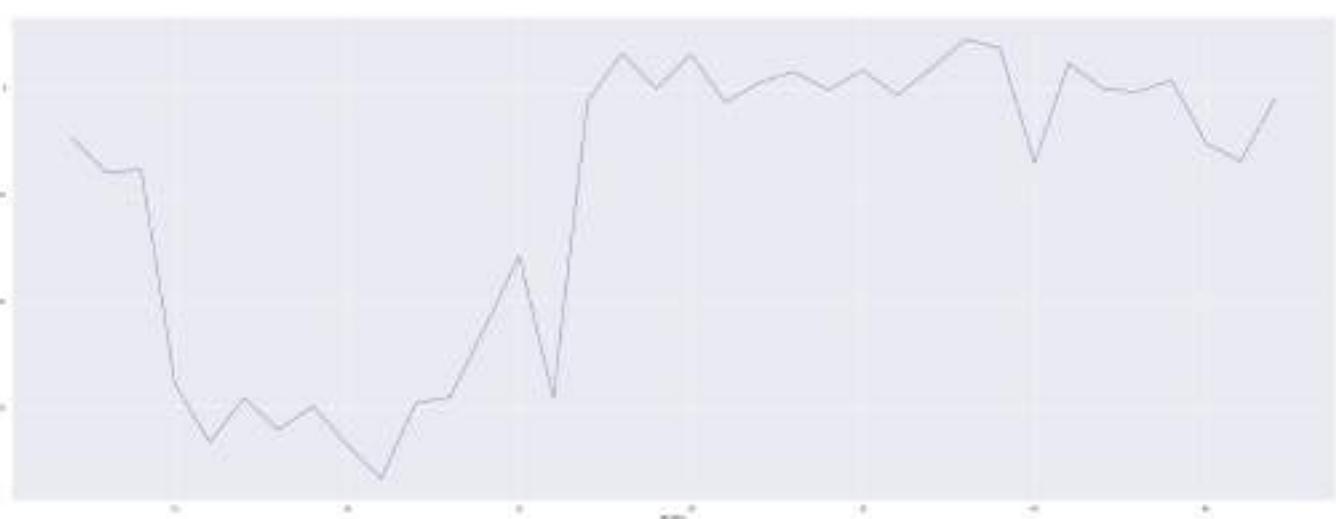
```
In [ ]: df_biomass['First Difference'] = df_biomass["Biomass"] - df_biomass["Biomass"].shift(1) # Seasonality values
df_biomass['Seasonal Difference']=df_biomass["Biomass"]- df_biomass["Biomass"].shift(12)
df_biomass.head()
```

	Price	Biomass	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	First Difference	Seasonal Difference
0	64.949019	483.735334	2015-01	7.447924	NaN	NaN	NaN	NaN
1	56.383854	470.150298	2015-02	8.338385	0.890462	NaN	-13.585037	NaN
2	55.522463	468.106326	2015-03	8.430936	0.092551	NaN	-2.043972	NaN
3	58.354083	426.320334	2015-04	7.305750	-1.125186	NaN	-41.785991	NaN
4	57.294059	503.569892	2015-05	8.789217	1.483467	NaN	77.249558	NaN

```
In [ ]:
```

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly Biomass output to price of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_biomass['Seasonal Difference'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60a930b90>
```



The blue line represents the trend line among the values themselves. As one can observe, there are patterns depicted in the average monthly biomass outputs.

```
In [ ]: print(biomass)
```

```
[483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 485.99860917941584, 512.4879032258065, 518.9475806451613, 517.1069444444445, 519.5564516129032, 503.6166666666667, 483.92462987886944, 459.9959677419355, 430.3448275862069, 429.7442799461642, 286.8902777777778, 337.2029569892473, 340.3375, 351.8896366083446, 369.64516129032256, 349.39166666666665, 336.2187919463087, 355.57301808066757, 338.7768817204301, 347.28360215053766, 351.3645833333333, 284.40242261103634, 280.761111111111, 353.5040322580645, 339.9791666666667, 367.7069892473118, 362.8790322580645, 351.96805555555557, 343.938255033557, 354.59305555555557, 347.1733870967742, 343.8481182795699, 360.51190476190476, 307.2543741588156, 299.3930555555556, 318.30913978494624, 351.7333333333335, 367.2207267833109, 361.3252688172043, 355.4680555555557, 317.9275167785235, 342.10277777777776, 320.16129032258067]
```

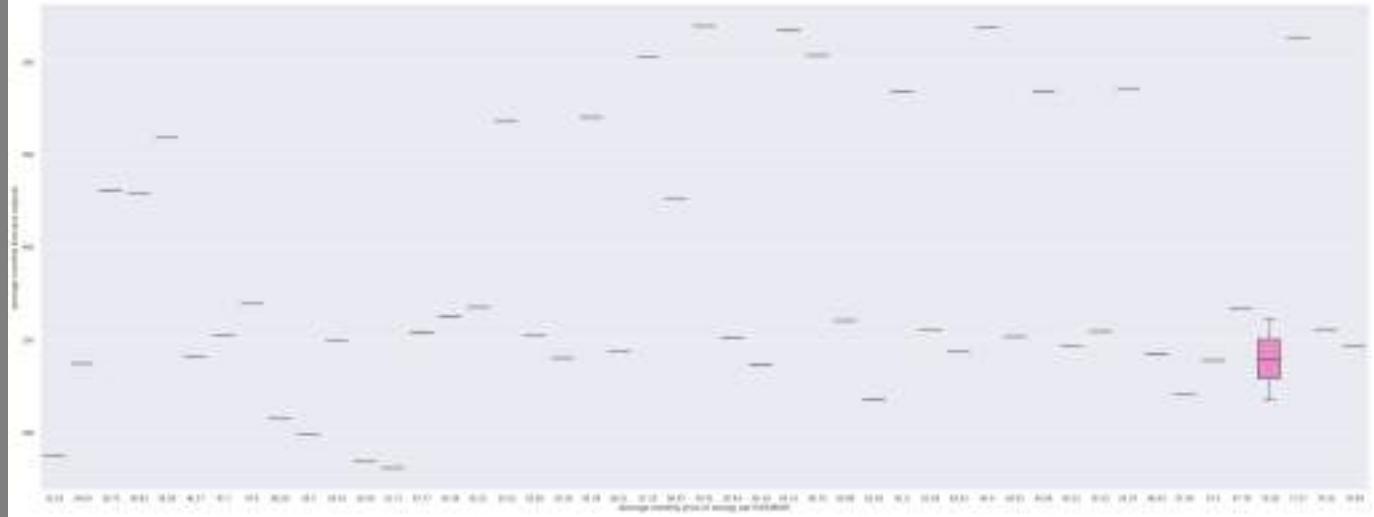
```
In [ ]: df_biomass.describe(include = 'all') # Description Tables
```

	Price	Biomass	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN	NaN
mean	57.859848	383.548775	NaN	6.841207	-0.049214	-0.885897	-3.013459	-51.340706
std	10.320573	71.598992	NaN	1.762708	0.947446	2.233551	34.052407	70.694693
min	32.618667	280.761111	NaN	4.368782	-2.876843	-6.091215	-142.854002	-183.337660
25%	51.528411	340.247917	NaN	5.426369	-0.494499	-1.658844	-14.762411	-140.859479
50%	59.622281	354.048544	NaN	6.469050	-0.048541	-0.634843	-3.325269	-15.252748
75%	64.999583	437.757613	NaN	7.890418	0.400730	0.227018	13.519765	2.807292
max	79.492083	519.556452	NaN	11.709400	2.274717	3.371014	77.249558	22.851952

Below is the box and whisker plot for the distribution of the average monthly outputs of biomass and its the average monthly prices of energy per EUR/MWH.

```
In [ ]: # Box and Whisker Plot
sns.set(style="darkgrid")

plt.suptitle('Distribution of average monthly biomass outputs by average monthly price of energy per EUR/MWH ')
plt.ylabel('Average monthly biomass outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')
sns.boxplot(x=Rounded_Y, y=[483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183])
plt.show()
```



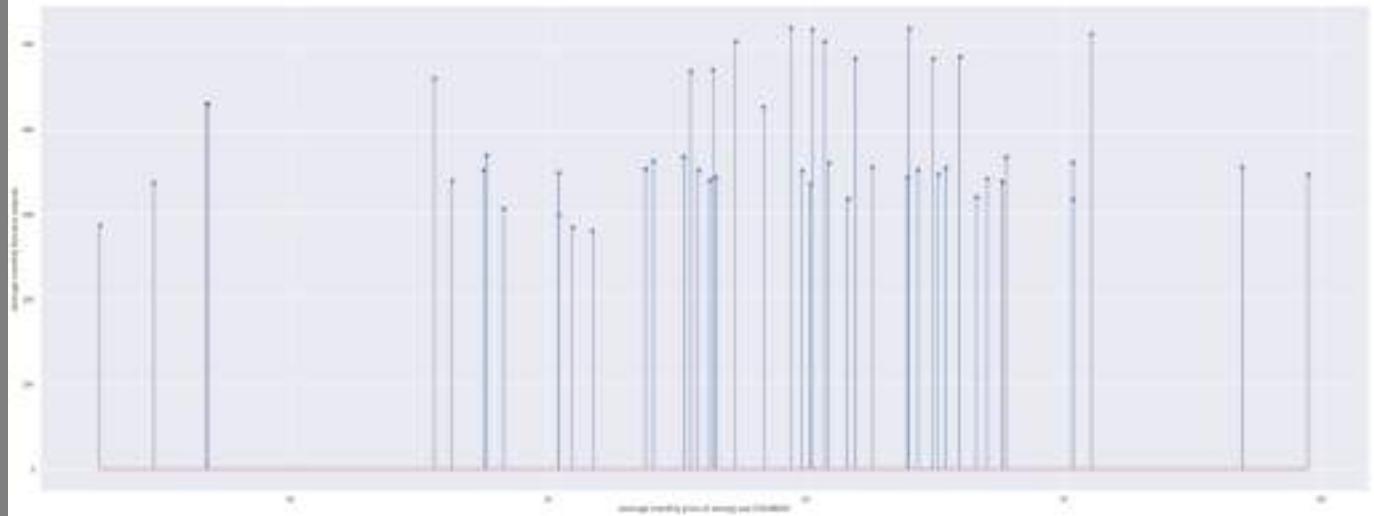
This box and whisker plot depicts the frequencies between the distribution of the monthly average output of biomass produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of biomass produced was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH,which was slightly above and below 350 units. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), biomass output was slightly above 250 units.The lowest amount produced, which was slightly above 200 units, was at the price of approximately 51.72 EUR/MWH. At approximately 59.41 EUR/MWH, biomass produced the highest output amount,which was roughly 550 units. When the monthly average price of energy per EUR/MWH was at its highest, at approximently 79.13 EUR/MWH, the output was at roughly 350 units.

```
In [ ]: plt.suptitle('Distribution of average monthly biomass outputs by average monthly price of energy per EUR/MWH ')
plt.ylabel('Average monthly biomass outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

# Stem Plot
plt.stem(Rounded_Y, biomass)
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

```
Out[ ]: <StemContainer object of 3 artists>
```



This plot depicts the frequencies between the distribution of the monthly average output of biomass produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of biomass produced was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH,which was slightly above and below 350 units. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), biomass output was slightly above 250 units.The lowest amount produced, which was slightly above 200 units, was at the price of approximately 51.72 EUR/MWH. At approximately 59.41 EUR/MWH, biomass produced the highest output amount,which was roughly 550 units. When the monthly average price of energy per EUR/MWH was at its highest, at approximently 79.13 EUR/MWH, the output was at roughly 350 units. Each blue dot at end of the blue lines represent an observation.

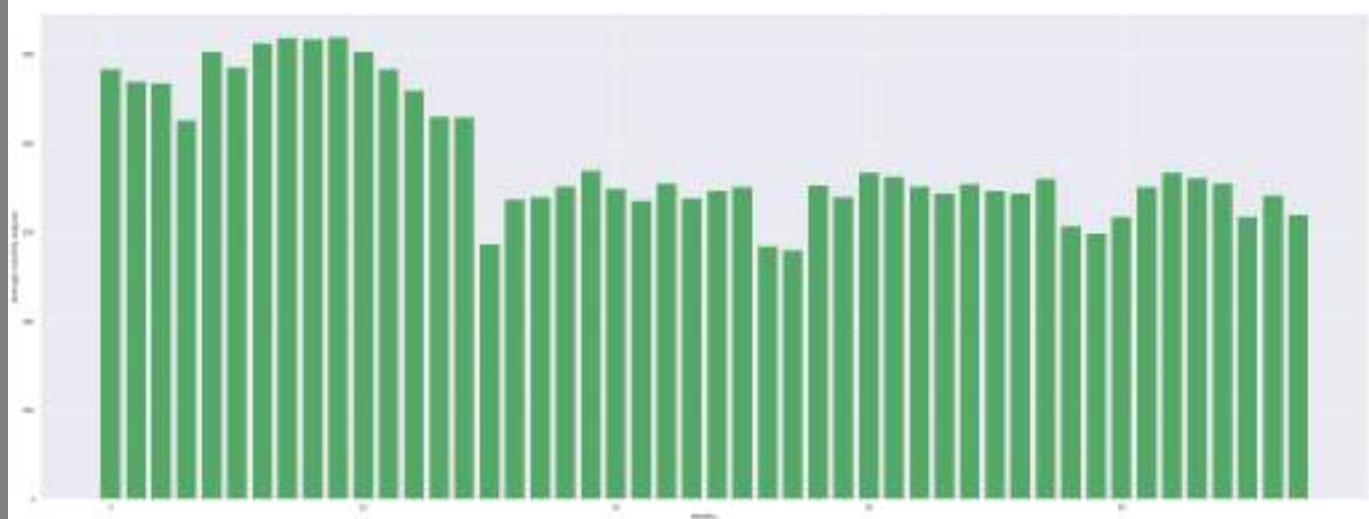
```
In [ ]: #Outputs in MWH Histogram
Biomass_Dict = {key: i for i, key in enumerate(biomass)}

def Hist_Biomass(Biomass_Dict):
    for k, v in Biomass_Dict.items(): print(f"{v}:{k}")
print(Biomass_Dict)

plt.bar(list(Biomass_Dict.values()), Biomass_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of biomas")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{483.73533424283767: 0, 470.1502976190476: 1, 468.1063257065949: 2, 426.32033426183847: 3, 503.5698924731183: 4, 485.99860917941584: 5, 512.4879032258065: 6, 518.9475806451613: 7, 517.1069444444445: 8, 519.5564516129032: 9, 503.61666666666667: 10, 483.92462987886944: 11, 459.9959677419355: 12, 430.3448275862069: 13, 429.7442799461642: 14, 286.890277777778: 15, 337.2029569892473: 16, 340.3375: 17, 351.8896366083446: 18, 369.64516129032256: 19, 349.39166666666665: 20, 336.2187919463087: 21, 355.57301808066757: 22, 338.7768817204301: 23, 347.28360215053766: 24, 351.3645833333333: 25, 284.40242261103634: 26, 280.761111111111: 27, 353.5040322580645: 28, 339.97916666666667: 29, 367.7069892473118: 30, 362.8790322580645: 31, 351.96805555555557: 32, 343.938255033557: 33, 354.593055555557: 34, 347.1733870967742: 35, 343.8481182795699: 36, 360.51190476190476: 37, 307.2543741588156: 38, 299.393055555556: 39, 318.30913978494624: 40, 351.73333333333335: 41, 367.2207267833109: 42, 361.3252688172043: 43, 355.4680555555557: 44, 317.9275167785235: 45, 342.10277777777776: 46, 320.16129032258067: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is skewed to the right, indicating that there was an external factor decreasing the average monthly outputs. Aside from the skewness on the left side of the histogram, the histogram is roughly uniform to the right of the skewness.

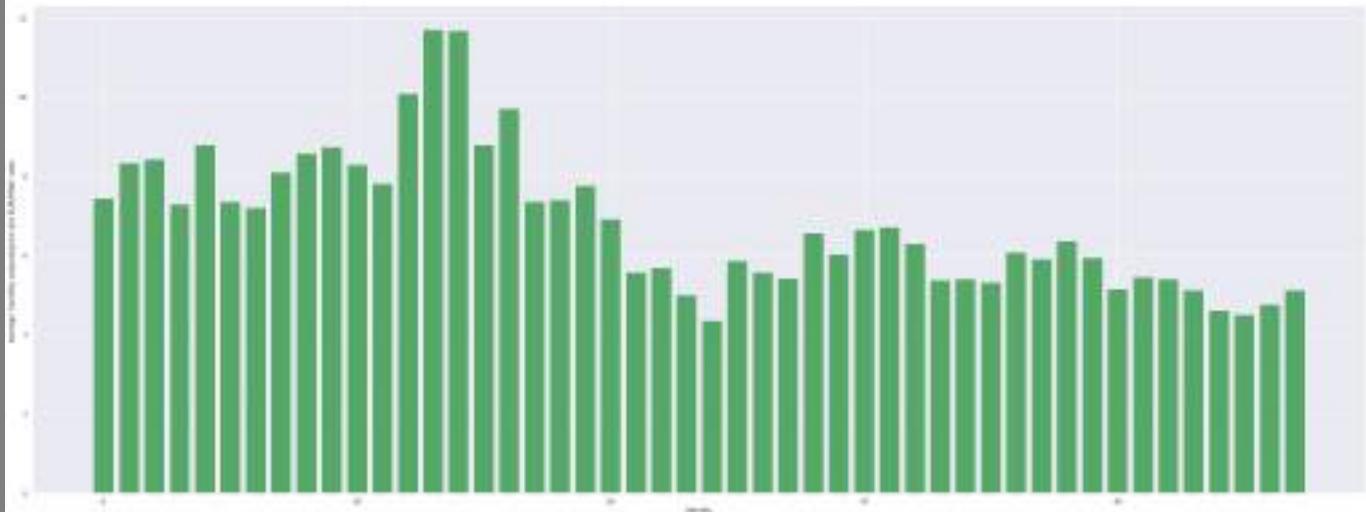
```
In [ ]: pdToListBiomass_Dict = {key: i for i, key in enumerate(pdToListBiomass)}

def Hist_pdToListBiomass(pdToListBiomass_Dict):
    for k, v in pdToListBiomass_Dict.items(): print(f"{v}:{k}") #Outputs in MWH Histogram
#Output/price per EUR/MWH Histogram print(pdToListBiomass_Dict)

plt.bar(list(pdToListBiomass_Dict.values()), pdToListBiomass_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of biomas")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram is skewed to the right, with a unimodal occurring between month 20 and 10.

Below is a scatterplot depicting the relationship between the average monthly price of energy per EUR/MWH and the average monthly outputs of biomass.

In []:

```
In [ ]: modelbiomass = stats.linregress([483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 64.9490188172043, 56.383854166666666, 55.522462987886975, 58.35408333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.75208333333374, 36.81800807537014, 32.61866666666666, 34.691370967741896, 46.266319444444434, 47.50201612903221, 47.6023387096774, 50.40559722222224, 60.182429530201404, 62.58105555555558, 67.5951344086021, 79.4920833333331, 59.83779761904767, 50.95989232839837, 51.71791666666662, 53.77262096774189, 56.25822222222224, 55.252580645161316, 54.08432795698931, 55.81655555555558, 63.92528859060403, 65.43065277777781, 65.15127688172035, 56.51197580645163, 60.877098214285674, 48.279717362045766, 50.4007361111113, 61.633763440860214, 64.34813888888884, 67.78344086021498, 70.36391129032262, 76.9140416666666, 70.36221476510062,
```

```
67.0426075268817,  
66.62351388888881])
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]: BiomassRatio = list(stanardized_residualsBiomass/standardized_residualsPricereg)
```

```
In [ ]:
```

```
In [ ]: BiomassRatioQuad = list(stanardized_residualsBiomassQuad/standardized_residualsPriceQuad)  
print(BiomassRatioQuad)
```

```
[-0.26616005224751593, 0.20970959312870507, 0.2811867102073189, 0.07466488975019675, 0.16374736266946352, -0.384  
2236575519416, -0.6623668237782986, -0.34799318467371904, -0.07756874582202361, -0.007504093294748567, -0.113216  
75300617333, -0.213835179708209, 2.8370709050765224, 7.747791894339479, 10.624002667930867, 24.35303887601845, 7  
9.72603158298884, 8.167725750847183, 12.065224316624121, 112.75618715810958, 1166.609825108993, -1.4284664290662  
272, -2.0965446691361236, -2.9514496016246503, -2.1496864159395406, 2.23894544183176, -0.8665601556788908, -0.56  
85951988424196, -0.7569024152516026, -0.18067010524544674, -0.46815338437579623, -0.5342266750410645, -0.2405317  
5782025943, 1.1715327647164837, 1.3291729709794613, 1.151633244142403, -0.08018174580523726, 0.29958305305367394  
, -0.5302093688556043, -0.3506599062565209, 0.4584462083834092, 0.580151765650505, 0.8787035064128571, 1.1536824  
779906056, 2.0937754347431734, 1.1546960706668628, 0.6718122812909971, 0.6836971809697372]
```

```
In [ ]:
```

```
#OLS predicted quadratic average monthly ratios versus residuals
```

```
BiomassQuadRatioPredict = Biomass_ypred/ypred
```

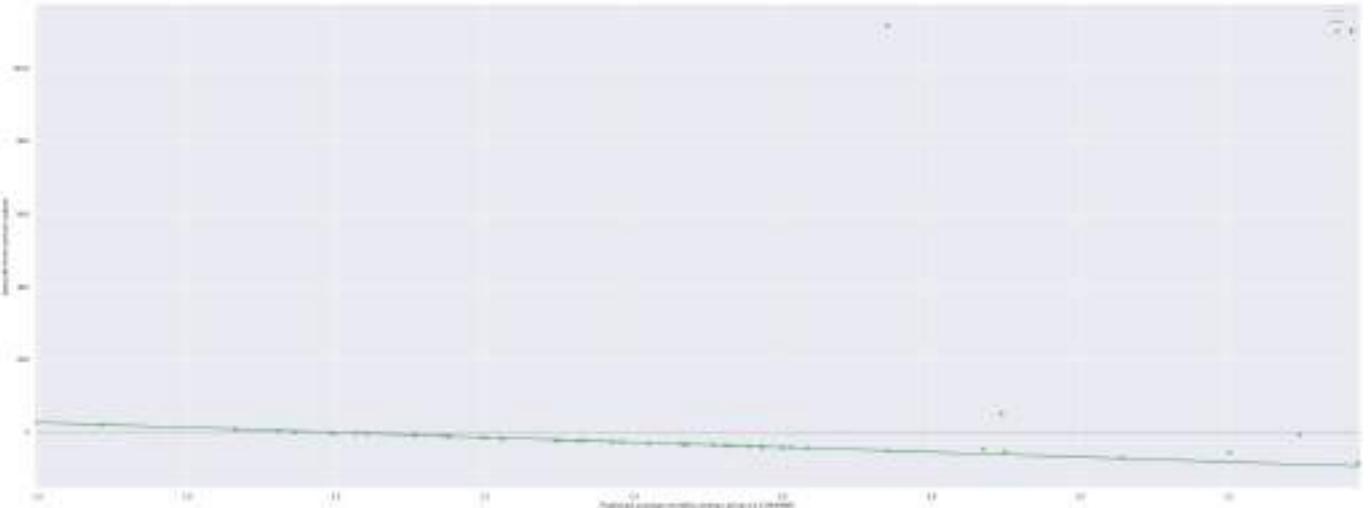
```
sns.residplot(x = BiomassQuadRatioPredict , y = stanardized_residualsBiomassQuad/standardized_residualsPriceQua
```

```
plt.suptitle("Predicted average monthly quadratic biomass energy prices to EUR/MWH versus respective quadratic model")  
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")  
plt.ylabel("Amount from actual values")
```

```
plt.legend("..#")
```

```
Out[ ]:
```

```
<matplotlib.legend.Legend at 0x7ff607c92a50>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]:
```

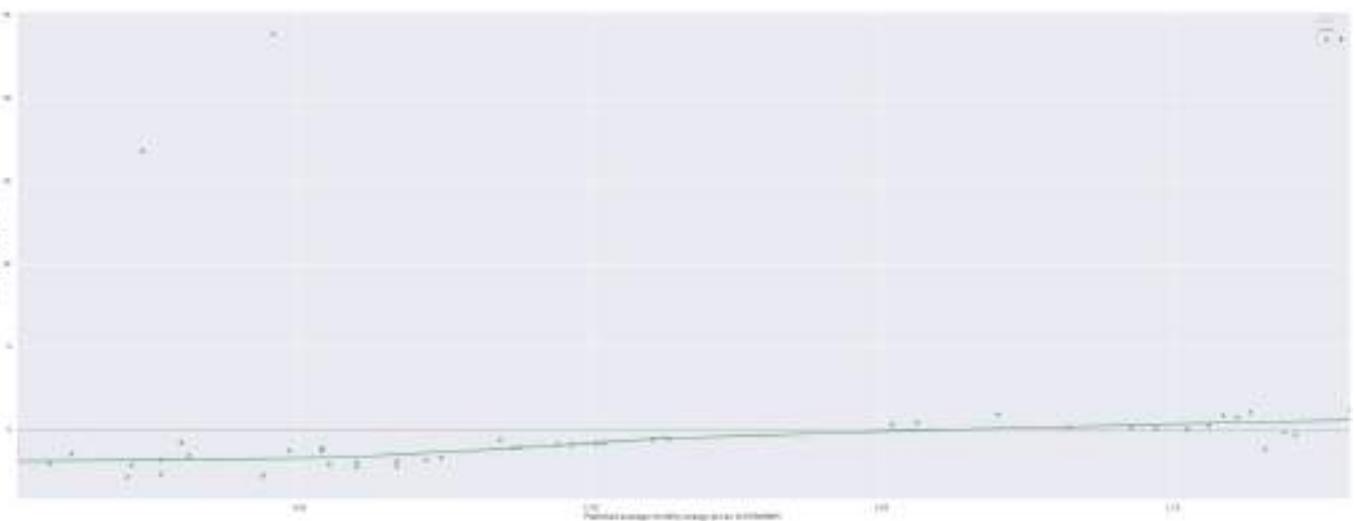
```
#Predicted OLS linear average monthly ratios versus residuals
```

```
BiomassRegRatioPredict = predictionsBiomass/predictions
```

```
sns.residplot(x = BiomassRegRatioPredict, y = stanardized_residualsBiomass/standardized_residualsPricereg, low=0, high=1)  
plt.suptitle("Predicted average monthly linear biomass energy prices to EUR/MWH versus respective linear model")  
plt.xlabel("Predicted average monthly energy prices to EUR/MWH ")  
plt.ylabel("Amount from actual values")
```

```
plt.legend("..#")
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff60692c350>
```



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]:
```

Below is the logarithmic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]:
```

```
print(Biomass_Logpred)

print(Logpred)

Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price
```

```
[59.69978787 59.45029685 59.41275903 58.64535369 60.06405219 59.74135322
 60.22783266 60.3464655 60.31266197 60.3576475 60.06491121 59.70326431
 59.26381122 58.71926405 58.70823491 56.08470199 57.0087011 57.0662674
 57.27842394 57.60450653 57.23254838 56.99062677 57.34606973 57.03760643
 57.1938335 57.26878126 56.0390122 55.97213902 57.30807253 57.05968656
 57.56891174 57.48024566 57.27986411 57.13239575 57.32807259 57.19180938
 57.13074037 57.43677305 56.45869135 56.31431718 56.66171361 57.27555341
 57.55998147 57.45171059 57.34414208 56.65470505 57.09868696 56.6957286 ]
[27.29734838 23.91758071 23.57768041 24.69502249 24.27674267 27.70215664
 29.71346062 26.92210695 25.44503172 25.11040502 25.63128037 26.09491682
 19.65393442 16.17099008 16.19700362 14.53996621 15.35784412 19.92525621
 20.41285544 20.45244219 21.55859284 25.41647801 26.36296287 28.34149128
 33.03596301 25.28048812 21.77731469 22.076427 22.88720221 23.86800702
 23.4711863 23.0102001 23.69372775 26.89338996 27.48739853 27.37715831
 23.96813682 25.69059052 20.71973214 21.55667467 25.98916653 27.0602444
 28.41579599 29.43403567 32.0186817 29.43336623 27.95809508 28.12346716]
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.
 36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.
 69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

```
In [ ]:
```

```
print(list(Biomass_Logpred))
```

```
print(list(Logpred))
```

```
[59.69978786855719, 59.45029684785652, 59.412759029384375, 58.64535368884337, 60.06405219224356, 59.741353216497  
75, 60.22783265650902, 60.34646549920078, 60.31266196836221, 60.357647496233525, 60.06491120658515, 59.703264308  
316285, 59.263811216957635, 58.71926404883028, 58.70823491082773, 56.08470199007345, 57.00870109572714, 57.06626  
739877363, 57.27842393787653, 57.60450653144312, 57.23254838467137, 56.99062677263914, 57.346069732923596, 57.03  
760643463518, 57.19383349693151, 57.268781264019076, 56.03901219606739, 55.972139021709175, 57.30807253063468, 5  
7.05968655901882, 57.56891174177156, 57.480245663754744, 57.27986411236584, 57.13239574818434, 57.32807258963946  
, 57.19180938267834, 57.13074037460106, 57.43677304962051, 56.45869135296222, 56.31431718182853, 56.661713607229  
83, 57.27555349725513, 57.559981466364015, 57.45171058964911, 57.344142082063996, 56.65470504950327, 57.09868696  
206187, 56.695728600411115]  
[27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639  
58084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.0949  
16817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 1  
9.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.36296287420  
1227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.88720  
2207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 2  
7.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.5566746731514  
95, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.4333662  
30197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]:  
dfBiomass_Log = ({ "Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.702156639  
58084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.0949  
16817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 1  
9.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.36296287420  
1227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.88720  
2207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 2  
7.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.5566746731514  
95, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.4333662  
30197276, 27.958095077371567, 28.123467161134005]  
  
#ADF Tests  
from statsmodels.tsa.stattools import adfuller  
def adfuller_test(test_result):  
    result=adfuller(test_result)  
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']  
    for value,label in zip(result,labels):  
        print(label+' : '+str(value) )  
  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")  
    else:  
        print("weak evidence against null hypothesis,indicating it is non-stationary ")  
  
adfuller_test(df_Biomass_Log["Biomass_Log"])  
  
test_result=adfuller(df_Biomass_Log["Biomass_Log"])  
  
df_Biomass_Log['First Difference'] = df_Biomass_Log["Biomass_Log"]- df_Biomass_Log["Biomass_Log"].shift(1) # Seasonal Difference  
df_Biomass_Log['Seasonal Difference']=df_Biomass_Log["Biomass_Log"]- df_Biomass_Log["Biomass_Log"].shift(12) # Predictive Logarithmic Seasonality Plot  
df_Biomass_Log.head() #Seasonality Plot  
df_Biomass_Log['Seasonal Difference'].plot()  
# Seasonality Plot
```

```
{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Biomass_Log': [59.69978786855719, 59.45029684785652, 59.412759029384375, 58.64535368884337, 60.06405219224356, 59.74135321649775, 60.22783265650902, 60.34646549920078, 60.31266196836221, 60.357647496233525, 60.06491120658515, 59.703264308316285, 59.263811216957635, 58.71926404883028, 58.70823491082773, 56.08470199007345, 57.00870109572714, 57.06626739877363, 57.27842393787653, 57.60450653144312, 57.23254838467137, 56.99062677263914, 57.346069732923596, 57.03760643463518, 57.19383349693151, 57.268781264019076, 56.03901219606739, 55.972139021709175, 57.30807253063468, 57.05968655901882, 57.56891174177156, 57.480245663754744, 57.27986411236584, 57.13239574818434, 57.32807258963946, 57.19180938267834, 57.13074037460106, 57.43677304962051, 56.45869135296222, 56.31431718182853, 56.66171360722983, 57.27555340725513, 57.559981466364015, 57.45171058964911, 57.344142082063996, 56.65470504950327, 57.09868696206187, 56.695728600411115]}
```

Price_Log Biomass_Log

	Price_Log	Biomass_Log
0	27.297348	59.699788
1	23.917581	59.450297
2	23.577680	59.412759
3	24.695022	58.645354
4	24.276743	60.064052
5	27.702157	59.741353
6	29.713461	60.227833
7	26.922107	60.346465
8	25.445032	60.312662
9	25.110405	60.357647
10	25.631280	60.064911
11	26.094917	59.703264
12	19.653934	59.263811
13	16.170990	58.719264
14	16.197004	58.708235
15	14.539966	56.084702
16	15.357844	57.008701
17	19.925256	57.066267
18	20.412855	57.278424
19	20.452442	57.604507
20	21.558593	57.232548
21	25.416478	56.990627
22	26.362963	57.346070
23	28.341491	57.037606
24	33.035963	57.193833
25	25.280488	57.268781
26	21.777315	56.039012
27	22.076427	55.972139
28	22.887202	57.308073
29	23.868007	57.059687
30	23.471186	57.568912
31	23.010200	57.480246
32	23.693728	57.279864
33	26.893390	57.132396
34	27.487399	57.328073
35	27.377158	57.191809
36	23.968137	57.130740
37	25.690591	57.436773
38	20.719732	56.458691
39	21.556675	56.314317
40	25.989167	56.661714
41	27.060244	57.275553
42	28.415796	57.559981
43	29.434036	57.451711
44	32.018682	57.344142
45	29.433366	56.654705
46	27.958095	57.098687
47	28.123467	56.695729

ADF Test Statistic : -1.8472609807682063

p-value : 0.35726297393198814

#Lags Used : 0

Number of Observations : 47

weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606862ed0>



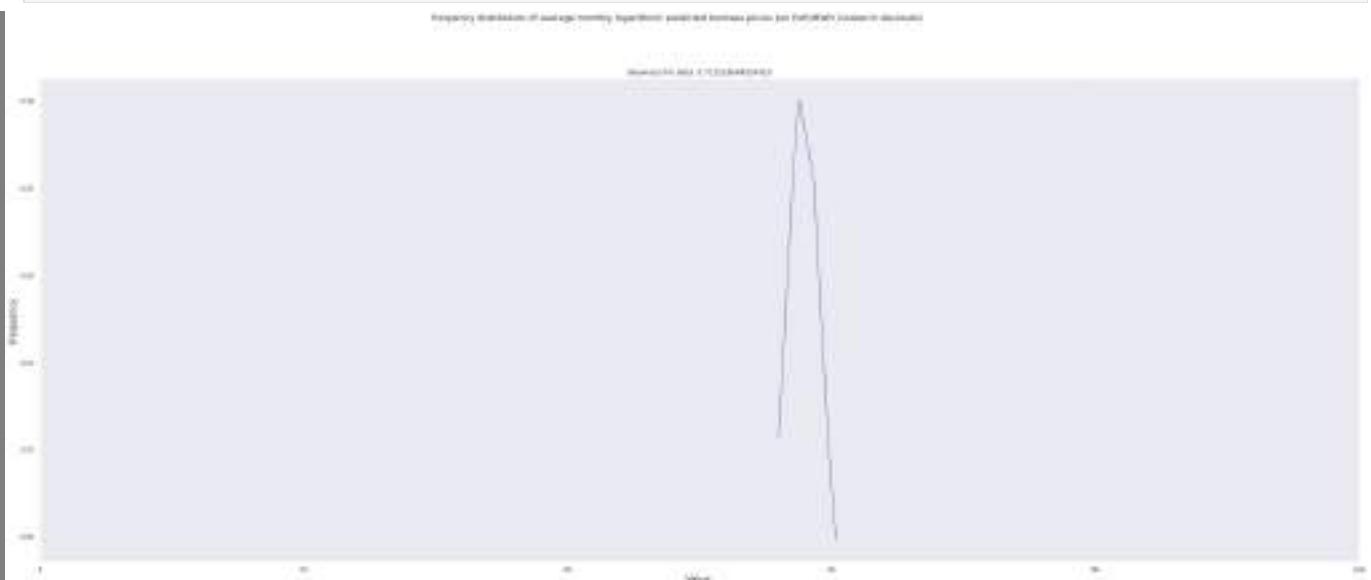
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Biomass_LogResults_mean = np.mean(df_Biomass_Log["Biomass_Log"])
Biomass_LogResults_std = np.std(df_Biomass_Log["Biomass_Log"])

Biomass_LogResultspdf = stats.norm.pdf(df_Biomass_Log["Biomass_Log"].sort_values(), Biomass_LogResults_mean, Biomass_LogResults_std)

plt.plot(df_Biomass_Log["Biomass_Log"].sort_values(), Biomass_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Biomass_Log["Biomass_Log"])}')
plt.suptitle("Frequency distribution of average monthly logarithmic predicted biomass prices per EUR/MWH (scaled)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)

```
Biomass_Log_Dict = {key: i for i, key in enumerate(df_Biomass_Log["Biomass_Log"])}

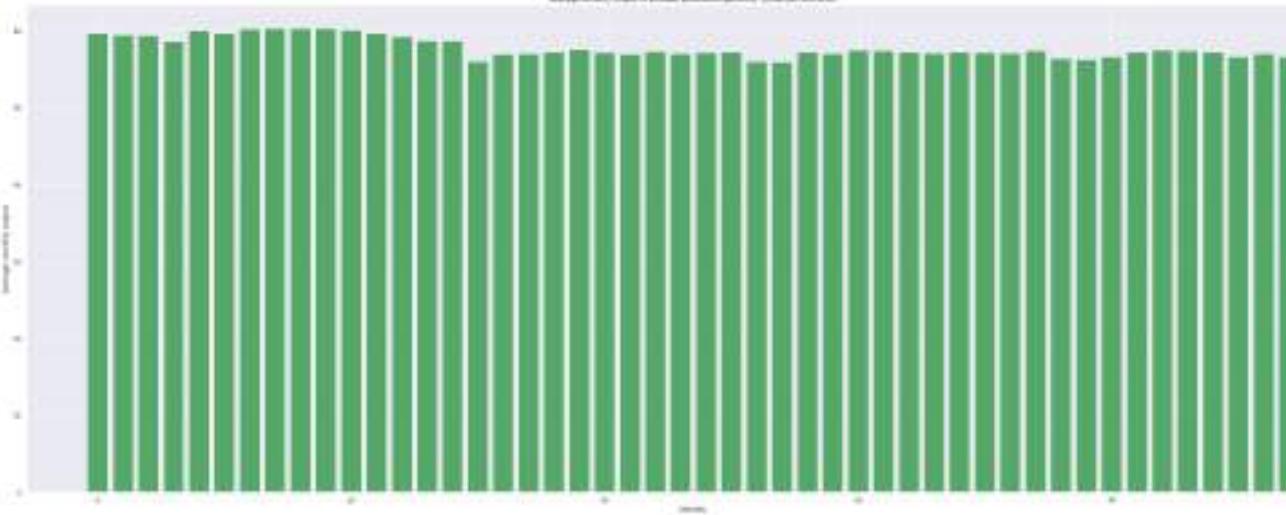
def Hist_Biomass_Log(Biomass_Log_Dict):
    for k, v in Biomass_Log_Dict.items(): print(f"{v}:{k}")
print(Biomass_Log_Dict)

plt.bar(list(Biomass_Log_Dict.values()), Biomass_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly output of biomass predicted logarithmic prices per EUR/MWH" )
```

```
plt.ylabel('Average monthly output')
plt.xlabel('Months')
```

```
plt.show()
```

```
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{59.69978786855719: 0, 59.45029684785652: 1, 59.412759029384375: 2, 58.64535368884337: 3, 60.06405219224356: 4, 59.74135321649775: 5, 60.22783265650902: 6, 60.34646549920078: 7, 60.31266196836221: 8, 60.357647496233525: 9, 60.06491120658515: 10, 59.703264308316285: 11, 59.263811216957635: 12, 58.71926404883028: 13, 58.70823491082773: 14, 56.08470199007345: 15, 57.00870109572714: 16, 57.06626739877363: 17, 57.27842393787653: 18, 57.6045065314431: 2: 19, 57.23254838467137: 20, 56.99062677263914: 21, 57.346069732923596: 22, 57.03760643463518: 23, 57.193833496: 93151: 24, 57.268781264019076: 25, 56.03901219606739: 26, 55.972139021709175: 27, 57.30807253063468: 28, 57.0596: 8655901882: 29, 57.56891174177156: 30, 57.480245663754744: 31, 57.27986411236584: 32, 57.13239574818434: 33, 57.32807258963946: 34, 57.19180938267834: 35, 57.13074037460106: 36, 57.43677304962051: 37, 56.45869135296222: 38, 56.31431718182853: 39, 56.66171360722983: 40, 57.27555340725513: 41, 57.559981466364015: 42, 57.45171058964911: 43, 57.344142082063996: 44, 56.65470504950327: 45, 57.09868696206187: 46, 56.695728600411115: 47}
```



The green bars represent the observation value for each respective month. This histogram is uniform throughout.

```
In [ ]: df_Biomass_Log.describe(include = 'all') # Description Tables
```

	Price_Log	Biomass_Log	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	-0.063916	-0.942879
std	4.072442	1.314925	0.627933	1.297109
min	14.539966	55.972139	-2.623533	-3.367021
25%	22.001649	57.064622	-0.300600	-2.586903
50%	25.195447	57.318073	-0.066873	-0.332681
75%	27.317301	58.855401	0.248292	0.051556
max	33.035963	60.357647	1.418699	0.419679

```
In [ ]: print(predictionsBiomass)

print(predictions)

print(predictionsBiomass/predictions)

Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price
```

```
[59.69978787 59.45029685 59.41275903 58.64535369 60.06405219 59.74135322
60.22783266 60.3464655 60.31266197 60.3576475 60.06491121 59.70326431
59.26381122 58.71926405 58.70823491 56.08470199 57.0087011 57.0662674
57.27842394 57.60450653 57.23254838 56.99062677 57.34606973 57.03760643
57.1938335 57.26878126 56.0390122 55.97213902 57.30807253 57.05968656
57.56891174 57.48024566 57.27986411 57.13239575 57.32807259 57.19180938
57.13074037 57.43677305 56.45869135 56.31431718 56.66171361 57.27555341
57.55998147 57.45171059 57.34414208 56.65470505 57.09868696 56.6957286 ]
[52.82161316 53.03600615 53.25039913 53.46479211 53.67918509 53.89357808
54.10797106 54.32236404 54.53675703 54.75115001 54.96554299 55.17993597
55.39432896 55.60872194 55.82311492 56.03750791 56.25190089 56.46629387
56.68068685 56.89507984 57.10947282 57.3238658 57.53825879 57.75265177
57.96704475 58.18143773 58.39583072 58.6102237 58.82461668 59.03900967
59.25340265 59.46779563 59.68218861 59.8965816 60.11097458 60.32536756
60.53976055 60.75415353 60.96854651 61.18293949 61.39733248 61.61172546
61.82611844 62.04051143 62.25490441 62.46929739 62.68369037 62.89808336]
[1.13021516 1.12094219 1.1157242 1.09689669 1.11894493 1.10850597
1.11310462 1.11089542 1.10590848 1.10239963 1.09277391 1.08197415
1.0698534 1.05593623 1.05168325 1.00084219 1.01345377 1.01062534
1.01054569 1.01246903 1.00215508 0.99418673 0.9966598 0.98761883
0.98666119 0.98431361 0.95964064 0.95498934 0.97421923 0.96647432
0.97157141 0.96657771 0.95974805 0.95385069 0.95370393 0.94805571
0.94368957 0.94539665 0.92602981 0.92042517 0.92286931 0.92962099
0.93099782 0.92603541 0.92111847 0.9069208 0.91090181 0.9013904 ]
[52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04,
56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59
.68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]
```

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(predictionsBiomass))

print(list(predictions))

[59.69978786855719, 59.45029684785652, 59.412759029384375, 58.64535368884337, 60.06405219224356, 59.741353216497
75, 60.22783265650902, 60.34646549920078, 60.31266196836221, 60.357647496233525, 60.06491120658515, 59.703264308
316285, 59.263811216957635, 58.71926404883028, 58.70823491082773, 56.08470199007345, 57.00870109572714, 57.06626
739877363, 57.27842393787653, 57.60450653144312, 57.23254838467137, 56.99062677263914, 57.346069732923596, 57.03
760643463518, 57.19383349693151, 57.268781264019076, 56.03901219606739, 55.972139021709175, 57.30807253063468, 5
7.05968655901882, 57.56891174177156, 57.480245663754744, 57.27986411236584, 57.13239574818434, 57.32807258963946
, 57.19180938267834, 57.13074037460106, 57.43677304962051, 56.45869135296222, 56.31431718182853, 56.661713607229
83, 57.27555340725513, 57.559981466364015, 57.45171058964911, 57.344142082063996, 56.65470504950327, 57.09868696
206187, 56.695728600411115]
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.
0390966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373
92337, 62.89808335677895]
```

```
In [ ]: dfBiomassReg = ({ "Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.
0390966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373
92337, 62.89808335677895]
"BiomassReg" : [59.69978786855719, 59.45029684785652, 59.412759029384375, 58.64535368884337, 60.06405219224356,
print(dfBiomassReg) #Dataframes
df_BiomassReg= pd.DataFrame.from_dict(dfBiomassReg, orient = "columns")
print(df_BiomassReg)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_BiomassReg["BiomassReg"])

test_result=adfuller(df_BiomassReg["BiomassReg"])

df_BiomassReg['First Difference'] = df_BiomassReg["BiomassReg"]- df_BiomassReg["BiomassReg"].shift(1) # Seasonal
```

```

df_BiomassReg['Seasonal Difference']=df_BiomassReg["BiomassReg"]- df_BiomassReg["BiomassReg"].shift(12) #
plt.suptitle('Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWh')
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_BiomassReg.head() #Predictive Linear Seasonality Plot
df_BiomassReg['Seasonal Difference'].plot()
# Seasonality Plot

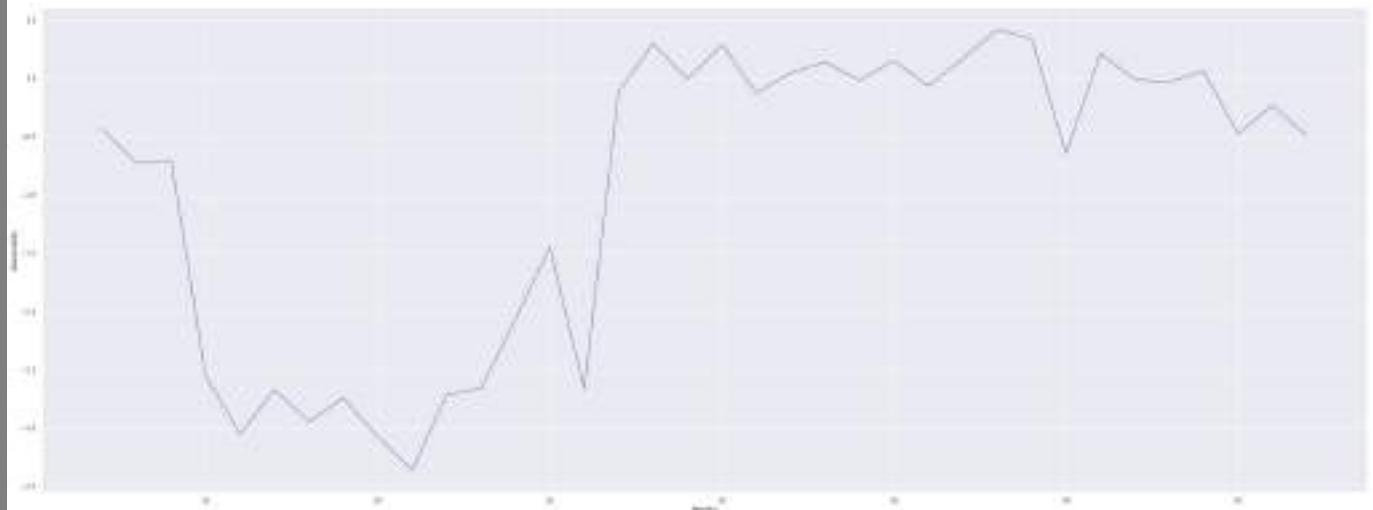
```

{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.7511500826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.1109745796539, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.8980835677895], 'BiomassReg': [59.69978786855719, 59.45029684785652, 59.412759029384375, 58.64535368884337, 60.06405219224356, 59.74135321649775, 60.22783265650902, 60.34646549920078, 60.31266196836221, 60.357647496233525, 60.06491120658515, 59.703264308316285, 59.263811216957635, 58.71926404883028, 58.70823491082773, 56.08470199007345, 57.00870109572714, 57.06626739877363, 57.27842393787653, 57.60450653144312, 57.23254838467137, 56.99062677263914, 57.346069732923596, 57.03760643463518, 57.19383349693151, 57.268781264019076, 56.03901219606739, 55.972139021709175, 57.30807253063468, 57.05968655901882, 57.56891174177156, 57.480245663754744, 57.27986411236584, 57.13239574818434, 57.32807258963946, 57.19180938267834, 57.13074037460106, 57.43677304962051, 56.45869135296222, 56.31431718182853, 56.66171360722983, 57.27555340725513, 57.559981466364015, 57.45171058964911, 57.344142082063996, 56.65470504950327, 57.09868696206187, 56.695728600411115]}

	Price	BiomassReg
0	52.821613	59.699788
1	53.036006	59.450297
2	53.250399	59.412759
3	53.464792	58.645354
4	53.679185	60.064052
5	53.893578	59.741353
6	54.107971	60.227833
7	54.322364	60.346465
8	54.536757	60.312662
9	54.751150	60.357647
10	54.965543	60.064911
11	55.179936	59.703264
12	55.394329	59.263811
13	55.608722	58.719264
14	55.823115	58.708235
15	56.037508	56.084702
16	56.251901	57.008701
17	56.466294	57.066267
18	56.680687	57.278424
19	56.895080	57.604507
20	57.109473	57.232548
21	57.323866	56.990627
22	57.538259	57.346070
23	57.752652	57.037606
24	57.967045	57.193833
25	58.181438	57.268781
26	58.395831	56.039012
27	58.610224	55.972139
28	58.824617	57.308073
29	59.039010	57.059687
30	59.253403	57.568912
31	59.467796	57.480246
32	59.682189	57.279864
33	59.896582	57.132396
34	60.110975	57.328073
35	60.325368	57.191809
36	60.539761	57.130740
37	60.754154	57.436773
38	60.968547	56.458691
39	61.182939	56.314317
40	61.397332	56.661714
41	61.611725	57.275553
42	61.826118	57.559981
43	62.040511	57.451711
44	62.254904	57.344142
45	62.469297	56.654705
46	62.683690	57.098687
47	62.898083	56.695729

ADF Test Statistic : -1.8472609807682063
p-value : 0.35726297393198814
#Lags Used : 0
Number of Observations : 47
weak evidence against null hypothesis, indicating it is non-stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff606059410>



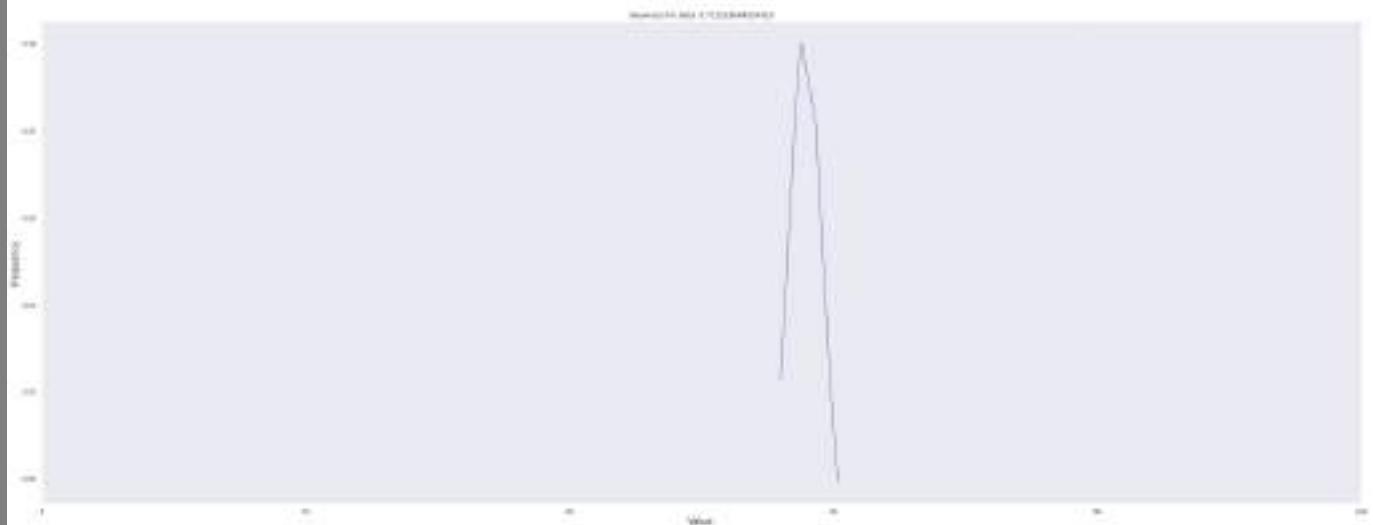
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
BiomassRegResults_mean = np.mean(df_BiomassReg["BiomassReg"])
BiomassRegResults_std = np.std(df_BiomassReg["BiomassReg"])

BiomassRegResultspdf = stats.norm.pdf(df_BiomassReg["BiomassReg"].sort_values(), BiomassRegResults_mean, BiomassRegResults_std)

plt.plot(df_BiomassReg["BiomassReg"].sort_values(), BiomassRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_BiomassReg["BiomassReg"])}')
plt.suptitle("Frequency distribution of average monthly predicted linear biomass prices per EUR/MWH (scaled in density)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)
BiomassReg_Dict = {key: i for i, key in enumerate(df_BiomassReg["BiomassReg"])}

```

def Hist_BiomassReg(BiomassReg_Dict):
    for k, v in BiomassReg_Dict.items(): print(f"{v}:{k}")
print(BiomassReg_Dict)

plt.bar(list(BiomassReg_Dict.values()), BiomassReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted biomass prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

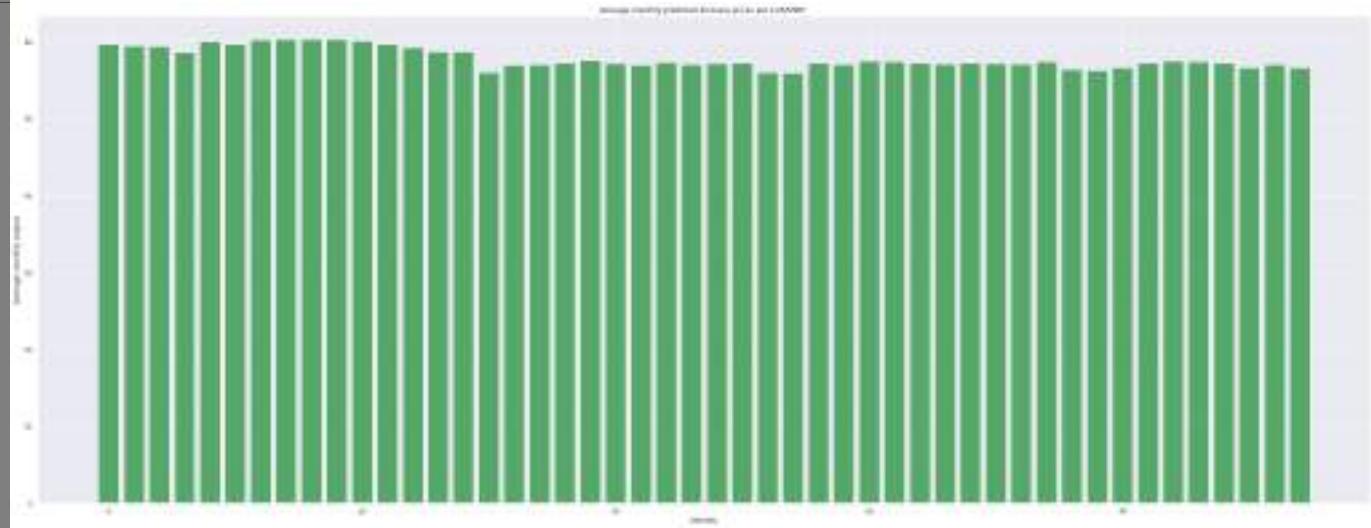
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{59.699786855719: 0, 59.45029684785652: 1, 59.412759029384375: 2, 58.64535368884337: 3, 60.06405219224356: 4, 59.74135321649775: 5, 60.22783265650902: 6, 60.34646549920078: 7, 60.31266196836221: 8, 60.357647496233525: 9, 60.06491120658515: 10, 59.703264308316285: 11, 59.263811216957635: 12, 58.71926404883028: 13, 58.70823491082773: 14, 56.08470199007345: 15, 57.00870109572714: 16, 57.06626739877363: 17, 57.27842393787653: 18, 57.60450653144312: 19, 57.23254838467137: 20, 56.99062677263914: 21, 57.346069732923596: 22, 57.03760643463518: 23, 57.19383349693151: 24, 57.268781264019076: 25, 56.03901219606739: 26, 55.972139021709175: 27, 57.30807253063468: 28, 57.05968655901882: 29, 57.56891174177156: 30, 57.480245663754744: 31, 57.27986411236584: 32, 57.13239574818434: 33, 57.32807258963946: 34, 57.19180938267834: 35, 57.13074037460106: 36, 57.43677304962051: 37, 56.45869135296222: 38, 56.31431718182853: 39, 56.66171360722983: 40, 57.27555340725513: 41, 57.559981466364015: 42, 57.45171058964911: 43, 57.344142082063996: 44, 56.65470504950327: 45, 57.09868696206187: 46, 56.695728600411115: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform throughout.

```
In [ ]: df_BiomassReg.describe(include = 'all') # Description Tables
```

	Price	BiomassReg	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	57.859848	57.859848	-0.063916	-0.942879
std	3.001502	1.314925	0.627933	1.297109
min	52.821613	55.972139	-2.623533	-3.367021
25%	55.340731	57.064622	-0.300600	-2.586903
50%	57.859848	57.318073	-0.066873	-0.332681
75%	60.378966	58.855401	0.248292	0.051556
max	62.898083	60.357647	1.418699	0.419679

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: Rounded_ypred = [round(item, 2) for item in ypred]
print(Rounded_ypred)
#Rounded Price
```

```
[27.19, 23.23, 22.88, 24.06, 23.6, 27.72, 30.59, 26.7, 24.9, 24.52, 25.12, 25.67, 19.55, 17.63, 17.64, 17.08, 17.33, 19.74, 20.09, 20.12, 21.01, 24.87, 26.0, 28.6, 36.04, 24.71, 21.19, 21.46, 22.2, 23.17, 22.77, 22.32, 23.0, 26.67, 27.44, 27.29, 23.28, 25.19, 20.33, 21.01, 25.54, 26.88, 28.7, 30.17, 34.27, 30.17, 28.07, 28.3]
```

```
In [ ]: print(list(Biomass_ypred))
print(list(ypred))
```

```
[59.70709902918792, 59.74596550984818, 59.746836062692246, 59.47928122360187, 59.54707599650557, 59.695035297384
855, 59.435175825544256, 59.33864519279253, 59.36747550934623, 59.32887593238186, 59.54655373996323, 59.70615121
158287, 59.737459071656744, 59.52872767266557, 59.521669488115656, 54.64981479202691, 57.09108952569416, 57.2170
8172056947, 57.65498291346188, 58.246956629008565, 57.56381741735808, 57.050899700402056, 57.785863599365435, 57
.15473574907375, 57.48536901467173, 57.63598205177119, 54.50863248922965, 54.298515882289806, 57.71286724730339,
57.20283359460335, 58.187113929414835, 58.03295708790631, 57.65781340001516, 57.35803460722605, 57.751455596638
56, 57.48122943748362, 57.35455547219662, 57.95472101387236, 55.73294386093205, 55.33012842320964, 56.2668114611
00446, 57.649335521013, 58.17191648560123, 57.98180045993831, 57.78219251186782, 56.24901635665803, 57.286688996
76909, 56.35253274917526]
[27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338
71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672
299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486
, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506
2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20
1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,
27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905
53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857
92181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: dfBiomass_Quad = ({ "Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.7247224338
71626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672
299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486
, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.00005506
2885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.20
1511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803,
27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.0063717885905
53, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.1694857
92181256, 28.070861800194585, 28.297337605159356]}

"Biomass_Quad" : [59.70709902918792, 59.74596550984818, 59.746836062692246, 59.47928122360187, 59.54707599650557
print(dfBiomass_Quad) #Dataframes
df_Biomass_Quad= pd.DataFrame.from_dict(dfBiomass_Quad, orient = "columns")
print(df_Biomass_Quad)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_Biomass_Quad["Biomass_Quad"])

test_result=adfuller(df_Biomass_Quad["Biomass_Quad"])

df_Biomass_Quad['First Difference'] = df_Biomass_Quad["Biomass_Quad"]- df_Biomass_Quad["Biomass_Quad"].shift(1)
df_Biomass_Quad['Seasonal Difference']=df_Biomass_Quad["Biomass_Quad"]- df_Biomass_Quad["Biomass_Quad"].shift(1)
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Biomass_Quad.head() #Predictive Quadratic Seasonality Plot
df_Biomass_Quad['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135577786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.886735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279493527835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Biomass_Quad': [59.70709902918792, 59.74596550984818, 59.746836062692246, 59.47928122360187, 59.54707599650557, 59.695035297384855, 59.435175825544256, 59.33864519279253, 59.36747550934623, 59.32887593238186, 59.54655373996323, 59.70615121158287, 59.737459071656744, 59.52872767266557, 59.521669488115656, 54.64981479202691, 57.09108952569416, 57.21708172056947, 57.65498291346188, 58.246956629008565, 57.56381741735808, 57.050899700402056, 57.785863599365435, 57.15473574907375, 57.48536901467173, 57.63598205177119, 54.50863248922965, 54.298515882289806, 57.71286724730339, 57.20283359460335, 58.187113929414835, 58.03295708790631, 57.65781340001516, 57.358034606722605, 57.75145559663856, 57.48122943748362, 57.35455547219662, 57.95472101387236, 55.73294386093205, 55.33012842320964, 56.266811461100446, 57.649335521013, 58.17191648560123, 57.98180045993831, 57.78219251186782, 56.24901635665803, 57.28668899676909, 56.35253274917526]}
```

	Price_Quad	Biomass_Quad
--	------------	--------------

0	27.188078	59.707099
1	23.225548	59.745966
2	22.878236	59.746836
3	24.055117	59.479281
4	23.602709	59.547076
5	27.724722	59.695035
6	30.587839	59.435176
7	26.702482	59.338645
8	24.901804	59.367476
9	24.518415	59.328876
10	25.119121	59.546554
11	25.672300	59.706151
12	19.546363	59.737459
13	17.633406	59.528728
14	17.644052	59.521669
15	17.075344	54.649815
16	17.328255	57.091090
17	19.736629	57.217082
18	20.093546	57.654983
19	20.123369	58.246957
20	21.007992	57.563817
21	24.868735	57.050900
22	26.000055	57.785864
23	28.599305	57.154736
24	36.035067	57.485369
25	24.712150	57.635982
26	21.194644	54.508632
27	21.456173	54.298516
28	22.201511	57.712867
29	23.174311	57.202834
30	22.771345	58.187114
31	22.319233	58.032957
32	22.995762	57.657813
33	26.665789	57.358035
34	27.438370	57.751456
35	27.292830	57.481229
36	23.278006	57.354555
37	25.188915	57.954721
38	20.328049	55.732944
39	21.006372	55.330128
40	25.544593	56.266811
41	26.879918	57.649336
42	28.703098	58.171916
43	30.170478	57.981800
44	34.272280	57.782193
45	30.169486	56.249016
46	28.070862	57.286689
47	28.297338	56.352533

ADF Test Statistic : -3.1344404044151752

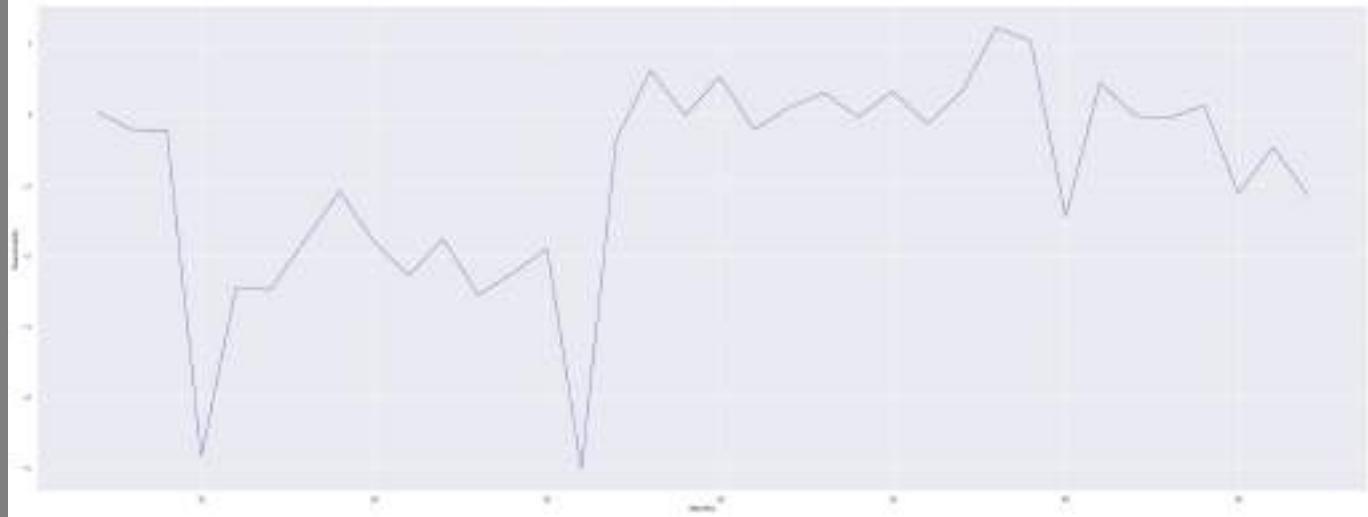
p-value : 0.02410722350929471

#Lags Used : 0

Number of Observations : 47

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff604c6c110>



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Biomass_QuadResults_mean = np.mean(df_Biomass_Quad["Biomass_Quad"])
Biomass_QuadResults_std = np.std(df_Biomass_Quad["Biomass_Quad"])

Biomass_QuadResultspdf = stats.norm.pdf(df_Biomass_Quad["Biomass_Quad"].sort_values(), Biomass_QuadResults_mean)

plt.plot(df_Biomass_Quad["Biomass_Quad"].sort_values(), Biomass_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Biomass_Quad["Biomass_Quad"])}')
plt.suptitle("Frequency distribution of average monthly quadratic predicted biomass prices per EUR/MWH (scaled :")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is slightly skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []: print(dicDates)

```
Biomass_Quad_Dict = {key: i for i, key in enumerate(df_Biomass_Quad["Biomass_Quad"])}

def Hist_Biomass_Quad(Biomass_Quad_Dict):
    for k, v in Biomass_Quad_Dict.items(): print(f"{v}:{k}")
```

```

print(Biomass_Quad_Dict)

plt.bar(list(Biomass_Quad_Dict.values()), Biomass_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic biomass prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

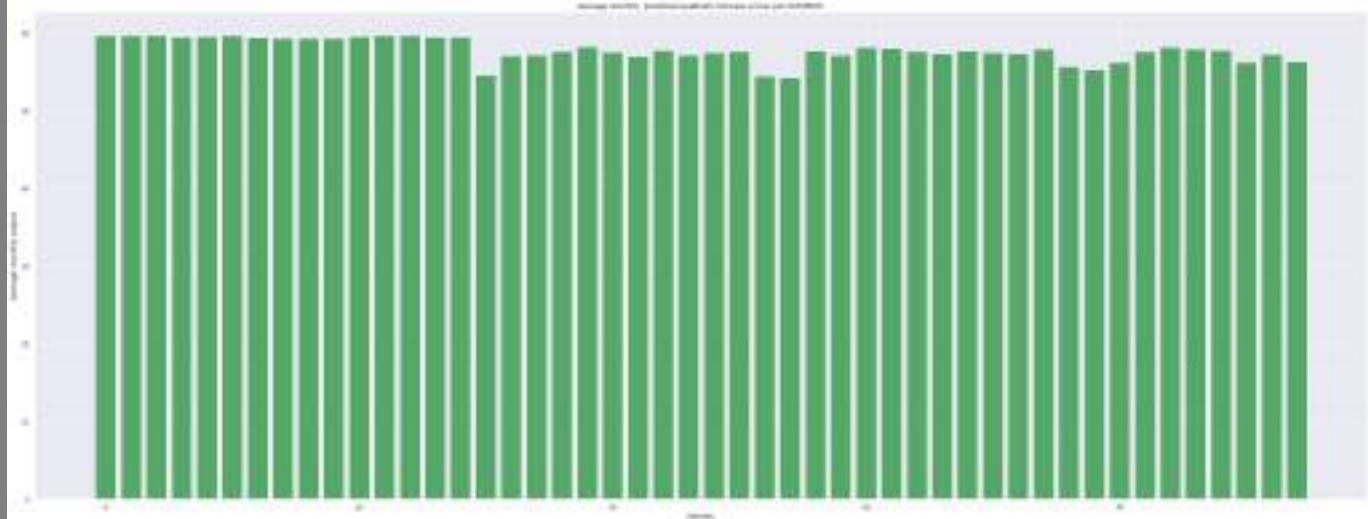
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{59.70709902918792: 0, 59.74596550984818: 1, 59.746836062692246: 2, 59.47928122360187: 3, 59.54707599650557: 4, 59.695035297384855: 5, 59.435175825544256: 6, 59.33864519279253: 7, 59.36747550934623: 8, 59.32887593238186: 9, 59.54655373996323: 10, 59.70615121158287: 11, 59.737459071656744: 12, 59.52872767266557: 13, 59.521669488115656: 14, 54.64981479202691: 15, 57.09108952569416: 16, 57.21708172056947: 17, 57.65498291346188: 18, 58.2469566290085 65: 19, 57.56381741735808: 20, 57.050899700402056: 21, 57.785863599365435: 22, 57.15473574907375: 23, 57.4853690 1467173: 24, 57.63598205177119: 25, 54.50863248922965: 26, 54.298515882289806: 27, 57.71286724730339: 28, 57.202 83359460335: 29, 58.187113929414835: 30, 58.03295708790631: 31, 57.65781340001516: 32, 57.358034606722605: 33, 5 7.75145559663856: 34, 57.48122943748362: 35, 57.35455547219662: 36, 57.95472101387236: 37, 55.73294386093205: 38 , 55.33012842320964: 39, 56.266811461100446: 40, 57.649335521013: 41, 58.17191648560123: 42, 57.98180045993831: 43, 57.78219251186782: 44, 56.24901635665803: 45, 57.28668899676909: 46, 56.35253274917526: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform.

```
In [ ]: df_Biomass_Quad.describe(include = 'all') # Description Tables
```

	Price_Quad	Biomass_Quad	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	-0.071374	-0.848098
std	4.174498	1.464208	1.217766	1.460524
min	17.075344	54.298516	-4.871855	-5.013037
25%	21.390791	57.213520	-0.285002	-1.786059
50%	24.615282	57.732161	-0.007058	-0.221202
75%	27.214266	59.384401	0.362027	0.101592
max	36.035067	59.746836	3.414351	1.224311

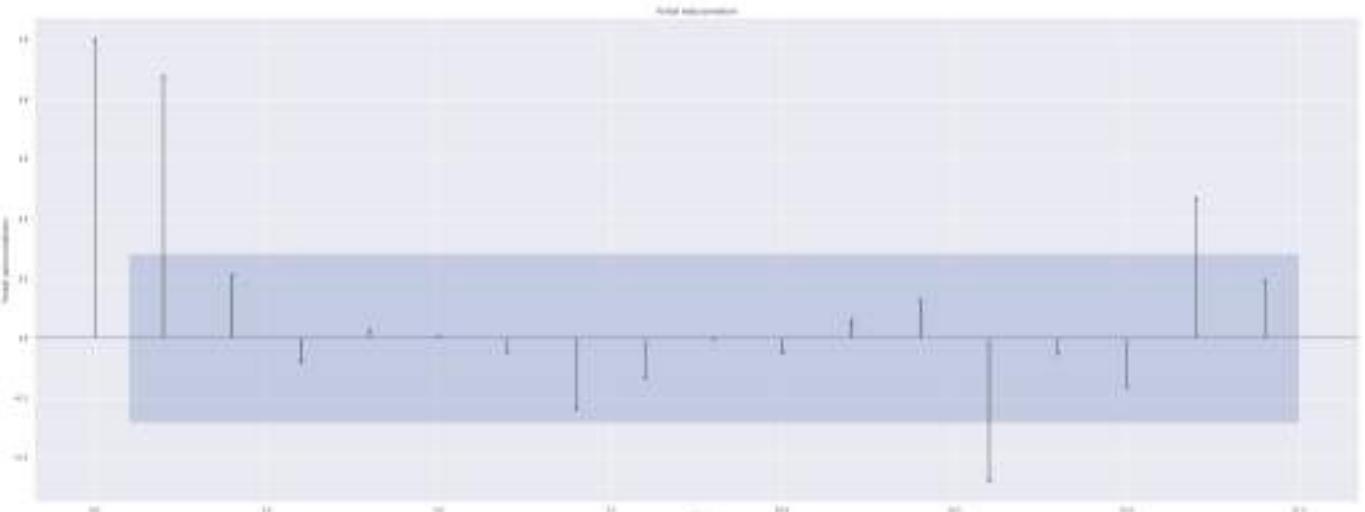
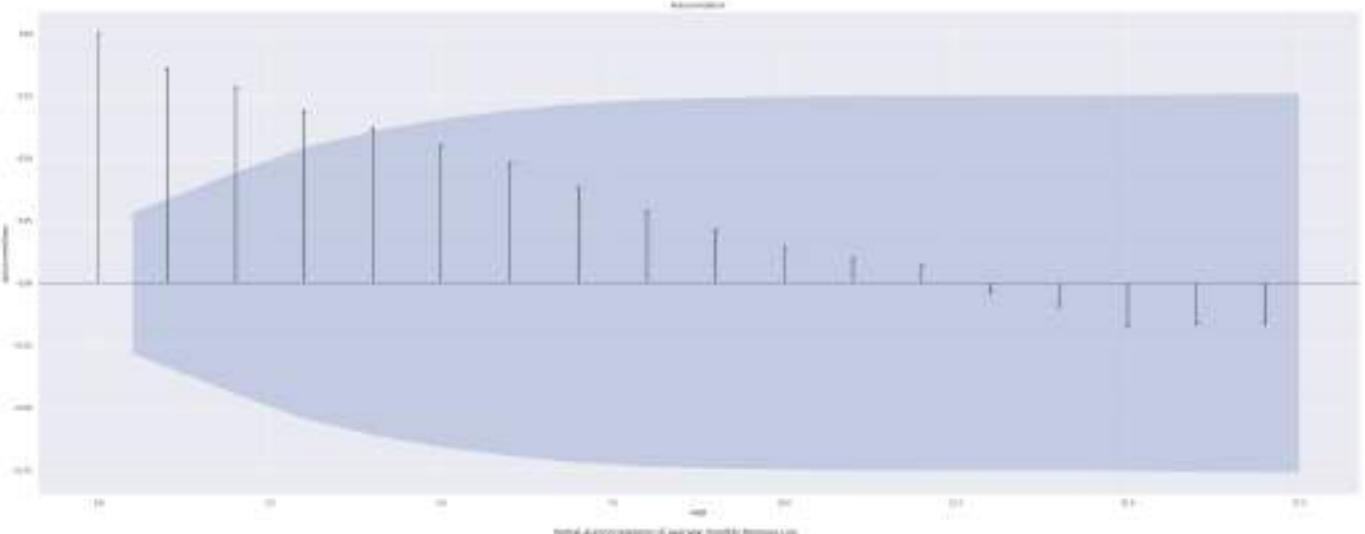
```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Biomass_Logpred)
plt.suptitle(" Autocorrelations of average monthly Biomass Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show()
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Biomass_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Biomass Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show()
Biomass_Log_Autocorrelations = sm.tsa.acf(Biomass_Logpred, fft=False) #Autocorrelations
print(Biomass_Log_Autocorrelations)

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,
```

```
[ 1.          0.85805346  0.78305057  0.68702404  0.61670319  0.54835849  
 0.48101225  0.37960821  0.28797332  0.21113499  0.14061441  0.09681304  
 0.06915393 -0.03567087 -0.09213371 -0.17142338 -0.1613589 -0.15787187  
-0.13324629 -0.14851053 -0.15271529 -0.15780625 -0.17011637 -0.14435201  
-0.14551607 -0.19433829 -0.22562156 -0.22828491 -0.20880273 -0.21332038  
-0.20098962 -0.23158963 -0.23737006 -0.26295471 -0.25810516 -0.24513517  
-0.23342398 -0.2199394 -0.21875793 -0.18906119 -0.1479295 ]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

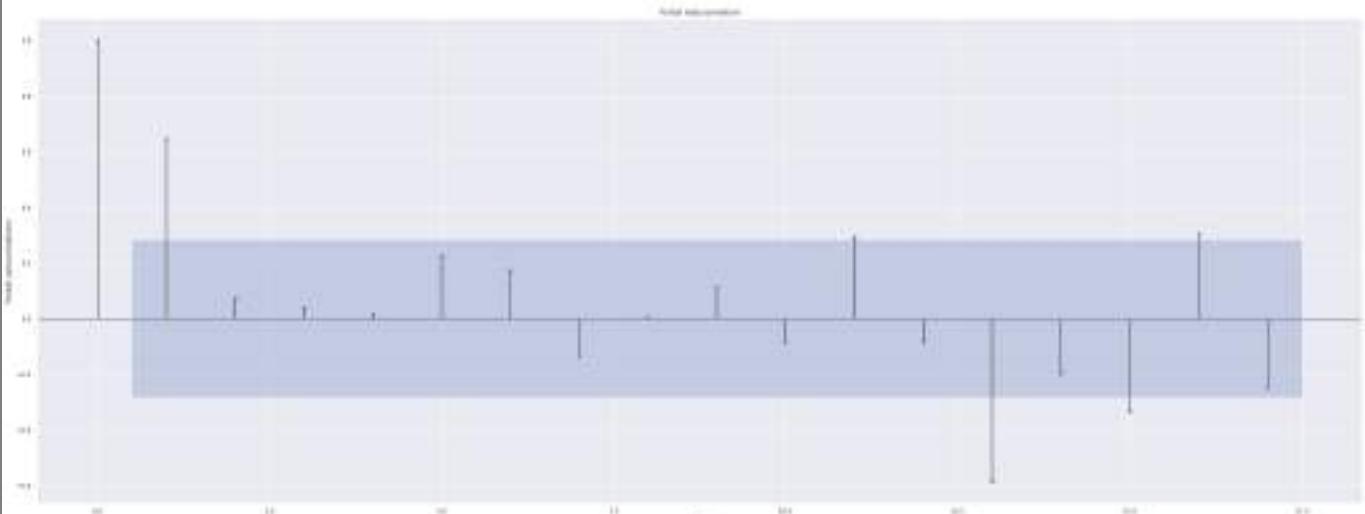
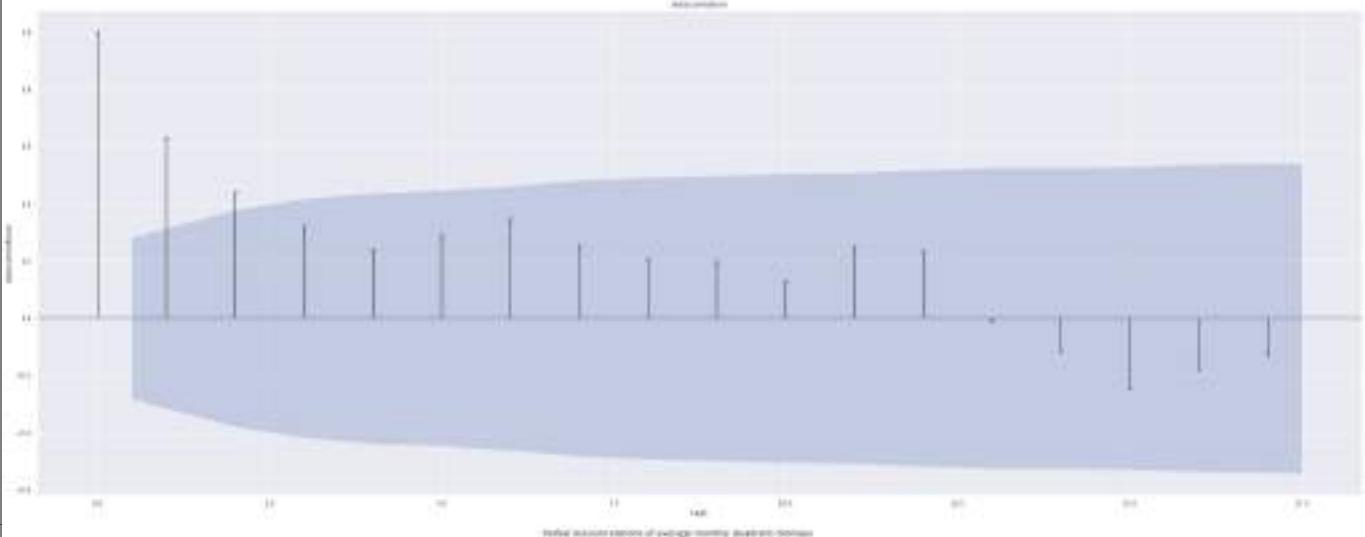
As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(Biomass_ypred)  
plt.suptitle(" Autocorrelations of average monthly Quadratic Biomass")  
plt.ylabel('Autocorrealtions')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(Biomass_ypred)  
plt.suptitle("Partial Autocorrelations of average monthly Quadratic Biomass")  
plt.ylabel('Partial autocorrealtions')  
plt.xlabel('Lags')  
plt.show  
Biomass_Quad_Autocorrelations = sm.tsa.acf(Biomass_ypred, fft=False) #Autocorrelations  
print(Biomass_Quad_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.63211055  0.44050185  0.32241461  0.23893367  0.29072783  
 0.34533734  0.25272344  0.20387013  0.19651901  0.12855887  0.24945821  
 0.23406115 -0.01012287 -0.11562898 -0.2458271  -0.17753147 -0.123465  
 -0.03063328 -0.05977543 -0.07387581 -0.09738817 -0.13317842 -0.06046043  
 -0.08178339 -0.18079244 -0.25366333 -0.21758553 -0.13965447 -0.10601797  
 -0.04519509 -0.1117226  -0.10732843 -0.18886611 -0.18670701 -0.18374971  
 -0.17367346 -0.16581219 -0.16751808 -0.13064187 -0.08706081]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(predictionsBiomass)  
plt.suptitle(" Autocorrelations of average monthly Linear Biomass")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(predictionsBiomass)  
plt.suptitle("Partial Autocorrelations of average monthly Linear Biomass")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
Biomass_Pred_Autocorrelations = sm.tsa.acf(predictionsBiomass, fft=False) #Autocorrelations  
print(Biomass_Pred_Autocorrelations)
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,

```
[ 1.      0.85805346  0.78305057  0.68702404  0.61670319  0.54835849
 0.48101225  0.37960821  0.28797332  0.21113499  0.14061441  0.09681304
 0.06915393 -0.03567087 -0.09213371 -0.17142338 -0.1613589 -0.15787187
-0.13324629 -0.14851053 -0.15271529 -0.15780625 -0.17011637 -0.14435201
-0.14551607 -0.19433829 -0.22562156 -0.22828491 -0.20880273 -0.21332038
-0.20098962 -0.23158963 -0.23737006 -0.26295471 -0.25810516 -0.24513517
-0.23342398 -0.2199394 -0.21875793 -0.18906119 -0.1479295 ]
```

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

The last resource analyzed is wind power.

```
In [ ]: wind1 = wind  
wind1 = sm.a
```

```
In [ ]: #Dataframes analyzed by resource
dfwind = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.294059139784,
                     "Wind" : [7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.444444444444,
                     print(dfwind)
df_wind= pd.DataFrame.from_dict(dfwind, orient = "columns")
print(df_wind)
df_wind["Ratio"] = df_wind["Wind"]/df_wind["Price"]
pdToListWind = list(df_wind["Ratio"])
```

```
print(pdToListWind)
```

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Wind': [7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.495132127956, 3955.2540322580644, 4856.173387096775, 4323.8, 4597.236559139785, 4493.763888888889, 4943.14131897712, 4993.399193548387, 6534.337643678161, 5884.69044145357, 5343.191666666667, 5309.384408602151, 5638.5375, 5715.833109017497, 5103.193548387097, 5484.695833333333, 4788.928859060403, 5766.3944444444, 4519.7661290322585, 6483.298387096775, 5330.943452380952, 5623.149394347241, 5741.669444444445, 5031.903225806452, 5586.527777777777, 4799.477150537635, 4348.514784946236, 3883.87916666666666, 5234.8, 5200.188888888889, 7257.2553763440865, 6197.444892473119, 6864.555059523809, 8771.240915208613, 5289.247222222222, 4389.743279569892, 4779.433333333333, 4059.907133243607, 4358.579301075269, 4265.988888888889, 6156.939597315436, 5886.720430107527, 6057.558333333333], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

Price Wind Dates
0 64.949019 7587.697135 2015-01
1 56.383854 7731.806548 2015-02
2 55.522463 6747.878869 2015-03
3 58.354083 5506.381616 2015-04
4 57.294059 6757.408602 2015-05
5 65.974903 4375.495132 2015-06
6 71.072043 3955.254032 2015-07
7 63.998065 4856.173387 2015-08
8 60.254792 4323.800000 2015-09
9 59.406765 4597.236559 2015-10
10 60.726792 4493.763889 2015-11
11 61.901761 4943.141319 2015-12
12 45.578723 4993.399194 2016-01
13 36.752083 6534.337644 2016-02
14 36.818008 5884.690444 2016-03
15 32.618667 5343.191667 2016-04
16 34.691371 5309.384409 2016-05
17 46.266319 5638.537500 2016-06
18 47.502016 5715.833109 2016-07
19 47.602339 5103.193548 2016-08
20 50.405597 5484.695833 2016-09
21 60.182430 4788.928859 2016-10
22 62.581056 5766.394444 2016-11
23 67.595134 4519.766129 2016-12
24 79.492083 6483.298387 2017-01
25 59.837798 5330.943452 2017-02
26 50.959892 5623.149394 2017-03
27 51.717917 5741.669444 2017-04
28 53.772621 5031.903226 2017-05
29 56.258222 5586.527778 2017-06
30 55.252581 4799.477151 2017-07
31 54.084328 4348.514785 2017-08
32 55.816556 3883.879167 2017-09
33 63.925289 5234.800000 2017-10
34 65.430653 5200.188889 2017-11
35 65.151277 7257.255376 2017-12
36 56.511976 6197.444892 2018-01
37 60.877098 6864.555060 2018-02
38 48.279717 8771.240915 2018-03
39 50.400736 5289.247222 2018-04
40 61.633763 4389.743280 2018-05
41 64.348139 4779.433333 2018-06
42 67.783441 4059.907133 2018-07
43 70.363911 4358.579301 2018-08
44 76.914042 4265.988889 2018-09
45 70.362215 6156.939597 2018-10
46 67.042608 5886.720430 2018-11
47 66.623514 6057.558333 2018-12

[116.82543128813965, 137.12802471367735, 121.53421347537002, 94.36154765974193, 117.9425703747738, 66.32059992366896, 55.65133440246909, 75.88000393157053, 71.75860840942802, 77.3857413604204, 73.9996921549133, 79.854615747137, 109.5554866815863, 177.79502686727042, 159.83185271997365, 163.80778899607589, 153.04625503382746, 121.87132168078833, 120.32822130099225, 107.20468125549537, 108.81124588511582, 79.57353826430645, 92.14281212187926, 66.86525840323019, 81.55904481595262, 89.08990077342021, 110.34460901350124, 111.01896237334473, 93.57742165525237, 99.30153419549536, 86.86430741326733, 80.40249272218763, 69.5829244210698, 81.88934481821691, 79.47634125782449, 111.39083873243722, 109.66604518834741, 112.76087824292748, 181.6754818474552, 104.94384864859504, 71.22302832897762, 74.27461642031433, 59.89526470950425, 61.94339145093432, 55.46437030805133, 87.50349342853899, 87.80566041896803, 90.92222820062902]
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per

EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the average monthly on shore wind outputs versus the predicted average monthly prices of energy per EUR/MWH.

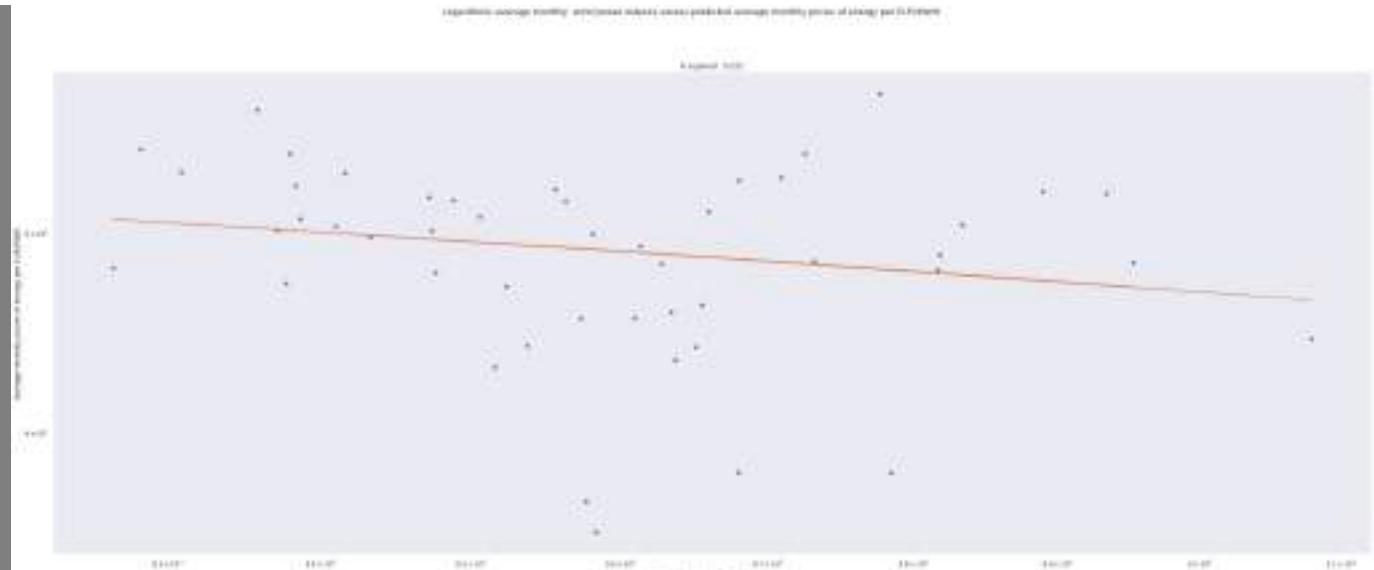
```
In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(Price_Actual)))
Logwindvalues = ((np.log(wind)))
Log = np.polyfit(np.log(Price_Actual), wind1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(wind1), Price_Actual)
Wind_Log = sm.OLS(Price_Actual, wind1).fit()

Wind_Logpred = Wind_Log.predict(wind1)
#OLS Logarithmic summary table
Wind_Log.summary()
#Log
Log = np.polyfit(np.log(wind), Price_Actual, 1)
print(Log)

y = -11.45231819 * Logwindvalues + 156.2322607
#OLS Logarithmic Scatterplots
plt.suptitle("Logarithmic average monthly wind power outputs versus predicted average monthly prices of energy per EUR/MWH")
plt.title("R squared : 0.032")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly outputs")
plt.yscale("log")
plt.xscale("log")
plt.plot(Logwindvalues, Price_Actual, "o")
plt.plot(Logwindvalues, y)
```

[-11.45231819 156.2322607]

Out[]: [`<matplotlib.lines.Line2D at 0x7ff604f425d0>`]



The blue dots represent the observations and the orange line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: Wind_Log.summary() #OLS Logarithmic summary table
```

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.032				
Model:	OLS	Adj. R-squared:	0.011				
Method:	Least Squares	F-statistic:	1.515				
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.225				
Time:	05:27:49	Log-Likelihood:	-178.86				
No. Observations:	48	AIC:	361.7				
Df Residuals:	46	BIC:	365.5				
Df Model:	1						
Covariance Type:	nonrobust						
		coef	std err	t	P> t	[0.025	0.975]
const	67.3579	7.859	8.571	0.000	51.539	83.176	
x1	-0.0017	0.001	-1.231	0.225	-0.005	0.001	
Omnibus:	2.925	Durbin-Watson:	0.436				
Prob(Omnibus):	0.232	Jarque-Bera (JB):	2.043				
Skew:	-0.483	Prob(JB):	0.360				
Kurtosis:	3.300	Cond. No.	2.95e+04				

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 [2] The condition number is large, 2.95e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
influenceWindLog = Wind_Log.get_influence()
#Logarithmic OLS regression residuals
```

```
standardized_residualsWindLog = influenceWindLog.resid_studentized_internal
```

```
print(standardized_residualsWindLog)
```

```
print(Wind_Logpred) # OLS logarithmic predicted values
```

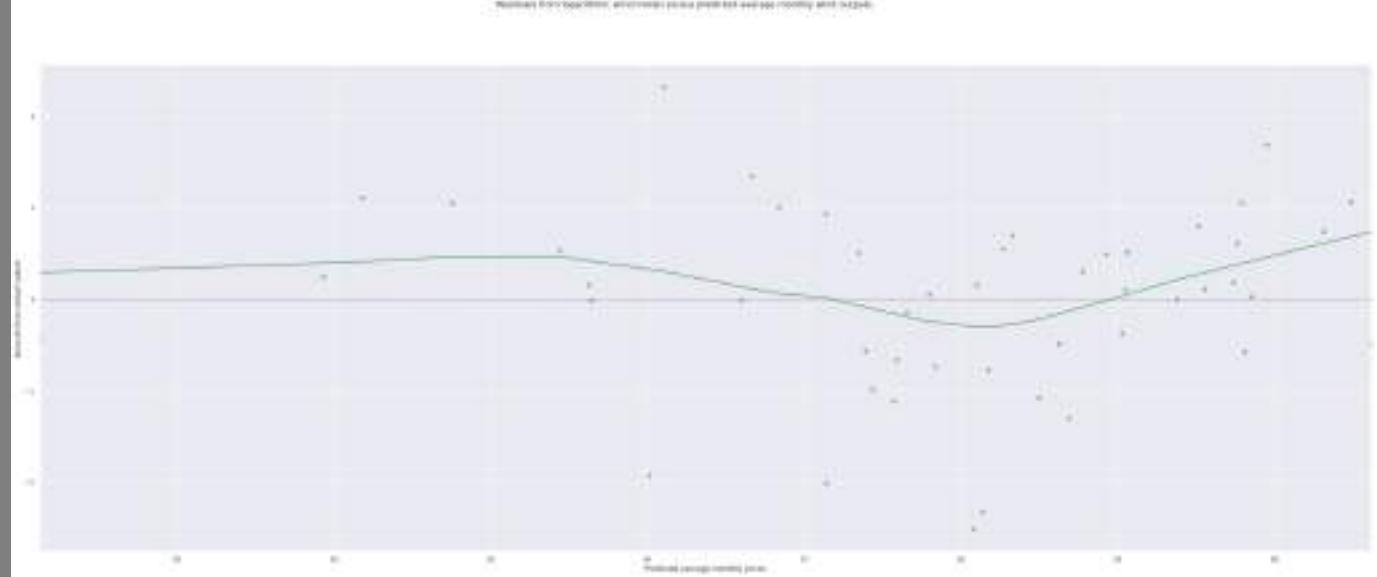
```
[ 1.10907159  0.25423421 -0.01184335  0.05488905  0.16708808  0.61905216
 1.06570343  0.50122839  0.04036697  0.00312906  0.11645406  0.30870957
-1.29342813 -1.91724852 -2.00407466 -2.50714755 -2.30904722 -1.11291025
-0.97830278 -1.07398322 -0.731379  0.11274947  0.51593788  0.80306389
 2.32617494  0.17100549 -0.65328364 -0.55864183 -0.47817444 -0.13777379
-0.37293612 -0.57039163 -0.48421412  0.55726371  0.69974965  1.05649802
-0.00837628  0.54581531 -0.42629472 -0.76551016  0.18903055  0.52316261
 0.75049937  1.05371123  1.69399357  1.35453242  1.00789611  0.93563521]
[54.18258701 53.93235568 55.64084597 57.79657961 55.62429856 59.76024891
 60.48995482 58.92560004 59.85001225 59.37521749 59.55488725 58.77458907
 58.68732139 56.01163858 57.13968483 58.07994235 58.13864522 57.56710437
 57.43288841 58.49667466 57.83423477 59.04236332 57.3450938  59.50973701
 56.10026305 58.10121013 57.59382425 57.38802624 58.62046306 57.65741396
 59.02404728 59.80709754 60.61389001 58.26815345 58.32825213 54.7563655
 56.59661856 55.43824964 52.12748367 58.17361139 59.73550846 59.05885133
 60.30823536 59.78962153 59.95039536 56.66695188 56.83951733 57.13615997]
```

In []:

```
WindLogRatioPredict = Wind_Logpred/Logpred
```

```
plt.suptitle("Residuals from logarithmic wind model versus predicted average monthly wind outputs")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Amount from actual values")
plt.legend("#")
sns.residplot(x = Wind_Logpred, y = standardized_residualsWindLog, lowess = True, color="g")
# OLS Logarithmic average monthly predictions versus residuals
```

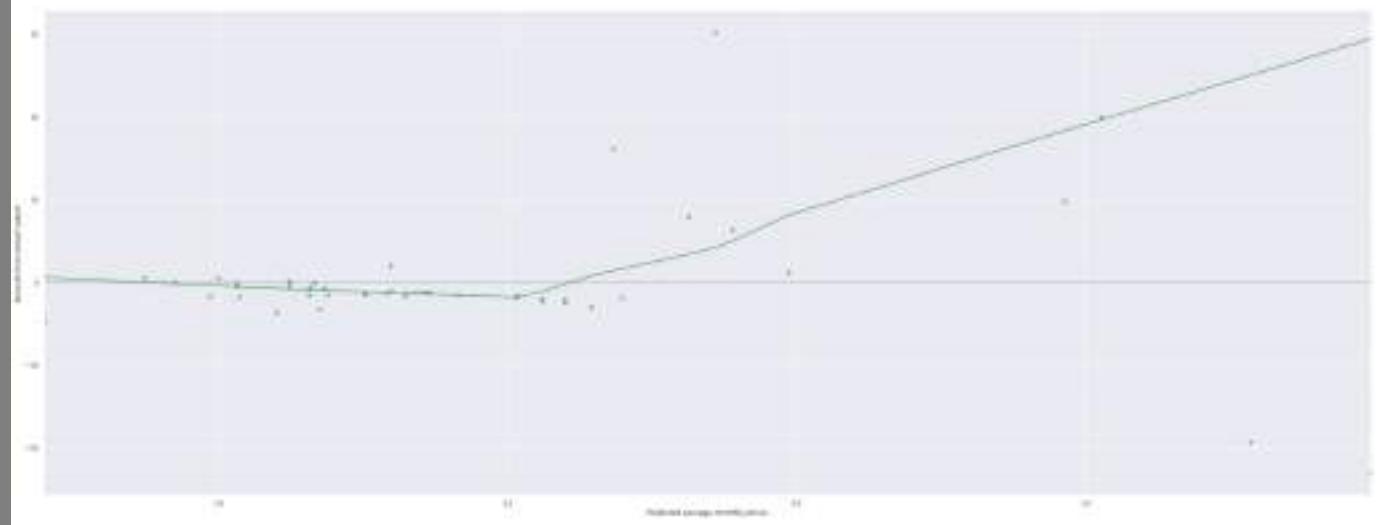
Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6051a3490>



As one can observe, there is a clustered hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. Given these circumstances, it would be reasonable to assume that the logarithmic model of fit is suitable. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals  
plt.suptitle("Predicted average monthly logarithmic wind output to price of energy per EUR/MWH ratio versus residuals")  
plt.xlabel("Predicted average monthly prices")  
plt.ylabel("Amount from actual values")  
plt.legend(..#)  
sns.residplot(x = WindLogRatioPredict, y = standardized_residualsWindLog/standardized_residualsPriceLog,lowess :
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60504d550>
```

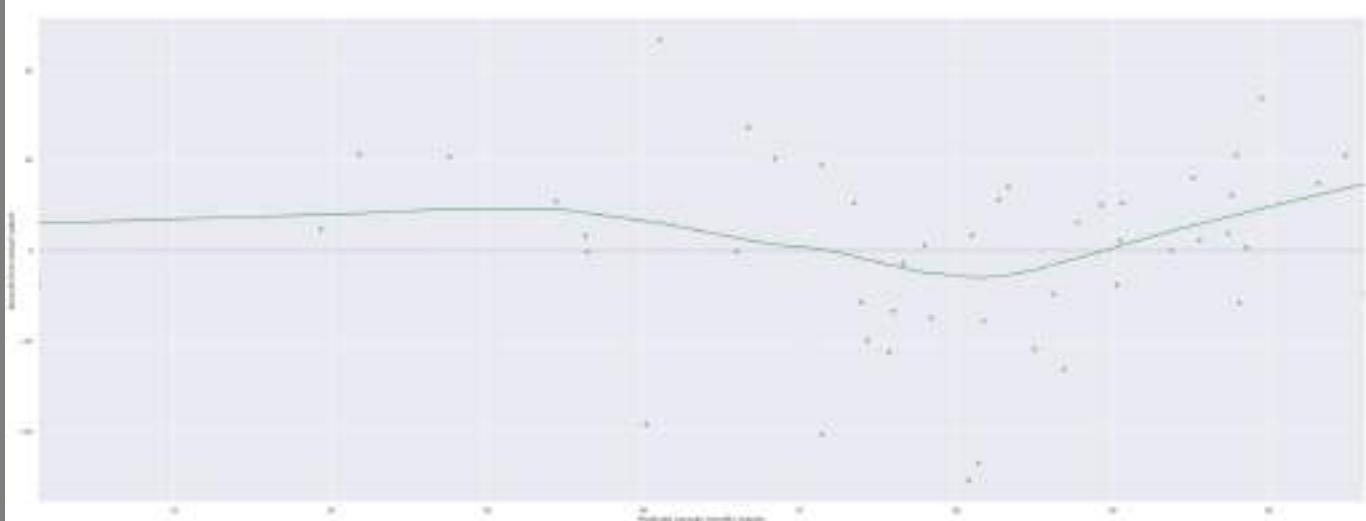


With the exception of a few heteroskedastic outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias otherwise. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly logarithmic wind energy prices per EUR/MWH versus actual values")  
plt.xlabel("Predicted average monthly outputs")
```

```
plt.ylabel("Amount from actual values")
plt.legend(..#")
sns.residplot(x = Wind_Logpred, y = Price_Actual, lowess = True, color="g")
# OLS predicted logarithmic average monthly values versus actual values
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6050ac1d0>



As one can observe this residual plot, one may notice the positive slope in the fitted model. In addition, the observations are spread out in a distinct pattern, indicating bias and homoscedasticity. It would be reasonable to assume that this model does not fit the data. The green dots are the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: influenceWindLog = Wind_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsWindLog = influenceWindLog.resid_studentized_internal

print(standardized_residualsWindLog)
```

```
[ 1.10907159  0.25423421 -0.01184335  0.05488905  0.16708808  0.61905216
 1.06570343  0.50122839  0.04036697  0.00312906  0.11645406  0.30870957
-1.29342813 -1.91724852 -2.00407466 -2.50714755 -2.30904722 -1.11291025
-0.97830278 -1.07398322 -0.731379  0.11274947  0.51593788  0.80306389
 2.32617494  0.17100549 -0.65328364 -0.55864183 -0.47817444 -0.13777379
-0.37293612 -0.57039163 -0.48421412  0.55726371  0.69974965  1.05649802
-0.00837628  0.54581531 -0.42629472 -0.76551016  0.18903055  0.52316261
 0.75049937  1.05371123  1.69399357  1.35453242  1.00789611  0.93563521]
```

In []:

```
In [ ]: modelwindonshore = stats.linregress([7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886
56.38385416666666,
55.522462987886975,
58.35408333333333,
57.29405913978498,
65.9749027777778,
71.07204301075271,
63.99806451612899,
60.254791666666634,
59.40676510067113,
60.72679166666668,
61.901760752688226,
45.57872311827956,
36.75208333333374,
36.81800807537014,
32.61866666666666,
34.691370967741896,
46.266319444444434,
47.50201612903221,
47.6023387096774,
50.4055972222224,
60.182429530201404,
62.5810555555558,
67.5951344086021,
79.49208333333331,
59.83779761904767,
50.95989232839837,
51.71791666666662,
```

```
53.77262096774189,  
56.25822222222224,  
55.252580645161316,  
54.08432795698931,  
55.81655555555558,  
63.92528859060403,  
65.4306527777781,  
65.15127688172035,  
56.51197580645163,  
60.877098214285674,  
48.279717362045766,  
50.40073611111113,  
61.633763440860214,  
64.34813888888884,  
67.78344086021498,  
70.363911290932262,  
76.9140416666666,  
70.36221476510062,  
67.0426075268817,  
66.62351388888881])
```

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the linear model for the average monthly on shore wind outputs versus the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Linear OLS regression  
wind1 = wind  
wind1 = sm.add_constant(wind1)  
modelwindreg = sm.OLS(Price_Actual, wind1).fit()  
predictionsWind = modelwindreg.predict(wind1)  
  
modelwindreg.summary()  
#OLS Linear Summary Table
```

```
Out[ ]: OLS Regression Results  
Dep. Variable: y R-squared: 0.032  
Model: OLS Adj. R-squared: 0.011  
Method: Least Squares F-statistic: 1.515  
Date: Wed, 30 Nov 2022 Prob (F-statistic): 0.225  
Time: 05:27:51 Log-Likelihood: -178.86  
No. Observations: 48 AIC: 361.7  
Df Residuals: 46 BIC: 365.5  
Df Model: 1  
Covariance Type: nonrobust  
  
coef std err t P>|t| [0.025 0.975]  
const 67.3579 7.859 8.571 0.000 51.539 83.176  
x1 -0.0017 0.001 -1.231 0.225 -0.005 0.001  
  
Omnibus: 2.925 Durbin-Watson: 0.436  
Prob(Omnibus): 0.232 Jarque-Bera (JB): 2.043  
Skew: -0.483 Prob(JB): 0.360  
Kurtosis: 3.300 Cond. No. 2.95e+04
```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.95e+04. This might indicate that there are strong multicollinearity or other numerical problems.

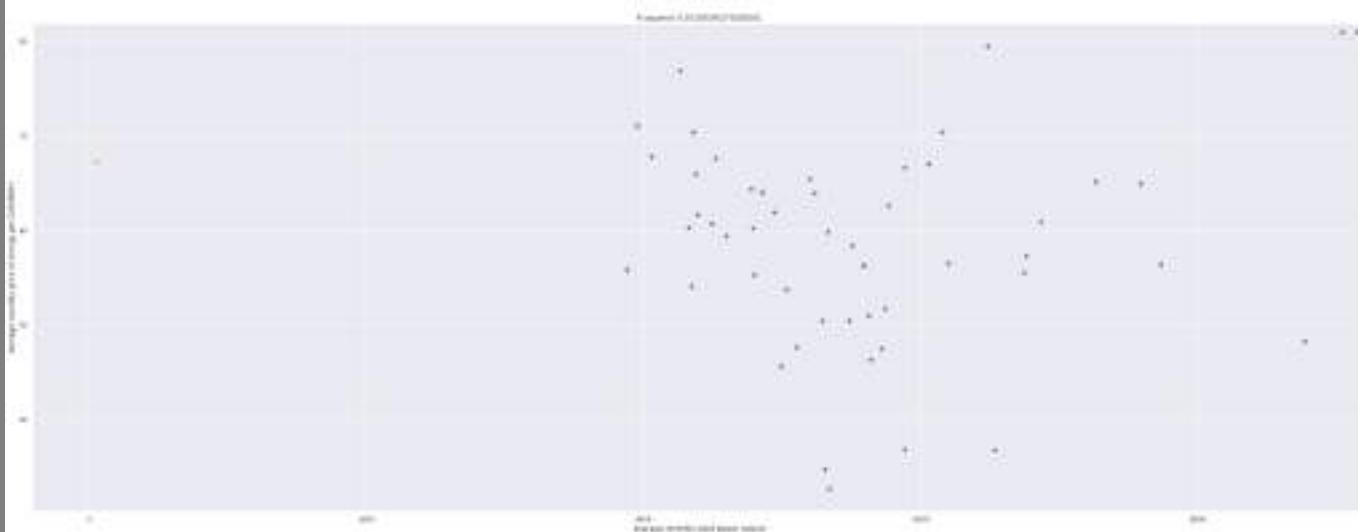
```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(wind, Price_Actual)  
print("slope: %f intercept: %f" % (slope, intercept))  
  
#OLS Linear Scatterplot  
plt.suptitle("Average monthly wind outputs to the predicted linear average monthly prices of energy per EUR/MWH")
```

```

plt.plot(wind, Price_Actual, "o")
plt.title(f"R squared: {modelwindonshore.rvalue**2}")
f = lambda x: -0.001736 *x + 67.357851
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.legend("#")
plt.ylabel('Average monthly price of energy per EUR/MWh')
plt.xlabel('Average monthly wind power output')
plt.show()

```

slope: -0.001736 intercept: 67.357851



There is a very weak yet negative correlation between the average monthly outputs and the respective months. The blue dots are the observations and orange line is the model of best fit.

In []: `print(predictionsWind)
#Linear OLS Predicted Values`

```
[54.18258701 53.93235568 55.64084597 57.79657961 55.62429856 59.76024891
 60.48995482 58.92560004 59.85001225 59.37521749 59.55488725 58.77458907
 58.68732139 56.01163858 57.13968483 58.07994235 58.13864522 57.56710437
 57.43288841 58.49667466 57.83423477 59.04236332 57.3450938 59.50973701
 56.10026305 58.10121013 57.59382425 57.38802624 58.62046306 57.65741396
 59.02404728 59.80709754 60.61389001 58.26815345 58.32825213 54.7563655
 56.59661856 55.43824964 52.12748367 58.17361139 59.73550846 59.05885133
 60.30823536 59.78962153 59.95039536 56.66695188 56.83951733 57.13615997]
```

In []: `#Linear OLS regression residuals
influencewindreg = modelwindreg.get_influence()

standardized_residualsWind = influencewindreg.resid_studentized_internal

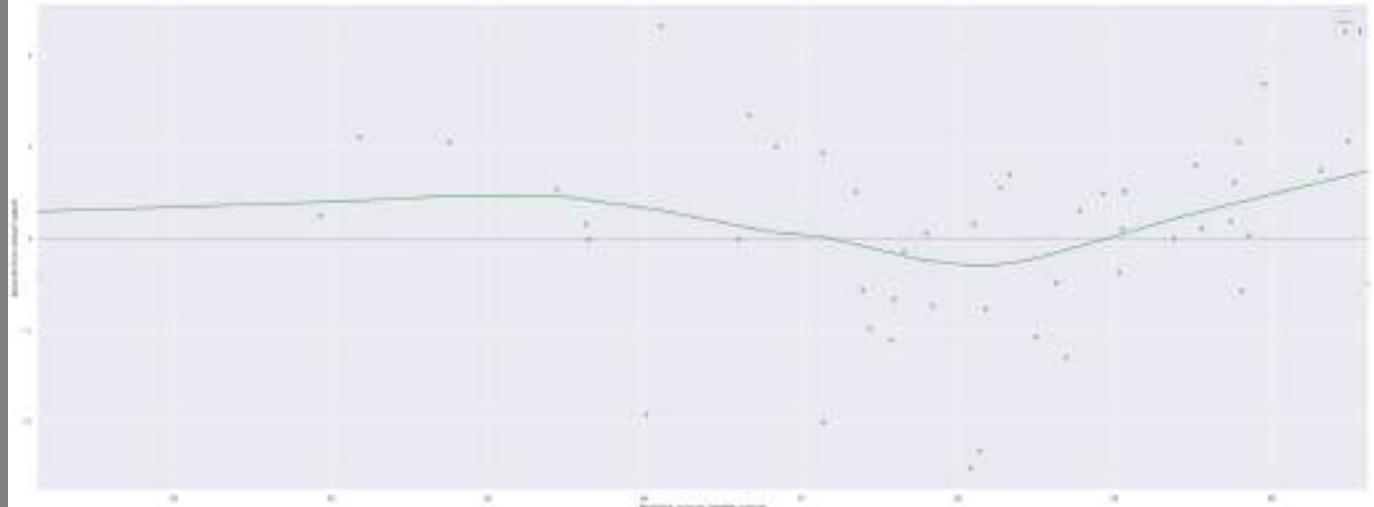
print(standardized_residualsWind)`

```
[ 1.10907159  0.25423421 -0.01184335  0.05488905  0.16708808  0.61905216
 1.06570343  0.50122839  0.04036697  0.00312906  0.11645406  0.30870957
-1.29342813 -1.91724852 -2.00407466 -2.50714755 -2.30904722 -1.11291025
-0.97830278 -1.07398322 -0.731379   0.11274947  0.51593788  0.80306389
 2.32617494  0.17100549 -0.65328364 -0.55864183 -0.47817444 -0.13777379
-0.37293612 -0.57039163 -0.48421412  0.55726371  0.69974965  1.05649802
-0.00837628  0.54581531 -0.42629472 -0.76551016  0.18903055  0.52316261
 0.75049937  1.05371123  1.69399357  1.35453242  1.00789611  0.93563521]
```

In []: `#Predicted OLS linear values versus residual values
sns.residplot(x = predictionsWind, y = standardized_residualsWind, lowess = True, color ="g")
plt.suptitle("Wind power residuals from linear model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#")`

Out[]: <matplotlib.legend.Legend at 0x7ff60521c990>



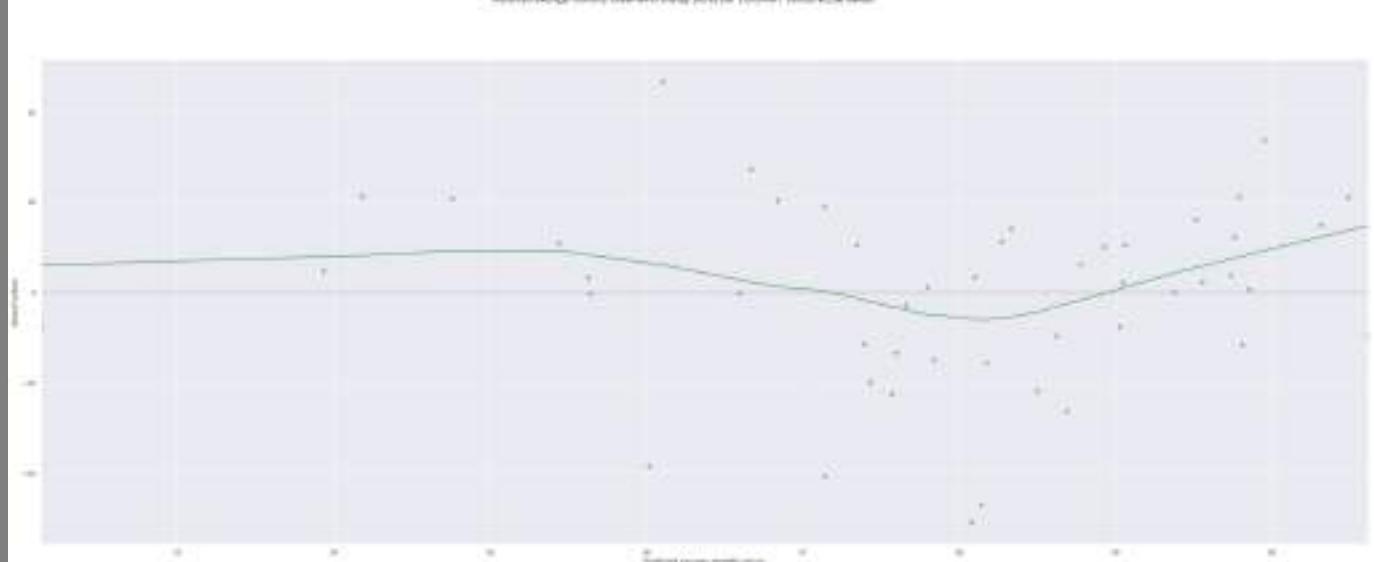
In []:

As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. Given these circumstances, it would be reasonable to assume that the quadratic model of fit is suitable. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

In []:

```
plt.suptitle("Predicted average monthly linear wind energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsWind, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

Out[]:



The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the quadratic model used for the average monthly on shore wind output versus the predicted average monthly prices of energy per EUR/MWH.

In []:

```
#Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)
```

```

modelWindquad = np.poly1d(np.polyfit(wind,Price_Actual,2))
print(modelWindquad)

wind1 = sm.add_constant(wind1)
wind2 = polynomial_features.fit_transform(wind1)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(wind2)

Wind_Q = poly.fit(X_poly, wind)
lin2 = LinearRegression()
lin2.fit(X_poly, wind1)
Wind_Quad = sm.OLS(Price_Actual, wind2).fit()

# OLS Predicted Quadratic values
Wind_ypred = Wind_Quad.predict(wind2)

#OLS Quadratic Summary Table
Wind_Quad.summary()

##OLS Quadratic Scatterplot
polyline = np.linspace(start = 0, stop =10000 , num = 100)
plt.plot(polyline, modelWindquad(polyline))
plt.title("R squared : 0.072")
plt.scatter(wind, Price_Actual, color = 'blue')
plt.suptitle('Quadratic for average monthly wind power outputs versus predicted average monthly prices of energy')
plt.xlabel('Average monthly outputs')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()

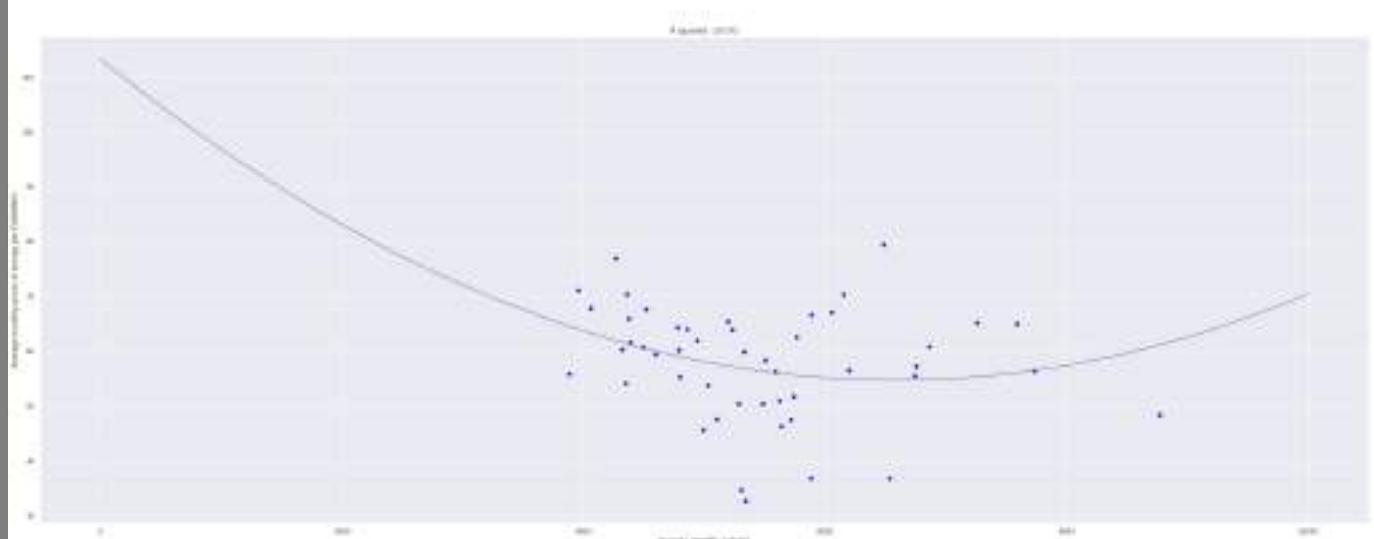
```

$$1.352e-06 x^2 - 0.0178 x + 113.3$$

```

/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:214: RuntimeWarning: Glyph 9 missing from current font.
    font.set_text(s, 0.0, flags=flags)
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py:183: RuntimeWarning: Glyph 9 missing from current font.
    font.set_text(s, 0, flags=flags)

```



The blue dots represent the observations and the blue line is the model of best fit. This is a very weak and positive correlation between the average monthly outputs and the average monthly price of energy per EUR/MWH.

In []: Wind_Quad.summary()

Out[]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.072			
Model:	OLS	Adj. R-squared:	0.031			
Method:	Least Squares	F-statistic:	1.749			
Date:	Wed, 30 Nov 2022	Prob (F-statistic):	0.186			
Time:	05:27:54	Log-Likelihood:	-177.85			
No. Observations:	48	AIC:	361.7			
Df Residuals:	45	BIC:	367.3			
Df Model:	2					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	37.7714	11.269	3.352	0.002	15.074	60.468
x1	37.7714	11.269	3.352	0.002	15.074	60.468
x2	-0.0089	0.006	-1.537	0.131	-0.021	0.003
x3	37.7714	11.269	3.352	0.002	15.074	60.468
x4	-0.0089	0.006	-1.537	0.131	-0.021	0.003
x5	1.352e-06	9.68e-07	1.397	0.169	-5.97e-07	3.3e-06
Omnibus:	1.666	Durbin-Watson:	0.542			
Prob(Omnibus):	0.435	Jarque-Bera (JB):	0.984			
Skew:	-0.330	Prob(JB):	0.612			
Kurtosis:	3.236	Cond. No.	3.61e+24			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 4.12e-33. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
influenceWind_Quad = Wind_Quad.get_influence() #Quadratic OLS residuals

standardized_residualsWindQuad = influenceWind_Quad.resid_studentized_internal

print(standardized_residualsWindQuad)
```

```
[ 0.93586929 -0.00830806  0.08109441  0.20942334  0.26015218  0.47435813
 0.74067191  0.52595166 -0.13688596 -0.06267143  0.01254035  0.35603551
-1.25061901 -1.81226918 -1.86167349 -2.41343557 -2.21723385 -0.96525009
-0.82306221 -1.00325372 -0.59223066  0.11308161  0.70105837  0.7180244
 2.5034432   0.30201183 -0.49883929 -0.39437812 -0.41677258  0.02224811
-0.37432616 -0.74718992 -0.93033158  0.67606455  0.81340379  1.01276741
 0.16336367  0.61769032 -2.08205461 -0.65495209  0.04336289  0.52474428
 0.46488514  0.90936439  1.52477149  1.55349994  1.20524838  1.13278878]
```

In []:

```
print(Wind_ypred) # OLS quadratic predicted values
```

```
[56.04329713 56.46171036 54.72189377 56.26126458 54.72619279 61.29002981
 64.03985535 58.73010551 61.60254104 60.03150036 60.60223442 58.33367544
 58.11390501 54.68995118 55.35077886 56.77346637 56.8885791 55.89922913
 55.70936511 57.6575438 56.32518146 59.05064677 55.59390786 60.45608846
 54.70057018 56.81481424 55.93895536 55.64950412 57.95015221 56.03607305
 58.99955677 61.45223205 64.55432381 57.15346466 57.28149426 55.29584525
 54.89021756 54.79142682 61.14256347 56.95861401 61.2051659 59.09689504
 63.31041944 61.39149539 61.96058239 54.93495932 55.06352678 55.34693688]
```

In []:

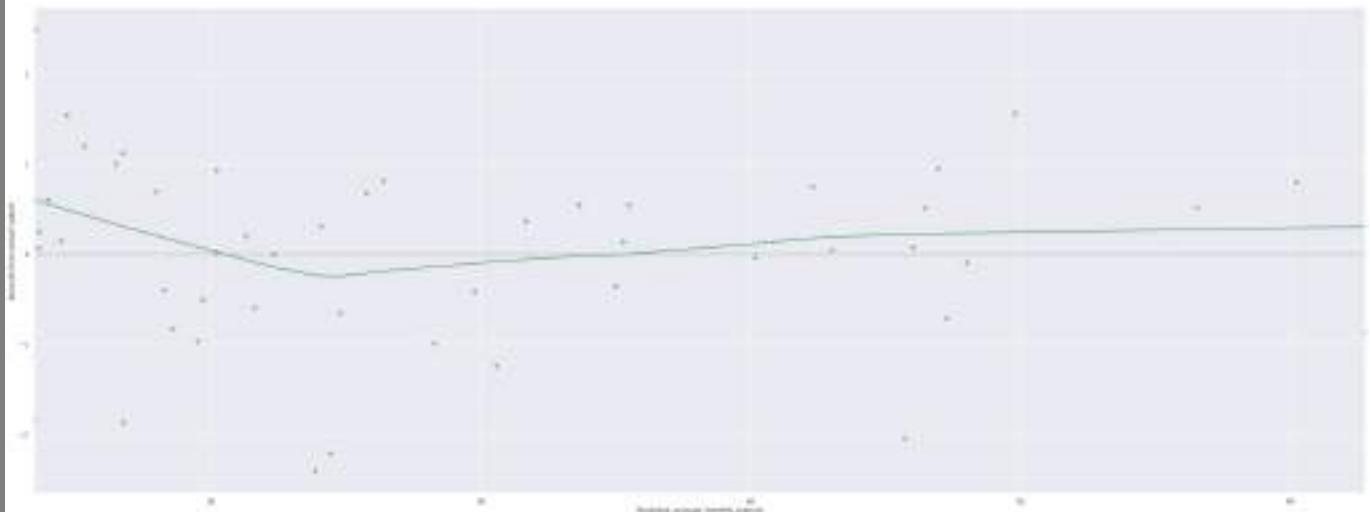
```
#Predicted average monthly OLS quadratic values versus residuals
plt.suptitle("Wind power residuals from quadratic model versus predicted values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(".#")

sns.residplot(x = Wind_ypred, y = standardized_residualsWindQuad, lowess = True, color="g")
```

Out[]:

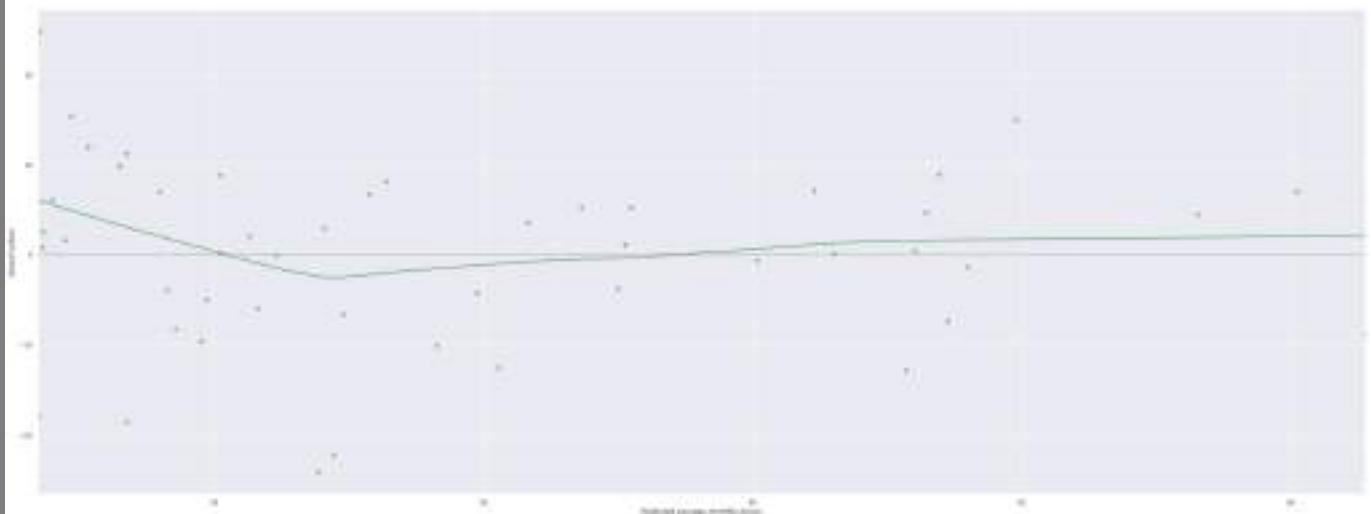
<matplotlib.axes._subplots.AxesSubplot at 0x7ff604746410>



As one can observe, there is a slight curve along the lowess line in the residual plot. Furthermore, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. Given these circumstances, it would be reasonable to assume that the quadratic model of fit is suitable. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted average monthly quadratic wind energy prices per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Wind_ypred, y = Price_Actual, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6046bfed0>
```



As one can observe this residual plot, one may notice some sharp spikes in the fitted model, which form a nonlinear pattern. However, the observations are spread out in a "C" shaped pattern; indicating heteroskedasticity and bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the list directly above will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The results are the outputs divided by the respective average monthly prices of energy per EUR/MWH.

If the data is deemed to be stationary based off the given tests below, then that means that seasonality and trends are not factors in the values of the tested data. If the data is deemed to be non stationary, than seasonality and trends are indeed factors after all.

```
In [ ]: #Dataframes analyzed by resource
dfwind = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.294059139784,
```

```

"Wind" : [7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375..]
print(dfwind)
df_wind=pd.DataFrame.from_dict(dfwind, orient = "columns")
print(df_wind)
df_wind["Ratio"] = df_wind["Wind"]/df_wind["Price"]
pdToListWind = list(df_wind["Ratio"])
print(pdToListWind)
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_wind["Ratio"])

test_result=adfuller(df_wind["Ratio"])

from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_wind["Ratio"])
plt.suptitle("Partialautocorrelations of Wind ratio")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
plt.show

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_wind["Ratio"])
plt.suptitle(" Autocorrelations of average monthly Wind ratio")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
plt.show

Wind_Ratio_Autocorrelations = sm.tsa.acf(df_wind["Ratio"],fft=False) #Autocorrelations
print(Wind_Ratio_Autocorrelations)

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.4920833333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.4306527777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Wind': [7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.495132127956, 3955.2540322580644, 4856.173387096775, 4323.8, 4597.236559139785, 4493.763888888889, 4943.14131897712, 4993.399193548387, 6534.337643678161, 5884.69044145357, 5343.191666666667, 5309.384408602151, 5638.5375, 5715.833109017497, 5103.193548387097, 5484.695833333333, 4788.928859060403, 5766.394444444444, 4519.7661290322585, 6483.298387096775, 5330.943452380952, 5623.149394347241, 5741.669444444445, 5031.903225806452, 5586.527777777777, 4799.477150537635, 4348.514784946236, 3883.8791666666666, 5234.8, 5200.18888888889, 7257.2553763440865, 6197.444892473119, 6864.555059523809, 8771.240915208613, 5289.247222222222, 4389.743279569892, 4779.433333333333, 4059.907133243607, 4358.579301075269, 4265.988888888889, 6156.939597315436, 5886.720430107527, 6057.558333333333], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}}

	Price	Wind	Dates
0	64.949019	7587.697135	2015-01
1	56.383854	7731.806548	2015-02
2	55.522463	6747.878869	2015-03
3	58.354083	5506.381616	2015-04
4	57.294059	6757.408602	2015-05
5	65.974903	4375.495132	2015-06
6	71.072043	3955.254032	2015-07
7	63.998065	4856.173387	2015-08
8	60.254792	4323.800000	2015-09
9	59.406765	4597.236559	2015-10
10	60.726792	4493.763889	2015-11

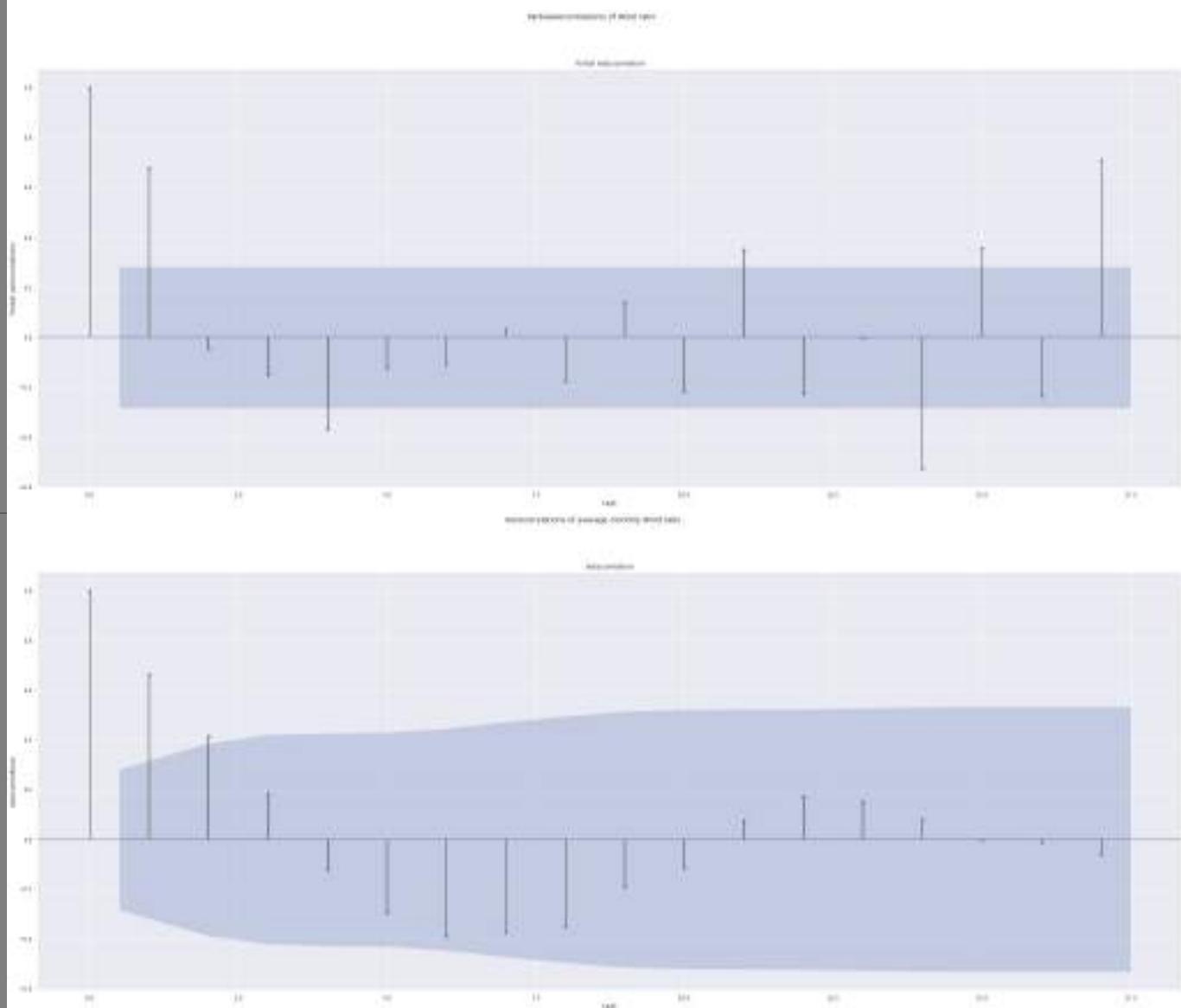
```

11 61.901761 4943.141319 2015-12
12 45.578723 4993.399194 2016-01
13 36.752083 6534.337644 2016-02
14 36.818008 5884.690444 2016-03
15 32.618667 5343.191667 2016-04
16 34.691371 5309.384409 2016-05
17 46.266319 5638.537500 2016-06
18 47.502016 5715.833109 2016-07
19 47.602339 5103.193548 2016-08
20 50.405597 5484.695833 2016-09
21 60.182430 4788.928859 2016-10
22 62.581056 5766.394444 2016-11
23 67.595134 4519.766129 2016-12
24 79.492083 6483.298387 2017-01
25 59.837798 5330.943452 2017-02
26 50.959892 5623.149394 2017-03
27 51.717917 5741.669444 2017-04
28 53.772621 5031.903226 2017-05
29 56.258222 5586.527778 2017-06
30 55.252581 4799.477151 2017-07
31 54.084328 4348.514785 2017-08
32 55.816556 3883.879167 2017-09
33 63.925289 5234.800000 2017-10
34 65.430653 5200.188889 2017-11
35 65.151277 7257.255376 2017-12
36 56.511976 6197.444892 2018-01
37 60.877098 6864.555060 2018-02
38 48.279717 8771.240915 2018-03
39 50.400736 5289.247222 2018-04
40 61.633763 4389.743280 2018-05
41 64.348139 4779.433333 2018-06
42 67.783441 4059.907133 2018-07
43 70.363911 4358.579301 2018-08
44 76.914042 4265.988889 2018-09
45 70.362215 6156.939597 2018-10
46 67.042608 5886.720430 2018-11
47 66.623514 6057.558333 2018-12
[116.82543128813965, 137.12802471367735, 121.53421347537002, 94.36154765974193, 117.9425703747738, 66.3205999236
6896, 55.65133440246909, 75.88000393157053, 71.75860840942802, 77.3857413604204, 73.9996921549133, 79.8546157471
37, 109.5554866815863, 177.79502686727042, 159.83185271997365, 163.80778899607589, 153.04625503382746, 121.87132
168078833, 120.32822130099225, 107.20468125549537, 108.81124588511582, 79.57353826430645, 92.14281212187926, 66.
86525840323019, 81.55904481595262, 89.0890077342021, 110.34460901350124, 111.01896237334473, 93.57742165525237,
99.30153419549536, 86.86430741326733, 80.40249272218763, 69.5829244210698, 81.88934481821691, 79.47634125782449,
111.39083873243722, 109.66604518834741, 112.76087824292748, 181.6754818474552, 104.94384864859504, 71.2230283289
7762, 74.27461642031433, 59.89526470950425, 61.94339145093432, 55.46437030805133, 87.50349342853899, 87.80566041
896803, 90.92222820062902]
ADF Test Statistic : -3.02809890904456
p-value : 0.032352341320210996
#Lags Used : 0
Number of Observations : 47
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary
[ 1.          0.66348814  0.41549751  0.18271922 -0.12179606 -0.29471681
 -0.38905653 -0.37434493 -0.35059378 -0.19254003 -0.11562872  0.07575072
  0.16968995  0.15074981  0.08087079 -0.0034258  -0.01157623 -0.06184103
 -0.00678974  0.00430297  0.05413984  0.08310186  0.16878385  0.18269667
  0.11903601  0.10810524 -0.07963925 -0.22816267 -0.27800166 -0.27877687
 -0.26402623 -0.18467315 -0.1173601  -0.02474459  0.1309573   0.13177234
  0.16157944  0.17476292  0.09534951  0.00720454 -0.04405933]

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.

FutureWarning,



In []:

The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial and autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly on shore wind outputs and the average monthly prices of energy per EUR/MWH.

In []:

```
df_wind['First Difference Ratio'] = df_wind["Ratio"]- df_wind["Ratio"].shift(1) # Seasonality values
df_wind['Seasonal Difference Ratio']=df_wind["Ratio"]- df_wind["Ratio"].shift(12)
df_wind.head()
```

```
df_wind['Seasonal Difference Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff604542610>
```



```
In [ ]:
```

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Bell Curves
```

```
windResults_mean = np.mean(df_wind["Ratio"])
windResults_std = np.std(df_wind["Ratio"])

windResultspdf = stats.norm.pdf(df_wind["Ratio"].sort_values(), windResults_mean, windResults_std)

plt.plot(df_wind["Ratio"].sort_values(), windResultspdf)
plt.xlim([0,200])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_wind["Ratio"])}') # Output/Price per EUR/MWH Bell Curve
plt.suptitle("Frequency distribution of average monthly wind outputs to energy prices per EUR/MWH ratios (scaled")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

#Bell Curves

windResults_mean = np.mean(df_wind["Wind"])
windResults_std = np.std(df_wind["Wind"])

windResultspdf = stats.norm.pdf(df_wind["Wind"].sort_values(), windResults_mean, windResults_std)

plt.plot(df_wind["Wind"].sort_values(), windResultspdf)
plt.xlim([0,10000])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_wind["Wind"])}')
plt.suptitle("Frequency distribution of average monthly wind outputs (scaled in decimals) " )
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

These bell shaped curves are skewed to the right. Hence, they have a asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.09589232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'Wind': [7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.495132127956, 3955.2540322580644, 4856.173387096775, 4323.8, 4597.236559139785, 4493.763888888889, 4943.14131897712, 4993.399193548387, 6534.337643678161, 5884.690444145357, 5343.191666666667, 5309.384408602151, 5638.5375, 5715.833109017497, 5103.193548387097, 5484.695833333333, 4788.928859060403, 5766.3944444444, 4519.7661290322585, 6483.298387096775, 5330.943452380952, 5623.149394347241, 5741.669444444445, 5031.903225806452, 5586.527777777777, 4799.477150537635, 4348.514784946236, 3883.879166666666, 5234.8, 5200.188888888889, 7257.2553763440865, 6197.444892473119, 6864.555059523809, 8771.240915208613, 5289.247222222222, 4389.743279569892, 4779.433333333333, 4059.907133243607, 4358.579301075269, 4265.988888888889, 6156.939597315436, 5886.720430107527, 6057.558333333333], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}]
```

	Price	Wind	Dates
0	64.949019	7587.697135	2015-01
1	56.383854	7731.806548	2015-02
2	55.522463	6747.878869	2015-03
3	58.354083	5506.381616	2015-04
4	57.294059	6757.408602	2015-05
5	65.974903	4375.495132	2015-06
6	71.072043	3955.254032	2015-07
7	63.998065	4856.173387	2015-08
8	60.254792	4323.800000	2015-09
9	59.406765	4597.236559	2015-10
10	60.726792	4493.763889	2015-11
11	61.901761	4943.141319	2015-12
12	45.578723	4993.399194	2016-01
13	36.752083	6534.337644	2016-02
14	36.818008	5884.690444	2016-03
15	32.618667	5343.191667	2016-04
16	34.691371	5309.384409	2016-05
17	46.266319	5638.537500	2016-06
18	47.502016	5715.833109	2016-07
19	47.602339	5103.193548	2016-08
20	50.405597	5484.695833	2016-09
21	60.182430	4788.928859	2016-10
22	62.581056	5766.394444	2016-11
23	67.595134	4519.766129	2016-12
24	79.492083	6483.298387	2017-01
25	59.837798	5330.943452	2017-02
26	50.959892	5623.149394	2017-03
27	51.717917	5741.669444	2017-04
28	53.772621	5031.903226	2017-05
29	56.258222	5586.527778	2017-06
30	55.252581	4799.477151	2017-07
31	54.084328	4348.514785	2017-08
32	55.816556	3883.879167	2017-09
33	63.925289	5234.800000	2017-10
34	65.430653	5200.188889	2017-11
35	65.151277	7257.255376	2017-12
36	56.511976	6197.444892	2018-01
37	60.877098	6864.555060	2018-02
38	48.279717	8771.240915	2018-03
39	50.400736	5289.247222	2018-04
40	61.633763	4389.743280	2018-05
41	64.348139	4779.433333	2018-06
42	67.783441	4059.907133	2018-07
43	70.363911	4358.579301	2018-08
44	76.914042	4265.988889	2018-09
45	70.362215	6156.939597	2018-10
46	67.042608	5886.720430	2018-11
47	66.623514	6057.558333	2018-12

[116.82543128813965, 137.12802471367735, 121.53421347537002, 94.36154765974193, 117.9425703747738, 66.32059992366896, 55.65133440246909, 75.88000393157053, 71.75860840942802, 77.3857413604204, 73.9996921549133, 79.854615747137, 109.5554866815863, 177.79502686727042, 159.83185271997365, 163.80778899607589, 153.04625503382746, 121.87132168078833, 120.32822130099225, 107.20468125549537, 108.81124588511582, 79.57353826430645, 92.14281212187926, 66.86525840323019, 81.55904481595262, 89.08990077342021, 110.34460901350124, 111.01896237334473, 93.57742165525237, 99.30153419549536, 86.86430741326733, 80.40249272218763, 69.5829244210698, 81.88934481821691, 79.47634125782449, 111.39083873243722, 109.66604518834741, 112.76087824292748, 181.6754818474552, 104.94384864859504, 71.22302832897762, 74.27461642031433, 59.89526470950425, 61.94339145093432, 55.46437030805133, 87.50349342853899, 87.80566041896803, 90.92222820062902]

In []: #ADF Tests
from statsmodels.tsa.stattools import adfuller

```

def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_wind["Wind"])

test_result=adfuller(df_wind["Wind"])

```

ADF Test Statistic : -5.052984930234936
 p-value : 1.7414361496316123e-05
 #Lags Used : 5
 Number of Observations : 42
 strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

In [ ]:
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(df_wind["Wind"])
plt.suptitle("Partialautocorrelations of Wind")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show

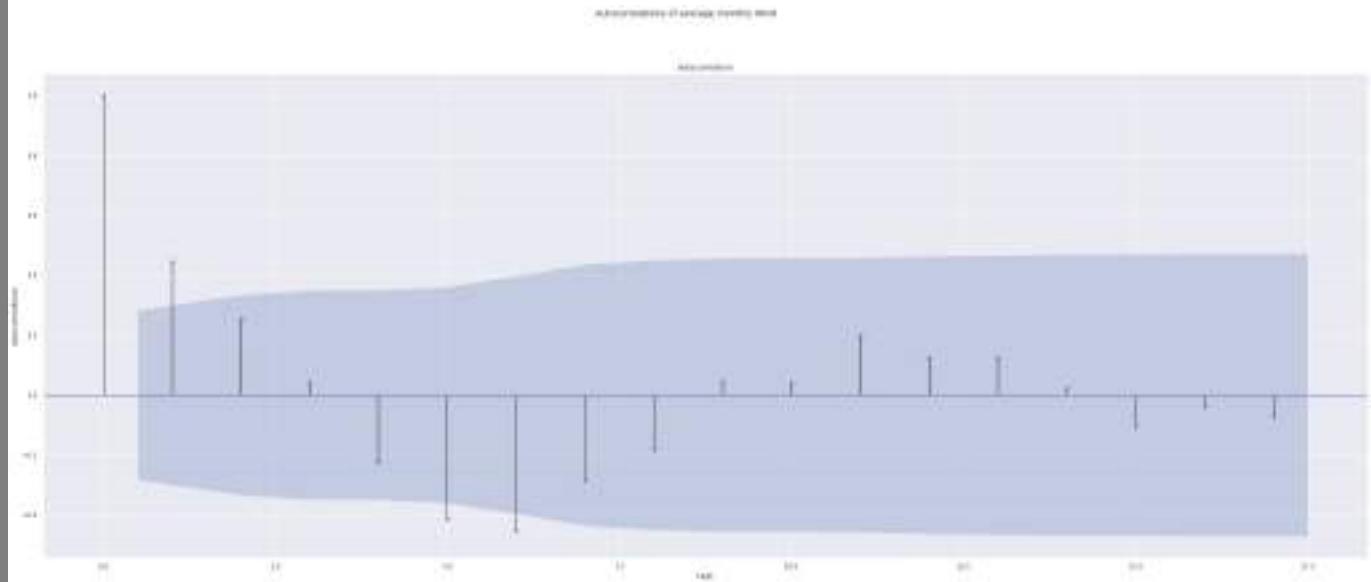
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_wind["Wind"])
plt.suptitle(" Autocorrelations of average monthly Wind")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
Wind_Autocorrelations = sm.tsa.acf(df_wind["Wind"], fft=False) #Autocorrelations
print(Wind_Autocorrelations)

```

[1. 0.44590021 0.25312566 0.04141429 -0.22441314 -0.41414517
 -0.45288423 -0.2849486 -0.18209934 0.04537576 0.04160762 0.19681751
 0.123513 0.123667 0.02128756 -0.10434801 -0.03851228 -0.07025643
 -0.05210998 -0.09234339 0.05105924 0.0679168 0.10070043 0.10763093
 0.11566576 0.09767418 -0.04591504 -0.12152736 -0.18291612 -0.20842968
 -0.2555668 -0.13138754 -0.08395882 0.08356122 0.27937109 0.21441609
 0.21074861 0.2173207 0.09427125 -0.11039446 -0.1697777]

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
 FutureWarning,





The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation. Meanwhile, the lags within partial autocorrelation plots depend on the lag directly beforehand.

As one can observe, there is statistical significance in the partial and autocorrelation plot, indicating that the lags directly beforehand influenced the average monthly shore wind outputs.

```
In [ ]: df_wind['First Difference'] = df_wind["Wind"] - df_wind["Wind"].shift(1) # Seasonality values  
df_wind['Seasonal Difference']=df_wind["Wind"] - df_wind["Wind"].shift(12)  
df_wind.head()  
  
plt.suptitle("Seasonal Difference of average monthly wind output to price of energy per EUR/MWH")  
plt.ylabel('Seasonality')  
plt.xlabel('Months')  
  
df_wind['Seasonal Difference'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6045154d0>
```



The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted

between the average monthly outputs for this particular resources and the average monthly prices of energy per EUR/MWH.

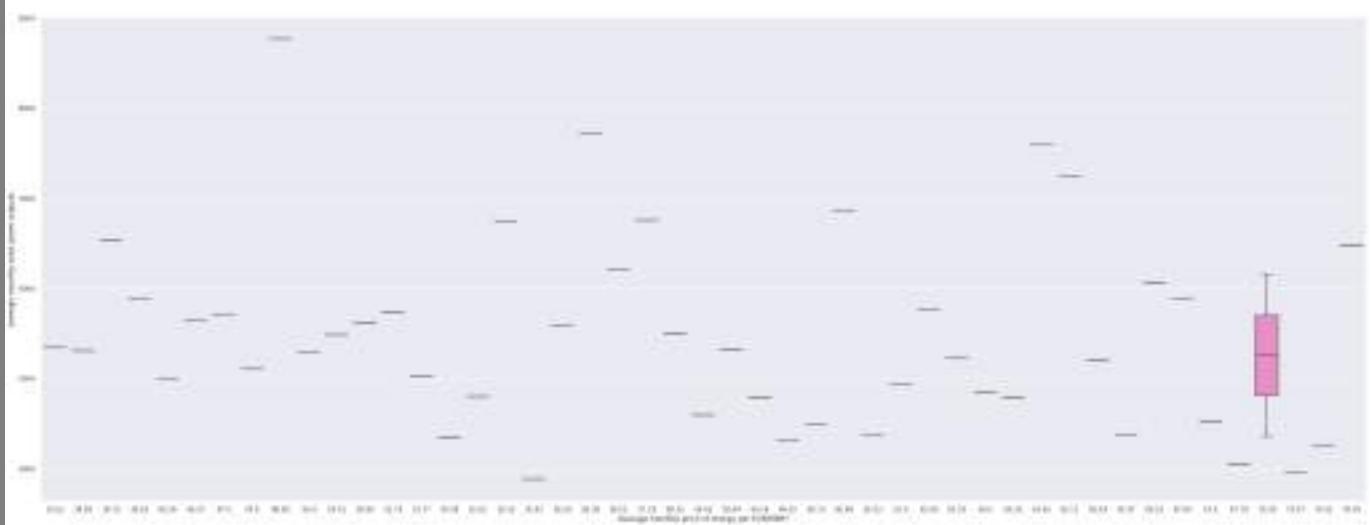
```
In [ ]: df_wind.describe(include = 'all') # Description Tables
```

	Price	Wind	Dates	Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN
mean	57.859848	5469.944902	NaN	98.792748	-32.556145	33.370036
std	10.320573	1061.177091	NaN	30.821488	1081.195011	1218.412210
min	32.618667	3883.879167	NaN	55.464370	-3481.993693	-2594.297942
25%	51.528411	4733.884140	NaN	77.009307	-672.707087	-767.782684
50%	59.622281	5320.163930	NaN	91.532520	-33.807258	-212.365499
75%	64.999583	5929.429906	NaN	111.733349	419.533742	745.433555
max	79.492083	8771.240915	NaN	181.675482	2057.066487	3148.091521

Below is the box and whisker plot for the distribution of the average monthly outputs of wind power and its the average monthly prices of energy per EUR/MWH.

```
In [ ]: # Box and Whisker Plots
sns.set(style="darkgrid")

plt.suptitle('Distribution of average monthly wind power outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly wind power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')
sns.boxplot(x=Rounded_Y, y=[7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408])
plt.show()
```



This box and whisker plot depicts the frequencies between the distribution of the monthly average output of wind power produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of wind power produced was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH,which was slightly above and below 5500 units. When the price of energy per EUR/MWH was at its lowest(at approximately 32.62 euros per MWH), wind power output was roughly 5250 units.The lowest amount produced, whcih was slightly below 4000 units, was at the price of approximately 55.82 EUR/MWH. At approximately 48.28 EUR/MWH, wind power produced the highest output amount,which was slightly below 9000 units. When the monthly average price of energy per EUR/MWH was at its highest, at approximately 79.13 EUR/MWH, the output was at roughly 6500 units.

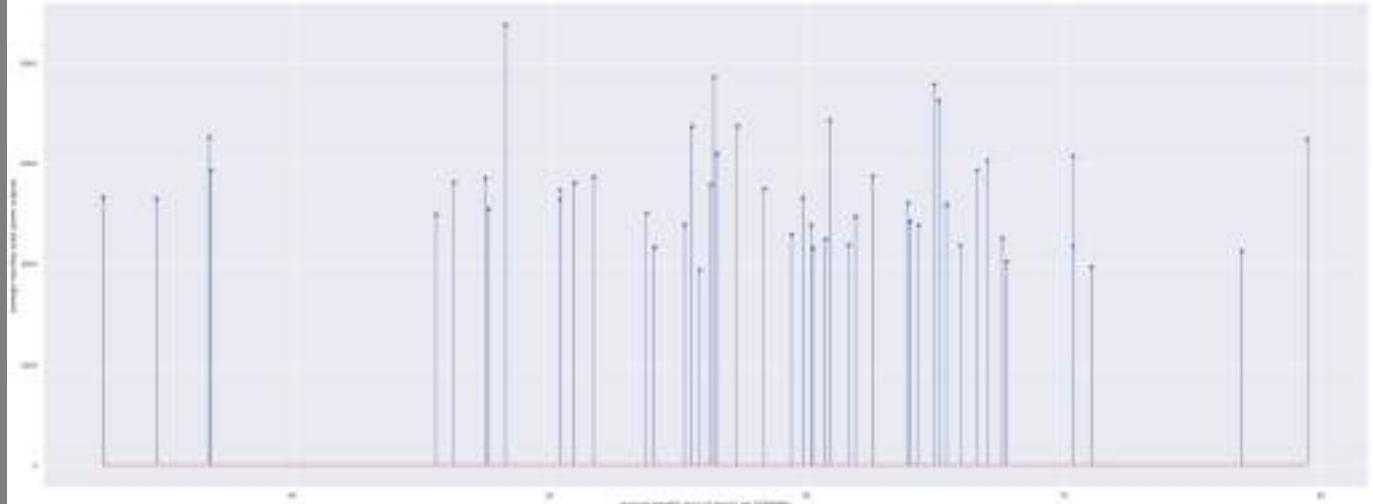
```
In [ ]: plt.suptitle('Distribution of average monthly wind power outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly wind power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

# Stem Plot
```

```
plt.stem(Rounded_Y, wind)
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.

In []: <StemContainer object of 3 artists>



This plot depicts the frequencies between the distribution of the monthly average output of wind power produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of wind power produced was when the price of energy per EUR/MWH was at roughly 70.36 euros/MWH,which was slightly above and below 5500 units. When the price of energy per EUR/MWH was at its lowest(at approximently 32.62 euros per MWH), wind power output was roughly 5250 units.The lowest amount produced, whcih was slightly below 4000 units, was at the price of approximently 55.82 EUR/MWH. At approximately 48.28 EUR/MWH, wind power produced the highest output amount,which was slightly below 9000 units. When the monthly average price of energy per EUR/MWH was at its highest, at approximently 79.13 EUR/MWH, the output was at roughly 6500 units. Each blue dot at the end of the blue lines represent an observation.

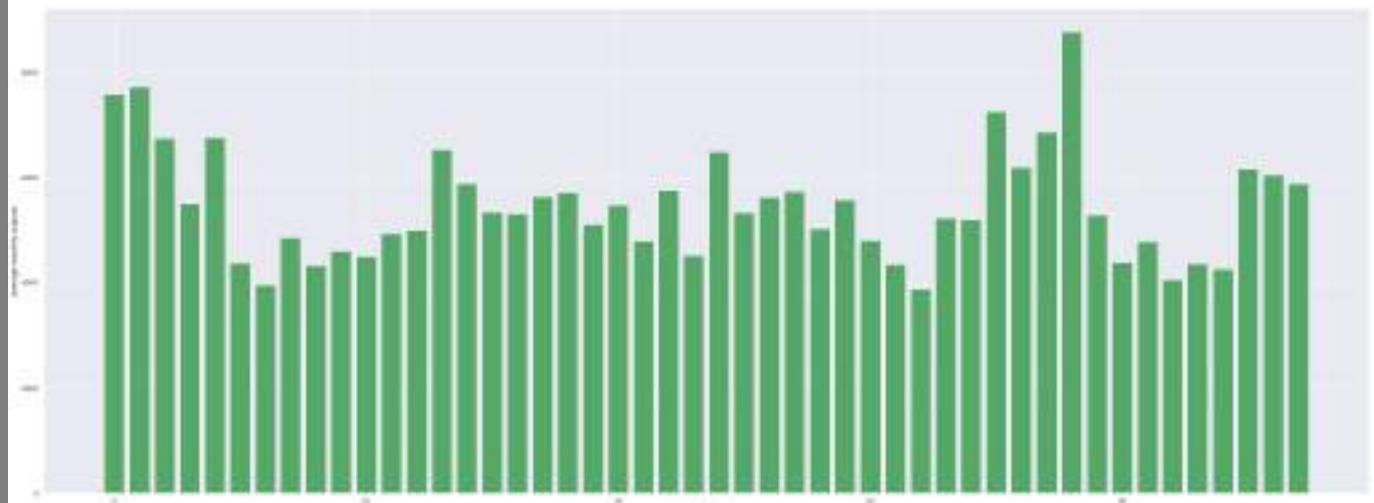
```
In [ ]: #Outputs in MWH Histogram
wind_Dict = {key: i for i, key in enumerate(wind)}

def Hist_wind(wind_Dict):
    for k, v in wind_Dict.items(): print(f"{v}:{k}")
print(wind_Dict)

plt.bar(list(wind_Dict.values()), wind_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs of wind power")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

plt.show()
plt.show()
```

```
{7587.697135061391: 0, 7731.806547619048: 1, 6747.878869448183: 2, 5506.381615598886: 3, 6757.408602150537: 4, 4375.495132127956: 5, 3955.2540322580644: 6, 4856.173387096775: 7, 4323.8: 8, 4597.236559139785: 9, 4493.763888888889: 10, 4943.1413189712: 11, 4993.399193548387: 12, 6534.337643678161: 13, 5884.690444145357: 14, 5343.19166666667: 15, 5309.384408602151: 16, 5638.5375: 17, 5715.833109017497: 18, 5103.193548387097: 19, 5484.695833333333: 20, 4788.928859060403: 21, 5766.394444444444: 22, 4519.7661290322585: 23, 6483.298387096775: 24, 5330.943452380952: 25, 5623.149394347241: 26, 5741.669444444445: 27, 5031.903225806452: 28, 5586.527777777777: 29, 4799.477150537635: 30, 4348.514784946236: 31, 3883.879166666666: 32, 5234.8: 33, 5200.18888888889: 34, 7257.2553763440865: 35, 6197.444892473119: 36, 6864.555059523809: 37, 8771.240915208613: 38, 5289.24722222222: 39, 4389.743279569892: 40, 4779.433333333333: 41, 4059.907133243607: 42, 4358.579301075269: 43, 4265.98888888889: 44, 6156.939597315436: 45, 6057.558333333333: 46, 5886.720430107527: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This unimodal histogram is roughly uniform and symmetrical throughout. It would be fair to assume that there was a singular external factor that increased the average monthly outputs, as indicated in the unimodal.

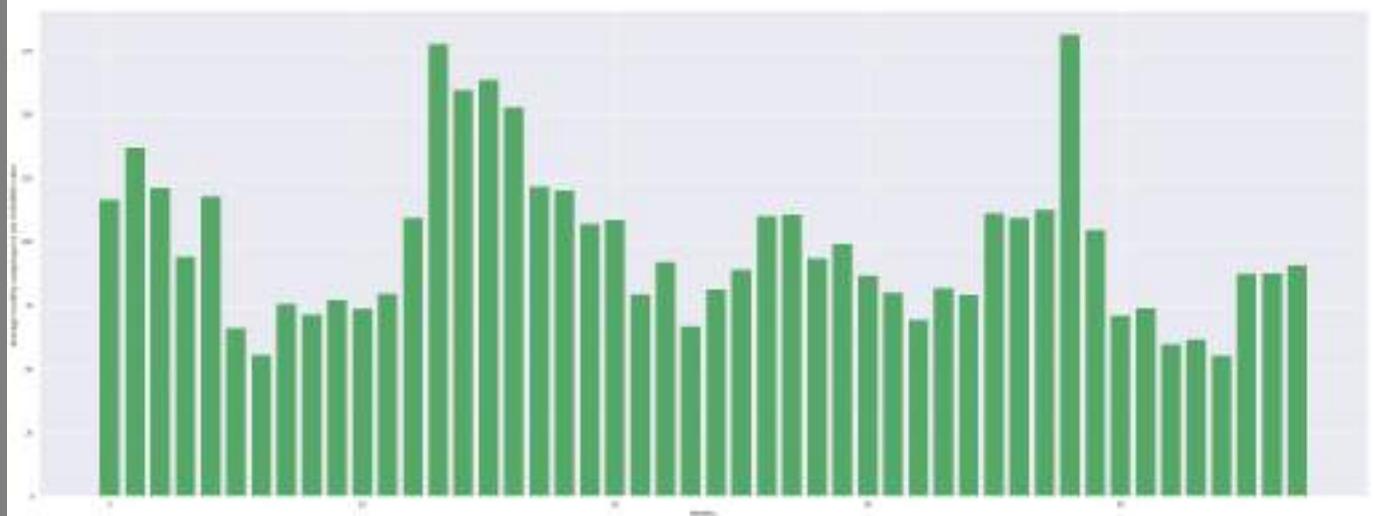
```
In [ ]: pdToListWind_Dict = {key: i for i, key in enumerate(pdToListWind)}

def Hist_pdToListWind(pdToListWind_Dict):
    for k, v in pdToListWind_Dict.items(): print(f"{v}:{k}")
#Output/price per EUR/MWH Histogram
print(pdToListWind_Dict)

plt.bar(list(pdToListWind_Dict.values()), pdToListWind_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price per EUR/MWH ratio of wind power")
plt.ylabel('Average monthly outputs/price per EUR/MWH ratio ')
plt.xlabel('Months')

plt.show()
plt.show()

{116.82543128813965: 0, 137.12802471367735: 1, 121.53421347537002: 2, 94.36154765974193: 3, 117.9425703747738: 4, 66.32059992366896: 5, 55.65133440246909: 6, 75.88000393157053: 7, 71.75860840942802: 8, 77.3857413604204: 9, 73.9996921549133: 10, 79.854615747137: 11, 109.5554866815863: 12, 177.79502686727042: 13, 159.83185271997365: 14, 163.80778899607589: 15, 153.04625503382746: 16, 121.87132168078833: 17, 120.32822130099225: 18, 107.204681255495: 19, 108.81124588511582: 20, 79.57353826430645: 21, 92.14281212187926: 22, 66.86525840323019: 23, 81.55904481: 24, 89.08990077342021: 25, 110.34460901350124: 26, 111.01896237334473: 27, 93.57742165525237: 28, 99.30153419549536: 29, 86.86430741326733: 30, 80.40249272218763: 31, 69.5829244210698: 32, 81.88934481821691: 33, 79.47634125782449: 34, 111.39083873243722: 35, 109.66604518834741: 36, 112.76087824292748: 37, 181.6754818474552: 38, 104.94384864859504: 39, 71.22302832897762: 40, 74.27461642031433: 41, 59.89526470950425: 42, 61.94339145093432: 43, 55.46437030805133: 44, 87.50349342853899: 45, 87.80566041896803: 46, 90.92222820062902: 47}
{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
```



The green bars represent the observation value for each respective month. This histogram has a significant trench between month 0 and month 10, meaning that there was a drop in the output produced per EUR/MWH. In addition, the histogram has a multimodal distribution, which could indicate that there wasn't any external factors that led to a singular price increase. However, the price fluctuations within the histogram appear to be seasonal.

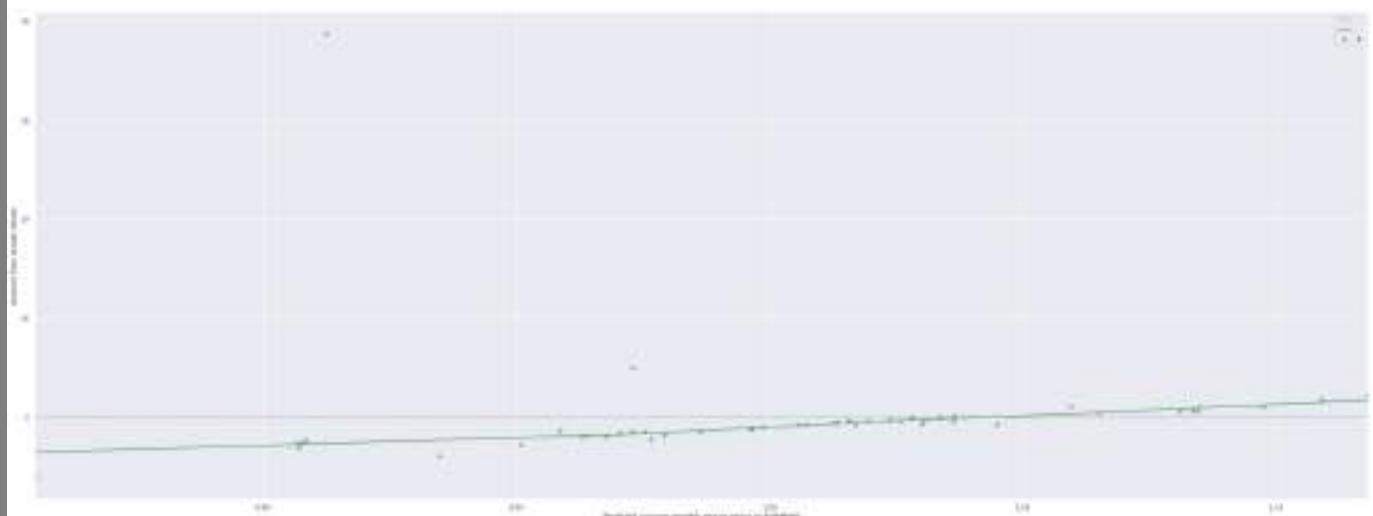
In []:

```
#Predicted OLS linear average monthly ratios versus residuals
WindRegRatioPredict = predictionsWind/predictions

sns.residplot(x = WindRegRatioPredict, y = standardized_residualsWind/standardized_residualsPricereg, lowess = True)

plt.suptitle("Predicted average monthly linear wind energy prices to EUR/MWH versus respective linear model residuals")
plt.xlabel("Predicted average monthly energy prices to EUR/MWH")
plt.ylabel("Amount from actual values")
plt.legend(..#..)
```

Out[]: <matplotlib.legend.Legend at 0x7ff603d1fa50>



With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: WindQuadRatioPredict = Wind_ypred/ypred
```

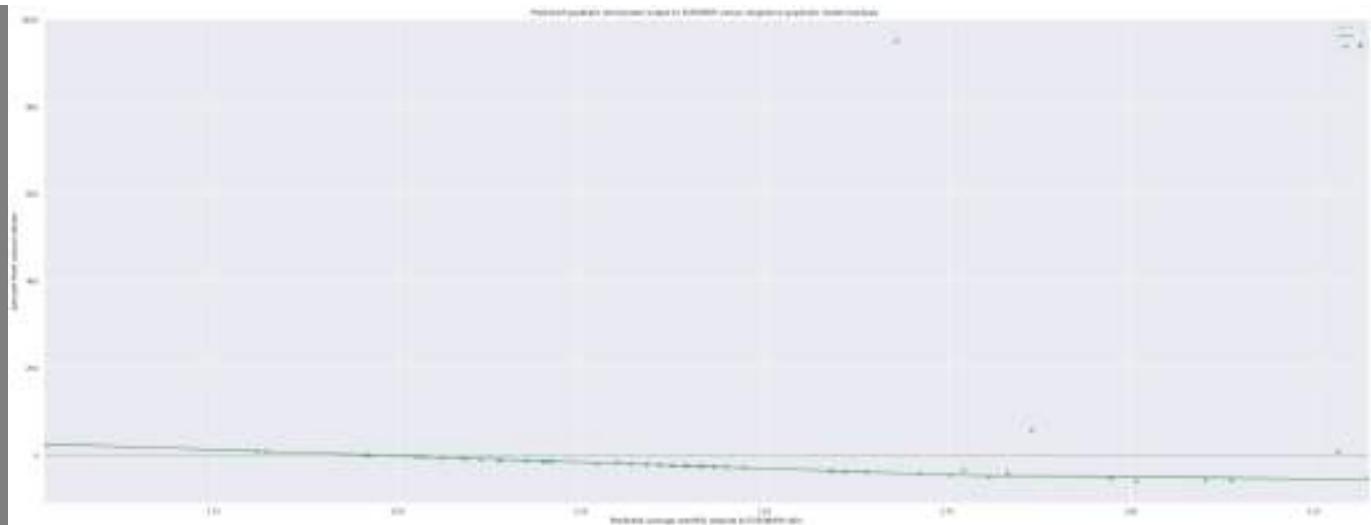
```
In [ ]: WindQuadRatioPredict = Wind_ypred/ypred

sns.residplot(x = WindQuadRatioPredict, y = standardized_residualsWindQuad/standardized_residualsPriceQuad , low=0, high=1)

plt.title("Predicted quadratic wind power output to EUR/MWH versus respective quadratic model residuals ")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#..)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff603af40d0>
```



With the exception of few outliers, the predictions seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

The results from the regression will be analyzed for seasonality since the output and the respective average monthly price of energy per EUR/MWH are taken into account simultaneously. The ratios are the outputs divided by the respective average monthly price of energy per EUR/MWH. This is the logarithmic model for the predicted average monthly prices of energy per EUR/MWH: given all other variables constant.

```
In [ ]: Rounded_Logpred = [round(item, 2) for item in Logpred]
print(Rounded_Logpred)
#Rounded Price
```

```
[27.3, 23.92, 23.58, 24.7, 24.28, 27.7, 29.71, 26.92, 25.45, 25.11, 25.63, 26.09, 19.65, 16.17, 16.2, 14.54, 15.36, 19.93, 20.41, 20.45, 21.56, 25.42, 26.36, 28.34, 33.04, 25.28, 21.78, 22.08, 22.89, 23.87, 23.47, 23.01, 23.69, 26.89, 27.49, 27.38, 23.97, 25.69, 20.72, 21.56, 25.99, 27.06, 28.42, 29.43, 32.02, 29.43, 27.96, 28.12]
```

```
In [ ]: print(list(Wind_Logpred))
```

```
print(list(Logpred))
```

```
[54.18258700970691, 53.932355679374034, 55.640845968039415, 57.796579614590954, 55.62429855694735, 59.7602489141322, 60.489954820322886, 58.92560003819699, 59.850012250575865, 59.375217490533274, 59.554887253045266, 58.774589074343325, 58.68732138939153, 56.011638576634404, 57.139684830525546, 58.079942354702915, 58.13864521810544, 57.567104369510915, 57.43288841005303, 58.49667466488741, 57.83423476874993, 59.04236331832913, 57.34509379620552, 59.5097370090242, 56.10026305193335, 58.101210132431795, 57.59382424908334, 57.38802624268958, 58.62046305554563, 57.65741395964059, 59.02404728357303, 59.807097541493185, 60.61389001131619, 58.26815345393382, 58.32825212635168, 54.756365497966875, 56.59661855689635, 55.43824963694193, 52.12748367289367, 58.17361139309415, 59.735508455990725, 59.05885133248951, 60.30823535996081, 59.78962153340026, 59.95039536216177, 56.66695188011892, 56.83951733198664, 57.136159966473606] [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]
```

```
In [ ]: dfWind_Log = ({"Price_Log": [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.631280368545422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.357844118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848, 26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.076426999163463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.893389962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.01868170373189, 29.433366230197276, 27.958095077371567, 28.123467161134005]}

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(dfWind_Log["Wind_Log"])

test_result=adfuller(dfWind_Log["Wind_Log"])
```

```

{'Price_Log': [27.297348375081718, 23.917580710720195, 23.57768040860245, 24.695022488031967, 24.276742672176834
, 27.70215663958084, 29.71346061831328, 26.922106949120572, 25.44503172036812, 25.110405022200517, 25.6312803685
45422, 26.094916817531914, 19.653934415930614, 16.170990077171673, 16.197003623963596, 14.5399662074255, 15.3578
44118752933, 19.925256208811728, 20.412855439874555, 20.452442187812732, 21.55859284131481, 25.416478012537848,
26.362962874201227, 28.341491281477328, 33.03596300746373, 25.28048812361594, 21.777314693468632, 22.07642699916
3463, 22.887202207806165, 23.868007024659466, 23.471186295255887, 23.01020010054923, 23.693727745821242, 26.8933
89962364463, 27.48739853201623, 27.37715831385097, 23.968136817101986, 25.690590519617974, 20.71973214185461, 21
.556674673151495, 25.98916652718635, 27.060244403968117, 28.415795989223025, 29.434035669877737, 32.018681703731
89, 29.433366230197276, 27.958095077371567, 28.123467161134005], 'Wind_Log': [54.18258700970691, 53.932355679374
034, 55.640845968039415, 57.796579614590954, 55.62429855694735, 59.7602489141322, 60.489954820322886, 58.9256000
3819699, 59.850012250575865, 59.375217490533274, 59.554887253045266, 58.774589074343325, 58.68732138939153, 56.0
11638576634404, 57.139684830525546, 58.079942354702915, 58.13864521810544, 57.567104369510915, 57.43288841005303
, 58.49667466488741, 57.83423476874993, 59.04236331832913, 57.34509379620552, 59.5097370090242, 56.1002630519333
5, 58.101210132431795, 57.59382424908334, 57.38802624268958, 58.62046305554563, 57.65741395964059, 59.0240472835
7303, 59.807097541493185, 60.61389001131619, 58.26815345393382, 58.32825212635168, 54.756365497966875, 56.596618
55689635, 55.43824963694193, 52.12748367289367, 58.17361139309415, 59.735508455990725, 59.05885133248951, 60.308
23535996081, 59.78962153340026, 59.95039536216177, 56.66695188011892, 56.83951733198664, 57.136159966473606]}

Price_Log      Wind_Log
0    27.297348  54.182587
1    23.917581  53.932356
2    23.577680  55.640846
3    24.695022  57.796580
4    24.276743  55.624299
5    27.702157  59.760249
6    29.713461  60.489955
7    26.922107  58.925600
8    25.445032  59.850012
9    25.110405  59.375217
10   25.631280  59.554887
11   26.094917  58.774589
12   19.653934  58.687321
13   16.170990  56.011639
14   16.197004  57.139685
15   14.539966  58.079942
16   15.357844  58.138645
17   19.925256  57.567104
18   20.412855  57.432888
19   20.452442  58.496675
20   21.558593  57.834235
21   25.416478  59.042363
22   26.362963  57.345094
23   28.341491  59.509737
24   33.035963  56.100263
25   25.280488  58.101210
26   21.777315  57.593824
27   22.076427  57.388026
28   22.887202  58.620463
29   23.868007  57.657414
30   23.471186  59.024047
31   23.010200  59.807098
32   23.693728  60.613890
33   26.893390  58.268153
34   27.487399  58.328252
35   27.377158  54.756365
36   23.968137  56.596619
37   25.690591  55.438250
38   20.719732  52.127484
39   21.556675  58.173611
40   25.989167  59.735508
41   27.060244  59.058851
42   28.415796  60.308235
43   29.434036  59.789622
44   32.018682  59.950395
45   29.433366  56.666952
46   27.958095  56.839517
47   28.123467  57.136160

ADF Test Statistic : -5.058912406033443
p-value : 1.6946137371175267e-05
#Lags Used : 5
Number of Observations : 42
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

```

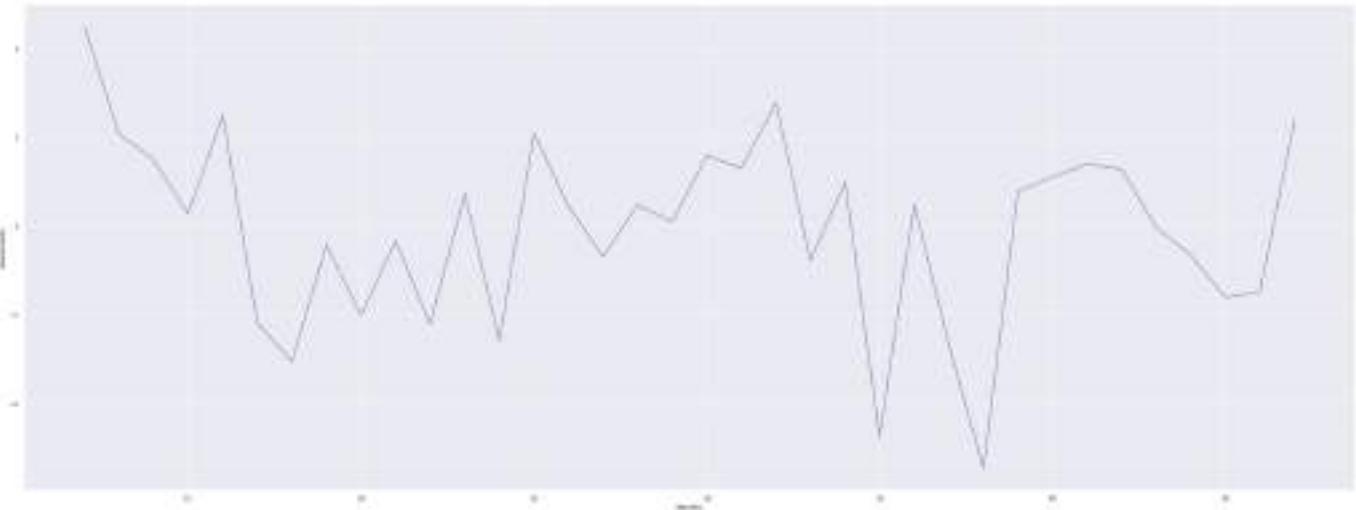
In [ ]: df_Wind_Log['First Difference'] = df_Wind_Log["Wind_Log"]- df_Wind_Log["Wind_Log"].shift(1) # Seasonality value
df_Wind_Log['Seasonal Difference']=df_Wind_Log["Wind_Log"]- df_Wind_Log["Wind_Log"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted logarithmic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Wind_Log.head() #Predictive Logarithmic Seasonality Plot
df_Wind_Log['Seasonal Difference'].plot()
# Seasonality Plot

```

```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff603af44d0>

```



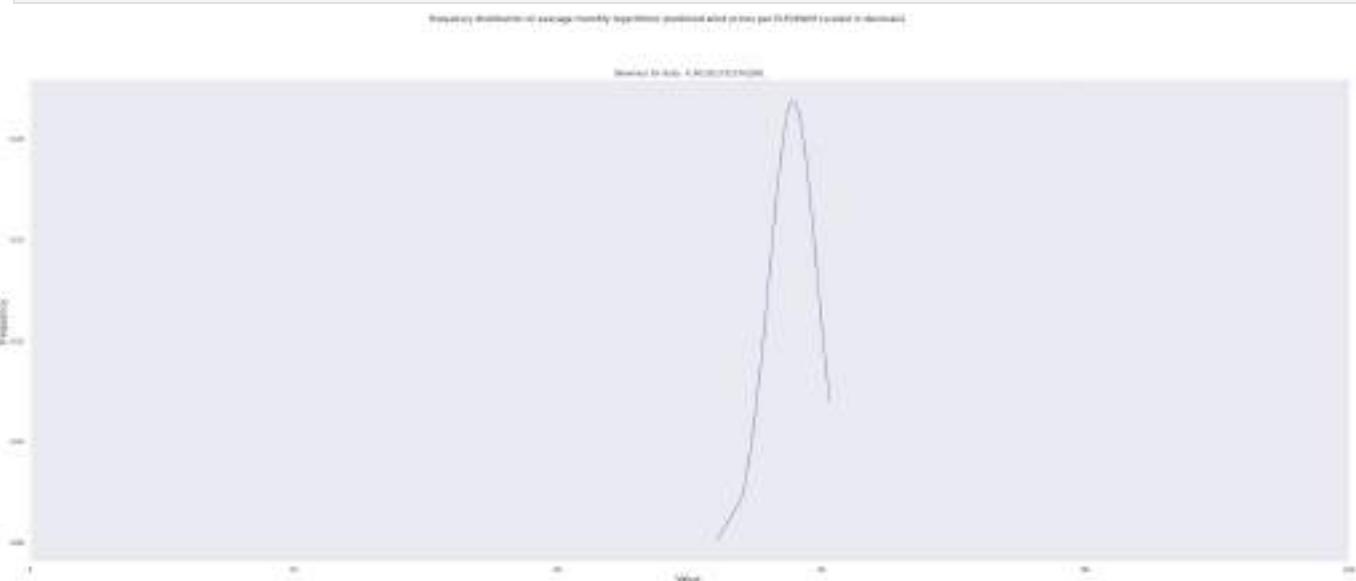
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Wind_LogResults_mean = np.mean(df_Wind_Log["Wind_Log"])
Wind_LogResults_std = np.std(df_Wind_Log["Wind_Log"])

Wind_LogResultspdf = stats.norm.pdf(df_Wind_Log["Wind_Log"].sort_values(), Wind_LogResults_mean, Wind_LogResults_std)

plt.plot(df_Wind_Log["Wind_Log"].sort_values(), Wind_LogResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Wind_Log["Wind_Log"])}')
plt.suptitle("Frequency distribution of average monthly logarithmic predicted wind prices per EUR/MWH (scaled in %)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
Wind_Log_Dict = {key: i for i, key in enumerate(df_Wind_Log["Wind_Log"])}

def Hist_Wind_Log(Wind_Log_Dict):
    for k, v in Wind_Log_Dict.items(): print(f"{v}:{k}")
```

```

print(Wind_Log_Dict)

plt.bar(list(Wind_Log_Dict.values()), Wind_Log_Dict.keys(), color='g') #Predictive Logarithmic Histogram
plt.title("Average monthly output of predicted logarithmic wind prices per EUR/MWh")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}

{54.18258700970691: 0, 53.932355679374034: 1, 55.640845968039415: 2, 57.796579614590954: 3, 55.62429855694735: 4, 59.7602489141322: 5, 60.489954820322886: 6, 58.92560003819699: 7, 59.850012250575865: 8, 59.375217490533274: 9, 59.554887253045266: 10, 58.774589074343325: 11, 58.68732138939153: 12, 56.011638576634404: 13, 57.139684830525546: 14, 58.079942354702915: 15, 58.13864521810544: 16, 57.567104369510915: 17, 57.43288841005303: 18, 58.49667466488741: 19, 57.83423476874993: 20, 59.04236331832913: 21, 57.34509379620552: 22, 59.5097370090242: 23, 56.10026305193335: 24, 58.101210132431795: 25, 57.59382424908334: 26, 57.38802624268958: 27, 58.62046305554563: 28, 57.65741395964059: 29, 59.02404728357303: 30, 59.807097541493185: 31, 60.61389001131619: 32, 58.26815345393382: 33, 58.32825212635168: 34, 54.756365497966875: 35, 56.59661855689635: 36, 55.43824963694193: 37, 52.12748367289367: 38, 58.17361139309415: 39, 59.735508455990725: 40, 59.05885133248951: 41, 60.30823535996081: 42, 59.78962153340026: 43, 59.95039536216177: 44, 56.66695188011892: 45, 56.83951733198664: 46, 57.136159966473606: 47}



The green bars represent the observation value for each respective month. This histogram is uniform.

In []: df_Wind_Log.describe(include = 'all') # Description Tables

	Price_Log	Wind_Log	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859848	0.062842	-0.057944
std	4.072442	1.842626	1.876077	2.129910
min	14.539966	52.127484	-3.571887	-5.466341
25%	22.001649	57.061999	-0.728478	-1.516851
50%	25.195447	58.119928	0.060099	0.368751
75%	27.317301	59.137943	1.168087	1.333177
max	33.035963	60.613890	6.046128	4.504734

In []: print(predictionsWind/predictions)

```

Rounded_predictions = [round(item, 2) for item in predictions]
print(Rounded_predictions)
#Rounded Price

```

```
[1.02576547 1.01690077 1.04489068 1.08102131 1.0362359 1.10885658
1.11794905 1.08473924 1.09742521 1.08445608 1.08349493 1.06514421
1.05944638 1.00724557 1.02358467 1.03644763 1.03354099 1.019495
1.01327086 1.02814997 1.01269075 1.02997874 0.99664284 1.03042432
0.96779581 0.99862108 0.98626603 0.97914703 0.99652945 0.9765986
0.99612925 1.00570564 1.01561105 0.97281267 0.97034281 0.90768391
0.9348669 0.91250139 0.85498977 0.95081426 0.97293329 0.95856513
0.97544916 0.96371903 0.96298269 0.90711684 0.90676725 0.9083927 ]
[52.82, 53.04, 53.25, 53.46, 53.68, 53.89, 54.11, 54.32, 54.54, 54.75, 54.97, 55.18, 55.39, 55.61, 55.82, 56.04,
56.25, 56.47, 56.68, 56.9, 57.11, 57.32, 57.54, 57.75, 57.97, 58.18, 58.4, 58.61, 58.82, 59.04, 59.25, 59.47, 59
.68, 59.9, 60.11, 60.33, 60.54, 60.75, 60.97, 61.18, 61.4, 61.61, 61.83, 62.04, 62.25, 62.47, 62.68, 62.9]
```

This is the linear seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(predictionsWind))
print(list(predictions))
```

```
[54.18258700970691, 53.932355679374034, 55.640845968039415, 57.796579614590954, 55.62429855694735, 59.7602489141
322, 60.489954820322886, 58.92560003819699, 59.850012250575865, 59.375217490533274, 59.554887253045266, 58.77458
9074343325, 58.68732138939153, 56.011638576634404, 57.139684830525546, 58.079942354702915, 58.13864521810544, 57
.567104369510915, 57.43288841005303, 58.49667466488741, 57.83423476874993, 59.04236331832913, 57.34509379620552,
59.5097370090242, 56.10026305193335, 58.101210132431795, 57.59382424908334, 57.38802624268958, 58.62046305554563
, 57.65741395964059, 59.02404728357303, 59.807097541493185, 60.61389001131619, 58.26815345393382, 58.32825212635
168, 54.756365497966875, 56.59661855689635, 55.43824963694193, 52.12748367289367, 58.17361139309415, 59.73550845
5990725, 59.05885133248951, 60.30823535996081, 59.78962153340026, 59.95039536216177, 56.66695188011892, 56.83951
733198664, 57.136159966473606]
[52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.8935780768445
4, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.1799359739
78026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.466293
87111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.75
2651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.
03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639,
60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988
, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.683690373
92337, 62.89808335677895]
```

```
In [ ]: dfWindReg = {"Price": [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.679185093
"WindReg" : [54.18258700970691, 53.932355679374034, 55.640845968039415, 57.796579614590954, 55.62429855694735, 59.7602489141
print(dfWindReg) #Dataframes
df_WindReg= pd.DataFrame.from_dict(dfWindReg, orient = "columns")
print(df_WindReg)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_WindReg["WindReg"])

test_result=adfuller(df_WindReg["WindReg"])

df_WindReg['First Difference'] = df_WindReg["WindReg"]- df_WindReg["WindReg"].shift(1) # Seasonality values
df_WindReg['Seasonal Difference']=df_WindReg["WindReg"]- df_WindReg["WindReg"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted linear prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_WindReg.head() #Predictive Linear Seasonality Plot
df_WindReg['Seasonal Difference'].plot()
# Seasonality Plot
```

```
{'Price': [52.821613162566635, 53.03600614542222, 53.2503991282778, 53.46479211113338, 53.67918509398896, 53.89357807684454, 54.10797105970013, 54.322364042555705, 54.53675702541128, 54.75115000826687, 54.96554299112245, 55.179935973978026, 55.39432895683361, 55.60872193968919, 55.823114922544775, 56.03750790540035, 56.25190088825593, 56.46629387111152, 56.680686853967096, 56.895079836822674, 57.10947281967826, 57.32386580253384, 57.53825878538942, 57.752651768245, 57.96704475110058, 58.181437733956166, 58.395830716811744, 58.61022369966732, 58.82461668252291, 59.03900966537849, 59.25340264823407, 59.46779563108965, 59.68218861394523, 59.896581596800814, 60.11097457965639, 60.32536756251197, 60.53976054536756, 60.754153528223135, 60.96854651107871, 61.1829394939343, 61.39733247678988, 61.611725459645456, 61.82611844250104, 62.04051142535662, 62.254904408212205, 62.469297391067784, 62.68369037392337, 62.89808335677895], 'WindReg': [54.18258700970691, 53.932355679374034, 55.640845968039415, 57.796579614590954, 55.62429855694735, 59.7602489141322, 60.489954820322886, 58.92560003819699, 59.850012250575865, 59.375217490533274, 59.554887253045266, 58.774589074343325, 58.68732138939153, 56.011638576634404, 57.139684830525546, 58.079942354702915, 58.13864521810544, 57.567104369510915, 57.43288841005303, 58.49667466488741, 57.83423476874993, 59.04236331832913, 57.34509379620552, 59.5097370090242, 56.10026305193335, 58.101210132431795, 57.59382424908334, 57.38802624268958, 58.62046305554563, 57.65741395964059, 59.02404728357303, 59.807097541493185, 60.61389001131619, 58.26815345393382, 58.32825212635168, 54.756365497966875, 56.59661855689635, 55.43824963694193, 52.12748367289367, 58.17361139309415, 59.735508455990725, 59.05885133248951, 60.30823535996081, 59.78962153340026, 59.95039536216177, 56.66695188011892, 56.83951733198664, 57.136159966473606]}
```

	Price	WindReg
0	52.821613	54.182587
1	53.036006	53.932356
2	53.250399	55.640846
3	53.464792	57.796580
4	53.679185	55.624299
5	53.893578	59.760249
6	54.107971	60.489955
7	54.322364	58.925600
8	54.536757	59.850012
9	54.751150	59.375217
10	54.965543	59.554887
11	55.179936	58.774589
12	55.394329	58.687321
13	55.608722	56.011639
14	55.823115	57.139685
15	56.037508	58.079942
16	56.251901	58.138645
17	56.466294	57.567104
18	56.680687	57.432888
19	56.895080	58.496675
20	57.109473	57.834235
21	57.323866	59.042363
22	57.538259	57.345094
23	57.752652	59.509737
24	57.967045	56.100263
25	58.181438	58.101210
26	58.395831	57.593824
27	58.610224	57.388026
28	58.824617	58.620463
29	59.039010	57.657414
30	59.253403	59.024047
31	59.467796	59.807098
32	59.682189	60.613890
33	59.896582	58.268153
34	60.110975	58.328252
35	60.325368	54.756365
36	60.539761	56.596619
37	60.754154	55.438250
38	60.968547	52.127484
39	61.182939	58.173611
40	61.397332	59.735508
41	61.611725	59.058851
42	61.826118	60.308235
43	62.040511	59.789622
44	62.254904	59.950395
45	62.469297	56.666952
46	62.683690	56.839517
47	62.898083	57.136160

ADF Test Statistic : -5.058912406033443

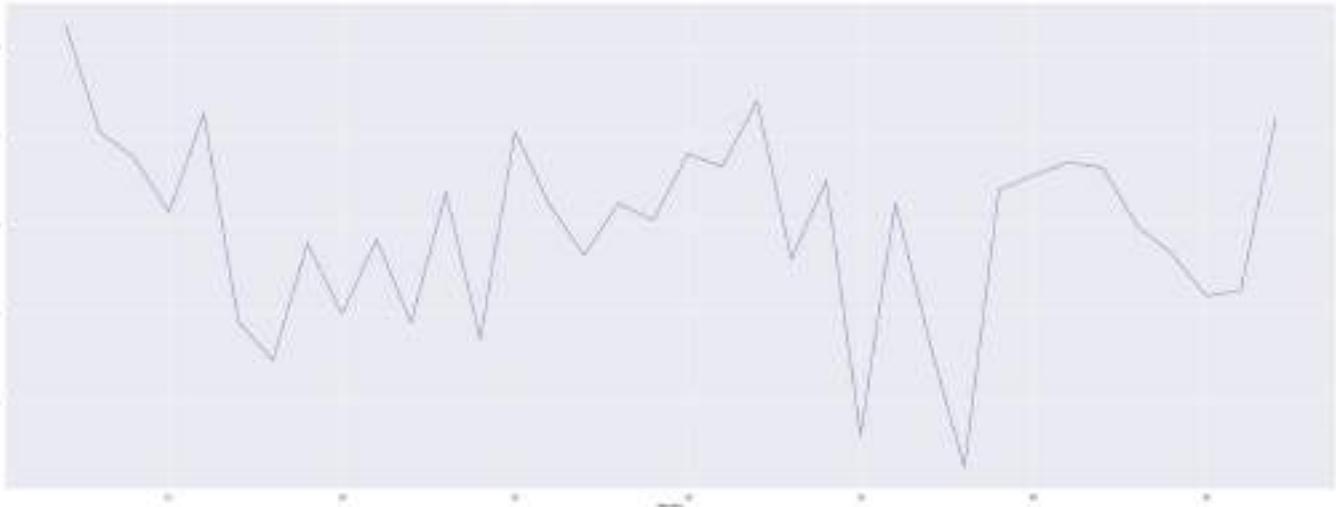
p-value : 1.6946137371175267e-05

#Lags Used : 5

Number of Observations : 42

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6039e6f10>



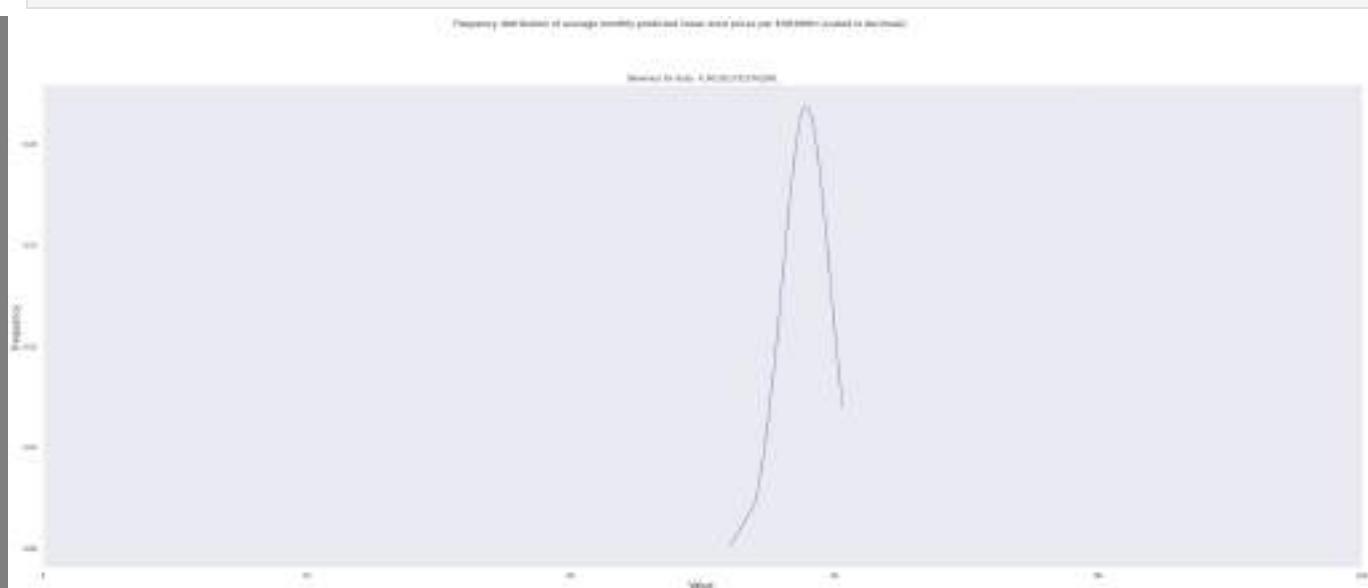
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
WindRegResults_mean = np.mean(df_WindReg["WindReg"])
WindRegResults_std = np.std(df_WindReg["WindReg"])

WindRegResultspdf = stats.norm.pdf(df_WindReg["WindReg"].sort_values(), WindRegResults_mean, WindRegResults_std)

plt.plot(df_WindReg["WindReg"].sort_values(), WindRegResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_WindReg["WindReg"])}')
plt.suptitle("Frequency distribution of average monthly predicted linear wind prices per EUR/MWH (scaled in deciles)", size=15)
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
WindReg_Dict = {key: i for i, key in enumerate(df_WindReg["WindReg"])}
def Hist_WindReg(WindReg_Dict):
    for k, v in WindReg_Dict.items(): print(f"{v}:{k}")
print(WindReg_Dict)
```

```

plt.bar(list(WindReg_Dict.values()), WindReg_Dict.keys(), color='g') #Predictive Linear Histogram
plt.title("Average monthly predicted linear wind prices per EUR/MWH ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

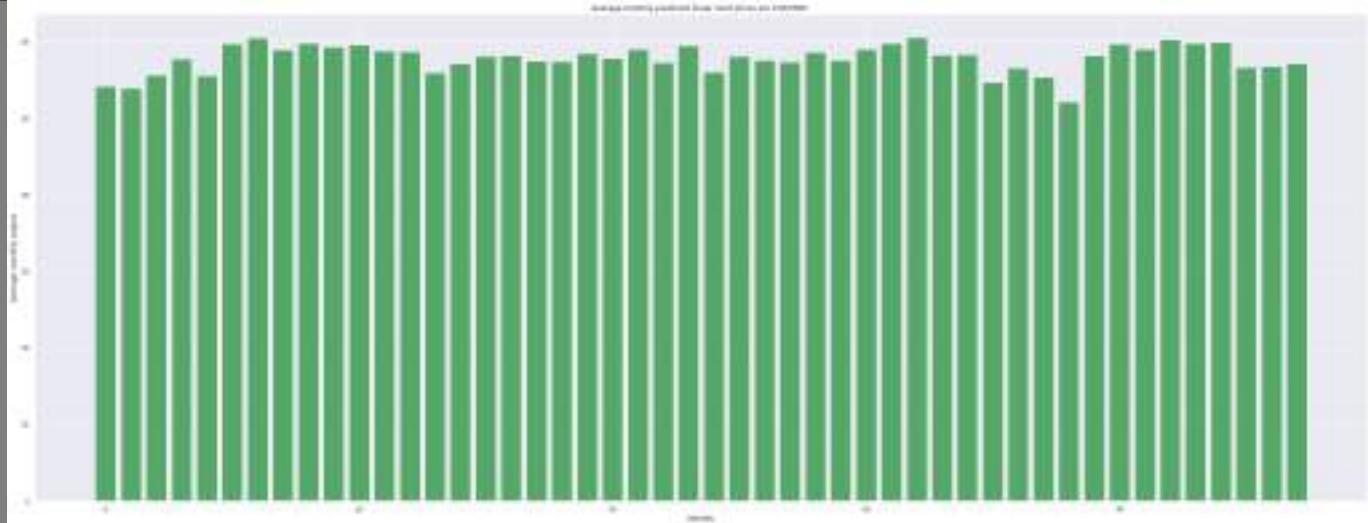
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{54.18258700970691: 0, 53.932355679374034: 1, 55.640845968039415: 2, 57.796579614590954: 3, 55.62429855694735: 4, 59.7602489141322: 5, 60.489954820322886: 6, 58.92560003819699: 7, 59.850012250575865: 8, 59.375217490533274: 9, 59.554887253045266: 10, 58.774589074343325: 11, 58.68732138939153: 12, 56.011638576634404: 13, 57.139684830525546: 14, 58.079942354702915: 15, 58.13864521810544: 16, 57.567104369510915: 17, 57.43288841005303: 18, 58.49667466488741: 19, 57.83423476874993: 20, 59.04236331832913: 21, 57.34509379620552: 22, 59.5097370090242: 23, 56.10026305193335: 24, 58.101210132431795: 25, 57.59382424908334: 26, 57.38802624268958: 27, 58.62046305554563: 28, 57.65741395964059: 29, 59.02404728357303: 30, 59.807097541493185: 31, 60.61389001131619: 32, 58.26815345393382: 33, 58.32825212635168: 34, 54.756365497966875: 35, 56.59661855689635: 36, 55.43824963694193: 37, 52.12748367289367: 38, 58.17361139309415: 39, 59.735508455990725: 40, 59.05885133248951: 41, 60.30823535996081: 42, 59.78962153340026: 43, 59.95039536216177: 44, 56.66695188011892: 45, 56.83951733198664: 46, 57.136159966473606: 47}

```



The green bars represent the observation value for each respective month. This histogram is uniform throughout.

```
In [ ]: df_WindReg.describe(include = 'all') # Description Tables
```

	Price	WindReg	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	57.859848	57.859848	0.062842	-0.057944
std	3.001502	1.842626	1.876077	2.129910
min	52.821613	52.127484	-3.571887	-5.466341
25%	55.340731	57.061999	-0.728478	-1.516851
50%	57.859848	58.119928	0.060099	0.368751
75%	60.378966	59.137943	1.168087	1.333177
max	62.898083	60.613890	6.046128	4.504734

Below is the quadratic seasonality model for the predicted average monthly prices of energy per EUR/MWH for this respective resource; given all other variables constant.

```
In [ ]: print(list(Wind_ypred))
print(list(ypred))
```

```
[56.04329713150446, 56.46171035804166, 54.72189377009627, 56.261264579432776, 54.726192786495695, 61.29002980953234, 64.03985534772364, 58.730105505938454, 61.60254103985825, 60.03150036395134, 60.60223441813166, 58.333675442530314, 58.1139050062686, 54.68995118414506, 55.35077885782508, 56.77346636500779, 56.888579101574315, 55.8992291276222, 55.70936510818535, 57.657543801339365, 56.32518146274837, 59.05064677307334, 55.593907857632075, 60.45608846226824, 54.700570182295706, 56.81481424386922, 55.93895536133555, 55.64950412298969, 57.95015221298191, 56.03607304869448, 58.99955677101843, 61.45223204692398, 64.55432381072195, 57.15346466271086, 57.28149426349869, 55.29584525074563, 54.89021755594394, 54.79142682024407, 61.1425634697473, 56.95861400587563, 61.205165898052336, 59.096895038457546, 63.31041943922497, 61.391495391480305, 61.96058238634525, 54.93495932312234, 55.06352677964472, 55.34693688244426] [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356]
```

```
In [ ]: dfWind_Quad = ({ "Price_Quad": [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.119121355777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.868735417613472, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.456172637404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.665789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 21.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527835, 30.169485792181256, 28.070861800194585, 28.297337605159356] })
```

```
#Dataframes
print(dfWind_Quad)
df_Wind_Quad= pd.DataFrame.from_dict(dfWind_Quad, orient = "columns")
print(df_Wind_Quad)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis,indicating it is non-stationary ")

adfuller_test(df_Wind_Quad["Wind_Quad"])

test_result=adfuller(df_Wind_Quad["Wind_Quad"])

df_Wind_Quad['First Difference'] = df_Wind_Quad["Wind_Quad"]- df_Wind_Quad["Wind_Quad"].shift(1) # Seasonality
df_Wind_Quad['Seasonal Difference']=df_Wind_Quad["Wind_Quad"]- df_Wind_Quad["Wind_Quad"].shift(12) #
plt.suptitle("Seasonal Difference of average monthly predicted quadratic prices of energy per EUR/MWH")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_Wind_Quad.head() #Predictive Quadratic Seasonality Plot
df_Wind_Quad[['Seasonal Difference']].plot()
# Seasonality Plot
```

```

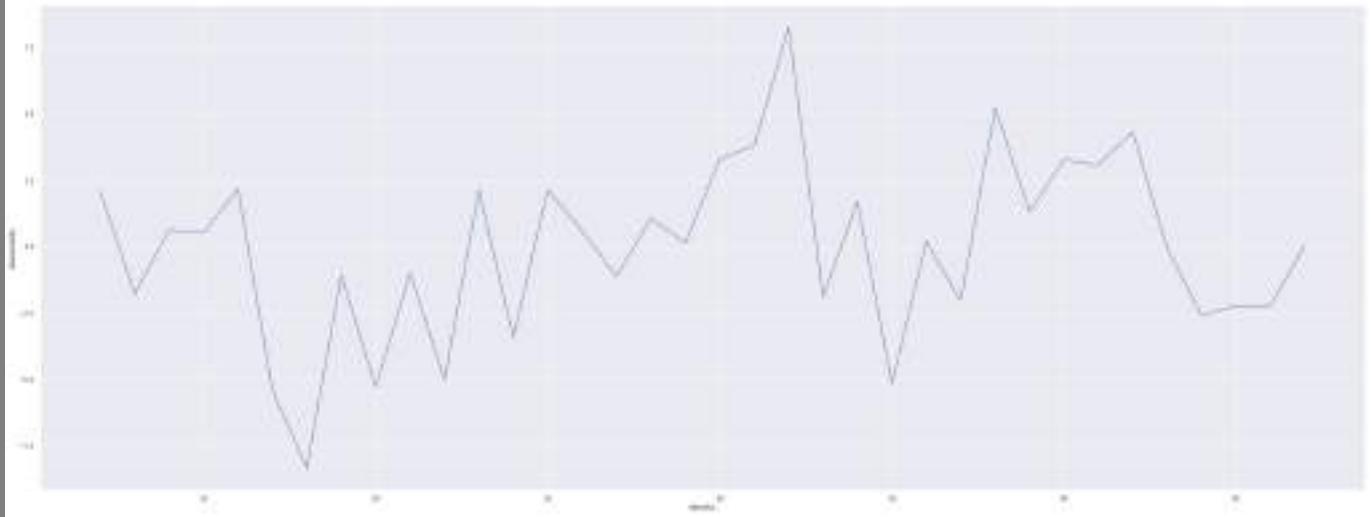
{'Price_Quad': [27.188077690295593, 23.22554817302482, 22.87823628204505, 24.055117412542895, 23.602708631972398
, 27.724722433871626, 30.587838932854936, 26.70248165923099, 24.901803909409455, 24.518414867253377, 25.11912135
5777786, 25.672299682598407, 19.546362569195352, 17.633406195018726, 17.644052319077353, 17.075344431983726, 17.
328254644736486, 19.736629098849825, 20.09354637020233, 20.123368510962656, 21.007991576184747, 24.8867354176134
72, 26.000055062885394, 28.599304786622582, 36.03506726766325, 24.71214950611809, 21.194644314739115, 21.4561726
37404084, 22.201511151656746, 23.17431072214061, 22.771345027936817, 22.31923276693945, 22.99576213803661, 26.66
5789045581803, 27.438369965065977, 27.292829521192132, 23.2780059994575, 25.18891499253064, 20.32804924075167, 2
1.006371788590553, 25.54459336089593, 26.879917631774212, 28.703097663581467, 30.17047810264582, 34.272279943527
835, 30.169485792181256, 28.070861800194585, 28.297337605159356], 'Wind_Quad': [56.04329713150446, 56.4617103580
4166, 54.72189377009627, 56.261264579432776, 54.726192786495695, 61.29002980953234, 64.03985534772364, 58.730105
505938454, 61.60254103985825, 60.03150036395134, 60.60223441813166, 58.333675442530314, 58.1139050062686, 54.689
95118414506, 55.35077885782508, 56.77346636500779, 56.888579101574315, 55.8992291276222, 55.70936510818535, 57.6
57543801339365, 56.32518146274837, 59.05064677307334, 55.593907857632075, 60.45608846226824, 54.700570182295706,
56.81481424386922, 55.93895536133555, 55.64950412298969, 57.95015221298191, 56.03607304869448, 58.99955677101843
, 61.45223204692398, 64.55432381072195, 57.15346466271086, 57.28149426349869, 55.29584525074563, 54.890217555943
94, 54.79142682024407, 61.1425634697473, 56.95861400587563, 61.205165898052336, 59.096895038457546, 63.310419439
22497, 61.391495391480305, 61.96058238634525, 54.93495932312234, 55.06352677964472, 55.34693688244426]}

    Price_Quad Wind_Quad
0      27.188078   56.043297
1      23.225548   56.461710
2      22.878236   54.721894
3      24.055117   56.261265
4      23.602709   54.726193
5      27.724722   61.290030
6      30.587839   64.039855
7      26.702482   58.730106
8      24.901804   61.602541
9      24.518415   60.031500
10     25.119121   60.602234
11     25.672300   58.333675
12     19.546363   58.113905
13     17.633406   54.689951
14     17.644052   55.350779
15     17.075344   56.773466
16     17.328255   56.888579
17     19.736629   55.899229
18     20.093546   55.709365
19     20.123369   57.657544
20     21.007992   56.325181
21     24.868735   59.050647
22     26.000055   55.593908
23     28.599305   60.456088
24     36.035067   54.700570
25     24.712150   56.814814
26     21.194644   55.938955
27     21.456173   55.649504
28     22.201511   57.950152
29     23.174311   56.036073
30     22.771345   58.999557
31     22.319233   61.452232
32     22.995762   64.554324
33     26.665789   57.153465
34     27.438370   57.281494
35     27.292830   55.295845
36     23.278006   54.890218
37     25.188915   54.791427
38     20.328049   61.142563
39     21.006372   56.958614
40     25.544593   61.205166
41     26.879918   59.096895
42     28.703098   63.310419
43     30.170478   61.391495
44     34.272280   61.960582
45     30.169486   54.934959
46     28.070862   55.063527
47     28.297338   55.346937

ADF Test Statistic : -3.9333127173042355
p-value : 0.0018029126536053127
#Lags Used : 2
Number of Observations : 45
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff60371f6d0>



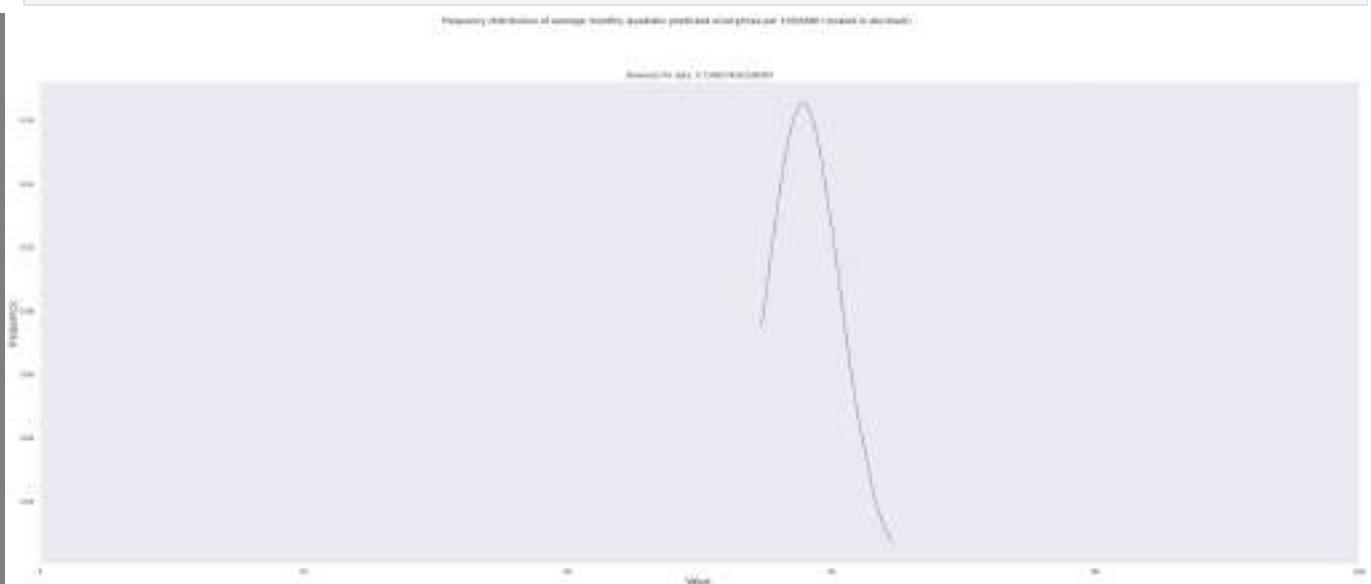
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly predicted prices of energy per EUR/MWH for this particular resource and the predicted average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
Wind_QuadResults_mean = np.mean(df_Wind_Quad["Wind_Quad"])
Wind_QuadResults_std = np.std(df_Wind_Quad["Wind_Quad"])

Wind_QuadResultspdf = stats.norm.pdf(df_Wind_Quad["Wind_Quad"].sort_values(), Wind_QuadResults_mean, Wind_QuadResults_std)

plt.plot(df_Wind_Quad["Wind_Quad"].sort_values(), Wind_QuadResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_Wind_Quad["Wind_Quad"])}')
plt.suptitle("Frequency distribution of average monthly quadratic predicted wind prices per EUR/MWH (scaled in eur/mwh) for the year 2018")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is skewed to the right. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
Wind_Quad_Dict = {key: i for i, key in enumerate(df_Wind_Quad["Wind_Quad"])}

def Hist_Wind_Quad(Wind_Quad_Dict):
    for k, v in Wind_Quad_Dict.items(): print(f"{v}:{k}")
print(Wind_Quad_Dict)
```

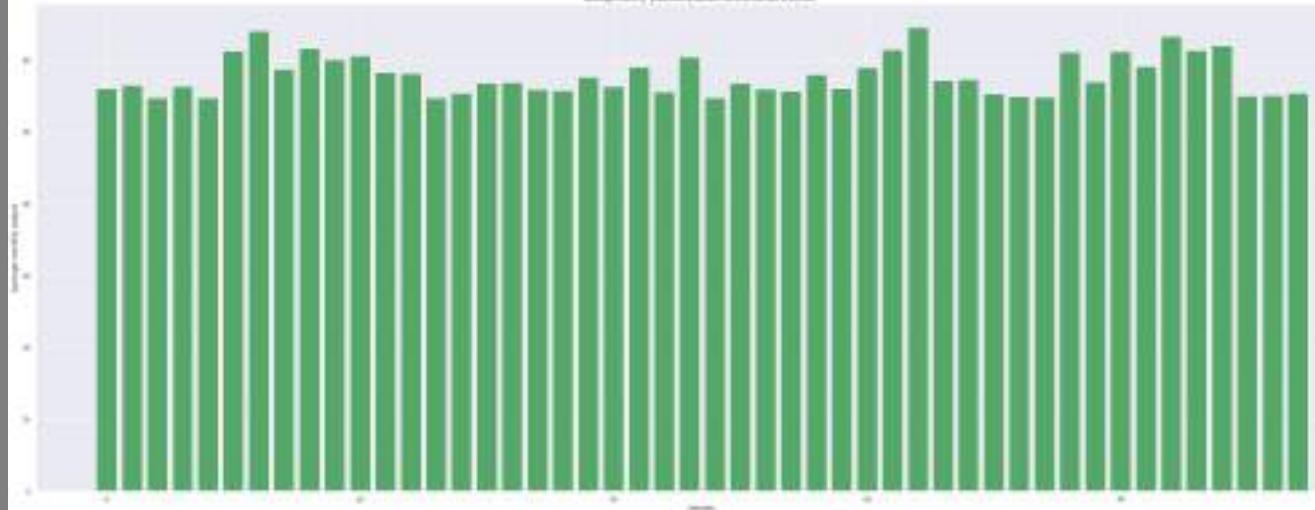
```

plt.bar(list(Wind_Quad_Dict.values()), Wind_Quad_Dict.keys(), color='g') #Predictive Quadratic Histogram
plt.title("Average monthly predicted quadratic wind prices per EUR/MWh ")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

plt.show()

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{56.04329713150446: 0, 56.46171035804166: 1, 54.72189377009627: 2, 56.261264579432776: 3, 54.726192786495695: 4, 61.29002980953234: 5, 64.03985534772364: 6, 58.730105505938454: 7, 61.60254103985825: 8, 60.03150036395134: 9, 60.60223441813166: 10, 58.333675442530314: 11, 58.1139050062686: 12, 54.68995118414506: 13, 55.35077885782508: 14, 56.77346636500779: 15, 56.888579101574315: 16, 55.8992291276222: 17, 55.70936510818535: 18, 57.657543801339365: 19, 56.32518146274837: 20, 59.05064677307334: 21, 55.593907857632075: 22, 60.45608846226824: 23, 54.700570182295706: 24, 56.81481424386922: 25, 55.93895536133555: 26, 55.64950412298969: 27, 57.95015221298191: 28, 56.03607304869448: 29, 58.99955677101843: 30, 61.45223204692398: 31, 64.55432381072195: 32, 57.15346466271086: 33, 57.28149426349869: 34, 55.29584525074563: 35, 54.89021755594394: 36, 54.79142682024407: 37, 61.1425634697473: 38, 56.95861400587563: 39, 61.205165898052336: 40, 59.096895038457546: 41, 63.31041943922497: 42, 61.391495391480305: 43, 61.96058238634525: 44, 54.93495932312234: 45, 55.06352677964472: 46, 55.34693688244426: 47}



The green bars represent the observation value for each respective month. This histogram is uniform throughout.

In []: df_Wind_Quad.describe(include = 'all') # Description Tables

	Price_Quad	Wind_Quad	First Difference	Seasonal Difference
count	48.000000	48.000000	47.000000	36.000000
mean	24.500000	57.859849	-0.014816	-0.076430
std	4.174498	2.771404	3.124463	3.395553
min	17.075344	54.689951	-7.400859	-8.330490
25%	21.390791	55.635605	-1.826948	-2.072032
50%	24.615282	56.923597	0.115113	0.163246
75%	27.214266	60.137647	2.207446	2.123026
max	36.035067	64.554324	6.563837	8.229142

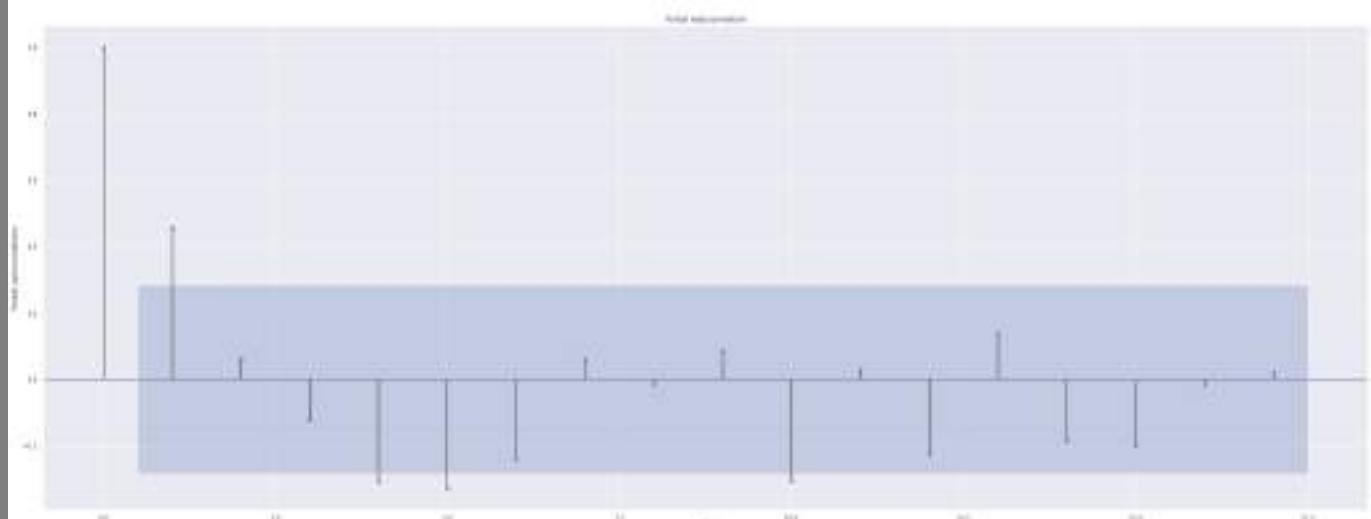
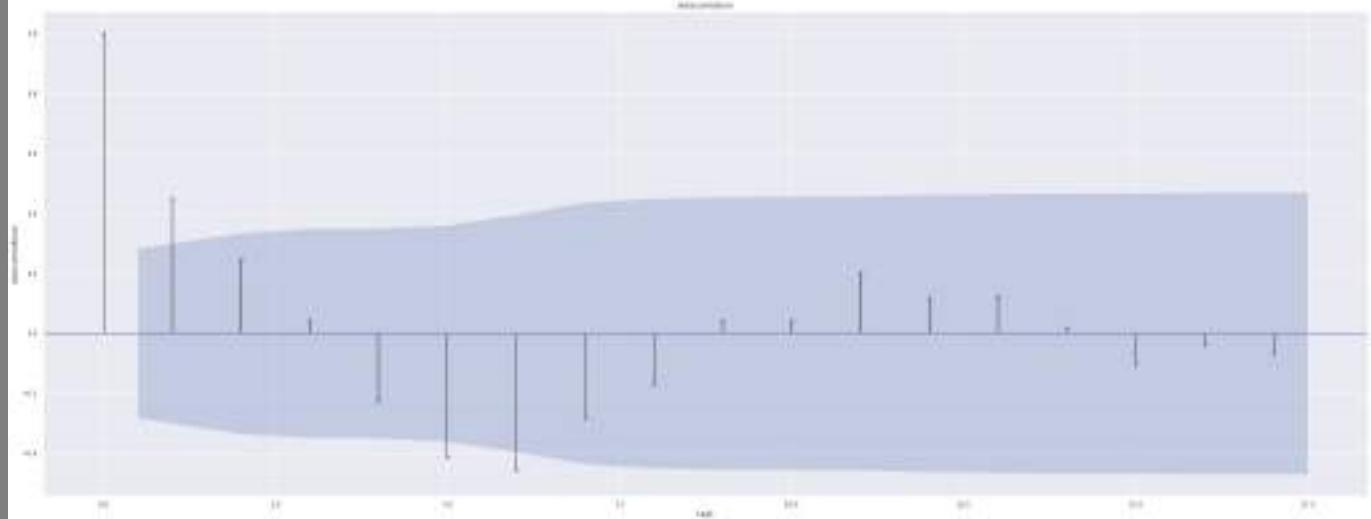
```

In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(Wind_Logpred)
plt.suptitle(" Autocorrelations of average monthly Wind Log")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot
plot_pacf(Wind_Logpred)
plt.suptitle("Partial Autocorrelations of average monthly Wind Log")
plt.ylabel('Partial autocorrealtions')
plt.xlabel('Lags')
plt.show
Wind_Log_Autocorrelations = sm.tsa.acf(Wind_Logpred, fft=False) #Autocorrelations
print(Wind_Log_Autocorrelations)

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.  
FutureWarning,
```

```
[ 1.          0.44811771  0.24702199  0.04171315 -0.2253772 -0.41182266  
-0.45414208 -0.28204515 -0.17086006  0.0392213  0.03945431  0.2002384  
0.11687314  0.12377872  0.01692702 -0.10284825 -0.03705665 -0.06771596  
-0.05390021 -0.09005239  0.05067668  0.06697361  0.10442004  0.10129298  
0.11968967  0.09451909 -0.04366922 -0.12275879 -0.18093863 -0.20867918  
-0.25662925 -0.13127842 -0.08221095  0.08565818  0.2743972  0.21425387  
0.20929809  0.21765469  0.09338864 -0.10867605 -0.17268571]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

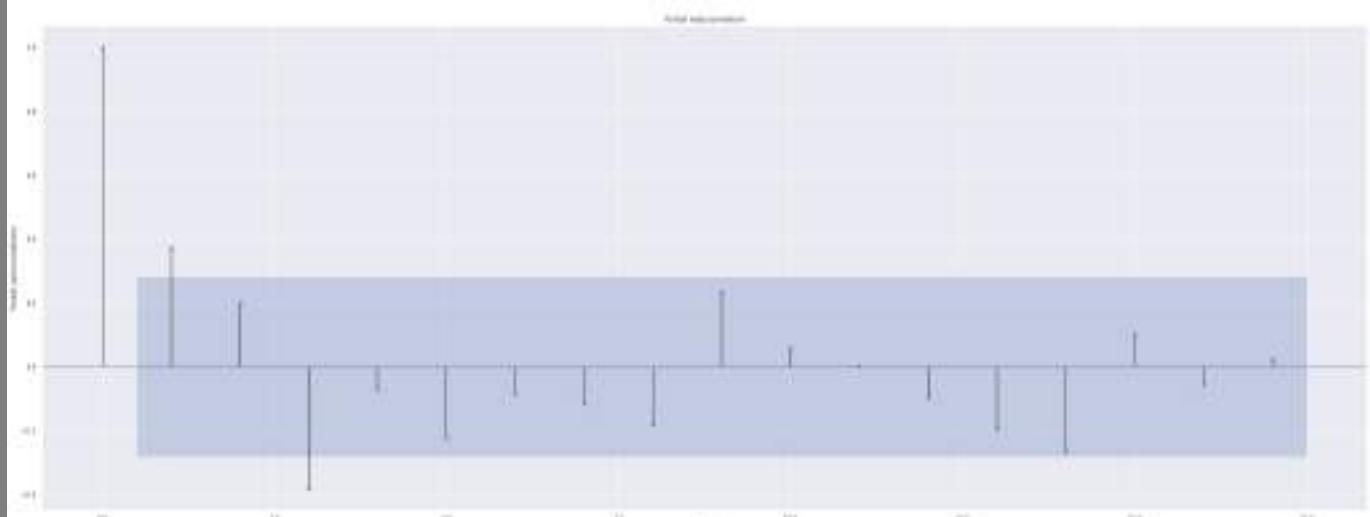
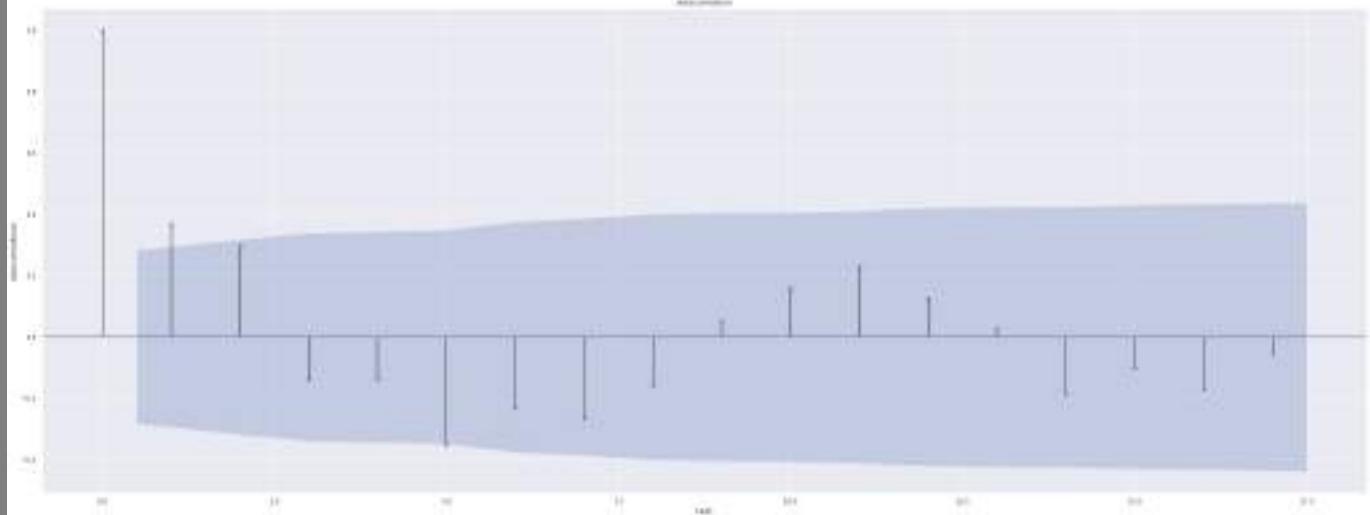
As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(Wind_ypred)  
plt.suptitle(" Autocorrelations of average monthly Quadratic Wind")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(Wind_ypred)  
plt.suptitle("Partial Autocorrelations of average monthly Quadratic Wind")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
Wind_Quad_Autocorrelations = sm.tsa.acf(Wind_ypred, fft=False) #Autocorrelations  
print(Wind_Quad_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.36468232  0.29679143 -0.13983578 -0.13754503 -0.35257203  
-0.23084437 -0.26498823 -0.1578457   0.050041   0.15901668  0.22784494  
0.12374168  0.02539725 -0.18558883 -0.10052796 -0.17041119 -0.05323301  
-0.14663938 -0.07772492  0.00607528  0.0637975   0.11048705  0.10803042  
0.04086687  0.02955977  0.04486775 -0.08004069 -0.15970722 -0.16689846  
-0.12234493 -0.05656512  0.17981367  0.20948639  0.20060264  0.14409376  
0.08720178  0.04090897 -0.04229999 -0.11787016 -0.20196331]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

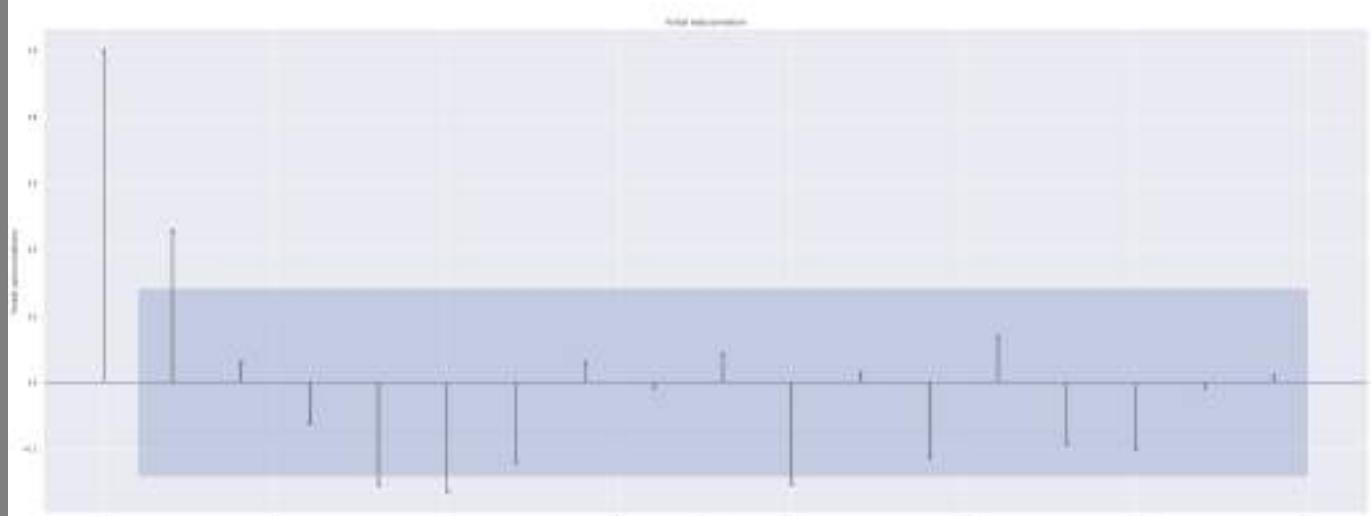
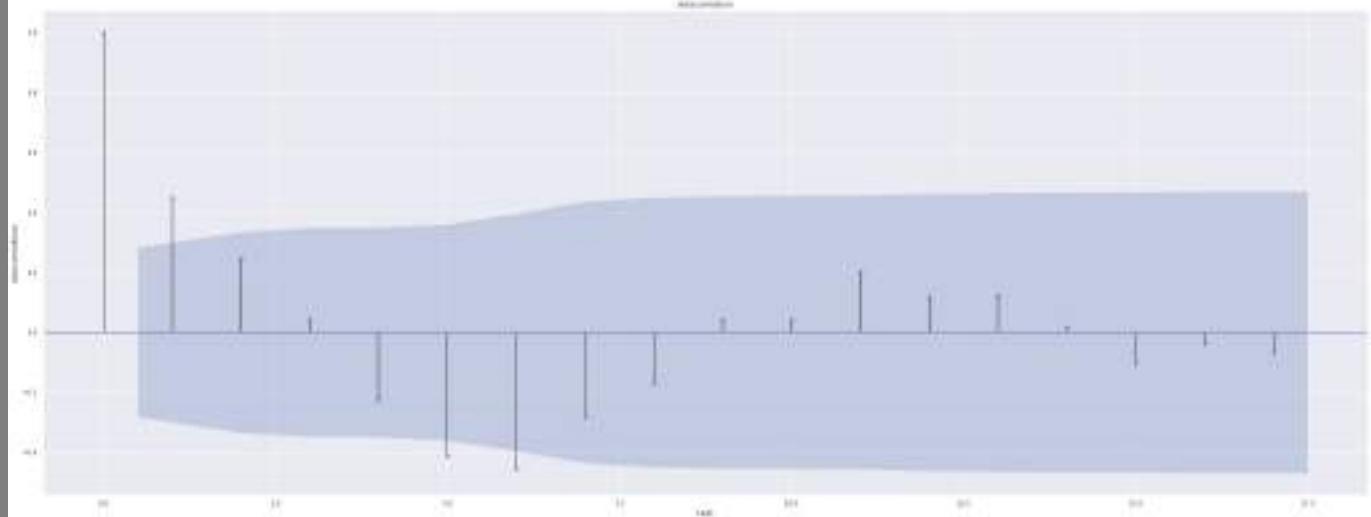
As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:  
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot  
plot_acf(predictionsWind)  
plt.suptitle(" Autocorrelations of average monthly Linear Wind")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf #Partialautocorrelation Plot  
plot_pacf(predictionsWind)  
plt.suptitle("Partial Autocorrelations of average monthly Linear Wind")  
plt.ylabel('Partial autocorrelations')  
plt.xlabel('Lags')  
plt.show  
Wind_Pred_Autocorrelations = sm.tsa.acf(predictionsWind, fft=False) #Autocorrelations  
print(Wind_Pred_Autocorrelations)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statauto.py:662: FutureWarning: The default number of lags is changing from 40 to min(int(10 * np.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

```
[ 1.          0.44811771  0.24702199  0.04171315 -0.2253772 -0.41182266  
-0.45414208 -0.28204515 -0.17086006  0.0392213  0.03945431  0.2002384  
0.11687314  0.12377872  0.01692702 -0.10284825 -0.03705665 -0.06771596  
-0.05390021 -0.09005239  0.05067668  0.06697361  0.10442004  0.10129298  
0.11968967  0.09451909 -0.04366922 -0.12275879 -0.18093863 -0.20867918  
-0.25662925 -0.13127842 -0.08221095  0.08565818  0.2743972  0.21425387  
0.20929809  0.21765469  0.09338864 -0.10867605 -0.17268571]
```



The plots above graphically summarize the strength of a relationship with an observation in a time series with observations at prior time steps. The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Each spike that rises above or falls below the shaded blue shapes is considered to be statistically significant. This means the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation. A spike that's close to zero is evidence against autocorrelation.

As one can observe, there is statistical significance in the partial autocorrelation plot, indicating that the lags directly beforehand influenced the relationship between average monthly predicted prices of energy per EUR/MWH for this particular resource and the average monthly predicted prices of energy per EUR/MWH.

```
In [ ]:
```

```
modelwindonshore = stats.linregress([7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886  
56.38385416666666,  
55.522462987886975,  
58.35408333333333,  
57.29405913978498,  
65.9749027777778,  
71.07204301075271,  
63.99806451612899,  
60.254791666666634,  
59.40676510067113,  
60.72679166666668,  
61.901760752688226,  
45.57872311827956,  
36.752083333333374,
```

```
36.81800807537014,  
32.61866666666666,  
34.691370967741896,  
46.266319444444434,  
47.50201612903221,  
47.6023387096774,  
50.4055972222224,  
60.182429530201404,  
62.58105555555558,  
67.5951344086021,  
79.49208333333331,  
59.83779761904767,  
50.95989232839837,  
51.71791666666662,  
53.77262096774189,  
56.2582222222224,  
55.252580645161316,  
54.08432795698931,  
55.81655555555558,  
63.92528859060403,  
65.43065277777781,  
65.15127688172035,  
56.51197580645163,  
60.877098214285674,  
48.279717362045766,  
50.40073611111113,  
61.633763440860214,  
64.34813888888884,  
67.78344086021498,  
70.36391129032262,  
76.9140416666666,  
70.36221476510062,  
67.0426075268817,  
66.62351388888881])
```

```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(wind,Price_Actual)  
print("slope: %f      intercept: %f" % (slope, intercept))
```

```
slope: -0.001736      intercept: 67.357851
```

```
In [ ]: import statsmodels.formula.api as smf
```

```
In [ ]:
```

```
In [ ]: print(nuclear)
```

```
[6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 5659.276773296245, 6483.623655913979, 6437.845430107527, 6466.175, 5854.012096774193, 5973.075, 6626.029609690444, 6223.8494623655915, 6159.264367816092, 6699.888290713325, 6706.406944444445, 5714.009408602151, 6647.4638888888885, 6628.9220430107525, 6633.3494623655915, 6675.755555555555, 6576.6859060402685, 5534.297222222222, 6325.970430107527, 6769.9166666666667, 6739.267857142857, 6755.951547779273, 6676.383333333333, 5561.240591397849, 6063.859722222222, 6117.403225806452, 6675.846774193548, 6427.68888888889, 6003.754362416107, 5565.216666666666, 6815.138440860215, 6723.689516129032, 6429.3735119047615, 6091.685060565276, 5504.175, 5465.200268817204, 5603.227777777778, 6121.515477792732, 6655.32123655914, 6621.99861111111, 6539.120805369127, 5403.497222222222, 5821.1518817204305]
```

```
In [ ]: print(Price_Actual)
```

```
[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.974902777777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.266319444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.2582222222222, 55.25258064516129, 54.08432795698924, 55.8165555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.633763440860222, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.62351388888889]
```

```
In [ ]: print(solar)
```

```
[1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657, 1998.6606397774688, 1973.7405913978494, 1738.9502688172042, 1591.2902777777779, 1082.279569892473, 1171.719444444444, 874.3135935397039, 1025.741935483871, 1258.058908045977, 1382.9057873485867, 1390.41666666666667, 1689.1814516129032, 1942.82916666666667, 1915.7752355316286, 1835.8521505376343, 1548.213888888889, 1013.2832214765101, 951.381944444445, 914.9758064516129, 1088.4180107526881, 1157.4613095238096, 1415.0915208613728, 1468.8277777777778, 1800.0524193548388, 1919.023611111112, 2025.119623655914, 1754.159946236559, 1731.36666666666666, 1344.3261744966444, 1180.645833333333, 1035.4260752688172, 1051.4193548387098, 1246.110119047619, 1352.3189771197847, 1501.647222222221, 1526.899193548387, 1855.197222222223, 2010.3458950201884, 1706.9301075268818, 1510.6527777777778, 999.786577181208, 831.668055555556, 875.5900537634409]
```

```
In [ ]: print(fossil_gas)
```

```
[4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915, 4934.930458970  
793, 6529.490591397849, 4919.491935483871, 5076.581944444444, 5231.5, 5387.741666666667, 5062.588156123822, 5271  
.1196236559135, 4450.360632183908, 4336.594885598924, 4263.481944444445, 4508.034946236559, 4994.327777777778, 5  
475.64333781965, 4869.987903225807, 4643.55, 6222.820134228188, 6439.015277777778, 6176.176075268817, 6412.59139  
7849463, 5561.144345238095, 5337.418573351279, 5169.840277777778, 5493.271505376344, 7194.280555555555, 7366.073  
924731183, 6701.935483870968, 7092.451388888889, 6961.8, 8534.818055555555, 5835.915322580645, 5687.715053763441  
, 5468.040178571428, 4936.013458950202, 4745.273611111111, 5777.162634408603, 5807.555555555556, 5865.6917900403  
77, 6119.346774193548, 5842.0625, 6127.481879194631, 6945.804166666667, 6463.5362903225805]
```

```
In [ ]: print( biomass )
```

```
[483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 485.9986091794  
1584, 512.4879032258065, 518.9475806451613, 517.1069444444445, 519.5564516129032, 503.6166666666667, 483.9246298  
7886944, 459.9959677419355, 430.3448275862069, 429.7442799461642, 286.890277777778, 337.2029569892473, 340.3375  
, 351.8896366083446, 369.64516129032256, 349.39166666666665, 336.2187919463087, 355.57301808066757, 338.77688172  
04301, 347.28360215053766, 351.3645833333333, 284.40242261103634, 280.761111111111, 353.5040322580645, 339.9791  
6666666667, 367.7069892473118, 362.8790322580645, 351.96805555555557, 343.938255033557, 354.59305555555557, 347.1  
733870967742, 343.8481182795699, 360.51190476190476, 307.2543741588156, 299.3930555555556, 318.30913978494624, 3  
51.73333333333335, 367.2207267833109, 361.3252688172043, 355.46805555555557, 317.9275167785235, 342.1027777777777  
76, 320.16129032258067]
```

```
In [ ]: print(wind)
```

```
[7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.49513212795  
6, 3955.2540322580644, 4856.173387096775, 4323.8, 4597.236559139785, 4493.763888888889, 4943.14131897712, 4993.3  
99193548387, 6534.337643678161, 5884.690444145357, 5343.191666666667, 5309.384408602151, 5638.5375, 5715.8331090  
17497, 5103.193548387097, 5484.695833333333, 4788.928859060403, 5766.394444444444, 4519.7661290322585, 6483.2983  
87096775, 5330.943452380952, 5623.149394347241, 5741.669444444445, 5031.903225806452, 5586.527777777777, 4799.47  
7150537635, 4348.514784946236, 3883.879166666666, 5234.8, 5200.188888888889, 7257.2553763440865, 6197.444892473  
119, 6864.555059523809, 8771.240915208613, 5289.247222222222, 4389.743279569892, 4779.433333333333, 4059.9071332  
43607, 4358.579301075269, 4265.988888888889, 6156.939597315436, 6057.558333333333, 5886.720430107527]
```

```
In [ ]: print(Fossil_Hard)
```

```
[5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.6129032258063, 6207.095966620  
306, 6748.469086021505, 5859.370967741936, 5653.873611111111, 5788.694892473119, 5982.631944444444, 5323.6769851  
95155, 4204.615591397849, 3027.655172413793, 2985.886944818304, 2226.186111111111, 2226.65188172043, 2877.080555  
5555557, 4003.9165545087485, 4172.501344086021, 4208.013888888889, 4271.040268456376, 4581.631944444444, 4984.29  
5698924731, 5533.782258064516, 4382.5327380952385, 2967.8492597577388, 3119.322222222222, 4428.919354838709, 509  
1.152777777777, 4561.283602150537, 3786.2325268817203, 4010.465277777778, 4321.130201342282, 5491.216666666666,  
4330.653225806452, 3783.6223118279568, 4144.389880952381, 2157.1830417227457, 2775.8791666666666, 3674.151881720  
43, 3059.616666666667, 4014.707940780619, 4121.565860215053, 4651.018055555555, 3782.8161073825504, 4838.7777777  
7777, 3365.7580645161293]
```

```
In [ ]: print(nuclear)
```

```
[6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 5659.27677329624  
5, 6483.623655913979, 6437.845430107527, 6466.175, 5854.012096774193, 5973.075, 6626.029609690444, 6223.84946236  
55915, 6159.264367816092, 6699.888290713325, 6706.406944444445, 5714.009408602151, 6647.463888888885, 6628.9220  
430107525, 6633.3494623655915, 6675.755555555555, 6576.6859060402685, 5534.297222222222, 6325.970430107527, 676  
9.916666666667, 6739.267857142857, 6755.951547779273, 6676.383333333333, 5561.240591397849, 6063.859722222222, 6  
117.403225806452, 6675.846774193548, 6427.688888888889, 6003.754362416107, 5565.216666666666, 6815.138440860215,  
6723.689516129032, 6429.3735119047615, 6091.685060565276, 5504.175, 5465.200268817204, 5603.227777777778, 6121.5  
15477792732, 6655.32123655914, 6621.998611111111, 6539.120805369127, 5403.497222222222, 5821.1518817204305]
```

```
In [ ]: print(hydro_water)
```

```
[2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.184139784946, 2531.168289290  
6815, 2412.5255376344085, 1963.0631720430108, 2261.156944444444, 2276.9193548387098, 2404.390277777776, 1943.4  
2799461642, 4075.5846774193546, 4881.568965517241, 3901.5450874831763, 5119.295833333334, 4581.743279569892, 298  
3.7930555555554, 2556.6971736204578, 2460.0201612903224, 2303.461111111111, 2500.489932885906, 2501.29722222222  
4, 2805.2970430107525, 2156.951612903226, 1874.8497023809523, 2348.4589502018844, 1611.4152777777779, 1639.61693  
5483871, 1448.644444444444, 1290.983870967742, 1260.6908602150538, 1570.509722222223, 1229.2845637583894, 1371  
.338888888889, 1452.9569892473119, 2271.896505376344, 2773.2038690476193, 4378.419919246298, 4159.898611111111,  
2931.19623655914, 3362.5291666666667, 2789.228802153432, 2296.0295698924733, 2267.641666666667, 2172.19194630872  
47, 2665.9, 2763.0241935483873]
```

```
In [ ]: print(fossil_oil)
```

```
[306.0204638472033, 319.239583333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963  
839, 360.23387096774195, 323.9139784946237, 343.4916666666667, 323.39112903225805, 346.06388888888887, 330.65141  
31897712, 337.46505376344084, 297.25431034482756, 294.0524890578735, 259.8875, 277.9153225806452, 289.829166666  
66665, 286.6900269541779, 283.30645161290323, 281.386111111111, 276.9959731543624, 271.5694444444446, 276.6397  
8494623655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.490277777778, 291.9206989247312, 299  
.3930555555556, 308.2002688172043, 306.23521505376345, 310.8013888888889, 287.4187919463087, 303.54305555555555,  
293.9260752688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916666666666, 280.45026881720  
43, 285.1208333333334, 281.5450874831763, 283.333333333333, 296.136111111111, 291.2993288590604, 272.9833333  
33335, 269.02150537634407]
```

```
In [ ]: print(fossil_brown)
```

Below is the multiple linear model for the predicted average monthly prices of energy per EUR/MWh.

```
In [ ]: #Dataframes  
No_Y_And_Constant = ({  
    'nuclear': [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 5659.  
    "solar": [1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657,  
    "fossil_fuel_gas": [4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.349462  
    "biomass": [483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183,  
    "wind_onshore": [7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.40860215053  
    "Fossil_Hard_coal": [5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.516713091922, 4019.6129032  
    "other_renewable": [70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306407, 70.56586021505  
    "fossil_oil": [306.0204638472033, 319.239583333333, 319.3337819650067, 338.78133704735376, 332.5672043010752  
    "hydro_water_reservoir": [2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.  
    "fossil_brown_coal": [572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.28091
```

```
In [ ]: import pandas as pd  
df_ResourcesNo_Y_And_Constant= pd.DataFrame.from_dict(No_Y_And_Constant, orient = "columns")  
print(df_ResourcesNo_Y_And_Constant) #Dataframes
```

	nuclear	solar	fossil_fuel_gas	biomass	wind_onshore	\
0	6665.969986	1130.392906	4850.009550	483.735334	7587.697135	
1	6681.123512	1244.525298	4674.135417	470.150298	7731.806548	
2	6687.913863	1283.479139	4614.752355	468.106326	6747.878869	
3	6068.169916	1461.519499	4952.123955	426.320334	5506.381616	
4	5403.817204	1920.272849	4415.349462	503.569892	6757.408602	
5	5659.276773	1998.660640	4934.930459	485.998609	4375.495132	
6	6483.623656	1973.740591	6529.490591	512.487903	3955.254032	
7	6437.845430	1738.950269	4919.491935	518.947581	4856.173387	
8	6466.175000	1591.290278	5076.581944	517.106944	4323.800000	
9	5854.012097	1082.279570	5231.500000	519.556452	4597.236559	
10	5973.075000	1171.719444	5387.741667	503.616667	4493.763889	
11	6626.029610	874.313594	5062.588156	483.924630	4943.141319	
12	6223.849462	1025.741935	5271.119624	459.995968	4993.399194	
13	6159.264368	1258.058908	4450.360632	430.344828	6534.337644	
14	6699.888291	1382.905787	4336.594886	429.744280	5884.690444	
15	6706.406944	1390.416667	4263.481944	286.890278	5343.191667	
16	5714.009409	1689.181452	4508.034946	337.202957	5309.384409	
17	6647.463889	1942.829167	4994.327778	340.337500	5638.537500	
18	6628.922043	1915.775236	5475.643338	351.889637	5715.833109	
19	6633.349462	1835.852151	4869.987903	369.645161	5103.193548	
20	6675.755556	1548.213889	4643.550000	349.391667	5484.695833	
21	6576.685906	1013.283221	6222.820134	336.218792	4788.928859	
22	5534.297222	951.381944	6439.015278	355.573018	5766.394444	
23	6325.970430	914.975806	6176.176075	338.776882	4519.766129	
24	6769.916667	1088.418011	6412.591398	347.283602	6483.298387	
25	6739.267857	1157.461310	5561.144345	351.364583	5330.943452	
26	6755.951548	1415.091521	5337.418573	284.402423	5623.149394	
27	6676.383333	1468.827778	5169.840278	280.761111	5741.669444	
28	5561.240591	1800.052419	5493.271505	353.504032	5031.903226	
29	6063.859722	1919.023611	7194.280556	339.979167	5586.527778	
30	6117.403226	2025.119624	7366.073925	367.706989	4799.477151	
31	6675.846774	1754.159946	6701.935484	362.879032	4348.514785	
32	6427.688889	1731.366667	7092.451389	351.968056	3883.879167	
33	6003.754362	1344.326174	6961.800000	343.938255	5234.800000	
34	5565.216667	1180.645833	8534.818056	354.593056	5200.188889	
35	6815.138441	1035.426075	5835.915323	347.173387	7257.255376	
36	6723.689516	1051.419355	5687.715054	343.848118	6197.444892	
37	6429.373512	1246.110119	5468.040179	360.511905	6864.555060	
38	6091.685061	1352.318977	4936.013459	307.254374	8771.240915	

39	5504.175000	1501.647222	4745.273611	299.393056	5289.247222
40	5465.200269	1526.899194	5777.162634	318.309140	4389.743280
41	5603.227778	1855.197222	5807.555556	351.733333	4779.433333
42	6121.515478	2010.345895	5865.691790	367.220727	4059.907133
43	6655.321237	1706.930108	6119.346774	361.325269	4358.579301
44	6621.998611	1510.652778	5842.062500	355.468056	4265.988889
45	6539.120805	999.786577	6127.481879	317.927517	6156.939597
46	5403.497222	831.668056	6945.804167	342.102778	6057.558333
47	5821.151882	875.590054	6463.536290	320.161290	5886.720430

	Fossil_Hard_coal	other_renewable	fossil_oil	hydro_water_reservoir	\
0	5411.263302	70.660300	306.020464	2572.339700	
1	4045.974702	70.514881	319.239583	3712.690476	
2	4233.818304	66.632571	319.333782	3081.620458	
3	4819.516713	69.700557	338.781337	2516.012535	
4	4019.612903	70.565860	332.567204	2798.184140	
5	6207.095967	65.511822	319.229485	2531.168289	
6	6748.469086	68.325269	360.233871	2412.525538	
7	5859.370968	68.176075	323.913978	1963.063172	
8	5653.873611	67.966667	343.491667	2261.156944	
9	5788.694892	70.669355	323.391129	2276.919355	
10	5982.631944	70.312500	346.063889	2404.390278	
11	5323.676985	71.623149	330.651413	1943.427995	
12	4204.615591	77.954301	337.465054	4075.584677	
13	3027.655172	74.454023	297.254310	4881.568966	
14	2985.886945	73.402423	294.052490	3901.545087	
15	2226.186111	78.275000	259.887500	5119.295833	
16	2226.651882	79.282258	277.915323	4581.743280	
17	2877.080556	83.483333	289.829167	2983.793056	
18	4003.916555	78.627187	286.690027	2556.697174	
19	4172.501344	78.430108	283.306452	2460.020161	
20	4208.013889	83.791667	281.386111	2303.461111	
21	4271.040268	83.781208	276.995973	2500.489933	
22	4581.631944	85.956944	271.569444	2501.297222	
23	4984.295699	90.090054	276.639785	2805.297043	
24	5533.782258	99.204301	279.522849	2156.951613	
25	4382.532738	94.909226	291.401786	1874.849702	
26	2967.849260	95.685061	302.507402	2348.458950	
27	3119.322222	95.970833	261.490278	1611.415278	
28	4428.919355	96.745968	291.920699	1639.616935	
29	5091.152778	92.105556	299.393056	1448.644444	
30	4561.283602	93.396505	308.200269	1290.983871	
31	3786.232527	92.247312	306.235215	1260.690860	
32	4010.465278	94.872222	310.801389	1570.509722	
33	4321.130201	95.626846	287.418792	1229.284564	
34	5491.216667	94.901389	303.543056	1371.338889	
35	4330.653226	91.815860	293.926075	1452.956989	
36	3783.622312	96.982527	285.739247	2271.896505	
37	4144.389881	96.757440	295.900298	2773.203869	
38	2157.183042	97.641992	288.126514	4378.419919	
39	2775.879167	97.151389	257.629167	4159.898611	
40	3674.151882	100.590054	280.450269	2931.196237	
41	3059.616667	101.109722	285.120833	3362.529167	
42	4014.707941	96.717362	281.545087	2789.228802	
43	4121.565860	96.615591	283.333333	2296.029570	
44	4651.018056	100.194444	296.136111	2267.641667	
45	3782.816107	98.617450	291.299329	2172.191946	
46	4838.777778	96.462500	272.983333	2665.900000	
47	3365.758065	95.881720	269.021505	2763.024194	

	fossil_brown_coal
0	572.851296
1	313.418155
2	244.437416
3	463.119777
4	374.280914
5	665.162726
6	684.220430
7	585.767473
8	548.083333
9	528.018817
10	695.284722
11	493.480485
12	417.927419
13	191.665230
14	173.203230
15	143.570833
16	179.002688
17	175.600000
18	398.815612
19	464.373656
20	473.720833
21	613.769128

```
22      649.545833
23      670.795699
24      688.645161
25      603.415179
26      335.667564
27      420.115278
28      500.770161
29      478.811111
30      650.677419
31      511.846774
32      642.630556
33      561.322148
34      667.647222
35      476.543011
36      379.564516
37      406.950893
38      124.415882
39      133.979167
40      307.704301
41      333.998611
42      506.368775
43      300.142473
44      558.161111
45      405.837584
46      375.736111
47      406.228495
```

```
In [ ]: import pandas as pd
df_ResourcesNO_Y = pd.DataFrame.from_dict(No_Y, orient = "columns")
print(df_ResourcesNO_Y) #Dataframes
```

```
constant      nuclear      solar      fossil_fuel_gas      biomass  \
0          1  6665.969986  1130.392906  4850.009550  483.735334
1          1  6681.123512  1244.525298  4674.135417  470.150298
2          1  6687.913863  1283.479139  4614.752355  468.106326
3          1  6068.169916  1461.519499  4952.123955  426.320334
4          1  5403.817204  1920.272849  4415.349462  503.569892
5          1  5659.276773  1998.660640  4934.930459  485.998609
6          1  6483.623656  1973.740591  6529.490591  512.487903
7          1  6437.845430  1738.950269  4919.491935  518.947581
8          1  6466.175000  1591.290278  5076.581944  517.106944
9          1  5854.012097  1082.279570  5231.500000  519.556452
10         1  5973.075000  1171.719444  5387.741667  503.616667
11         1  6626.029610  874.313594  5062.588156  483.924630
12         1  6223.849462  1025.741935  5271.119624  459.995968
13         1  6159.264368  1258.058908  4450.360632  430.344828
14         1  6699.888291  1382.905787  4336.594886  429.744280
15         1  6706.406944  1390.416667  4263.481944  286.890278
16         1  5714.009409  1689.181452  4508.034946  337.202957
17         1  6647.463889  1942.829167  4994.327778  340.337500
18         1  6628.922043  1915.775236  5475.643338  351.889637
19         1  6633.349462  1835.852151  4869.987903  369.645161
20         1  6675.755556  1548.213889  4643.550000  349.391667
21         1  6576.685906  1013.283221  6222.820134  336.218792
22         1  5534.297222  951.381944  6439.015278  355.573018
23         1  6325.970430  914.975806  6176.176075  338.776882
24         1  6769.916667  1088.418011  6412.591398  347.283602
25         1  6739.267857  1157.461310  5561.144345  351.364583
26         1  6755.951548  1415.091521  5337.418573  284.402423
27         1  6676.383333  1468.827778  5169.840278  280.761111
28         1  5561.240591  1800.052419  5493.271505  353.504032
29         1  6063.859722  1919.023611  7194.280556  339.979167
30         1  6117.403226  2025.119624  7366.073925  367.706989
31         1  6675.846774  1754.159946  6701.935484  362.879032
32         1  6427.688889  1731.366667  7092.451389  351.968056
33         1  6003.754362  1344.326174  6961.800000  343.938255
34         1  5565.216667  1180.645833  8534.818056  354.593056
35         1  6815.138441  1035.426075  5835.915323  347.173387
36         1  6723.689516  1051.419355  5687.715054  343.848118
37         1  6429.373512  1246.110119  5468.040179  360.511905
38         1  6091.685061  1352.318977  4936.013459  307.254374
39         1  5504.175000  1501.647222  4745.273611  299.393056
40         1  5465.200269  1526.899194  5777.162634  318.309140
41         1  5603.227778  1855.197222  5807.555556  351.733333
42         1  6121.515478  2010.345895  5865.691790  367.220727
43         1  6655.321237  1706.930108  6119.346774  361.325269
44         1  6621.998611  1510.652778  5842.062500  355.468056
45         1  6539.120805  999.786577  6127.481879  317.927517
46         1  5403.497222  831.668056  6945.804167  342.102778
47         1  5821.151882  875.590054  6463.536290  320.161290

wind_onshore      Fossil_Hard_coal      other_renewable      fossil_oil  \
0      7587.697135      5411.263302      70.660300  306.020464
1      7731.806548      4045.974702      70.514881  319.239583
```

2	6747.878869	4233.818304	66.632571	319.333782
3	5506.381616	4819.516713	69.700557	338.781337
4	6757.408602	4019.612903	70.565860	332.567204
5	4375.495132	6207.095967	65.511822	319.229485
6	3955.254032	6748.469086	68.325269	360.233871
7	4856.173387	5859.370968	68.176075	323.913978
8	4323.800000	5653.873611	67.966667	343.491667
9	4597.236559	5788.694892	70.669355	323.391129
10	4493.763889	5982.631944	70.312500	346.063889
11	4943.141319	5323.676985	71.623149	330.651413
12	4993.399194	4204.615591	77.954301	337.465054
13	6534.337644	3027.655172	74.454023	297.254310
14	5884.690444	2985.886945	73.402423	294.052490
15	5343.191667	2226.186111	78.275000	259.887500
16	5309.384409	2226.651882	79.282258	277.915323
17	5638.537500	2877.080556	83.483333	289.829167
18	5715.833109	4003.916555	78.627187	286.690027
19	5103.193548	4172.501344	78.430108	283.306452
20	5484.695833	4208.013889	83.791667	281.386111
21	4788.928859	4271.040268	83.781208	276.995973
22	5766.394444	4581.631944	85.956944	271.569444
23	4519.766129	4984.295699	90.090054	276.639785
24	6483.298387	5533.782258	99.204301	279.522849
25	5330.943452	4382.532738	94.909226	291.401786
26	5623.149394	2967.849260	95.685061	302.507402
27	5741.669444	3119.322222	95.970833	261.490278
28	5031.903226	4428.919355	96.745968	291.920699
29	5586.527778	5091.152778	92.105556	299.393056
30	4799.477151	4561.283602	93.396505	308.200269
31	4348.514785	3786.232527	92.247312	306.235215
32	3883.879167	4010.465278	94.872222	310.801389
33	5234.800000	4321.130201	95.626846	287.418792
34	5200.188889	5491.216667	94.901389	303.543056
35	7257.255376	4330.653226	91.815860	293.926075
36	6197.444892	3783.622312	96.982527	285.739247
37	6864.555060	4144.389881	96.757440	295.900298
38	8771.240915	2157.183042	97.641992	288.126514
39	5289.247222	2775.879167	97.151389	257.629167
40	4389.743280	3674.151882	100.590054	280.450269
41	4779.433333	3059.616667	101.109722	285.120833
42	4059.907133	4014.707941	96.717362	281.545087
43	4358.579301	4121.565860	96.615591	283.333333
44	4265.988889	4651.018056	100.194444	296.136111
45	6156.939597	3782.816107	98.617450	291.299329
46	6057.558333	4838.777778	96.462500	272.983333
47	5886.720430	3365.758065	95.881720	269.021505

	hydro_water_reservoir	fossil_brown_coal
0	2572.339700	572.851296
1	3712.690476	313.418155
2	3081.620458	244.437416
3	2516.012535	463.119777
4	2798.184140	374.280914
5	2531.168289	665.162726
6	2412.525538	684.220430
7	1963.063172	585.767473
8	2261.156944	548.083333
9	2276.919355	528.018817
10	2404.390278	695.284722
11	1943.427995	493.480485
12	4075.584677	417.927419
13	4881.568966	191.665230
14	3901.545087	173.203230
15	5119.295833	143.570833
16	4581.743280	179.002688
17	2983.793056	175.600000
18	2556.697174	398.815612
19	2460.020161	464.373656
20	2303.461111	473.720833
21	2500.489933	613.769128
22	2501.297222	649.545833
23	2805.297043	670.795699
24	2156.951613	688.645161
25	1874.849702	603.415179
26	2348.458950	335.667564
27	1611.415278	420.115278
28	1639.616935	500.770161
29	1448.644444	478.811111
30	1290.983871	650.677419
31	1260.690860	511.846774
32	1570.509722	642.630556
33	1229.284564	561.322148
34	1371.338889	667.647222

```

35      1452.956989      476.543011
36      2271.896505      379.564516
37      2773.203869      406.950893
38      4378.419919      124.415882
39      4159.898611      133.979167
40      2931.196237      307.704301
41      3362.529167      333.998611
42      2789.228802      506.368775
43      2296.029570      300.142473
44      2267.641667      558.161111
45      2172.191946      405.837584
46      2665.900000      375.736111
47      2763.024194      406.228495

```

```
In [ ]: Price_Actual = [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 0]
```

```
In [ ]: result = smf.ols('Price_Actual ~ + wind_onshore + nuclear + Fossil_Hard_coal + other_renewable + fossil_oil'
print(result_.summary()) #OLS Multilinear Summary Table
```

OLS Regression Results

```
=====
Dep. Variable:      Price_Actual    R-squared:           0.744
Model:                 OLS    Adj. R-squared:        0.675
Method:               Least Squares   F-statistic:         10.77
Date:       Wed, 30 Nov 2022   Prob (F-statistic):  2.93e-08
Time:          05:29:33    Log-Likelihood:     -146.91
No. Observations:      48    AIC:                  315.8
Df Residuals:          37    BIC:                  336.4
Df Model:                10
Covariance Type:    nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-55.7365	34.633	-1.609	0.116	-125.910	14.437
wind_onshore	0.0007	0.001	0.679	0.501	-0.001	0.003
nuclear	0.0013	0.002	0.600	0.552	-0.003	0.006
Fossil_Hard_coal	0.0077	0.002	3.541	0.001	0.003	0.012
other_renewable	0.7652	0.159	4.824	0.000	0.444	1.087
fossil_oil	-0.0663	0.075	-0.883	0.383	-0.218	0.086
solar	-0.0013	0.003	-0.454	0.652	-0.007	0.004
fossil_fuel_gas	-0.0005	0.002	-0.267	0.791	-0.004	0.003
biomass	0.0680	0.038	1.767	0.086	-0.010	0.146
hydro_water_reservoir	0.0004	0.002	0.233	0.817	-0.003	0.004
fossil_brown_coal	0.0010	0.012	0.088	0.931	-0.023	0.025

```
=====
Omnibus:            1.916    Durbin-Watson:        0.877
Prob(Omnibus):      0.384    Jarque-Bera (JB):    1.226
Skew:              -0.003    Prob(JB):            0.542
Kurtosis:           2.217    Cond. No.        4.63e+05
=====
```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 4.63e+05. This might indicate that there are strong multicollinearity or other numerical problems.

Our null hypothesis is that the coefficients total average nuclear,solar,fossil fuel gas, wind onshore, fossil hard coal, hydro, other renewable,fossil oil, hydro water reservoir and fossil brown coal/lignite generated are all 0.

Our alternative hypothesis is that the coefficients total average nuclear,solar,fossil fuel gas, wind onshore, fossil hard coal, hydro, other renewable,fossil oil, hydro water reservoir and fossil brown coal/lignite generated are not 0.

We will evaluate the significance behind these results under the alpha threshold of .05. This would mean that any variables having an impact of 5% or less of the time would be deemed significant. Furthermore, this would indicate that the average monthly outputs of a resource is indeed impacted by the average monthly price of energy per EUR/MWH and by other resources in the regression. Given that the p values of respective monthly nuclear,solar,fossil fuel gas,biomass, wind onshore, fossil hard coal, other renewable,fossil oil, hydro water reservoir and fossil brown coal/lignite output averages generated in a given month from 2015 to 2018 compared to the total average price of energy per EUR/MWH are 0.978, 0.786,0.342,0.000,0.001,0.498,0.070, 0.977,0.647 and 0.497 respectively, one could conclude that there is significance between the average price of energy per EUR/MWH and the hydro generated from 2015 to 2018. In addition, one could conclude that there is significance between the average price of energy per EUR/MWH and other renewable energy generated from 2015 to 2018.This would imply that the average monthly prices of energy per EUR/MWH from 2015 to 2018 in Spain are affected by the output amount generated from fossil hard coal and other renewables; making them significantly elastic. Below is the predicted model generated from this particular regression.

0.0013(nuclear) + 0.0007(wind) -0.077(fossil hard coal)+ 0.7652(other renewable) - 0.0663(fossil oil) - 0.0013(solar) - 0.0005(fossil fuel gas) + 0.0680(biomass) + 0.0004(hydro) + 0.0010(fossil brown coal) -55.7365

Given the multi variable regression model above, one can observe and conclude patterns of volatility amongst the variables. For example, when the production of nuclear power increases by one unit, the average price of energy per EUR/MWH increases by 0.0013

units. When the production of solar increases by one unit, the average price of energy per EUR/MWH decreases by 0.0013 units. When the production of fossil fuel gas increases by one unit, the average price of energy per EUR/MWH decreases by .0005 units. When the production of wind onshore increases by one unit, the average price of energy per EUR/MWH increases by .7494 units. When the production of fossil hard coal increases by one unit, the price of energy per EUR/MWH decreases by .0077 units. When the production of hydro increases by one unit, the price of energy per EUR/MWH increases by .0004 units. When the production of other renewables increases by one unit, the price of energy per EUR/MWH increases by .7652 units. When the production of fossil oil increases by one unit, the price of energy per EUR/MWH decreases by .0663 units. When the production of hydro power increases by one unit, the price of energy per EUR/MWH decreases by .0004 units. When the production of fossil brown coal increases by one unit, the price of energy per EUR/MWH increases by .0010 units.

Given these circumstances, it appears that the average price of energy per EUR/MWH decreases when fossil fuel and oil production increase. In addition, the average price of energy per EUR/MWH decreases when hydro power production increases. However, since their p values were above the alpha threshold, it would not be fair to assume that this is truly the case.

Based off this regression analysis, the price of energy per EUR/MWH is dependant to the outputs of some resources nonetheless.

```
In [ ]: influence1 = result_.get_influence()# Multiple regression OLS residuals

standardized_residuals = influence1.resid_studentized_internal

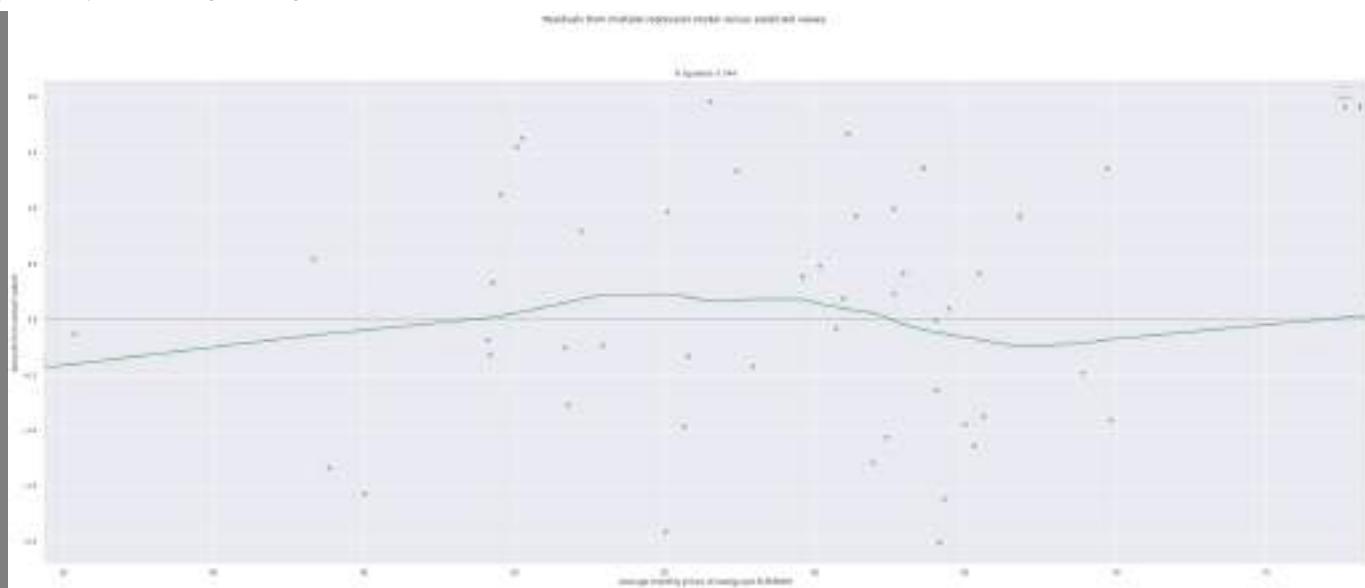
print(list(standardized_residuals))

#Multiple linear regression residual plot

sns.residplot(x = predictionsresults, y = standardized_residuals, lowess = True, color="g")
plt.suptitle("Residuals from multiple regression model versus predicted values")
plt.title("R squared: 0.744")
plt.xlabel("Average monthly prices of energy per EUR/MWH")
plt.ylabel("Amount from actual values")
plt.legend(..#)
```

```
[0.09984542386801075, 0.7938498013453089, 1.1262010594367513, 1.635563974743426, 1.558752583308333, 0.9332608814915974, 0.9277832359842159, -0.00432779180954391, -0.0840636943325935, -1.1365885179207715, -0.6319313618056847, 0.18818654209795854, -1.9048407234896907, -1.562068279830994, -1.3347992810492972, -0.36662615400205106, -0.12948768807844255, 0.5511496809656857, -0.3158090624331396, -0.7639141421582851, -0.9608461134920531, 0.970662551053189, 0.4834069396821492, 0.414151515392992, 0.22495102233146844, -0.9465963305786229, 0.3367850333820362, -0.2327216661790827, -2.0052439223287326, -1.6151046617331908, -1.287855154880161, -0.3277374960470071, -0.41761716206191546, 0.23405872945588574, -0.9065572175867098, 0.41589481097873193, -1.059230792884355, -0.86742289665914, -0.1832100285149976, -0.25312781097081444, 0.3912774976132853, 1.3430827057438444, 1.0007771590302654, 1.3690343713562232, 1.3612363953124003, 1.6710464923614174, -0.47762776367874754, 1.9625104872590995]
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff603bbe490>
```



As one can observe, there is a subtle hump along the lowess line in the residual plot. In addition, there residuals are quite spread out in no particular pattern which indicates constant variance, a lack of bias, and homoscedasticity. Given these circumstances, it would be reasonable to assume that the multiple regression model of fit is suitable. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #OLS Multilinear Scatterplot
df_ResourcesNO_Y1 = df_ResourcesNO_Y
df_ResourcesNO_Y1 = sm.add_constant(df_ResourcesNO_Y1)
```

```

predictionsresults = result_.predict(df_ResourcesNO_Y1)
print(predictionsresults)
plt.plot(df_ResourcesNO_Y['biomass'],Price_Actual,"r.")
plt.plot(df_ResourcesNO_Y['nuclear'],Price_Actual,"g.")
plt.plot(df_ResourcesNO_Y['Fossil_Hard_coal'],Price_Actual,'m.')
plt.plot(df_ResourcesNO_Y['fossil_oil'],Price_Actual, 'y.')
plt.plot(df_ResourcesNO_Y['solar'],Price_Actual, 'k.')
plt.plot(df_ResourcesNO_Y['hydro_water_reservoir'],Price_Actual,'c.')
plt.plot(df_ResourcesNO_Y['fossil_brown_coal'],Price_Actual,'b.')
plt.plot(df_ResourcesNO_Y['wind_onshore'],Price_Actual,"o",color = '0.2')
plt.plot(df_ResourcesNO_Y['Fossil_Hard_coal'],Price_Actual,'o',color = '0.8')
plt.plot(df_ResourcesNO_Y['other_renewable'],Price_Actual,"o",color = '0.75')
x = np.linspace(0,10000,48)
plt.suptitle("Predicted multilinear average monthly outputs of all resources versus average monthly prices of energy")
plt.title("R squared : 0.744")
plt.ylabel("Average monthly prices of energy per EUR/MWh")
plt.xlabel("Average monthly output")

plt.plot(x, predictionsresults)

```

```

0    64.446393
1    52.209858
2    49.546940
3    50.265259
4    50.079152
5    61.357328
6    66.832693
7    64.021262
8    60.715044
9    65.280009
10   64.038154
11   60.937027
12   55.016679
13   45.020112
14   43.824661
15   34.338128
16   35.363781
17   43.305112
18   49.195383
19   51.774700
20   55.640053
21   55.092853
22   60.162553
23   65.436593
24   78.401833
25   64.995108
26   49.275338
27   52.929806
28   64.148790
29   64.302920
30   61.937268
31   55.787776
32   57.947545
33   62.641225
34   69.835742
35   62.928831
36   62.389740
37   65.604841
38   49.122206
39   51.691764
40   59.576439
41   57.394384
42   62.614986
43   63.610852
44   69.718850
45   61.091618
46   68.905620
47   56.519510
dtype: float64

```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of pandas all arguments of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)

```

```

Out[ ]: [<matplotlib.lines.Line2D at 0x7ff6039893d0>]

```



The blue line represents the multiple linear model calculated in the multiple linear regression. The dots within the graph represent the observations of each resource by color. These dots are represented as follows: biomass is red, nuclear is green, fossil hard coal is magenta, fossil oil is yellow, hydro is cyan, fossil brown coal is blue. For the abnormally represented resources; they are represented as follows: solar observations are the smaller black dots, wind observations are the larger black dots, fossil gas observations are the larger gray dots while other renewable representations are the smaller gray dots. Overall, there is a positively strong correlation between the average monthly prices of energy per EUR/MWH and the average monthly outputs of each resource analyzed.

```
In [ ]: print(list(predictionsresults))

[64.44639257240718, 52.20985779008717, 49.54693969874262, 50.265258852453215, 50.079151601975944, 61.35732785391
12, 66.83269286003241, 64.02126150911813, 60.71504399343908, 65.28000908225576, 64.03815417233908, 60.9370274827
27075, 55.01667877768171, 45.020112377685365, 43.824660897588245, 34.33812841307532, 35.36378095666895, 43.30511
18811379, 49.19538329178341, 51.77469978881437, 55.640053400172015, 55.09285280948558, 60.16255313816916, 65.436
5926093738, 78.40183260004486, 64.99510805378303, 49.27533769957097, 52.92980620528309, 64.14879021490862, 64.3
0292023201893, 61.937268059392714, 55.787775846134366, 57.94754532721058, 62.64122456143839, 69.83574247112989,
62.92883095996241, 62.389739505244165, 65.6048406711875, 49.12220603308488, 51.69176377825795, 59.57643929110074
, 57.39438421369175, 62.61498598425229, 63.610851948248595, 69.71884989582897, 61.09161760816475, 68.90561989944
31, 56.51950954221806]

In [ ]: dfMultiLinear = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.3540833333333335, 57.29405
"MultiLinear" : [64.4428230282671, 52.20157898985954, 49.56816869321892, 50.28031014217656, 50.04644249104982, 0
print(dfMultiLinear) #Dataframes
df_MultiLinear= pd.DataFrame.from_dict(dfMultiLinear, orient = "columns")
print(df_MultiLinear)

#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_MultiLinear["MultiLinear"])

test_result=adfuller(df_MultiLinear["MultiLinear"])

df_MultiLinear['First Difference MultiLinear'] = df_MultiLinear["MultiLinear"]- df_MultiLinear["MultiLinear"].shift(1)
df_MultiLinear['Seasonal Difference MultiLinear']=df_MultiLinear["MultiLinear"]- df_MultiLinear["MultiLinear"].shift(12)
```

```

df_MultiLinear.head()
plt.suptitle("Seasonal Difference of average monthly multilinear outputs of all resources to the predicted price")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_MultiLinear['Seasonal Difference MultiLinear'].plot()
# Seasonality Plot

```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'MultiLinear': [64.4428230282671, 52.20157898985954, 49.56816869321892, 50.28031014217656, 50.04644249104982, 61.38958503030355, 66.88188216397906, 64.02949403524474, 60.7206256922445, 65.30423328925912, 64.03046387443615, 60.94268603473407, 54.99392254777341, 45.00180466335265, 43.79762069239902, 34.328977182268645, 35.33288458312469, 43.30890747490712, 49.20235501896019, 51.762928561473906, 55.63536808132733, 55.0511372001192, 60.13699936438, 65.43354182673784, 78.43164624855042, 64.96040052885877, 49.25430862115097, 52.89062966983732, 64.16425379052689, 64.3752902267347, 61.90345945929638, 55.73349871509462, 57.87702992345805, 62.62424883960866, 69.8717253647544, 62.93136189821938, 62.389467241217616, 65.62426037003297, 49.12376390619883, 51.735972094628025, 59.618322159030264, 57.374079534767034, 62.593420352658114, 63.65572956442148, 69.72298951203331, 61.095241528822115, 68.99819318747669, 56.498683065317934]}

```

	Price	MultiLinear
0	64.949019	64.442823
1	56.383854	52.201579
2	55.522463	49.568169
3	58.354083	50.280310
4	57.294059	50.046442
5	65.974903	61.389585
6	71.072043	66.881882
7	63.998065	64.029494
8	60.254792	60.720626
9	59.406765	65.304233
10	60.726792	64.030464
11	61.901761	60.942686
12	45.578723	54.993923
13	36.752083	45.001805
14	36.818008	43.797621
15	32.618667	34.328977
16	34.691371	35.332885
17	46.266319	43.308907
18	47.502016	49.202355
19	47.602339	51.762929
20	50.405597	55.635368
21	60.182430	55.051137
22	62.581056	60.136999
23	67.595134	65.433542
24	79.492083	78.431646
25	59.837798	64.960401
26	50.959892	49.254309
27	51.717917	52.890630
28	53.772621	64.164254
29	56.258222	64.375290
30	55.252581	61.903459
31	54.084328	55.733499
32	55.816556	57.877030
33	63.925289	62.624249
34	65.430653	69.871725
35	65.151277	62.931362
36	56.511976	62.389467
37	60.877098	65.624260
38	48.279717	49.123764
39	50.400736	51.735972
40	61.633763	59.618322
41	64.348139	57.374080
42	67.783441	62.593420
43	70.363911	63.655730
44	76.914042	69.722990
45	70.362215	61.095242
46	67.042608	68.998193
47	66.623514	56.498683

ADF Test Statistic : -2.9913668014973256

p-value : 0.03570853585643594

#Lags Used : 0

Number of Observations : 47

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff603966190>

The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the months and the average monthly prices of energy per EUR/MWh.

```
In [1]: #Bell Curves
```

```
MultiLinearResults_mean = np.mean(df_MultiLinear["MultiLinear"])
MultiLinearResults_std = np.std(df_MultiLinear["MultiLinear"])

MultiLinearResultspdf = stats.norm.pdf(df_MultiLinear["MultiLinear"].sort_values(), MultiLinearResults_mean, Mu

plt.plot(df_MultiLinear["MultiLinear"].sort_values(), MultiLinearResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_MultiLinear["MultiLinear"])}')
plt.suptitle("Frequency distribution of average monthly predicted multilinear prices of energy per EUR/MWh (sc
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

This bell shaped curve is slightly skewed to the left. Hence, it has an asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: print(dicDates)
MultiLinear Dict = {key: i for i, key in enumerate(df MultiLinear["MultiLinear"])}{}
```

```

def Hist_MultiLinear(MultiLinear_Dict):
    for k, v in MultiLinear_Dict.items(): print(f"{v}:{k}")
print(MultiLinear_Dict)

plt.bar(list(MultiLinear_Dict.values()), MultiLinear_Dict.keys(), color='g')
plt.title("Average monthly predicted multilinear prices of energy per EUR/MWH")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

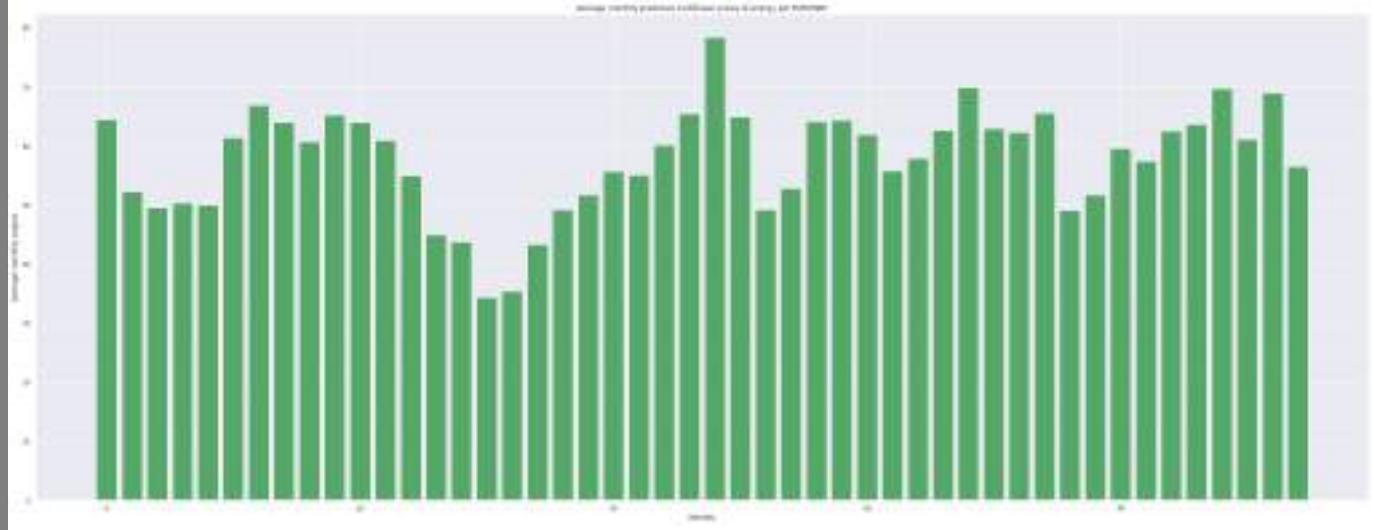
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{64.4428230282671: 0, 52.20157898985954: 1, 49.56816869321892: 2, 50.28031014217656: 3, 50.04644249104982: 4, 61.38958503030355: 5, 66.88188216397906: 6, 64.02949403524474: 7, 60.7206256922445: 8, 65.30423328925912: 9, 64.03046387443615: 10, 60.94268603473407: 11, 54.99392254777341: 12, 45.00180466335265: 13, 43.79762069239902: 14, 34.328977182268645: 15, 35.33288458312469: 16, 43.30890747490712: 17, 49.20235501896019: 18, 51.762928561473906: 19, 55.63536808132733: 20, 55.0511372001192: 21, 60.13699936438: 22, 65.43354182673784: 23, 78.43164624855042: 24, 64.96040052885877: 25, 49.25430862115097: 26, 52.89062966983732: 27, 64.16425379052689: 28, 64.3752902267347: 29, 61.90345945929638: 30, 55.73349871509462: 31, 57.87702992345805: 32, 62.62424883960866: 33, 69.8717253647544: 34, 62.93136189821938: 35, 62.389467241217616: 36, 65.62426037003297: 37, 49.12376390619883: 38, 51.735972094628025: 39, 59.618322159030264: 40, 57.374079534767034: 41, 62.593420352658114: 42, 63.65572956442148: 43, 69.72298951203331: 44, 61.095241528822115: 45, 68.99819318747669: 46, 56.498683065317934: 47}

```



The green bars represent the observation value for each respective month.

Below is the multiple polynomial model for the predicted average monthly prices of energy per EUR/MWH.

```
In [ ]: result_poly = smf.ols('y ~ nuclear + I(nuclear**2) + wind_onshore + I(wind_onshore**2) + Fossil_Hard_coal + I(Fossil_Hard_coal**2)', data=energy)
print(result_poly.summary()) #OLS Multipolynomial Summary Table
```

OLS Regression Results

```
=====
Dep. Variable:                      y      R-squared:                   1.000
Model:                            OLS      Adj. R-squared:             1.000
Method:                           Least Squares      F-statistic:                 7422.
Date:                            Wed, 30 Nov 2022      Prob (F-statistic):        2.05e-45
Time:                             05:33:26      Log-Likelihood:                102.92
No. Observations:                  48      AIC:                     -163.8
Df Residuals:                      27      BIC:                     -124.5
Df Model:                          20
Covariance Type:                nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	77.9120	2.003	38.897	0.000	73.802	82.022
nuclear	-0.0008	0.001	-1.499	0.146	-0.002	0.000
I(nuclear ** 2)	7.152e-08	4.57e-08	1.564	0.130	-2.23e-08	1.65e-07
wind_onshore	-0.0039	6.19e-05	-62.483	0.000	-0.004	-0.004
I(wind_onshore ** 2)	1.586e-07	5.02e-09	31.582	0.000	1.48e-07	1.69e-07
Fossil_Hard_coal	4.3e-05	7.77e-05	0.553	0.585	-0.000	0.000
I(Fossil_Hard_coal ** 2)	-6.603e-09	8.34e-09	-0.792	0.435	-2.37e-08	1.05e-08
other_renewable	-0.0219	0.018	-1.221	0.233	-0.059	0.015
I(other_renewable ** 2)	0.0001	0.000	1.261	0.218	-8.1e-05	0.000
fossil_oil	-0.0076	0.007	-1.019	0.317	-0.023	0.008
I(fossil_oil ** 2)	1.222e-05	1.21e-05	1.007	0.323	-1.27e-05	3.71e-05
solar	2.158e-05	0.000	0.102	0.920	-0.000	0.000
I(solar ** 2)	3.216e-09	7.14e-08	0.045	0.964	-1.43e-07	1.5e-07
fossil_fuel_gas	7.805e-05	0.000	0.736	0.468	-0.000	0.000
I(fossil_fuel_gas ** 2)	-4.232e-09	8.16e-09	-0.518	0.608	-2.1e-08	1.25e-08
biomass	0.0008	0.002	0.428	0.672	-0.003	0.005
I(biomass ** 2)	-5.442e-07	2.22e-06	-0.245	0.808	-5.1e-06	4.01e-06
hydro_water_reservoir	1.502e-05	4.22e-05	0.356	0.724	-7.15e-05	0.000
I(hydro_water_reservoir ** 2)	7.202e-10	7e-09	0.103	0.919	-1.37e-08	1.51e-08
fossil_brown_coal	0.0002	0.000	0.504	0.618	-0.000	0.001
I(fossil_brown_coal ** 2)	-8.879e-08	3.36e-07	-0.264	0.794	-7.79e-07	6.01e-07
Omnibus:	5.294	Durbin-Watson:		2.561		
Prob(Omnibus):	0.071	Jarque-Bera (JB):		4.407		
Skew:	0.501	Prob(JB):		0.110		
Kurtosis:	4.096	Cond. No.		2.34e+10		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.34e+10. This might indicate that there are strong multicollinearity or other numerical problems.

Our null hypothesis is that the coefficients total average nuclear,solar,fossil fuel gas, wind onshore, fossil hard coal, hydro, other renewable,fossil oil, hydro water reservoir and fossil brown coal/lignite generated are all 0.

Our alternative hypothesis is that the coefficients total average nuclear,solar,fossil fuel gas, wind onshore, fossil hard coal, hydro, other renewable,fossil oil, hydro water reservoir and fossil brown coal/lignite generated are not 0.

We will evaluate the significance behind these results under the alpha threshold of .05. This would mean that any variables having an impact of 5% or less of the time would be deemed significant. Furthermore, this would indicate that the average monthly outputs of a resource is indeed impacted by the average monthly price of energy per EUR/MWH and by other resources in the regression. Given that the p values of respective monthly nuclear,solar,fossil fuel gas,biomass, wind onshore, fossil hard coal, other renewable,fossil oil, hydro water reservoir and fossil brown coal/lignite output averages generated in a given month from 2015 to 2018 compared to the total average price of energy per EUR/MWH are 0.146,0.130,0.000,0.585,0.435,0.233,0.317,0.920,0.468,0.672,0.724, and 0.618 respectively, one could conclude that there is significance between the average price of energy per EUR/MWH and the hydro generated from 2015 to 2018. In addition, one could conclude that there is significance between the average price of energy per EUR/MWH and other renewable energy generated from 2015 to 2018.This would imply that the average monthly prices of energy per EUR/MWH from 2015 to 2018 in Spain are affected by the output amount generated from wind outputs; making wind power significantly elastic. Below is the predicted model generated from this particular regression:

$$\begin{aligned}
 &-0.0008(\text{nuclear}) + 7.152e-08^2(\text{nuclear}) -0.0039(\text{wind}) + 1.586e-07^2(\text{wind}) + 4.3e-05(\text{fossil hard coal}) - -6.603e-09(\text{fossil hard coal}) \\
 &-0.0219(\text{other renewable}) + 0.0001^2(\text{other renewable}) -.0076(\text{fossil oil}) + 1.222e-05^2(\text{fossil oil}) + 2.158e-05(\text{solar}) + 3.216e-09^2(\text{solar}) \\
 &+ 7.805e-05(\text{fossil fuel gas}) - 4.232e-09^2(\text{fossil fuel gas}) + 0.0008(\text{biomass}) - 5.442e-07^2(\text{biomass}) + 1.502e-05(\text{hydro}) + 7.202e-10^2(\text{hydro}) \\
 &+ 0.0002(\text{fossil brown coal}) - 8.879e-08^2(\text{fossil brown coal})+ 77.9120
 \end{aligned}$$

Given the multi variable regression model above, one can observe and conclude patterns of volatility amongst the variables. For example, when the production of nuclear power increases by one unit, the average price of energy per EUR/MWH decreases by 0.0008 units. When the production of solar increases by one unit, the average price of energy per EUR/MWH increases by 2.158e-05 units. When the production of fossil fuel gas increases by one unit, the average price of energy per EUR/MWH increases by 7.805e-05 units. When the production of wind onshore increases by one unit, the average price of energy per EUR/MWH decreases by .0039 units. When the production of fossil hard coal increases by one unit, the price of energy per EUR/MWH increases by 4.3e-05 units. When the

production of hydro increases by one unit, the price of energy per EUR/MWH increases by 1.502e-05 units. When the production of other renewables increases by one unit, the price of energy per EUR/MWH decreases by 0.0219 units. When the production of fossil oil increases by one unit, the price of energy per EUR/MWH decreases by .0076 units. When the production of hydro power increases by one unit, the price of energy per EUR/MWH decreases by 1.502e-05 units. When the production of fossil brown coal increases by one unit, the price of energy per EUR/MWH increases by 0.0002 units.

Based off this regression analysis, the prices of energy per EUR/MWH is dependant to the outputs of wind power nonetheless.

```
In [ ]: import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

from sklearn import *
import statsmodels.api as sm

import statsmodels.formula.api as smf
polynomial_features = PolynomialFeatures(degree=2)
M=2
df_ResourcesNO_Y1 = df_ResourcesNO_Y
df_ResourcesNO_Y1 = sm.add_constant(df_ResourcesNO_Y1)

poly_features=PolynomialFeatures(degree=M, include_bias=False)
X_poly=poly_features.fit_transform(df_ResourcesNO_Y1)
pr_model=LinearRegression()
pr_model.fit(X_poly,y)

# Plot
X_plot=np.linspace(0,100,10000).reshape(-1, 1)
X_plot_poly=poly_features.fit_transform(X_plot)

plt.plot(df_ResourcesNO_Y['biomass'],Price_Actual,"r.")
plt.plot(df_ResourcesNO_Y['nuclear'],Price_Actual,"g.")
plt.plot(df_ResourcesNO_Y['Fossil_Hard_coal'],Price_Actual,'m.')
plt.plot(df_ResourcesNO_Y['fossil_oil'],Price_Actual, 'y.')
plt.plot(df_ResourcesNO_Y['solar'],Price_Actual, 'k.')
plt.plot(df_ResourcesNO_Y['hydro_water_reservoir'],Price_Actual,'c.')
plt.plot(df_ResourcesNO_Y["fossil_brown_coal"],Price_Actual,'b.')
plt.plot(df_ResourcesNO_Y['wind_onshore'],Price_Actual,"o",color = '0.2')
plt.plot(df_ResourcesNO_Y['fossil_fuel_gas'],Price_Actual,'o',color = '0.8')
plt.plot(df_ResourcesNO_Y['other_renewable'],Price_Actual,"o",color = '0.75')

y_pred1 = pr_model.predict(X_poly)

plt.suptitle("Multilpolynomial monthly average outputs of all resources versus average monthly predicted prices")
plt.title("R squared : 1.0")
plt.ylabel("Average monthly prices of energy per EUR/MWH")
plt.xlabel("Average monthly output")

x = np.linspace(0,10000,48)
plt.plot(x, pr_model.predict(X_poly))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of pandas all arguments of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x7ff6079fdd10>]
```



The blue line represents the polynomial calculated in the multiple polynomial regression. The dots within the graph represent the observations of each resource by color. These dots are represented as follows: biomass is red, nuclear is green, fossil hard coal is magenta, fossil oil is yellow, hydro is cyan, fossil brown coal is blue. For the abnormally represented resources; they are represented as follows: solar observations are the smaller black dots, wind observations are the larger black dots, fossil gas observations are the larger gray dots while other renewable representations are the smaller gray dots. Overall, there is an eminently positive correlation between the average monthly prices of energy per EUR/MWH and the average monthly outputs of each resource analyzed.

```
In [ ]: print(y_pred1)
```

```
[53.91400421 53.69853568 55.25736026 57.58585133 55.24119807 60.21860216
61.37499835 59.02491476 60.35471344 59.65244893 59.91315799 58.82163308
58.70578308 55.62563506 56.82488573 57.93038915 58.00308007 57.31423659
57.15830939 58.45669916 57.63104304 59.1846058 57.05744947 59.84708249
55.71543951 57.95667147 57.34553378 57.10666001 58.61781315 57.42036268
59.15940819 60.28943851 61.58354954 58.16509898 58.2410701 54.42393503
56.23185086 55.06103305 52.25398893 58.04659847 60.18136998 59.20733609
61.07591797 60.26296307 60.50886885 56.30694665 56.49331059 56.82093581]
```

```
In [ ]:
```

```
In [ ]: y = [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277]
```

```
X = [[6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068.16991643454, 5403.817204301075, 5659.276777
[1130.392905866303, 1244.5252976190477, 1283.479138627187, 1461.5194986072424, 1920.2728494623657, 1998.660
[4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.123955431754, 4415.3494623655915, 4934.9304589
[483.73533424283767, 470.1502976190476, 468.1063257065949, 426.32033426183847, 503.5698924731183, 485.9986091
[7587.697135061391, 7731.806547619048, 6747.878869448183, 5506.381615598886, 6757.408602150537, 4375.49513212
[5411.263301500682, 4045.974702380953, 4233.818304172274, 4819.516713091922, 4019.6129032258063, 6207.0959666
[70.66030013642565, 70.51488095238095, 66.63257065948856, 69.70055710306407, 70.56586021505376, 65.51182197490
[306.0204638472033, 319.239583333333, 319.3337819650067, 338.78133704735376, 332.56720430107526, 319.229485390
[2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.0125348189417, 2798.184139784946, 2531.16828929
[572.8512960436562, 313.41815476190476, 244.43741588156124, 463.11977715877435, 374.2809139784946, 665.16272600
def reg(y, x):
    ones = np.ones(len(x[0]))
    X = sm.add_constant(np.column_stack((x[0], ones)))
    for ele in x[1:]:
        X = sm.add_constant(np.column_stack((ele, X)))
    results = sm.OLS(y, X).fit()
    predictionsresults = results.predict(X)
    return results
```

```
In [ ]:
```

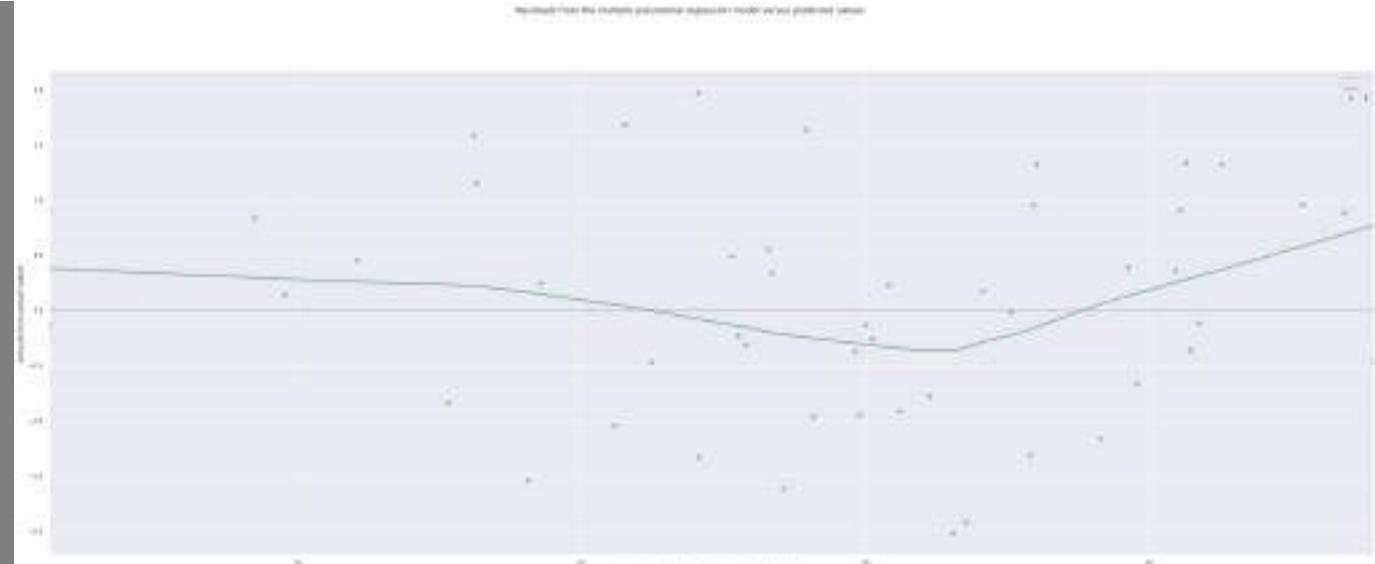
```
In [ ]:
```

```
In [ ]: #Multiple polynomial regression residual plot
```

```
sns.residplot(x=[y_pred1],y =[standardized_residuals],lowess = True, color="g")
plt.suptitle("Residuals from the multiple polynomial regression model versus predicted values")

plt.xlabel("Average monthly prices of energy per EUR/MWh")
plt.ylabel("Amount from actual values")
plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7ff608f95890>
```



As one can observe, there is a subtle hump along the lowess line in the residual plot. In addition, there residuals are quite spread out in no particular pattern which indicates constant variance, a lack of bias, and homoscedasticity. Given these circumstances, it would be reasonable to assume that the multiple regression model of fit is suitable. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: print(list(y_pred1))
```

```
[53.91400421360618, 53.69853567564433, 55.25736026311268, 57.585851330234476, 55.24119806745172, 60.218602162356795, 61.37499834987312, 59.02491475593839, 60.354713436173434, 59.65244892775017, 59.91315799396435, 58.82163307583801, 58.70578308184851, 55.62563506124494, 56.82488572693495, 57.93038915282896, 58.00308007258524, 57.31423658508532, 57.158309389384115, 58.456699160906645, 57.63104304180718, 59.18460579702193, 57.05744947498409, 59.84708249341074, 55.71543950812963, 57.956671471210484, 57.3455337770102, 57.1066001216424, 58.61781314762135, 57.42036267620668, 59.15940819087747, 60.289438511432344, 61.5835495352146, 58.165098981315566, 58.241070104073835, 54.42393503013379, 56.2318508532473, 55.061033054485655, 52.253988925645494, 58.046598472207315, 60.18136998356598, 59.20733608576309, 61.075917968783756, 60.26296306708013, 60.50886885129249, 56.306946654884975, 56.49331059383376, 56.82093581037206]
```

```
In [ ]: dfMultiPoly = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.3540833333333335, 57.2940591, "MultiPoly"] : [53.91400421360618, 53.69853567564433, 55.25736026311268, 57.585851330234476, 55.24119806745172, "MultiPoly"] : [53.91400421360618, 53.69853567564433, 55.25736026311268, 57.585851330234476, 55.24119806745172, print(dfMultiPoly) #Dataframes
df_MultiPoly= pd.DataFrame.from_dict(dfMultiPoly, orient = "columns")
print(df_MultiPoly)
```

```
#ADF Tests
from statsmodels.tsa.stattools import adfuller
def adfuller_test(test_result):
    result=adfuller(test_result)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary ")

adfuller_test(df_MultiPoly["MultiPoly"])

test_result=adfuller(df_MultiPoly["MultiPoly"])
```

```

df_MultiPoly['First Difference MultiPoly'] = df_MultiPoly["MultiPoly"] - df_MultiPoly["MultiPoly"].shift(1) # Seasonal Difference MultiPoly
df_MultiPoly['Seasonal Difference MultiPoly']=df_MultiPoly["MultiPoly"] - df_MultiPoly["MultiPoly"].shift(12)
df_MultiPoly.head()
plt.suptitle("Seasonal Difference of average monthly predicted multipolynomial prices of energy per EUR/MWh")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_MultiPoly['Seasonal Difference MultiPoly'].plot()
# Seasonality Plot

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.61866666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742, 50.4055972222222, 60.18242953020134, 62.58105555555556, 67.59513440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.71791666666666, 53.772620967741936, 56.2582222222222, 55.25258064516129, 54.08432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 50.40073611111111, 61.63376344086022, 64.3481388888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.6235138888889], 'MultiPoly': [53.91400421360618, 53.69853567564433, 55.25736026311268, 57.585851330234476, 55.24119806745172, 60.218602162356795, 61.37499834987312, 59.02491475593839, 60.354713436173434, 59.65244892775017, 59.91315799396435, 58.82163307583801, 58.70578308184851, 55.62563506124494, 56.82488572693495, 57.93038915282896, 58.00308007258524, 57.31423658508532, 57.158309389384115, 58.456699160906645, 57.63104304180718, 59.18460579702193, 57.05744947498409, 59.84708249341074, 55.71543950812963, 57.956671471210484, 57.34553377770102, 57.10666001216424, 58.61781314762135, 57.42036267620668, 59.15940819087747, 60.289438511432344, 61.5835495352146, 58.165098981315566, 58.241070104073835, 54.42393503013379, 56.23185085532473, 55.061033054485655, 52.253988925645494, 58.046598472207315, 60.18136998356598, 59.20733608576309, 61.075917968783756, 60.26296306708013, 60.50886885129249, 56.306946654884975, 56.49331059383376, 56.82093581037206]}

	Price	MultiPoly
0	64.949019	53.914004
1	56.383854	53.698536
2	55.522463	55.257360
3	58.354083	57.585851
4	57.294059	55.241198
5	65.974903	60.218602
6	71.072043	61.374998
7	63.998065	59.024915
8	60.254792	60.354713
9	59.406765	59.652449
10	60.726792	59.913158
11	61.901761	58.821633
12	45.578723	58.705783
13	36.752083	55.625635
14	36.818008	56.824886
15	32.618667	57.930389
16	34.691371	58.003080
17	46.266319	57.314237
18	47.502016	57.158309
19	47.602339	58.456699
20	50.405597	57.631043
21	60.182430	59.184606
22	62.581056	57.057449
23	67.595134	59.847082
24	79.492083	55.715440
25	59.837798	57.956671
26	50.959892	57.345534
27	51.717917	57.106660
28	53.772621	58.617813
29	56.258222	57.420363
30	55.252581	59.159408
31	54.084328	60.289439
32	55.816556	61.583550
33	63.925289	58.165099
34	65.430653	58.241070
35	65.151277	54.423935
36	56.511976	56.231851
37	60.877098	55.061033
38	48.279717	52.253989
39	50.400736	58.046598
40	61.633763	60.181370
41	64.348139	59.207336
42	67.783441	61.075918
43	70.363911	60.262963
44	76.914042	60.508869
45	70.362215	56.306947
46	67.042608	56.493311
47	66.623514	56.820936

ADF Test Statistic : -5.249902069926404

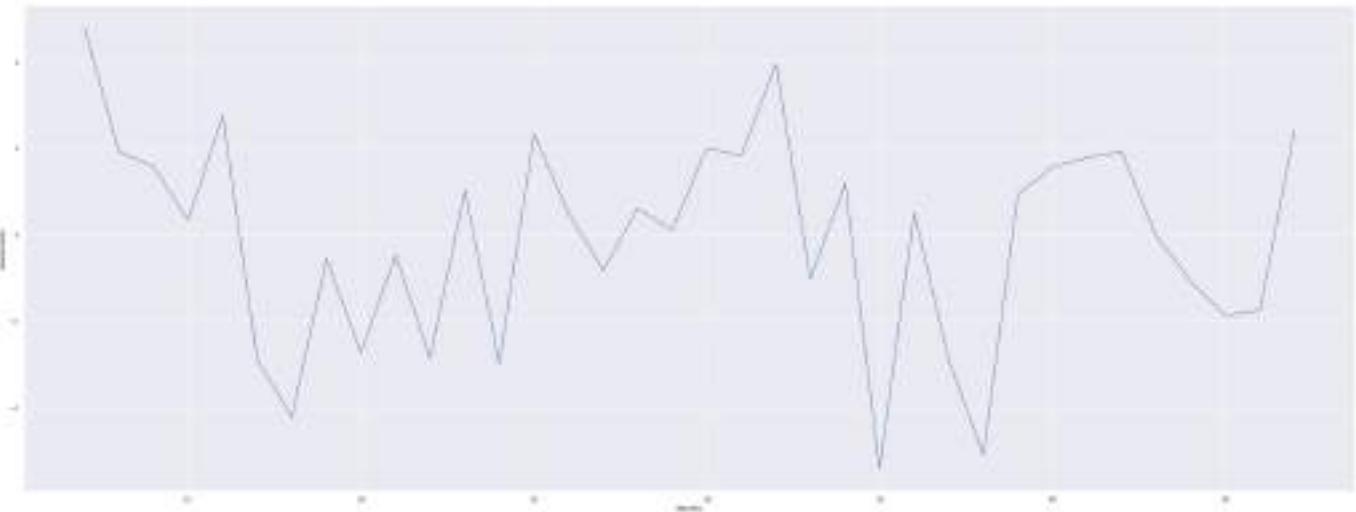
p-value : 6.947656843781993e-06

#Lags Used : 5

Number of Observations : 42

strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is stationary

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff607aebb50>



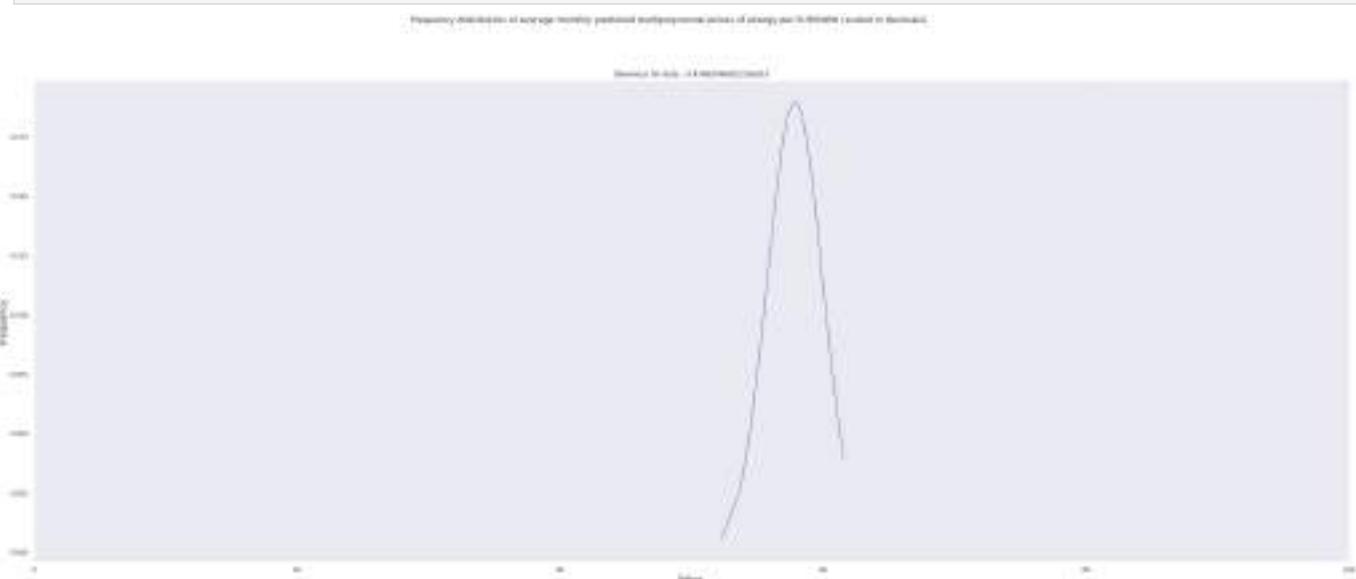
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the months and the average monthly prices of energy per EUR/MWH.

In []: #Bell Curves

```
MultiPolyResults_mean = np.mean(df_MultiPoly["MultiPoly"])
MultiPolyResults_std = np.std(df_MultiPoly["MultiPoly"])

MultiPolyResultspdf = stats.norm.pdf(df_MultiPoly["MultiPoly"].sort_values(), MultiPolyResults_mean, MultiPolyResults_std)

plt.plot(df_MultiPoly["MultiPoly"].sort_values(), MultiPolyResultspdf)
plt.xlim([0,100])
plt.xlabel("Value ", size=15)
plt.title(f'Skewness for data: {skew(df_MultiPoly["MultiPoly"])}')
plt.suptitle("Frequency distribution of average monthly predicted multipolynomial prices of energy per EUR/MWH")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```



This bell shaped curve is roughly symmetric. Hence, it has a normal distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

In []:

```
print(dicDates)
MultiPoly_Dict = {key: i for i, key in enumerate(df_MultiPoly["MultiPoly"])}

def Hist_MultiPoly(MultiPoly_Dict):
    for k, v in MultiPoly_Dict.items(): print(f"{v}:{k}")
print(MultiPoly_Dict)
```

```

plt.bar(list(MultiPoly_Dict.values()), MultiPoly_Dict.keys(), color='g')
plt.title("Average monthly predicted multipolynomial prices of energy per EUR/MWh")
plt.ylabel('Average monthly output')
plt.xlabel('Months')

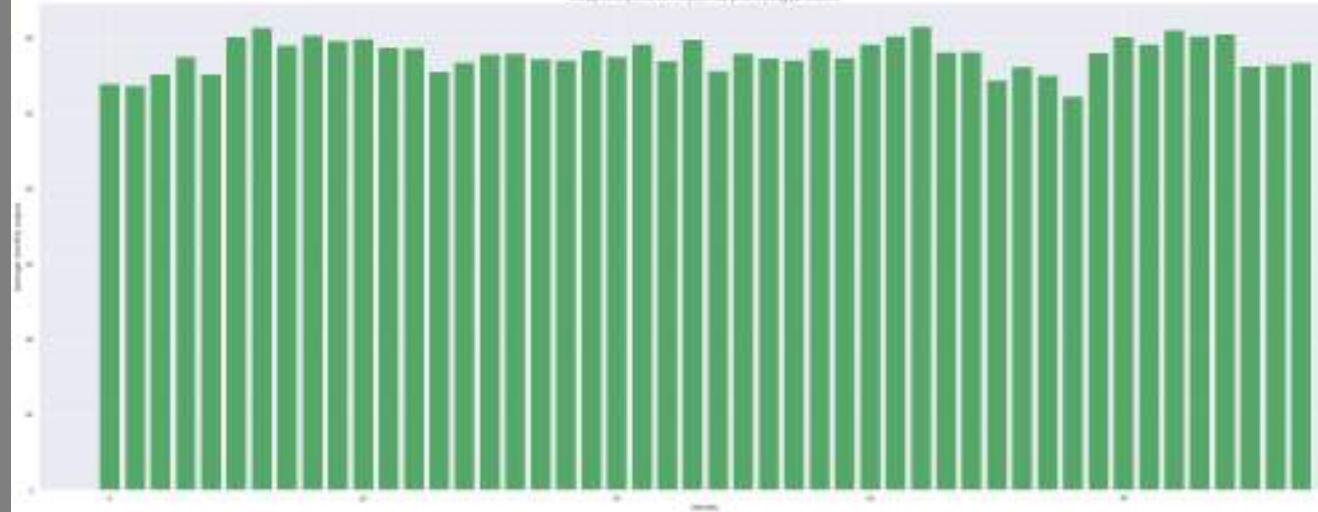
plt.show()

```

```

{'15-01': 1, '15-02': 2, '15-03': 3, '15-04': 4, '15-05': 5, '15-06': 6, '15-07': 7, '15-08': 8, '15-09': 9, '15-10': 10, '15-11': 11, '15-12': 12, '16-01': 13, '16-02': 14, '16-03': 15, '16-04': 16, '16-05': 17, '16-06': 18, '16-07': 19, '16-08': 20, '16-09': 21, '16-10': 22, '16-11': 23, '16-12': 24, '17-01': 25, '17-02': 26, '17-03': 27, '17-04': 28, '17-05': 29, '17-06': 30, '17-07': 31, '17-08': 32, '17-09': 33, '17-10': 34, '17-11': 35, '17-12': 36, '18-01': 37, '18-02': 38, '18-03': 39, '18-04': 40, '18-05': 41, '18-06': 42, '18-07': 43, '18-08': 44, '18-09': 45, '18-10': 46, '18-11': 47, '18-12': 48}
{53.91400421360618: 0, 53.69853567564433: 1, 55.25736026311268: 2, 57.585851330234476: 3, 55.24119806745172: 4, 60.218602162356795: 5, 61.37499834987312: 6, 59.02491475593839: 7, 60.354713436173434: 8, 59.65244892775017: 9, 59.91315799396435: 10, 58.82163307583801: 11, 58.70578308184851: 12, 55.62563506124494: 13, 56.82488572693495: 14, 57.93038915282896: 15, 58.00308007258524: 16, 57.31423658508532: 17, 57.158309389384115: 18, 58.456699160906645: 19, 57.63104304180718: 20, 59.18460579702193: 21, 57.05744947498409: 22, 59.84708249341074: 23, 55.71543950812963: 24, 57.956671471210484: 25, 57.34553377770102: 26, 57.10666001216424: 27, 58.61781314762135: 28, 57.42036267620668: 29, 59.15940819087747: 30, 60.289438511432344: 31, 61.5835495352146: 32, 58.165098981315566: 33, 58.241070104073835: 34, 54.42393503013379: 35, 56.23185085532473: 36, 55.061033054485655: 37, 52.253988925645494: 38, 58.046598472207315: 39, 60.18136998356598: 40, 59.20733608576309: 41, 61.075917968783756: 42, 60.26296306708013: 43, 60.50886885129249: 44, 56.306946654884975: 45, 56.49331059383376: 46, 56.82093581037206: 47}

```



In []: Price_Actual = [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335, 57.29405913978494, 0]

In []:

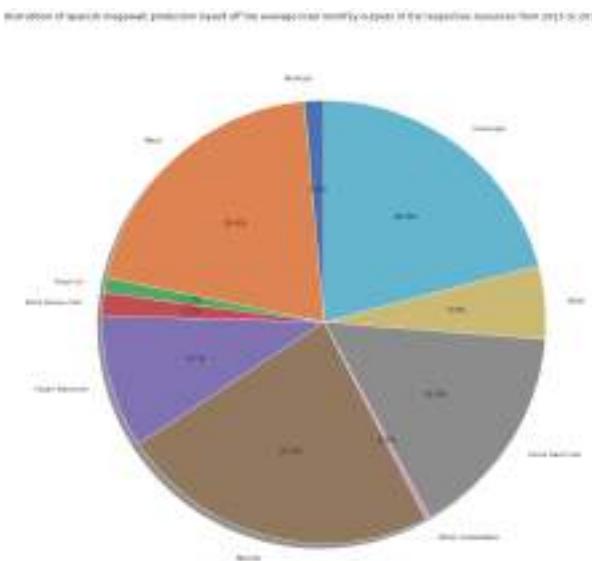
The green bars represent the observation value for each respective month.

```

In [ ]: data = [383.548775,5469.944902,298.324069,447.860317,2608.982390,6264.260822,85.633141,4255.364629,1431.838433,!label = ['Biomass', 'Wind', 'Fossil oil', 'fossil brown coal', 'Hydro Reservoir', 'Nuclear', 'Other renewables', 'Solar power']

plt.pie(data, labels=label, autopct='%1.1f%%', explode=[0,0,0,0,0,0,0,0], shadow=True, startangle=90)
plt.suptitle('Distribution of Spanish megawatt production based off the average total monthly outputs of the respective sources from 2013-16-07-20')
plt.axis('equal')
plt.show()
#Pie Chart

```



Based off of this pie chart, nuclear output is the most productive resource since it produced 23.3% of Spains energy while other renewables were the least productive resource since they produced only 0.3% of Spains energy.

In []: #Conclusion

Concluding thoughts:

The simple fact that the multipolynomial model yielded a R squared of a 1 out of |1| meant that it is propostrous to argue against output yields affecting the price of energy per EUR/MWH collectively. Nevertheless, I wanted to investigate wheter or not if outputs of each resource respectively affected the price of energy per EUR/MWh singlehandedly. While the volatily of the yielded R squares per resource was apparent; it was enough to understand that there were other variables imapcting the outputs and prices of energy per EUR/MWH; collectively or individually.However, it is worth mentioning that some of the R squared values from the OLS regressions conducted individually yielded some notable values; the highest being from the quadratic hydro reservoir model which yielded an R squared value of 0.498. This would imply that the price of energy per EUR/MWH and hydro power outputs have a moderately strong and positive correlation.

While there are several variables to consider when evaluating time series relationships since elasticity is involved; it is always worth noting that by honing down on the scale of time, more precision will be yielded from our analytics due to the fact that seasonality has been discovered in this regression analysis. As such, a weekly average analysis rather than a monthly average analysis will be conducted for each respective season from the origonal dataframe.

By investigating energy outputs and prices of energy per EUR/MWH on a weekly scale for each season of the year, predictions for the upcoming season can be timed in a fashionable manner. But why would Spanish economists ponder about such matters? Some important variables taken into considiration when investigating energy outputs and prices of energy per EUR/MWH are inflation, the weather, and the population of Spain. By using the mentioned variables from the previous sentance in predicting energy prices, outputs can be managed accordingly so that the target price of energy per EUR/MWH can be as close to the market equilibrium as it possibly can.

In []: :

In []: ::::

In []: ::

In []: ::::::::::::::::::::

In []: :

In []: ::::::::::::::::::::

In []: ::::::::::::::::::::

In []: :

In []: :

In []: :

In []: ::::::::::::

In []: :

In []: :

In []: :

In []: ::

In []: ::::::::::::

In []: :

In []: ::::::::::::::::::::

In []: ::::::::::::::::::::

In []: ::::::::::::::::::::

In []: :

In []: :*****

In []: :}

In []: :

In []: :

In []: :::

In []: :

In []: ::

In []: ::::

In []: :

In []: ::}}

In []: ::}>})

In []: :

In []: :::::

In []: ::

In []: ::::::

In []: :

In []: :

In []: ::

In []: ::/****

In []: :*****

In []: ::

In []: :

In []: :

In []: ::

In []: :::

In []: :

In []: :

In []: ::::}

In []: :

In []: ::

In []: ::}>::?****

In []: ::

In []: ::

In []: ?/*****

In []: ::

In []: :

In []: ::::::::::::::} /'?:

In []: "?"

In []: :

In []: ::::::::::::::

In []: :::

In []: :::

In []: ::::::::::::::

In []: ::

In []: :::/*****

In []: ::

In []: :>}>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In []: ::::::::::::::::::::::::::::::::

In []: ::

In []: ::

In []: ::}

In []: ::

In []: ::

In []: ::

In []: ::

In []: ::}

In []: ::}

In []: ::

In []: ::

In []: ::

In []: ::

In []: ::>} }>}:::}

In []: ::::

In []: ?*****

In []: ::::::::::::::

In []: :::

In []: ::::::::::::::::::::

In []: :::

In []: ::

In []: ::::::::::::::::::::?*****

In []: ::

In []: :}

In []: :

In []: :

In []: :

In []: :>}:

In []: :}

In []: :

In []: :}

In []: :}

In []: ::::

In []: :

In []: ::

In []: ::::::::::

In []: :

In []: :

In []: :}

In []: :

In []: :}

In []: ::::::

In []: ::::::::

In []: ::

In []: ::::

In []: ::}

In []: ::

In []: :

In []: :

In []: :}

In []: :

In []: :

In []: ::

In []: :::

In []: ::

In []: ::}

In []: ::

In []: :

In []: ::

In []: :>}))))))))))))))))))))))>}

In []: :::

In []: :

In []: :

In []: :}

In []: :

In []: :

In []: :}

In []: :}

In []: :

In []: :}

In []: :}

In []: :}:

In []: :}

In []: ::?"/"/******

In []: :

In []: :>}

In []: :

In []: :

In []: :

In []: ::

In []: ::}

In []: :

In []: :

In []: :

In []: :

```
In [ ]: :  
In [ ]: :}  
In [ ]: ""  
In [ ]: :  
In [ ]: ""  
In [ ]: :  
In [ ]: :}  
In [ ]: :}  
In [ ]: ::}  
In [ ]: :}  
In [ ]: :}  
In [ ]: :}  
In [ ]: ::}  
In [ ]: :}  
In [ ]: :>}  
In [ ]: :  
In [ ]: :>}*****//*****  
In [ ]: ??"?/"///*****/*  
In [ ]: :  
In [ ]: *****  
In [ ]: :::::::::::::::>}*****  
In [ ]: :  
In [ ]: :  
In [ ]: :
```

```
In [ ]: :  
In [ ]: :}:  
In [ ]: :>}  
In [ ]: :}>:  
In [ ]: //////////////////////////////////////////////////////////////////  
In [ ]: ::/  
In [ ]: :  
In [ ]: :://///////////////////////////////////////////////////////////////  
In [ ]: ::  
In [ ]: ?  
In [ ]: }:::
```

Loading [MathJax]/extensions/Safe.js