

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

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

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, could lower energy prices as well as appropriately decrease the output amount 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. 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.

```
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
```

```
In [ ]: import matplotlib
```

```
In [ ]:
```

Here 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.

```
In [ ]: #Linear OLS regression
```

```
In [ ]: #OLS Linear Summary Table
```

```
In [ ]: #Linear OLS regression standard residuals
```

```
In [ ]: #Multiple regression residual plot
```

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

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

```
In [ ]: #Pie Chart
```

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals
```

```
In [ ]: # Stem Plot
```

```
In [ ]: #OLS logarithmic predicted average monthly ratios versus actual ratios
```

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

```
In [ ]: #Predicted OLS average monthly quadratic
```

```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals
```

```
In [ ]: #Predicted average monthly OLS quadratic price of energy per EUR/MWH versus
```

```
In [ ]: #OLS predicted quadratic average monthly ratios versus actual values
```

```
In [ ]: # OLS predicted linear average monthly ratios versus actual ratios
```

```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
```

```
In [ ]: #Rounded Price
```

```
In [ ]: #Multiple OLS regression
In [ ]: # OLS Predicted Linear Regession ratio versus actual ratios
In [ ]: #ADF Tests
In [ ]: #Autocorrelation Plot
In [ ]: #Partialautocorrelation Plot
In [ ]: #Seasonality values
In [ ]: #Bell Curves
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,
In [ ]: # Average monthly values
In [ ]: #Autocorrelations
In [ ]: #Outilers
In [ ]: #Quadratic Scatterplots
In [ ]: #Log
In [ ]: #Quadratic Summary Table:
In [ ]: # Box and Whisker Plot
In [ ]: #Quadratic OLS regression
In [ ]: #OLS linear scatterplot
In [ ]: #slope and intercept for OLS linear regression
In [ ]: #Dataframes analyzed by resource
In [ ]: #Logarithmic OLS regressions
In [ ]: #Logarithmic OLS regression scatterplot
```

```
In [ ]: #Logarithmic OLS regression residuals
```

```
In [ ]: #OLS Logarithmic summary table
```

```
In [ ]: # Histograms
```

```
In [ ]: #Datasets
```

```
In [ ]: #Column Values
```

As one can see, there have been many plug ins downloaded for this regressive analysis. Below is the dataset used for this regressive 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	gen fossil
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	4
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	4
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	4
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	4
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	3
...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	2
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	2
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	2
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	2
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	2

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	genel fossil
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	2
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	2
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	2
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	2
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	2
...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	2
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	2
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	2
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	2
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	2

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.

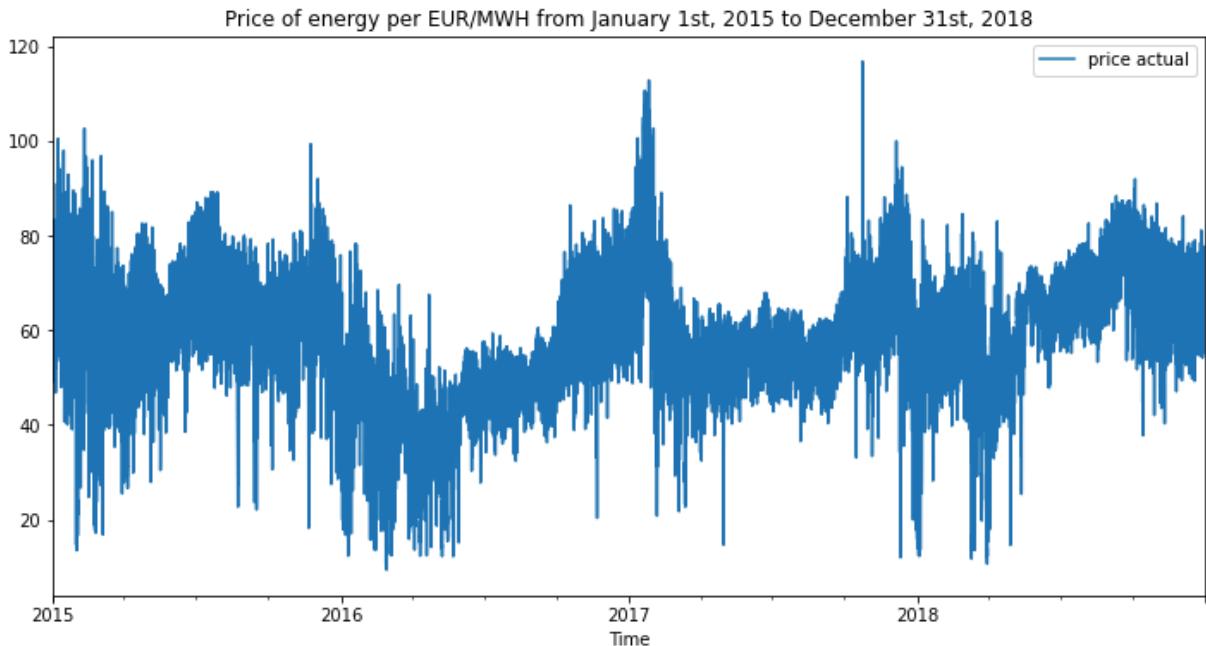
```
In [ ]: print(df3.columns.values) #Column Values
```

```
['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,  
 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')
```

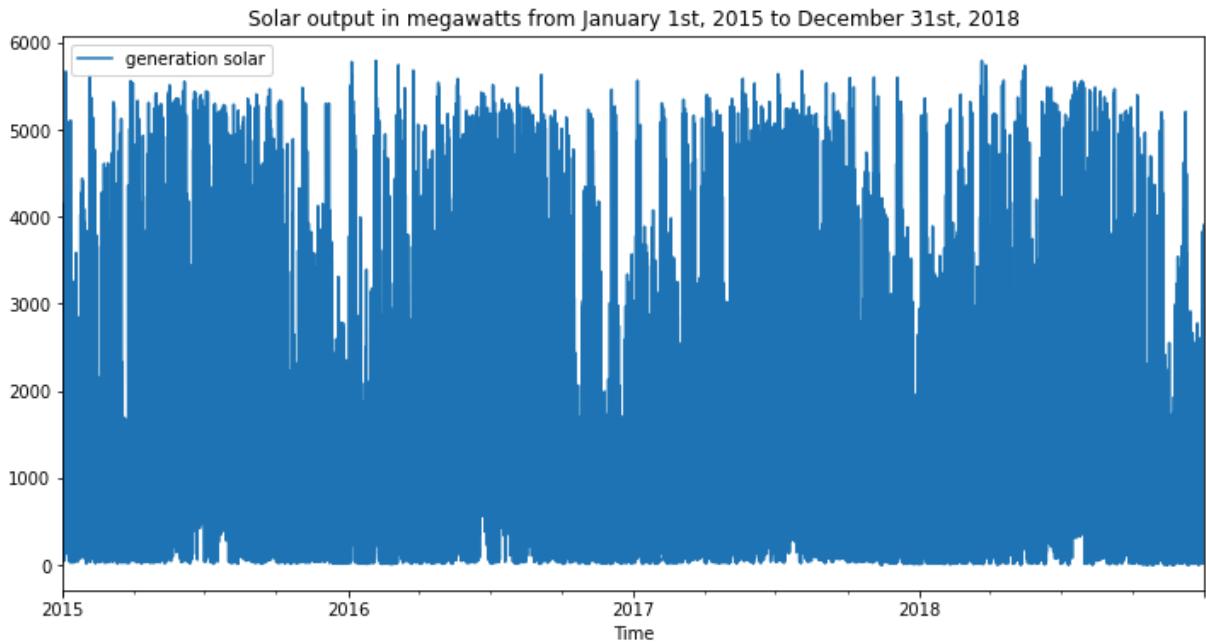


This line graph displays the relationship between the time and the price of energy per EUR/MWH. As one can observe, there were periods of gradual decline in price from 2015 to 2016. From 2016 to 2017, there was a gradual increase which was followed by a sharp increase in the price. Afterwards, there was a sharp decline in the price, followed by a period of consolidation until the end of

2017, which experienced a sharp increase. From 2018, there were only a few sharp price declines; but the rest of the year was roughly consistent.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st,
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')
```

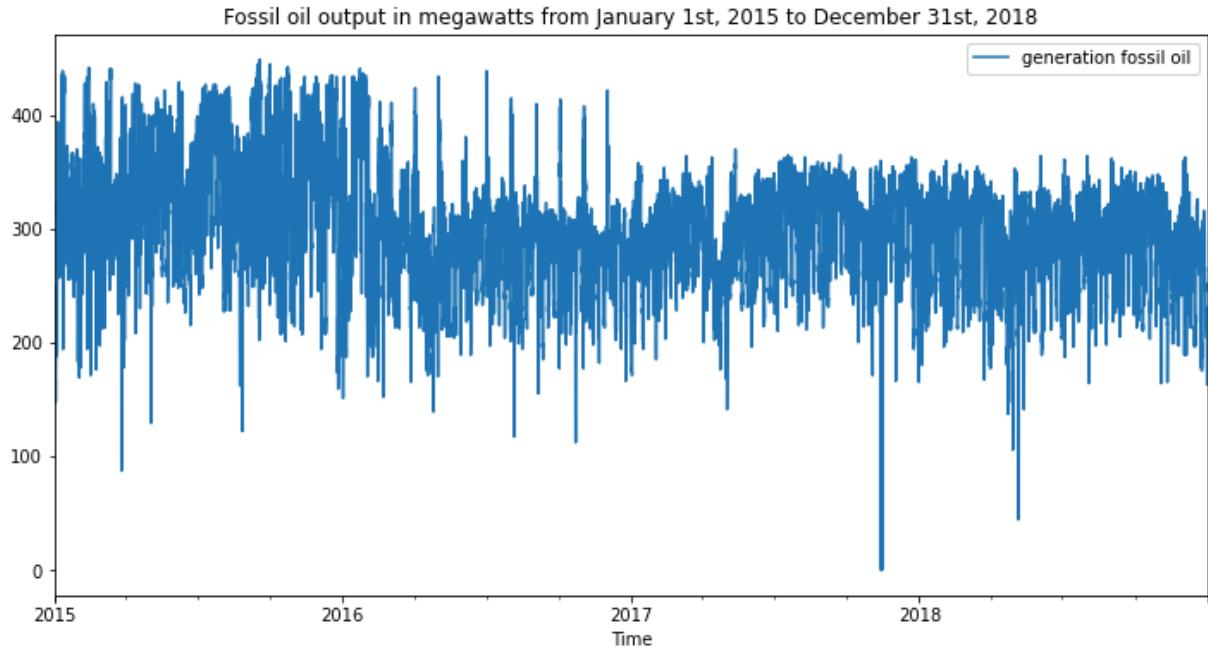
```
Out[ ]: Text(0.5, 1.0, 'Solar output in megawatts from January 1st, 2015 to December 31st, 2018')
```



Within this line graph, one would be able to notice a sharp drop in the output amount at the beginning of every year observed on this graph. These drops would be followed by a gradual but steep increase in solar output. It would be reasonable to assume that these sharp drops are due to the wintertime, as Spain, like the rest of the countries within the Northern Hemisphere experience, shorter daytime hours, resulting in less sunlight for production.

```
In [ ]: #Observed values on an hourly scale from January 1st, 2015 to December 31st,
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')
```

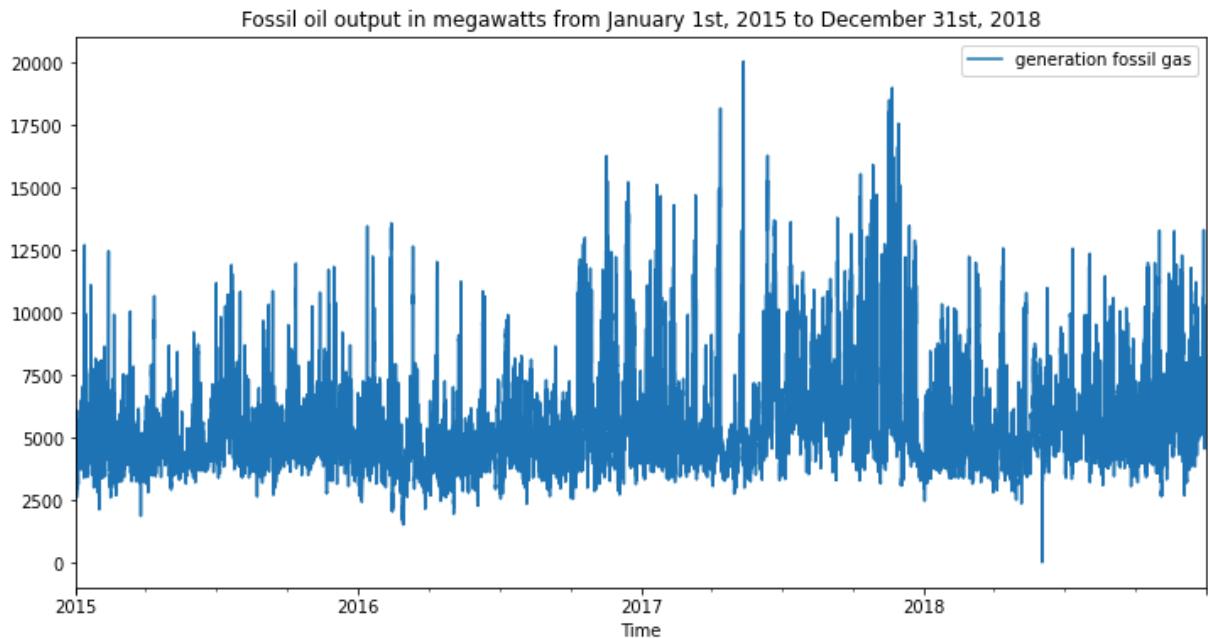
```
Out[ ]: Text(0.5, 1.0, 'Fossil oil output in megawatts from January 1st, 2015 to December 31st, 2018')
```



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,
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 Decem
```

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



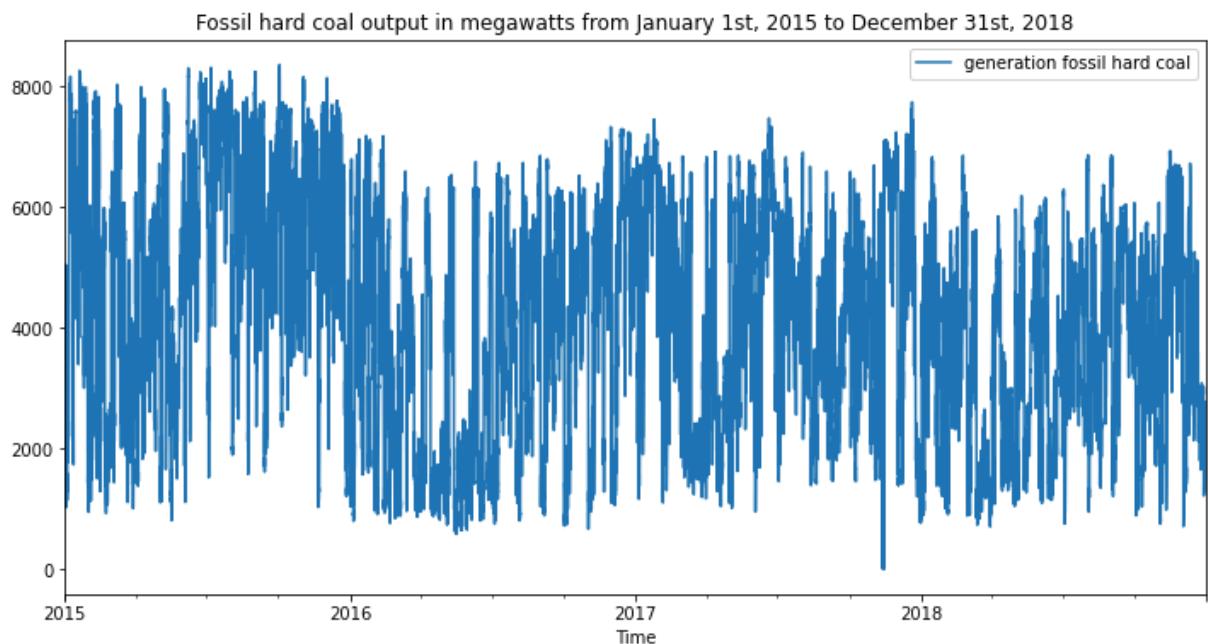
Based off of the constant yet consistent output spikes in the production of fossil oil, it would be reasonable to assume that the production yield was continuously

discrete, meaning that the output spikes, which were always followed by periods consolidation, were a frequent and regular occurrence.

In []:

```
#Observed values on an hourly scale from January 1st, 2015 to December 31st, 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
```

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

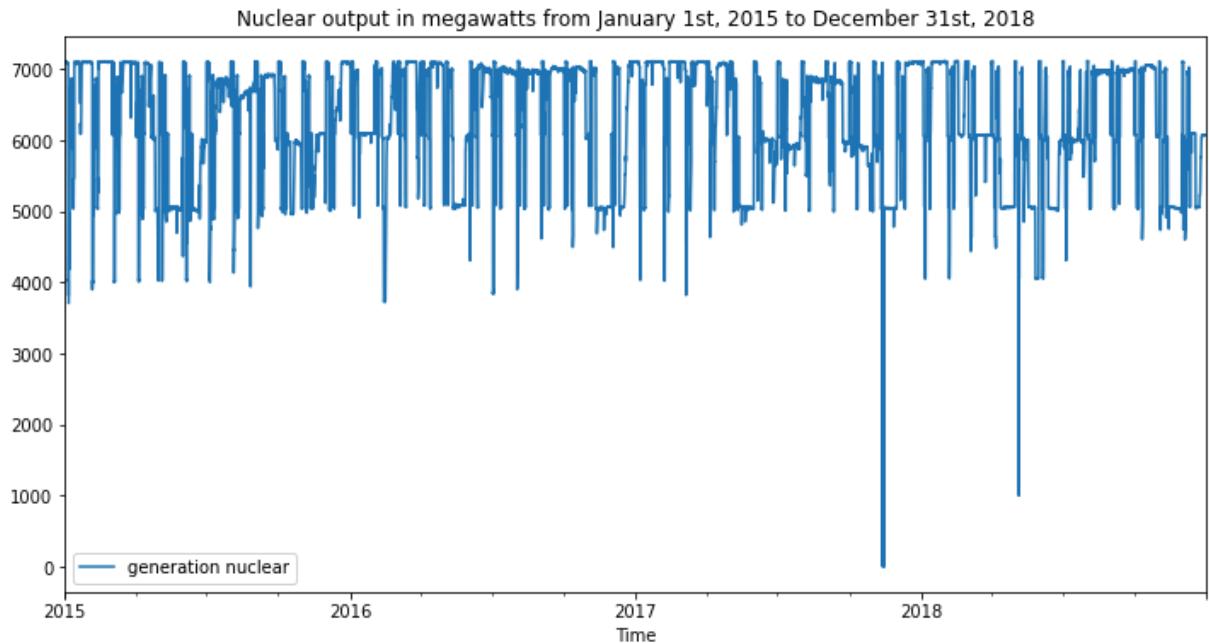


Based on this graph, the production output of fossil hard coal was a consistent roller coaster. There were no unusual output spikes or sharp drops, but production output fluctuated throughout the years.

In []:

```
#Observed values on an hourly scale from January 1st, 2015 to December 31st, 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
```

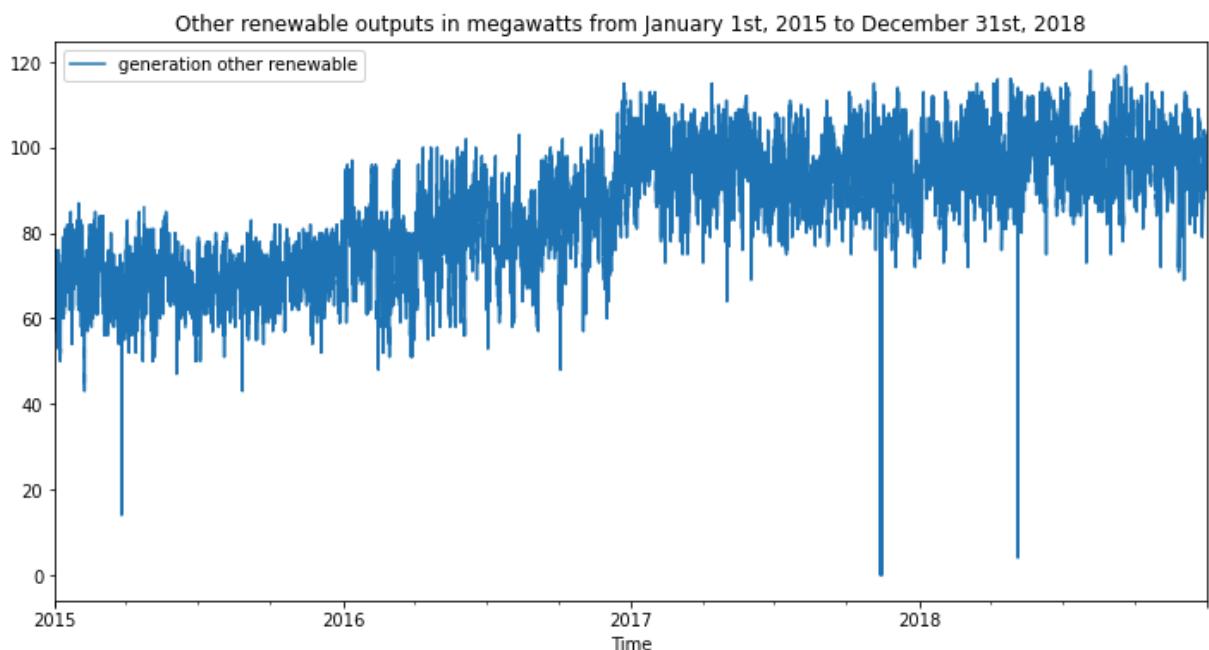
Out[]: Text(0.5, 1.0, 'Nuclear output in megawatts from January 1st, 2015 to December 31st, 2018')



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')
```

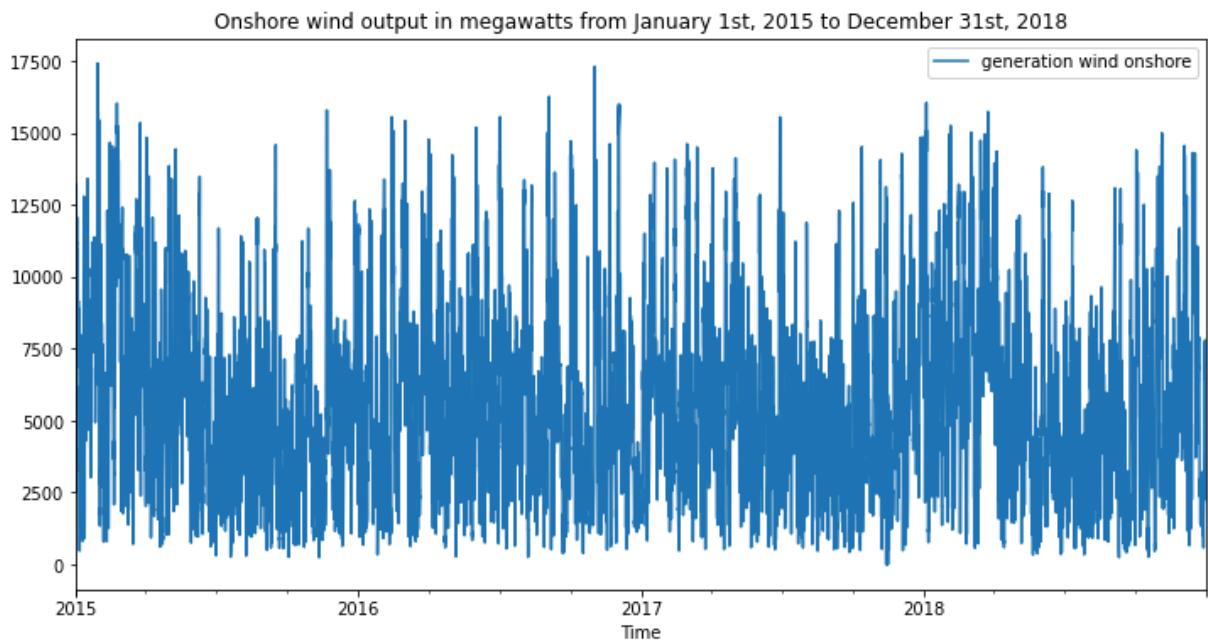
```
Out[ ]: Text(0.5, 1.0, 'Other renewable outputs in megawatts from January 1st, 2015 to December 31st, 2018')
```



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,  
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 Dec
```

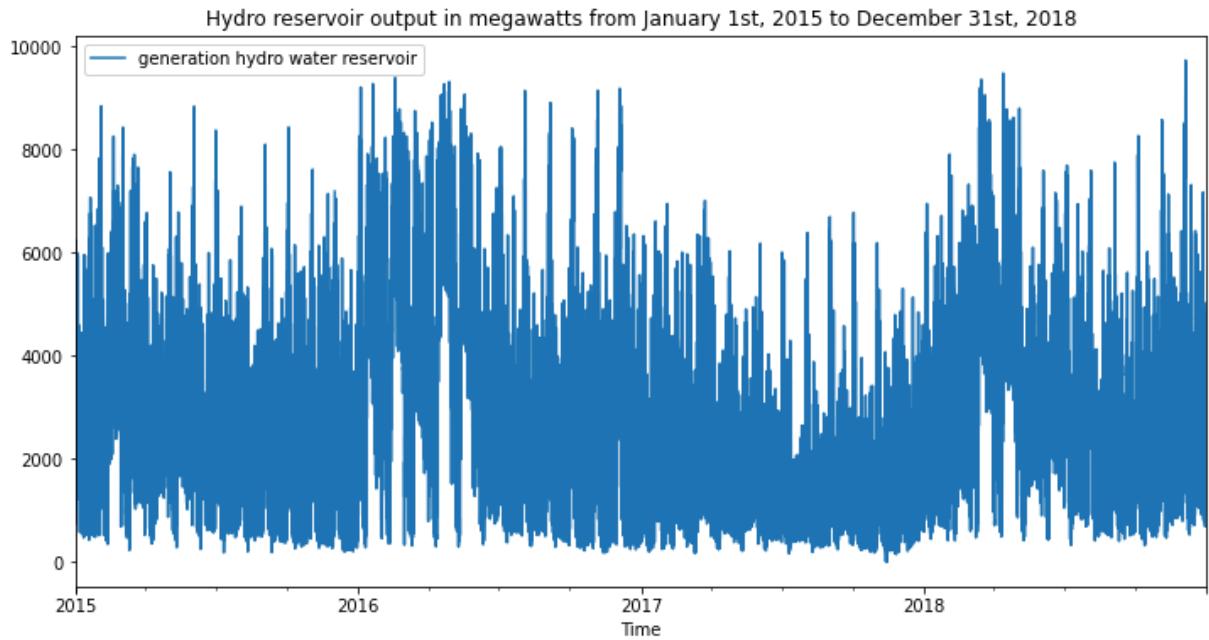
```
Out[ ]: Text(0.5, 1.0, 'Onshore wind output in megawatts from January 1st, 2015 to  
December 31st, 2018')
```



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,  
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
```

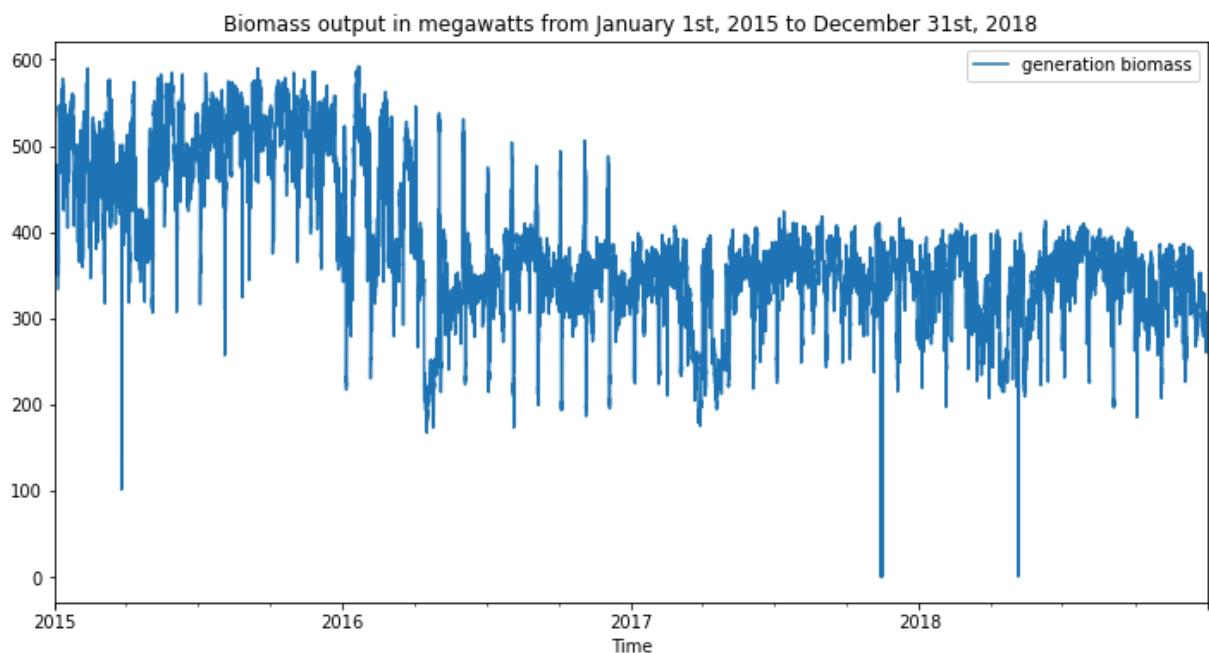
```
Out[ ]: Text(0.5, 1.0, 'Hydro reservoir output in megawatts from January 1st, 2015  
to December 31st, 2018')
```



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,
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')
```

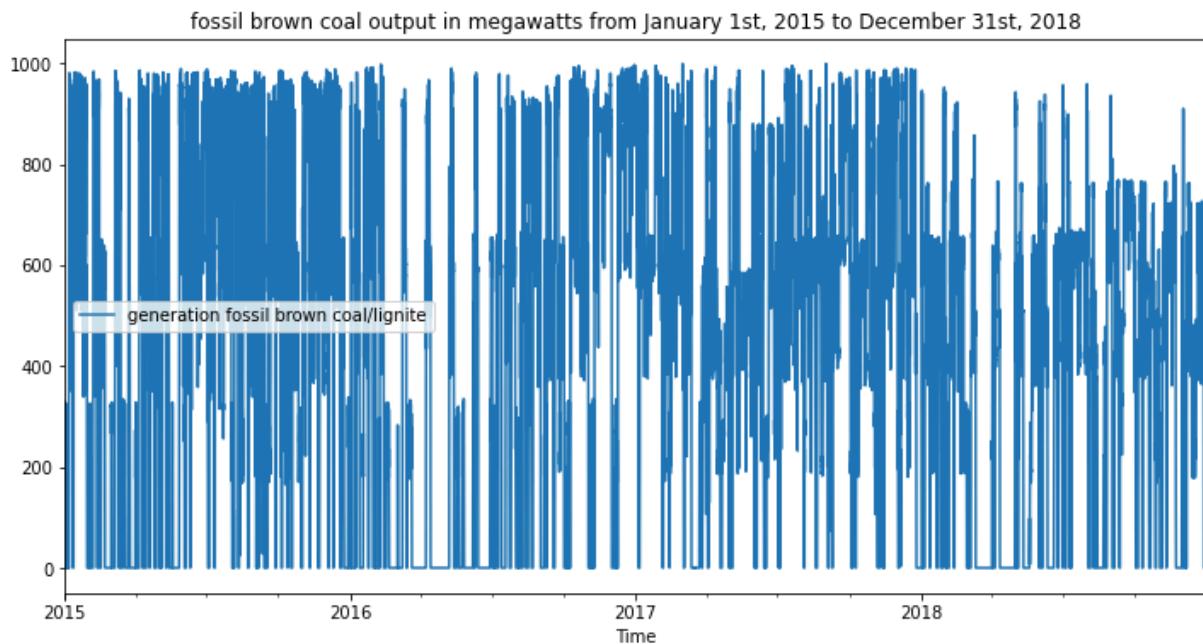
```
Out[ ]: Text(0.5, 1.0, 'Biomass output in megawatts from January 1st, 2015 to December 31st, 2018')
```



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, df3['Time'] = pd.to_datetime(df3['time'], utc=True) ax = df3.plot(x='Time', y='generation fossil brown coal/lignite', figsize=(1 ax.set_title('fossil brown coal output in megawatts from January 1st, 2015 t
```

```
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

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

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")

```

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	gener fossil
26304	2018-01-01 00:00:00+01:00	282.0	0.0	0.0	3471.0	1
26305	2018-01-01 01:00:00+01:00	275.0	0.0	0.0	3269.0	1
26306	2018-01-01 02:00:00+01:00	278.0	0.0	0.0	3541.0	1
26307	2018-01-01 03:00:00+01:00	278.0	0.0	0.0	3450.0	1
26308	2018-01-01 04:00:00+01:00	279.0	0.0	0.0	3318.0	1
...
27043	2018-01-31 19:00:00+01:00	387.0	936.0	0.0	10082.0	2
27044	2018-01-31 20:00:00+01:00	380.0	918.0	0.0	10207.0	2
27045	2018-01-31 21:00:00+01:00	382.0	942.0	0.0	9671.0	2
27046	2018-01-31 22:00:00+01:00	375.0	938.0	0.0	8773.0	2
27047	2018-01-31 23:00:00+01:00	374.0	816.0	0.0	8081.0	2

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	genera fossil l
8760	2016-01-01 00:00:00+01:00	410.0	0.0	0.0	4610.0	25
8761	2016-01-01 01:00:00+01:00	404.0	0.0	0.0	4434.0	18
8762	2016-01-01 02:00:00+01:00	395.0	0.0	0.0	4286.0	14
8763	2016-01-01 03:00:00+01:00	391.0	0.0	0.0	3813.0	11
8764	2016-01-01 04:00:00+01:00	384.0	0.0	0.0	3872.0	10
...
9499	2016-01-31 19:00:00+01:00	514.0	0.0	0.0	3532.0	25
9500	2016-01-31 20:00:00+01:00	491.0	0.0	0.0	3725.0	26
9501	2016-01-31 21:00:00+01:00	486.0	0.0	0.0	3926.0	26
9502	2016-01-31 22:00:00+01:00	484.0	0.0	0.0	4158.0	26
9503	2016-01-31 23:00:00+01:00	497.0	0.0	0.0	3826.0	25

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	459.995968	417.927419	0.0	5271.119624	4204.615591	337.46
std	88.130508	394.067088	0.0	1952.831382	1913.785542	67.96
min	217.000000	0.000000	0.0	2417.000000	791.000000	151.00
25%	382.000000	0.000000	0.0	3932.000000	2378.000000	296.00
50%	487.000000	454.000000	0.0	4760.000000	4515.500000	321.00
75%	522.000000	840.250000	0.0	6130.500000	5852.000000	407.25
max	592.000000	992.000000	0.0	13444.000000	7183.000000	441.00

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	generat fossil ha
0	2015-01-01 00:00:00+01:00	447.0	329.0	0.0	4844.0	482
1	2015-01-01 01:00:00+01:00	449.0	328.0	0.0	5196.0	475
2	2015-01-01 02:00:00+01:00	448.0	323.0	0.0	4857.0	458
3	2015-01-01 03:00:00+01:00	438.0	254.0	0.0	4314.0	413
4	2015-01-01 04:00:00+01:00	428.0	187.0	0.0	4130.0	384
...
739	2015-01-31 19:00:00+01:00	373.0	0.0	0.0	3922.0	132
740	2015-01-31 20:00:00+01:00	369.0	0.0	0.0	4206.0	137
741	2015-01-31 21:00:00+01:00	369.0	0.0	0.0	4354.0	136
742	2015-01-31 22:00:00+01:00	365.0	0.0	0.0	4007.0	135
743	2015-01-31 23:00:00+01:00	366.0	0.0	0.0	3189.0	133

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	genera fossil
8760	2016-01-01 00:00:00+01:00	410.0	0.0	0.0	4610.0	25
8761	2016-01-01 01:00:00+01:00	404.0	0.0	0.0	4434.0	18
8762	2016-01-01 02:00:00+01:00	395.0	0.0	0.0	4286.0	14
8763	2016-01-01 03:00:00+01:00	391.0	0.0	0.0	3813.0	11
8764	2016-01-01 04:00:00+01:00	384.0	0.0	0.0	3872.0	10
...
9499	2016-01-31 19:00:00+01:00	514.0	0.0	0.0	3532.0	25
9500	2016-01-31 20:00:00+01:00	491.0	0.0	0.0	3725.0	26
9501	2016-01-31 21:00:00+01:00	486.0	0.0	0.0	3926.0	26
9502	2016-01-31 22:00:00+01:00	484.0	0.0	0.0	4158.0	26
9503	2016-01-31 23:00:00+01:00	497.0	0.0	0.0	3826.0	25

744 rows × 29 columns

In []:

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	459.995968	417.927419	0.0	5271.119624	4204.615591	337.46
std	88.130508	394.067088	0.0	1952.831382	1913.785542	67.96
min	217.000000	0.000000	0.0	2417.000000	791.000000	151.00
25%	382.000000	0.000000	0.0	3932.000000	2378.000000	296.00
50%	487.000000	454.000000	0.0	4760.000000	4515.500000	321.00
75%	522.000000	840.250000	0.0	6130.500000	5852.000000	407.25
max	592.000000	992.000000	0.0	13444.000000	7183.000000	441.00

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	gener fossil
17544	2017-01-01 00:00:00+01:00	341.0	901.0	0.0	5412.0	€
17545	2017-01-01 01:00:00+01:00	338.0	900.0	0.0	5401.0	€
17546	2017-01-01 02:00:00+01:00	337.0	908.0	0.0	4753.0	€
17547	2017-01-01 03:00:00+01:00	335.0	915.0	0.0	4321.0	€
17548	2017-01-01 04:00:00+01:00	336.0	904.0	0.0	4320.0	€
...
18283	2017-01-31 19:00:00+01:00	376.0	644.0	0.0	10418.0	€
18284	2017-01-31 20:00:00+01:00	374.0	656.0	0.0	10127.0	€
18285	2017-01-31 21:00:00+01:00	376.0	590.0	0.0	8080.0	€
18286	2017-01-31 22:00:00+01:00	378.0	558.0	0.0	7145.0	€
18287	2017-01-31 23:00:00+01:00	371.0	591.0	0.0	6118.0	€

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	347.283602	688.645161	0.0	6412.591398	5533.782258	279.52
std	35.523525	230.923176	0.0	2548.380276	1632.328442	41.54
min	224.000000	0.000000	0.0	3323.000000	1157.000000	171.00
25%	338.000000	595.000000	0.0	4415.000000	4983.500000	256.75
50%	358.000000	649.000000	0.0	5552.000000	6157.500000	291.00
75%	366.000000	895.250000	0.0	8030.750000	6690.250000	308.00
max	399.000000	995.000000	0.0	15092.000000	7452.000000	358.00

8 rows × 28 columns

In []: 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	gener fossil
26304	2018-01-01 00:00:00+01:00	282.0	0.0	0.0	3471.0	1
26305	2018-01-01 01:00:00+01:00	275.0	0.0	0.0	3269.0	1
26306	2018-01-01 02:00:00+01:00	278.0	0.0	0.0	3541.0	1
26307	2018-01-01 03:00:00+01:00	278.0	0.0	0.0	3450.0	1
26308	2018-01-01 04:00:00+01:00	279.0	0.0	0.0	3318.0	1
...
27043	2018-01-31 19:00:00+01:00	387.0	936.0	0.0	10082.0	2
27044	2018-01-31 20:00:00+01:00	380.0	918.0	0.0	10207.0	2
27045	2018-01-31 21:00:00+01:00	382.0	942.0	0.0	9671.0	2
27046	2018-01-31 22:00:00+01:00	375.0	938.0	0.0	8773.0	2
27047	2018-01-31 23:00:00+01:00	374.0	816.0	0.0	8081.0	2

744 rows × 29 columns

In []: df2018_1.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	343.848118	379.564516	0.0	5687.715054	3783.622312	285.73
std	35.794761	295.186322	0.0	1802.274578	1613.661523	44.67
min	219.000000	0.000000	0.0	2452.000000	895.000000	180.00
25%	326.750000	0.000000	0.0	4201.500000	2218.500000	251.00
50%	354.000000	443.000000	0.0	5269.000000	4114.500000	294.50
75%	370.250000	602.250000	0.0	6724.750000	4897.500000	325.00
max	402.000000	952.000000	0.0	10326.000000	6827.000000	351.00

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	genera fossil
744	2015-02-01 00:00:00+01:00	433.0	323.0	0.0	4826.0	48
745	2015-02-01 01:00:00+01:00	445.0	316.0	0.0	4891.0	46
746	2015-02-01 02:00:00+01:00	441.0	320.0	0.0	4652.0	46
747	2015-02-01 03:00:00+01:00	440.0	323.0	0.0	4603.0	45
748	2015-02-01 04:00:00+01:00	432.0	310.0	0.0	4212.0	45
...
1411	2015-02-28 19:00:00+01:00	457.0	0.0	0.0	4591.0	21
1412	2015-02-28 20:00:00+01:00	450.0	0.0	0.0	4652.0	23
1413	2015-02-28 21:00:00+01:00	458.0	0.0	0.0	4358.0	21
1414	2015-02-28 22:00:00+01:00	453.0	0.0	0.0	3835.0	21
1415	2015-02-28 23:00:00+01:00	447.0	0.0	0.0	3747.0	21

672 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	672.000000	672.000000	672.0	672.000000	672.000000	672.00
mean	470.150298	313.418155	0.0	4674.135417	4045.974702	319.23
std	51.148418	375.102898	0.0	1586.403229	2240.121251	65.01
min	346.000000	0.000000	0.0	2591.000000	922.000000	169.00
25%	440.000000	0.000000	0.0	3577.750000	2008.750000	267.00
50%	477.000000	0.000000	0.0	4293.500000	3478.000000	320.00
75%	505.250000	605.250000	0.0	5230.750000	5997.500000	373.25
max	590.000000	984.000000	0.0	12447.000000	7922.000000	442.00

8 rows × 28 columns

In []: 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	gener fossil
9504	2016-02-01 00:00:00+01:00	383.0	0.0	0.0	3204.0	1
9505	2016-02-01 01:00:00+01:00	348.0	0.0	0.0	3268.0	1
9506	2016-02-01 02:00:00+01:00	344.0	0.0	0.0	3361.0	1
9507	2016-02-01 03:00:00+01:00	343.0	0.0	0.0	3455.0	1
9508	2016-02-01 04:00:00+01:00	348.0	0.0	0.0	3584.0	1
...
10195	2016-02-29 19:00:00+01:00	515.0	0.0	0.0	4170.0	2
10196	2016-02-29 20:00:00+01:00	520.0	0.0	0.0	3944.0	2
10197	2016-02-29 21:00:00+01:00	519.0	0.0	0.0	3814.0	2
10198	2016-02-29 22:00:00+01:00	514.0	0.0	0.0	3204.0	2
10199	2016-02-29 23:00:00+01:00	510.0	0.0	0.0	2802.0	1

696 rows × 29 columns

In []: df2016_2.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	696.000000	696.000000	696.0	696.000000	696.000000	696.00
mean	430.344828	191.665230	0.0	4450.360632	3027.655172	297.25
std	80.313301	322.133055	0.0	2159.888718	1878.734871	59.03
min	230.000000	0.000000	0.0	1518.000000	756.000000	152.00
25%	368.750000	0.000000	0.0	3239.000000	1263.500000	250.00
50%	416.000000	0.000000	0.0	3925.000000	2457.000000	293.00
75%	514.250000	261.500000	0.0	5134.750000	4496.000000	350.00
max	563.000000	997.000000	0.0	13566.000000	7176.000000	417.00

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	gener fossil
18288	2017-02-01 00:00:00+01:00	332.0	838.0	0.0	3806.0	5
18289	2017-02-01 01:00:00+01:00	334.0	860.0	0.0	3954.0	4
18290	2017-02-01 02:00:00+01:00	333.0	893.0	0.0	3899.0	4
18291	2017-02-01 03:00:00+01:00	331.0	939.0	0.0	3879.0	4
18292	2017-02-01 04:00:00+01:00	332.0	903.0	0.0	4026.0	4
...
18955	2017-02-28 19:00:00+01:00	389.0	754.0	0.0	4804.0	2
18956	2017-02-28 20:00:00+01:00	388.0	828.0	0.0	4919.0	2
18957	2017-02-28 21:00:00+01:00	388.0	871.0	0.0	4661.0	3
18958	2017-02-28 22:00:00+01:00	391.0	875.0	0.0	4682.0	2
18959	2017-02-28 23:00:00+01:00	388.0	837.0	0.0	4378.0	2

672 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	672.000000	672.000000	672.0	672.000000	672.000000	672.00
mean	351.364583	603.415179	0.0	5561.144345	4382.532738	291.40
std	39.029197	236.988650	0.0	1938.412409	1444.421345	31.23
min	210.000000	0.000000	0.0	3203.000000	1097.000000	185.00
25%	342.000000	525.750000	0.0	4224.750000	3404.000000	268.75
50%	358.000000	599.000000	0.0	4926.000000	4623.500000	296.00
75%	377.000000	821.500000	0.0	6276.750000	5652.750000	315.00
max	407.000000	975.000000	0.0	14283.000000	6735.000000	354.00

8 rows × 28 columns

In []: 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	gener fossil
27048	2018-02-01 00:00:00+01:00	273.0	0.0	0.0	3900.0	1
27049	2018-02-01 01:00:00+01:00	266.0	0.0	0.0	3753.0	1
27050	2018-02-01 02:00:00+01:00	268.0	0.0	0.0	3714.0	1
27051	2018-02-01 03:00:00+01:00	262.0	0.0	0.0	3717.0	1
27052	2018-02-01 04:00:00+01:00	265.0	0.0	0.0	3692.0	1
...
27715	2018-02-28 19:00:00+01:00	356.0	0.0	0.0	7479.0	1
27716	2018-02-28 20:00:00+01:00	360.0	0.0	0.0	7737.0	1
27717	2018-02-28 21:00:00+01:00	351.0	0.0	0.0	6928.0	1
27718	2018-02-28 22:00:00+01:00	332.0	0.0	0.0	6239.0	1
27719	2018-02-28 23:00:00+01:00	335.0	0.0	0.0	5107.0	1

672 rows × 29 columns

In []:

In []: df2018_2.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	672.000000	672.000000	672.0	672.000000	672.000000	672.00
mean	360.511905	406.950893	0.0	5468.040179	4144.389881	295.90
std	39.887993	268.120331	0.0	1726.737880	1466.846267	43.56
min	197.000000	0.000000	0.0	3347.000000	901.000000	196.00
25%	350.000000	235.000000	0.0	4099.000000	3281.750000	260.00
50%	373.000000	398.000000	0.0	4972.500000	4233.500000	313.00
75%	388.000000	623.250000	0.0	6541.750000	5380.000000	331.00
max	410.000000	923.000000	0.0	12228.000000	6853.000000	360.00

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	genera fossil
1416	2015-03-01 00:00:00+01:00	484.0	322.0	0.0	5642.0	59
1417	2015-03-01 01:00:00+01:00	481.0	317.0	0.0	5636.0	58
1418	2015-03-01 02:00:00+01:00	481.0	306.0	0.0	5285.0	54
1419	2015-03-01 03:00:00+01:00	479.0	313.0	0.0	5068.0	54
1420	2015-03-01 04:00:00+01:00	479.0	315.0	0.0	4754.0	54
...
2154	2015-03-31 19:00:00+02:00	470.0	0.0	0.0	3535.0	27
2155	2015-03-31 20:00:00+02:00	465.0	0.0	0.0	3616.0	28
2156	2015-03-31 21:00:00+02:00	448.0	0.0	0.0	3582.0	31
2157	2015-03-31 22:00:00+02:00	441.0	0.0	0.0	3632.0	27
2158	2015-03-31 23:00:00+02:00	434.0	0.0	0.0	3781.0	25

743 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	743.000000	743.000000	743.0	743.000000	743.000000	743.00
mean	468.106326	244.437416	0.0	4614.752355	4233.818304	319.33
std	48.253048	355.778435	0.0	1087.721947	1912.194199	55.16
min	101.000000	0.000000	0.0	1854.000000	1001.000000	87.00
25%	447.000000	0.000000	0.0	3878.500000	2674.500000	277.00
50%	470.000000	0.000000	0.0	4368.000000	3807.000000	319.00
75%	491.000000	342.500000	0.0	5038.500000	5935.000000	360.00
max	577.000000	985.000000	0.0	10036.000000	8031.000000	441.00

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	gener fossil
10200	2016-03-01 00:00:00+01:00	386.0	0.0	0.0	3900.0	1
10201	2016-03-01 01:00:00+01:00	377.0	0.0	0.0	3879.0	1
10202	2016-03-01 02:00:00+01:00	362.0	0.0	0.0	3839.0	1
10203	2016-03-01 03:00:00+01:00	364.0	0.0	0.0	3740.0	1
10204	2016-03-01 04:00:00+01:00	368.0	0.0	0.0	3691.0	1
...	1
10938	2016-03-31 19:00:00+02:00	468.0	0.0	0.0	3651.0	1
10939	2016-03-31 20:00:00+02:00	469.0	0.0	0.0	3890.0	1
10940	2016-03-31 21:00:00+02:00	469.0	0.0	0.0	3783.0	1
10941	2016-03-31 22:00:00+02:00	468.0	0.0	0.0	3797.0	1
10942	2016-03-31 23:00:00+02:00	468.0	0.0	0.0	3749.0	1

743 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	743.000000	743.000000	743.0	743.000000	743.000000	743.0
mean	429.744280	173.203230	0.0	4336.594886	2985.886945	294.0
std	61.118278	287.462484	0.0	1509.730090	1763.223103	43.8
min	279.000000	0.000000	0.0	2139.000000	862.000000	165.0
25%	381.000000	0.000000	0.0	3371.000000	1187.000000	268.5
50%	437.000000	0.000000	0.0	3897.000000	2828.000000	292.0
75%	479.500000	292.000000	0.0	4895.500000	4491.000000	326.0
max	534.000000	949.000000	0.0	12635.000000	6593.000000	411.0

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	gener fossil
18960	2017-03-01 00:00:00+01:00	327.0	888.0	0.0	4863.0	€
18961	2017-03-01 01:00:00+01:00	321.0	888.0	0.0	4918.0	€
18962	2017-03-01 02:00:00+01:00	341.0	940.0	0.0	4693.0	€
18963	2017-03-01 03:00:00+01:00	343.0	899.0	0.0	4370.0	€
18964	2017-03-01 04:00:00+01:00	342.0	908.0	0.0	4380.0	€
...
19698	2017-03-31 19:00:00+02:00	251.0	402.0	0.0	3962.0]
19699	2017-03-31 20:00:00+02:00	250.0	411.0	0.0	4124.0]
19700	2017-03-31 21:00:00+02:00	250.0	387.0	0.0	4142.0]
19701	2017-03-31 22:00:00+02:00	241.0	294.0	0.0	3839.0]
19702	2017-03-31 23:00:00+02:00	239.0	107.0	0.0	3715.0]

743 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generat fossil
count	743.000000	743.000000	743.0	743.000000	743.000000	743.000
mean	284.402423	335.667564	0.0	5337.418573	2967.84926	302.507
std	60.818216	323.278236	0.0	1949.442272	1561.83622	28.266
min	175.000000	0.000000	0.0	3312.000000	1165.000000	239.000
25%	242.500000	0.000000	0.0	4154.000000	1781.000000	274.000
50%	277.000000	328.000000	0.0	4698.000000	2295.000000	309.000
75%	339.000000	562.500000	0.0	5517.000000	3956.000000	325.000
max	392.000000	999.000000	0.0	14680.000000	6836.000000	364.000

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	gener fossil
27720	2018-03-01 00:00:00+01:00	292.0	0.0	0.0	3970.0	1
27721	2018-03-01 01:00:00+01:00	292.0	0.0	0.0	3578.0	1
27722	2018-03-01 02:00:00+01:00	295.0	0.0	0.0	3394.0	
27723	2018-03-01 03:00:00+01:00	290.0	0.0	0.0	3337.0	
27724	2018-03-01 04:00:00+01:00	289.0	0.0	0.0	3591.0	
...
28458	2018-03-31 19:00:00+02:00	244.0	0.0	0.0	3425.0	1
28459	2018-03-31 20:00:00+02:00	246.0	0.0	0.0	3815.0	1
28460	2018-03-31 21:00:00+02:00	259.0	0.0	0.0	4222.0	1
28461	2018-03-31 22:00:00+02:00	267.0	0.0	0.0	4143.0	1
28462	2018-03-31 23:00:00+02:00	271.0	0.0	0.0	4022.0	1

743 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	743.000000	743.000000	743.0	743.000000	743.000000	743.00
mean	307.254374	124.415882	0.0	4936.013459	2157.183042	288.12
std	41.145572	243.478477	0.0	1700.369234	1410.681768	48.10
min	207.000000	0.000000	0.0	2792.000000	700.000000	167.00
25%	282.000000	0.000000	0.0	3929.000000	1222.000000	246.00
50%	308.000000	0.000000	0.0	4372.000000	1518.000000	303.00
75%	332.500000	0.000000	0.0	5367.000000	2662.000000	327.00
max	401.000000	857.000000	0.0	11982.000000	5621.000000	355.00

8 rows × 28 columns

In []: 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	genera fossil l
2159	2015-04-01 00:00:00+02:00	457.0	270.0	0.0	5613.0	61
2160	2015-04-01 01:00:00+02:00	469.0	273.0	0.0	4989.0	57
2161	2015-04-01 02:00:00+02:00	466.0	247.0	0.0	4393.0	52
2162	2015-04-01 03:00:00+02:00	463.0	203.0	0.0	4027.0	49
2163	2015-04-01 04:00:00+02:00	462.0	182.0	0.0	4020.0	46
...
2874	2015-04-30 19:00:00+02:00	393.0	891.0	0.0	4062.0	61
2875	2015-04-30 20:00:00+02:00	388.0	851.0	0.0	4542.0	61
2876	2015-04-30 21:00:00+02:00	386.0	742.0	0.0	5155.0	61
2877	2015-04-30 22:00:00+02:00	377.0	679.0	0.0	4804.0	60
2878	2015-04-30 23:00:00+02:00	377.0	565.0	0.0	4529.0	55

720 rows × 29 columns

In []: df2015_4.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	718.000000	718.000000	718.0	718.000000	718.000000	718.00
mean	426.320334	463.119777	0.0	4952.123955	4819.516713	338.78
std	56.851839	397.334525	0.0	1136.587141	1667.076972	48.05
min	318.000000	0.000000	0.0	2797.000000	1375.000000	208.00
25%	388.250000	0.000000	0.0	4211.250000	3517.250000	303.25
50%	406.000000	542.500000	0.0	4808.000000	4924.500000	347.00
75%	469.000000	902.500000	0.0	5388.000000	5958.500000	375.00
max	574.000000	986.000000	0.0	10654.000000	7981.000000	428.00

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	gener fossil
10943	2016-04-01 00:00:00+02:00	374.0	0.0	0.0	3483.0	1
10944	2016-04-01 01:00:00+02:00	370.0	0.0	0.0	3384.0	1
10945	2016-04-01 02:00:00+02:00	372.0	0.0	0.0	3365.0	
10946	2016-04-01 03:00:00+02:00	369.0	0.0	0.0	3318.0	
10947	2016-04-01 04:00:00+02:00	370.0	0.0	0.0	3430.0	1
...
11658	2016-04-30 19:00:00+02:00	243.0	0.0	0.0	2981.0	1
11659	2016-04-30 20:00:00+02:00	247.0	0.0	0.0	3045.0	
11660	2016-04-30 21:00:00+02:00	251.0	0.0	0.0	3113.0	1
11661	2016-04-30 22:00:00+02:00	256.0	0.0	0.0	3073.0	1
11662	2016-04-30 23:00:00+02:00	252.0	0.0	0.0	3448.0	1

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	286.890278	143.570833	0.0	4263.481944	2226.186111	259.88
std	86.579542	302.080515	0.0	1477.267774	1611.082243	51.37
min	167.000000	0.000000	0.0	2451.000000	818.000000	139.00
25%	220.000000	0.000000	0.0	3340.500000	1129.500000	217.75
50%	246.000000	0.000000	0.0	3867.000000	1528.500000	266.00
75%	352.000000	0.000000	0.0	4722.000000	2212.750000	288.00
max	546.000000	967.000000	0.0	12017.000000	6320.000000	424.00

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	gener fossil
19703	2017-04-01 00:00:00+02:00	362.0	970.0	0.0	4983.0	€
19704	2017-04-01 01:00:00+02:00	359.0	980.0	0.0	4733.0	€
19705	2017-04-01 02:00:00+02:00	353.0	898.0	0.0	4727.0	€
19706	2017-04-01 03:00:00+02:00	346.0	902.0	0.0	4655.0	€
19707	2017-04-01 04:00:00+02:00	347.0	888.0	0.0	4656.0	€
...
20418	2017-04-30 19:00:00+02:00	257.0	0.0	0.0	2947.0]
20419	2017-04-30 20:00:00+02:00	260.0	0.0	0.0	3057.0]
20420	2017-04-30 21:00:00+02:00	266.0	0.0	0.0	3174.0]
20421	2017-04-30 22:00:00+02:00	256.0	0.0	0.0	3375.0]
20422	2017-04-30 23:00:00+02:00	239.0	0.0	0.0	3586.0]

720 rows × 29 columns

In []: df2017_4.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	280.761111	420.115278	0.0	5169.840278	3119.322222	261.49
std	57.848381	275.878406	0.0	2504.057579	1549.673825	41.94
min	194.000000	0.000000	0.0	2736.000000	1042.000000	168.00
25%	235.000000	226.750000	0.0	3741.750000	1922.250000	233.00
50%	257.000000	421.500000	0.0	4331.500000	2820.500000	260.00
75%	337.000000	531.000000	0.0	5052.500000	3754.750000	287.00
max	396.000000	993.000000	0.0	18149.000000	6915.000000	363.00

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	gener fossil
28463	2018-04-01 00:00:00+02:00	277.0	0.0	0.0	3969.0	1
28464	2018-04-01 01:00:00+02:00	269.0	0.0	0.0	3652.0	1
28465	2018-04-01 02:00:00+02:00	271.0	0.0	0.0	3519.0	1
28466	2018-04-01 03:00:00+02:00	280.0	0.0	0.0	3634.0	1
28467	2018-04-01 04:00:00+02:00	280.0	0.0	0.0	3511.0	1
...
29178	2018-04-30 19:00:00+02:00	338.0	0.0	0.0	3585.0	1
29179	2018-04-30 20:00:00+02:00	337.0	0.0	0.0	3851.0	1
29180	2018-04-30 21:00:00+02:00	332.0	0.0	0.0	4499.0	1
29181	2018-04-30 22:00:00+02:00	332.0	0.0	0.0	4350.0	1
29182	2018-04-30 23:00:00+02:00	335.0	0.0	0.0	4331.0	1

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	299.393056	133.979167	0.0	4745.273611	2775.879167	257.62
std	53.001558	251.734784	0.0	1759.070145	1368.301099	47.30
min	214.000000	0.000000	0.0	2497.000000	1005.000000	106.00
25%	247.000000	0.000000	0.0	3644.500000	1604.750000	226.75
50%	306.000000	0.000000	0.0	4184.000000	2467.500000	256.00
75%	346.000000	0.000000	0.0	5156.500000	3592.750000	296.00
max	402.000000	766.000000	0.0	12565.000000	5846.000000	342.00

8 rows × 28 columns

In []: 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	genera fossil
2879		2015-05-01 00:00:00+02:00	494.0	318.0	0.0	5316.0	60
2880		2015-05-01 01:00:00+02:00	496.0	317.0	0.0	5341.0	59
2881		2015-05-01 02:00:00+02:00	493.0	351.0	0.0	4926.0	60
2882		2015-05-01 03:00:00+02:00	493.0	410.0	0.0	4647.0	58
2883		2015-05-01 04:00:00+02:00	493.0	514.0	0.0	4305.0	58
...
3618		2015-05-31 19:00:00+02:00	534.0	580.0	0.0	3601.0	48
3619		2015-05-31 20:00:00+02:00	534.0	695.0	0.0	3756.0	50
3620		2015-05-31 21:00:00+02:00	540.0	829.0	0.0	3947.0	52
3621		2015-05-31 22:00:00+02:00	540.0	931.0	0.0	4337.0	55
3622		2015-05-31 23:00:00+02:00	541.0	918.0	0.0	4192.0	54

744 rows × 29 columns

In []: df2015_5.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generat fossil
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000
mean	503.569892	374.280914	0.0	4415.349462	4019.612903	332.567
std	57.413618	417.889809	0.0	934.243399	1889.671809	53.402
min	306.000000	0.000000	0.0	2901.000000	803.000000	129.000
25%	479.000000	0.000000	0.0	3841.250000	2466.500000	286.750
50%	510.000000	0.000000	0.0	4175.000000	3793.000000	340.000
75%	545.000000	869.000000	0.0	4700.750000	5343.000000	381.000
max	585.000000	989.000000	0.0	8673.000000	7957.000000	422.000

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	gener fossil
11663	2016-05-01 00:00:00+02:00	413.0	0.0	0.0	3790.0	1
11664	2016-05-01 01:00:00+02:00	411.0	0.0	0.0	3770.0	1
11665	2016-05-01 02:00:00+02:00	383.0	0.0	0.0	3723.0	1
11666	2016-05-01 03:00:00+02:00	377.0	0.0	0.0	3463.0	1
11667	2016-05-01 04:00:00+02:00	372.0	0.0	0.0	3781.0	1
...
12402	2016-05-31 19:00:00+02:00	328.0	0.0	0.0	4012.0	1
12403	2016-05-31 20:00:00+02:00	328.0	0.0	0.0	4029.0	1
12404	2016-05-31 21:00:00+02:00	329.0	0.0	0.0	4574.0	1
12405	2016-05-31 22:00:00+02:00	332.0	0.0	0.0	5130.0	2
12406	2016-05-31 23:00:00+02:00	329.0	0.0	0.0	4661.0	2

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	337.202957	179.002688	0.0	4508.034946	2226.651882	277.91
std	47.875123	288.272130	0.0	1406.654443	1677.710361	47.95
min	214.000000	0.000000	0.0	1938.000000	576.000000	189.00
25%	315.000000	0.000000	0.0	3730.250000	966.000000	230.00
50%	328.000000	0.000000	0.0	4217.000000	1716.000000	286.00
75%	347.000000	299.250000	0.0	4956.500000	2369.250000	305.00
max	538.000000	990.000000	0.0	11232.000000	6535.000000	434.00

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	gener fossil
20423	2017-05-01 00:00:00+02:00	359.0	919.0	0.0	4689.0	€
20424	2017-05-01 01:00:00+02:00	361.0	893.0	0.0	4585.0	€
20425	2017-05-01 02:00:00+02:00	359.0	909.0	0.0	4590.0	€
20426	2017-05-01 03:00:00+02:00	353.0	923.0	0.0	4475.0	€
20427	2017-05-01 04:00:00+02:00	355.0	913.0	0.0	4450.0	€
...
21162	2017-05-31 19:00:00+02:00	337.0	836.0	0.0	6502.0	€
21163	2017-05-31 20:00:00+02:00	337.0	867.0	0.0	6411.0	€
21164	2017-05-31 21:00:00+02:00	338.0	849.0	0.0	6631.0	€
21165	2017-05-31 22:00:00+02:00	336.0	847.0	0.0	6753.0	€
21166	2017-05-31 23:00:00+02:00	338.0	832.0	0.0	6648.0	€

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	353.504032	500.770161	0.0	5493.271505	4428.919355	291.92
std	32.993208	245.659871	0.0	2571.092763	1626.989616	37.90
min	220.000000	0.000000	0.0	2974.000000	1007.000000	141.00
25%	346.000000	396.500000	0.0	4011.000000	3185.500000	262.00
50%	359.500000	527.500000	0.0	4735.500000	4482.000000	299.50
75%	374.000000	594.000000	0.0	6361.750000	5949.000000	320.00
max	403.000000	986.000000	0.0	20034.000000	6853.000000	370.00

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	gener fossil
29183	2018-05-01 00:00:00+02:00	285.0	0.0	0.0	3867.0	1
29184	2018-05-01 01:00:00+02:00	282.0	0.0	0.0	3939.0	1
29185	2018-05-01 02:00:00+02:00	283.0	0.0	0.0	3717.0	1
29186	2018-05-01 03:00:00+02:00	279.0	0.0	0.0	3518.0	1
29187	2018-05-01 04:00:00+02:00	283.0	0.0	0.0	3539.0	1
...
29922	2018-05-31 19:00:00+02:00	351.0	421.0	0.0	7121.0	1
29923	2018-05-31 20:00:00+02:00	350.0	434.0	0.0	7182.0	1
29924	2018-05-31 21:00:00+02:00	351.0	442.0	0.0	7414.0	1
29925	2018-05-31 22:00:00+02:00	355.0	442.0	0.0	7558.0	1
29926	2018-05-31 23:00:00+02:00	339.0	399.0	0.0	7604.0	1

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	genera fossil
3623	2015-06-01 00:00:00+02:00	502.0	633.0	0.0	5601.0	64
3624	2015-06-01 01:00:00+02:00	495.0	642.0	0.0	5247.0	61
3625	2015-06-01 02:00:00+02:00	495.0	638.0	0.0	4660.0	55
3626	2015-06-01 03:00:00+02:00	495.0	529.0	0.0	4360.0	51
3627	2015-06-01 04:00:00+02:00	491.0	385.0	0.0	4290.0	49
...
4338	2015-06-30 19:00:00+02:00	497.0	953.0	0.0	5304.0	81
4339	2015-06-30 20:00:00+02:00	500.0	916.0	0.0	5457.0	81
4340	2015-06-30 21:00:00+02:00	504.0	913.0	0.0	6045.0	81
4341	2015-06-30 22:00:00+02:00	508.0	882.0	0.0	6175.0	80
4342	2015-06-30 23:00:00+02:00	508.0	882.0	0.0	5713.0	79

720 rows × 29 columns

In []: df2015_6.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generat fossil
count	719.000000	719.000000	719.0	719.000000	719.000000	719.000000
mean	485.998609	665.162726	0.0	4934.930459	6207.095967	319.229
std	44.492762	278.050075	0.0	1117.826707	1569.125048	50.380
min	307.000000	0.000000	0.0	3191.000000	1106.000000	215.000
25%	471.000000	573.500000	0.0	4177.500000	5474.000000	277.000
50%	487.000000	653.000000	0.0	4618.000000	6486.000000	325.000
75%	507.500000	915.000000	0.0	5410.500000	7443.500000	359.500
max	583.000000	983.000000	0.0	9199.000000	8303.000000	429.000

8 rows × 28 columns

In []: df2016_6 = df2[df2['time'].str.contains("2016-06")]
df2016_6
Monthly Datasets ("Year_Month" ="0000_0")

```
Out[ ]:
```

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	gener fossil
12407	2016-06-01 00:00:00+02:00	376.0	0.0	0.0	3673.0	1
12408	2016-06-01 01:00:00+02:00	369.0	0.0	0.0	3620.0	1
12409	2016-06-01 02:00:00+02:00	371.0	0.0	0.0	3478.0	
12410	2016-06-01 03:00:00+02:00	367.0	0.0	0.0	3654.0	
12411	2016-06-01 04:00:00+02:00	370.0	0.0	0.0	3626.0	
...
13122	2016-06-30 19:00:00+02:00	340.0	646.0	0.0	6622.0	1
13123	2016-06-30 20:00:00+02:00	347.0	643.0	0.0	6367.0	1
13124	2016-06-30 21:00:00+02:00	350.0	643.0	0.0	6782.0	1
13125	2016-06-30 22:00:00+02:00	350.0	644.0	0.0	7250.0	1
13126	2016-06-30 23:00:00+02:00	347.0	654.0	0.0	6758.0	1

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	340.337500	175.600000	0.0	4994.327778	2877.080556	289.82
std	39.726936	301.926344	0.0	1501.655962	1790.711862	34.85
min	223.000000	0.000000	0.0	2258.000000	799.000000	189.00
25%	324.000000	0.000000	0.0	4143.750000	1338.750000	265.00
50%	337.000000	0.000000	0.0	4691.500000	2200.500000	296.50
75%	347.000000	313.500000	0.0	5420.500000	4274.500000	314.00
max	531.000000	981.000000	0.0	10849.000000	6749.000000	381.00

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 coal-derived gas	generation fossil gas	gener fossil
21167	2017-06-01 00:00:00+02:00	356.0	604.0	0.0	5670.0	€
21168	2017-06-01 01:00:00+02:00	351.0	596.0	0.0	5565.0	€
21169	2017-06-01 02:00:00+02:00	349.0	634.0	0.0	5399.0	€
21170	2017-06-01 03:00:00+02:00	349.0	600.0	0.0	5208.0	€
21171	2017-06-01 04:00:00+02:00	352.0	605.0	0.0	4949.0	€
...
21882	2017-06-30 19:00:00+02:00	358.0	557.0	0.0	4541.0	€
21883	2017-06-30 20:00:00+02:00	355.0	563.0	0.0	4686.0	€
21884	2017-06-30 21:00:00+02:00	350.0	607.0	0.0	4983.0	€
21885	2017-06-30 22:00:00+02:00	359.0	612.0	0.0	5264.0	€
21886	2017-06-30 23:00:00+02:00	355.0	583.0	0.0	5418.0	€

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	339.979167	478.811111	0.0	7194.280556	5091.152778	299.39
std	36.219250	305.371145	0.0	2855.328760	1778.218245	37.00
min	218.000000	0.000000	0.0	3282.000000	947.000000	196.00
25%	325.000000	265.000000	0.0	4799.500000	3376.750000	270.00
50%	349.000000	466.500000	0.0	6551.000000	5764.000000	309.00
75%	360.000000	783.750000	0.0	9252.000000	6585.250000	331.00
max	405.000000	985.000000	0.0	16266.000000	7472.000000	352.00

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 coal-derived gas	generation fossil gas	gener fossil
29927	2018-06-01 00:00:00+02:00	298.0	0.0	0.0	5078.0	1
29928	2018-06-01 01:00:00+02:00	286.0	0.0	0.0	4699.0	1
29929	2018-06-01 02:00:00+02:00	285.0	0.0	0.0	4573.0	1
29930	2018-06-01 03:00:00+02:00	285.0	0.0	0.0	4791.0	1
29931	2018-06-01 04:00:00+02:00	278.0	0.0	0.0	4635.0	1
...
30642	2018-06-30 19:00:00+02:00	345.0	618.0	0.0	5523.0	1
30643	2018-06-30 20:00:00+02:00	358.0	635.0	0.0	5765.0	1
30644	2018-06-30 21:00:00+02:00	362.0	656.0	0.0	5989.0	1
30645	2018-06-30 22:00:00+02:00	360.0	663.0	0.0	5996.0	1
30646	2018-06-30 23:00:00+02:00	363.0	623.0	0.0	6124.0	1

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	351.733333	333.998611	0.0	5807.555556	3059.616667	285.12
std	32.707470	303.016432	0.0	1483.951041	1328.116035	34.83
min	222.000000	0.000000	0.0	0.000000	1088.000000	189.00
25%	345.000000	0.000000	0.0	4803.750000	1851.000000	263.75
50%	360.000000	410.000000	0.0	5446.000000	3057.500000	293.00
75%	370.000000	631.250000	0.0	6530.750000	3940.500000	310.00
max	413.000000	938.000000	0.0	10969.000000	6151.000000	364.00

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	gener fossil
13127	2016-07-01 00:00:00+02:00	350.0	0.0	0.0	3616.0	1
13128	2016-07-01 01:00:00+02:00	345.0	0.0	0.0	3657.0	1
13129	2016-07-01 02:00:00+02:00	336.0	0.0	0.0	3660.0	1
13130	2016-07-01 03:00:00+02:00	327.0	0.0	0.0	3600.0	1
13131	2016-07-01 04:00:00+02:00	327.0	0.0	0.0	3666.0	1
...
13866	2016-07-31 19:00:00+02:00	373.0	0.0	0.0	3679.0	2
13867	2016-07-31 20:00:00+02:00	382.0	0.0	0.0	3859.0	2
13868	2016-07-31 21:00:00+02:00	384.0	0.0	0.0	4095.0	2
13869	2016-07-31 22:00:00+02:00	381.0	0.0	0.0	4252.0	2
13870	2016-07-31 23:00:00+02:00	384.0	0.0	0.0	4247.0	2

744 rows × 29 columns

```
In [ ]: df2016_7.describe() # Description tables of monthly datasets ("Year_Month" =
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generat fossil
count	743.000000	743.000000	743.0	743.000000	743.000000	742.000
mean	351.889637	398.815612	0.0	5475.643338	4003.916555	286.690
std	42.165969	385.267866	0.0	1705.003608	1746.961208	41.032
min	214.000000	0.000000	0.0	2551.000000	745.000000	197.000
25%	327.000000	0.000000	0.0	4003.500000	2862.500000	258.250
50%	357.000000	405.000000	0.0	5125.000000	4162.000000	292.000
75%	379.500000	818.000000	0.0	6861.000000	5497.000000	315.000
max	475.000000	979.000000	0.0	9899.000000	6735.000000	439.000

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	genera fossil
4343	2015-07-01 00:00:00+02:00	494.0	890.0	0.0	5277.0	67
4344	2015-07-01 01:00:00+02:00	497.0	936.0	0.0	5253.0	66
4345	2015-07-01 02:00:00+02:00	487.0	895.0	0.0	5279.0	64
4346	2015-07-01 03:00:00+02:00	484.0	899.0	0.0	5128.0	63
4347	2015-07-01 04:00:00+02:00	489.0	977.0	0.0	5298.0	63
...
5082	2015-07-31 19:00:00+02:00	530.0	570.0	0.0	4859.0	75
5083	2015-07-31 20:00:00+02:00	535.0	486.0	0.0	4916.0	69
5084	2015-07-31 21:00:00+02:00	559.0	528.0	0.0	5489.0	73
5085	2015-07-31 22:00:00+02:00	565.0	529.0	0.0	5148.0	73
5086	2015-07-31 23:00:00+02:00	558.0	469.0	0.0	4914.0	72

744 rows × 29 columns

In []: df2015_7.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	512.487903	684.220430	0.0	6529.490591	6748.469086	360.23
std	42.861885	298.294673	0.0	2205.578526	1401.111496	49.97
min	316.000000	0.000000	0.0	3374.000000	1516.000000	242.00
25%	497.000000	565.000000	0.0	4639.750000	6325.000000	316.00
50%	523.000000	791.000000	0.0	5987.500000	7260.500000	381.50
75%	541.000000	936.000000	0.0	8098.750000	7661.500000	402.00
max	584.000000	988.000000	0.0	11895.000000	8313.000000	427.00

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	gener fossil
21887	2017-07-01 00:00:00+02:00	360.0	590.0	0.0	5615.0	E
21888	2017-07-01 01:00:00+02:00	357.0	593.0	0.0	4770.0	E
21889	2017-07-01 02:00:00+02:00	357.0	586.0	0.0	4724.0	E
21890	2017-07-01 03:00:00+02:00	358.0	585.0	0.0	4507.0	L
21891	2017-07-01 04:00:00+02:00	357.0	585.0	0.0	4775.0	L
...
22626	2017-07-31 19:00:00+02:00	383.0	726.0	0.0	9351.0	E
22627	2017-07-31 20:00:00+02:00	362.0	682.0	0.0	8958.0	E
22628	2017-07-31 21:00:00+02:00	355.0	734.0	0.0	9205.0	E
22629	2017-07-31 22:00:00+02:00	359.0	740.0	0.0	9224.0	E
22630	2017-07-31 23:00:00+02:00	360.0	644.0	0.0	7892.0	L

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	367.706989	650.677419	0.0	7366.073925	4561.283602	308.20
std	36.910829	231.481496	0.0	2242.910240	1101.879716	41.14
min	225.000000	0.000000	0.0	3325.000000	1450.000000	210.00
25%	356.000000	476.500000	0.0	5617.000000	3828.250000	271.75
50%	376.000000	633.000000	0.0	6913.500000	4671.500000	323.00
75%	393.000000	867.000000	0.0	9392.000000	5402.750000	343.00
max	424.000000	995.000000	0.0	13609.000000	6825.000000	365.00

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	gener fossil
30647	2018-07-01 00:00:00+02:00	289.0	0.0	0.0	5175.0	1
30648	2018-07-01 01:00:00+02:00	273.0	0.0	0.0	5361.0	1
30649	2018-07-01 02:00:00+02:00	268.0	0.0	0.0	4835.0	1
30650	2018-07-01 03:00:00+02:00	266.0	0.0	0.0	4637.0	1
30651	2018-07-01 04:00:00+02:00	266.0	0.0	0.0	4968.0	2
...
31386	2018-07-31 19:00:00+02:00	391.0	441.0	0.0	9640.0	5
31387	2018-07-31 20:00:00+02:00	391.0	441.0	0.0	7630.0	5
31388	2018-07-31 21:00:00+02:00	396.0	441.0	0.0	7307.0	5
31389	2018-07-31 22:00:00+02:00	395.0	424.0	0.0	7069.0	5
31390	2018-07-31 23:00:00+02:00	396.0	439.0	0.0	6380.0	5

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	743.000000	743.000000	743.0	743.000000	743.000000	743.00
mean	367.220727	506.368775	0.0	5865.691790	4014.707941	281.54
std	34.900007	233.993460	0.0	1536.562105	1178.254070	38.29
min	231.000000	0.000000	0.0	3333.000000	747.000000	187.00
25%	351.000000	422.000000	0.0	4852.500000	3056.500000	254.00
50%	379.000000	614.000000	0.0	5474.000000	4256.000000	289.00
75%	392.000000	663.000000	0.0	6602.000000	4891.500000	310.00
max	410.000000	957.000000	0.0	12551.000000	6291.000000	361.00

8 rows × 28 columns

In []: df2015_8 = df2[df2['time'].str.contains("2015-08")]
df2015_8
Monthly Datasets ("Year_Month" ="0000_0")

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	genera fossil
5087	2015-08-01 00:00:00+02:00	521.0	882.0	0.0	6796.0	76
5088	2015-08-01 01:00:00+02:00	527.0	934.0	0.0	6253.0	76
5089	2015-08-01 02:00:00+02:00	522.0	946.0	0.0	5841.0	77
5090	2015-08-01 03:00:00+02:00	523.0	953.0	0.0	5437.0	77
5091	2015-08-01 04:00:00+02:00	522.0	959.0	0.0	5383.0	78
...
5826	2015-08-31 19:00:00+02:00	504.0	958.0	0.0	7437.0	78
5827	2015-08-31 20:00:00+02:00	498.0	951.0	0.0	8322.0	78
5828	2015-08-31 21:00:00+02:00	493.0	957.0	0.0	8683.0	78
5829	2015-08-31 22:00:00+02:00	491.0	949.0	0.0	8272.0	77
5830	2015-08-31 23:00:00+02:00	492.0	945.0	0.0	7743.0	77

744 rows × 29 columns

In []: df2015_8.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	518.947581	585.767473	0.0	4919.491935	5859.370968	323.91
std	49.287070	340.411157	0.0	1281.720111	1699.635272	53.82
min	257.000000	0.000000	0.0	2630.000000	1570.000000	122.00
25%	497.000000	308.750000	0.0	4038.500000	5058.500000	288.00
50%	535.000000	635.500000	0.0	4630.000000	6237.500000	332.00
75%	552.000000	909.000000	0.0	5439.000000	7215.500000	364.00
max	575.000000	985.000000	0.0	10821.000000	8109.000000	425.00

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	gener fossil
13871	2016-08-01 00:00:00+02:00	342.0	0.0	0.0	3459.0	1
13872	2016-08-01 01:00:00+02:00	310.0	0.0	0.0	3321.0	1
13873	2016-08-01 02:00:00+02:00	311.0	0.0	0.0	3161.0	1
13874	2016-08-01 03:00:00+02:00	316.0	0.0	0.0	3424.0	1
13875	2016-08-01 04:00:00+02:00	322.0	0.0	0.0	3671.0	1
...
14610	2016-08-31 19:00:00+02:00	387.0	911.0	0.0	5609.0	€
14611	2016-08-31 20:00:00+02:00	387.0	910.0	0.0	6098.0	€
14612	2016-08-31 21:00:00+02:00	387.0	912.0	0.0	6562.0	€
14613	2016-08-31 22:00:00+02:00	389.0	900.0	0.0	5883.0	€
14614	2016-08-31 23:00:00+02:00	385.0	887.0	0.0	5411.0	€

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generat fossil
count	744.000000	744.000000	744.0	744.000000	744.000000	744.000000
mean	369.645161	464.373656	0.0	4869.987903	4172.501344	283.306
std	43.260305	373.010567	0.0	1142.879146	1720.690363	47.762
min	173.000000	0.000000	0.0	2339.000000	799.000000	117.000
25%	359.000000	0.000000	0.0	4032.000000	2620.750000	246.000
50%	380.000000	587.000000	0.0	4693.500000	4900.500000	293.000
75%	392.000000	844.750000	0.0	5784.750000	5489.500000	312.000
max	504.000000	944.000000	0.0	8109.000000	6852.000000	415.000

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 coal-derived gas	generation fossil gas	gener fossil
22631	2017-08-01 00:00:00+02:00	358.0	595.0	0.0	5604.0	E
22632	2017-08-01 01:00:00+02:00	352.0	593.0	0.0	5092.0	E
22633	2017-08-01 02:00:00+02:00	353.0	598.0	0.0	4704.0	E
22634	2017-08-01 03:00:00+02:00	358.0	598.0	0.0	4591.0	E
22635	2017-08-01 04:00:00+02:00	358.0	598.0	0.0	4653.0	E
...
23370	2017-08-31 19:00:00+02:00	400.0	583.0	0.0	7323.0	E
23371	2017-08-31 20:00:00+02:00	401.0	655.0	0.0	7679.0	E
23372	2017-08-31 21:00:00+02:00	400.0	644.0	0.0	7641.0	E
23373	2017-08-31 22:00:00+02:00	393.0	609.0	0.0	6974.0	E
23374	2017-08-31 23:00:00+02:00	390.0	564.0	0.0	6213.0	E

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	362.879032	511.846774	0.0	6701.935484	3786.232527	306.23
std	37.830057	257.066075	0.0	1823.156124	1235.674654	40.19
min	248.000000	0.000000	0.0	3520.000000	1382.000000	211.00
25%	343.000000	370.750000	0.0	5252.500000	2813.500000	266.00
50%	370.000000	589.000000	0.0	6260.500000	3698.500000	319.00
75%	393.000000	643.000000	0.0	8300.500000	4726.250000	339.00
max	418.000000	985.000000	0.0	10898.000000	6906.000000	363.00

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 coal-derived gas	generation fossil gas	gener fossil
31391	2018-08-01 00:00:00+02:00	276.0	880.0	0.0	5187.0	4
31392	2018-08-01 01:00:00+02:00	273.0	777.0	0.0	5031.0	4
31393	2018-08-01 02:00:00+02:00	272.0	852.0	0.0	4734.0	4
31394	2018-08-01 03:00:00+02:00	274.0	869.0	0.0	4679.0	4
31395	2018-08-01 04:00:00+02:00	270.0	863.0	0.0	4757.0	4
...
32130	2018-08-31 19:00:00+02:00	382.0	727.0	0.0	7654.0	5
32131	2018-08-31 20:00:00+02:00	382.0	727.0	0.0	7816.0	5
32132	2018-08-31 21:00:00+02:00	383.0	756.0	0.0	7294.0	5
32133	2018-08-31 22:00:00+02:00	380.0	704.0	0.0	5612.0	5
32134	2018-08-31 23:00:00+02:00	373.0	683.0	0.0	4749.0	5

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	361.325269	300.142473	0.0	6119.346774	4121.565860	283.33
std	32.709523	309.837792	0.0	2053.746498	1439.512717	38.78
min	225.000000	0.000000	0.0	3086.000000	944.000000	164.00
25%	353.000000	0.000000	0.0	4429.500000	2960.000000	259.75
50%	369.000000	243.500000	0.0	5457.500000	4134.500000	293.00
75%	381.000000	578.000000	0.0	7693.000000	5340.000000	312.00
max	403.000000	959.000000	0.0	12347.000000	6860.000000	364.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	genera fossil
5831	2015-09-01 00:00:00+02:00	532.0	956.0	0.0	5023.0	80
5832	2015-09-01 01:00:00+02:00	532.0	916.0	0.0	4409.0	75
5833	2015-09-01 02:00:00+02:00	531.0	820.0	0.0	4425.0	73
5834	2015-09-01 03:00:00+02:00	525.0	913.0	0.0	4400.0	73
5835	2015-09-01 04:00:00+02:00	524.0	925.0	0.0	4303.0	73
...
6546	2015-09-30 19:00:00+02:00	530.0	906.0	0.0	5343.0	74
6547	2015-09-30 20:00:00+02:00	535.0	924.0	0.0	6297.0	74
6548	2015-09-30 21:00:00+02:00	535.0	929.0	0.0	6894.0	74
6549	2015-09-30 22:00:00+02:00	528.0	912.0	0.0	5482.0	73
6550	2015-09-30 23:00:00+02:00	535.0	887.0	0.0	4957.0	73

720 rows × 29 columns

In []: df2015_9.describe() # Description tables of monthly datasets ("Year_Month" =

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	517.106944	548.083333	0.0	5076.581944	5653.873611	343.49
std	43.157324	363.538515	0.0	1285.125140	1536.541230	55.86
min	344.000000	0.000000	0.0	2691.000000	1608.000000	202.00
25%	502.000000	189.000000	0.0	4295.250000	4729.000000	300.00
50%	521.000000	704.000000	0.0	4806.000000	5921.000000	328.50
75%	544.000000	885.000000	0.0	5471.250000	6906.000000	397.25
max	590.000000	984.000000	0.0	10853.000000	8244.000000	449.00

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	gener fossil
14615	2016-09-01 00:00:00+02:00	377.0	0.0	0.0	3619.0	1
14616	2016-09-01 01:00:00+02:00	354.0	0.0	0.0	3675.0	1
14617	2016-09-01 02:00:00+02:00	358.0	0.0	0.0	3775.0	1
14618	2016-09-01 03:00:00+02:00	358.0	0.0	0.0	3942.0	1
14619	2016-09-01 04:00:00+02:00	354.0	0.0	0.0	3898.0	1
...
15330	2016-09-30 19:00:00+02:00	357.0	636.0	0.0	5982.0	€
15331	2016-09-30 20:00:00+02:00	358.0	613.0	0.0	6046.0	€
15332	2016-09-30 21:00:00+02:00	360.0	596.0	0.0	6016.0	€
15333	2016-09-30 22:00:00+02:00	356.0	594.0	0.0	5859.0	€
15334	2016-09-30 23:00:00+02:00	351.0	592.0	0.0	5301.0	€

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	generat fossil
count	720.000000	720.000000	720.0	720.000000	720.000000	720.000
mean	349.391667	473.720833	0.0	4643.550000	4208.013889	281.386
std	39.648833	337.968867	0.0	1156.294013	1708.676767	39.528
min	199.000000	0.000000	0.0	2640.000000	884.000000	155.000
25%	339.000000	48.250000	0.0	3654.000000	2720.750000	254.750
50%	354.000000	546.500000	0.0	4551.500000	4557.000000	290.000
75%	366.000000	740.250000	0.0	5479.000000	5769.250000	304.000
max	477.000000	976.000000	0.0	8645.000000	6794.000000	410.000

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 coal-derived gas	generation fossil gas	gener fossil
23375	2017-09-01 00:00:00+02:00	352.0	574.0	0.0	4674.0	4
23376	2017-09-01 01:00:00+02:00	348.0	621.0	0.0	4488.0	3
23377	2017-09-01 02:00:00+02:00	349.0	551.0	0.0	4475.0	3
23378	2017-09-01 03:00:00+02:00	342.0	540.0	0.0	4514.0	3
23379	2017-09-01 04:00:00+02:00	343.0	642.0	0.0	4676.0	3
...
24090	2017-09-30 19:00:00+02:00	379.0	647.0	0.0	7478.0	4
24091	2017-09-30 20:00:00+02:00	378.0	662.0	0.0	7669.0	4
24092	2017-09-30 21:00:00+02:00	379.0	656.0	0.0	7493.0	4
24093	2017-09-30 22:00:00+02:00	377.0	641.0	0.0	7089.0	4
24094	2017-09-30 23:00:00+02:00	374.0	645.0	0.0	6800.0	3

720 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	351.968056	642.630556	0.0	7092.451389	4010.465278	310.80
std	36.314272	259.615345	0.0	2174.924207	1240.158034	32.59
min	243.000000	0.000000	0.0	3282.000000	1461.000000	213.00
25%	331.000000	551.000000	0.0	5393.250000	3093.250000	287.00
50%	359.000000	650.000000	0.0	6650.500000	4040.500000	322.50
75%	381.000000	887.250000	0.0	8744.000000	5138.000000	336.00
max	408.000000	999.000000	0.0	13783.000000	6592.000000	365.00

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	gener fossil
32135	2018-09-01 00:00:00+02:00	283.0	922.0	0.0	6745.0	€
32136	2018-09-01 01:00:00+02:00	276.0	926.0	0.0	5428.0	€
32137	2018-09-01 02:00:00+02:00	280.0	936.0	0.0	4826.0	€
32138	2018-09-01 03:00:00+02:00	281.0	801.0	0.0	4970.0	€
32139	2018-09-01 04:00:00+02:00	283.0	694.0	0.0	4685.0	€
...
32850	2018-09-30 19:00:00+02:00	354.0	718.0	0.0	4206.0	€
32851	2018-09-30 20:00:00+02:00	353.0	679.0	0.0	4212.0	€
32852	2018-09-30 21:00:00+02:00	355.0	704.0	0.0	4141.0	€
32853	2018-09-30 22:00:00+02:00	354.0	610.0	0.0	3988.0	€
32854	2018-09-30 23:00:00+02:00	347.0	572.0	0.0	3950.0	€

720 rows × 29 columns

In []: df2018_9.describe()

```
Out[ ]:
```

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	355.468056	558.161111	0.0	5842.062500	4651.018056	296.13
std	37.043583	252.992557	0.0	1599.083686	1329.139806	40.00
min	196.000000	0.000000	0.0	3039.000000	1203.000000	205.00
25%	343.000000	478.750000	0.0	4533.250000	3935.500000	255.75
50%	363.000000	680.000000	0.0	5474.000000	5090.500000	308.00
75%	379.000000	744.000000	0.0	7007.000000	5604.750000	331.00
max	405.000000	936.000000	0.0	10578.000000	6858.000000	362.00

8 rows × 28 columns

```
In [ ]: df2018_5.describe()
```

```
Out[ ]:
```

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	318.309140	307.704301	0.0	5777.162634	3674.151882	280.45
std	42.403733	273.388593	0.0	1595.993360	1517.692063	42.22
min	0.000000	0.000000	0.0	2352.000000	1039.000000	44.00
25%	291.750000	0.000000	0.0	4437.750000	2173.500000	247.75
50%	323.000000	384.000000	0.0	5500.000000	4055.000000	293.00
75%	350.000000	557.750000	0.0	6830.250000	5092.250000	310.00
max	399.000000	943.000000	0.0	10778.000000	6185.000000	353.00

8 rows × 28 columns

```
In [ ]: df2018_3.describe()
```

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	743.000000	743.000000	743.0	743.000000	743.000000	743.00
mean	307.254374	124.415882	0.0	4936.013459	2157.183042	288.12
std	41.145572	243.478477	0.0	1700.369234	1410.681768	48.10
min	207.000000	0.000000	0.0	2792.000000	700.000000	167.00
25%	282.000000	0.000000	0.0	3929.000000	1222.000000	246.00
50%	308.000000	0.000000	0.0	4372.000000	1518.000000	303.00
75%	332.500000	0.000000	0.0	5367.000000	2662.000000	327.00
max	401.000000	857.000000	0.0	11982.000000	5621.000000	355.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	genera fossil
6551	2015-10-01 00:00:00+02:00	506.0	959.0	0.0	5522.0	76
6552	2015-10-01 01:00:00+02:00	506.0	879.0	0.0	5235.0	73
6553	2015-10-01 02:00:00+02:00	509.0	911.0	0.0	4592.0	67
6554	2015-10-01 03:00:00+02:00	511.0	890.0	0.0	4274.0	63
6555	2015-10-01 04:00:00+02:00	512.0	917.0	0.0	4122.0	61
...
7291	2015-10-31 19:00:00+01:00	521.0	0.0	0.0	4976.0	60
7292	2015-10-31 20:00:00+01:00	518.0	0.0	0.0	5079.0	59
7293	2015-10-31 21:00:00+01:00	525.0	0.0	0.0	5115.0	59
7294	2015-10-31 22:00:00+01:00	525.0	0.0	0.0	4825.0	56
7295	2015-10-31 23:00:00+01:00	522.0	0.0	0.0	4578.0	52

745 rows × 29 columns

In []: `df2015_10.describe() # Description tables of monthly datasets ("Year_Month")`

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	519.556452	528.018817	0.0	5231.500000	5788.694892	323.39
std	42.329599	352.768164	0.0	1340.424742	1372.217094	61.14
min	375.000000	0.000000	0.0	3485.000000	2360.000000	201.00
25%	498.000000	216.750000	0.0	4349.250000	4714.250000	277.00
50%	529.000000	564.000000	0.0	4895.500000	5985.500000	320.00
75%	552.000000	909.500000	0.0	5677.000000	7008.750000	374.00
max	585.000000	986.000000	0.0	11956.000000	8359.000000	442.00

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	gener fossil
15335	2016-10-01 00:00:00+02:00	355.0	0.0	0.0	3497.0	
15336	2016-10-01 01:00:00+02:00	361.0	0.0	0.0	3363.0	
15337	2016-10-01 02:00:00+02:00	367.0	0.0	0.0	3314.0	
15338	2016-10-01 03:00:00+02:00	365.0	0.0	0.0	3237.0	
15339	2016-10-01 04:00:00+02:00	357.0	0.0	0.0	3100.0	
...	
16075	2016-10-31 19:00:00+01:00	362.0	642.0	0.0	6179.0	
16076	2016-10-31 20:00:00+01:00	368.0	619.0	0.0	5484.0	
16077	2016-10-31 21:00:00+01:00	367.0	626.0	0.0	5460.0	
16078	2016-10-31 22:00:00+01:00	365.0	546.0	0.0	4919.0	
16079	2016-10-31 23:00:00+01:00	359.0	524.0	0.0	4364.0	

745 rows × 29 columns

In []: df2016_10.describe() # Description tables of monthly datasets ("Year_Month")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	745.000000	745.000000	745.0	745.000000	745.000000	745.00
mean	336.218792	613.769128	0.0	6222.820134	4271.040268	276.99
std	41.611544	352.860276	0.0	2697.861633	1740.678858	41.44
min	193.000000	0.000000	0.0	2536.000000	715.000000	112.00
25%	318.000000	319.000000	0.0	4117.000000	3126.000000	243.00
50%	340.000000	697.000000	0.0	5246.000000	4974.000000	284.00
75%	356.000000	912.000000	0.0	8627.000000	5562.000000	305.00
max	494.000000	995.000000	0.0	12981.000000	6521.000000	414.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	gener fossil
24095	2017-10-01 00:00:00+02:00	349.0	878.0	0.0	4702.0	E
24096	2017-10-01 01:00:00+02:00	350.0	863.0	0.0	4234.0	E
24097	2017-10-01 02:00:00+02:00	352.0	879.0	0.0	4348.0	E
24098	2017-10-01 03:00:00+02:00	351.0	859.0	0.0	4138.0	E
24099	2017-10-01 04:00:00+02:00	352.0	858.0	0.0	4014.0	E
...
24835	2017-10-31 19:00:00+01:00	367.0	825.0	0.0	14705.0	E
24836	2017-10-31 20:00:00+01:00	363.0	800.0	0.0	14058.0	E
24837	2017-10-31 21:00:00+01:00	367.0	701.0	0.0	10720.0	E
24838	2017-10-31 22:00:00+01:00	365.0	610.0	0.0	8643.0	E
24839	2017-10-31 23:00:00+01:00	363.0	645.0	0.0	7793.0	E

745 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	745.000000	745.000000	745.0	745.000000	745.000000	745.00
mean	343.938255	561.322148	0.0	6961.800000	4321.130201	287.41
std	28.832785	254.442651	0.0	2701.462979	1319.103218	36.37
min	252.000000	0.000000	0.0	3271.000000	1449.000000	200.00
25%	330.000000	381.000000	0.0	4799.000000	3253.000000	261.00
50%	347.000000	584.000000	0.0	5928.000000	4638.000000	291.00
75%	361.000000	761.000000	0.0	8893.000000	5351.000000	316.00
max	407.000000	986.000000	0.0	15892.000000	6587.000000	354.00

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	gener fossil
32855	2018-10-01 00:00:00+02:00	265.0	583.0	0.0	4612.0	E
32856	2018-10-01 01:00:00+02:00	262.0	557.0	0.0	4272.0	E
32857	2018-10-01 02:00:00+02:00	266.0	605.0	0.0	4250.0	E
32858	2018-10-01 03:00:00+02:00	266.0	632.0	0.0	4350.0	E
32859	2018-10-01 04:00:00+02:00	267.0	600.0	0.0	4186.0	L
...
33595	2018-10-31 19:00:00+01:00	297.0	510.0	0.0	13270.0	E
33596	2018-10-31 20:00:00+01:00	299.0	510.0	0.0	12980.0	E
33597	2018-10-31 21:00:00+01:00	298.0	504.0	0.0	12441.0	E
33598	2018-10-31 22:00:00+01:00	295.0	508.0	0.0	10839.0	E
33599	2018-10-31 23:00:00+01:00	291.0	500.0	0.0	8416.0	E

745 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	745.000000	745.000000	745.0	745.000000	745.000000	745.00
mean	317.927517	405.837584	0.0	6127.481879	3782.816107	291.29
std	37.563532	259.168722	0.0	2233.644072	1442.747645	41.25
min	185.000000	0.000000	0.0	3197.000000	884.000000	203.00
25%	296.000000	175.000000	0.0	4306.000000	2596.000000	249.00
50%	317.000000	508.000000	0.0	5498.000000	3990.000000	306.00
75%	345.000000	590.000000	0.0	7196.000000	5117.000000	326.00
max	388.000000	766.000000	0.0	13270.000000	6067.000000	350.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	genera fossil l
7296	2015-11-01 00:00:00+01:00	525.0	713.0	0.0	5689.0	68
7297	2015-11-01 01:00:00+01:00	527.0	572.0	0.0	4808.0	60
7298	2015-11-01 02:00:00+01:00	518.0	497.0	0.0	4265.0	53
7299	2015-11-01 03:00:00+01:00	511.0	371.0	0.0	4290.0	45
7300	2015-11-01 04:00:00+01:00	513.0	337.0	0.0	4240.0	38
...
8011	2015-11-30 19:00:00+01:00	458.0	859.0	0.0	11821.0	69
8012	2015-11-30 20:00:00+01:00	463.0	913.0	0.0	11586.0	70
8013	2015-11-30 21:00:00+01:00	461.0	948.0	0.0	11430.0	71
8014	2015-11-30 22:00:00+01:00	463.0	961.0	0.0	9825.0	71
8015	2015-11-30 23:00:00+01:00	464.0	937.0	0.0	8403.0	71

720 rows × 29 columns

In []: df2015_11.describe() # Description tables of monthly datasets ("Year_Month")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	503.616667	695.284722	0.0	5387.741667	5982.631944	346.06
std	48.514228	293.445189	0.0	1670.962167	1606.453974	57.27
min	365.000000	0.000000	0.0	2771.000000	1021.000000	207.00
25%	464.000000	568.500000	0.0	4340.500000	5224.750000	302.00
50%	510.000000	816.000000	0.0	4875.500000	6517.000000	347.50
75%	545.000000	937.000000	0.0	6119.500000	7130.000000	400.00
max	586.000000	983.000000	0.0	11821.000000	8159.000000	434.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	gener fossil
16080	2016-11-01 00:00:00+01:00	321.0	0.0	0.0	3218.0	
16081	2016-11-01 01:00:00+01:00	325.0	0.0	0.0	3058.0	
16082	2016-11-01 02:00:00+01:00	330.0	0.0	0.0	2923.0	
16083	2016-11-01 03:00:00+01:00	317.0	0.0	0.0	2845.0	
16084	2016-11-01 04:00:00+01:00	314.0	0.0	0.0	2948.0	
...	
16795	2016-11-30 19:00:00+01:00	382.0	894.0	0.0	8492.0	€
16796	2016-11-30 20:00:00+01:00	381.0	892.0	0.0	8542.0	€
16797	2016-11-30 21:00:00+01:00	382.0	894.0	0.0	8680.0	€
16798	2016-11-30 22:00:00+01:00	377.0	893.0	0.0	7691.0	€
16799	2016-11-30 23:00:00+01:00	371.0	892.0	0.0	6747.0	€

720 rows × 29 columns

In []: `df2016_11.describe() # Description tables of monthly datasets ("Year_Month")`

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	719.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	355.573018	649.545833	0.0	6439.015278	4581.631944	271.56
std	44.776541	349.322269	0.0	2947.627392	1961.932734	42.30
min	186.000000	0.000000	0.0	2845.000000	661.000000	177.00
25%	339.000000	323.750000	0.0	4146.250000	2533.000000	232.00
50%	359.000000	879.000000	0.0	5231.000000	5292.000000	278.00
75%	377.500000	891.000000	0.0	8352.750000	5956.750000	293.00
max	506.000000	987.000000	0.0	16250.000000	7327.000000	408.00

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 coal-derived gas	generation fossil gas	gener fossil
24840	2017-11-01 00:00:00+01:00	349.0	865.0	0.0	4318.0	€
24841	2017-11-01 01:00:00+01:00	347.0	889.0	0.0	4003.0	€
24842	2017-11-01 02:00:00+01:00	348.0	915.0	0.0	4111.0	€
24843	2017-11-01 03:00:00+01:00	345.0	940.0	0.0	4287.0	€
24844	2017-11-01 04:00:00+01:00	344.0	911.0	0.0	4062.0	€
...
25555	2017-11-30 19:00:00+01:00	382.0	988.0	0.0	14920.0	€
25556	2017-11-30 20:00:00+01:00	381.0	988.0	0.0	15078.0	€
25557	2017-11-30 21:00:00+01:00	389.0	985.0	0.0	13571.0	€
25558	2017-11-30 22:00:00+01:00	385.0	984.0	0.0	11956.0	€
25559	2017-11-30 23:00:00+01:00	383.0	980.0	0.0	11502.0	€

720 rows × 29 columns

In []: df2017_11.describe() # Description tables of monthly datasets ("Year_Month")

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	354.593056	667.647222	0.0	8534.818056	5491.216667	303.54
std	39.487111	273.416870	0.0	4059.602836	1678.616981	39.85
min	0.000000	0.000000	0.0	3161.000000	0.000000	0.00
25%	341.000000	530.500000	0.0	5079.500000	4579.250000	282.00
50%	360.000000	657.000000	0.0	7092.500000	6167.500000	314.00
75%	379.000000	914.250000	0.0	11854.250000	6763.250000	331.00
max	411.000000	988.000000	0.0	18981.000000	7239.000000	360.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	gener fossil
33600	2018-11-01 00:00:00+01:00	262.0	194.0	0.0	4152.0	€
33601	2018-11-01 01:00:00+01:00	272.0	217.0	0.0	3597.0	€
33602	2018-11-01 02:00:00+01:00	262.0	202.0	0.0	3557.0	€
33603	2018-11-01 03:00:00+01:00	270.0	213.0	0.0	3645.0	€
33604	2018-11-01 04:00:00+01:00	273.0	201.0	0.0	3632.0	€
...	€
34315	2018-11-30 19:00:00+01:00	355.0	0.0	0.0	11477.0	€
34316	2018-11-30 20:00:00+01:00	357.0	0.0	0.0	11413.0	€
34317	2018-11-30 21:00:00+01:00	356.0	0.0	0.0	10380.0	€
34318	2018-11-30 22:00:00+01:00	354.0	0.0	0.0	8134.0	€
34319	2018-11-30 23:00:00+01:00	352.0	0.0	0.0	6629.0	€

720 rows × 29 columns

In []: `df2018_11.describe() # Description tables of monthly datasets ("Year_Month")`

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	720.000000	720.000000	720.0	720.000000	720.000000	720.00
mean	342.102778	375.736111	0.0	6945.804167	4838.777778	272.98
std	37.448461	314.466433	0.0	2616.456572	1696.159079	41.87
min	207.000000	0.000000	0.0	2651.000000	745.000000	164.00
25%	328.000000	0.000000	0.0	5003.750000	3752.000000	243.00
50%	353.000000	468.500000	0.0	6125.500000	5273.000000	286.00
75%	368.000000	693.000000	0.0	9575.500000	6263.250000	305.00
max	386.000000	797.000000	0.0	13250.000000	6928.000000	346.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	genera fossil
8016	2015-12-01 00:00:00+01:00	537.0	647.0	0.0	4947.0	64
8017	2015-12-01 01:00:00+01:00	530.0	637.0	0.0	4419.0	62
8018	2015-12-01 02:00:00+01:00	526.0	547.0	0.0	4115.0	58
8019	2015-12-01 03:00:00+01:00	520.0	493.0	0.0	3996.0	56
8020	2015-12-01 04:00:00+01:00	517.0	480.0	0.0	4132.0	53
...
8755	2015-12-31 19:00:00+01:00	363.0	0.0	0.0	6770.0	40
8756	2015-12-31 20:00:00+01:00	376.0	0.0	0.0	6709.0	40
8757	2015-12-31 21:00:00+01:00	392.0	0.0	0.0	5791.0	40
8758	2015-12-31 22:00:00+01:00	401.0	0.0	0.0	4822.0	37
8759	2015-12-31 23:00:00+01:00	400.0	0.0	0.0	4594.0	33

744 rows × 29 columns

In []: `df2015_12.describe() # Description tables of monthly datasets ("Year_Month")`

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	743.00000	743.00000	743.0	743.00000	743.00000	743.00
mean	483.92463	493.480485	0.0	5062.588156	5323.676985	330.65
std	59.59178	400.370555	0.0	1245.427021	1931.720695	63.94
min	356.00000	0.000000	0.0	3047.000000	1201.000000	159.00
25%	435.00000	0.000000	0.0	4134.500000	4020.000000	291.00
50%	510.00000	582.000000	0.0	4715.000000	5767.000000	329.00
75%	533.00000	917.000000	0.0	5821.000000	7097.000000	383.00
max	570.00000	988.000000	0.0	10388.000000	8135.000000	435.00

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	gener fossil
16800	2016-12-01 00:00:00+01:00	358.0	0.0	0.0	3293.0	1
16801	2016-12-01 01:00:00+01:00	359.0	0.0	0.0	3195.0	1
16802	2016-12-01 02:00:00+01:00	362.0	0.0	0.0	3146.0	1
16803	2016-12-01 03:00:00+01:00	368.0	0.0	0.0	3408.0	1
16804	2016-12-01 04:00:00+01:00	371.0	0.0	0.0	3358.0	1
...
17539	2016-12-31 19:00:00+01:00	336.0	977.0	0.0	4880.0	€
17540	2016-12-31 20:00:00+01:00	337.0	904.0	0.0	4911.0	€
17541	2016-12-31 21:00:00+01:00	340.0	895.0	0.0	4667.0	€
17542	2016-12-31 22:00:00+01:00	337.0	896.0	0.0	4564.0	€
17543	2016-12-31 23:00:00+01:00	337.0	905.0	0.0	5116.0	€

744 rows × 29 columns

In []: `df2016_12.describe() # Description tables of monthly datasets ("Year_Month")`

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	338.776882	670.795699	0.0	6176.176075	4984.295699	276.63
std	42.204925	376.661196	0.0	2895.579185	1908.667080	43.10
min	195.000000	0.000000	0.0	2703.000000	1053.000000	166.00
25%	325.000000	332.500000	0.0	4166.000000	3958.750000	250.00
50%	334.000000	892.000000	0.0	5092.000000	5471.000000	282.00
75%	346.000000	962.250000	0.0	7146.250000	6469.250000	301.00
max	488.000000	996.000000	0.0	15205.000000	7289.000000	422.00

8 rows × 28 columns

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

Out[]:

	time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	generation fossil gas	gener fossil
25560	2017-12-01 00:00:00+01:00	360.0	891.0	0.0	5304.0	€
25561	2017-12-01 01:00:00+01:00	359.0	899.0	0.0	4627.0	€
25562	2017-12-01 02:00:00+01:00	360.0	924.0	0.0	4580.0	€
25563	2017-12-01 03:00:00+01:00	359.0	929.0	0.0	4576.0	€
25564	2017-12-01 04:00:00+01:00	354.0	930.0	0.0	4499.0	€
...
26299	2017-12-31 19:00:00+01:00	273.0	0.0	0.0	3101.0	
26300	2017-12-31 20:00:00+01:00	267.0	0.0	0.0	3022.0	
26301	2017-12-31 21:00:00+01:00	268.0	0.0	0.0	3205.0	
26302	2017-12-31 22:00:00+01:00	277.0	0.0	0.0	3266.0	
26303	2017-12-31 23:00:00+01:00	278.0	0.0	0.0	3552.0	

744 rows × 29 columns

In []: `df2017_12.describe() # Description tables of monthly datasets ("Year_Month")`

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.000000	744.000000	744.0	744.000000	744.000000	744.00
mean	347.173387	476.543011	0.0	5835.915323	4330.653226	293.92
std	41.488337	350.132012	0.0	2573.985107	2255.178294	50.78
min	215.000000	0.000000	0.0	2949.000000	764.000000	165.00
25%	327.000000	202.500000	0.0	3951.000000	2039.750000	268.00
50%	358.000000	522.500000	0.0	4841.500000	4630.000000	310.00
75%	379.250000	800.750000	0.0	7087.500000	6540.250000	334.00
max	416.000000	990.000000	0.0	13463.000000	7740.000000	363.00

8 rows × 28 columns

In []: 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	gener fossil
34320	2018-12-01 00:00:00+01:00	272.0	299.0	0.0	4919.0	E
34321	2018-12-01 01:00:00+01:00	271.0	292.0	0.0	4240.0	E
34322	2018-12-01 02:00:00+01:00	282.0	247.0	0.0	4038.0	E
34323	2018-12-01 03:00:00+01:00	306.0	203.0	0.0	4006.0	E
34324	2018-12-01 04:00:00+01:00	294.0	205.0	0.0	4251.0	E
...
35059	2018-12-31 19:00:00+01:00	297.0	0.0	0.0	7634.0	Z
35060	2018-12-31 20:00:00+01:00	296.0	0.0	0.0	7241.0	Z
35061	2018-12-31 21:00:00+01:00	292.0	0.0	0.0	7025.0	Z
35062	2018-12-31 22:00:00+01:00	293.0	0.0	0.0	6562.0	Z
35063	2018-12-31 23:00:00+01:00	290.0	0.0	0.0	6926.0	Z

744 rows × 29 columns

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

Out[]:

	generation biomass	generation fossil brown coal/lignite	generation fossil coal- derived gas	generation fossil gas	generation fossil hard coal	genera foss
count	744.00000	744.00000	744.0	744.00000	744.00000	744.00
mean	320.16129	406.228495	0.0	6463.536290	3365.758065	269.02
std	33.88101	269.542385	0.0	2102.415078	1415.262858	44.05
min	226.00000	0.000000	0.0	2673.000000	704.000000	163.00
25%	301.00000	202.750000	0.0	4878.500000	2419.250000	232.00
50%	317.00000	448.000000	0.0	6045.000000	2995.500000	279.00
75%	343.00000	663.000000	0.0	7747.750000	4676.750000	301.00
max	386.00000	910.000000	0.0	13292.000000	6721.000000	363.00

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 []:

```
io2 = []
df2018_12.describe(include='all').loc['mean']
io2.append(df2018_12.describe(include='all').loc['mean'][["price actual"]])
df2018_11.describe(include='all').loc['mean']
io2.append(df2018_11.describe(include='all').loc['mean'][["price actual"]])
df2018_10.describe(include='all').loc['mean']
io2.append(df2018_10.describe(include='all').loc['mean'][["price actual"]])
df2018_9.describe(include='all').loc['mean']
io2.append(df2018_9.describe(include='all').loc['mean'][["price actual"]])
df2018_8.describe(include='all').loc['mean']
io2.append(df2018_8.describe(include='all').loc['mean'][["price actual"]])
df2018_7.describe(include='all').loc['mean']
io2.append(df2018_7.describe(include='all').loc['mean'][["price actual"]])
df2018_6.describe(include='all').loc['mean']
io2.append(df2018_6.describe(include='all').loc['mean'][["price actual"]])
df2018_5.describe(include='all').loc['mean']
io2.append(df2018_5.describe(include='all').loc['mean'][["price actual"]])
df2018_4.describe(include='all').loc['mean']
io2.append(df2018_4.describe(include='all').loc['mean'][["price actual"]])
df2018_3.describe(include='all').loc['mean']
io2.append(df2018_3.describe(include='all').loc['mean'][["price actual"]])
df2018_2.describe(include='all').loc['mean']
io2.append(df2018_2.describe(include='all').loc['mean'][["price actual"]])
df2018_1.describe(include='all').loc['mean']
```



```

io2.append(df2015_9.describe(include='all')).loc['mean']["price actual"])
df2015_8.describe(include='all').loc['mean']
io2.append(df2015_8.describe(include='all')).loc['mean']["price actual"])
df2015_7.describe(include='all').loc['mean']
io2.append(df2015_7.describe(include='all')).loc['mean']["price actual"])
df2015_6.describe(include='all').loc['mean']
io2.append(df2015_6.describe(include='all')).loc['mean']["price actual"])
df2015_5.describe(include='all').loc['mean']
io2.append(df2015_5.describe(include='all')).loc['mean']["price actual"])
df2015_4.describe(include='all').loc['mean']
io2.append(df2015_4.describe(include='all')).loc['mean']["price actual"])
df2015_3.describe(include='all').loc['mean']
io2.append(df2015_3.describe(include='all')).loc['mean']["price actual"])
df2015_2.describe(include='all').loc['mean']
io2.append(df2015_2.describe(include='all')).loc['mean']["price actual"])
df2015_1.describe(include='all').loc['mean']
io2.append(df2015_1.describe(include='all')).loc['mean']["price actual"])

io2.reverse() # Average monthly values
print(io2)

```

```
[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335,
57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 6
0.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176,
45.57872311827957, 36.75208333333333, 36.818008075370116, 32.618666666666666,
34.69137096774194, 46.26631944444444, 47.502016129032256, 47.60233870967742,
50.40559722222222, 60.18242953020134, 62.58105555555556, 67.59513440860215,
79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717916666666666,
53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08432795698924,
55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15127688172043,
56.511975806451616, 60.877098214285716, 48.279717362045766, 50.4007361111111
1, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.3639112903225
8, 76.91404166666666, 70.36221476510067, 66.6235138888889, 67.0426075268817
2]
```

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"]

```



```

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, 1461.519498607242
4, 1920.2728494623657, 1998.6606397774688, 1973.7405913978494, 1738.95026881
72042, 1591.2902777777779, 1082.279569892473, 1171.7194444444444, 874.313593
5397039, 1025.741935483871, 1258.058908045977, 1382.9057873485867, 1390.4166
666666667, 1689.1814516129032, 1942.8291666666667, 1915.7752355316286, 1835.
8521505376343, 1548.213888888889, 1013.2832214765101, 951.3819444444445, 91
4.9758064516129, 1088.4180107526881, 1157.4613095238096, 1415.0915208613728,
1468.8277777777778, 1800.0524193548388, 1919.0236111111112, 2025.11962365591
4, 1754.159946236559, 1731.3666666666666, 1344.3261744966444, 1180.645833333
3333, 1035.4260752688172, 1051.4193548387098, 1246.110119047619, 1352.318977
1197847, 1501.647222222221, 1526.899193548387, 1855.197222222223, 2010.345
8950201884, 1706.9301075268818, 1510.6527777777778, 999.786577181208, 831.66
80555555556, 875.5900537634409]
```

In []:

```

In [ ]: biomass= []
df2018_12.describe(include='all').loc['mean']
biomass.append(df2018_12.describe(include='all').loc['mean']['generation biodigester'])
df2018_11.describe(include='all').loc['mean']
biomass.append(df2018_11.describe(include='all').loc['mean']['generation biodigester'])
df2018_10.describe(include='all').loc['mean']
biomass.append(df2018_10.describe(include='all').loc['mean']['generation biodigester'])
df2018_9.describe(include='all').loc['mean']
biomass.append(df2018_9.describe(include='all').loc['mean']['generation biodigester'])
df2018_8.describe(include='all').loc['mean']
biomass.append(df2018_8.describe(include='all').loc['mean']['generation biodigester'])

```



```

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, 426.3203342618384
 7, 503.5698924731183, 485.99860917941584, 512.4879032258065, 518.94758064516
 13, 517.1069444444445, 519.5564516129032, 503.6166666666667, 483.92462987886
 944, 459.9959677419355, 430.3448275862069, 429.7442799461642, 286.8902777777
 778, 337.2029569892473, 340.3375, 351.8896366083446, 369.64516129032256, 34
 9.39166666666665, 336.2187919463087, 355.57301808066757, 338.7768817204301,
 347.28360215053766, 351.3645833333333, 284.40242261103634, 280.76111111111
 1, 353.5040322580645, 339.9791666666667, 367.7069892473118, 362.879032258064
 5, 351.9680555555557, 343.938255033557, 354.5930555555557, 347.17338709677
 42, 343.8481182795699, 360.51190476190476, 307.2543741588156, 299.3930555555
 556, 318.30913978494624, 351.73333333333335, 367.2207267833109, 361.32526881
 72043, 355.4680555555557, 317.9275167785235, 342.10277777777776, 320.161290
 32258067]

In []: len(biomass)

Out[]: 48

In []: fossil_gas =[]

```

df2018_12.describe(include='all').loc['mean']

```



```
fossil_gas.append(df2016_8.describe(include='all')).loc['mean'][["generation f
df2016_7.describe(include='all')).loc['mean']
fossil_gas.append(df2016_7.describe(include='all')).loc['mean'][["generation f
df2016_6.describe(include='all')).loc['mean']
fossil_gas.append(df2016_6.describe(include='all')).loc['mean'][["generation f
df2016_5.describe(include='all')).loc['mean']
fossil_gas.append(df2016_5.describe(include='all')).loc['mean'][["generation f
df2016_4.describe(include='all')).loc['mean']
fossil_gas.append(df2016_4.describe(include='all')).loc['mean'][["generation f
df2016_3.describe(include='all')).loc['mean']
fossil_gas.append(df2016_3.describe(include='all')).loc['mean'][["generation f
df2016_2.describe(include='all')).loc['mean']
fossil_gas.append(df2016_2.describe(include='all')).loc['mean'][["generation f
df2016_1.describe(include='all')).loc['mean']
fossil_gas.append(df2016_1.describe(include='all')).loc['mean'][["generation f
df2015_12.describe(include='all')).loc['mean']
fossil_gas.append(df2015_12.describe(include='all')).loc['mean'][["generation f
df2015_11.describe(include='all')).loc['mean']
fossil_gas.append(df2015_11.describe(include='all')).loc['mean'][["generation f
df2015_10.describe(include='all')).loc['mean']
fossil_gas.append(df2015_10.describe(include='all')).loc['mean'][["generation f
df2015_9.describe(include='all')).loc['mean']
fossil_gas.append(df2015_9.describe(include='all')).loc['mean'][["generation f
df2015_8.describe(include='all')).loc['mean']
fossil_gas.append(df2015_8.describe(include='all')).loc['mean'][["generation f
df2015_7.describe(include='all')).loc['mean']
fossil_gas.append(df2015_7.describe(include='all')).loc['mean'][["generation f
df2015_6.describe(include='all')).loc['mean']
fossil_gas.append(df2015_6.describe(include='all')).loc['mean'][["generation f
df2015_5.describe(include='all')).loc['mean']
fossil_gas.append(df2015_5.describe(include='all')).loc['mean'][["generation f
df2015_4.describe(include='all')).loc['mean']
fossil_gas.append(df2015_4.describe(include='all')).loc['mean'][["generation f
df2015_3.describe(include='all')).loc['mean']
fossil_gas.append(df2015_3.describe(include='all')).loc['mean'][["generation f
df2015_2.describe(include='all')).loc['mean']
fossil_gas.append(df2015_2.describe(include='all')).loc['mean'][["generation f
df2015_1.describe(include='all')).loc['mean']
fossil_gas.append(df2015_1.describe(include='all')).loc['mean'][["generation f

fossil_gas.reverse() # Average monthly values

print(fossil_gas)
```

```
[4850.009549795362, 4674.135416666667, 4614.7523553162855, 4952.12395543175  
4, 4415.3494623655915, 4934.930458970793, 6529.490591397849, 4919.4919354838  
71, 5076.581944444444, 5231.5, 5387.741666666667, 5062.588156123822, 5271.11  
96236559135, 4450.360632183908, 4336.594885598924, 4263.481944444445, 4508.0  
34946236559, 4994.327777777778, 5475.64333781965, 4869.987903225807, 4643.5  
5, 6222.820134228188, 6439.015277777778, 6176.176075268817, 6412.59139784946  
3, 5561.144345238095, 5337.418573351279, 5169.840277777777, 5493.27150537634  
4, 7194.280555555555, 7366.073924731183, 6701.935483870968, 7092.45138888888  
9, 6961.8, 8534.818055555555, 5835.915322580645, 5687.715053763441, 5468.040  
178571428, 4936.013458950202, 4745.273611111111, 5777.162634408603, 5807.555  
555555556, 5865.691790040377, 6119.346774193548, 5842.0625, 6127.48187919463  
1, 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']['genera  
df2018_11.describe(include='all').loc['mean']  
other_renewable.append(df2018_11.describe(include='all').loc['mean']['genera  
df2018_10.describe(include='all').loc['mean']  
other_renewable.append(df2018_10.describe(include='all').loc['mean']['genera  
df2018_9.describe(include='all').loc['mean']  
other_renewable.append(df2018_9.describe(include='all').loc['mean']['generat  
df2018_8.describe(include='all').loc['mean']  
other_renewable.append(df2018_8.describe(include='all').loc['mean']['generat  
df2018_7.describe(include='all').loc['mean']  
other_renewable.append(df2018_7.describe(include='all').loc['mean']['generat  
df2018_6.describe(include='all').loc['mean']  
other_renewable.append(df2018_6.describe(include='all').loc['mean']['generat  
df2018_5.describe(include='all').loc['mean']  
other_renewable.append(df2018_5.describe(include='all').loc['mean']['generat  
df2018_4.describe(include='all').loc['mean']  
other_renewable.append(df2018_4.describe(include='all').loc['mean']['generat  
df2018_3.describe(include='all').loc['mean']  
other_renewable.append(df2018_3.describe(include='all').loc['mean']['generat  
df2018_2.describe(include='all').loc['mean']  
other_renewable.append(df2018_2.describe(include='all').loc['mean']['generat  
df2018_1.describe(include='all').loc['mean']  
other_renewable.append(df2018_1.describe(include='all').loc['mean']['generat  
df2017_12.describe(include='all').loc['mean']  
other_renewable.append(df2017_12.describe(include='all').loc['mean']['genera  
df2017_11.describe(include='all').loc['mean']  
other_renewable.append(df2017_11.describe(include='all').loc['mean']['genera  
df2017_10.describe(include='all').loc['mean']  
other_renewable.append(df2017_10.describe(include='all').loc['mean']['genera  
df2017_9.describe(include='all').loc['mean']  
other_renewable.append(df2017_9.describe(include='all').loc['mean']['generat  
df2017_8.describe(include='all').loc['mean']  
other_renewable.append(df2017_8.describe(include='all').loc['mean']['generat  
df2017_7.describe(include='all').loc['mean']  
other_renewable.append(df2017_7.describe(include='all').loc['mean']['generat  
df2017_6.describe(include='all').loc['mean']  
other_renewable.append(df2017_6.describe(include='all').loc['mean']['generat
```



```
df2015_1.describe(include='all').loc['mean']
other_renewable.append(df2015_1.describe(include='all')).loc['mean']['generation']

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, 92.105555555555, 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']
df2018_1.describe(include='all').loc['mean']
nuclear.append(df2018_1.describe(include='all')).loc['mean']['generation nuclear']
df2017_12.describe(include='all').loc['mean']
nuclear.append(df2017_12.describe(include='all')).loc['mean']['generation nuclear']
df2017_11.describe(include='all').loc['mean']
nuclear.append(df2017_11.describe(include='all')).loc['mean']['generation nuclear']
df2017_10.describe(include='all').loc['mean']
nuclear.append(df2017_10.describe(include='all')).loc['mean']['generation nuclear']
df2017_9.describe(include='all').loc['mean']
nuclear.append(df2017_9.describe(include='all')).loc['mean']['generation nuclear']
df2017_8.describe(include='all').loc['mean']
nuclear.append(df2017_8.describe(include='all')).loc['mean']['generation nuclear']
```



```

df2015_3.describe(include='all').loc['mean']
nuclear.append(df2015_3.describe(include='all').loc['mean']['generation nucl'])
df2015_2.describe(include='all').loc['mean']
nuclear.append(df2015_2.describe(include='all').loc['mean']['generation nucl'])
df2015_1.describe(include='all').loc['mean']
nuclear.append(df2015_1.describe(include='all').loc['mean']['generation nucl'])

nuclear.reverse() # Average monthly values
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, 6675.75555555555555, 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 or solar'])
df2018_11.describe(include='all').loc['mean']
wind.append(df2018_11.describe(include='all').loc['mean']['generation wind or solar'])
df2018_10.describe(include='all').loc['mean']
wind.append(df2018_10.describe(include='all').loc['mean']['generation wind or solar'])
df2018_9.describe(include='all').loc['mean']
wind.append(df2018_9.describe(include='all').loc['mean']['generation wind or solar'])
df2018_8.describe(include='all').loc['mean']
wind.append(df2018_8.describe(include='all').loc['mean']['generation wind or solar'])
df2018_7.describe(include='all').loc['mean']
wind.append(df2018_7.describe(include='all').loc['mean']['generation wind or solar'])
df2018_6.describe(include='all').loc['mean']
wind.append(df2018_6.describe(include='all').loc['mean']['generation wind or solar'])
df2018_5.describe(include='all').loc['mean']
wind.append(df2018_5.describe(include='all').loc['mean']['generation wind or solar'])
df2018_4.describe(include='all').loc['mean']
wind.append(df2018_4.describe(include='all').loc['mean']['generation wind or solar'])
df2018_3.describe(include='all').loc['mean']
wind.append(df2018_3.describe(include='all').loc['mean']['generation wind or solar'])
df2018_2.describe(include='all').loc['mean']
wind.append(df2018_2.describe(include='all').loc['mean']['generation wind or solar'])
df2018_1.describe(include='all').loc['mean']
wind.append(df2018_1.describe(include='all').loc['mean']['generation wind or solar'])
df2017_12.describe(include='all').loc['mean']
wind.append(df2017_12.describe(include='all').loc['mean']['generation wind or solar'])
df2017_11.describe(include='all').loc['mean']
wind.append(df2017_11.describe(include='all').loc['mean']['generation wind or solar'])
df2017_10.describe(include='all').loc['mean']
wind.append(df2017_10.describe(include='all').loc['mean']['generation wind or solar'])

```



```

df2015_5.describe(include='all').loc['mean']
wind.append(df2015_5.describe(include='all').loc['mean']['generation wind or
df2015_4.describe(include='all').loc['mean']
wind.append(df2015_4.describe(include='all').loc['mean']['generation wind or
df2015_3.describe(include='all').loc['mean']
wind.append(df2015_3.describe(include='all').loc['mean']['generation wind or
df2015_2.describe(include='all').loc['mean']
wind.append(df2015_2.describe(include='all').loc['mean']['generation wind or
df2015_1.describe(include='all').loc['mean']
wind.append(df2015_1.describe(include='all').loc['mean']['generation wind or

wind.reverse() # Average monthly values
print(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, 4993.3991935
48387, 6534.337643678161, 5884.690444145357, 5343.191666666667, 5309.3844086
02151, 5638.5375, 5715.833109017497, 5103.193548387097, 5484.695833333333, 4
788.928859060403, 5766.394444444444, 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, 6864.55505
9523809, 8771.240915208613, 5289.24722222222, 4389.743279569892, 4779.43333
333333, 4059.907133243607, 4358.579301075269, 4265.988888888889, 6156.93959
7315436, 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
df2018_11.describe(include='all').loc['mean']
fossil_oil.append(df2018_11.describe(include='all').loc['mean']['generation
df2018_10.describe(include='all').loc['mean']
fossil_oil.append(df2018_10.describe(include='all').loc['mean']['generation
df2018_9.describe(include='all').loc['mean']
fossil_oil.append(df2018_9.describe(include='all').loc['mean']['generation f
df2018_8.describe(include='all').loc['mean']
fossil_oil.append(df2018_8.describe(include='all').loc['mean']['generation f
df2018_7.describe(include='all').loc['mean']
fossil_oil.append(df2018_7.describe(include='all').loc['mean']['generation f
df2018_6.describe(include='all').loc['mean']
fossil_oil.append(df2018_6.describe(include='all').loc['mean']['generation f
df2018_5.describe(include='all').loc['mean']
fossil_oil.append(df2018_5.describe(include='all').loc['mean']['generation f
df2018_4.describe(include='all').loc['mean']
fossil_oil.append(df2018_4.describe(include='all').loc['mean']['generation f
df2018_3.describe(include='all').loc['mean']
fossil_oil.append(df2018_3.describe(include='all').loc['mean']['generation f
df2018_2.describe(include='all').loc['mean']
fossil_oil.append(df2018_2.describe(include='all').loc['mean']['generation f
df2018_1.describe(include='all').loc['mean']
fossil_oil.append(df2018_1.describe(include='all').loc['mean']['generation f
df2017_12.describe(include='all').loc['mean']
fossil_oil.append(df2017_12.describe(include='all').loc['mean']['generation

```



```

df2015_7.describe(include='all').loc['mean']
fossil_oil.append(df2015_7.describe(include='all').loc['mean'][['generation f
df2015_6.describe(include='all').loc['mean']
fossil_oil.append(df2015_6.describe(include='all').loc['mean'][['generation f
df2015_5.describe(include='all').loc['mean']
fossil_oil.append(df2015_5.describe(include='all').loc['mean'][['generation f
df2015_4.describe(include='all').loc['mean']
fossil_oil.append(df2015_4.describe(include='all').loc['mean'][['generation f
df2015_3.describe(include='all').loc['mean']
fossil_oil.append(df2015_3.describe(include='all').loc['mean'][['generation f
df2015_2.describe(include='all').loc['mean']
fossil_oil.append(df2015_2.describe(include='all').loc['mean'][['generation f
df2015_1.describe(include='all').loc['mean']
fossil_oil.append(df2015_1.describe(include='all').loc['mean'][['generation f

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

```

[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.7813370473537
6, 332.56720430107526, 319.2294853963839, 360.23387096774195, 323.9139784946
237, 343.49166666666667, 323.39112903225805, 346.06388888888887, 330.65141318
97712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875,
277.9153225806452, 289.82916666666665, 286.6900269541779, 283.3064516129032
3, 281.3861111111111, 276.9959731543624, 271.56944444444446, 276.63978494623
655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.49027777
77778, 291.9206989247312, 299.3930555555556, 308.2002688172043, 306.23521505
376345, 310.8013888888889, 287.4187919463087, 303.5430555555555, 293.926075
2688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916
666666666, 280.4502688172043, 285.12083333333334, 281.5450874831763, 283.333
333333333, 296.1361111111111, 291.2993288590604, 272.9833333333335, 269.02
150537634407]

In []: Fossil_hard = []
df2018_12.describe(include='all').loc['mean']
Fossil_hard.append(df2018_12.describe(include='all').loc['mean'][['generation f
df2018_11.describe(include='all').loc['mean']
Fossil_hard.append(df2018_11.describe(include='all').loc['mean'][['generation f
df2018_10.describe(include='all').loc['mean']
Fossil_hard.append(df2018_10.describe(include='all').loc['mean'][['generation f
df2018_9.describe(include='all').loc['mean']
Fossil_hard.append(df2018_9.describe(include='all').loc['mean'][['generation f
df2018_8.describe(include='all').loc['mean']
Fossil_hard.append(df2018_8.describe(include='all').loc['mean'][['generation f
df2018_7.describe(include='all').loc['mean']
Fossil_hard.append(df2018_7.describe(include='all').loc['mean'][['generation f
df2018_6.describe(include='all').loc['mean']
Fossil_hard.append(df2018_6.describe(include='all').loc['mean'][['generation f
df2018_5.describe(include='all').loc['mean']
Fossil_hard.append(df2018_5.describe(include='all').loc['mean'][['generation f
df2018_4.describe(include='all').loc['mean']
Fossil_hard.append(df2018_4.describe(include='all').loc['mean'][['generation f
df2018_3.describe(include='all').loc['mean']
Fossil_hard.append(df2018_3.describe(include='all').loc['mean'][['generation f
df2018_2.describe(include='all').loc['mean']
Fossil_hard.append(df2018_2.describe(include='all').loc['mean'][['generation f
df2018_1.describe(include='all').loc['mean']


```

Fossil_hard.append(df2015_9.describe(include='all')).loc['mean']['generation']
df2015_8.describe(include='all').loc['mean']
Fossil_hard.append(df2015_8.describe(include='all')).loc['mean']['generation']
df2015_7.describe(include='all').loc['mean']
Fossil_hard.append(df2015_7.describe(include='all')).loc['mean']['generation']
df2015_6.describe(include='all').loc['mean']
Fossil_hard.append(df2015_6.describe(include='all')).loc['mean']['generation']
df2015_5.describe(include='all').loc['mean']
Fossil_hard.append(df2015_5.describe(include='all')).loc['mean']['generation']
df2015_4.describe(include='all').loc['mean']
Fossil_hard.append(df2015_4.describe(include='all')).loc['mean']['generation']
df2015_3.describe(include='all').loc['mean']
Fossil_hard.append(df2015_3.describe(include='all')).loc['mean']['generation']
df2015_2.describe(include='all').loc['mean']
Fossil_hard.append(df2015_2.describe(include='all')).loc['mean']['generation']
df2015_1.describe(include='all').loc['mean']
Fossil_hard.append(df2015_1.describe(include='all')).loc['mean']['generation']

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

```

```
[5411.263301500682, 4045.9747023809523, 4233.818304172274, 4819.51671309192
2, 4019.6129032258063, 6207.095966620306, 6748.469086021505, 5859.3709677419
36, 5653.873611111111, 5788.694892473119, 5982.631944444444, 5323.6769851951
55, 4204.615591397849, 3027.655172413793, 2985.886944818304, 2226.1861111111
11, 2226.65188172043, 2877.0805555555557, 4003.9165545087485, 4172.501344086
021, 4208.013888888889, 4271.040268456376, 4581.631944444444, 4984.295698924
731, 5533.782258064516, 4382.5327380952385, 2967.8492597577388, 3119.322222
22222, 4428.919354838709, 5091.152777777777, 4561.283602150537, 3786.2325268
817203, 4010.465277777778, 4321.130201342282, 5491.216666666666, 4330.653225
806452, 3783.6223118279568, 4144.389880952381, 2157.1830417227457, 2775.8791
666666666, 3674.15188172043, 3059.616666666667, 4014.707940780619, 4121.5658
60215053, 4651.018055555555, 3782.8161073825504, 4838.777777777777, 3365.758
0645161293]
```

In []:

```

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

```



```

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

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

```

[572.8512960436562, 313.41815476190476, 244.43741588156124, 463.119777158774
 35, 374.2809139784946, 665.162726008345, 684.2204301075269, 585.767473118279
 6, 548.0833333333334, 528.0188172043011, 695.2847222222222, 493.48048452207
 3, 417.9274193548387, 191.66522988505747, 173.20323014804845, 143.5708333333
 3333, 179.002688172043, 175.6, 398.8156123822342, 464.3736559139785, 473.720
 83333333336, 613.7691275167786, 649.5458333333333, 670.7956989247311, 688.64
 51612903226, 603.4151785714286, 335.66756393001344, 420.1152777777778, 500.7
 7016129032256, 478.8111111111113, 650.6774193548387, 511.8467741935484, 64
 2.6305555555556, 561.3221476510067, 667.647222222222, 476.5430107526882, 37
 9.56451612903226, 406.95089285714283, 124.41588156123822, 133.9791666666666
 6, 307.7043010752688, 333.9986111111113, 506.36877523553164, 300.1424731182
 7955, 558.1611111111112, 405.83758389261743, 375.7361111111111, 406.22849462
 365593]

In []:

In []: hydro_water= []

```

df2018_12.describe(include='all').loc['mean']
hydro_water.append(df2018_12.describe(include='all').loc['mean']['generation'])
df2018_11.describe(include='all').loc['mean']
hydro_water.append(df2018_11.describe(include='all').loc['mean']['generation'])
df2018_10.describe(include='all').loc['mean']
hydro_water.append(df2018_10.describe(include='all').loc['mean']['generation'])
df2018_9.describe(include='all').loc['mean']
hydro_water.append(df2018_9.describe(include='all').loc['mean']['generation'])
df2018_8.describe(include='all').loc['mean']
hydro_water.append(df2018_8.describe(include='all').loc['mean']['generation'])
df2018_7.describe(include='all').loc['mean']
hydro_water.append(df2018_7.describe(include='all').loc['mean']['generation'])

```



```

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

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

```

```
[2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.012534818941
7, 2798.184139784946, 2531.1682892906815, 2412.5255376344085, 1963.063172043
0108, 2261.1569444444444, 2276.9193548387098, 2404.3902777777776, 1943.42799
461642, 4075.5846774193546, 4881.568965517241, 3901.5450874831763, 5119.2958
33333334, 4581.743279569892, 2983.793055555554, 2556.6971736204578, 2460.02
01612903224, 2303.461111111111, 2500.489932885906, 2501.297222222224, 2805.
2970430107525, 2156.951612903226, 1874.8497023809523, 2348.4589502018844, 16
11.4152777777779, 1639.616935483871, 1448.6444444444444, 1290.983870967742,
1260.6908602150538, 1570.509722222223, 1229.2845637583894, 1371.33888888888
9, 1452.9569892473119, 2271.896505376344, 2773.2038690476193, 4378.419919246
298, 4159.898611111111, 2931.19623655914, 3362.5291666666667, 2789.228802153
432, 2296.0295698924733, 2267.641666666667, 2172.1919463087247, 2665.9, 276
3.0241935483873]
```

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.

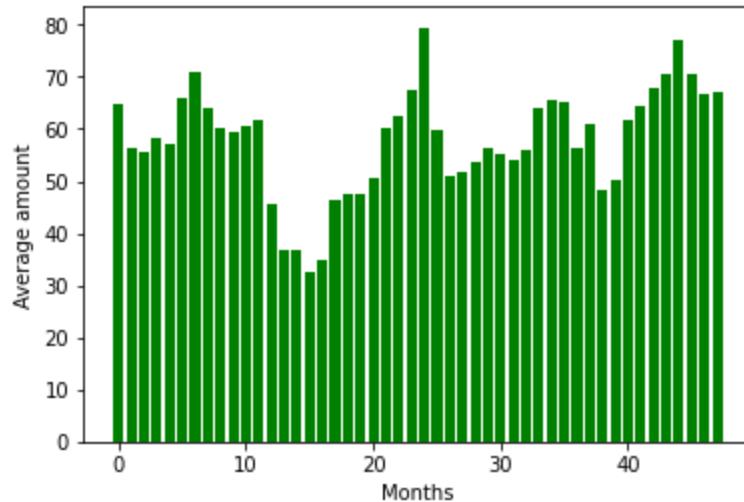
```
In [ ]: #Histograms

Price_Actual_Dict = {key: i for i, key in enumerate(io2)}
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")
plt.ylabel('Average amount')
plt.xlabel('Months')
plt.bar(list(Price_Actual_Dict.values()), Price_Actual_Dict.keys(), color='green')
print(dicDates)
import matplotlib.pyplot as plt

plt.show()

{64.9490188172043: 0, 56.38385416666667: 1, 55.52246298788695: 2, 58.35408333333335: 3, 57.29405913978494: 4, 65.97490277777777: 5, 71.0720430107527: 6, 63.99806451612903: 7, 60.25479166666667: 8, 59.40676510067113: 9, 60.72679166666667: 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, 79.49208333333334: 24, 59.83779761904762: 25, 50.95989232839838: 26, 51.71791666666666: 27, 53.772620967741936: 28, 56.25822222222222: 29, 55.25258064516129: 30, 54.08432795698924: 31, 55.81655555555556: 32, 63.92528859060402: 33, 65.43065277777778: 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, 66.623513888889: 46, 67.04260752688172: 47}

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, 2
1, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
0, 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, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]

In [ ]: Months1 = Months
Months1 = sm.add_constant(Months1)
io21 = io2
io21 = sm.add_constant(io21)

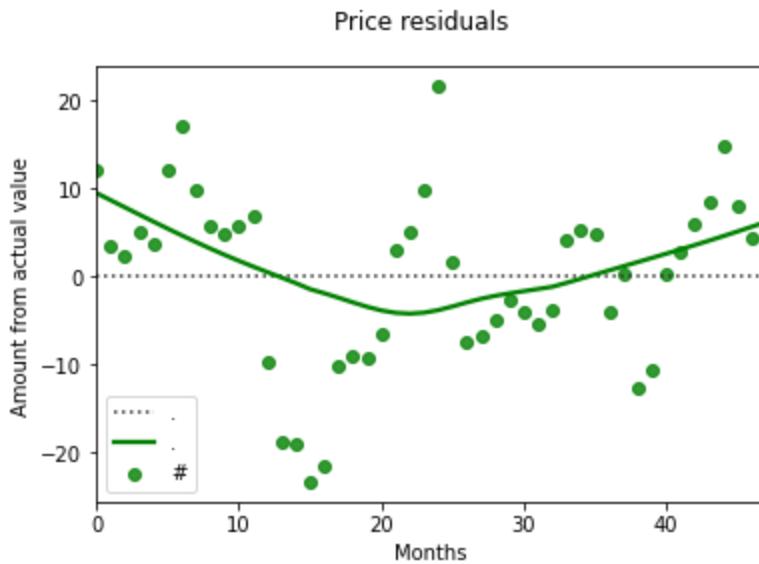
In [ ]: sns.residplot(x= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
56.383854166666666,
55.522462987886975,
58.354083333333333,
57.29405913978498,
65.9749027777778,
71.07204301075271,
63.99806451612899,
60.254791666666634,
59.40676510067113,
60.72679166666668,
61.901760752688226,
45.57872311827956,
36.752083333333374,
36.81800807537014,
32.618666666666666,
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,
56.25822222222224,
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], lowess = True, color="g")
plt.suptitle("Price residuals")
plt.xlabel("Months")
plt.ylabel("Amount from actual value")
plt.legend("#")

```

Out[]: <matplotlib.legend.Legend at 0x7f81cb143c10>



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

In [ ]: y = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
x = np.array([64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354

In [ ]:

In [ ]: modelprice = stats.linregress ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
,[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335
])

In [ ]: io2 = [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.3540833333

In [ ]:
```

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, io2, 2))
print(modelpricequad)

io21 = sm.add_constant(io2)
xp = polynomial_features.fit_transform(io21)

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

Quad = poly.fit(X_poly, io2)
lin2 = LinearRegression()
lin2.fit(X_poly, io2)
Price_Quad = sm.OLS(Months, xp).fit()

ypred = Price_Quad.predict(xp)

Price_Quad.summary()

# Quadratic Scatterplots
plt.scatter(Months, io2, color = 'blue')

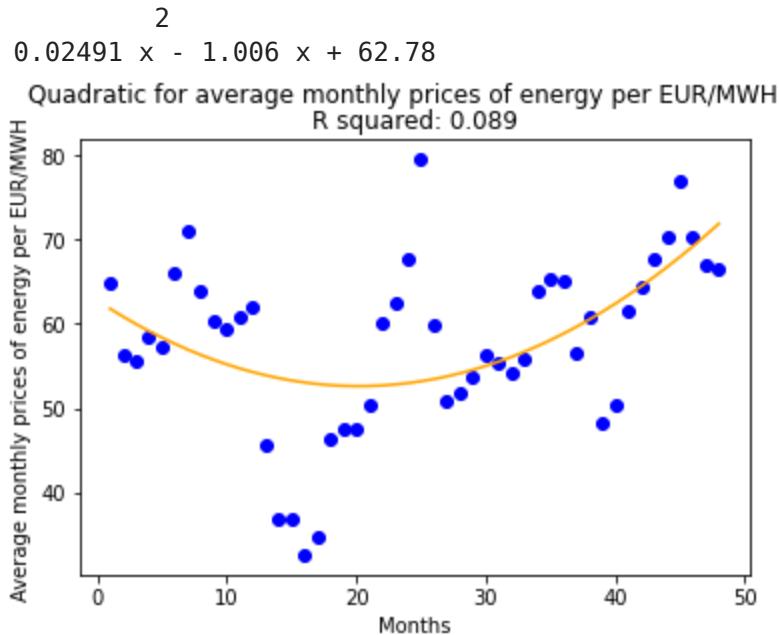
plt.plot(Months, lin2.predict(poly.fit_transform(Months1)), color = 'orange')
plt.title("R squared: 0.089")
```

```

plt.suptitle('Quadratic for 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 linear 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 quadratic model of best fit.

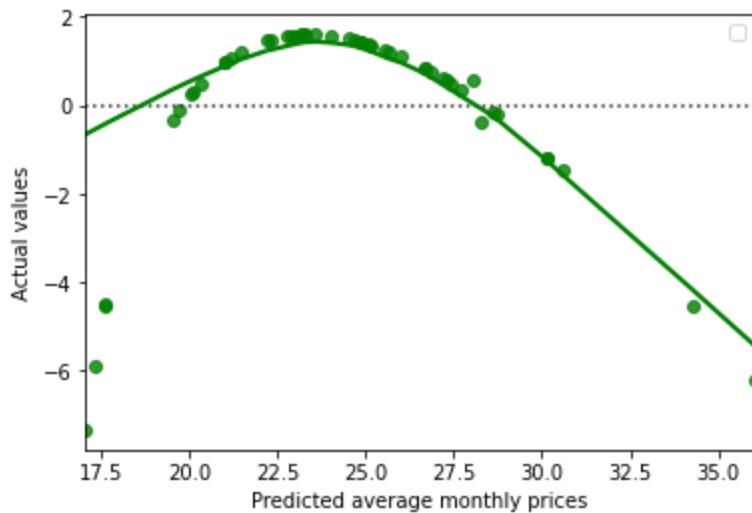
```

In [ ]: plt.suptitle("Predicted average monthly OLS quadratic price of energy per EU")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend("..#")
sns.residplot(x = ypred, y = io2, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values

```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81cae090d0>
```

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



As one can observe this residual plot, one may notice that the lowess line has an arching hump and that the residuals are spread out in a 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. heteroskedasticity

```
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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.123				
Time:	06:03:40	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_inter

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]

```

This is the logarithmic model for the average monthly prices of energy per EUR/MWH.

In []: #Logarithmic OLS regressions
Logpricevalues = ((np.log(io2)))
LogMonthsvalues = ((np.log(Months)))
Log = np.polyfit(np.log(Months), io21, 1)
lin2 = LinearRegression()
lin2.fit(np.log(io21), Months)
Price_Log = sm.OLS(Months, io21).fit()

predictionLog = Price_Log.predict(io21)

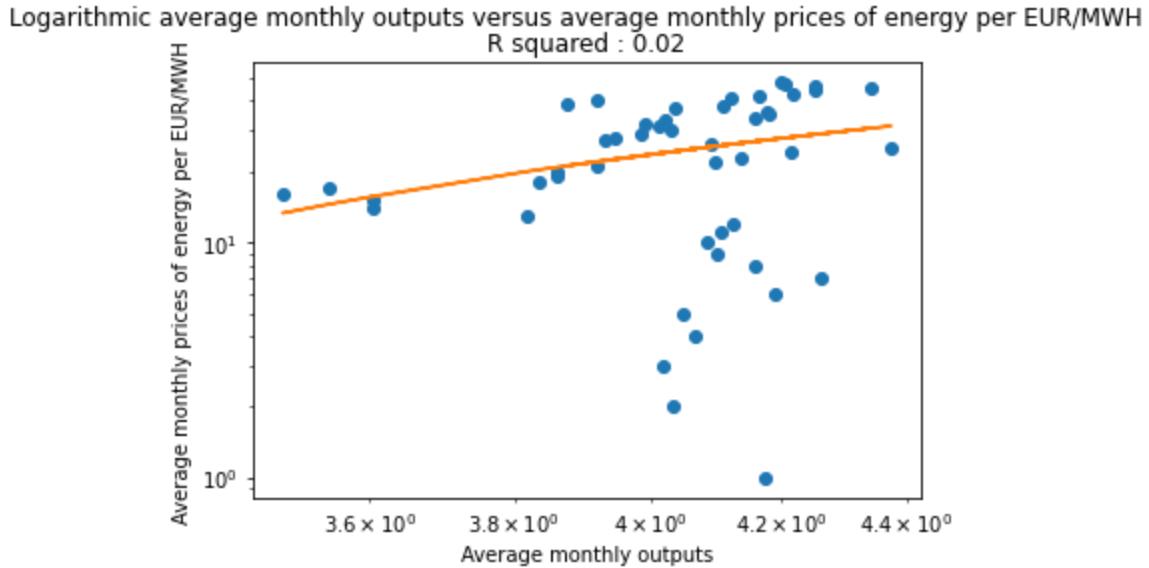
Price_Log.summary() #OLS Logarithmic summary table
#Log
Log = np.polyfit(np.log(io2), Months, 1)
print(Log)

y = 20.14460424 * Logpricevalues -56.89509997

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

```
[ 20.14460424 -56.89509997]
```

```
Out[ ]: <matplotlib.lines.Line2D at 0x7f81cb2b9f90>
```



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(predictionLog)# 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 residuals versus predictions
```

```
sns.residplot(x = predictionLog, y =standardized_residualsPriceLog, lowess =  
plt.suptitle("Price residuals from quadratic model versus predicted average
```

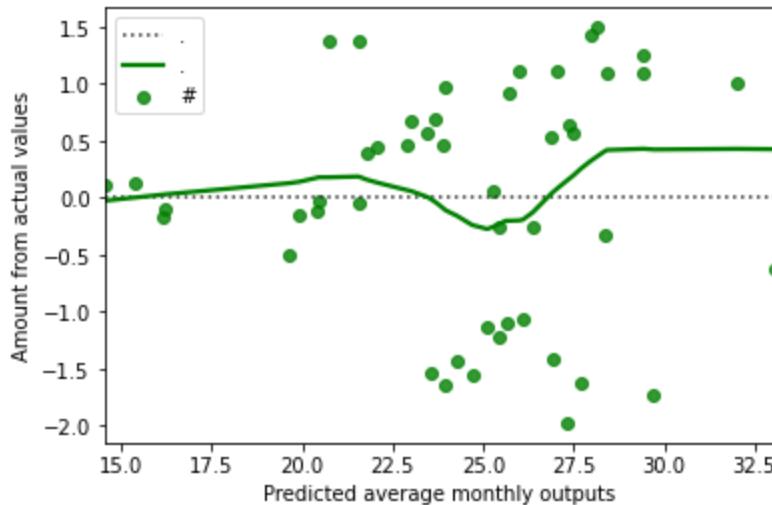
```

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 0x7f81ca21b750>

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.

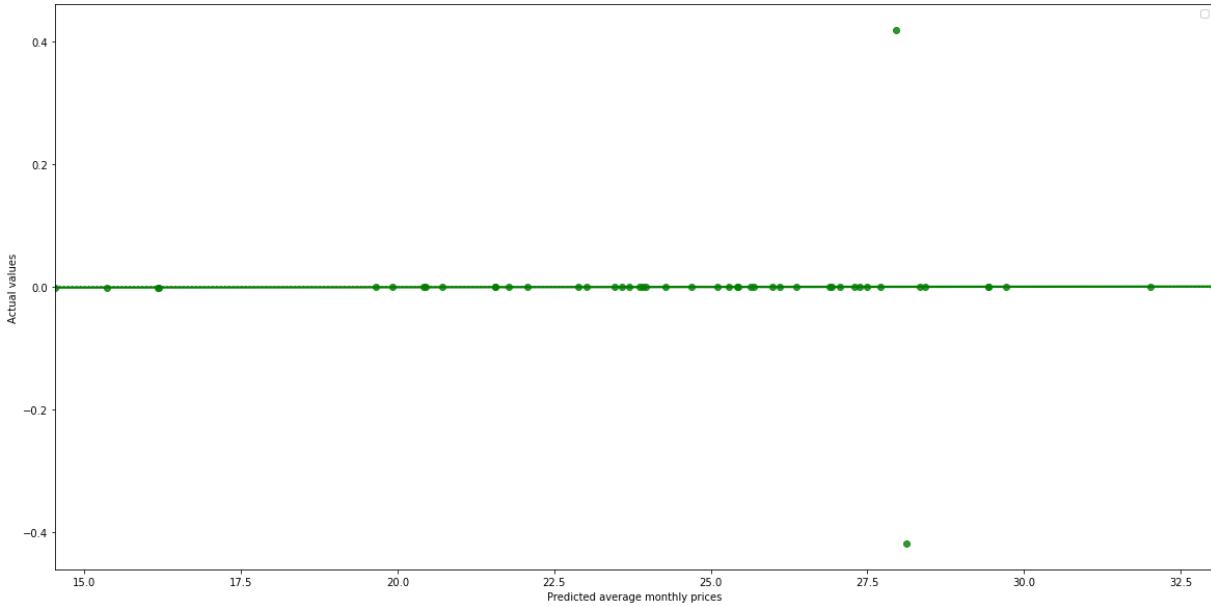
```

In [ ]: plt.suptitle("Predicted average monthly OLS logarithmic price of energy per
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionLog, y = io2, lowess = True, color="g")
#Predicted OLS average monthly logarithmic values versus actual values

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81ca40e450>

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.

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(io2, Months1).fit()
predictions = modelpricereg.predict(Months1)

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

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:	Sun, 09 Oct 2022		Prob (F-statistic):	0.0449		
Time:	06:03:41		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.

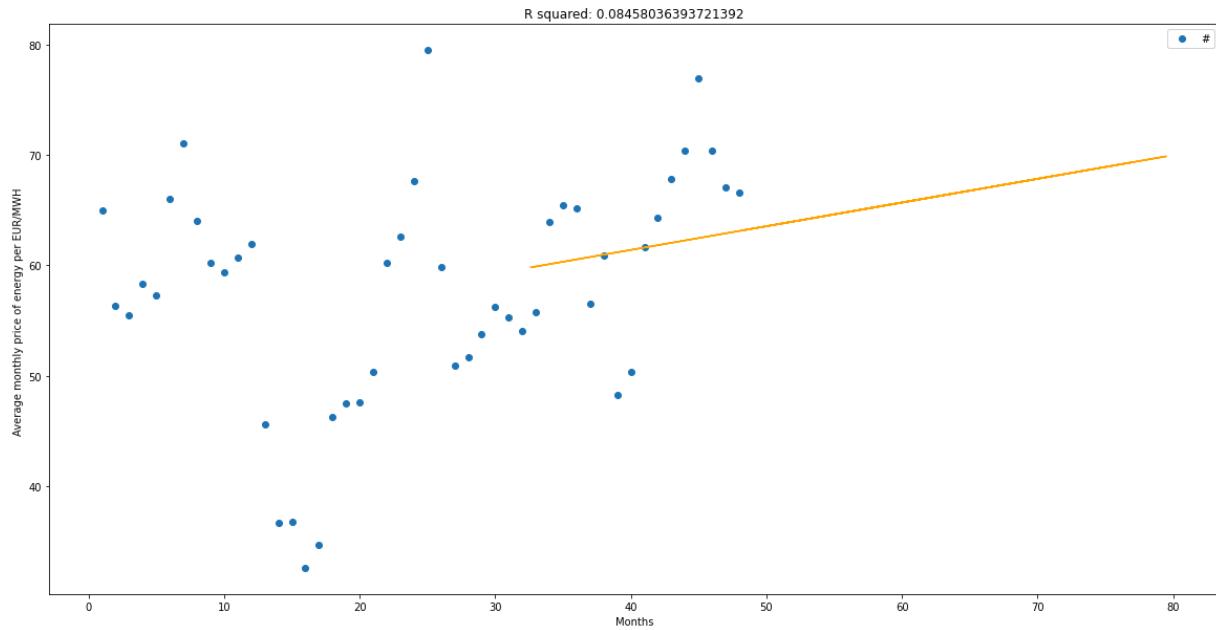
```
In [ ]: modelprice = stats.linregress ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335])
```

```
In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(Months,io2)
print("slope: %f      intercept: %f" % (slope, intercept))

# OLS linear Scatterplot
plt.suptitle("Average monthly price of energy per EUR/MWH")
plt.plot(Months, io2, "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()
```

Average monthly price of energy per EUR/MWH



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 linear 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

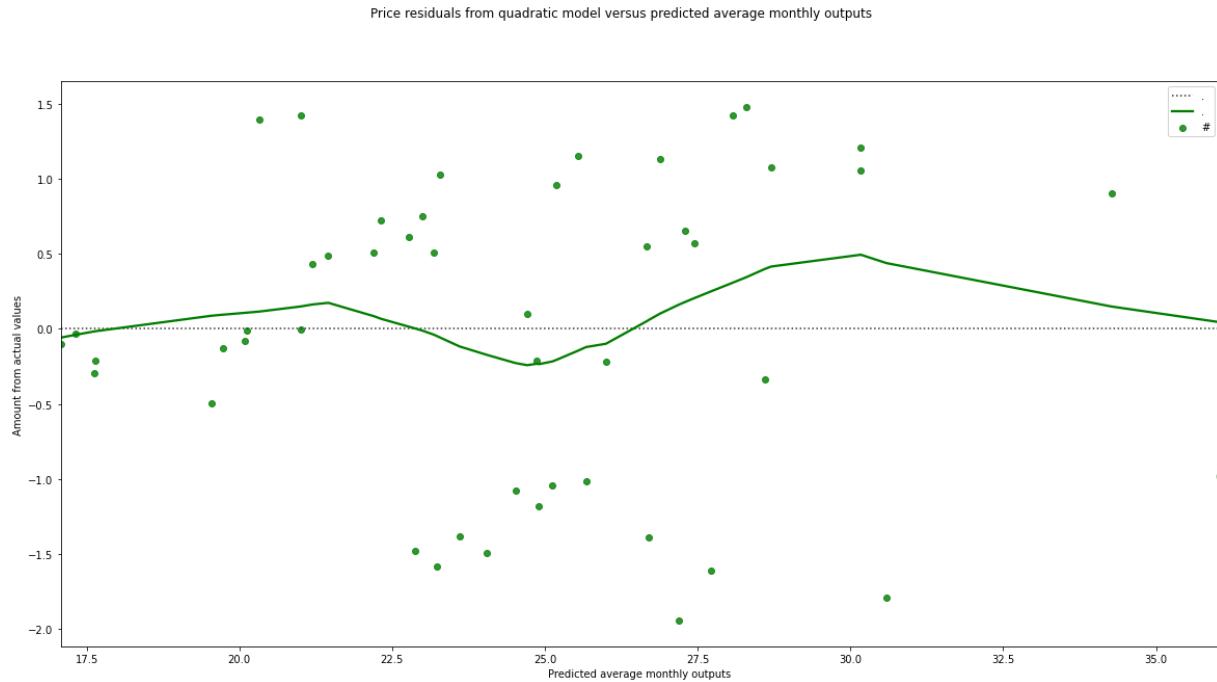
print(standardized_residualsPricereg)
```

```
[ 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,
[64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083333333335,
```

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = ypred, y = standardized_residualsPriceQuad, lowess = True,
plt.suptitle("Price residuals from quadratic model versus predicted average
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 0x7f81ca148490>
```



As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. In addition, 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.

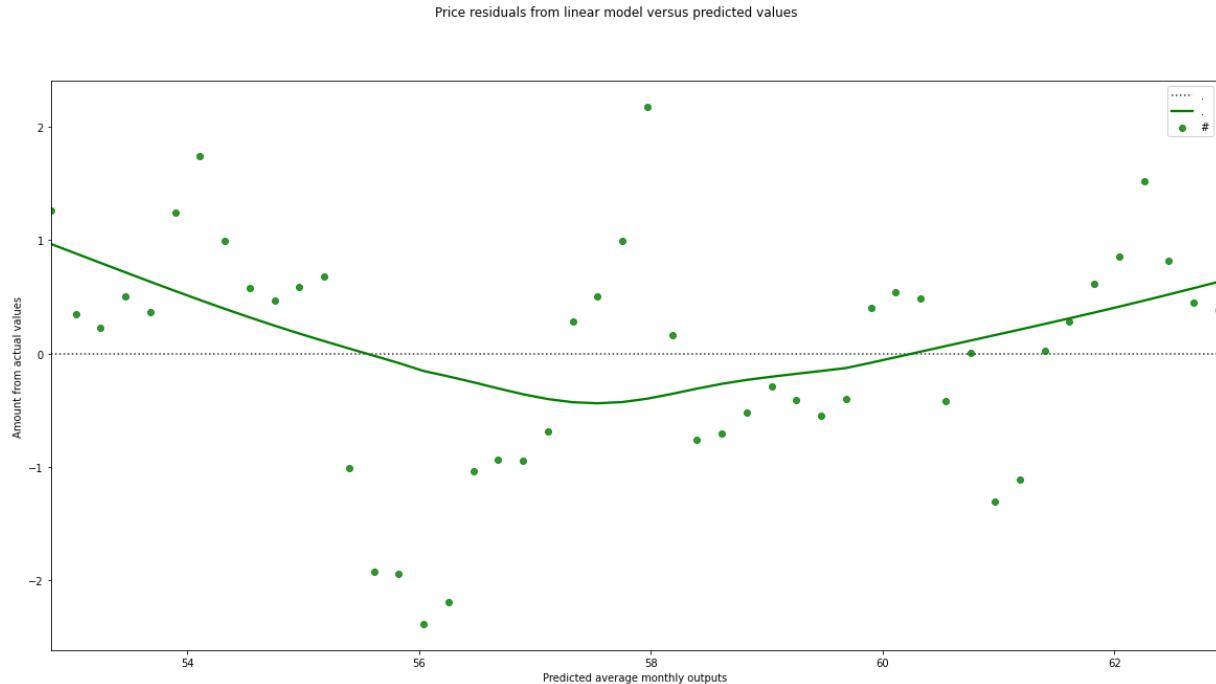
```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictions, y = standardized_residualsPricereg, lowess = 1
```

```

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 0x7f81ca0c7310>



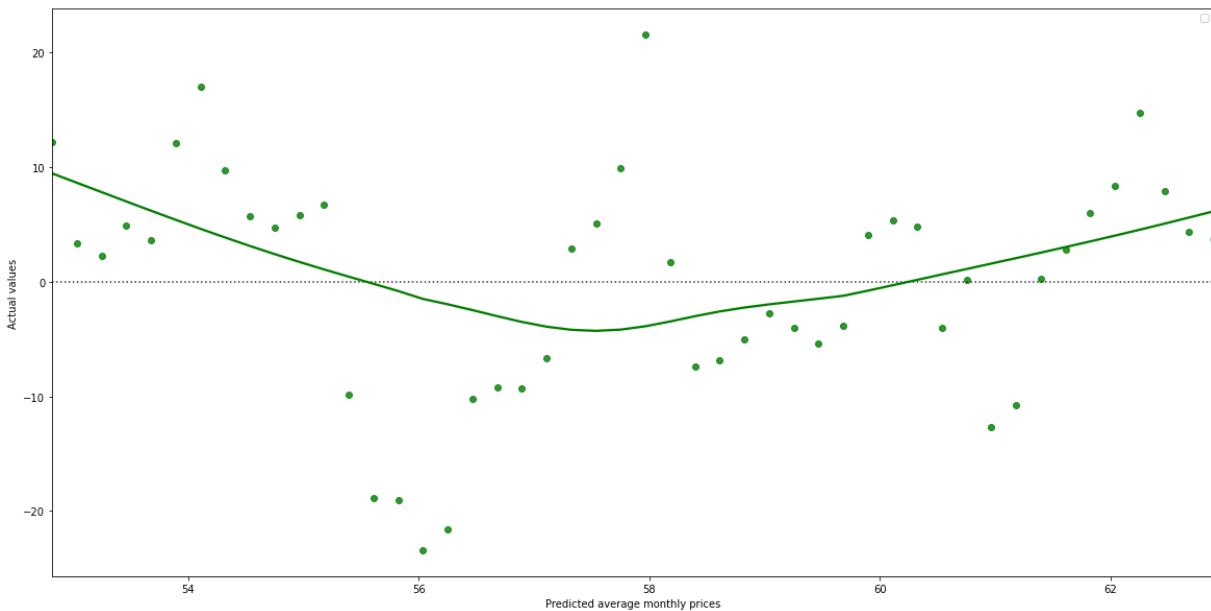
As one can observe, there is a subtle hump along the lowess line in the residual plot. In addition, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. The green dots represent the respective observations, while the dotted 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 price of energy per EUR/MWh")
plt.xlabel("Predicted average monthly prices")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictions, y = io2, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81ca02c690>



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,io2)
print("slope: %f      intercept: %f" % (slope, intercept))
```

slope: 0.214393 intercept: 52.607220

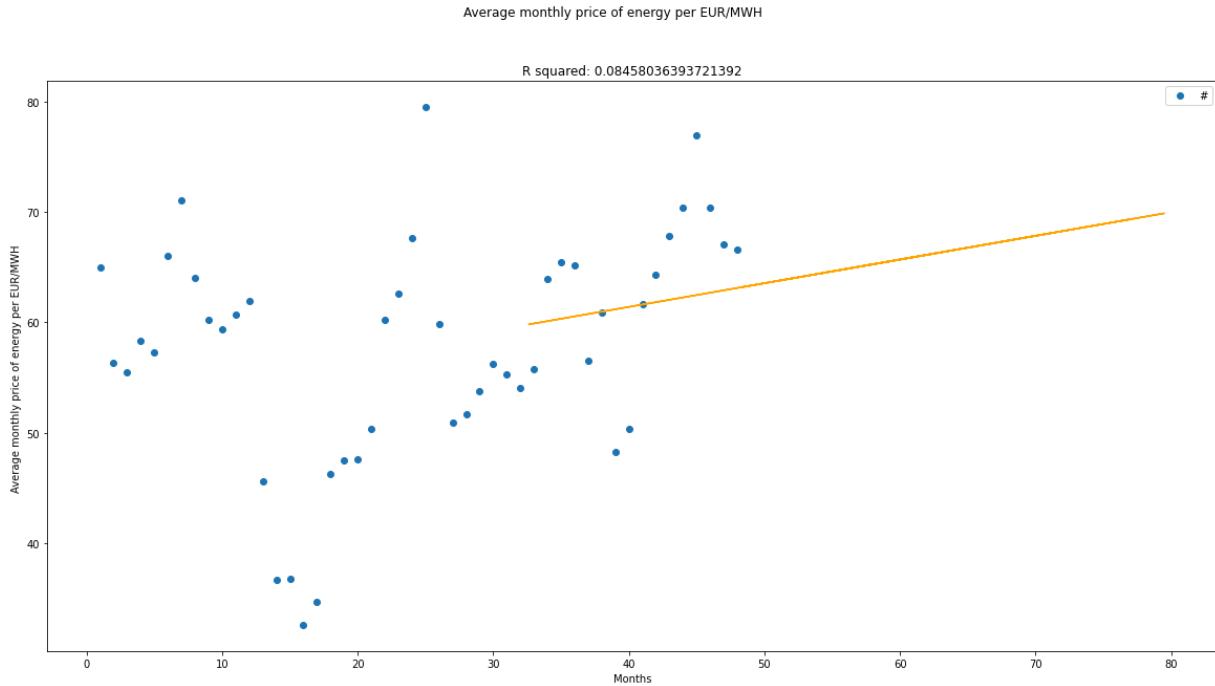
Below is a scatter graph depicting the relationship between the average monthly prices of energy per EUR/MWh.

```
In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(Months,io2)
print("slope: %f      intercept: %f" % (slope, intercept))

# OLS linear Scatterplot
plt.suptitle("Average monthly price of energy per EUR/MWh")
plt.plot(Months, io2, "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()
```

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.



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 linear model of best fit.

In []:

Below are the prices of energy per EUR/MWH.

```
In [ ]: print(io2)
```

```
[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, 62.5810555555556, 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, 50.4007361111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 70.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 66.62351388888889]
```

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,
print(df_Price)
df_nuclear= pd.DataFrame.from_dict(dfnuclear, orient = "columns")
print(df_Price)
```

```
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")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary")

adfuller_test(df_Price['Price'])

test_result=adfuller(df_Price['Price'])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_Price["Price"])
plt.suptitle("Autocorrelations of Price")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function
plot_pacf(df_Price["Price"])
plt.suptitle("Partialautocorrelations of Price")
plt.ylabel('Partialautocorrelations')
```

```

plt.xlabel('Lags')
plt.show()
Price_Autocorrelations = sm.tsa.acf(df_Price["Price"], fft=False) #Autocorrelation
print(Price_Autocorrelations)

The plots above graphically summarize the strength of a relationship with an
lag. As one can observe, there is statistical significance in the partial autocorrelation
coefficients at lags 1 and 2. This indicates that there is a strong linear relationship
between consecutive observations. We can also see that the partial autocorrelation
coefficients decrease rapidly towards zero, which suggests that the data is stationary.
df_Price['First Difference Price'] = df_Price["Price"]- df_Price["Price"].shift(1)
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

The blue line represents the trend line among the values themselves. As one can observe, the
This bell shaped curve is slightly skewed to the right. Hence, it has a asymmetric distribution.
If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is greater than 0.5, the data is skewed to the right. If the skewness is less than -0.5, the data is skewed to the left.
#Bell Curves

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

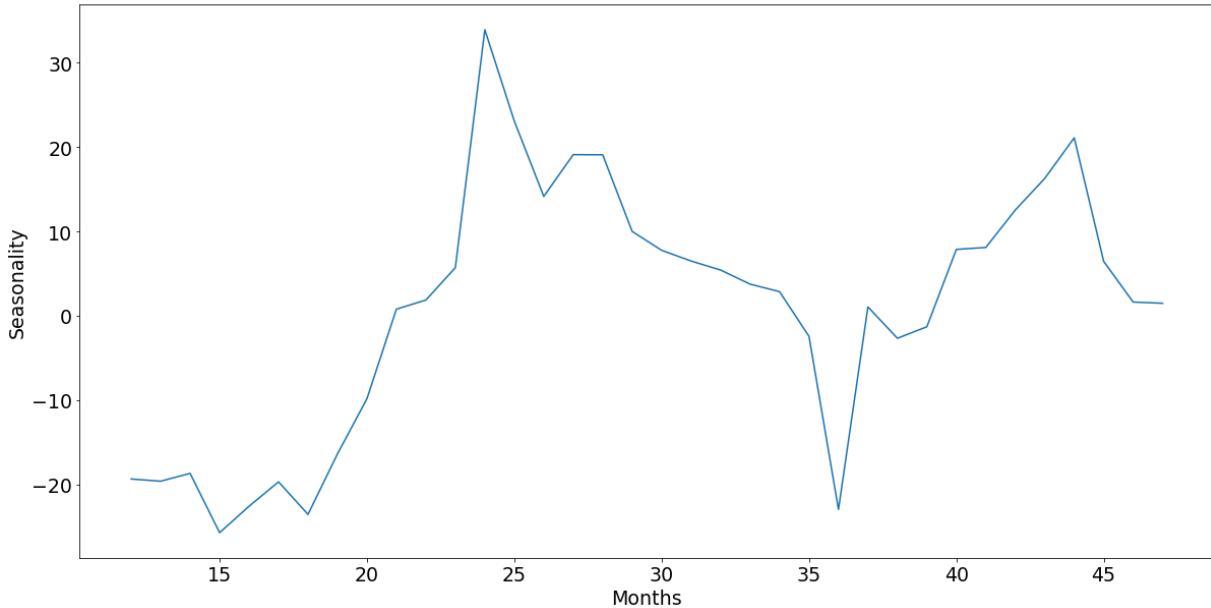
PriceResultspdf = stats.norm.pdf(df_Price["Price"].sort_values(), PriceResults_mean, PriceResults_std)

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 standard deviation units)")
plt.title(f'Skewness for data: {skew(df_Price["Price"])}')
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81de976450>

Seasonal Difference of 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")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary")

adfuller_test(df_Price['Price'])

test_result=adfuller(df_Price['Price'])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_Price["Price"])
plt.suptitle("Autocorrelations of Price")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function
plot_pacf(df_Price["Price"])
plt.suptitle("Partialautocorrelations of Price")
plt.ylabel('Partialautocorrelations')
plt.xlabel('Lags')
plt.show
Price_Autocorrelations = sm.tsa.acf(df_Price["Price"], fft=False) #Autocorrelation Function
print(Price_Autocorrelations)
```

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 average monthly prices of energy per EUR/MWH.

```
In [ ]: df_Price['First Difference Price'] = df_Price["Price"] - df_Price["Price"].shift(1)
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
```

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)

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 standard deviation units)")
plt.title(f'Skewness for data: {skew(df_Price["Price"])}')
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 represents 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')
```

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

```
In [ ]: Rounded_Y = [round(item, 2) for item in io2]
        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 values. A consideration to keep in mind when reading this analysis is that the conducted plots and tests are in no particular order. That being said, the analysis begins below this paragraph.

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, 55.52246298788,
                           "Fossil_Oil" : [306.0204638472033, 319.2395833333333, 319.3337819650067, 33
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, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Fossil_Oil': [306.0204638472033, 319.2395833333333, 319.3
337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963839, 36
0.23387096774195, 323.9139784946237, 343.491666666667, 323.39112903225805,
346.06388888888887, 330.6514131897712, 337.46505376344084, 297.2543103448275
6, 294.05248990578735, 259.8875, 277.9153225806452, 289.8291666666665, 286.
6900269541779, 283.30645161290323, 281.386111111111, 276.9959731543624, 27
1.569444444444446, 276.63978494623655, 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, 295.900297619047
6, 288.1265141318977, 257.6291666666666, 280.4502688172043, 285.12083333333
334, 281.5450874831763, 283.3333333333333, 296.136111111111, 291.2993288590
604, 269.02150537634407, 272.98333333333335], 'Dates': ['2015-01', '2015-0
2', '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', '201
6-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '20
17-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2
018-01', '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, 4.51143997994741
3, 5.056260079995415, 4.8606176427379415, 5.967858344556159, 5.1116151577371
7, 4.550270065632229, 4.4309103302219315, 4.153596865402372, 4.0266853865512
91, 3.8502216850665367, 4.139996585263025, 4.012694543070627, 4.097402214308
303]
```

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

This is the logarithmic model for the average monthly outputs of fossil oil versus the average monthly prices of energy per EUR/MWH.

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

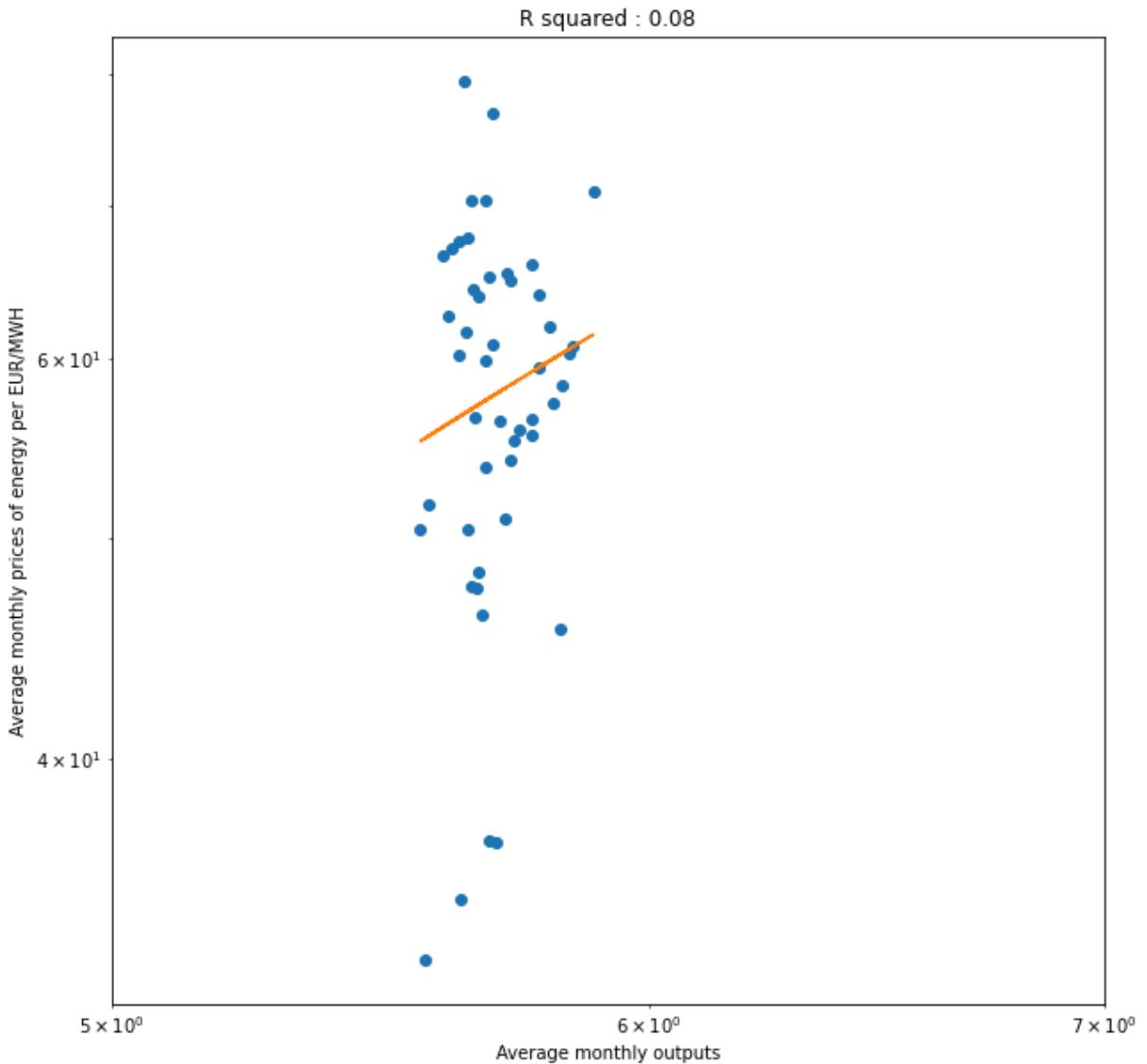
```
[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.7813370473537  
6, 332.56720430107526, 319.2294853963839, 360.23387096774195, 323.9139784946  
237, 343.49166666666667, 323.39112903225805, 346.06388888888887, 330.65141318  
97712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875,  
277.9153225806452, 289.82916666666665, 286.6900269541779, 283.3064516129032  
3, 281.3861111111111, 276.9959731543624, 271.56944444444446, 276.63978494623  
655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.49027777  
77778, 291.9206989247312, 299.3930555555556, 308.2002688172043, 306.23521505  
376345, 310.8013888888889, 287.4187919463087, 303.54305555555555, 293.926075  
2688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916  
666666666, 280.4502688172043, 285.1208333333334, 281.5450874831763, 283.33  
333333333, 296.1361111111111, 291.2993288590604, 272.98333333333335, 269.02  
150537634407]
```

```
In [ ]: #Logarithmic OLS regressions  
Logpricevalues = ((np.log(io2)))  
LogFossilOilvalues = ((np.log(fossil_oil)))  
Log = np.polyfit(np.log(io2), fossil_oill, 1)  
lin2 = LinearRegression()  
lin2.fit(np.log(fossil_oill), io2)  
FossilOil_Log = sm.OLS(io2, fossil_oill).fit()  
  
FossilOil_Logpred = FossilOil_Log.predict(fossil_oill)  
  
FossilOil_Log.summary() #OLS Logarithmic summary table  
#Log  
Log = np.polyfit(np.log(fossil_oil), io2, 1)  
print(Log)  
  
y = 18.80220214 * LogFossilOilvalues - 49.22000295  
  
#Logarithmic OLS regression scatterplot  
plt.suptitle("Logarithmic average monthly outputs versus average monthly pri  
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, io2, "o")  
plt.plot(LogFossilOilvalues, y)  
  
plt.xlim([5, 7])
```

[18.80220214 -49.22000295]

Out[]: (5, 7)

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



The blue dots represent the observations and the orange line is the linear 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 linear model of best fit.

```
In [ ]: Fossiloil_Log.summary()
```

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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.333			
Time:	06:03:43	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

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
```

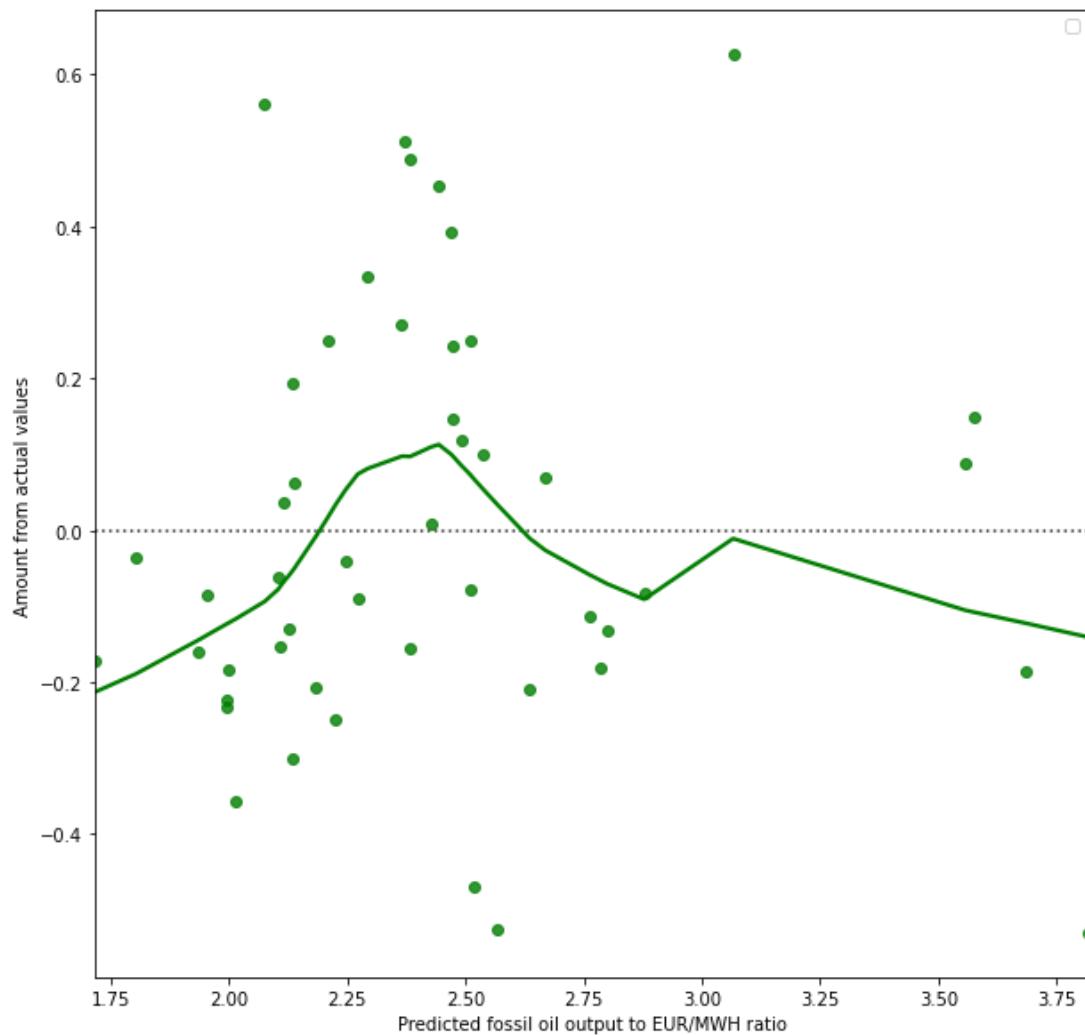
```
[58.33094074 59.14007647 59.14584232 60.3362175 59.95585352 59.13945838  
61.64931643 59.42619387 60.62453443 59.39419052 60.78197887 59.8385889  
60.25564845 57.79436884 57.598387 55.50716496 56.61063901 57.33987949  
57.14773431 56.94062734 56.82308425 56.55436609 56.22221097 56.53256399  
56.70903493 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 [ ]:
```

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

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c9f24e10>
```

Predicted average monthly logarithmic output to price of energy per EUR/MWH ratio versus actual ratio values



In []:

As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. In addition, the residuals are quite spread out in no particular pattern which indicates 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.

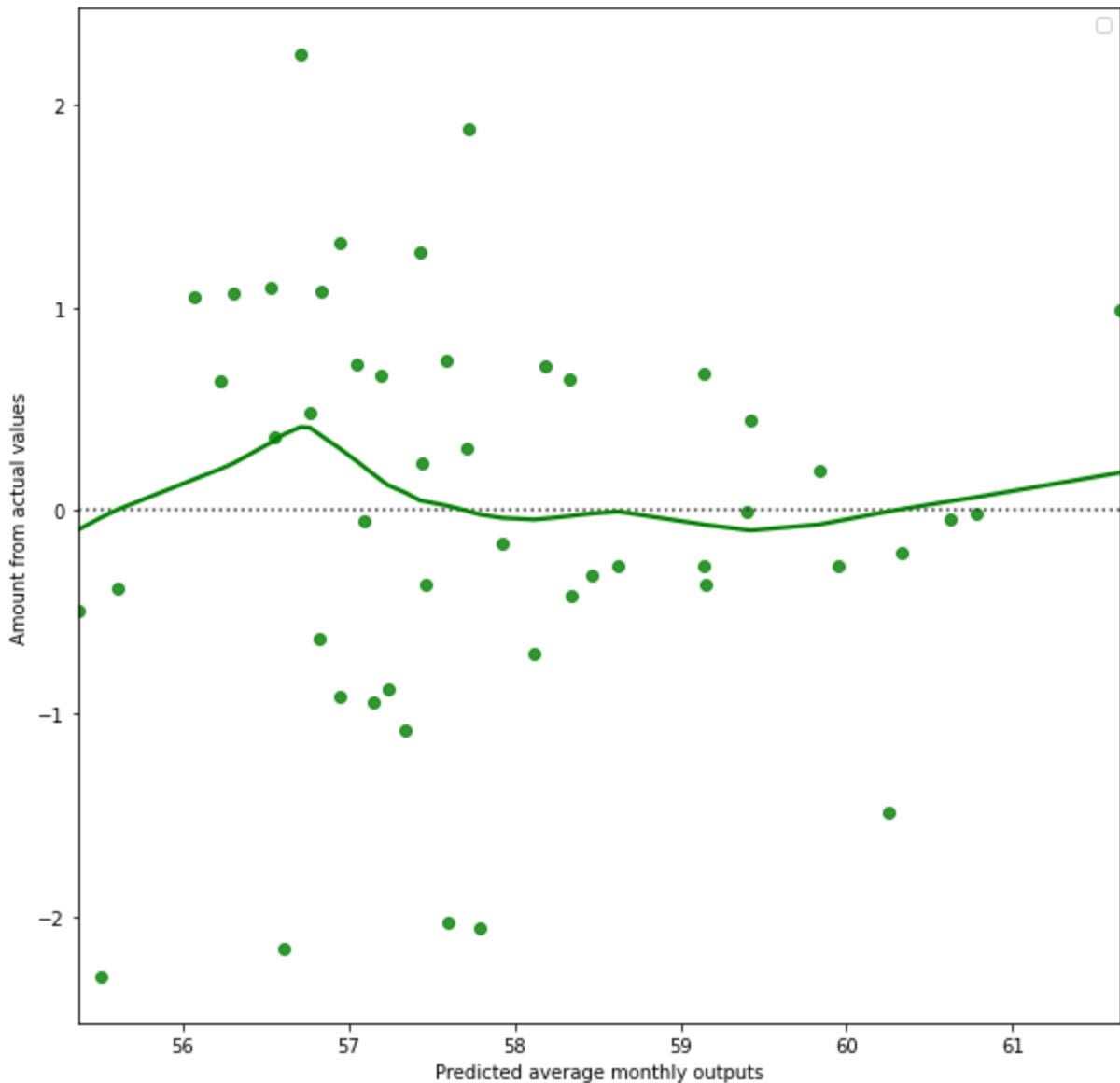
In []: # OLS Logarithmic average monthly predictions versus residuals

```
plt.suptitle("Output residuals from logarithmic model versus predicted avera  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Amount from actual values")
```

```
plt.legend(..#")
sns.residplot(x = FossilOil_Logpred, y = standardized_residualsFossilOilLog,
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c9d94fd0>
```

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



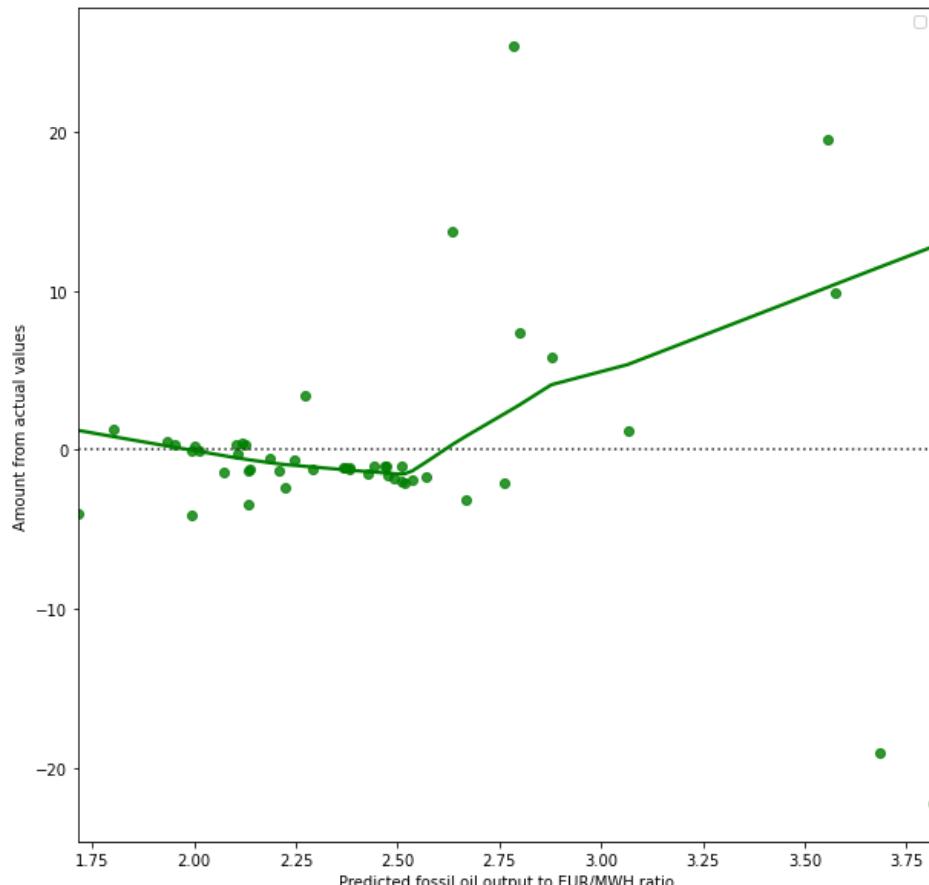
As one can observe, there is a decreasing trend in the variance of the predicted values versus the actual values. This indicates that there is heteroskedasticity but no bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green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

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
plt.suptitle("Predicted average monthly logarithmic fossil oil output to pri
plt.xlabel("Predicted fossil oil output to EUR/MWH ratio")
```

```
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = FossilOilLogRatioPredict, y = standardized_residualsFossil
```

Out []: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c9d137d0>

Predicted average monthly logarithmic fossil oil output to price of energy per EUR/MWH ratio 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.

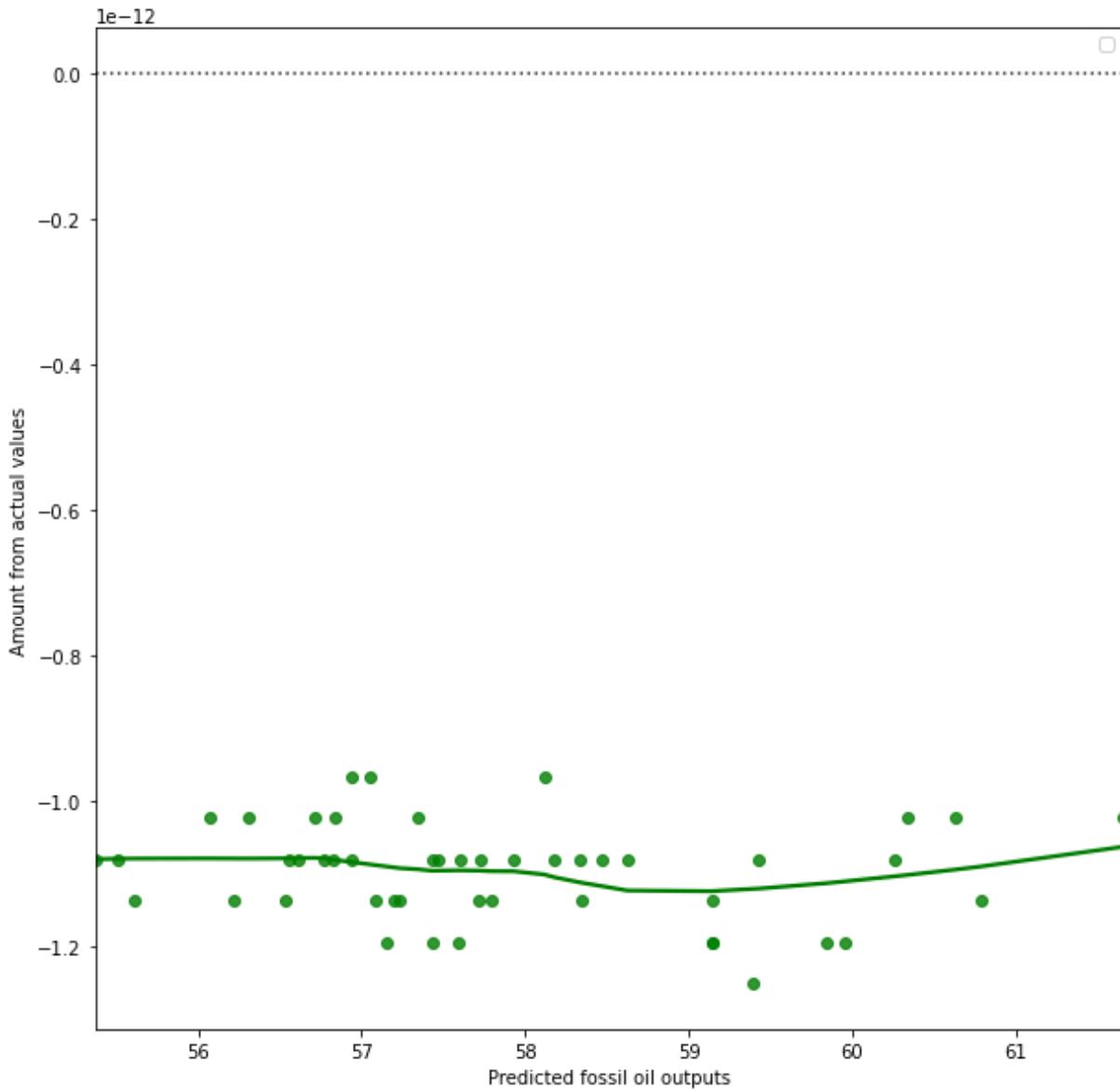
In []:

```
plt.suptitle("Predicted average monthly logarithmic fossil oil outputs per E
plt.xlabel("Predicted fossil oil outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = FossilOil_Logpred, y = fossil_oil, lowess = True, color="
```

```
# OLS predicted logarithmic average monthly values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c9e039d0>
```

Predicted average monthly logarithmic fossil oil outputs per EUR/MWH versus actual values



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 [ ]:
```

```
In [ ]: modelFossilOil = stats.linregress([306.0204638472033, 319.2395833333333, 319  
[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,  
79.49208333333331,  
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] )
```

Tn []:

```
In [ ]: modelFossilOil = stats.linregress([306.0204638472033, 319.2395833333333, 319.  
[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,
36.752083333333374,
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] )
```

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(io2, 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			
Model:	OLS	Adj. R-squared:	-0.001			
Method:	Least Squares	F-statistic:	0.9576			
Date:	Sun, 09 Oct 2022	Prob (F-statistic):	0.333			
Time:	06:03:44	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,io2)
print("slope: %f      intercept: %f" % (slope, intercept))

# OLS linear Scatterplot

plt.plot(fossil_oil,io2, "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 outputs versus average monthly prices of energy")
plt.ylabel('Average monthly price of energy per EUR/MWh ')
```

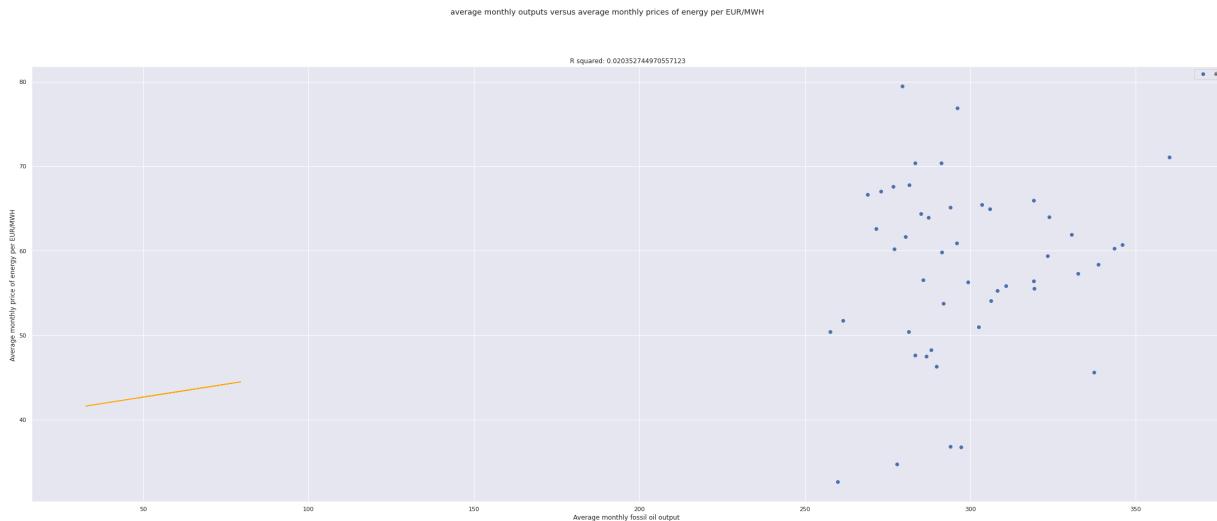
```

plt.xlabel('Average monthly fossil oil output')

plt.show()

```

There **is** a very weak yet positive correlation between the average monthly ou



```

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_ir

print(standardized_residualsFossilOil)

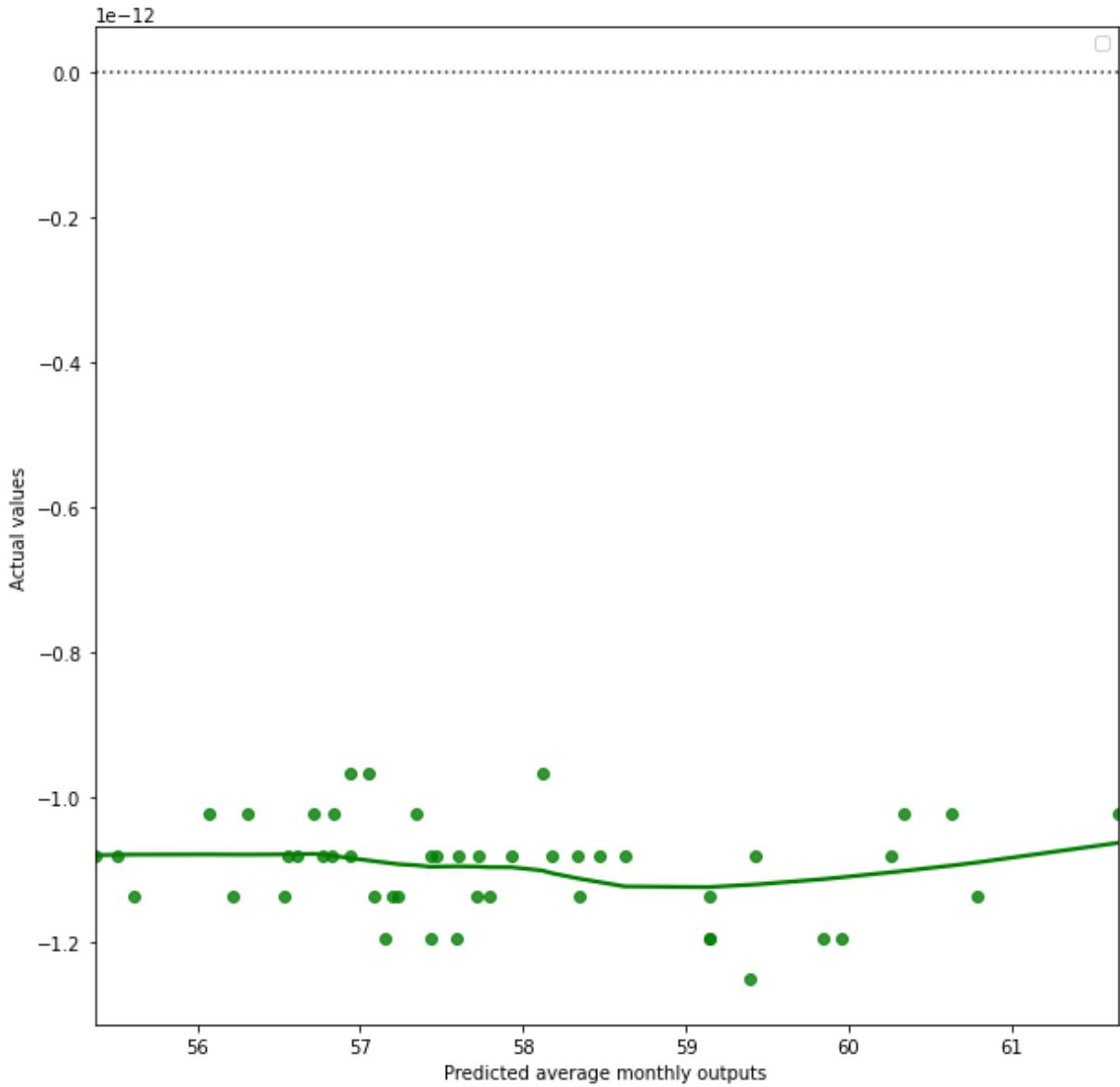
```

```
[ 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 outputs per EUR/MWh")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#")
sns.residplot(x = predictionsFossilOil, y = fossil_oil, lowess = True, color
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c9c82c10>
```

Predicted average monthly linear fossil oil outputs per EUR/MWH versus actual values



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.

This is the quadratic model used for the average monthly fossil oil outputs versus the 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, io2, 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(io2, fossil_oil2).fit()

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

#OLS Quadratic Summary Table
FossilOil_Quad.summary()
```

$$\begin{aligned} & 2 \\ & -0.0003178 x^2 + 0.2555 x + 10.1 \end{aligned}$$

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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.276			
Time:	06:03:45	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.

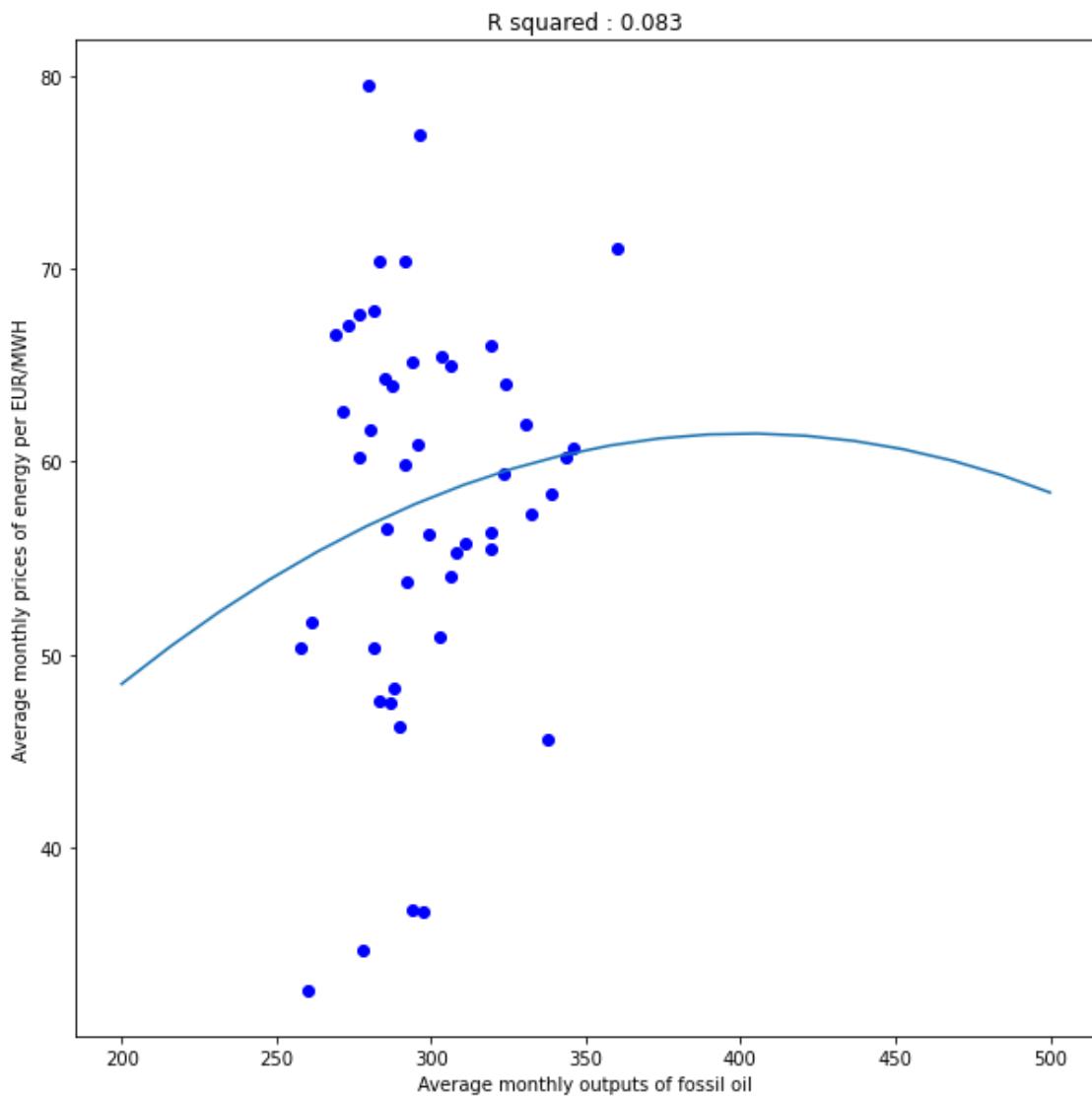
[2] The smallest eigenvalue is 2.71e-59. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

In []:

```
#Quadratic Scatterplots
polyline = np.linspace(start = 200, stop = 500 , num = 20)
plt.plot(polyline, modelFossilOilquad(polyline))
```

```
plt.scatter(fossil油,io2, color = 'blue')
plt.title("R squared : 0.083")
plt.suptitle('Quadratic for average monthly outputs of fossil oil versus ave')
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 average monthly prices of energy per EUR/MWH



The blue dots represent the observations and the blue line is the quadratic 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 linear model of best fit.

In []:

```
In [ ]: 
```

```
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 [ ]: influenceFossilOilQuad = FossilOil_Quad.get_influence() #Quadratic OLS residuals
```

```
standardized_residualsFossilOilQuad = influenceFossilOilQuad.resid_studentized
```

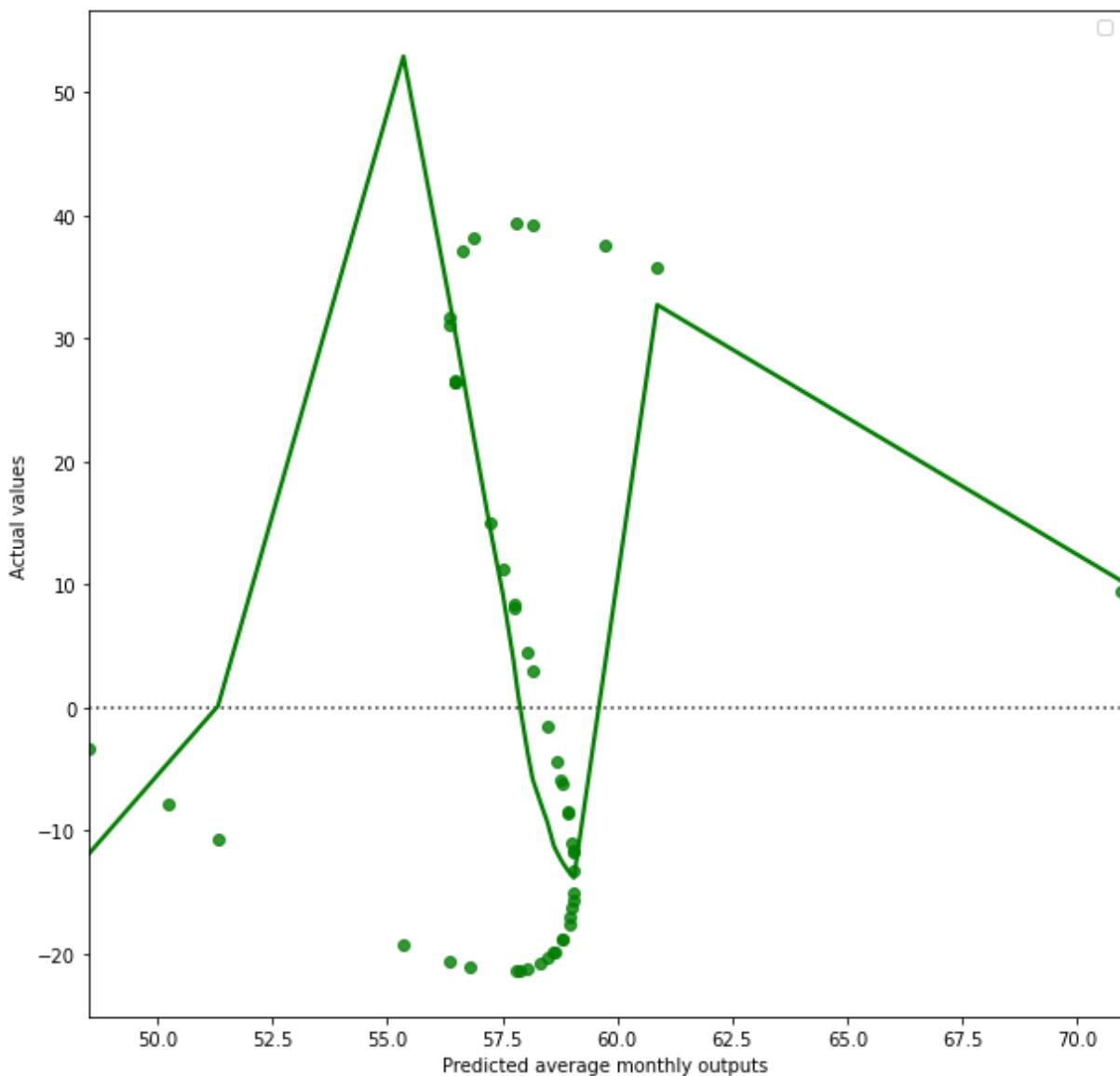
```
print(standardized_residualsFossilOilQuad)
```

```
[ 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 [ ]: plt.suptitle("Predicted average monthly quadratic fossil oil outputs per EUF")  
plt.xlabel("Predicted average monthly outputs")  
plt.ylabel("Actual values")  
plt.legend(..#)  
sns.residplot(x = FossilOil_ypred, y = fossil_oil, lowess = True, color="g")  
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c9a9cb90>
```

Predicted average monthly quadratic fossil oil outputs 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 "S" 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.

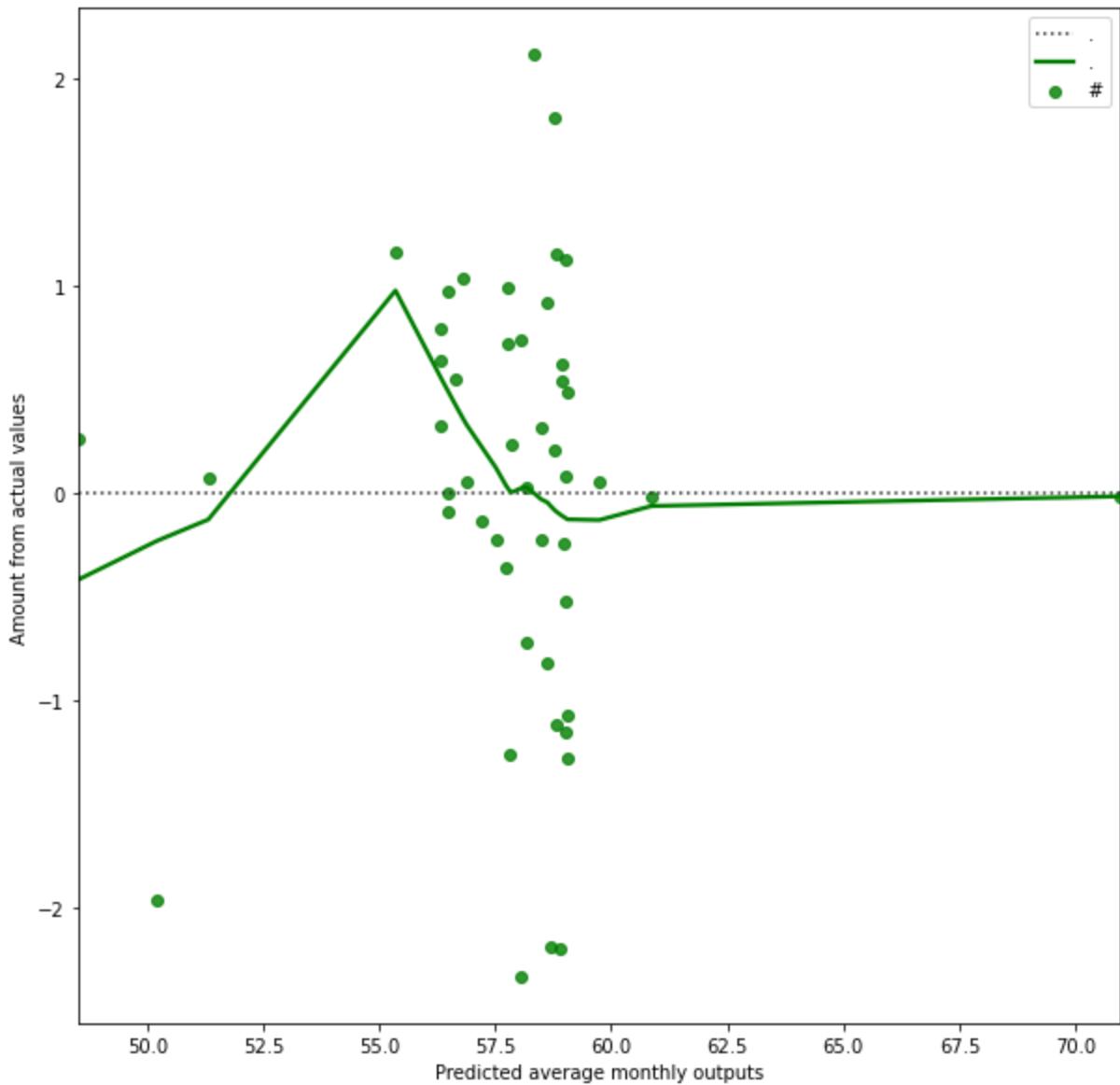
In []:

```
In [ ]: #Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = FossilOil_ypred, y = standardized_residualsFossilOilQuad,
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 0x7f81c9a23410>

Fossil oil output residuals from quadratic model versus predicted values



Due to the sudden and punctual hump in the lowess line, there is indication that there is heteroskedasticity and bias in this plot. 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 [ ]: modelFossilOilregpredictions = stats.linregress([predictionsFossilOil],[foss
/usr/local/lib/python3.7/dist-packages/scipy/stats/_stats_mstats_common.py:1
84: RuntimeWarning: invalid value encountered in sqrt
t = r * np.sqrt(df / ((1.0 - r + TINY)*(1.0 + r + TINY))))
```

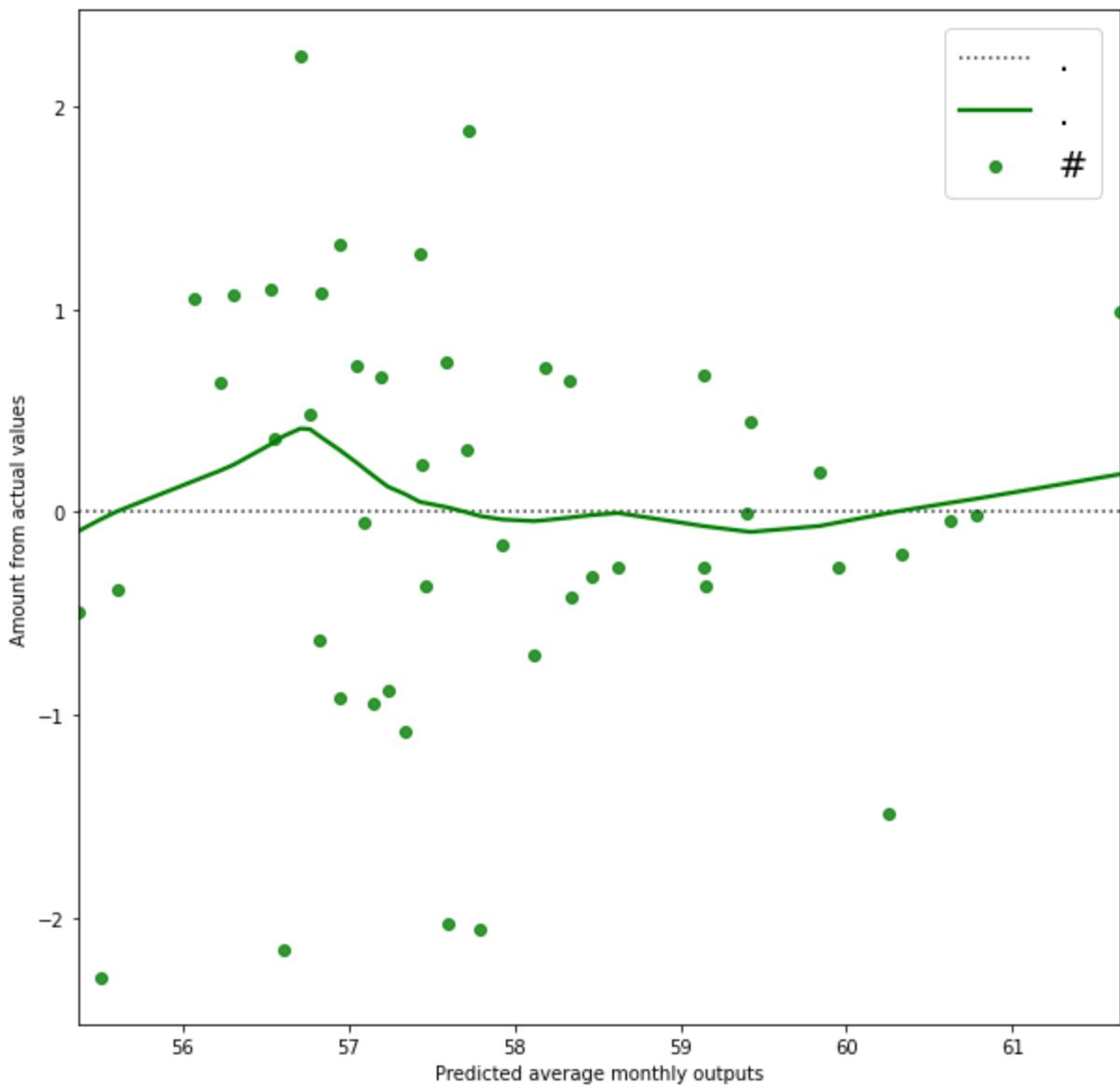
```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionsFossilOil, y =standardized_residualsFossilOil,
plt.rcParams["figure.figsize"] = [20, 10]

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

```
/usr/local/lib/python3.7/dist-packages/scipy/stats/_stats_mstats_common.py:1
84: RuntimeWarning: invalid value encountered in sqrt
t = r * np.sqrt(df / ((1.0 - r + TINY)*(1.0 + r + TINY))))
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c99c1990>
```

Fossil oil output residuals from linear model versus predicted values



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, 55.52246298788,
    "Fossil_Oil" : [306.0204638472033, 319.2395833333333, 319.3337819650067, 33
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")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary")

adfuller_test(df_FossilOil["Ratio"])

test_result=adfuller(df_FossilOil["Ratio"])

#Autocorrelation Plot
from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_FossilOil["Ratio"])
plt.suptitle("Autocorrelations of Fossil Oil Ratios")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')
plt.show
#Partial autocorrelation Function
from statsmodels.graphics.tsaplots import plot_pacf # Partialautocorrelation Function
plot_pacf(df_FossilOil["Ratio"])
plt.suptitle("Autocorrelations of Fossil Oil Ratios")
plt.ylabel('Partialautocorrelations')
plt.xlabel('Lags')
plt.show
```

```

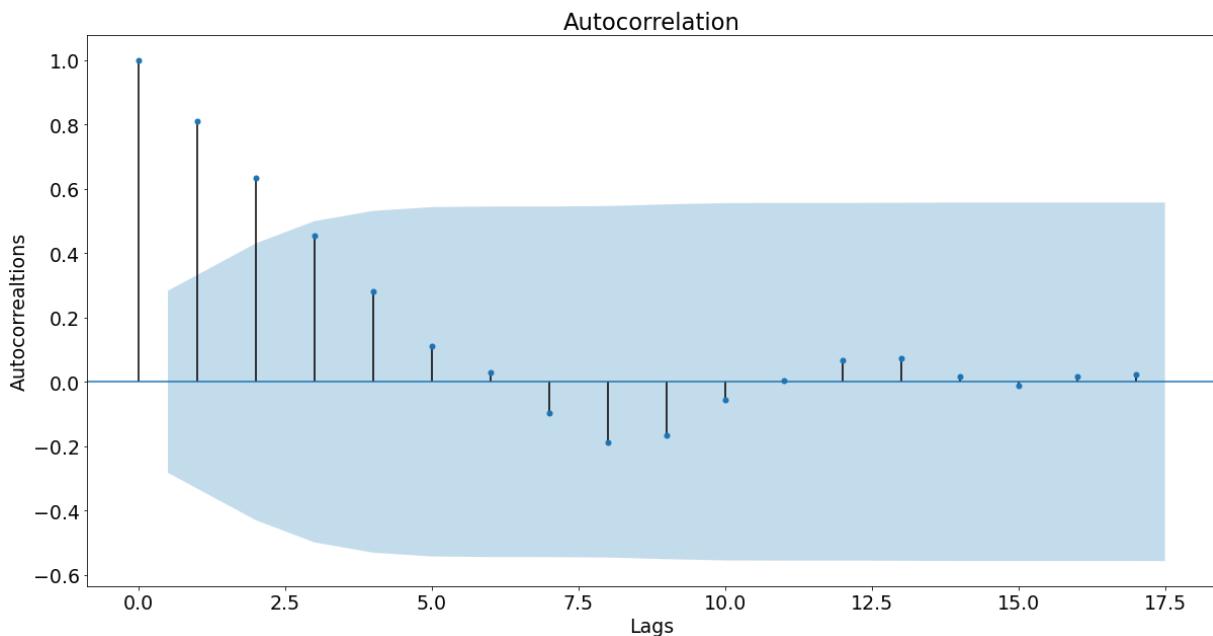
{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Fossil_Oil': [306.0204638472033, 319.2395833333333, 319.3
337819650067, 338.78133704735376, 332.56720430107526, 319.2294853963839, 36
0.23387096774195, 323.9139784946237, 343.491666666667, 323.39112903225805,
346.06388888888887, 330.6514131897712, 337.46505376344084, 297.2543103448275
6, 294.05248990578735, 259.8875, 277.9153225806452, 289.8291666666665, 286.
6900269541779, 283.30645161290323, 281.386111111111, 276.9959731543624, 27
1.569444444444446, 276.63978494623655, 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, 295.900297619047
6, 288.1265141318977, 257.6291666666666, 280.4502688172043, 285.12083333333
334, 281.5450874831763, 283.3333333333333, 296.136111111111, 291.2993288590
604, 269.02150537634407, 272.98333333333335], 'Dates': ['2015-01', '2015-0
2', '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', '201
6-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '20
17-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2
018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07',
'2018-08', '2018-09', '2018-10', '2018-11', '2018-12']}

```

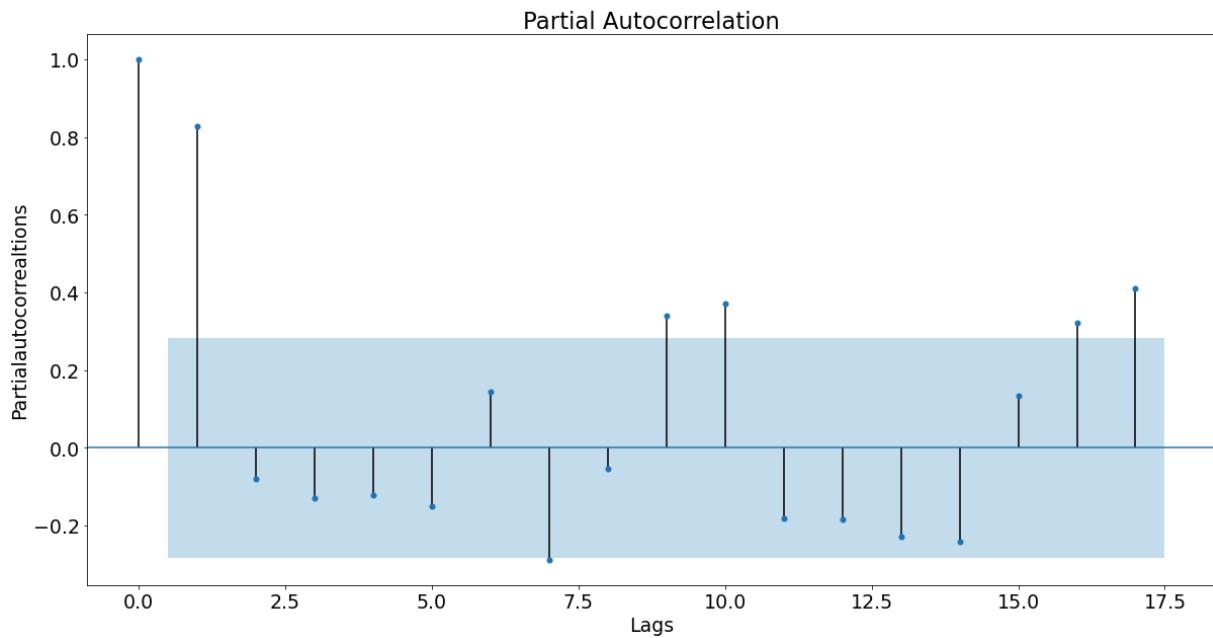
	Price	Fossil_Oil	Dates
0	64.949019	306.020464	2015-01
1	56.383854	319.239583	2015-02
2	55.522463	319.333782	2015-03
3	58.354083	338.781337	2015-04
4	57.294059	332.567204	2015-05
5	65.974903	319.229485	2015-06
6	71.072043	360.233871	2015-07
7	63.998065	323.913978	2015-08
8	60.254792	343.491667	2015-09
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, 4.51143997994741
3, 5.056260079995415, 4.8606176427379415, 5.967858344556159, 5.1116151577371
7, 4.550270065632229, 4.4309103302219315, 4.153596865402372, 4.0266853865512
91, 3.8502216850665367, 4.139996585263025, 4.012694543070627, 4.097402214308
303]
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 Fossil Oil Ratios



Autocorrelations of Fossil Oil Ratios



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 [ ]: Fossil_Oil_Ratio_Autocorrelations = sm.tsa.acf(df_FossilOil["Ratio"], fft=False)
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 a
n integer to silence this warning.
FutureWarning,
```

```
In [ ]: df_FossilOil['First Difference Fossil Oil Ratio'] = df_FossilOil["Ratio"]- c
df_FossilOil['Seasonal Difference Fossil Oil Ratio']=df_FossilOil["Ratio"]-
df_FossilOil.head()
```

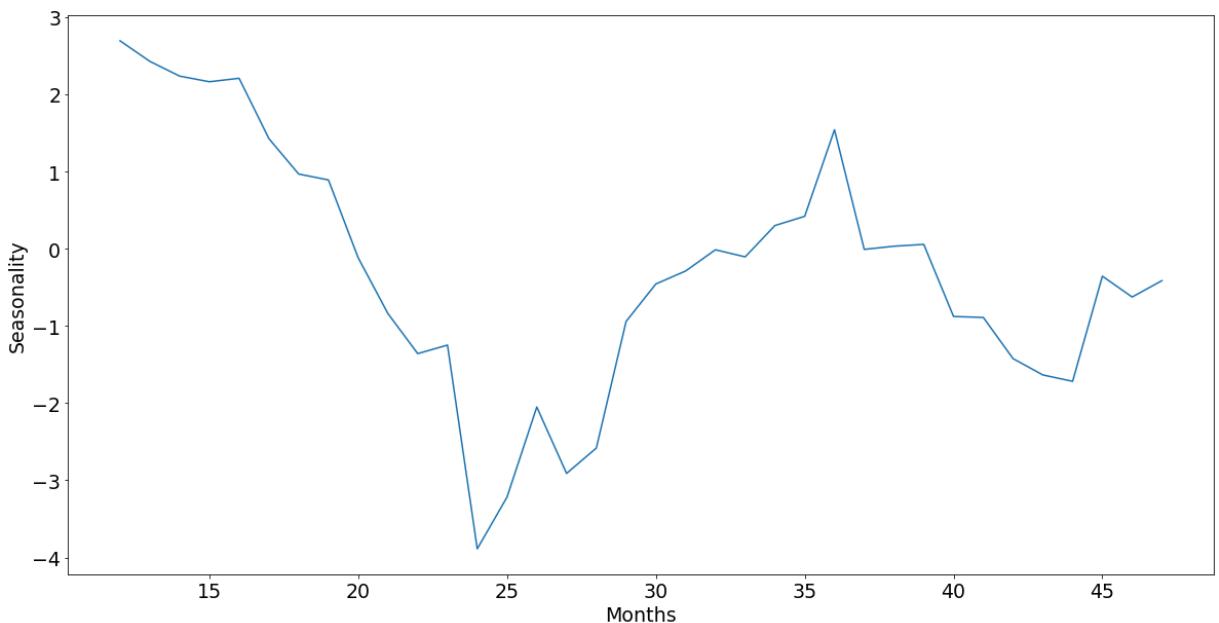
Out[]:

	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	NaN
1	56.383854	319.239583	2015-02	5.661897	0.950195	NaN
2	55.522463	319.333782	2015-03	5.751434	0.089537	NaN
3	58.354083	338.781337	2015-04	5.805615	0.054181	NaN
4	57.294059	332.567204	2015-05	5.804567	-0.001048	NaN

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

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c7028490>

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")
    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"])

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

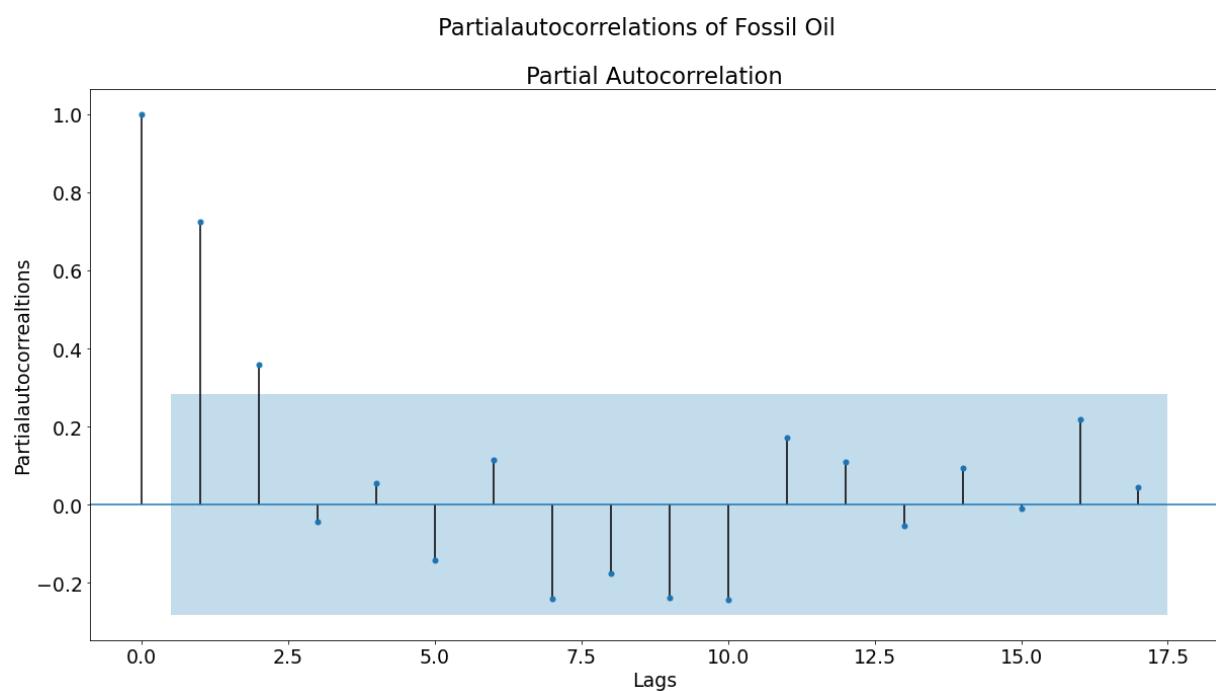
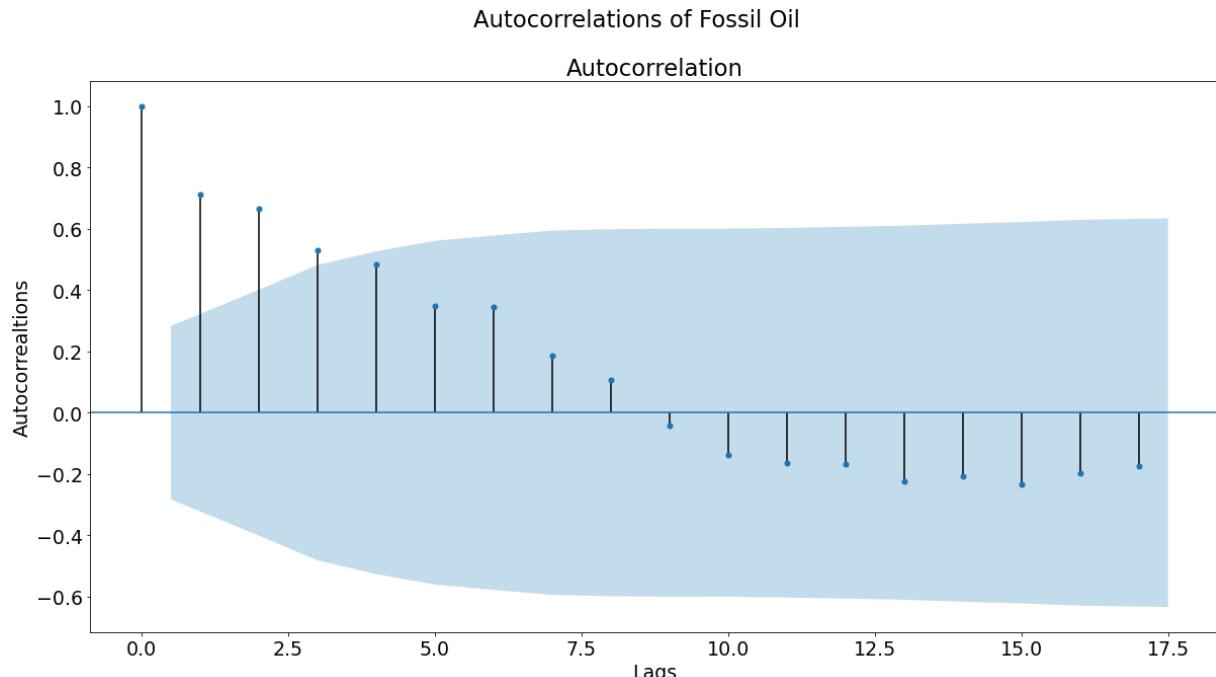
```

plot_pacf(df_FossilOil["Fossil_Oil"])
plt.suptitle("Partialautocorrelations of Fossil Oil")
plt.ylabel('Partialautocorrealtions')
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[]: <function matplotlib.pyplot.show(*args, **kw)>



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 []:

Out[]:

	Price	Fossil_Oil	Dates	Ratio	First Difference Fossil Oil Ratio	Seasonal Difference Fossil Oil Ratio	First Difference Price
0	64.949019	306.020464	2015-01	4.711703	NaN	NaN	NaN
1	56.383854	319.239583	2015-02	5.661897	0.950195	NaN	-8.565165
2	55.522463	319.333782	2015-03	5.751434	0.089537	NaN	-0.861391
3	58.354083	338.781337	2015-04	5.805615	0.054181	NaN	2.831620
4	57.294059	332.567204	2015-05	5.804567	-0.001048	NaN	-1.060024

In []:

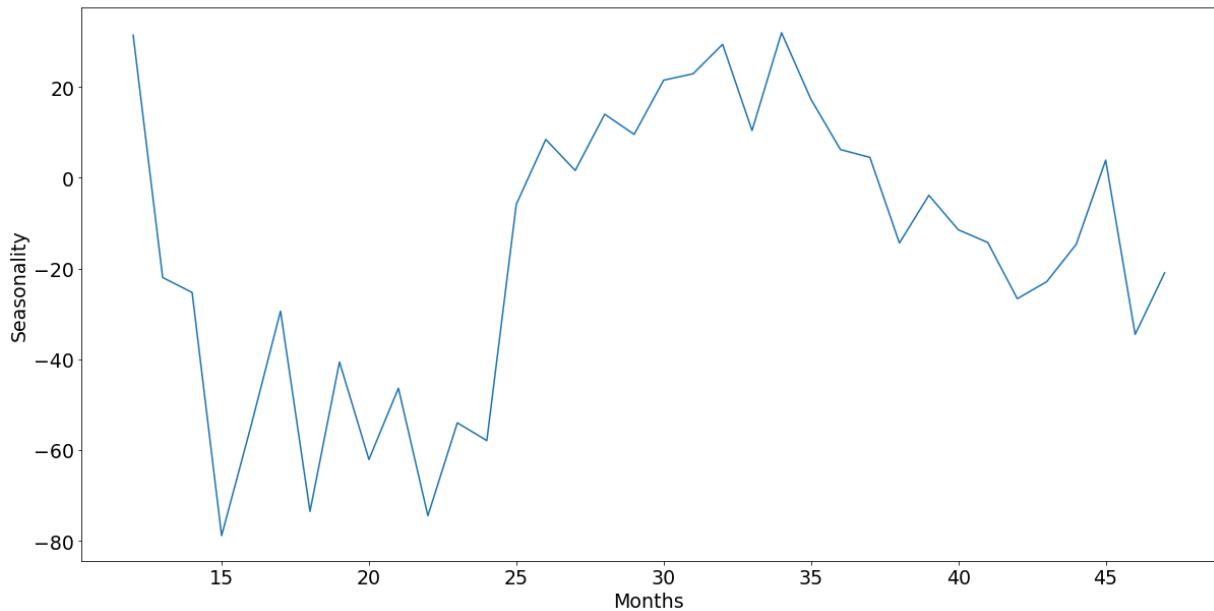
```
df_FossilOil['First Difference'] = df_FossilOil["Fossil_Oil"] - df_FossilOil.shift(1)
df_FossilOil['Seasonal Difference']=df_FossilOil["Fossil_Oil"]- df_FossilOil.shift(12)
df_FossilOil.head()
```

```
Out[ ]:
```

	Price	Fossil_Oil	Dates	Ratio	First Difference Fossil Oil Ratio	Seasonal Difference Fossil Oil Ratio	First Difference Price
0	64.949019	306.020464	2015-01	4.711703	NaN	NaN	NaN
1	56.383854	319.239583	2015-02	5.661897	0.950195	NaN	-8.565165
2	55.522463	319.333782	2015-03	5.751434	0.089537	NaN	-0.861391
3	58.354083	338.781337	2015-04	5.805615	0.054181	NaN	2.831620
4	57.294059	332.567204	2015-05	5.804567	-0.001048	NaN	-1.060024

```
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 0x7f81caf36e90>
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)
```

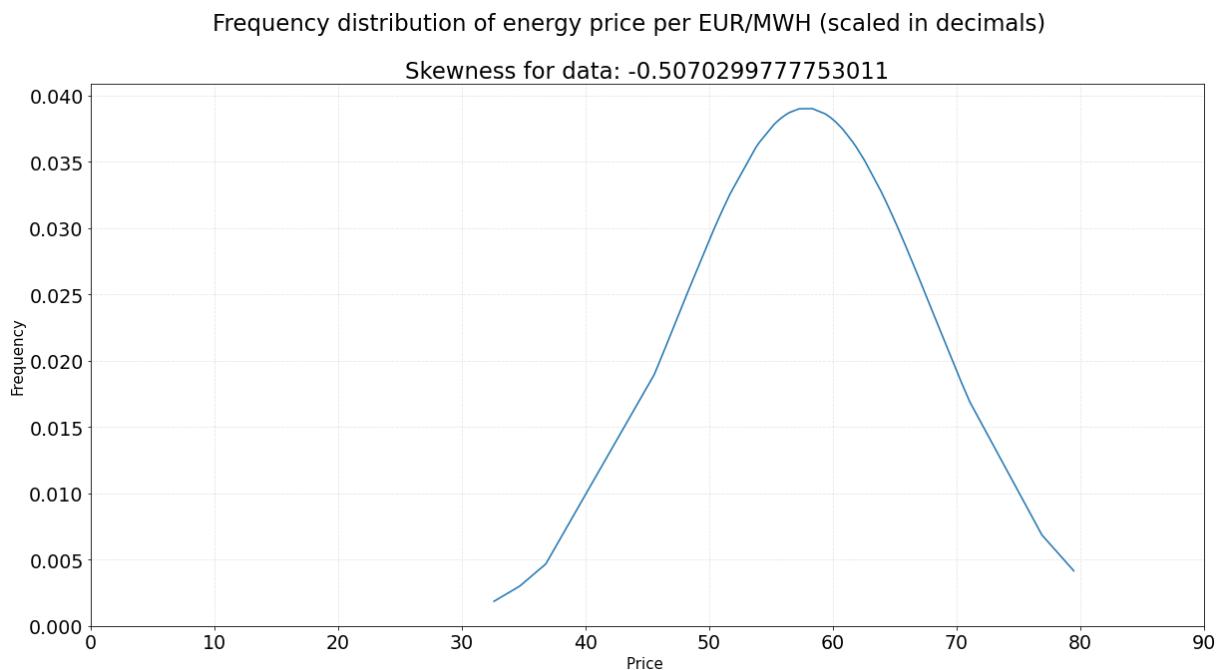
```
print(Fossil_Oil_Autocorrelations)

[ 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 * n p.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(), Fc
```

```
plt.plot(df_FossilOil["Ratio"].sort_values(), FossilOilResultspdf)
plt.xlim([0,20])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
```

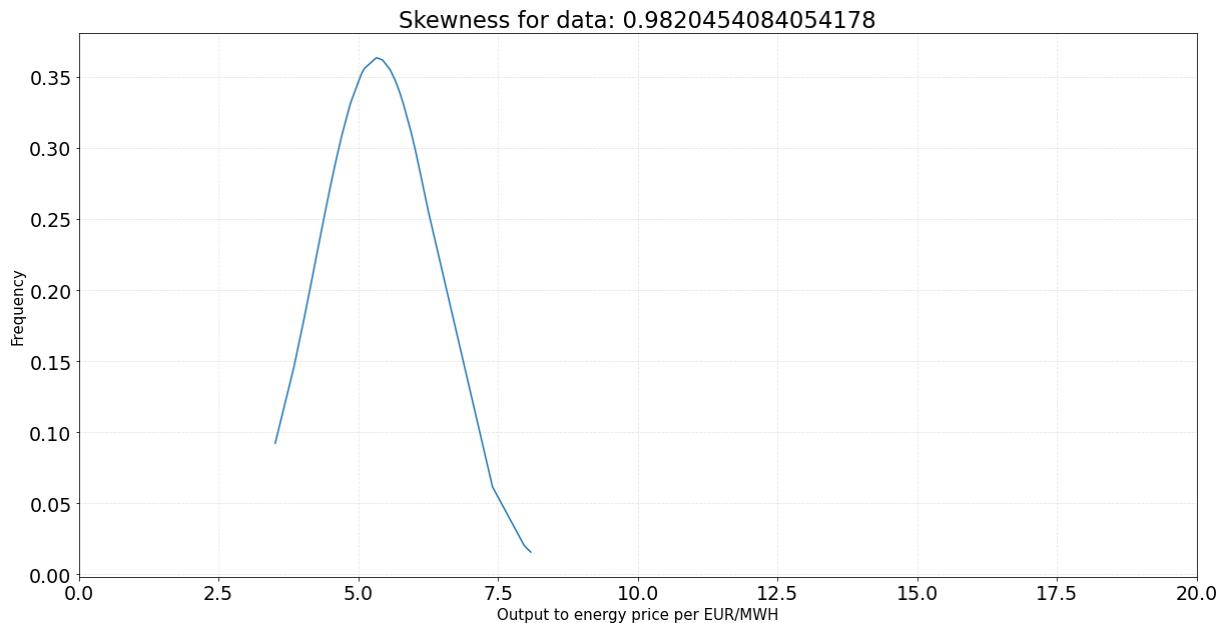
```

plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")

plt.title(f'Skewness for data: {skew(df_FossilOil["Ratio"])}')
plt.suptitle("Frequency distribution of output to energy price per EUR/MWH r
plt.show()

```

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



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.

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

```

In [ ]: #Bell Curves

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

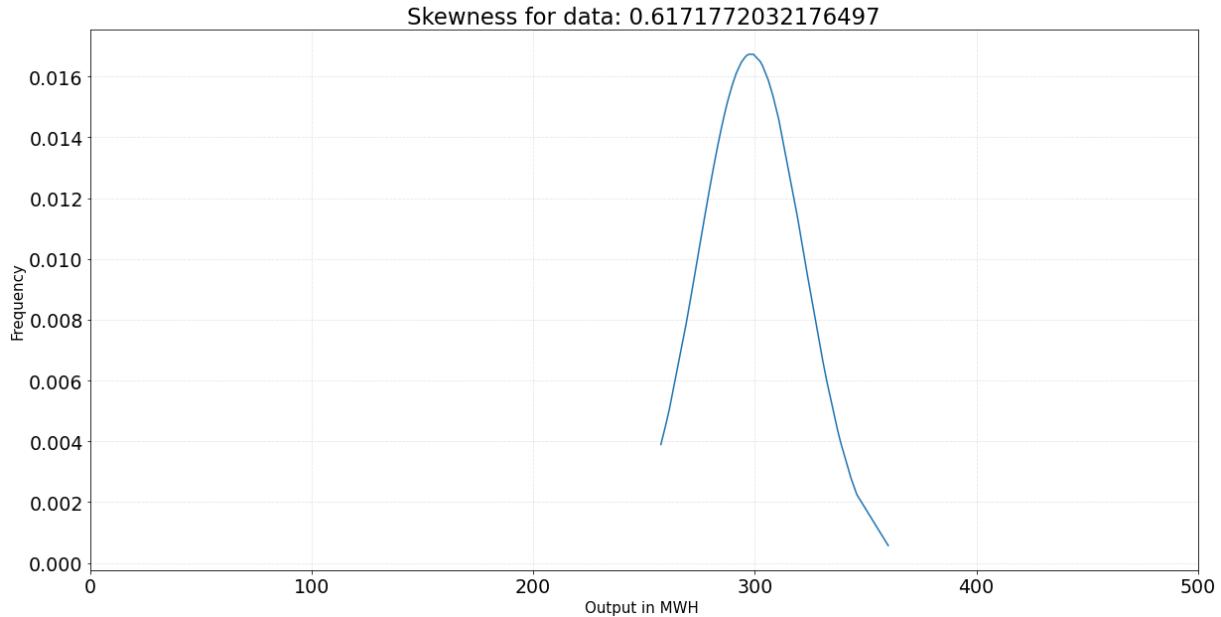
FossilOilResultspdf = stats.norm.pdf(df_FossilOil["Fossil_Oil"].sort_values()

plt.plot(df_FossilOil["Fossil_Oil"].sort_values(), FossilOilResultspdf)
plt.xlim([0,500])
plt.xlabel("Output in MWH ", size=15)
plt.title(f'Skewness for data: {skew(df_FossilOil["Fossil_Oil"])}')
plt.suptitle("Frequency distribution of average monthly fossil oil outputs v
plt.ylabel("Frequency", size=15)

```

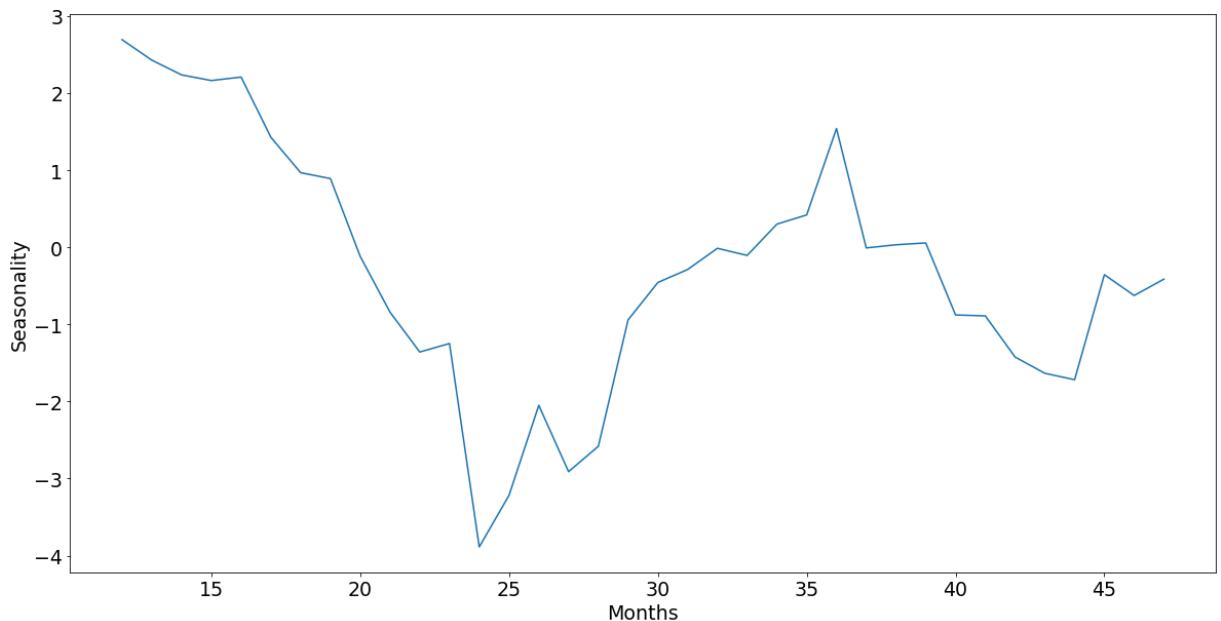
```
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

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



```
In [ ]: df_FossilOil['First Difference ratios'] = df_FossilOil["Ratio"]- df_FossilOil["Ratio"].shift(1)
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 0x7f81cadcc4110>
Seasonal Difference of average monthly fossil oil outputs to price of energy per EUR/MWH



```
In [ ]:
```

```
In [ ]:
```

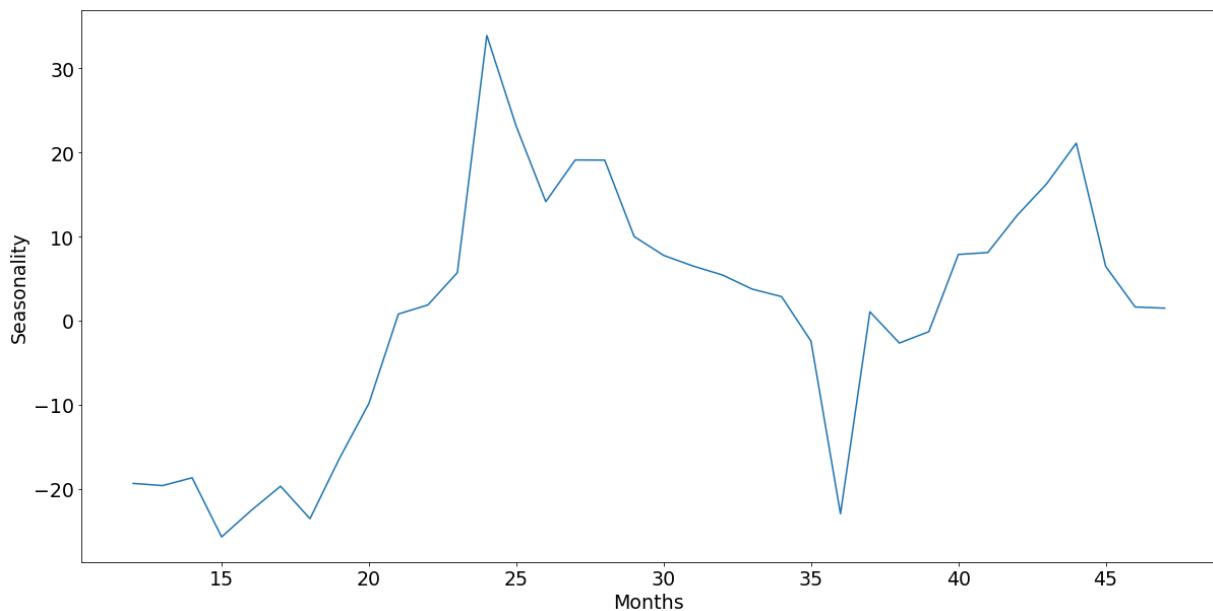
The blue line represents the trend line among the values themselves. As one can observe, there are no obvious patterns depicted between the average monthly megawatts from fossil oil and the average monthly prices of energy per EUR/MWH.

```
In [ ]:
```

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

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c6eef90>
```

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 table of Dataframes
```

Out[]:

	Price	Fossil_Oil	Dates	Ratio	First Difference Fossil Oil Ratio	Seasonal Difference Fossil Oil Ratio	Differ
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	47.000000
unique	NaN	NaN	48	NaN	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN	NaN
mean	57.859848	298.324069	NaN	5.334508	-0.013070	-0.295283	0.013070
std	10.320573	24.078477	NaN	1.109870	0.659037	1.612341	6.712341
min	32.618667	257.629167	NaN	3.516361	-1.746713	-3.887643	-19.618667
25%	51.528411	281.505343	NaN	4.540563	-0.260050	-1.276519	-2.240563
50%	59.622281	292.923387	NaN	5.216690	-0.093915	-0.322757	1.216690
75%	64.999583	312.908413	NaN	5.764717	0.242475	0.536684	2.864717
max	79.492083	360.233871	NaN	8.088094	2.062453	2.692301	11.892083

Below is a histogram of average monthly fossil oil outputs.

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

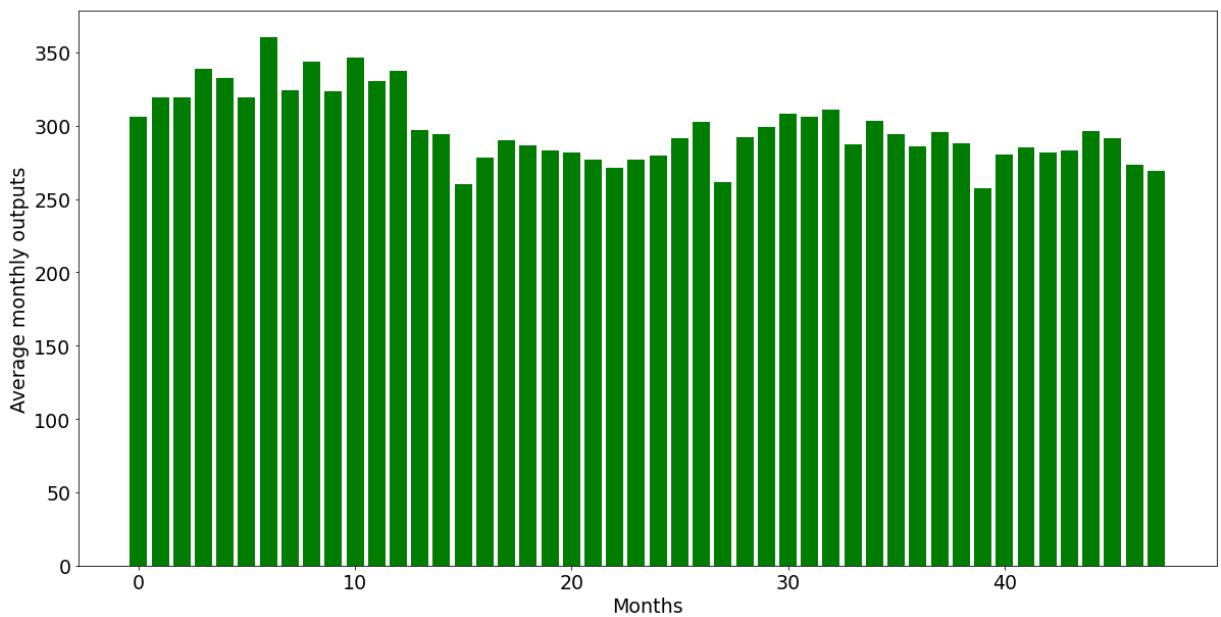
```
def Hist_fossil_oil(fossil_oil_Dict):
    for k, v in fossil_oil_Dict.items(): print(f"{v}:{k}")
print(fossil_oil_Dict)
```

```
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')
#Histograms
```

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

```
{306.0204638472033: 0, 319.2395833333333: 1, 319.3337819650067: 2, 338.78133
704735376: 3, 332.56720430107526: 4, 319.2294853963839: 5, 360.2338709677419
5: 6, 323.9139784946237: 7, 343.4916666666667: 8, 323.39112903225805: 9, 34
6.06388888888887: 10, 330.6514131897712: 11, 337.46505376344084: 12, 297.254
31034482756: 13, 294.05248990578735: 14, 259.8875: 15, 277.9153225806452: 1
6, 289.82916666666665: 17, 286.6900269541779: 18, 283.30645161290323: 19, 28
1.3861111111111: 20, 276.9959731543624: 21, 271.56944444444446: 22, 276.6397
8494623655: 23, 279.52284946236557: 24, 291.4017857142857: 25, 302.507402422
61105: 26, 261.4902777777778: 27, 291.9206989247312: 28, 299.3930555555556:
29, 308.2002688172043: 30, 306.23521505376345: 31, 310.8013888888889: 32, 28
7.4187919463087: 33, 303.54305555555555: 34, 293.9260752688172: 35, 285.7392
4731182797: 36, 295.9002976190476: 37, 288.1265141318977: 38, 257.62916666666
6666: 39, 280.4502688172043: 40, 285.12083333333334: 41, 281.5450874831763:
42, 283.333333333333: 43, 296.136111111111: 44, 291.2993288590604: 45, 27
2.98333333333335: 46, 269.02150537634407: 47}
```

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 (January 2015 to December 2018). 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}") #Histogram
print(pdtoListFossil_Oil_Dict)

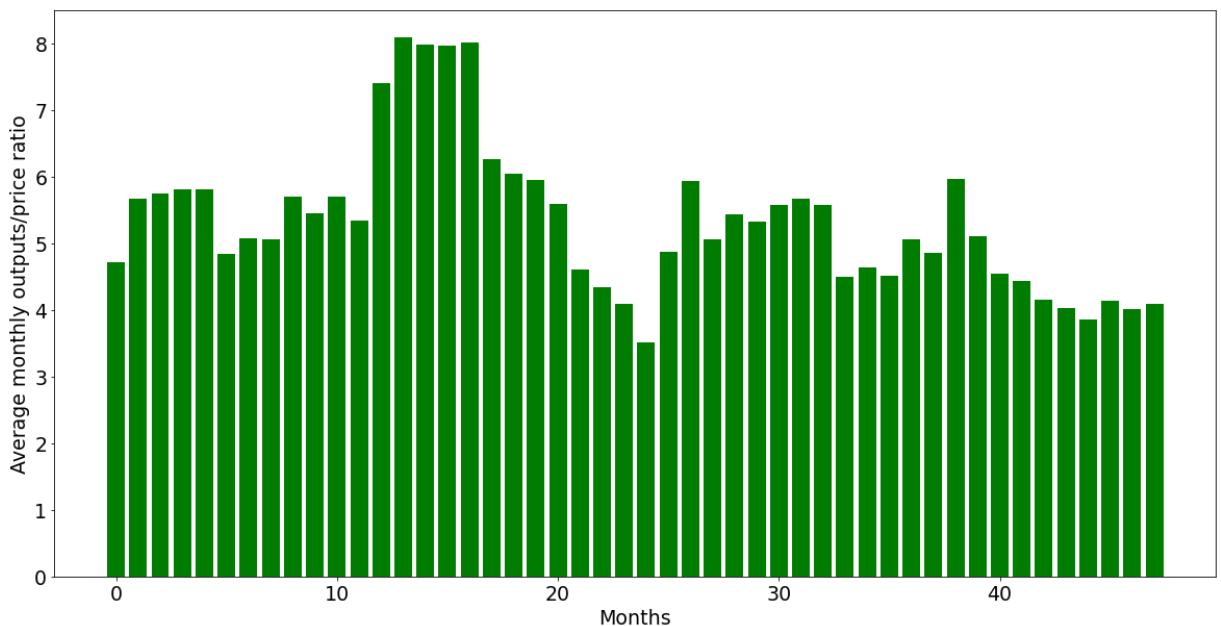
plt.bar(list(pdtoListFossil_Oil_Dict.values()), pdtoListFossil_Oil_Dict
print(dicDates)
plt.suptitle("Average monthly outputs/price ratio of fossil oil")
```

```
plt.ylabel('Average monthly outputs/price ratio')
plt.xlabel('Months')
```

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

```
{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, 7.4040041202492315: 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, 5.568265289669134: 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 ratio of fossil oil



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 prices')
plt.xlabel('Average monthly prices')

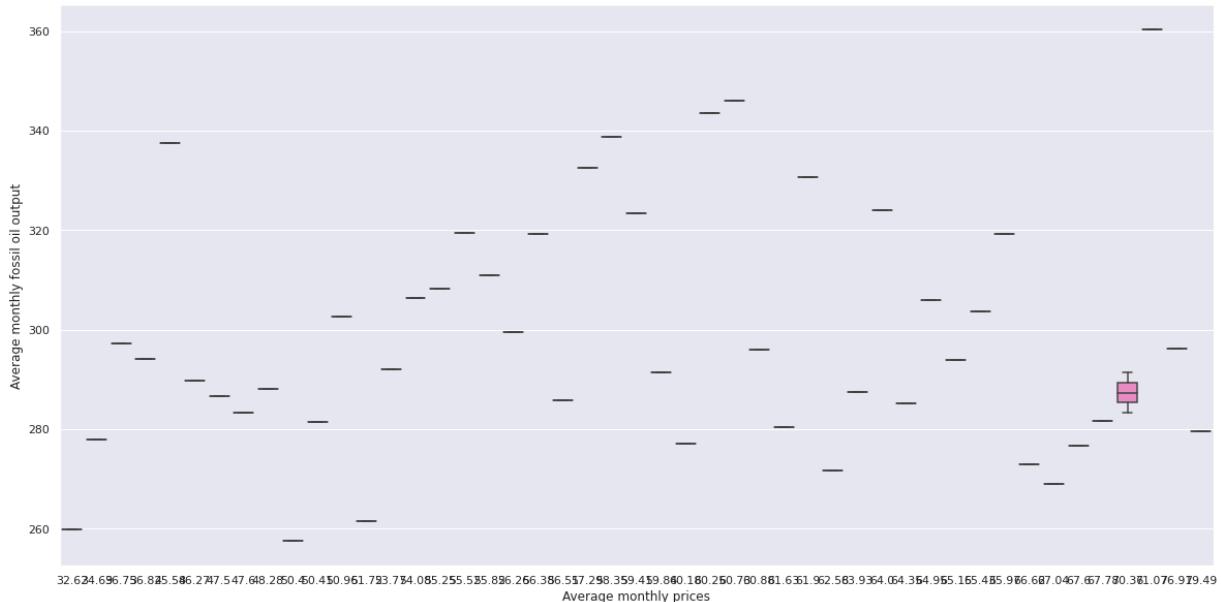
plt.rcParams["figure.figsize"] = [40, 20]
plt.ylabel('Average monthly fossil oil output')

sns.boxplot(x=Rounded_Y, y=[306.0204638472033, 319.2395833333333, 319.3333333333333])
plt.show

```

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

Distribution of average monthly fossil oil outputs based off of average monthly price of energy per EUR/MWH



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 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.

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

```

plt.suptitle('Distribution of average monthly fossil oil outputs based off of average monthly prices')
plt.xlabel('Average monthly prices')

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

```

```

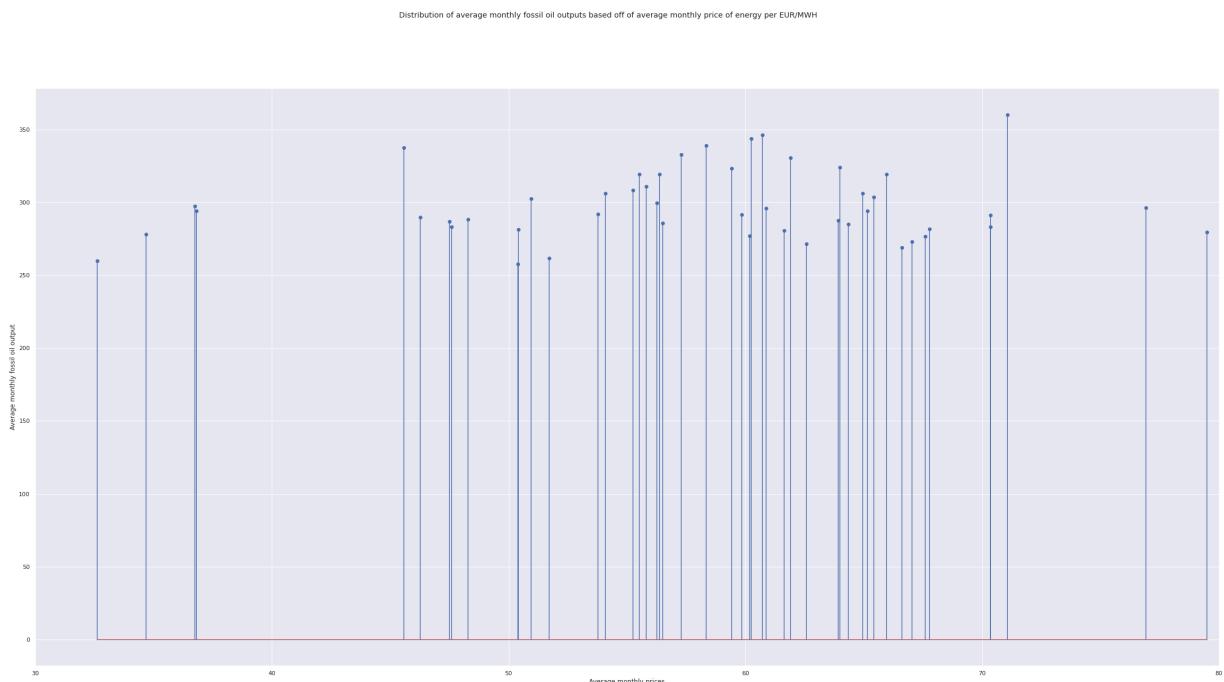
plt.xlim(30, 80)

# 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 Line Collection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behavior, 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 approximently 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. Each blue dot and the end of the blue lines represent an observation.

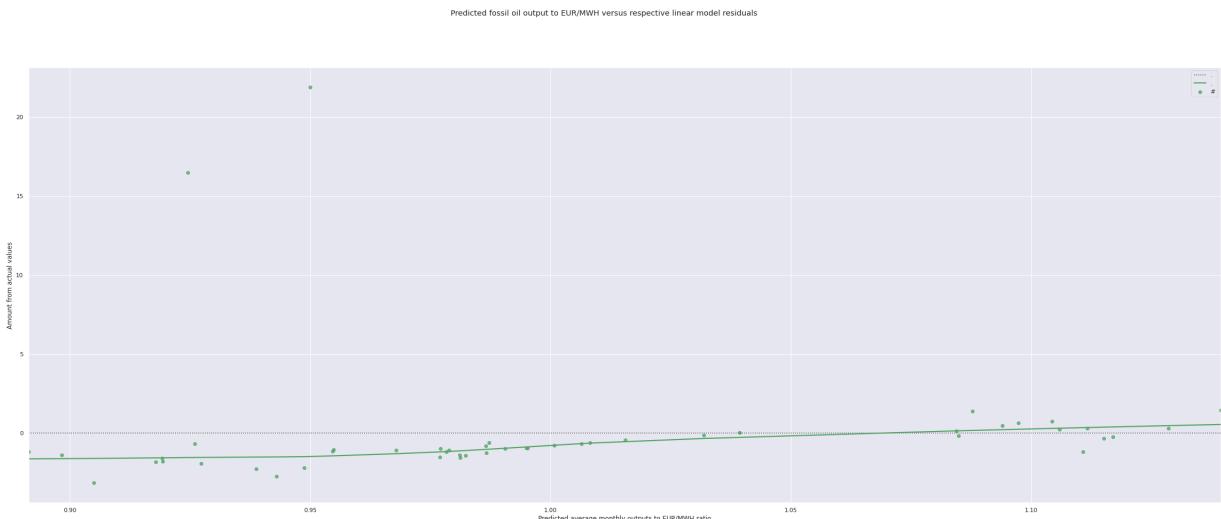
In []: print(fossil_oil)

```
[306.0204638472033, 319.2395833333333, 319.3337819650067, 338.7813370473537  
6, 332.56720430107526, 319.2294853963839, 360.23387096774195, 323.9139784946  
237, 343.49166666666667, 323.39112903225805, 346.06388888888887, 330.65141318  
97712, 337.46505376344084, 297.25431034482756, 294.05248990578735, 259.8875,  
277.9153225806452, 289.82916666666665, 286.6900269541779, 283.3064516129032  
3, 281.3861111111111, 276.9959731543624, 271.56944444444446, 276.63978494623  
655, 279.52284946236557, 291.4017857142857, 302.50740242261105, 261.49027777  
77778, 291.9206989247312, 299.3930555555556, 308.2002688172043, 306.23521505  
376345, 310.8013888888889, 287.4187919463087, 303.54305555555555, 293.926075  
2688172, 285.73924731182797, 295.9002976190476, 288.1265141318977, 257.62916  
666666666, 280.4502688172043, 285.1208333333334, 281.5450874831763, 283.333  
333333333, 296.1361111111111, 291.2993288590604, 272.9833333333335, 269.02  
150537634407]
```

In []:

```
#Predicted OLS linear average monthly ratios versus residuals  
FossilOilRegRatioPredict = predictionsFossilOil/predictions  
  
sns.residplot(x = FossilOilRegRatioPredict, y = standardized_residualsFossil  
plt.suptitle("Predicted fossil oil output to EUR/MWH versus respective linear  
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")  
plt.ylabel("Amount from actual values")  
  
plt.legend(..#)
```

Out[]: <matplotlib.legend.Legend at 0x7f81c66bec90>



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.

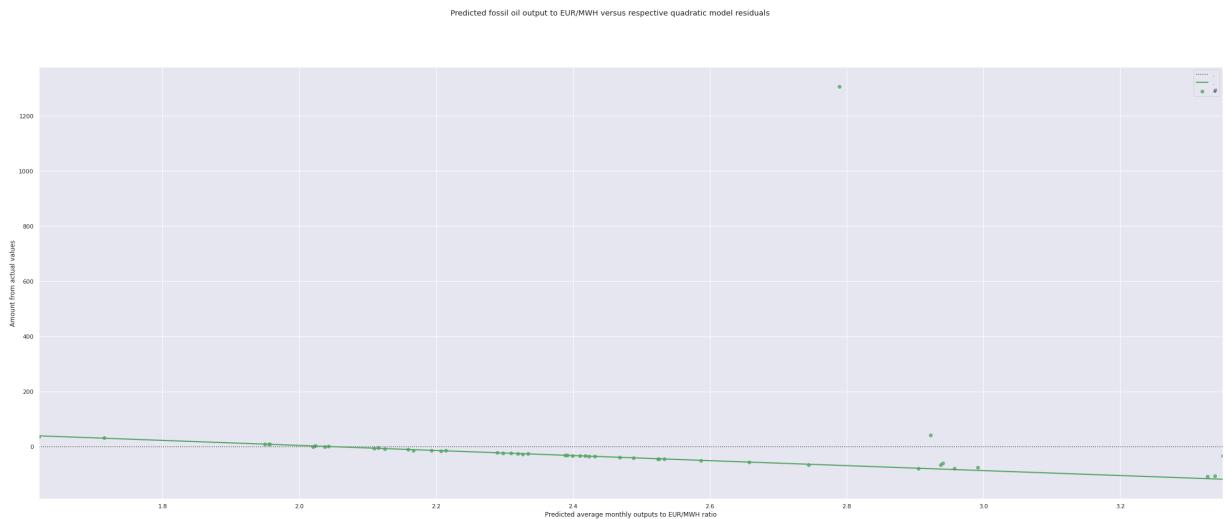
```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
FossilOilQuadRatioPredict = FossilOil_ypred/ypred

sns.residplot(x = FossilOilQuadRatioPredict , y = standardized_residualsFoss

plt.suptitle("Predicted fossil oil 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 0x7f81c663df90>
```

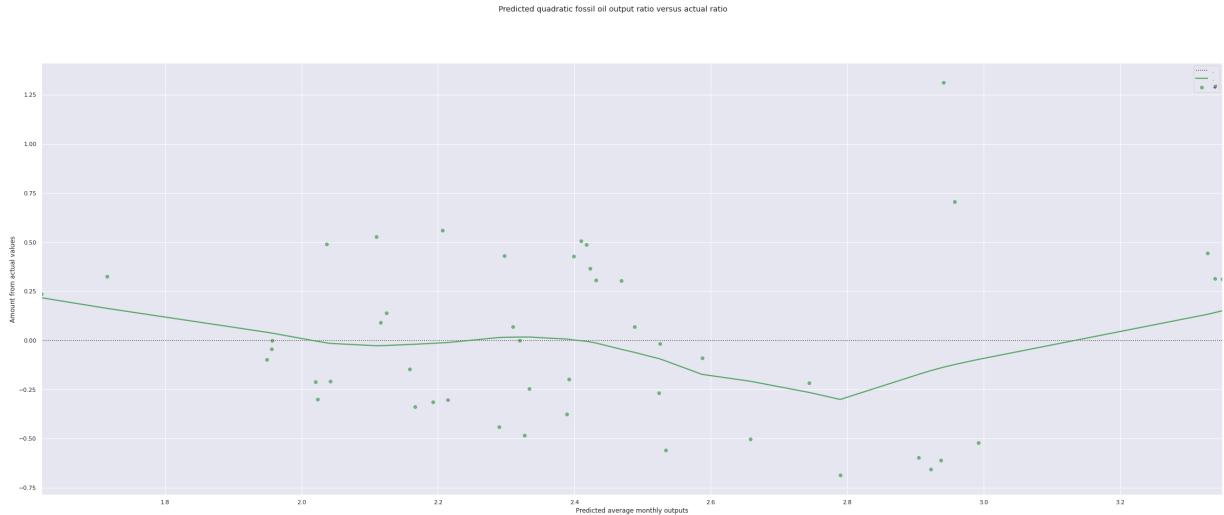


With the exception of a few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green 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 actual ratios
sns.residplot(x = FossilOilQuadRatioPredict , y = pdToListFossilOil, lowess
plt.suptitle("Predicted quadratic fossil oil output ratio versus actual ratio")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c659b450>
```

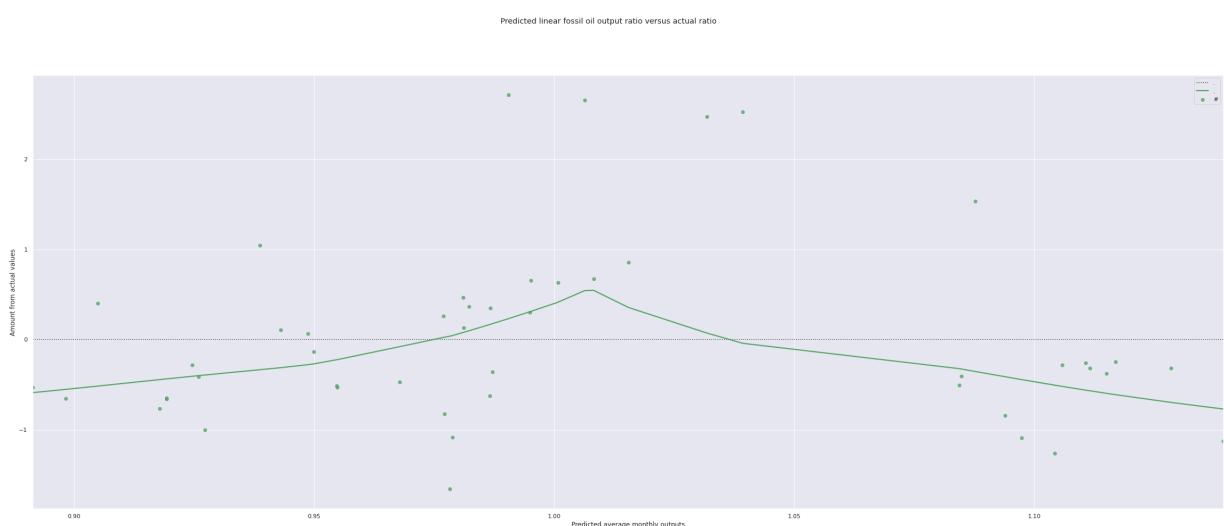


As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. In addition, the residuals are quite spread out in no particular pattern which indicates constant variance, homoscedasticity, and a lack of bias. Given these circumstances, it would be reasonable to assume 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 [ ]: # OLS predicted linear average monthly ratios versus actual ratios
sns.residplot(x = FossilOilRegRatioPredict, y = pdToListFossilOil, lowess =
plt.suptitle("Predicted linear fossil oil output ratio versus actual ratio")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c6613650>
```



As one can observe this residual plot, one may notice that the lowess line is subtle and that the residuals are spread out; indicating homoscedacity, 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 []:

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  
[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,  
62.5810555555558,  
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] )
```

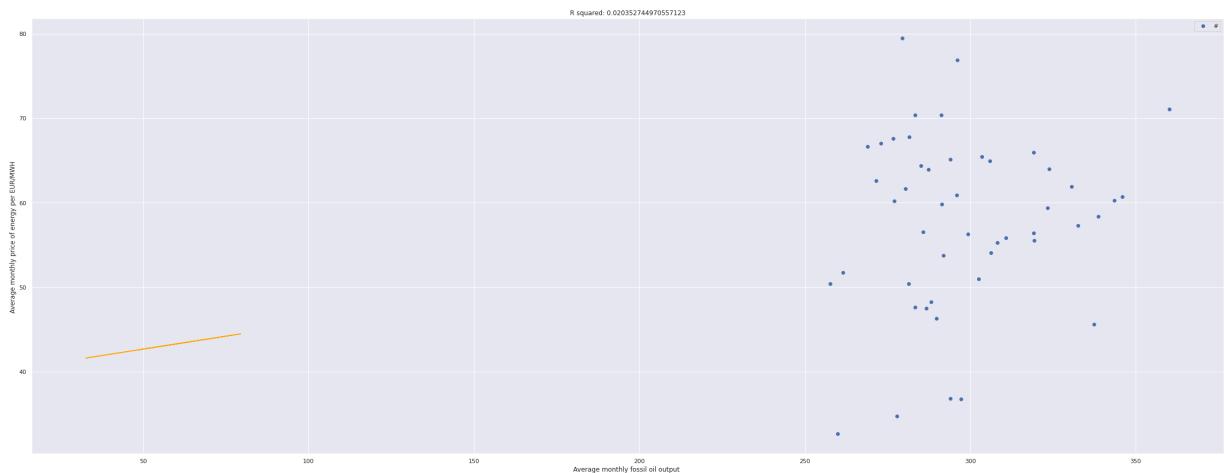
In []:

```
slope: 0.061210      intercept: 39.599580
```

In []:

In []:

Average monthly outputs versus average monthly prices of energy per EUR/MWh



The next resource analyzed is brown fossil coal.

```
#Dataframes analyzed by resource  
dffossilreg = ({"Price": [64.9490188172043, 56.38385416666667, 55.52246298788,  
"Fossil_Brown" : [572.8512960436562, 313.41815476190476, 244.43741588156124  
dffossilreg["Dates"] += ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05',  
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, 58.35408333333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.99806451612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.618666666666, 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, 548.083333333334, 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, 478.81111111111113, 650.6774193548387, 511.8467741935484, 642.630555555556, 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, 375.736111111111], '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, 2.658277973784002
8, 4.992463284681974, 5.190493724889739, 7.470390538004405, 4.26557403666614
1, 7.256946833324577, 5.767834131536049, 6.059258576134178, 5.63969219242538
8]
```

```
In [ ]: modelFossilBrowncoal = stats.linregress([572.8512960436562, 313.418154761904
[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,
79.49208333333331,
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)
```

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(io2, fossil_brown1).fit()
predictionsFossilBrown = modelFossilBrownreg.predict(io21)

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:	Sun, 09 Oct 2022	Prob (F-statistic):	1.03e-06			
Time:	06:03:58	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.

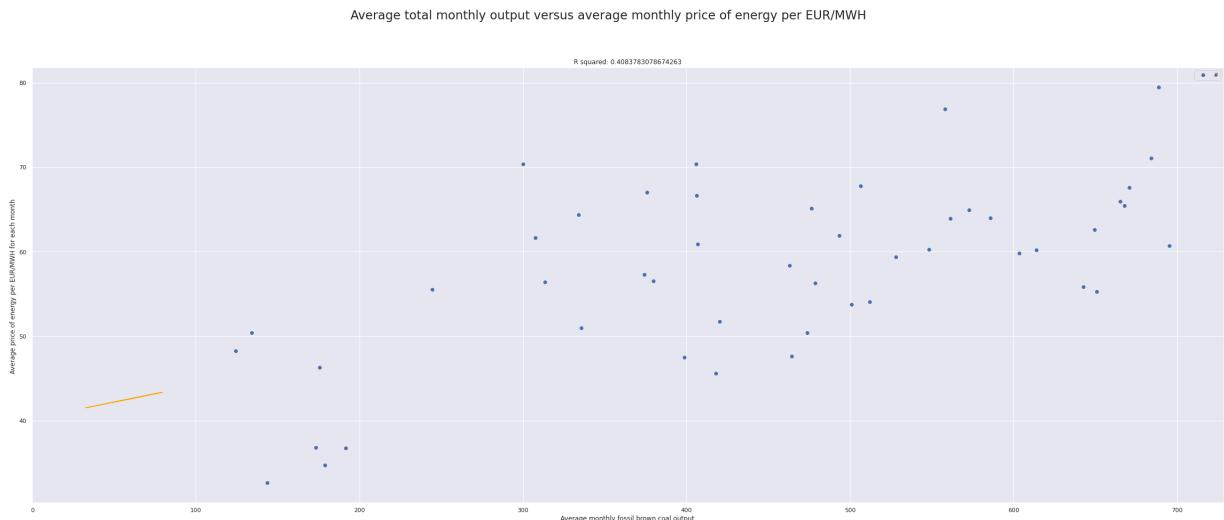
In []:

```
In [ ]: modelFossilBrowncoal = stats.linregress([572.8512960436562, 313.418154761904
[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,
62.5810555555558,
67.5951344086021,
79.4920833333331,
59.83779761904767,
50.95989232839837,
51.71791666666662,
53.77262096774189,
56.2582222222224,
55.252580645161316,
54.08432795698931,
55.8165555555558,
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 [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(fossil_brown,
print("slope: %f      intercept: %f" % (slope, intercept))

# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of energy")
plt.plot(fossil_brown,io2, "o")
f = lambda x: 0.039383 *x + 40.221857
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average total monthly output versus average monthly price of energy")
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 output')
plt.show()
There is a moderately positive correlation the average monthly outputs of brown coal
```



```
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_int

print(standardized_residualsFossilBrown)
```

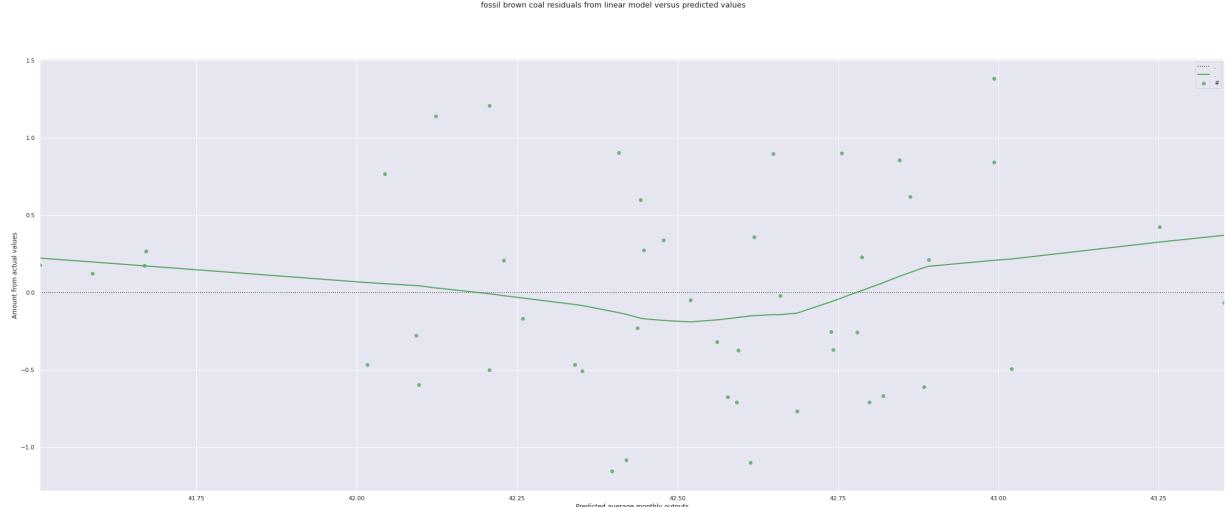
```
[ 0.27449969  0.48426507  0.72622188 -0.01344027  0.29426516 -0.05682334
  0.50256118  0.08969648 -0.19621151 -0.20323495 -0.88731681  0.28295657
 -1.39850883 -1.42412307 -1.32694542 -1.73274324 -1.63045272 -0.11299218
 -1.06204996 -1.37368487 -1.06717828 -0.53605465 -0.41223875  0.12270134
  1.56546771 -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)

plt.suptitle("fossil brown coal residuals from linear model versus predicted")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend("#")
```

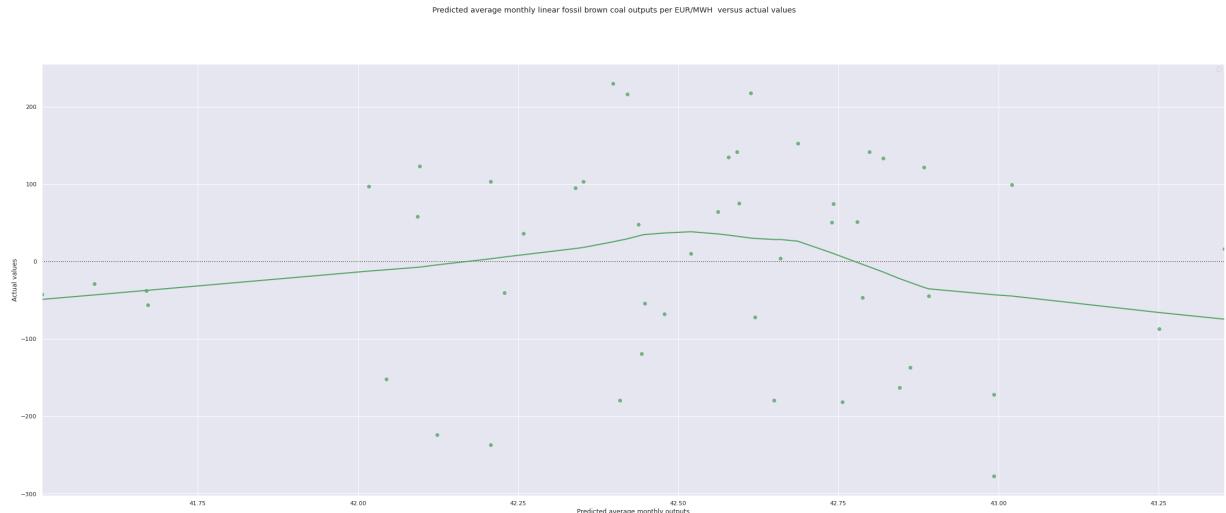
```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c643ff90>
```



As one can observe this residual plot, one may notice that the lowess line is subtle and that the residuals are spread out; indicating homoscedacity, 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 outputs per EUR/MWh versus actual values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsFossilBrown, y = fossil_brown, lowess = True, c
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c636d650>
```



As one can observe this residual plot, one may notice that the lowess line is subtle and that the observations are spread out; indicating homoscedacity, 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 [ ]: print(fossil_brown)
```

```
[572.8512960436562, 313.41815476190476, 244.43741588156124, 463.119777158774  
35, 374.2809139784946, 665.162726008345, 684.2204301075269, 585.767473118279  
6, 548.083333333334, 528.0188172043011, 695.284722222222, 493.48048452207  
3, 417.9274193548387, 191.66522988505747, 173.20323014804845, 143.5708333333  
3333, 179.002688172043, 175.6, 398.8156123822342, 464.3736559139785, 473.720  
83333333336, 613.7691275167786, 649.5458333333333, 670.7956989247311, 688.64  
51612903226, 603.4151785714286, 335.66756393001344, 420.1152777777778, 500.7  
7016129032256, 478.8111111111113, 650.6774193548387, 511.8467741935484, 64  
2.6305555555556, 561.3221476510067, 667.647222222222, 476.5430107526882, 37  
9.56451612903226, 406.95089285714283, 124.41588156123822, 133.9791666666666  
6, 307.7043010752688, 333.9986111111113, 506.36877523553164, 300.1424731182  
7955, 558.161111111112, 405.83758389261743, 375.736111111111, 406.22849462  
365593]
```

This is the average monthly logarithmic outputs of fossil brown coal the average monthly prices of energy per EUR/MWH.

```
In [ ]: #Logarithmic OLS regressions  
Logpricevalues = ((np.log(io2)))  
LogFossilBrownvalues = ((np.log(fossil_brown)))  
Log = np.polyfit(np.log(io2), fossil_brown1, 1)  
lin2 = LinearRegression()  
lin2.fit(np.log(fossil_brown1), io2)  
FossilBrown_Log = sm.OLS(io2, 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), io2, 1)  
print(Log)  
  
y = 14.56449676 * LogFossilBrownvalues - 29.7139618  
  
#Logarithmic OLS regression scatterplot  
plt.suptitle("Logarithmic average monthly outputs versus average monthly pri  
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, io2, "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 linear 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:	Sun, 09 Oct 2022	Prob (F-statistic):	1.03e-06			
Time:	06:04:00	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_student

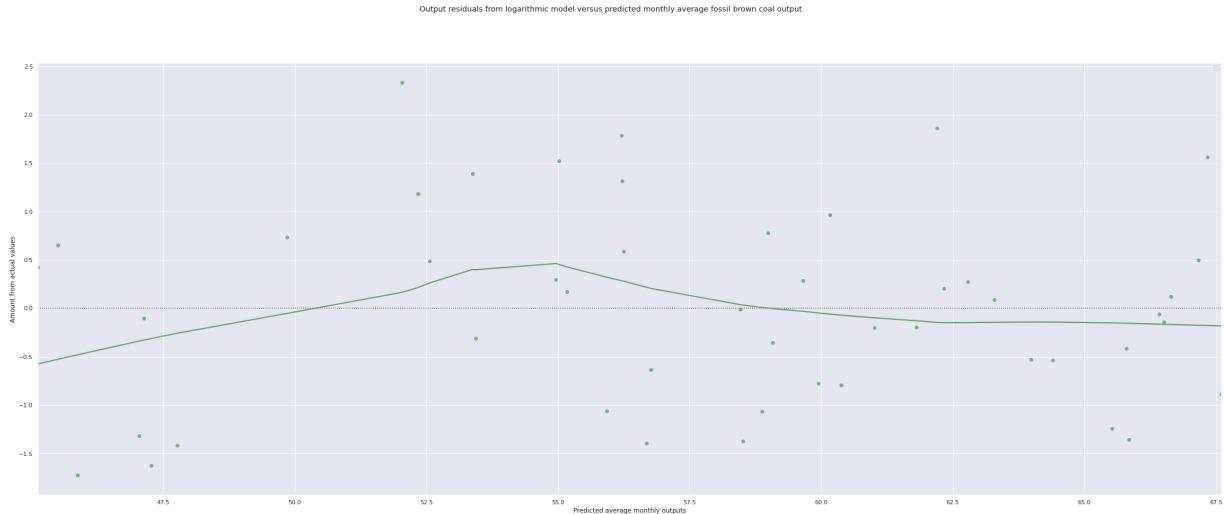
print(standardized_residualsFossilBrownLog)

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 outputs")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = FossilBrown_Logpred, y = standardized_residualsFossilBrown)
# OLS Logarithmic average monthly predictions versus residuals
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c624d990>

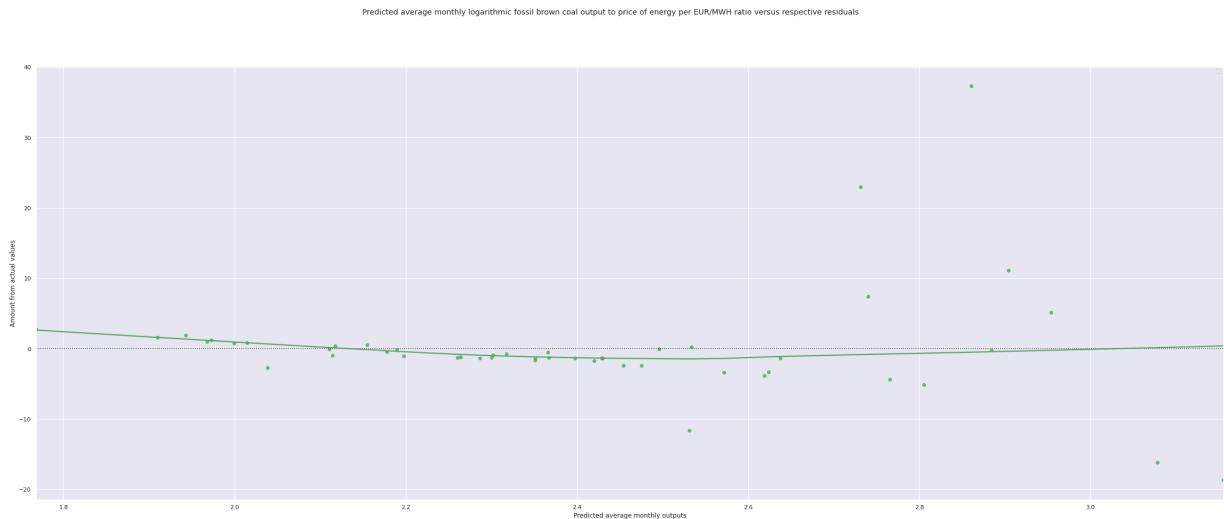


As one can observe this residual plot, one may notice the slight hump in the fitted model, which form a nonlinear pattern. The slight hump is too subtle to suggest a different model. In addition, the residuals are spread out without the distinct pattern, indicating constant variance, a lack of bias and 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/predictionLog
```

```
In [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
plt.suptitle("Predicted average monthly logarithmic fossil brown coal output
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#")
sns.residplot(x = FossilBrownLogRatioPredict, y = standardized_residualsFoss
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c6127990>
```

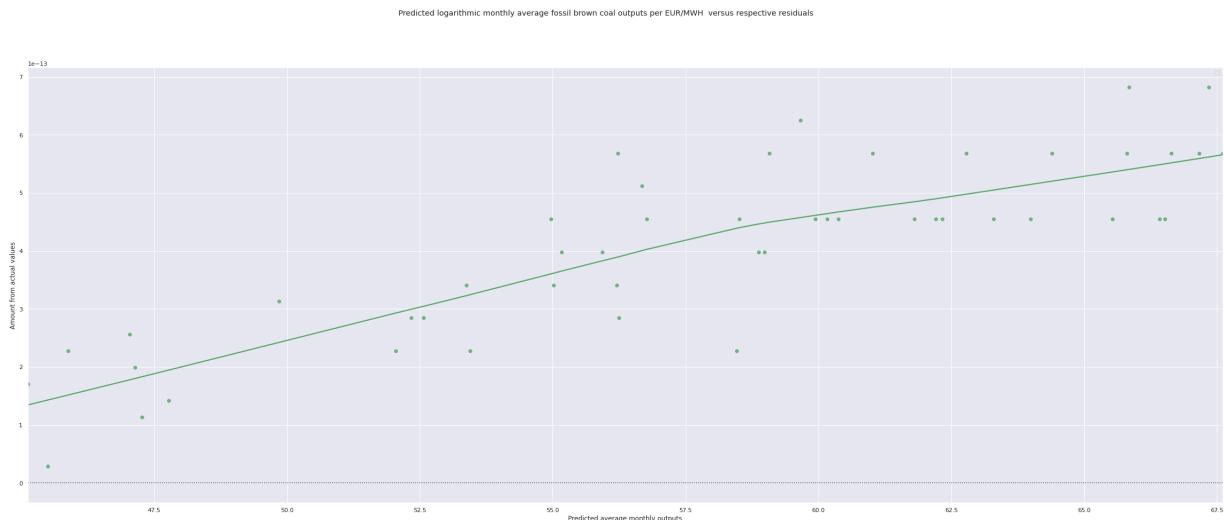


With the exception of a few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green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

```
In [ ]: # OLS predicted logarithmic average monthly values versus actual values
```

```
plt.suptitle("Predicted logarithmic monthly average fossil brown coal output
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#")
sns.residplot(x = FossilBrown_Logpred, y = fossil_brown, lowess = True, colc
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c60f9910>
```



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.

```
In [ ]: #OLS Logarithmic predicted average monthly ratios versus actual ratios
plt.suptitle("Predicted logarithmic monthly average fossil brown coal output
plt.xlabel("fossil brown coal Predicted average monthly outputs to EUR/MWh r
plt.ylabel("Amount from actual values")
plt.legend("..#")
FossilBrownLogRatioPredict = FossilBrown_Logpred/predictionLog
sns.residplot(x = FossilBrownLogRatioPredict, y = pdToList, lowess = True, c
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c607aad0>
```



As one can observe this residual plot, one may notice that the lowess line has an arching hump and that the residuals are spread out in a pattern; indicating heteroskedacity 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.

This is the quadratic model used for the average monthly fossil brown output versus the 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,io2,2))
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)

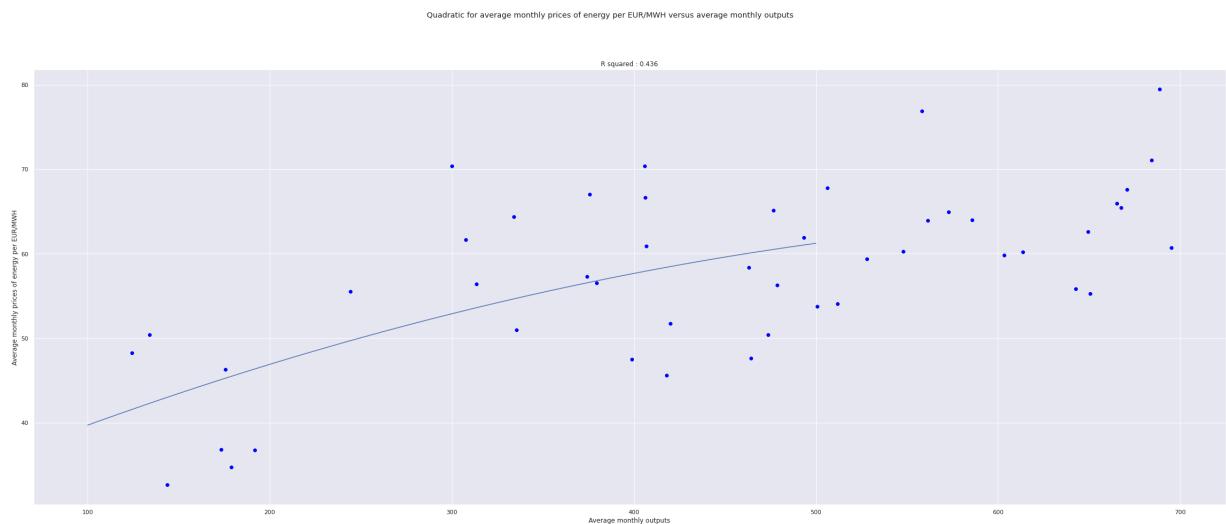
FossilBrown_Q = poly.fit(X_poly, fossil_brown)
lin2 = LinearRegression()
lin2.fit(X_poly, fossil_brown)
FossilBrown_Quad = sm.OLS(io2, fossil_brown2).fit()

# OLS Predicted Quadratic values
FossilBrown_ypred = FossilBrown_Quad.predict(fossil_brown2)

#OLS Quadratic Summary Table
FossilBrown_Quad.summary()

#Quadratic Scatterplots
polyline = np.linspace(start = 100, stop =500 , num = 100)
plt.title("R squared : 0.436")
plt.plot(polyline, modelFossilBrownquad(polyline))
plt.scatter(fossil_brown,io2, color = 'blue')
plt.suptitle('Quadratic for average monthly prices of energy per EUR/MWH ver')
plt.xlabel('Average monthly outputs')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()
```

$$2 \\ -6.058e-05 x^2 + 0.09019 x + 31.28$$



The blue dots represent the observations and the blue line is the quadratic 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:	Sun, 09 Oct 2022	Prob (F-statistic):	2.58e-06				
Time:	06:04:04	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
#Quadratic OLS residuals
print(standardized_residualsFossilBrownquad)

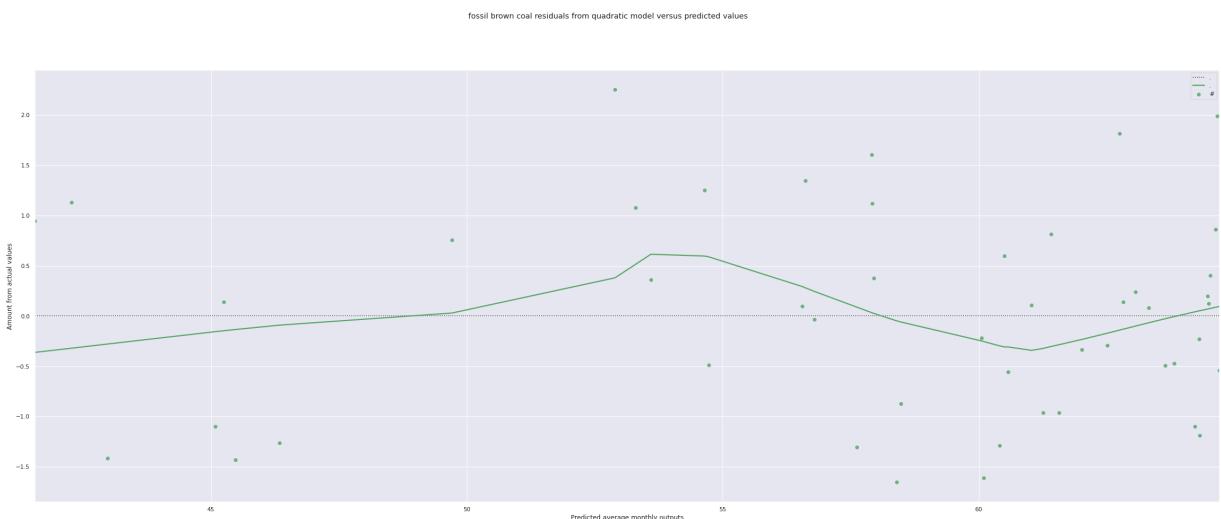
[ 0.24151186  0.35938576  0.75396049 -0.21925071  0.09590346  0.19845127
 0.8616789   0.08638358 -0.28995122 -0.33451973 -0.53774368  0.11133079
 -1.65288886 -1.26539187 -1.10326914 -1.42015968 -1.43475754  0.13551199
 -1.30477666 -1.60912659 -1.28767612 -0.46906945 -0.22540135  0.40678445
 1.99188545 -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)

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("#")
```

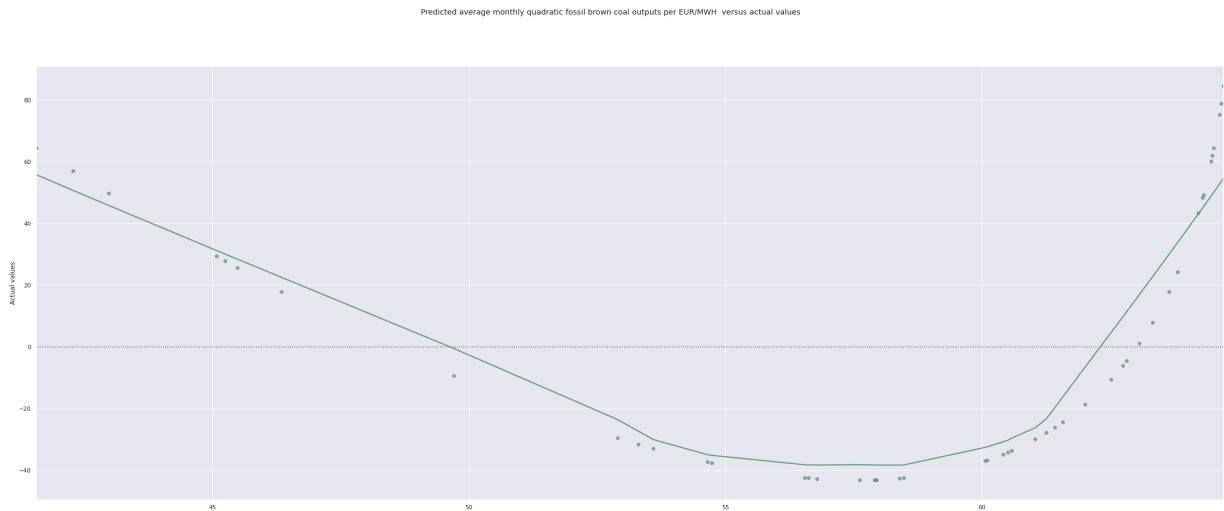
```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c5fd2d50>
```



As one can observe, there is a double yet subtle hump along the lowess line in the residual plot. In addition, there is a decreasing trend in the variance which indicates heteroskedasticity 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 quadratic fossil brown coal outputs")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = FossilBrown_ypred, y = fossil_brown, lowess = True, color =
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c5f49950>
```



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. 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 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, 55.52246298788,
 "Fossil_Brown" : [572.8512960436562, 313.41815476190476, 244.43741588156124,
 dffossilreg["Dates"] += ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05',
 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")
    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)
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 # Partial autocorrelation Function
plot_pacf(df_fossilbrown["Ratio"])
plt.show

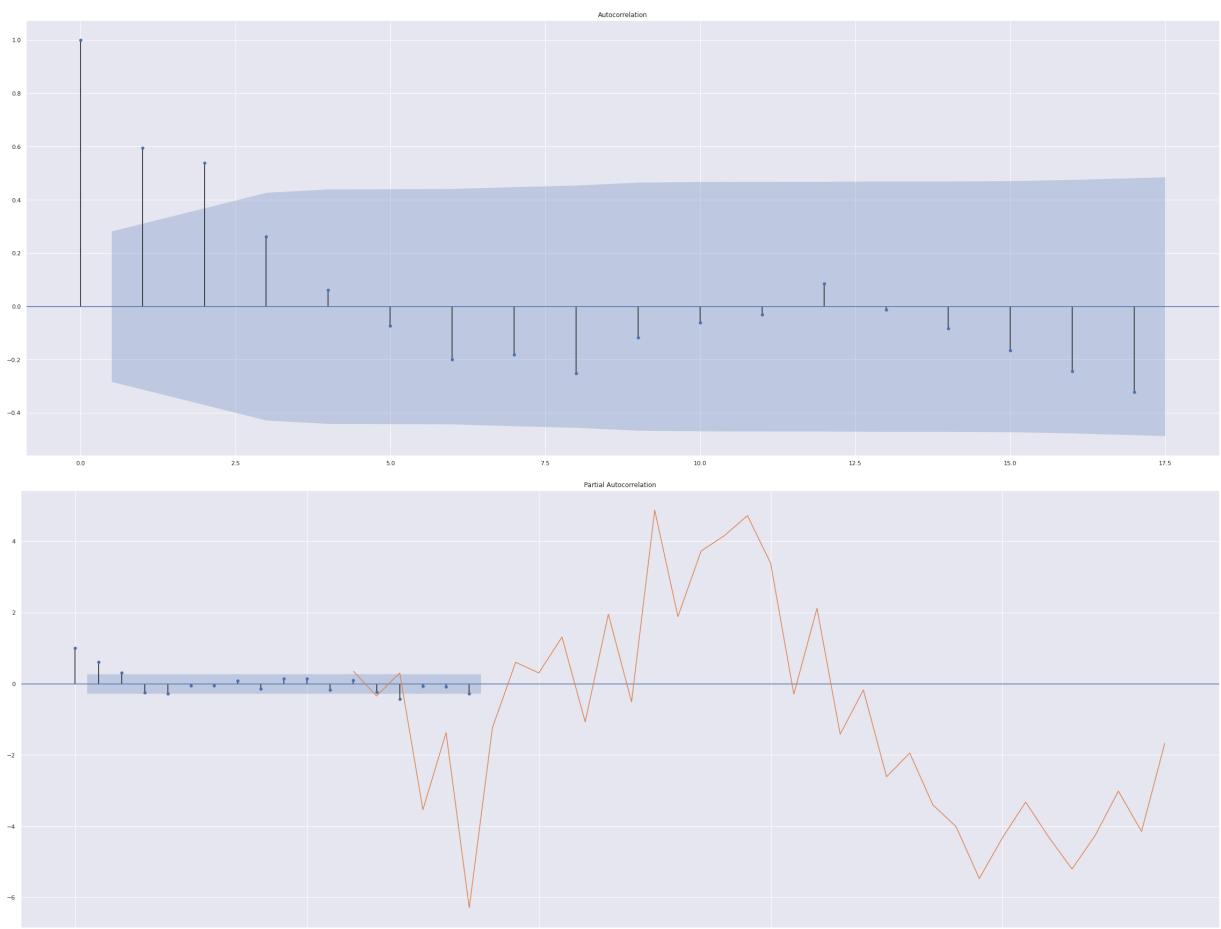
df_fossilbrown['Seasonal Difference'].plot() # Seasonality Plot
```

```
{'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, 32.618666666666, 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, 548.083333333334, 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, 478.81111111111113, 650.6774193548387, 511.8467741935484, 642.630555555556, 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, 375.736111111111], '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, 2.658277973784002
8, 4.992463284681974, 5.190493724889739, 7.470390538004405, 4.26557403666614
1, 7.256946833324577, 5.767834131536049, 6.059258576134178, 5.63969219242538
8]
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 0x7f81c5de4750>
```



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 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 table of Dataframes
```

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"])}')
plt.suptitle("Frequency distribution of output to energy price per EUR/MWH")
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"])

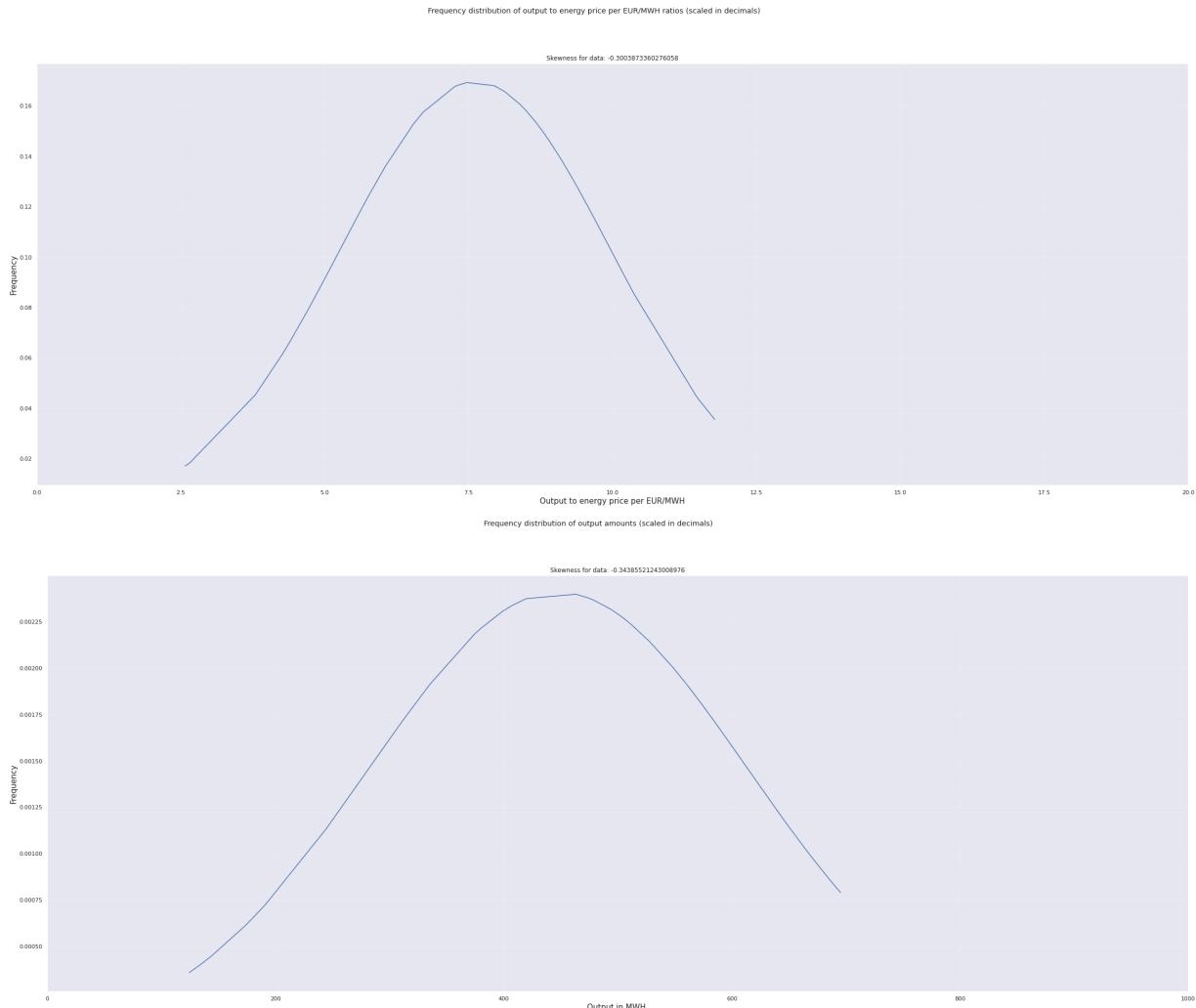
patientsResultspdf = stats.norm.pdf(df_fossilbrown["Fossil_Brown"].sort_values(), patientsResults_mean, patientsResults_std)

plt.plot(df_fossilbrown["Fossil_Brown"].sort_values(), patientsResultspdf)
plt.xlim([0,1000])
plt.xlabel("Output in MWH ", size=15)
```

```

plt.title(f'Skewness for data: {skew(df_fossilbrown["Fossil_Brown"])}')
plt.suptitle("Frequency distribution of output amounts (scaled in decimals)")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

```



These bell shaped curves are slightly skewed to the left. 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 [ ]: Fossil_Brown_Ratio_Autocorrelations = sm.tsa.acf(df_fossilbrown["Ratio"], fft
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  
-0.01006611  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 * n p.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to an integer to silence this warning.
```

```
FutureWarning,
```

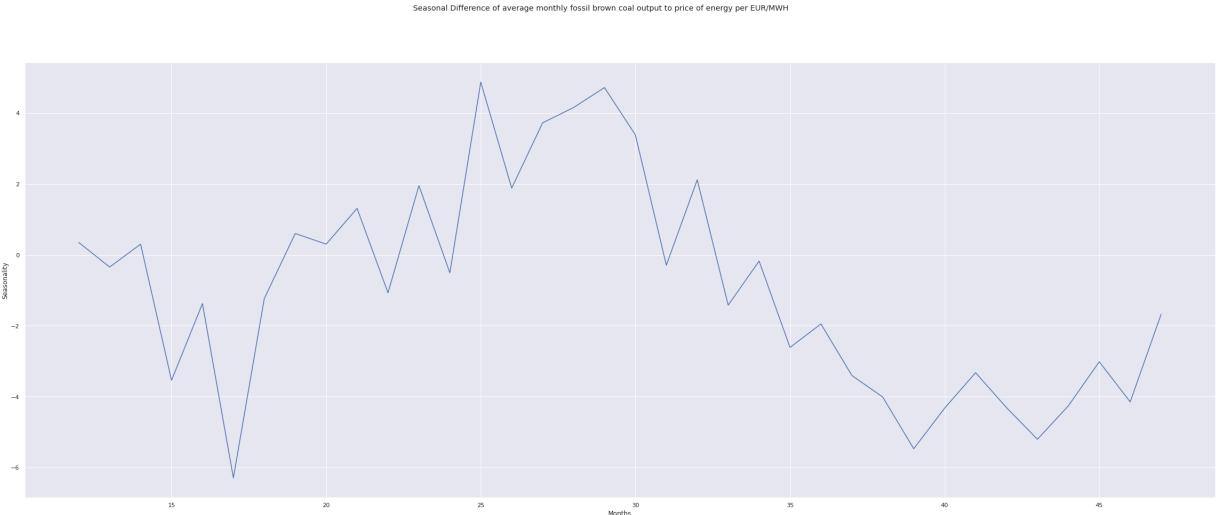
```
In [ ]: df_fossilbrown['First Difference fossil brown coal Ratio'] = df_fossilbrown[  
df_fossilbrown['Seasonal Difference fossil brown coal Ratio']=df_fossilbrown[  
df_fossilbrown.head()
```

```
Out[ ]:
```

	Price	Fossil_Brown	Dates	Ratio	First Difference	Seasonal Difference	First Difference fossil brown coal Ratio
0	64.949019	572.851296	2015-01	8.820015	NaN	NaN	NaN
1	56.383854	313.418155	2015-02	5.558651	-3.261364	NaN	-3.261364
2	55.522463	244.437416	2015-03	4.402496	-1.156155	NaN	-1.156155
3	58.354083	463.119777	2015-04	7.936373	3.533877	NaN	3.533877
4	57.294059	374.280914	2015-05	6.532630	-1.403743	NaN	-1.403743

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly fossil brown coal output")  
plt.ylabel('Seasonality')  
plt.xlabel('Months')  
df_fossilbrown['Seasonal Difference fossil brown coal Ratio'].plot() # Seasonal difference
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c5d7aa10>
```



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 []:

```
#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")
    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 fossil brown coal")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function
plot_pacf(df_fossilbrown["Fossil_Brown"])
plt.suptitle("Partialautocorrelations of fossil brown coal")
plt.ylabel('Partialautocorrelations')
plt.xlabel('Lags')
plt.show
```

```

Fossil_Brown_Autocorrelations = sm.tsa.acf(df_fossilbrown["Fossil_Brown"], f
print(Fossil_Brown_Autocorrelations)
df_fossilbrown['First Difference'] = df_fossilbrown["Fossil_Brown"]- df_foss
df_fossilbrown['Seasonal Difference']=df_fossilbrown["Fossil_Brown"]- df_fos
df_fossilbrown.head()
df_fossilbrown['First Difference fossil brown coal'] = df_fossilbrown["Fossi
df_fossilbrown['Seasonal Difference fossil brown coal']=df_fossilbrown["Foss
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

```

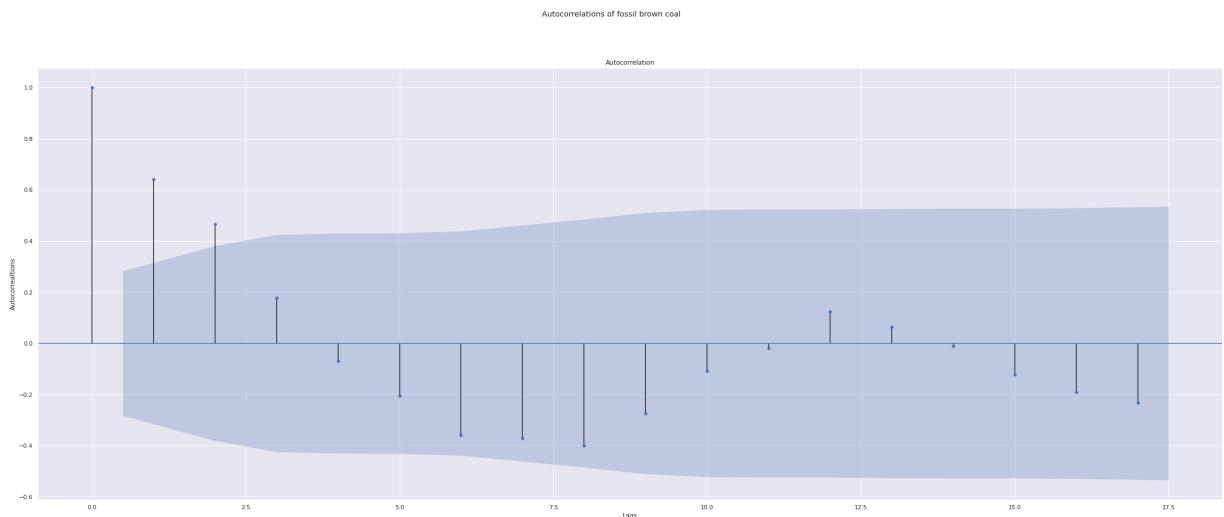
ADF Test Statistic : -3.4811300328069645
p-value : 0.008485147743162468
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(Ho), reject the null hypothesis.
Data is stationary

```
[ 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 * n p.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to a n integer to silence this warning.

FutureWarning,

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c5c10390>



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 brown coal outputs.

```
In [ ]: df_fossilbrown.describe(include = 'all') # Description table of Dataframes
```

Out[]:

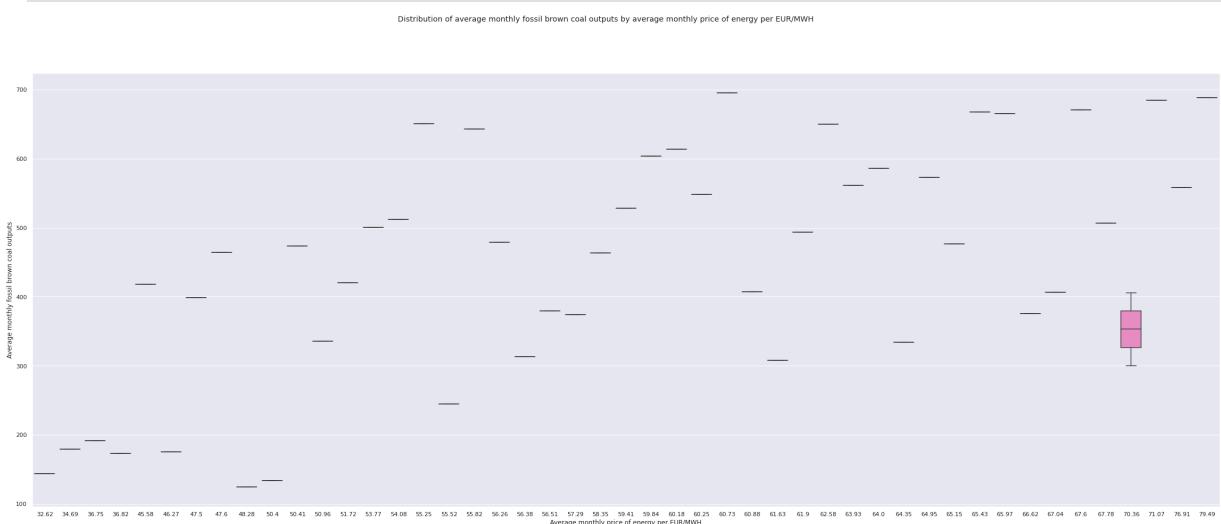
	Price	Fossil_Brown	Dates	Ratio	First Difference	Seasonal Difference
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	Nan	Nan	48	Nan	Nan	Nan
top	Nan	Nan	2015-01	Nan	Nan	Nan
freq	Nan	Nan	1	Nan	Nan	Nan
mean	57.859848	447.860317	Nan	7.617232	-4.193940	-53.584378
std	10.320573	167.425524	Nan	2.378039	141.783995	216.041609
min	32.618667	124.415882	Nan	2.576980	-282.535011	-489.562726
25%	51.528411	335.250326	Nan	5.619432	-87.034423	-195.574741
50%	59.622281	469.047245	Nan	8.047600	0.390911	-111.100358
75%	64.999583	576.080340	Nan	9.414601	82.551299	104.928816
max	79.492083	695.284722	Nan	11.776417	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 EUR/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, 244.43741]
plt.show()
```



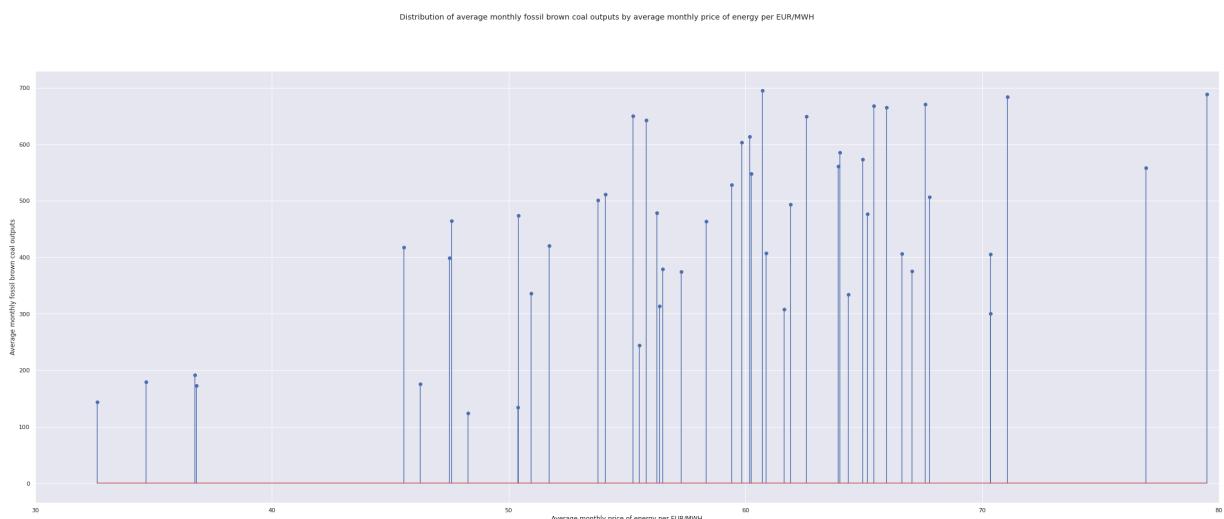
This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil brown coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of fossil brown coal produced, which was in between 300 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 slighty 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 a')
plt.ylabel('Average monthly fossil brown coal outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

plt.xlim(30, 80)
matplotlib.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)}

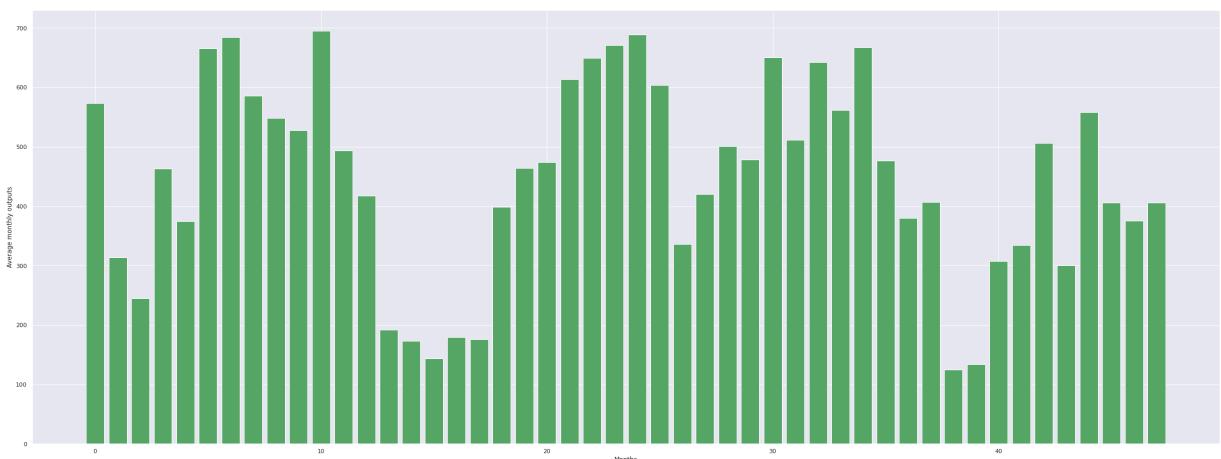
def Hist_fossil_brown(fossil_brown_Dict):
    for k, v in fossil_brown.items(): print(f"{v}:{k}")
print(fossil_brown_Dict)

plt.bar(list(fossil_brown_Dict.values()), fossil_brown_Dict.keys(), color='c'
print(dicDates)
plt.suptitle("Average monthly outputs of fossil brown coal")
plt.ylabel('Average monthly outputs')
plt.xlabel('Months')

matplotlib.rcParams.update({'font.size': 19})
#Histograms
plt.show()

{572.8512960436562: 0, 313.41815476190476: 1, 244.43741588156124: 2, 463.119
77715877435: 3, 374.2809139784946: 4, 665.162726008345: 5, 684.220430107526
9: 6, 585.7674731182796: 7, 548.083333333334: 8, 528.0188172043011: 9, 695.
284722222222: 10, 493.4804845222073: 11, 417.9274193548387: 12, 191.6652298
8505747: 13, 173.20323014804845: 14, 143.5708333333333: 15, 179.00268817204
3: 16, 175.6: 17, 398.8156123822342: 18, 464.3736559139785: 19, 473.72083333
333336: 20, 613.7691275167786: 21, 649.545833333333: 22, 670.7956989247311:
23, 688.6451612903226: 24, 603.4151785714286: 25, 335.66756393001344: 26, 42
0.1152777777778: 27, 500.77016129032256: 28, 478.8111111111113: 29, 650.677
4193548387: 30, 511.8467741935484: 31, 642.6305555555556: 32, 561.3221476510
067: 33, 667.647222222222: 34, 476.5430107526882: 35, 379.56451612903226: 3
6, 406.95089285714283: 37, 124.41588156123822: 38, 133.9791666666666: 39, 3
07.7043010752688: 40, 333.9986111111113: 41, 506.36877523553164: 42, 300.14
247311827955: 43, 558.161111111112: 44, 405.83758389261743: 45, 375.736111
111111: 46, 406.22849462365593: 47}
```

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}") #Histograms
print(pdtoList_Dict)

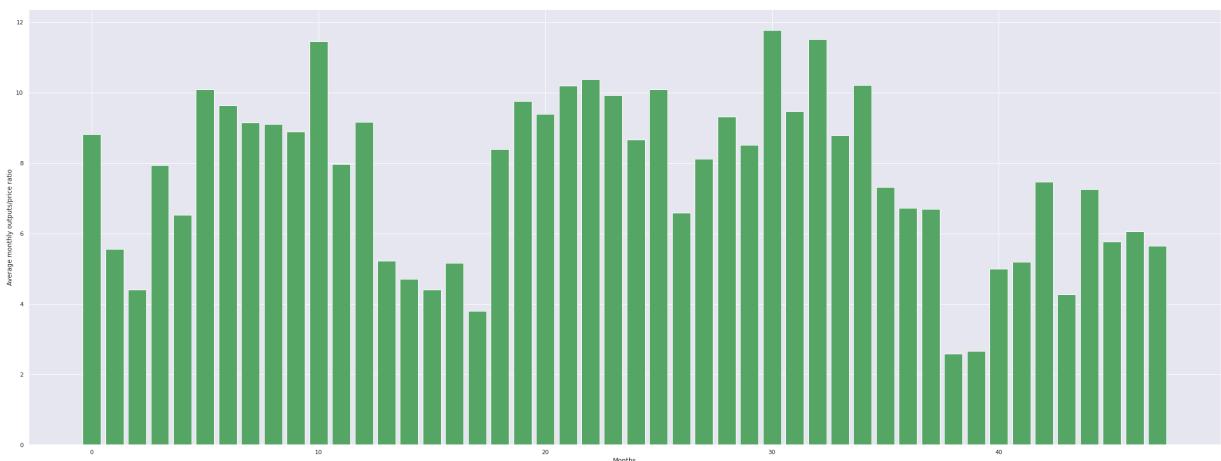
plt.bar(list(pdtoList_Dict.values()), pdtoList_Dict.keys(), color='g')
print(dicDates)
plt.suptitle("Average monthly outputs/price ratio of fossil brown coal")
plt.ylabel('Average monthly outputs/price ratio')
plt.xlabel('Months')
import matplotlib

matplotlib.rcParams.update({'font.size': 19})

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

```
{8.820014628025667: 0, 5.558650776789095: 1, 4.402495903953125: 2, 7.9363730
98576781: 3, 6.532630426225016: 4, 10.082056933813183: 5, 9.62713890191688:
6, 9.152893568689914: 7, 9.096095400434958: 8, 8.888193395306352: 9, 11.4493
90016167584: 10, 7.971994310368259: 11, 9.169353390403051: 12, 5.21508476530
9653: 13, 4.704307462627643: 14, 4.401493010137345: 15, 5.15986204000096: 1
6, 3.795417532852522: 17, 8.395761798802605: 18, 9.755269772482263: 19, 9.39
8179159446304: 20, 10.198477068938715: 21, 10.37927257006247: 22, 9.92372756
9944232: 23, 8.663065960954047: 24, 10.084180945512422: 25, 6.58689703986984
7: 26, 8.123205744839126: 27, 9.312734850524272: 28, 8.510953460629953: 29,
11.776416807272176: 30, 9.46386492221178: 31, 11.513260701225642: 32, 8.7809
09089763766: 33, 10.203890590695366: 34, 7.314407845265: 35, 6.7165323935762
97: 36, 6.684794525269369: 37, 2.5769803213273477: 38, 2.6582779737840028: 3
9, 4.992463284681974: 40, 5.190493724889739: 41, 7.470390538004405: 42, 4.26
5574036666141: 43, 7.256946833324577: 44, 5.767834131536049: 45, 6.059258576
134178: 46, 5.639692192425388: 47}
```

Average monthly outputs/price 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_residualsFc

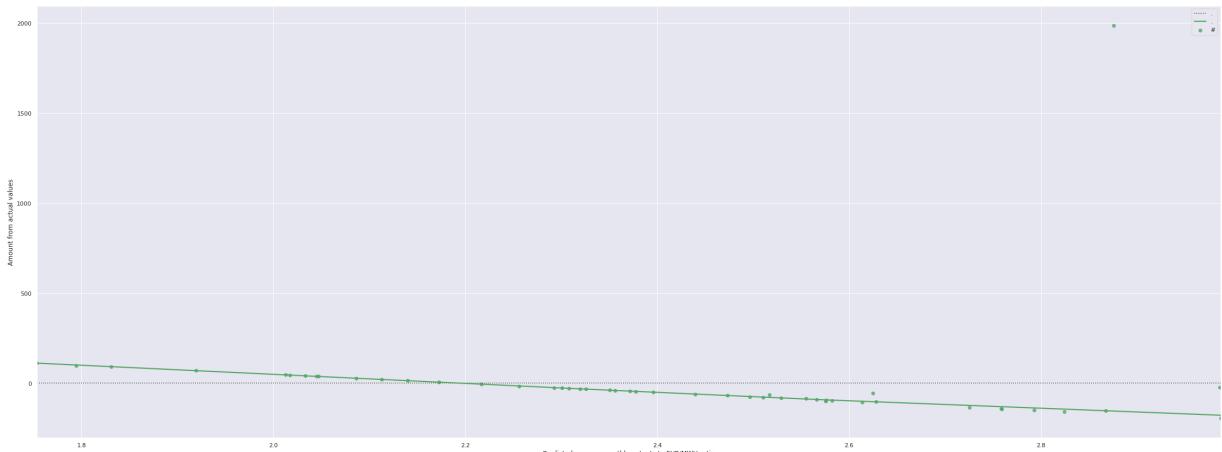
plt.suptitle("Predicted quadratic fossil brown coal output to EUR/MWH versus
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

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

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

Out[]: <matplotlib.legend.Legend at 0x7f81cb1596d0>

Predicted quadratic fossil brown coal output to EUR/MWH versus respective quadratic model residuals



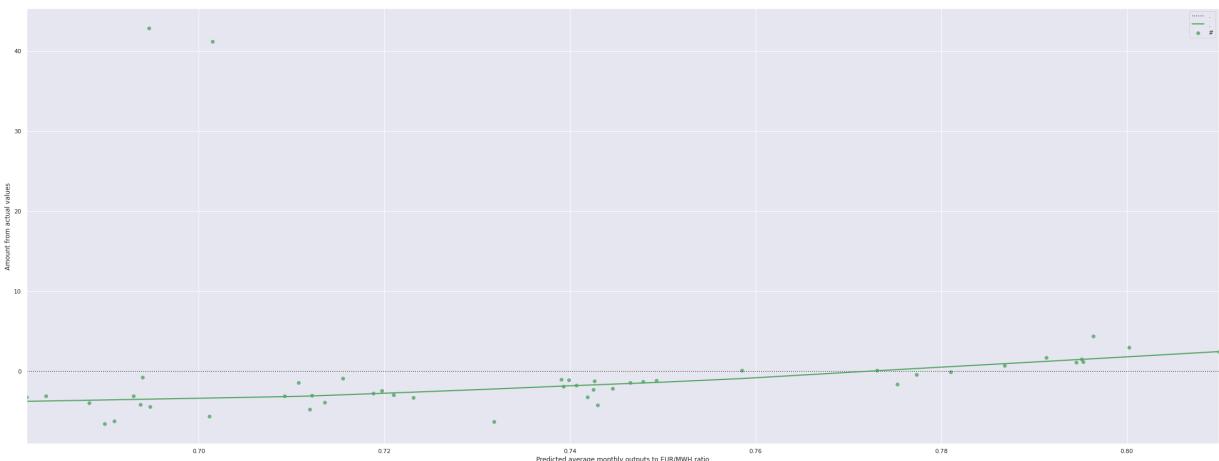
With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line 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_residualsFoss  
plt.suptitle("Predicted linear fossil brown coal output to EUR/MWH versus re  
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")  
plt.ylabel("Amount from actual values")  
plt.legend("...#")
```

Out[]: <matplotlib.legend.Legend at 0x7f81cb020910>

Predicted linear fossil brown coal output to EUR/MWH versus respective linear model residuals

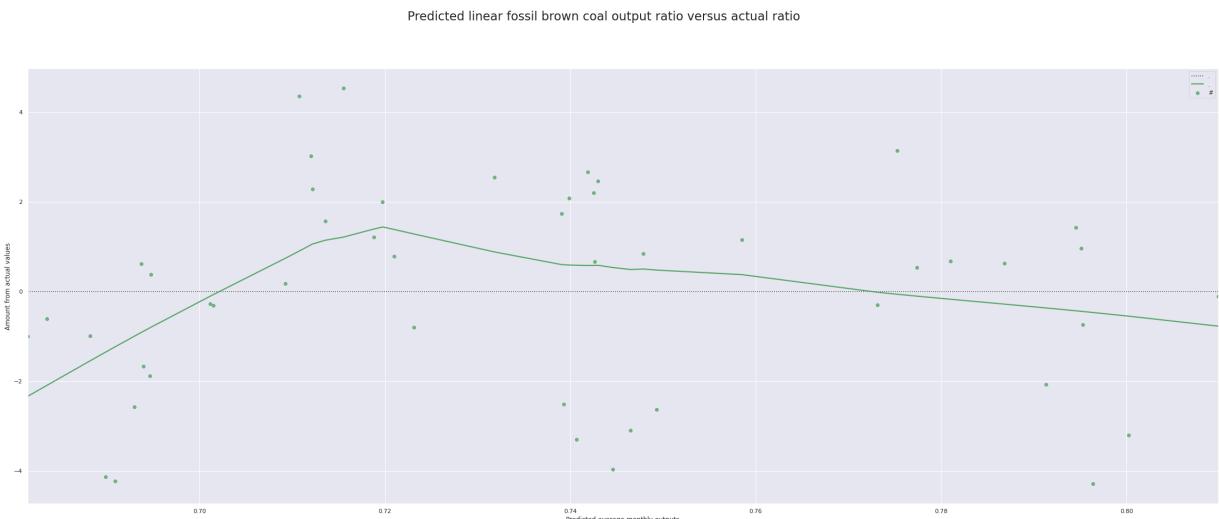


With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line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

```
In [ ]: # OLS predicted linear average monthly ratios versus actual ratios
sns.residplot(x = FossilBrownRegRatioPredict, y = pdToList, lowess = True, c
plt.suptitle("Predicted linear fossil brown coal output ratio versus actual ratio")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

Out[]: <matplotlib.legend.Legend at 0x7f81c9ee0d50>



As one can observe this residual plot, one may notice a hump forming a nonlinear pattern. However, the residuals are spread out without any distinct pattern which indicates homoskedascity, a lack of bias and constant

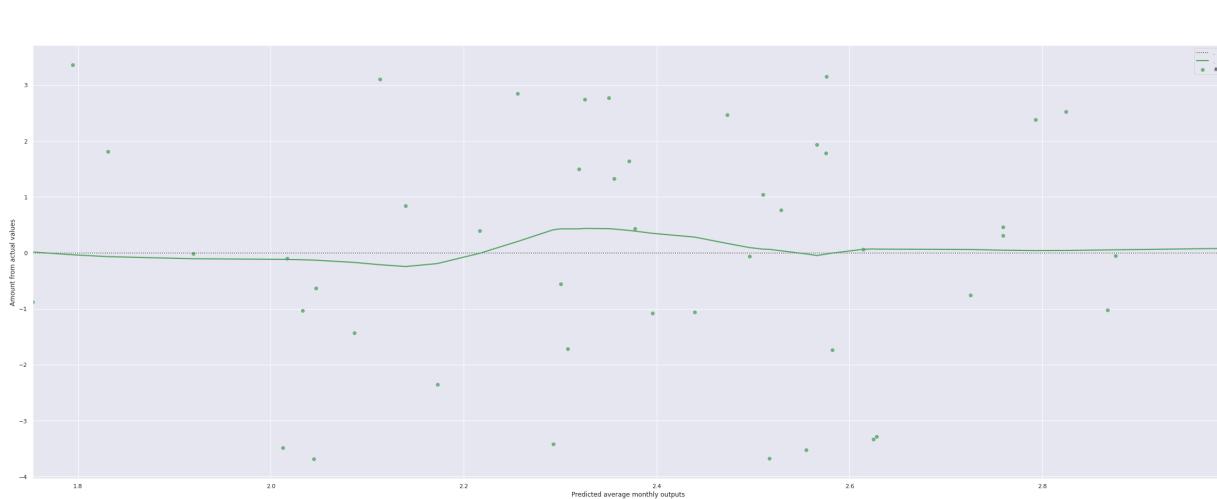
variance. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: #OLS predicted quadratic average monthly ratios versus actual ratios

sns.residplot(x = FossilBrownQuadRatioPredict , y = pdToList, lowess = True,
plt.suptitle("Predicted quadratic fossil brown coal output ratio versus actual")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c9e3ae10>
```



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.

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, 313.418154761904
[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,
79.49208333333331,
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: 0.039383    intercept: 40.221857
```

In []:

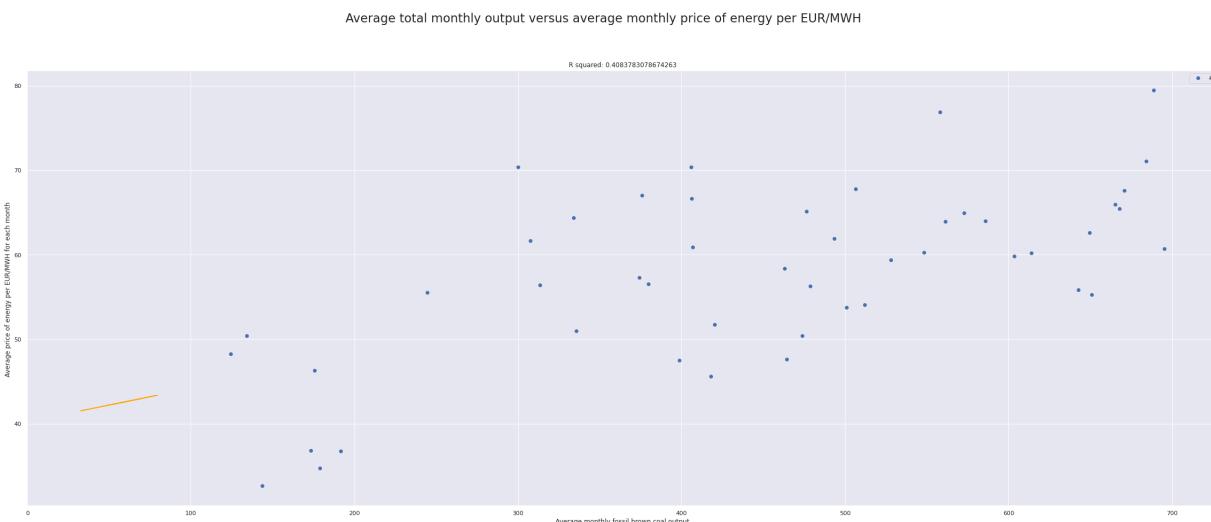
```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(fossil_brown,
print("slope: %f    intercept: %f" % (slope, intercept))

# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of e
```

```

plt.plot(fossil_brown,io2, "o")
f = lambda x: 0.039383 *x + 40.221857
plt.plot(x,f(x), c="orange", label="line of best fit")
plt.suptitle("Average total monthly output versus average monthly price of e")
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 output')
plt.show()
There is a moderately positive correlation the average monthly outputs of br

```



In []:

The next resource analyzed was hydro reservoir power.

Tn []:

```
In [ ]: modelhydrowater = stats.linregress([2572.3396998635744, 3712.690476190476, 364.9490188172043, 56.383854166666666, 55.522462987886975, 58.354083333333333, 57.29405913978498, 65.9749027777778, 71.07204301075271, 63.99806451612899, 60.254791666666634, 59.40676510067113, 60.72679166666668, 61.901760752688226, 45.57872311827956, 36.752083333333374, 36.81800807537014, 32.618666666666666, 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])
```

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,
                     "Hydro" : [2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.6
                     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, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Hydro': [2572.3396998635744, 3712.690476190476, 3081.6204
57604307, 2516.0125348189417, 2798.184139784946, 2531.1682892906815, 2412.52
55376344085, 1963.0631720430108, 2261.156944444444, 2276.9193548387098, 240
4.3902777777776, 1943.42799461642, 4075.5846774193546, 4881.568965517241, 39
01.5450874831763, 5119.295833333334, 4581.743279569892, 2983.793055555554,
2556.6971736204578, 2460.0201612903224, 2303.461111111111, 2500.48993288590
6, 2501.2972222222224, 2805.2970430107525, 2156.951612903226, 1874.849702380
9523, 2348.4589502018844, 1611.415277777779, 1639.616935483871, 1448.644444
4444444, 1290.983870967742, 1260.6908602150538, 1570.5097222222223, 1229.284
5637583894, 1371.338888888889, 1452.9569892473119, 2271.896505376344, 2773.2
038690476193, 4378.419919246298, 4159.898611111111, 2931.19623655914, 3362.5
291666666667, 2789.228802153432, 2296.0295698924733, 2267.641666666667, 217
2.1919463087247, 2763.0241935483873, 2665.9], 'Dates': ['2015-01', '2015-0
2', '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', '201
6-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '20
17-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2
018-01', '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, 4
8.83899276464233, 38.365623634435295, 33.94478947607304, 30.67378969794114,
37.52659136145235, 38.32761051675219, 39.59356672382155, 31.395358887784525,
89.41857951665217, 132.8242788644791, 105.96839132351556, 156.9437489780902
7, 132.07155415766834, 64.49168836821842, 53.82291915095909, 51.678556725831
78, 45.6985183799308, 41.548504312726116, 39.96892030690864, 41.501464085464
57, 27.13416886884827, 31.332197657357437, 46.08445667561115, 31.15777629179
6583, 30.4916685475955, 25.749915074142248, 23.36513255839751, 23.3097259009
60125, 28.13698743303958, 19.230019775602145, 20.95866127984951, 22.30128185
952668, 40.20203634637343, 45.55414023326174, 90.68859882531778, 82.53646537
900543, 47.55828742101278, 52.25526681467521, 41.149117937306535, 32.6307837
0414942, 29.48280466776494, 30.871568690104986, 41.21295837785713, 40.014400
988306384]

```

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(io2, 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:	Sun, 09 Oct 2022	Prob (F-statistic):	2.37e-05			
Time:	06:04:18	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.

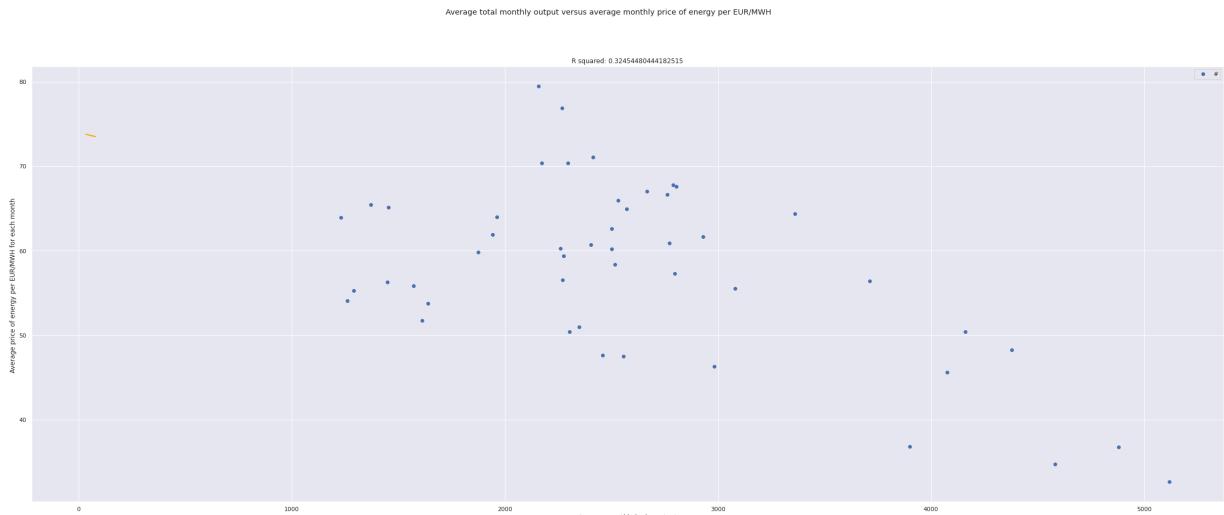
[2] The condition number is large, 8.18e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

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

# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of energy")
plt.plot(hydro_water,io2,"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 total monthly output versus average monthly price of energy")
plt.ylabel('Average price of energy per EUR/MWh for each month ')
plt.xlabel('Average monthly hydro output')
plt.show()
```

The blue dots represent the observations and the orange line is the linear regression 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()

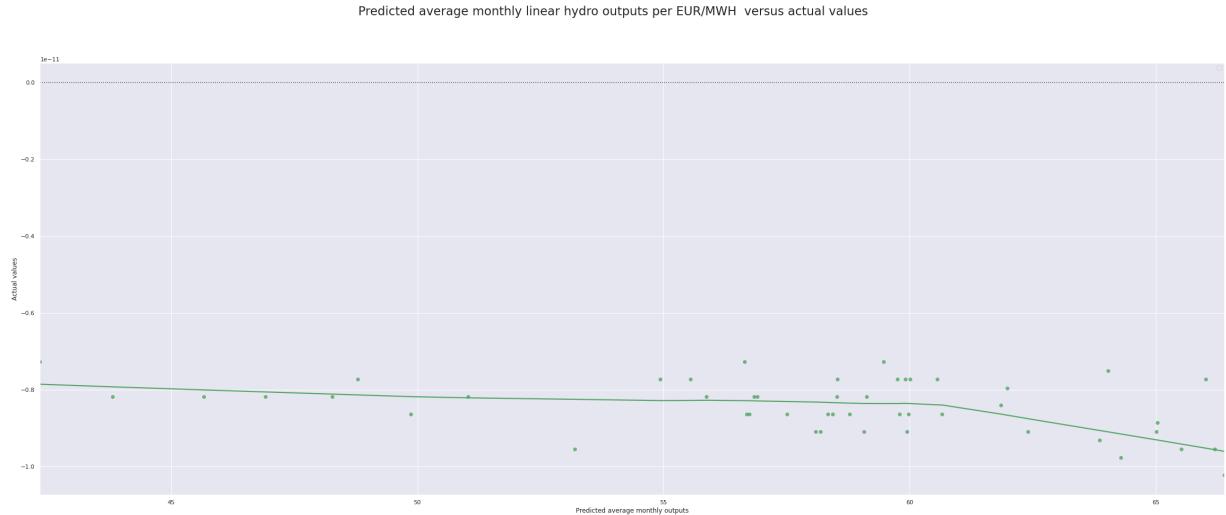
standardized_residualsHydro = influenceHydroreg.resid_studentized_internal

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 outputs per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend("#")
sns.residplot(x = predictionshydropower, y = hydro_water, lowess = True, color = "blue")
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c70b9450>
```

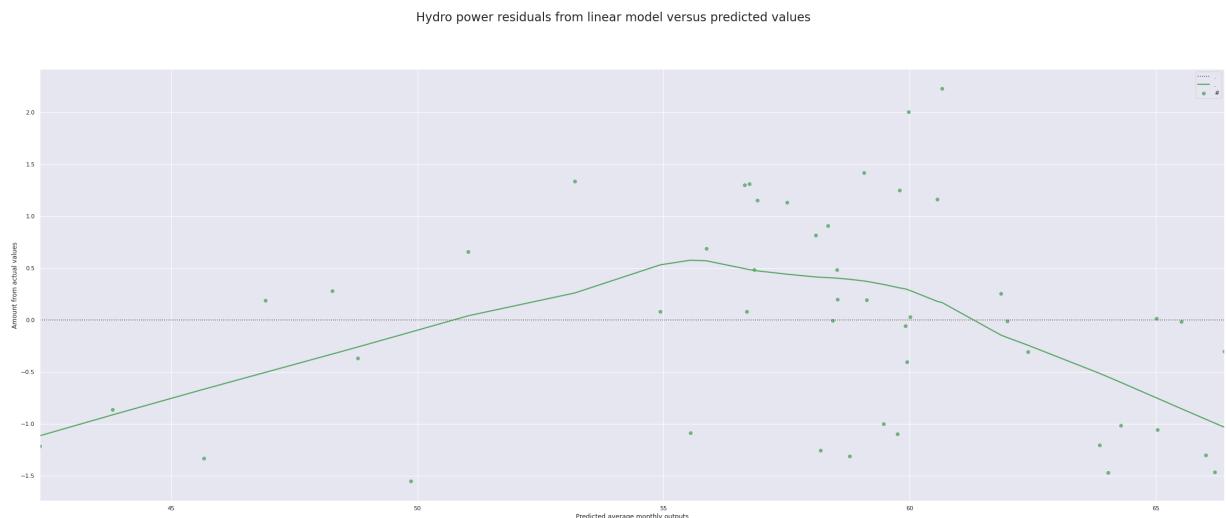


With the exception of a few outliers, the predictions seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations, while the dotted horizontal line represents the regression model. The green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

```
In [ ]: #Predicted OLS linear values versus residual values
sns.residplot(x = predictionshydropower, y =standardized_residualsHydro, low
```

plt.suptitle("Hydro power residuals from linear model versus predicted value")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#)

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c630a8d0>
```



As one can observe this residual plot, one may notice the hump in the fitted model, which form a nonlinear pattern, indicating bias and homoscedasticity. In addition, the residuals are spread out within the distinct pattern, meaning that a nonlinear model could be a better fit than a linear model. 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 [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(io2)))
Loghydrovalues = ((np.log(hydro_water)))
Log = np.polyfit(np.log(io2), hydro1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(hydro1), io2)
Hydro_Log = sm.OLS(io2, hydro1).fit()

Hydro_Logpred = Hydro_Log.predict(hydro1)

#OLS Logarithmic summary table
Hydro_Log.summary()
#Log
Log = np.polyfit(np.log(hydro_water), io2, 1)
print(Log)

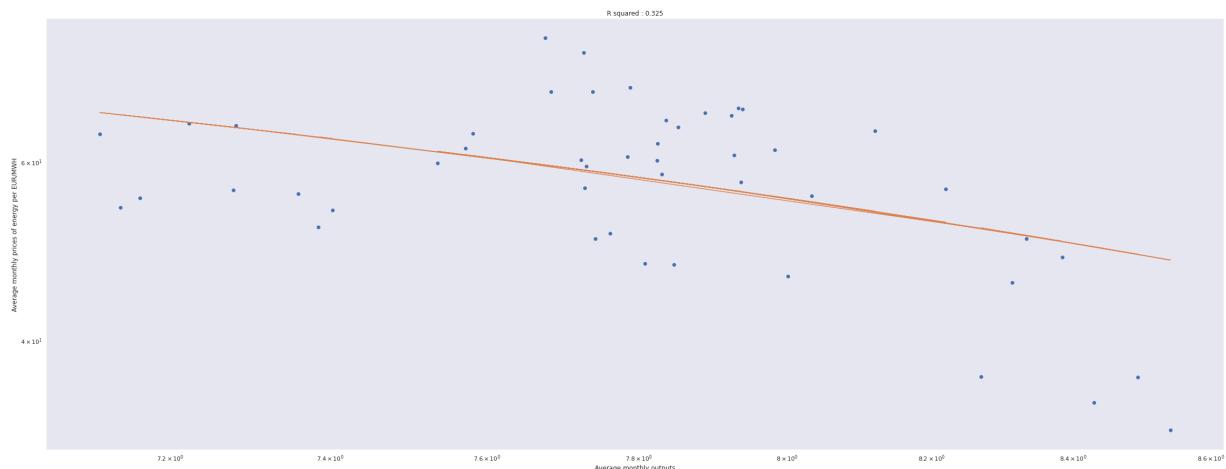
y = -13.40692102 * Loghydrovalues + 162.49392679

#Logarithmic OLS regression scatterplot
plt.suptitle("Logarithmic average monthly outputs versus average monthly price")
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, io2, "o")
plt.plot(Loghydrovalues, y)

[-13.40692102 162.49392679]

Out[ ]: [<matplotlib.lines.Line2D at 0x7f81c6fb3310>]
```

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



The blue dots represent the observations and the orange line is the linear 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. The blue dots represent the observations and the orange line is the linear 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: Sun, 09 Oct 2022 Prob (F-statistic): 2.37e-05
Time: 06:04:20 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)

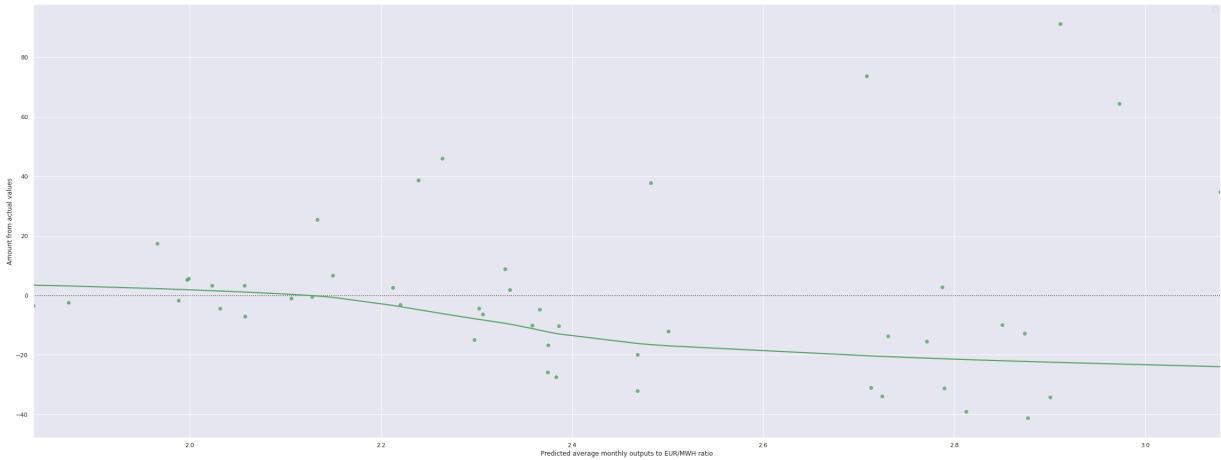
print(Hydro_Logpred) # OLS logarithmic predicted values
```

```
[ 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 [ ]: plt.suptitle("Predicted average monthly hydro output to EUR/MWH ratio versus
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")
plt.legend("..#")
HydroLogRatioPredict = Hydro_Logpred/predictionLog
sns.residplot(x = HydroLogRatioPredict, y = pdToListHydro, lowess = True, cc
#OLS Logarithmic predicted average monthly ratios versus actual ratios
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c70996d0>
```

Predicted average monthly hydro output to EUR/MWH ratio versus actual ratios

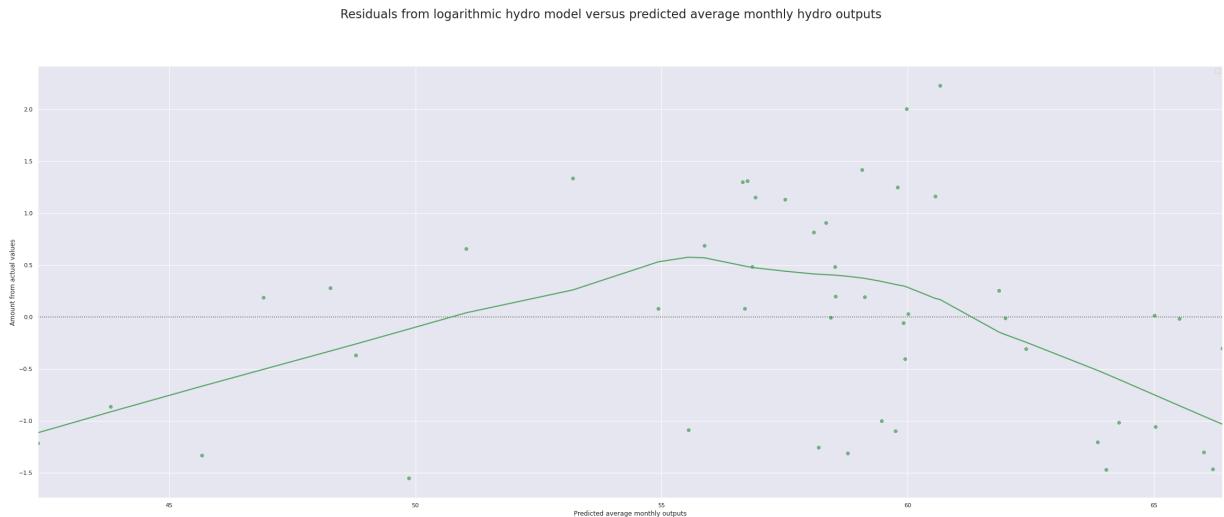


```
In [ ]:
```

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 [ ]: # OLS Logarithmic average monthly predictions versus residuals
plt.suptitle("Residuals from logarithmic hydro model versus predicted average monthly outputs")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = Hydro_Logpred, y = standardized_residualsHydroLog, lowess = ..)
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c6bf910>

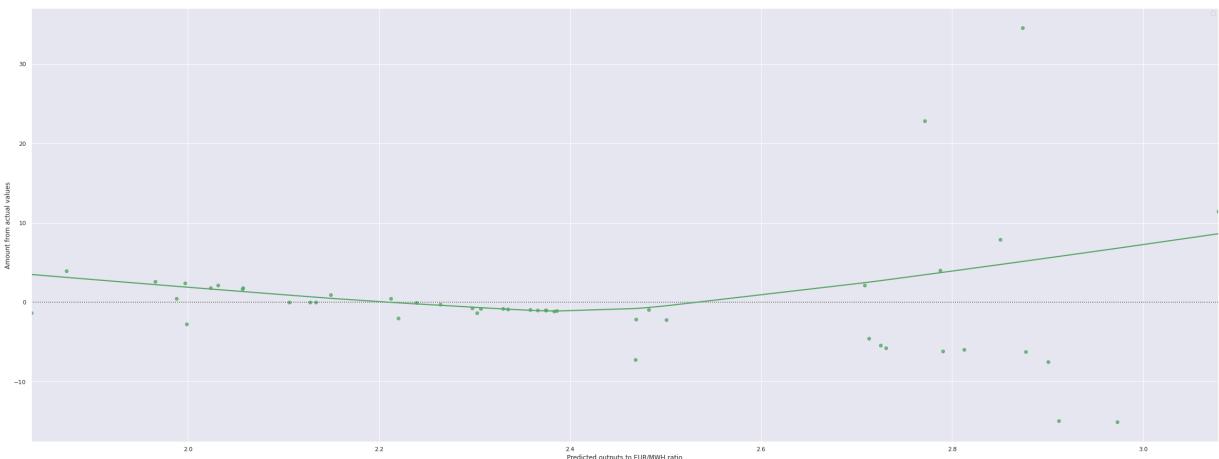


As one can observe this residual plot, one may notice that the lowess line has an arching hump; indicating heteroskedasticity, decreasing variance, and bias. This would indicate that this model does not fit the data. The green dots represent the respective observations while the dotted horizontal line 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 [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
plt.suptitle("Predicted average monthly hydro outputs to prices of energy per kWh ratio versus residuals")
plt.xlabel("Predicted outputs to EUR/MWh ratio")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = HydroLogRatioPredict, y = standardized_residualsHydroLog/s)
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c6bc17d0>

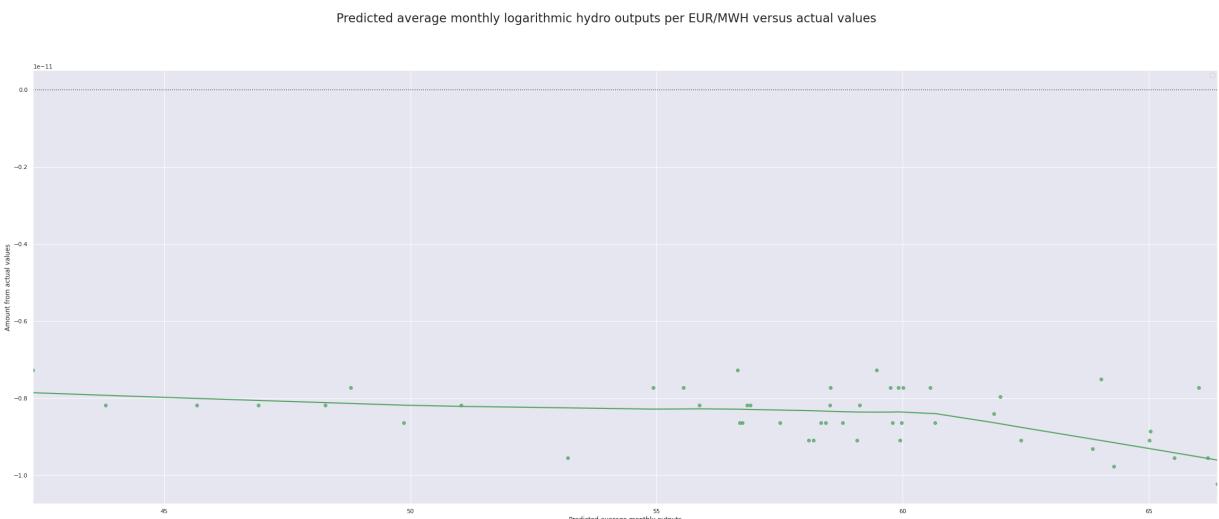
Predicted average monthly hydro outputs to prices of energy per EUR/MWh ratio versus respective hydro residuals



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 logarithmic average monthly values versus actual values
plt.suptitle("Predicted average monthly logarithmic hydro outputs per EUR/MWh")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#)
sns.residplot(x = Hydro_Logpred, y = hydro_water, lowess = True, color="g")
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c68c4e90>



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 [ ]: 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]
```

This is the quadratic model used for the average monthly hydro reservoir outputs versus the 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,io2,2))
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(io2, hydro2).fit()

# OLS Predicted Quadratic values
Hydro_ypred = Hydro_Quad.predict(hydro2)

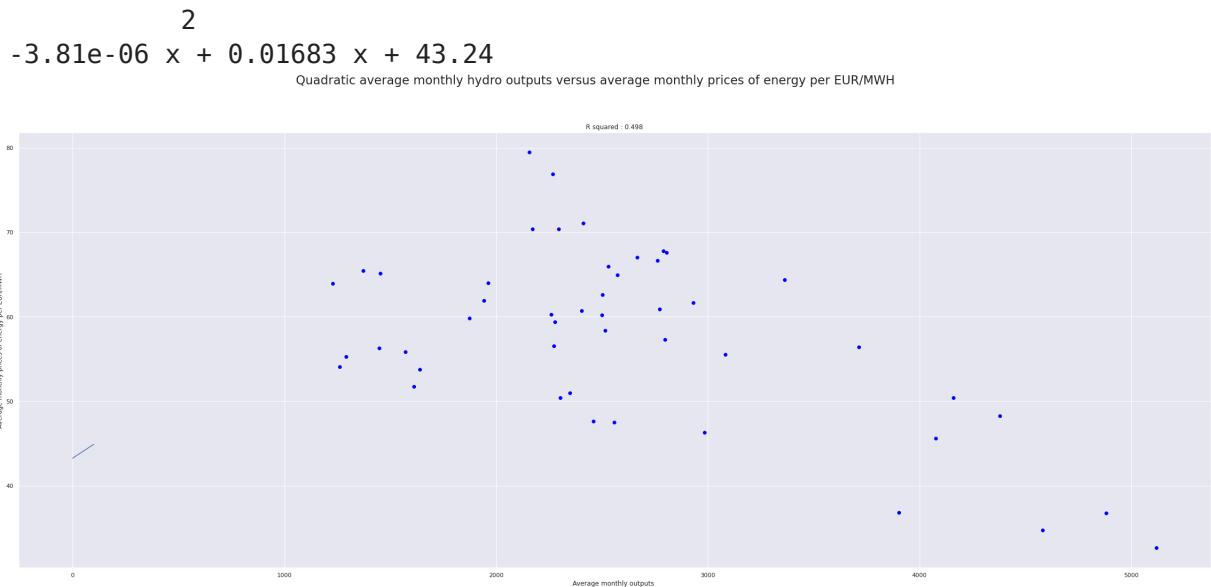
#OLS Quadratic Summary Table
Hydro_Quad.summary()

#Quadratic Scatterplots
polyline = np.linspace(start = 0, stop = 100 , num = 10)
plt.title("R squared : 0.498")
plt.plot(polyline, modelHydroquad(polyline))
```

```

plt.scatter(hydro_water,io2, color = 'blue')
plt.suptitle('Quadratic average monthly hydro outputs versus average monthly')
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 linear 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 []:

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:	Sun, 09 Oct 2022	Prob (F-statistic):	1.87e-07			
Time:	06:04:24	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_intercept
print(standardized_residualsHydroQuad)

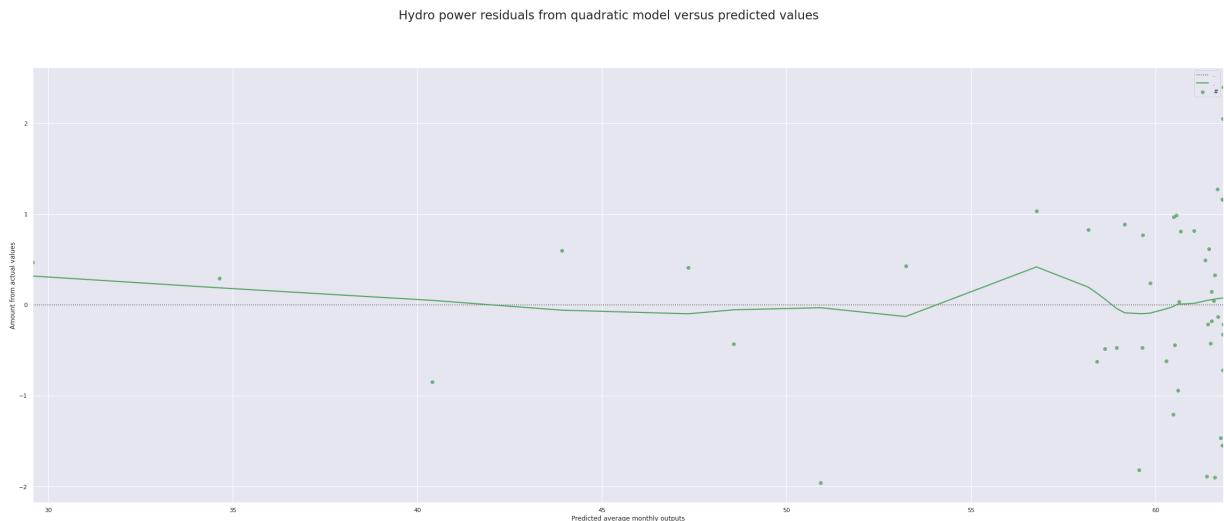
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = Hydro_ypred , y = standardized_residualsHydroQuad, lowess=True)

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 0x7f81c67fab10>
```



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

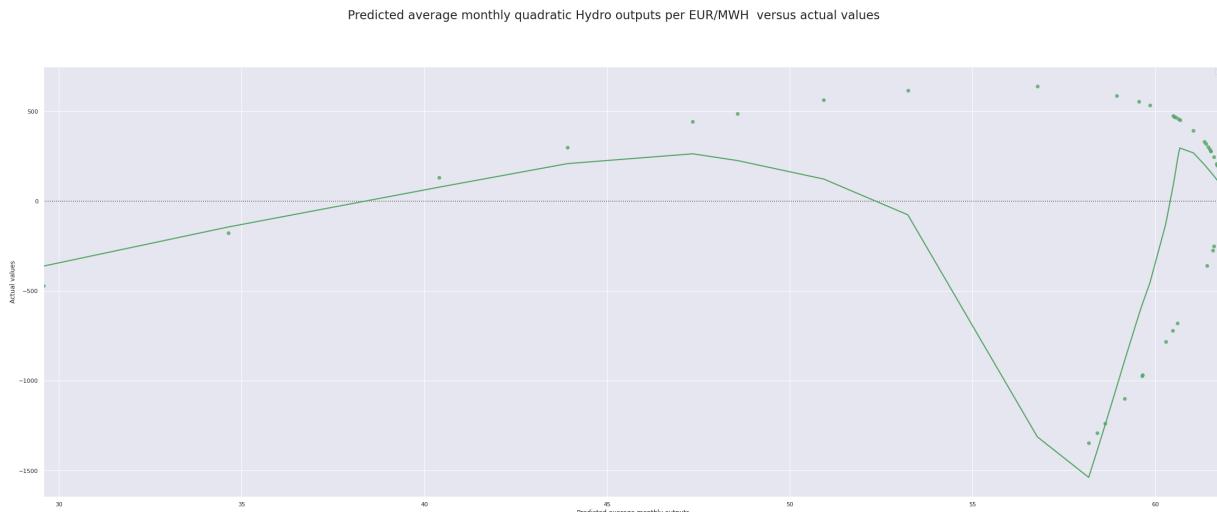
model does not fit the data. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
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 outputs per EUR/MWH")
plt.xlabel("Predicted average monthly outputs")
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 0x7f81c6b92c90>
```



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,
                     "Hydro" : [2572.3396998635744, 3712.690476190476, 3081.620457604307, 2516.6
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")
    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 Hydro Ratio")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function
plot_pacf(df_hydro["Ratio"])
plt.suptitle("Partial Autocorrelations of Hydro Ratio")
plt.ylabel('Partial Autocorrelations')
plt.xlabel('Lags')

plt.show

df_hydro['First Difference Ratio'] = df_hydro["Ratio"]- df_hydro["Ratio"].shift(1)
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) #Autocorrelation Function
print(Hydro_Ratio_Autocorrelations)
```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Hydro': [2572.3396998635744, 3712.690476190476, 3081.6204
57604307, 2516.0125348189417, 2798.184139784946, 2531.1682892906815, 2412.52
55376344085, 1963.0631720430108, 2261.156944444444, 2276.9193548387098, 240
4.3902777777776, 1943.42799461642, 4075.5846774193546, 4881.568965517241, 39
01.5450874831763, 5119.295833333334, 4581.743279569892, 2983.793055555554,
2556.6971736204578, 2460.0201612903224, 2303.461111111111, 2500.48993288590
6, 2501.2972222222224, 2805.2970430107525, 2156.951612903226, 1874.849702380
9523, 2348.4589502018844, 1611.415277777779, 1639.616935483871, 1448.644444
4444444, 1290.983870967742, 1260.6908602150538, 1570.5097222222223, 1229.284
5637583894, 1371.338888888889, 1452.9569892473119, 2271.896505376344, 2773.2
038690476193, 4378.419919246298, 4159.898611111111, 2931.19623655914, 3362.5
291666666667, 2789.228802153432, 2296.0295698924733, 2267.641666666667, 217
2.1919463087247, 2763.0241935483873, 2665.9], 'Dates': ['2015-01', '2015-0
2', '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', '201
6-11', '2016-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '20
17-06', '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2
018-01', '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

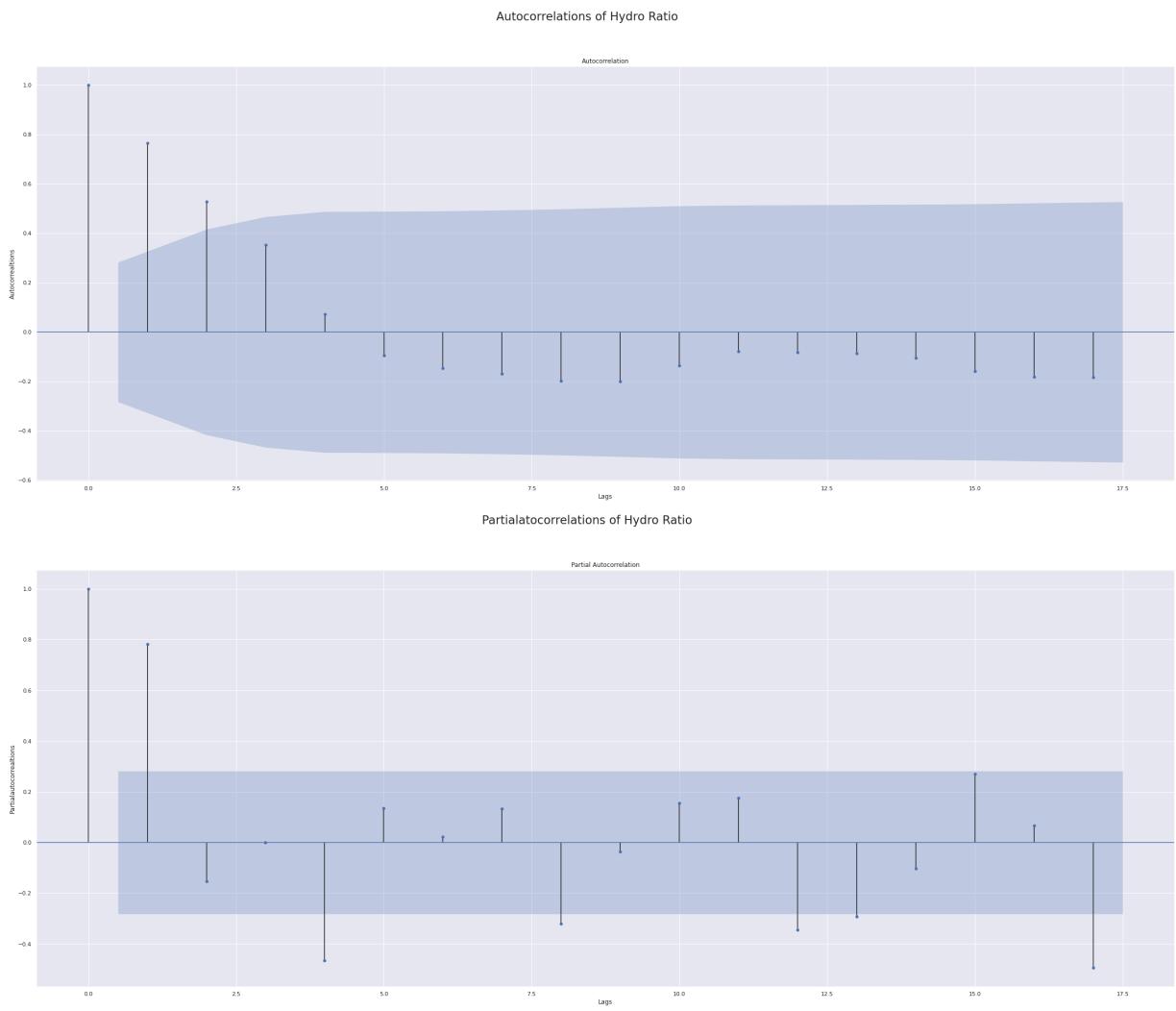
```

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, 4
8.83899276464233, 38.365623634435295, 33.94478947607304, 30.67378969794114,
37.52659136145235, 38.32761051675219, 39.59356672382155, 31.395358887784525,
89.41857951665217, 132.8242788644791, 105.96839132351556, 156.9437489780902
7, 132.07155415766834, 64.49168836821842, 53.82291915095909, 51.678556725831
78, 45.6985183799308, 41.548504312726116, 39.96892030690864, 41.501464085464
57, 27.13416886884827, 31.332197657357437, 46.08445667561115, 31.15777629179
6583, 30.4916685475955, 25.749915074142248, 23.36513255839751, 23.3097259009
60125, 28.13698743303958, 19.230019775602145, 20.95866127984951, 22.30128185
952668, 40.20203634637343, 45.55414023326174, 90.68859882531778, 82.53646537
900543, 47.55828742101278, 52.25526681467521, 41.149117937306535, 32.6307837
0414942, 29.48280466776494, 30.871568690104986, 41.21295837785713, 40.014400
988306384]
[ 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 * n * log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to a n 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 hydro reservoir outputs and the average monthly prices of energy per EUR/MWH.

```
In [ ]: df_hydro.describe(include = 'all') # Description table of Dataframes
```

Out[]:

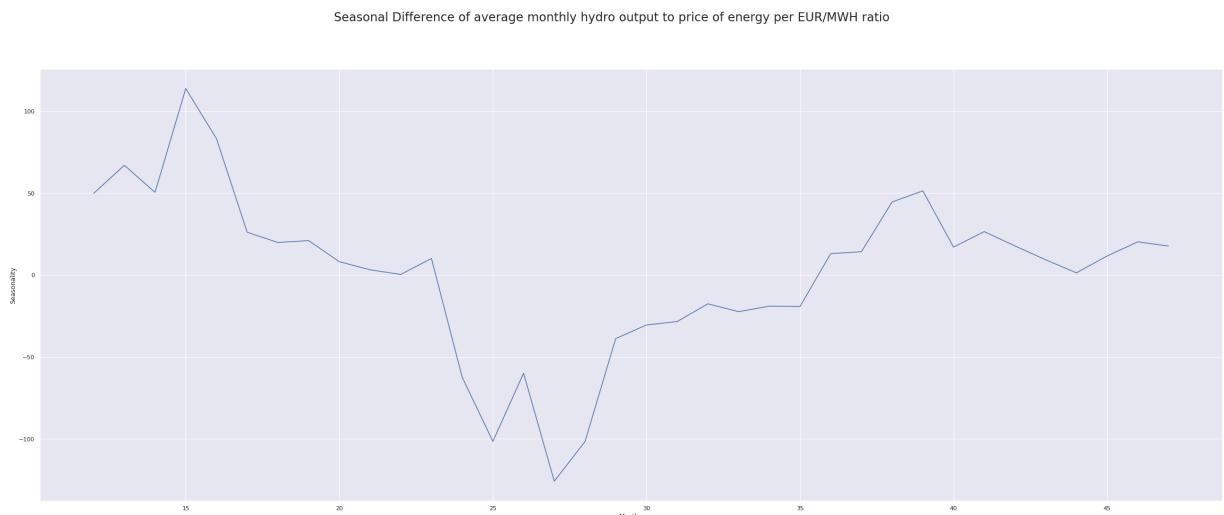
	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN
mean	57.859848	2608.982390	NaN	49.210055	0.008700	1.983871
std	10.320573	950.659135	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 0x7f81c5990150>



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)

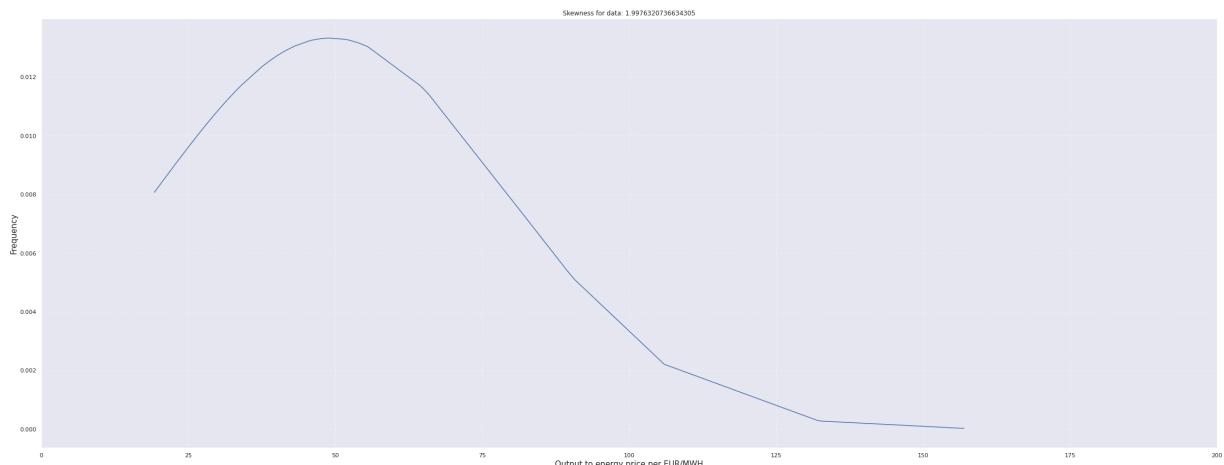
plt.plot(df_hydro["Ratio"].sort_values(), hydroResultspdf)
plt.xlim([0,200])
plt.xlabel("Output to energy price per EUR/MWH", size=15)
plt.title(f'Skewness for data: {skew(df_hydro["Ratio"])}')
plt.suptitle("Frequency distribution of output to energy price per EUR/MWH ratios (scaled in decimals)")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

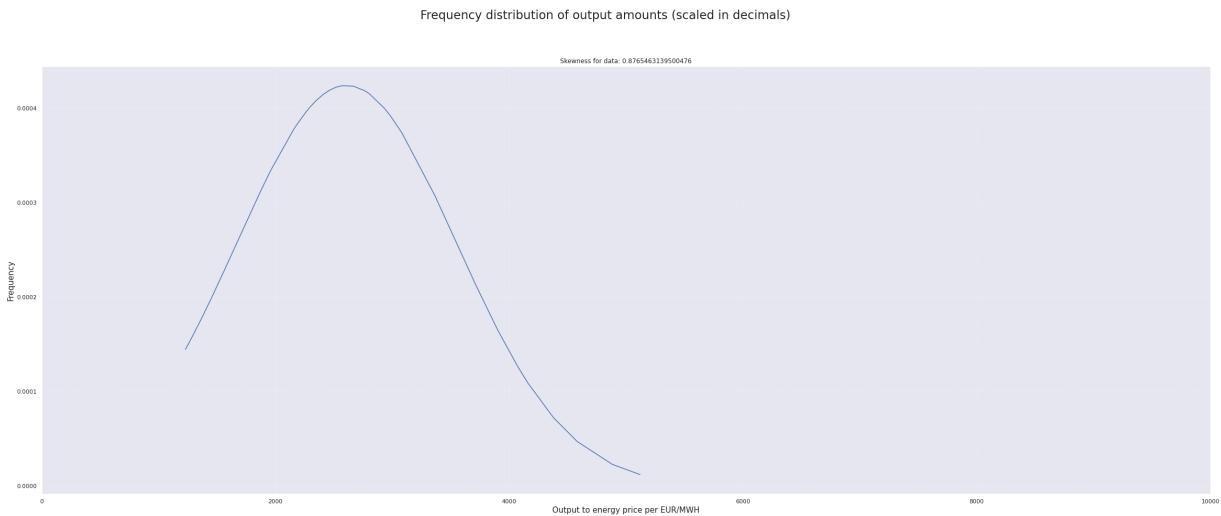
#Bell Curves
hydroResults_mean = np.mean(df_hydro["Hydro"])
hydroResults_std = np.std(df_hydro["Hydro"])

hydroResultspdf = stats.norm.pdf(df_hydro["Hydro"].sort_values(), hydroResults_mean, hydroResults_std)

plt.plot(df_hydro["Hydro"].sort_values(), hydroResultspdf)
plt.xlim([0,10000])
plt.xlabel("Output in MWH ", 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 output amounts (scaled in decimals)")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()
```

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





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 [ ]: 
```

```
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")
    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 Hydro")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Plot
plot_pacf(df_hydro["Hydro"])
plt.suptitle("Partial autocorrelations of Hydro")
plt.ylabel('Partial autocorrelations')
plt.xlabel('Lags')

plt.show
Hydro_Autocorrelations = sm.tsa.acf(df_hydro["Hydro"], fft=False) #Autocorrelation Function
print(Hydro_Autocorrelations)
df_hydro['First Difference'] = df_hydro["Hydro"] - df_hydro["Hydro"].shift(1)
df_hydro['Seasonal Difference']=df_hydro["Hydro"] - df_hydro["Hydro"].shift(12)
df_hydro.head()

```

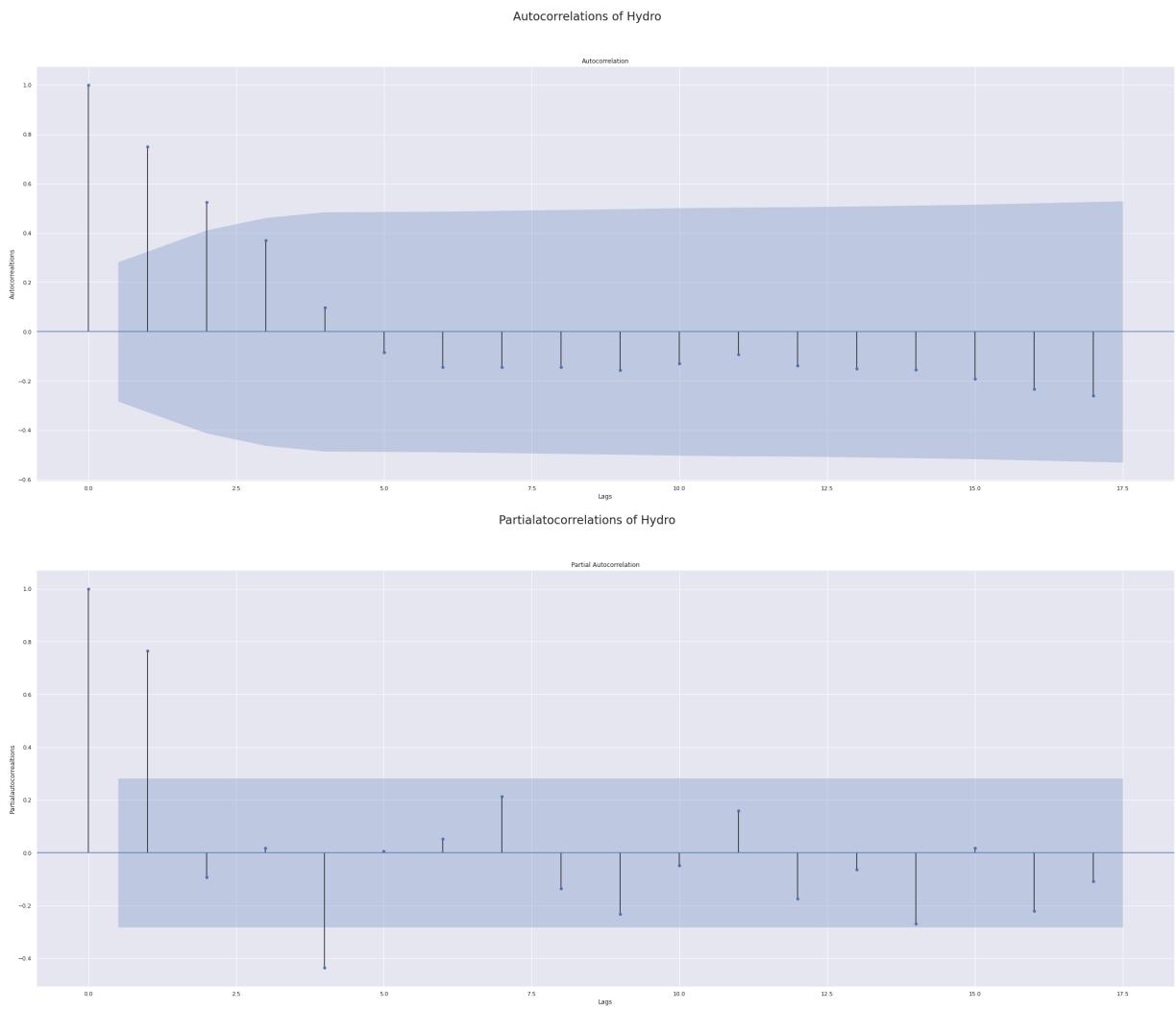
ADF Test Statistic : -3.1673141085291867
p-value : 0.021956352091091067
#Lags Used : 3
Number of Observations : 44
strong evidence against the null hypothesis(H_0), reject the null hypothesis.
Data is stationary

[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 * n * p.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to a n integer to silence this warning.
FutureWarning,

Out[]:

	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	Fir Differen
0	64.949019	2572.339700	2015-01	39.605521	NaN	NaN	NaN
1	56.383854	3712.690476	2015-02	65.846695	26.241175	NaN	1140.3507
2	55.522463	3081.620458	2015-03	55.502229	-10.344466	NaN	-631.0700
3	58.354083	2516.012535	2015-04	43.116306	-12.385923	NaN	-565.6079
4	57.294059	2798.184140	2015-05	48.838993	5.722686	NaN	282.1716



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)
df_hydro['Seasonal Difference']=df_hydro["Ratio"]- df_hydro["Ratio"].shift(12)
df_hydro.head()
```

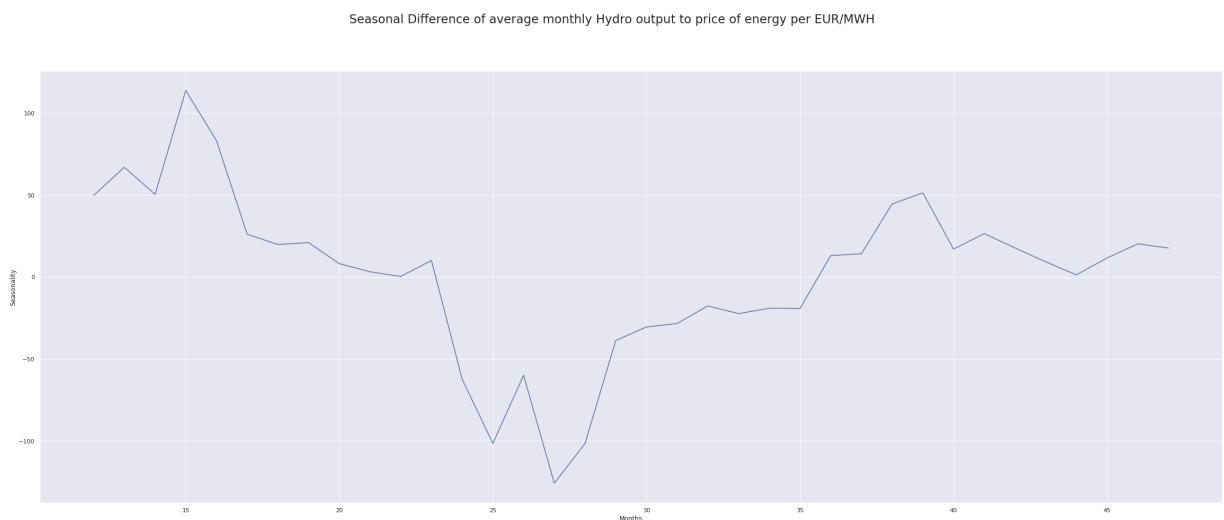
```
Out[ ]:
```

	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	Firs Difference
0	64.949019	2572.339700	2015-01	39.605521	NaN	NaN	NaN
1	56.383854	3712.690476	2015-02	65.846695	26.241175	NaN	26.241175
2	55.522463	3081.620458	2015-03	55.502229	-10.344466	NaN	-10.344466
3	58.354083	2516.012535	2015-04	43.116306	-12.385923	NaN	-12.385923
4	57.294059	2798.184140	2015-05	48.838993	5.722686	NaN	5.722686

```
In [ ]:
```

```
plt.suptitle("Seasonal Difference of average monthly Hydro output to price c  
plt.ylabel('Seasonality')  
plt.xlabel('Months')  
df_hydro['Seasonal Difference'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c5748050>
```



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 table of Dataframes
```

Out[]:

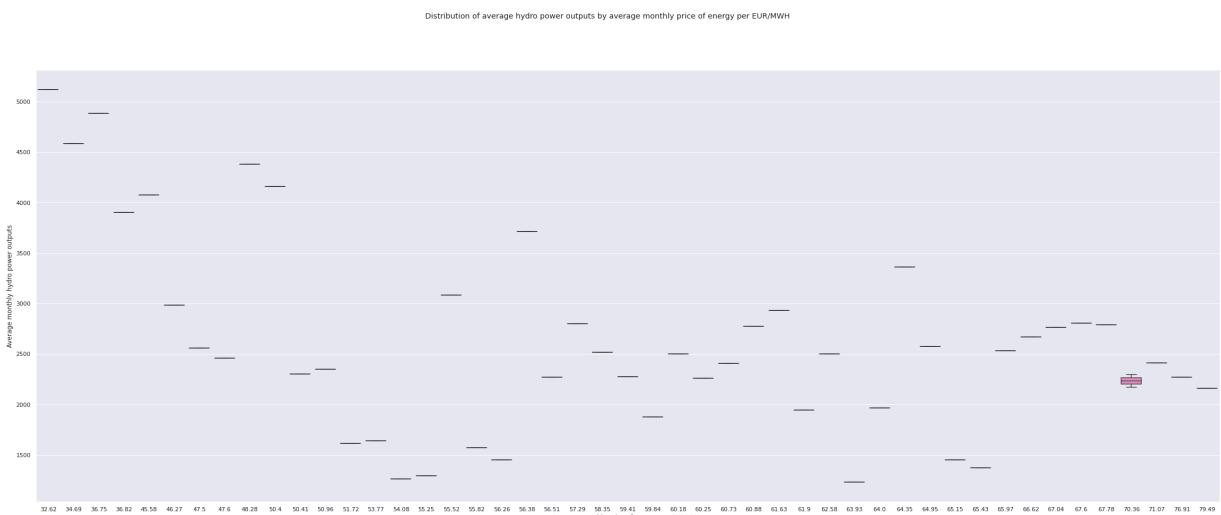
	Price	Hydro	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio	Di
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	4
unique	NaN	NaN	48	NaN	NaN	NaN	
top	NaN	NaN	2015-01	NaN	NaN	NaN	
freq	NaN	NaN	1	NaN	NaN	NaN	
mean	57.859848	2608.982390	NaN	49.210055	0.008700	1.983871	
std	10.320573	950.659135	NaN	30.241054	20.843801	49.826097	2
min	32.618667	1229.284564	NaN	19.230020	-67.579866	-125.785973	-6
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	5

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, 3081.6204
plt.show()
```



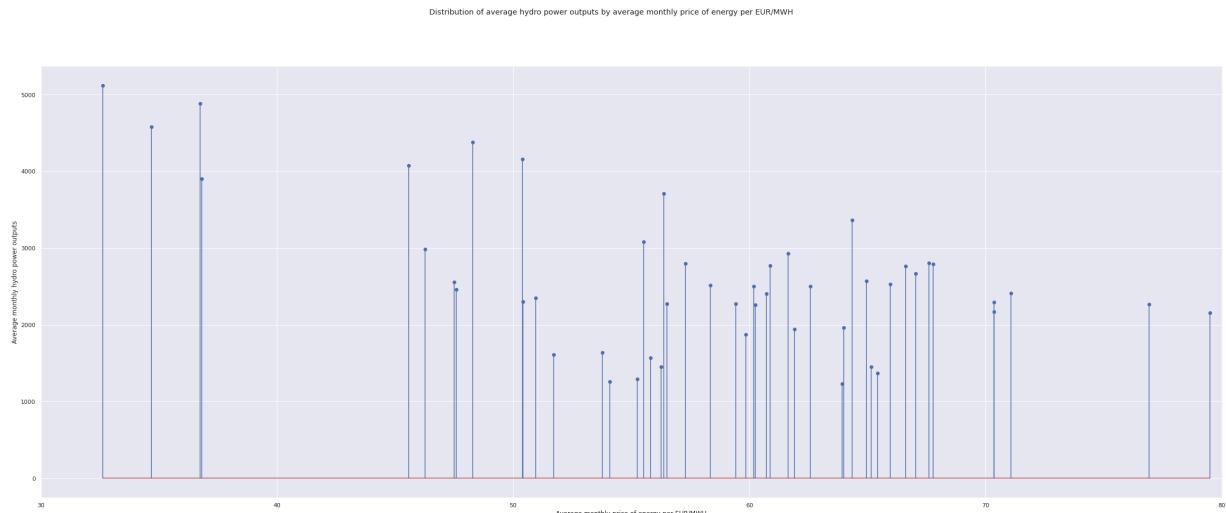
This box and whisker plot depicts the frequencies between the distribution of the monthly average output of fossil brown coal produced and its the average monthly prices of energy per EUR/MWH. As one can observe, the highest concentrated amount of hydro power produced, which was in between 2000 and 2500 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

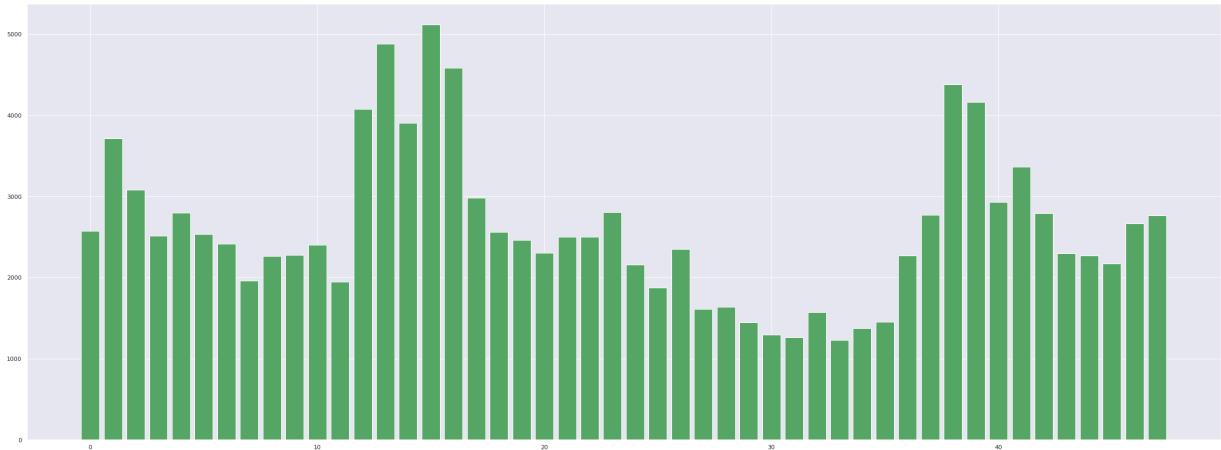
the price of energy per EUR/MWH was at roughly 70.36 euros/MWH. When the price of energy per EUR/MWH was at its lowest (at approximately 32.62 euros per MWH), hydro power output was at its highest, at slightly above 5000 units. At approximately 60.73 EUR/MWH, hydropower produced the most units at roughly 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 [ ]: #Histograms
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, 2261.156944444444: 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, 2501.2972222222224: 22, 2805.2970430107525: 23, 2156.951612903226: 24, 1874.8497023809523: 25, 2348.4589502018844: 26, 1611.4152777777779: 27, 1639.616935483871: 28, 1448.644444444444: 29, 1290.983870967742: 30, 1260.6908602150538: 31, 1570.509722222223: 32, 1229.2845637583894: 33, 1371.338888888889: 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, 2267.641666666667: 44, 2172.1919463087247: 45, 2665.9: 46, 2763.0241935483873: 47}
```



```
In [ ]:
```

```
In [ ]:
```

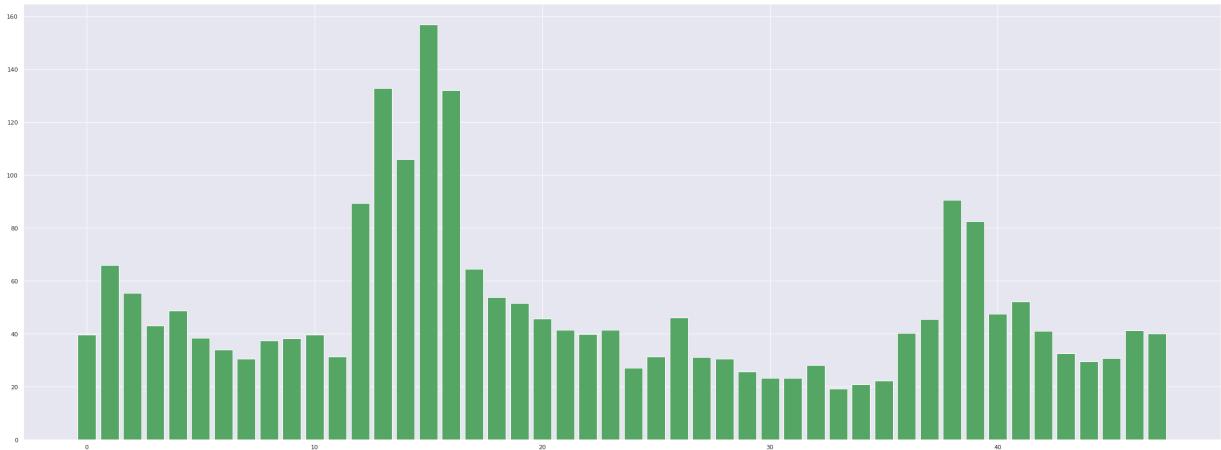
The green bars represent the observation value for each respective month. This histogram is multimodal, containing fluctuating patterns of gradual decrease in the average monthly outputs followed by a sudden increase in the average monthly outputs. This histogram is multimodal; there isn't an indication 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}") #Histograms
print(pdToListHydro_Dict)

plt.bar(list(pdToListHydro_Dict.values()), pdToListHydro_Dict.keys(), color=
print(dicDates)
plt.show()

{39.60552055610406: 0, 65.8466954957713: 1, 55.50222903974214: 2, 43.1163063
679167: 3, 48.83899276464233: 4, 38.365623634435295: 5, 33.94478947607304: 6, 30.67378969794114: 7, 37.52659136145235: 8, 38.32761051675219: 9, 39.5935
6672382155: 10, 31.395358887784525: 11, 89.41857951665217: 12, 132.824278864
4791: 13, 105.96839132351556: 14, 156.94374897809027: 15, 132.0715541576683
4: 16, 64.49168836821842: 17, 53.82291915095909: 18, 51.67855672583178: 19,
45.6985183799308: 20, 41.548504312726116: 21, 39.96892030690864: 22, 41.5014
6408546457: 23, 27.13416886884827: 24, 31.332197657357437: 25, 46.0844566756
1115: 26, 31.157776291796583: 27, 30.4916685475955: 28, 25.749915074142248:
29, 23.36513255839751: 30, 23.309725900960125: 31, 28.13698743303958: 32, 1
9.230019775602145: 33, 20.95866127984951: 34, 22.30128185952668: 35, 40.2020
3634637343: 36, 45.55414023326174: 37, 90.68859882531778: 38, 82.53646537900
543: 39, 47.55828742101278: 40, 52.25526681467521: 41, 41.149117937306535: 4
2, 32.63078370414942: 43, 29.48280466776494: 44, 30.871568690104986: 45, 41.
21295837785713: 46, 40.014400988306384: 47}
```



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, 3  
[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 []:

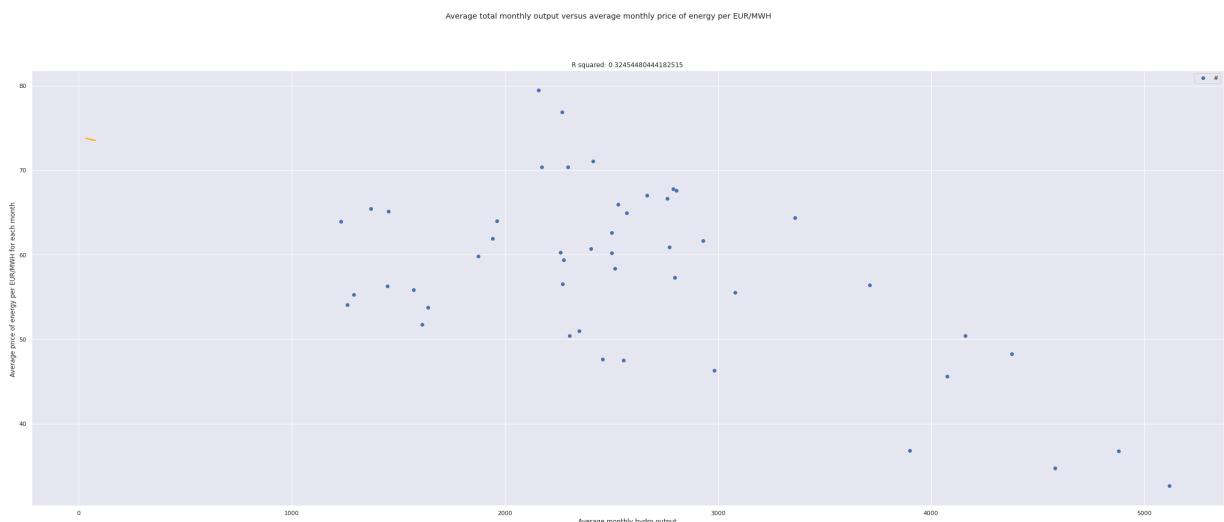
```
slope: -0.006186    intercept: 73.998036
```

```
In [ ]: #slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(hydro_water,io2)
print("slope: %f    intercept: %f" % (slope, intercept))

# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of energy per EUR/MWH")
plt.plot(hydro_water,io2,"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 total monthly output versus average monthly price 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()
```

The blue dots represent the observations and the orange line is the linear model.



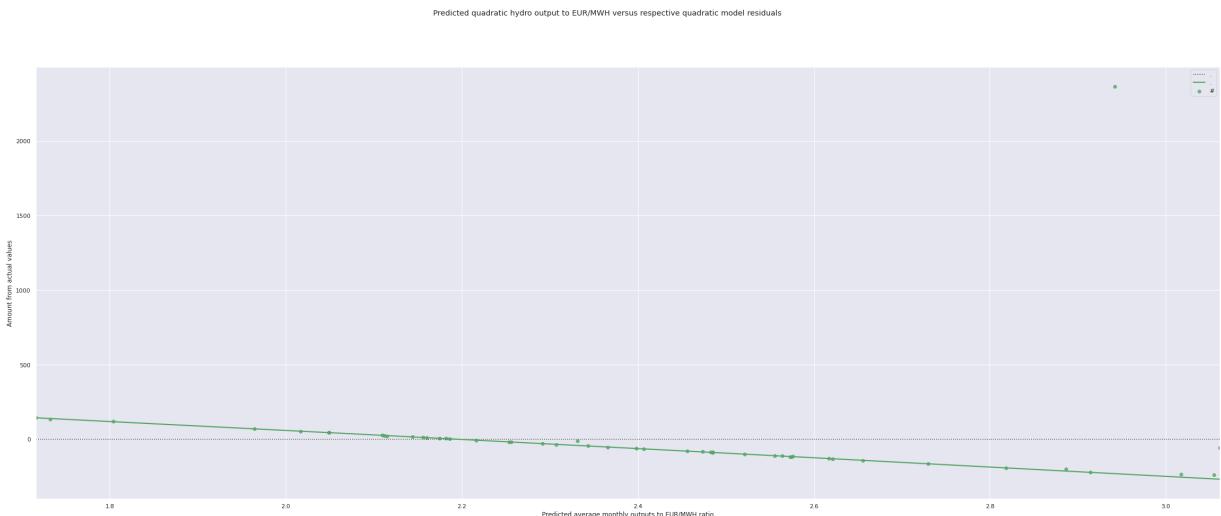
```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
HydroQuadRatioPredict = Hydro_ypred/ypred

sns.residplot(x = HydroQuadRatioPredict , y = standardized_residualsHydroQuadRatioPredict)

plt.suptitle("Predicted quadratic hydro output to EUR/MWH versus respective ratio")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

Out[]: <matplotlib.legend.Legend at 0x7f81c4ed8d90>



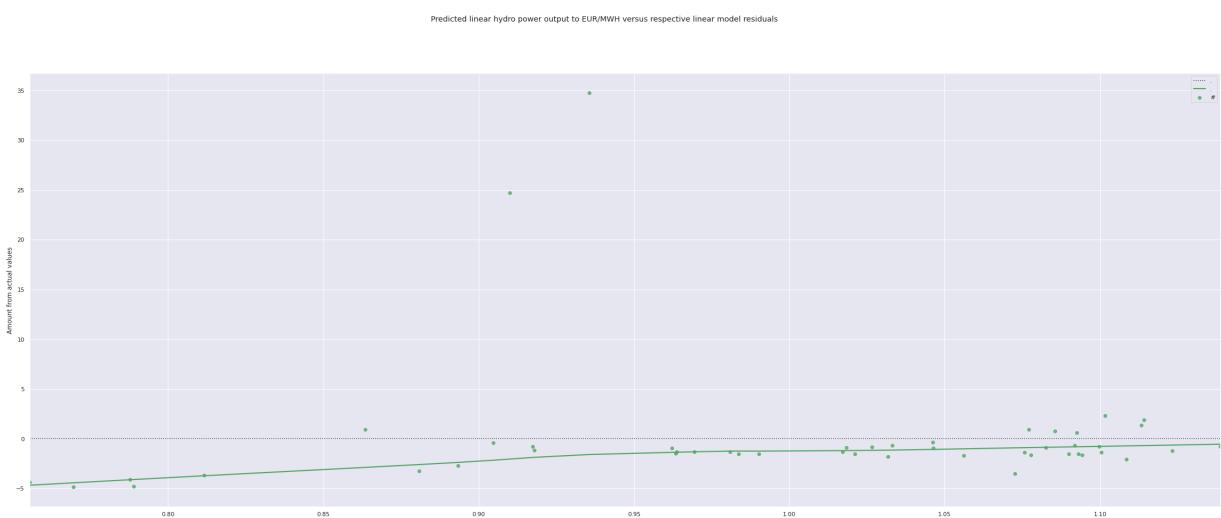
With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line 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/star
plt.suptitle("Predicted linear hydro power output to EUR/MWh versus respecti
plt.xlabel("Predicted average monthly outputs to EUR/MWh ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c4e45090>
```



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 [ ]: #OLS predicted quadratic average monthly ratios versus actual ratios
sns.residplot(x = HydroQuadRatioPredict , y = pdToListHydro, lowess = True,
plt.suptitle("Predicted quadratic hydro power output ratio versus actual rat
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

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

Out[]: <matplotlib.legend.Legend at 0x7f81c4e1f210>



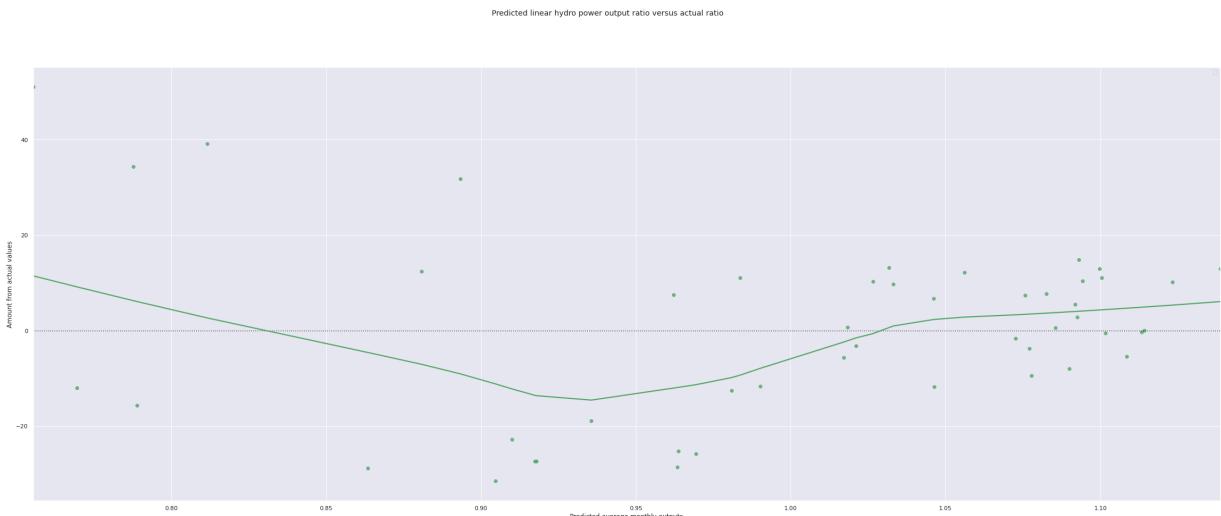
With the exception of few outliers, the predictions seem to be nearly the same as the actual values. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green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

```
In [ ]: # OLS predicted linear average monthly ratios versus actual ratios
plt.suptitle("Predicted linear hydro power output ratio versus actual ratio"
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#")

sns.residplot(x = HydroRegRatioPredict, y = pdToListHydro, lowess = True, co
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c4d48f90>



As one can observe this residual plot, one may notice the 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.

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, 55.5224629878869,
"Nuclear" : [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068
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, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Nuclear': [6665.969986357435, 6681.1235119047615, 6687.91
3862718708, 6068.16991643454, 5403.817204301075, 5659.276773296245, 6483.623
655913979, 6437.845430107527, 6466.175, 5854.012096774193, 5973.075, 6626.02
9609690444, 6223.8494623655915, 6159.264367816092, 6699.888290713325, 6706.4
06944444445, 5714.009408602151, 6647.463888888885, 6628.9220430107525, 663
3.3494623655915, 6675.755555555555, 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, 6655.3212
3655914, 6621.998611111111, 6539.120805369127, 5821.1518817204305, 5403.4972
2222222222], '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
[102.63388281689778, 118.49355831823476, 120.45420002671308, 103.98877970152
685, 94.3172343770747, 85.77923627046935, 91.22607682648228, 100.59437701408
982, 107.31387199496582, 98.54116929029787, 98.35979863363437, 107.041052291
92045, 136.55164156780614, 167.589529876522, 181.97313328298446, 205.6002783
0008724, 164.7098182979096, 143.678251667955, 139.55033034817424, 139.349234
5580292, 132.44075903166618, 109.27916930870813, 88.43406639744545, 93.58618
021036858, 85.16466524444257, 112.62559996020988, 132.5738976103446, 129.092
27137597384, 103.42141579325329, 107.78619520307156, 110.71705890251806, 12
3.43403396826045, 115.15739057905957, 93.91829892025801, 85.05519096023419,
104.60483304468198, 118.97813552223086, 105.61235177921166, 126.174828549310
13, 109.20822640101433, 88.67218166973136, 87.07676514860786, 90.30989575192
407, 94.58429917431938, 86.09609464822834, 92.93511904364472, 86.82764731945
06, 81.1049568960575]

```

This is the logarithmic average monthly nuclear outputs verusus the average monthly prices of enegery per EUR/MWH.

```

In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(io2)))
Lognuclearvalues = ((np.log(nuclear)))
Log = np.polyfit(np.log(io2), nuclear1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(nuclear1), io2)
Nuclear_Log = sm.OLS(io2, nuclear1).fit()

Nuclear_Logpred = Nuclear_Log.predict(nuclear1)
#OLS Logarithmic summary table
Nuclear_Log.summary()
#Log
Log = np.polyfit(np.log(nuclear), io2, 1)

```

```

print(Log)

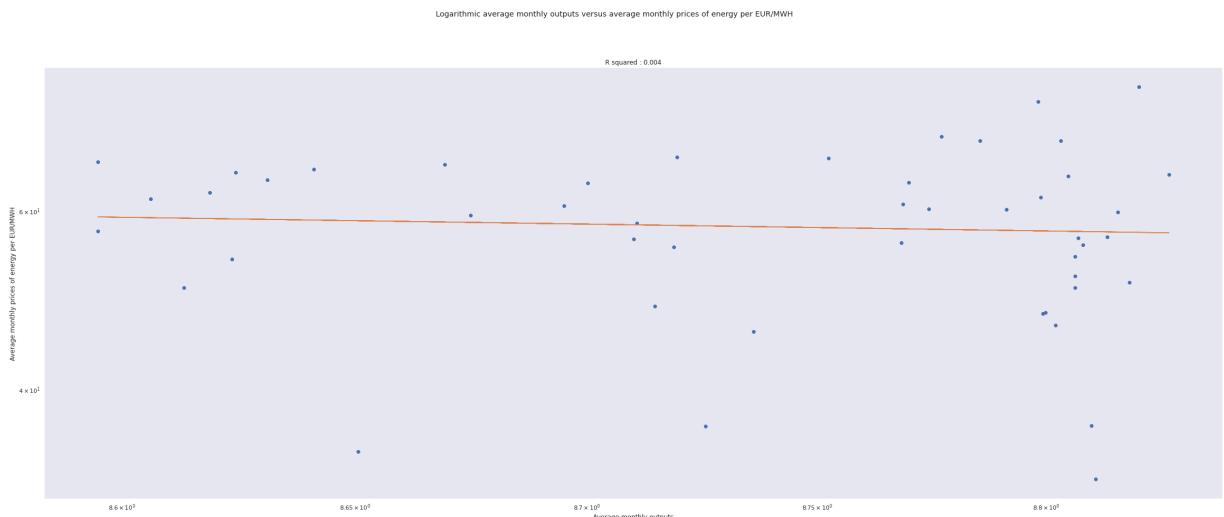
y = -9.05428682 * Lognuclearvalues + 136.99359892

#Logarithmic OLS regression scatterplot
plt.suptitle("Logarithmic average monthly outputs versus average monthly pri
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, io2, "o")
plt.plot(Lognuclearvalues, y)

```

[-9.05428682 136.99359892]

Out[]: [<matplotlib.lines.Line2D at 0x7f81c4e9ad50>]



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 logarithmic model of best fit.

```

In [ ]: influenceNuclearLog = Nuclear_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsNuclearLog = influenceNuclearLog.resid_studentized_int

print(standardized_residualsNuclearLog)

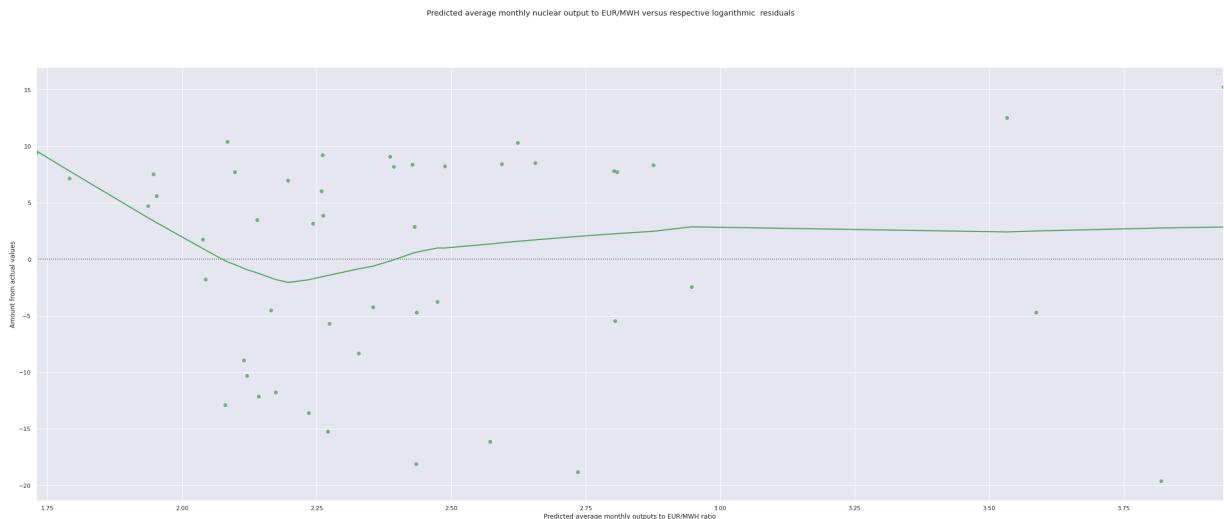
print(Nuclear_Logpred) # OLS logarithmic predicted values

```

```
[ 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 average monthly nuclear output to EUR/MWH versus res")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")
plt.legend("..#")
NuclearLogRatioPredict = Nuclear_Logpred/predictionLog
sns.residplot(x = NuclearLogRatioPredict, y = pdToListNuclear, lowess = True
               #OLS Logarithmic predicted average monthly ratios versus actual ratios
```

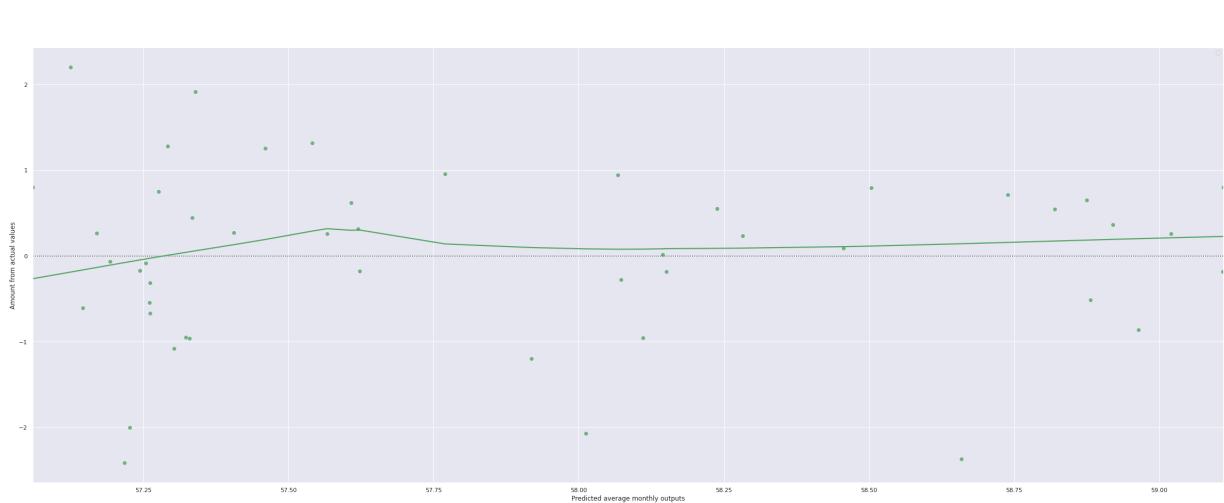
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c4f65a90>
```



As one can observe, there is a decreasing trend in the variance of the residual values versus the actual values. This indicates that there is heteroskedasticity and no bias. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted nuclear residuals from model versus predicted averag
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend("..#")
sns.residplot(x = Nuclear_Logpred, y = standardized_residualsNuclearLog, low
               # OLS Logarithmic average monthly predictions versus residuals
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c50283d0>
```

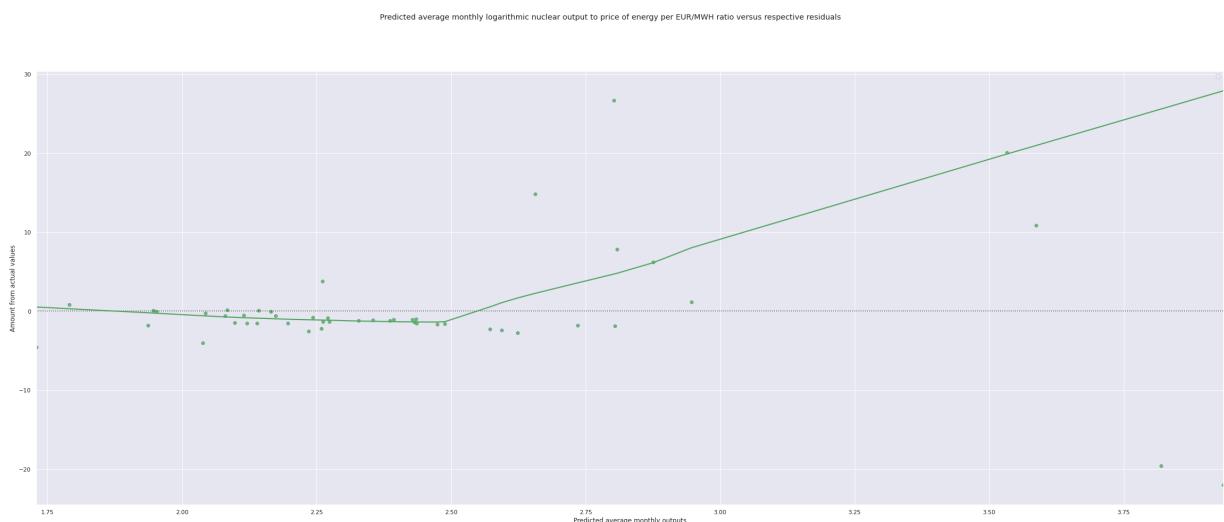


As one can observe this residual plot, one may notice an arching hump in the fitted model, which forms a nonlinear pattern. However, the observations are spread out without a distinct pattern, indicating constant variance, a lack of bias, and homoscedasticity. The green dots represent the respective 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 [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals

plt.suptitle("Predicted average monthly logarithmic nuclear output to price")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#")
sns.residplot(x = NuclearLogRatioPredict, y = standardized_residualsNuclearL
```

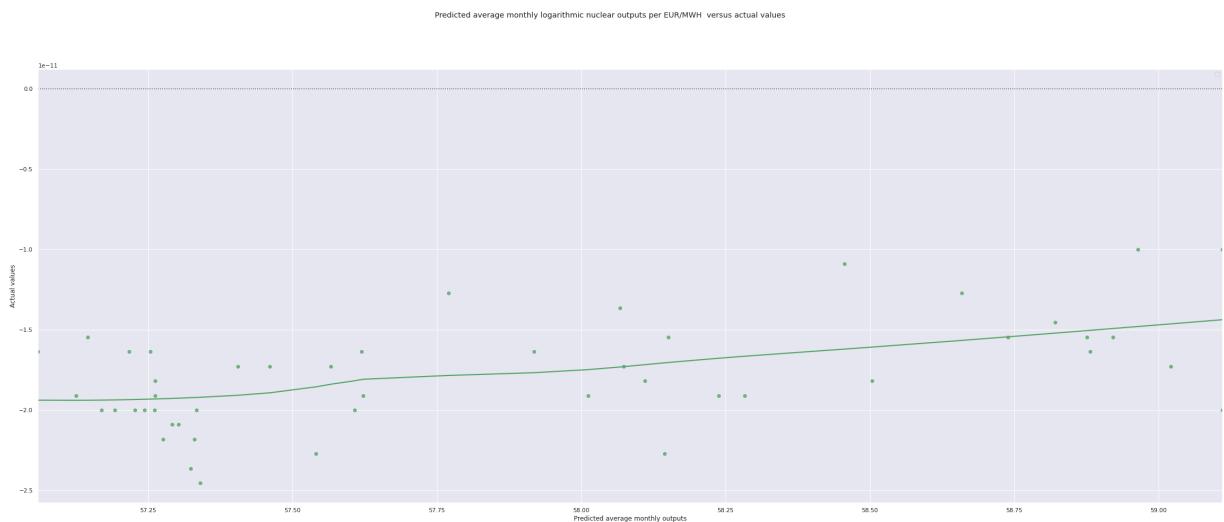
```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c565d990>
```



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 outputs per EUR/MWH versus actual values")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Nuclear_Logpred, y = nuclear, lowess = True, color="g")
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c57af50>
```



```
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_int

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]
```

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, 668
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,
79.49208333333331,
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 [ ]: #Linear OLS regression  
nuclear1 = nuclear  
  
nuclear1 = sm.add_constant(nuclear1)  
modelnuclearreg = sm.OLS(io2, 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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.664			
Time:	06:04:42	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.

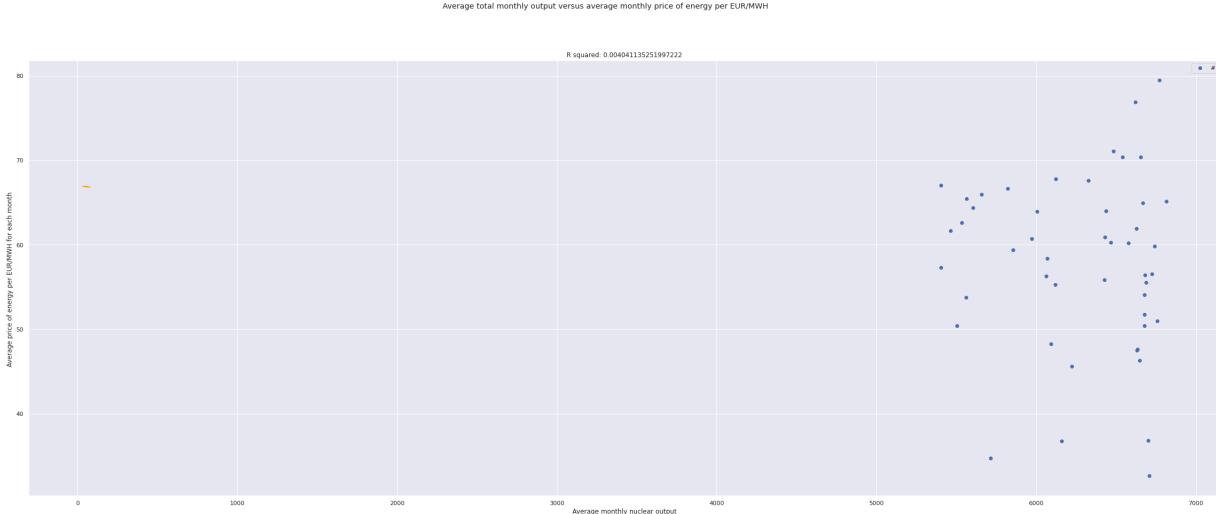
[2] The condition number is large, 8.72e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In []:

```
#slope and intercept for OLS linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(nuclear,io2)
print("slope: %f    intercept: %f" % (slope, intercept))

# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of energy")
plt.plot(nuclear,io2, "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 total monthly output versus average monthly price of energy")
plt.legend("#")
plt.ylabel('Average price of energy per EUR/MWh for each month ')
plt.xlabel('Average monthly nuclear output')
plt.show()
```

There **is** a very weak yet positive correlation between the output **and** their average price.



```
In [ ]: #Linear OLS regression residuals
influenceNuclearreg = modelnuclearreg.get_influence()

standardized_residualsNuclear = influenceNuclearreg.resid_studentized_interresiduals

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
```

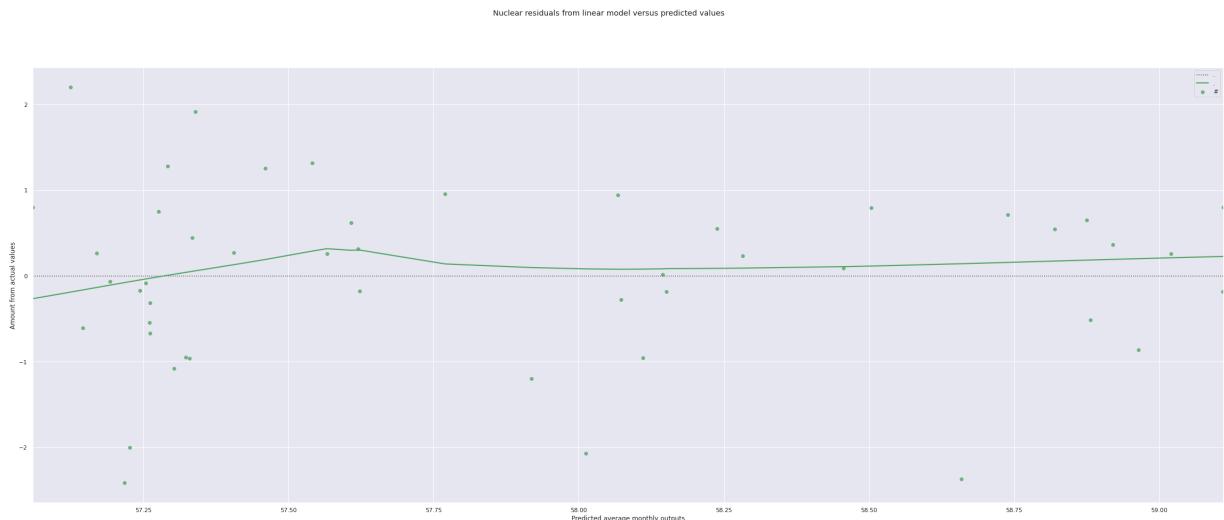
[57.27620225 57.25418559 57.24431984 58.14475009 59.10999271 58.73883375
57.54113449 57.60764599 57.56648576 58.45590145 58.28291414 57.3342319
57.9185622 58.01239833 57.22692211 57.21745112 58.65931232 57.30308989
57.33002946 57.32359684 57.26198473 57.40592371 58.92041739 57.77018995
57.12517741 57.16970727 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)

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 0x7f81c5857f90>
```



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 [ ]:
```

This is the quadratic model used for the average monthly nuclear output versus the average monthly prices of energy per EUR/MWH.

```
#Quadratic OLS regression
from sklearn.preprocessing import PolynomialFeatures
polynomial_features = PolynomialFeatures(degree=2)

modelNuclearquad = np.poly1d(np.polyfit(nuclear,io2,2))
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, io2)
lin2 = LinearRegression()
lin2.fit(X_poly, io2)
```

```

Nuclear_Quad = sm.OLS(io2, nuclear2).fit()

# OLS Predicted Quadratic values
Nuclear_ypred = Nuclear_Quad.predict(nuclear2)

#OLS Quadratic Summary Table
Nuclear_Quad.summary()

```

2
2.973e-06 x² - 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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.872
Time:	06:04:43	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:

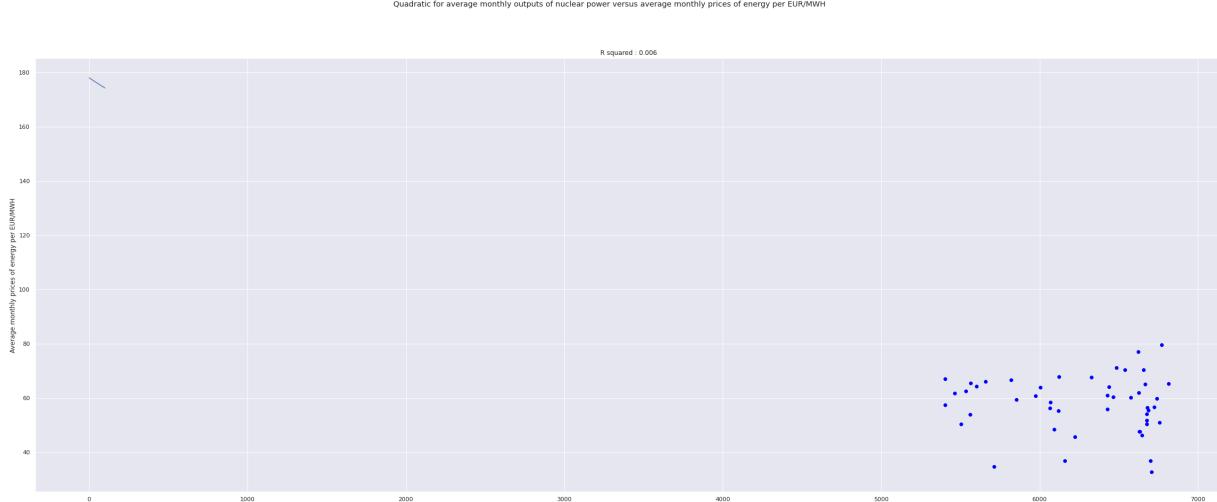
- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 1.11e-33. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

```
In [ ]: #Quadratic Scatterplots
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelNuclearquad(polyline))
plt.scatter(nuclear,io2, color = 'blue')
plt.title("R squared : 0.006")
plt.suptitle('Quadratic for average monthly outputs of nuclear power versus
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_i

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 linear 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
```

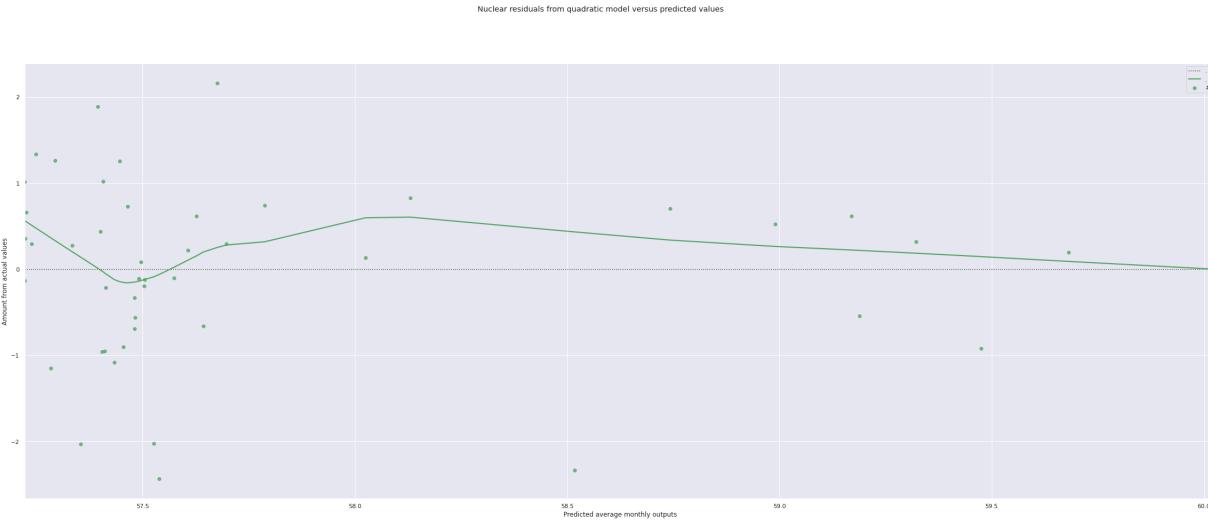
```
[ 0.72684891 -0.1076632 -0.19284055  0.08449366 -0.28717678  0.70872737
 1.33790549  0.65696554  0.29218058  0.13533639  0.29789044  0.43574267
 -1.14954956 -2.02774465 -2.01972351 -2.43309034 -2.32930507 -1.0824437
 -0.9585776 -0.9497852 -0.68781518  0.27525474  0.32525038  1.01324839
 2.16064282  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, lowe
```

```
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 0x7f81c6ceb310>
```

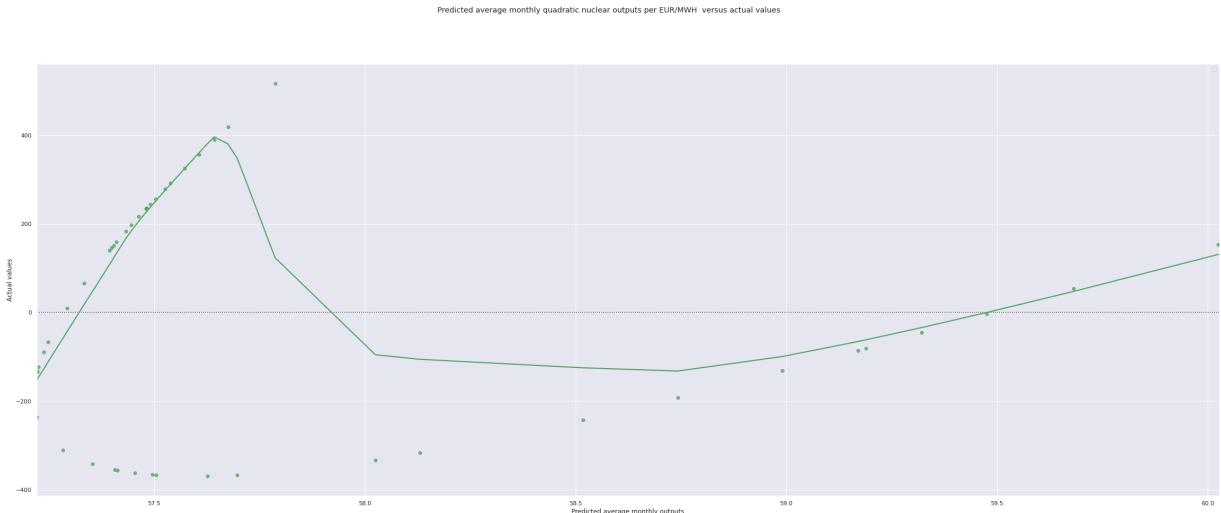


As one can observe this residual plot, one may notice that the lowess line has a slight hump and that the residuals are clustered on the left side; indicating heteroskedasticity and bias. This indicates that the model does not fit the data. The green dots represent the respective observations while the 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 outputs per EUR/MWh")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Nuclear_ypred, y = nuclear, lowess = True, color="g")
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c7085690>
```



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
dfnuclear = ({"Price": [64.9490188172043, 56.38385416666667, 55.5224629878869,
"Nuclear" : [6665.969986357435, 6681.1235119047615, 6687.913862718708, 6068
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)
#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")
    else:
        print("weak evidence against null hypothesis, indicating it is non-stationary")

adfuller_test(df_nuclear["Ratio"])

test_result=adfuller(df_nuclear["Ratio"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_nuclear["Ratio"])
plt.suptitle("Autocorrelations of Nuclear Ratio")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function
plot_pacf(df_nuclear["Ratio"])
plt.suptitle("Partial Autocorrelations of Nuclear Ratio")
plt.ylabel('Partial Autocorrelations')
plt.xlabel('Lags')

plt.show

df_nuclear['First Difference Nuclear Ratio'] = df_nuclear["Ratio"]- df_nuclear["Ratio"].shift(1)
df_nuclear['Seasonal Difference Nuclear Ratio']=df_nuclear["Ratio"]- df_nuclear["Ratio"].shift(12)
df_nuclear.head()

nuclear_Ratio_Autocorrelations = sm.tsa.acf(df_nuclear["Ratio"],fft=False) # Compute Autocorrelation Function
print(nuclear_Ratio_Autocorrelations)

```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Nuclear': [6665.969986357435, 6681.1235119047615, 6687.91
3862718708, 6068.16991643454, 5403.817204301075, 5659.276773296245, 6483.623
655913979, 6437.845430107527, 6466.175, 5854.012096774193, 5973.075, 6626.02
9609690444, 6223.8494623655915, 6159.264367816092, 6699.888290713325, 6706.4
06944444445, 5714.009408602151, 6647.463888888885, 6628.9220430107525, 663
3.3494623655915, 6675.755555555555, 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, 6655.3212
3655914, 6621.998611111111, 6539.120805369127, 5821.1518817204305, 5403.4972
2222222222], '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

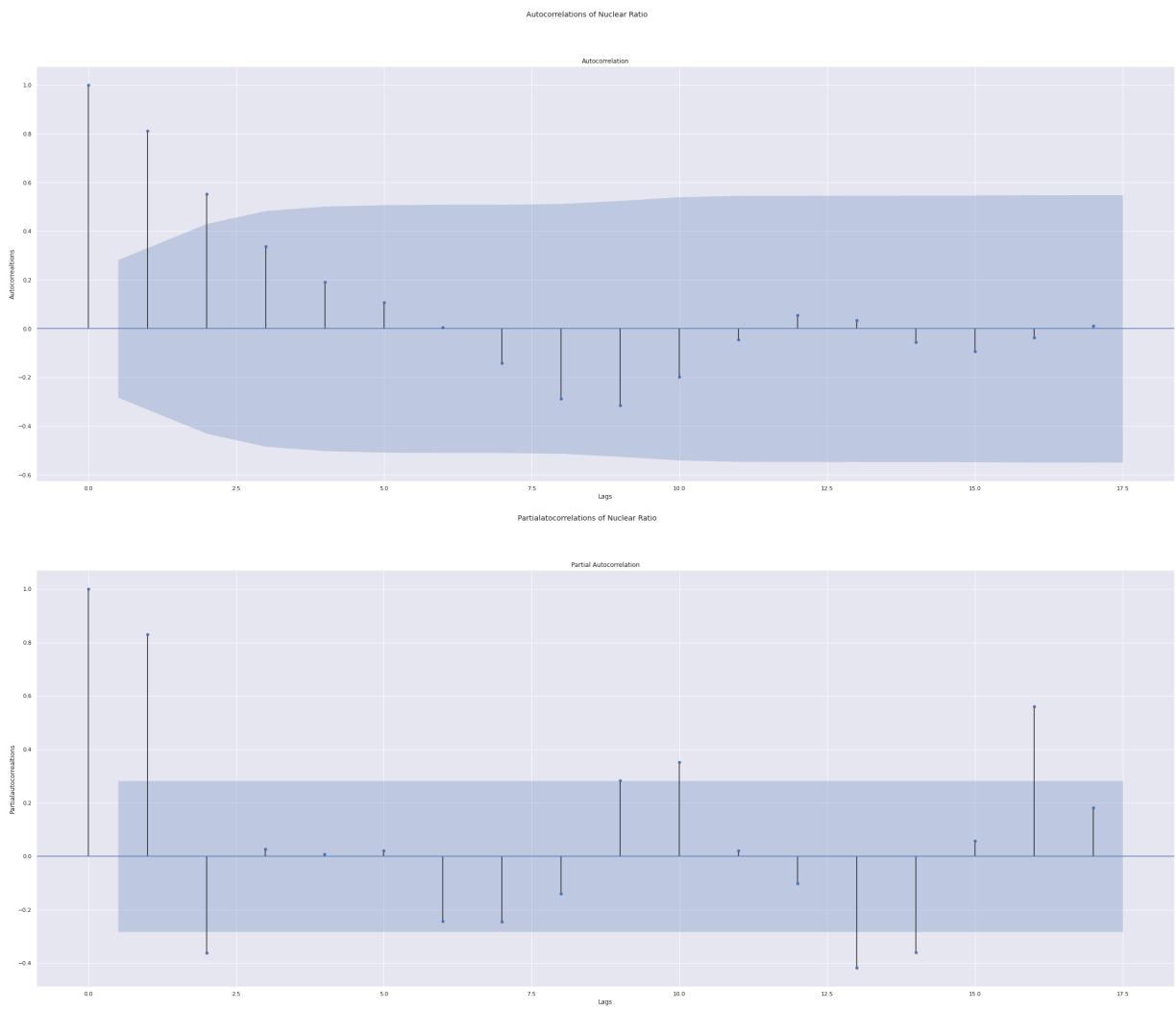
```

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
[102.63388281689778, 118.49355831823476, 120.45420002671308, 103.98877970152
685, 94.3172343770747, 85.77923627046935, 91.22607682648228, 100.59437701408
982, 107.31387199496582, 98.54116929029787, 98.35979863363437, 107.041052291
92045, 136.55164156780614, 167.589529876522, 181.97313328298446, 205.6002783
0008724, 164.7098182979096, 143.678251667955, 139.55033034817424, 139.349234
5580292, 132.44075903166618, 109.27916930870813, 88.43406639744545, 93.58618
021036858, 85.16466524444257, 112.62559996020988, 132.5738976103446, 129.092
27137597384, 103.42141579325329, 107.78619520307156, 110.71705890251806, 12
3.43403396826045, 115.15739057905957, 93.91829892025801, 85.05519096023419,
104.60483304468198, 118.97813552223086, 105.61235177921166, 126.174828549310
13, 109.20822640101433, 88.67218166973136, 87.07676514860786, 90.30989575192
407, 94.58429917431938, 86.09609464822834, 92.93511904364472, 86.82764731945
06, 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 * n * log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to a n 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 and the partial 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.

```
In [ ]: df_nuclear.describe(include = 'all') # Description table of Dataframes
```

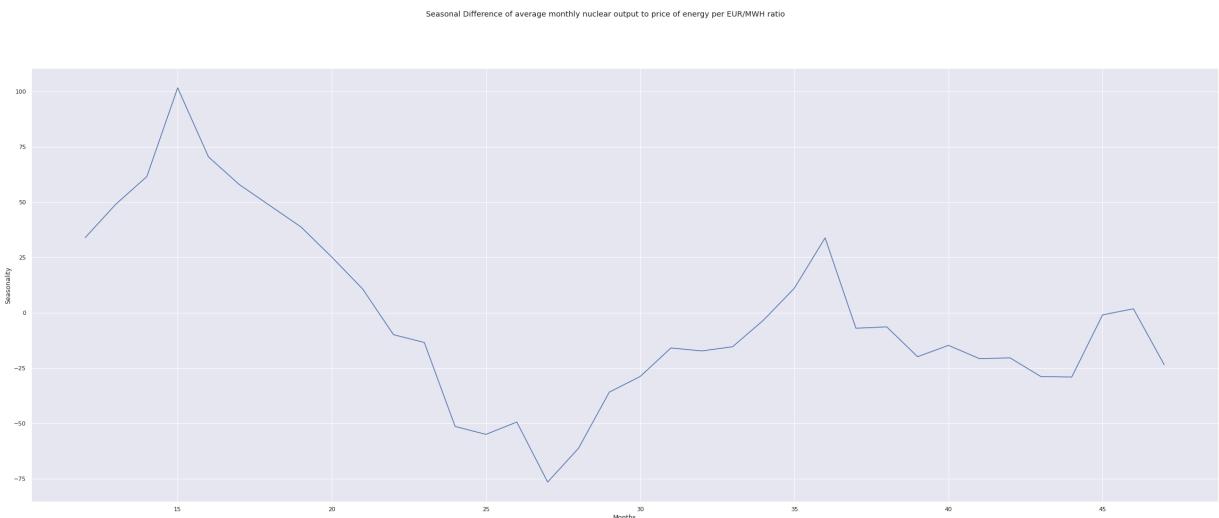
```
Out[ ]:
```

	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")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_nuclear['Seasonal Difference Nuclear Ratio'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c6e8db10>
```



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 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)

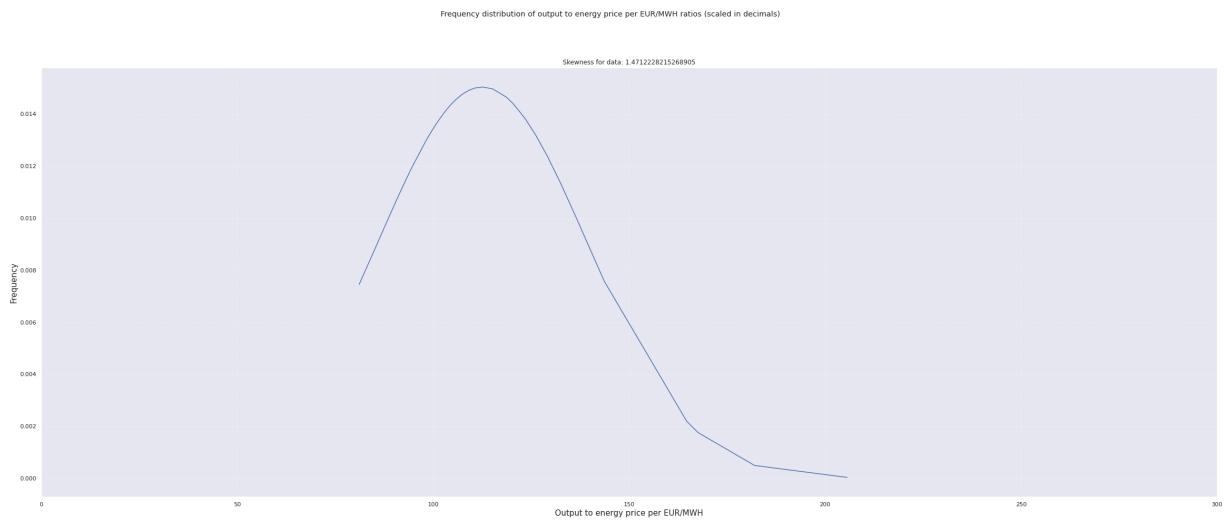
plt.plot(df_nuclear["Ratio"].sort_values(), nuclearResultspdf)
plt.xlim([0,300])
plt.xlabel("Output to energy price per EUR/MWh", size=15)
plt.title(f'Skewness for data: {skew(df_nuclear["Ratio"])}')
plt.suptitle("Frequency distribution of output to energy price per EUR/MWh ratios (scaled in decimals)")
plt.ylabel("Frequency", size=15)
plt.grid(True, alpha=0.3, linestyle="--")
plt.show()

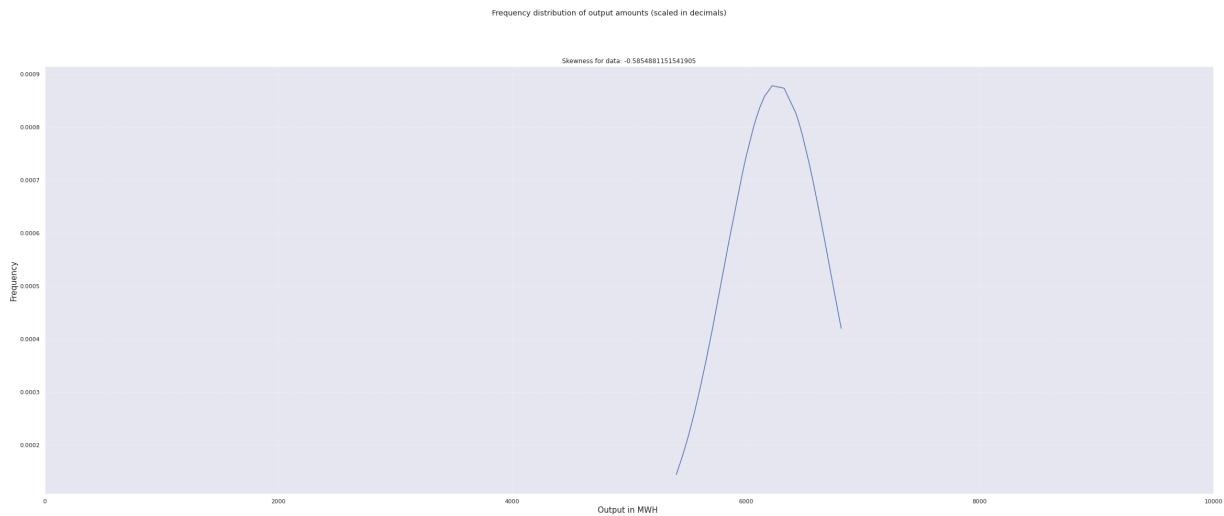
#Bell Curves
nuclearResults_mean = np.mean(df_nuclear["Nuclear"])
nuclearResults_std = np.std(df_nuclear["Nuclear"])

nuclearResultspdf = stats.norm.pdf(df_nuclear["Nuclear"].sort_values(), nuclearResults_mean, nuclearResults_std)

plt.plot(df_nuclear["Nuclear"].sort_values(), nuclearResultspdf)
plt.xlim([0,10000])
plt.xlabel("Output in MWh ", size=15)
plt.title(f'Skewness for data: {skew(df_nuclear["Nuclear"])}')
plt.suptitle("Frequency distribution of output amounts (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 and left, respectively. Hence they have a asymmetrical distribution. The blue line in this plot represent the observation values and their likelihood of occurring.

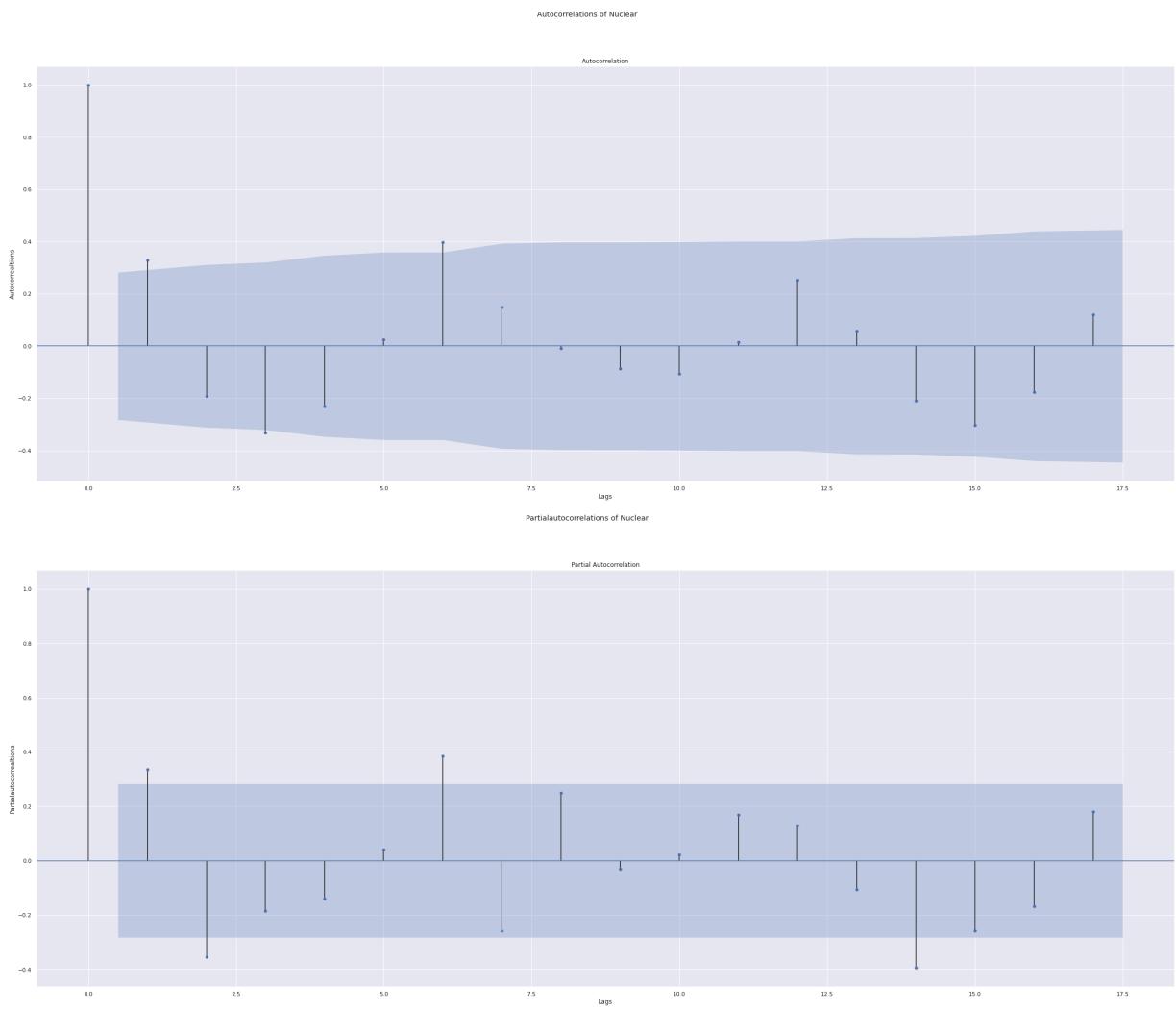
If the skewness is between -0.5 and 0.5, the data is fairly symmetrical. If the skewness is between -1 and - 0.5 or between 0.5 and 1, the data is moderately skewed. If the skewness is less than -1 or greater than 1, the data is highly skewed.

```
In [ ]: from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_nuclear["Nuclear"])
plt.suptitle("Autocorrelations of Nuclear")
plt.ylabel('Autocorrealtions')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Plot
plot_pacf(df_nuclear["Nuclear"])
plt.suptitle("Partialautocorrelations of Nuclear")
plt.ylabel('Partialautocorrealtions')
plt.xlabel('Lags')

plt.show
```

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



```
In [ ]: Nuclear_Autocorrelations = sm.tsa.acf(df_nuclear["Nuclear"], fft=False) #Autocorrelation
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 * n * p.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to a 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 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")
    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)
df_nuclear['Seasonal Difference Nuclear']=df_nuclear["Nuclear"]- df_nuclear["Nuclear"].shift(12)
df_nuclear.head()
```

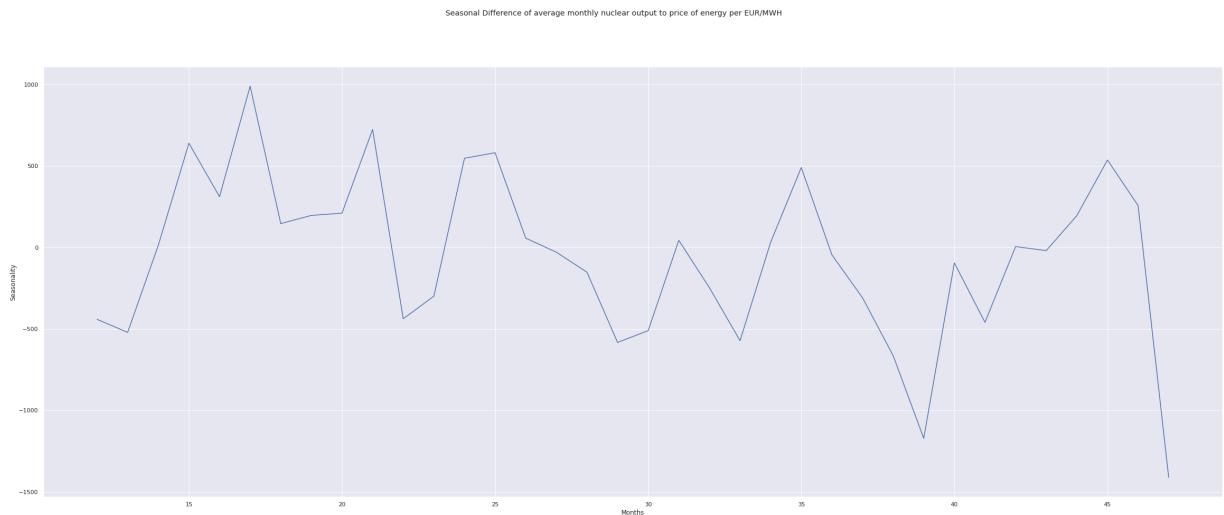
```
Out[ ]:
```

	Price	Nuclear	Dates	Ratio	First Difference Nuclear Ratio	Seasonal Difference Nuclear Ratio	Fi
0	64.949019	6665.969986	2015-01	102.633883	NaN	NaN	NaN
1	56.383854	6681.123512	2015-02	118.493558	15.859676	NaN	15.1535
2	55.522463	6687.913863	2015-03	120.454200	1.960642	NaN	6.7903
3	58.354083	6068.169916	2015-04	103.988780	-16.465420	NaN	-619.7439
4	57.294059	5403.817204	2015-05	94.317234	-9.671545	NaN	-664.3527

```
In [ ]: plt.suptitle("Seasonal Difference of average monthly nuclear output to price")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_nuclear['Seasonal Difference Nuclear'].plot() # Seasonality Plot
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c5dc85d0>
```



```
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 table of Dataframes
```

Out[]:

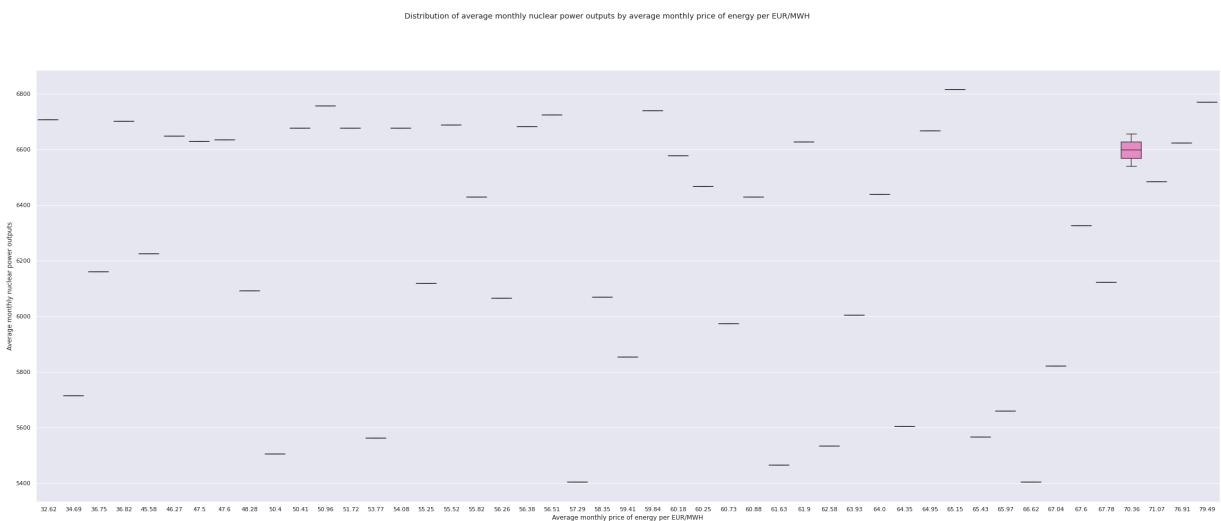
	Price	Nuclear	Dates	Ratio	First Difference Nuclear Ratio	Seasonal Difference Nuclear Ratio	
count	48.000000	48.000000	48	48.000000	47.000000	36.000000	
unique	NaN	NaN	48	NaN	NaN	NaN	
top	NaN	NaN	2015-01	NaN	NaN	NaN	
freq	NaN	NaN	1	NaN	NaN	NaN	
mean	57.859848	6264.260822	NaN	112.554520	-0.458062	-1.698965	
std	10.320573	457.169177	NaN	26.825833	15.863282	40.147662	5
min	32.618667	5403.497222	NaN	81.104957	-40.890460	-76.508007	-11
25%	51.528411	5943.309274	NaN	93.423415	-8.817905	-24.833225	-3
50%	59.622281	6433.609471	NaN	106.326702	-0.201096	-11.690302	
75%	64.999583	6668.416379	NaN	124.119233	9.024777	27.298533	1
max	79.492083	6815.138441	NaN	205.600278	31.037888	101.611499	12

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/MWH')
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, 6687.9138])
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 approximently 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 approximently 79.49 EUR/MWH, the output was slighty below 6600 units.

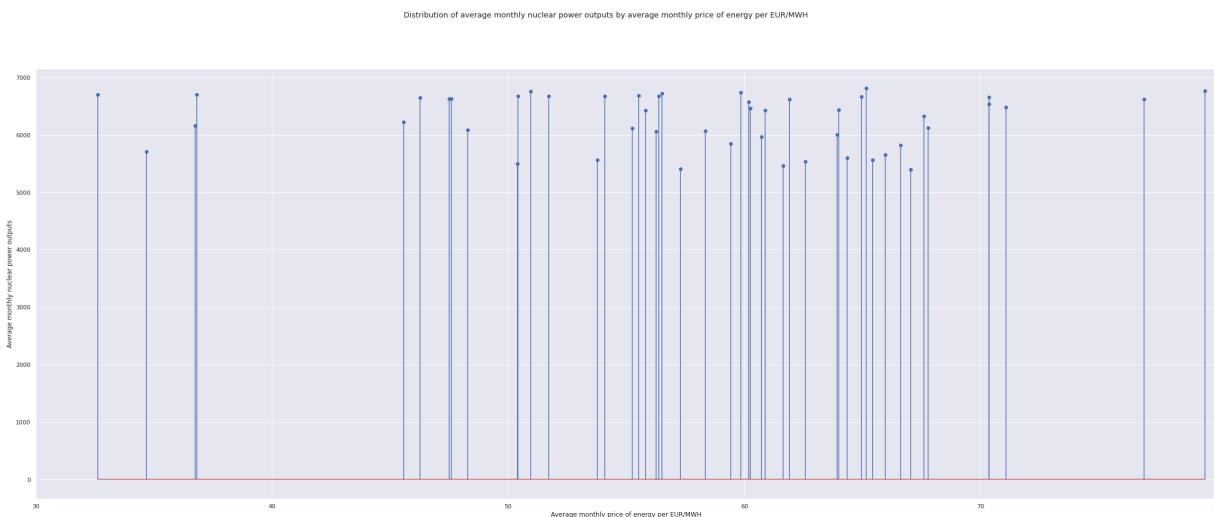
```
In [ ]: plt.suptitle('Distribution of average monthly nuclear power outputs by average monthly price of energy per EUR/MWH')
plt.ylabel('Average monthly nuclear power outputs')
plt.xlabel('Average monthly price of energy per EUR/MWH')

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 slighty below 6600 units. Each blue dot at the end of the blue line represents an observation.

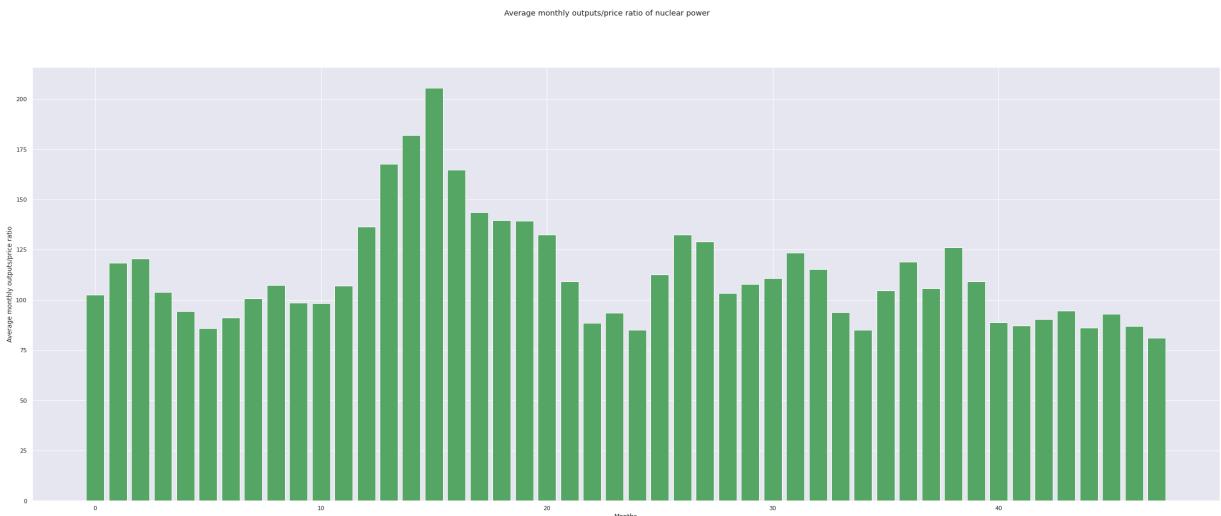
```
In [ ]: #Histograms
pdToListNuclear_Dict = {key: i for i, key in enumerate(pdToListNuclear)}

def Hist_pdToListNuclear(pdToListNuclear_Dict):
    for k, v in pdToListNuclear_Dict.items(): print(f"{v}:{k}") #Histograms
print(pdToListNuclear_Dict)

plt.bar(list(pdToListNuclear_Dict.values()),pdToListNuclear_Dict.keys(), col
print(dicDates)
plt.suptitle("Average monthly outputs/price ratio of nuclear power")
plt.ylabel('Average monthly outputs/price ratio')
plt.xlabel('Months')

plt.show()

{102.63388281689778: 0, 118.49355831823476: 1, 120.45420002671308: 2, 103.98
877970152685: 3, 94.3172343770747: 4, 85.77923627046935: 5, 91.2260768264822
8: 6, 100.59437701408982: 7, 107.31387199496582: 8, 98.54116929029787: 9, 9
8.35979863363437: 10, 107.04105229192045: 11, 136.55164156780614: 12, 167.58
9529876522: 13, 181.97313328298446: 14, 205.60027830008724: 15, 164.70981829
79096: 16, 143.678251667955: 17, 139.55033034817424: 18, 139.3492345580292:
19, 132.44075903166618: 20, 109.27916930870813: 21, 88.43406639744545: 22, 9
3.58618021036858: 23, 85.16466524444257: 24, 112.62559996020988: 25, 132.573
8976103446: 26, 129.09227137597384: 27, 103.42141579325329: 28, 107.78619520
307156: 29, 110.71705890251806: 30, 123.43403396826045: 31, 115.157390579059
57: 32, 93.91829892025801: 33, 85.05519096023419: 34, 104.60483304468198: 3
5, 118.97813552223086: 36, 105.61235177921166: 37, 126.17482854931013: 38, 1
09.20822640101433: 39, 88.67218166973136: 40, 87.07676514860786: 41, 90.3098
9575192407: 42, 94.58429917431938: 43, 86.09609464822834: 44, 92.93511904364
472: 45, 86.8276473194506: 46, 81.1049568960575: 47}
```



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.

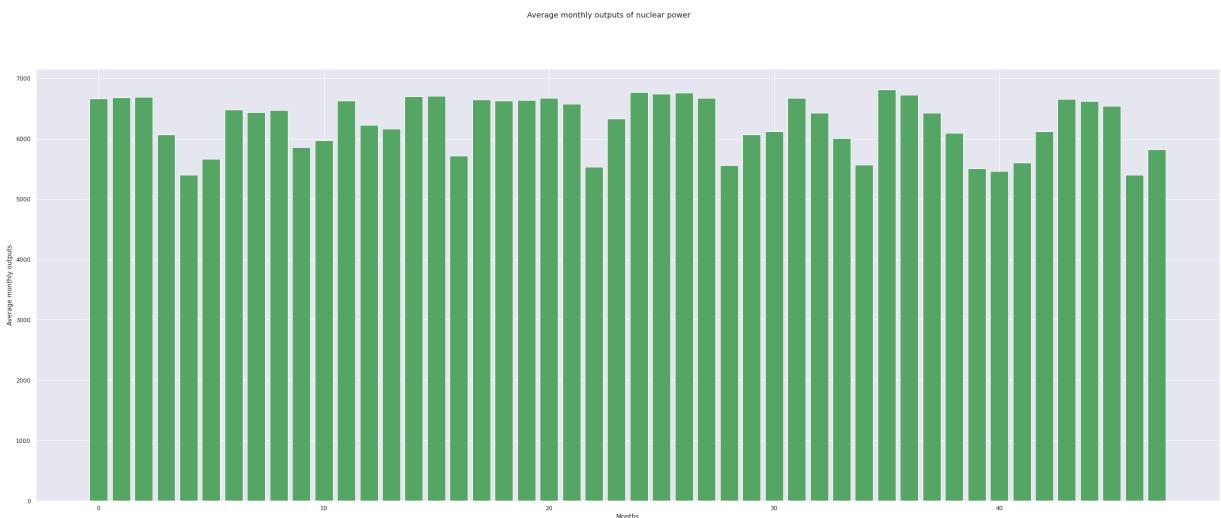
```
In [ ]: Nuclear_Dict = {key: i for i, key in enumerate(nuclear)}

def Hist_Nuclear(Nuclear_Dict):
    for k, v in Nuclear_Dict.items(): print(f"{v}:{k}") #Histograms
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, 6068.169
91643454: 3, 5403.817204301075: 4, 5659.276773296245: 5, 6483.623655913979:
6, 6437.845430107527: 7, 6466.175: 8, 5854.012096774193: 9, 5973.075: 10, 66
26.029609690444: 11, 6223.8494623655915: 12, 6159.264367816092: 13, 6699.888
290713325: 14, 6706.406944444445: 15, 5714.009408602151: 16, 6647.4638888888
885: 17, 6628.9220430107525: 18, 6633.3494623655915: 19, 6675.7555555555555:
20, 6576.6859060402685: 21, 5534.29722222222: 22, 6325.970430107527: 23, 67
69.9166666666667: 24, 6739.267857142857: 25, 6755.951547779273: 26, 6676.3833
33333333: 27, 5561.240591397849: 28, 6063.85972222222: 29, 6117.40322580645
2: 30, 6675.846774193548: 31, 6427.68888888889: 32, 6003.754362416107: 33,
5565.216666666666: 34, 6815.138440860215: 35, 6723.689516129032: 36, 6429.37
35119047615: 37, 6091.685060565276: 38, 5504.175: 39, 5465.200268817204: 40,
5603.227777777778: 41, 6121.515477792732: 42, 6655.32123655914: 43, 6621.998
61111111: 44, 6539.120805369127: 45, 5403.49722222222: 46, 5821.1518817204
305: 47}
```



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.

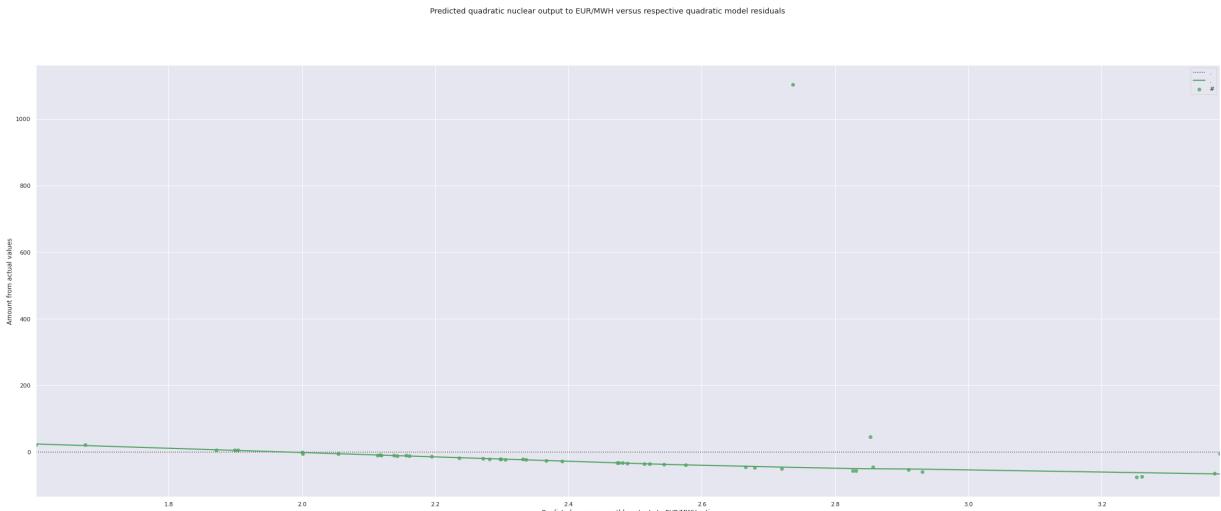
```
In [ ]: #OLS predicted quadratic average monthly ratios versus residuals
NuclearQuadRatioPredict = Nuclear_ypred/ypred

sns.residplot(x = NuclearQuadRatioPredict , y = standardized_residualsNuclear

plt.suptitle("Predicted quadratic nuclear output to EUR/MWH versus respective")
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

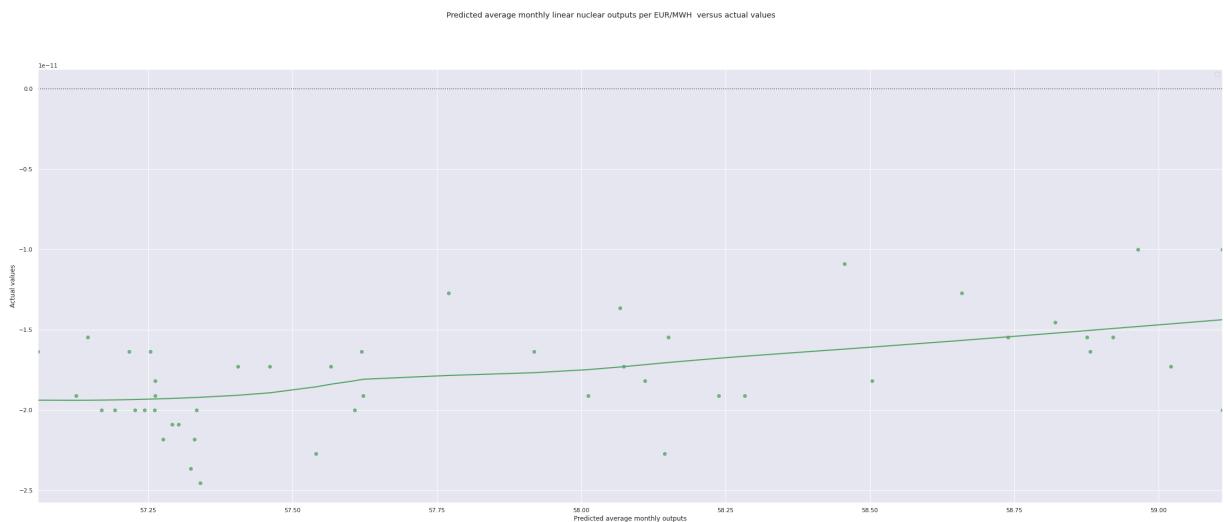
Out[]: <matplotlib.legend.Legend at 0x7f81c54d78d0>



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 outputs per EUR/MWH")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsnuclear, y = nuclear, lowess = True, color="g")
#Predicted OLS average monthly linear values versus actual values
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c54da850>

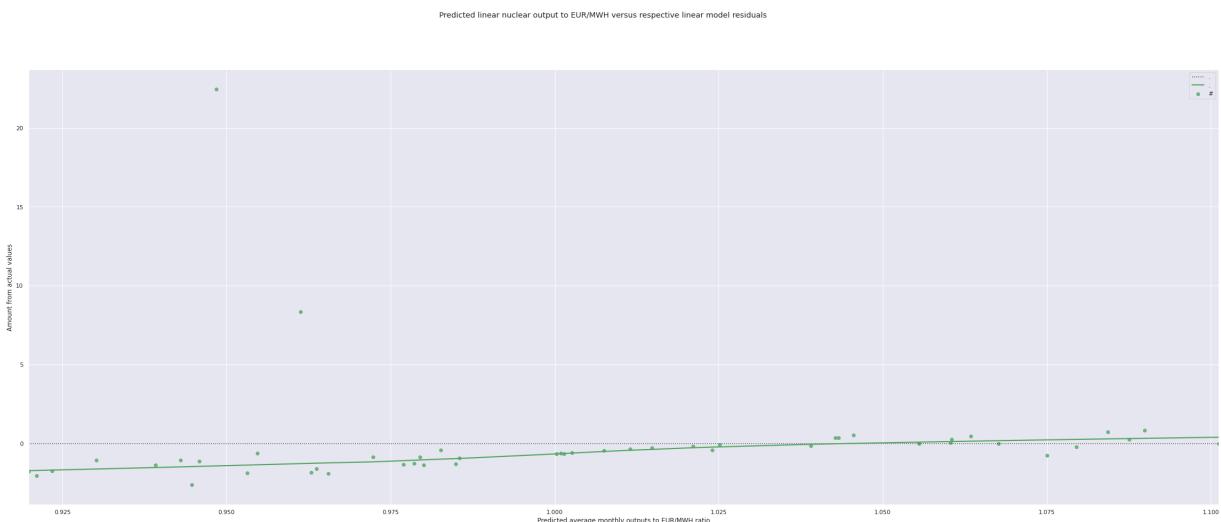


```
In [ ]: #Predicted OLS linear average monthly ratios versus residuals
NuclearRegRatioPredict = predictionsnuclear/predictions

sns.residplot(x = NuclearRegRatioPredict, y = standardized_residualsNuclear/
plt.suptitle("Predicted linear nuclear output to EUR/MWH versus respective l
plt.xlabel("Predicted average monthly outputs to EUR/MWH ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#)
```

Out[]: <matplotlib.legend.Legend at 0x7f81c54b9110>



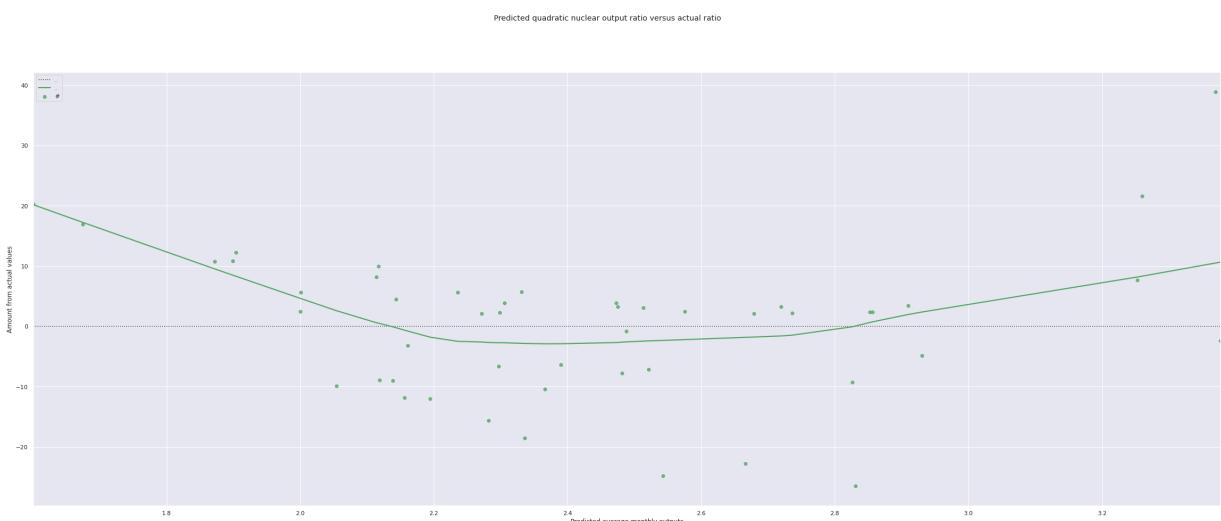
With the exception of few outliers, the predictions seem to be nearly the same as the residuals. This indicates homoscedasticity, constant variance, and a lack of bias. The green dots represent the respective observations while the dotted horizontal line represents the regression model. The green line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

```
In [ ]: #OLS predicted quadratic average monthly ratios versus actual ratios

sns.residplot(x = NuclearQuadRatioPredict, y = pdToListNuclear, lowess = True
plt.suptitle("Predicted quadratic nuclear output ratio versus actual ratio")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

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

```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c53bcd0>
```



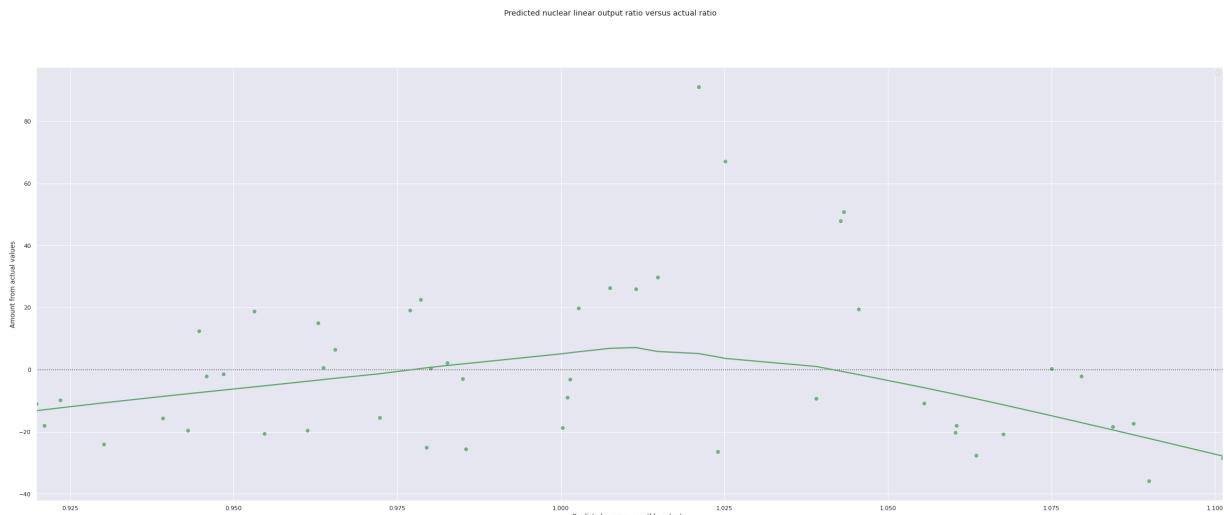
As one can observe, there is an increasing trend along the lowess line in the variance of the predicted values versus the actual values. This indicates that there is heteroskedasticity and no bias. The green line represents the expected accuracy from the observed regression value versus the actual value of the observation itself.

```
In [ ]: plt.suptitle("Predicted nuclear linear output ratio versus actual ratio")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")

plt.legend(..#)

sns.residplot(x = NuclearRegRatioPredict, y = pdToListNuclear, lowess = True
# OLS predicted linear average monthly ratios versus actual ratios
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c4c0a750>



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. 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, 6685.522462987886975,
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: -0.001453    intercept: 66.961236
```

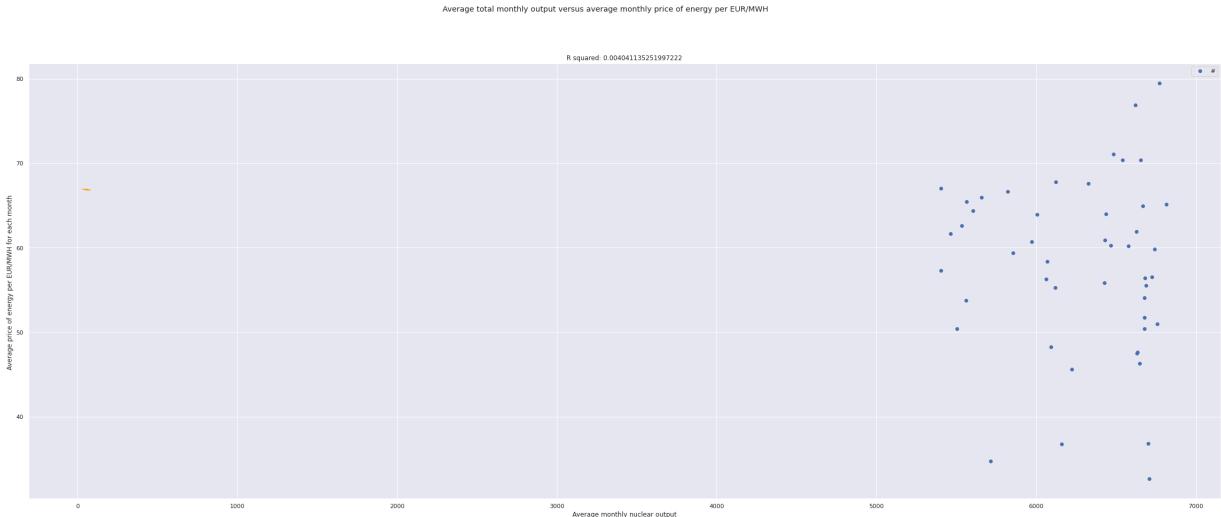
```
In [ ]: #slope and intercept for OLS linear regression  
slope, intercept, r_value, p_value, std_err = stats.linregress(nuclear,io2)  
print("slope: %f    intercept: %f" % (slope, intercept))  
  
# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of e  
plt.plot(nuclear,io2, "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 total monthly output versus average monthly price of energy per EUR/MWH")
plt.legend("#")
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly nuclear output')
plt.show()

```

There **is** a very weak yet positive correlation between the output **and** their average price.



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, 55.5224625],
                     "Other_Renewable" : [70.66030013642565, 70.51488095238095, 66.6325706594885]
                    })
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, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Other_Renewable': [70.66030013642565, 70.51488095238095,
66.63257065948856, 69.70055710306407, 70.56586021505376, 65.51182197496523,
68.3252688172043, 68.1760752688172, 67.96666666666667, 70.66935483870968, 7
0.3125, 71.62314939434724, 77.95430107526882, 74.45402298850574, 73.40242261
103634, 78.275, 79.28225806451613, 83.48333333333333, 78.6271870794078, 78.4
3010752688173, 83.79166666666667, 83.78120805369127, 85.95694444444445, 90.0
9005376344086, 99.20430107526882, 94.90922619047619, 95.6850605652759, 95.97
083333333333, 96.74596774193549, 92.10555555555555, 93.39650537634408, 92.24
731182795699, 94.87222222222222, 95.6268456375839, 94.90138888888889, 91.815
86021505376, 96.98252688172043, 96.75744047619048, 97.64199192462988, 97.151
38888888889, 100.59005376344086, 101.10972222222222, 96.71736204576042, 96.6
1559139784946, 100.19444444444444, 98.61744966442953, 95.88172043010752, 96.
4625], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '201
5-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '20
16-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2
016-08', '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	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, 1.1944418131789
818, 1.2316435818046778, 0.9929809551311909, 0.9613522550191385, 1.065283392
306891, 1.1279877464793602, 1.1895842959796394, 1.1578497409504178, 1.157045
4301049409, 1.7103221797805228, 2.025844965392141, 1.9936554541672733, 2.399
6995585349907, 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, 1.45041176971292
97, 1.4092718456115882, 1.7161411452658597, 1.5893898249815874, 2.0224226084
925134, 1.9275787693797464, 1.6320608729330732, 1.571292099011756, 1.4268582
53555079, 1.3730844352755214, 1.3026807884920595, 1.40156829903176, 1.430160
967287293, 1.4478746972258918]

```

This is the logarithmic model for the average monthly other renewable outputs versus the average monthly prices of energy per EUR/MWH.

```

In [ ]: #Logarithmic OLS regressions
Logpricevalues = ((np.log(io2)))
Logotherrenewablevalues = ((np.log(other_renewable)))
Log = np.polyfit(np.log(io2), other_renewable1, 1)
lin2 = LinearRegression()
lin2.fit(np.log(other_renewable1), io2)
Otherrenewable_Log = sm.OLS(io2, 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), io2, 1)

```

```

print(Log)

y = 17.05011223 * Logotherrenewablevalues - 17.84401394

#Logarithmic OLS regression scatterplot
plt.suptitle("Logarithmic average monthly outputs versus average monthly pri
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, io2, "o")
plt.plot(Logotherrenewablevalues, y)

plt.xlim([1, 7])

```

[17.05011223 -17.84401394]

Out[]: (1, 7)



The blue dots represent the observations and the orange line is the linear 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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.0706				
Time:	06:05:00	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 []:

```
influenceOtherrenewableLog = Otherrenewable_Log.get_influence()
#Logarithmic OLS regression residuals

standardized_residualsOtherrenewableLog = influenceOtherrenewableLog.resid_s

print(standardized_residualsOtherrenewableLog)

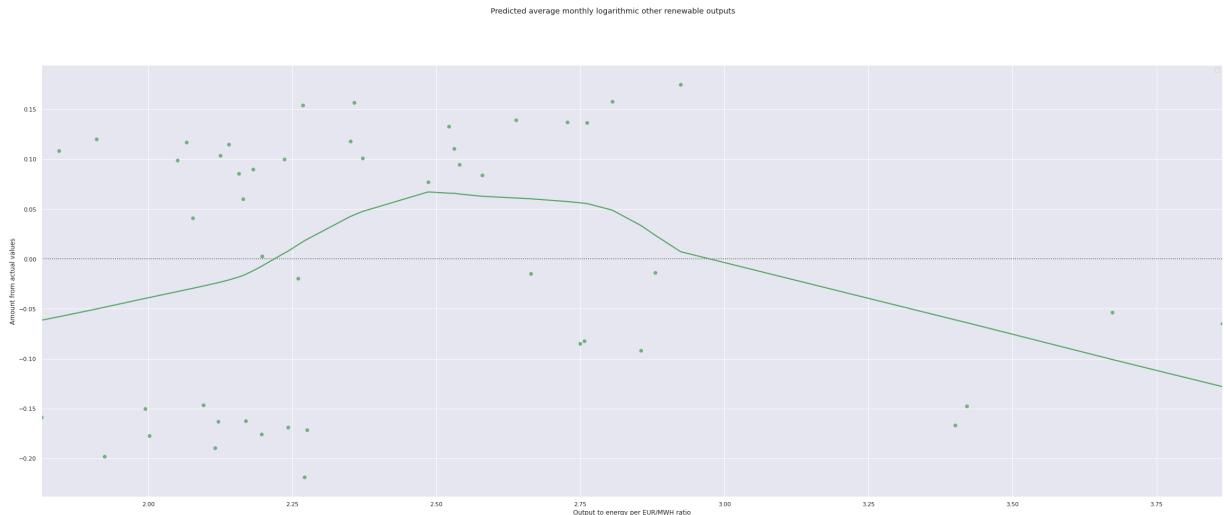
print(Otherrenewable_Logpred) # OLS logarithmic predicted values
```

```
[ 1.07173466  0.20008828  0.20431036  0.42119401  0.29189581  1.3148206
 1.76196653  1.03882106  0.65920866  0.50527028  0.64896446  0.7364449
-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]
[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 [ ]: plt.suptitle("Predicted average monthly logarithmic other renewable outputs")
plt.xlabel("Output to energy per EUR/MWh ratio")
plt.ylabel("Amount from actual values")

plt.legend(..#)
OtherrenewableLogRatioPredict = Otherrenewable_Logpred/predictionLog
sns.residplot(x = OtherrenewableLogRatioPredict, y = pdToList0ther, lowess =
#OLS Logarithmic predicted average monthly ratios versus actual ratios
```

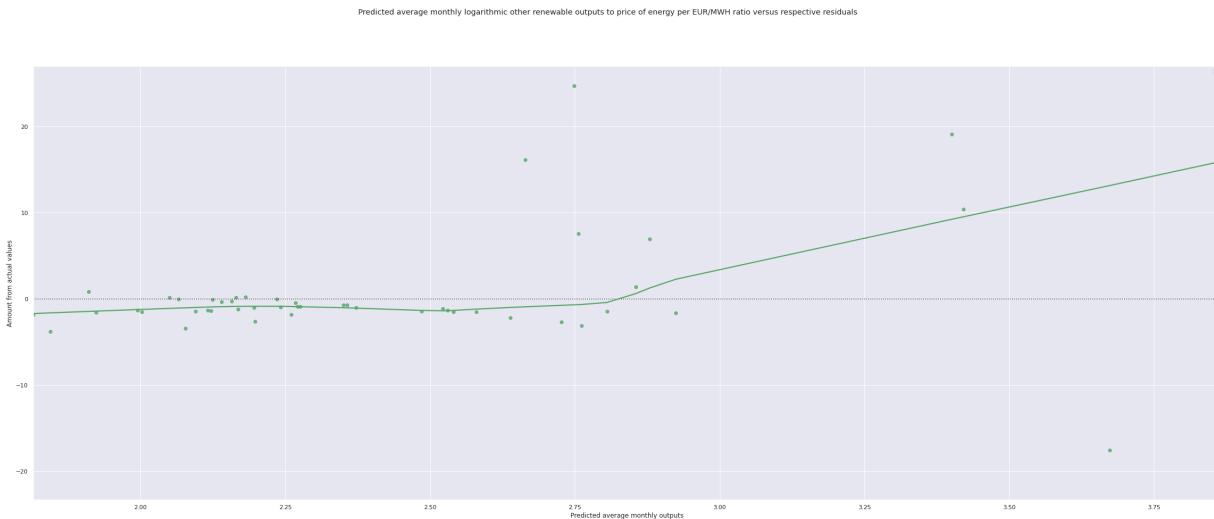
Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c49cb850>



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 [ ]: # OLS Logarithmic average monthly predicted ratios versus residuals
plt.suptitle("Predicted average monthly logarithmic other renewable outputs")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#")
sns.residplot(x = OtherrenewableLogRatioPredict, y = standardized_residuals0
```

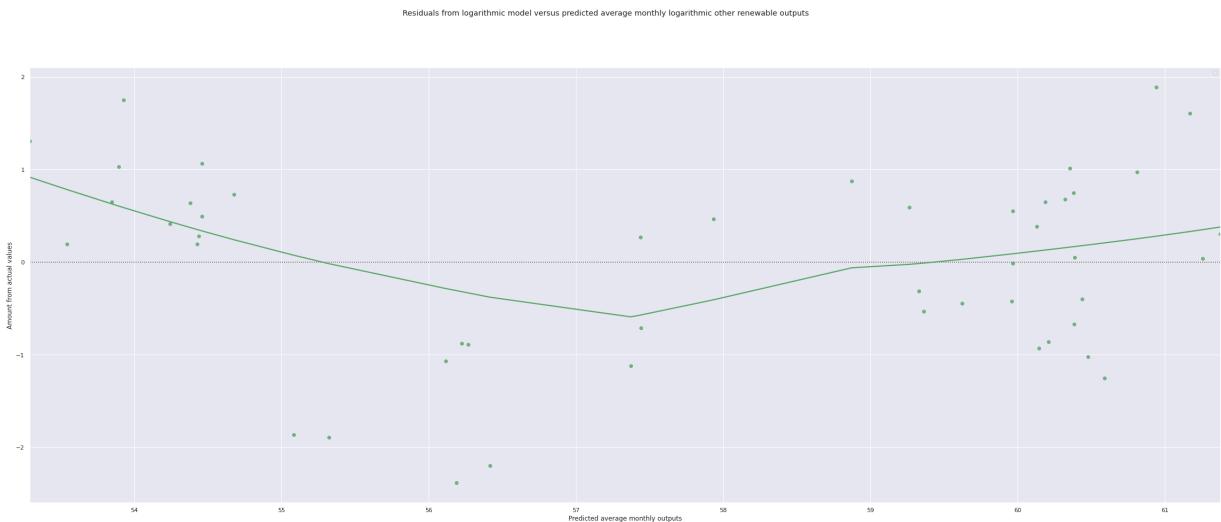
Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c49416d0>



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 mont")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
plt.legend(..#")
sns.residplot(x = Otherrenewable_Logpred, y = standardized_residuals0therre
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c493a7d0>

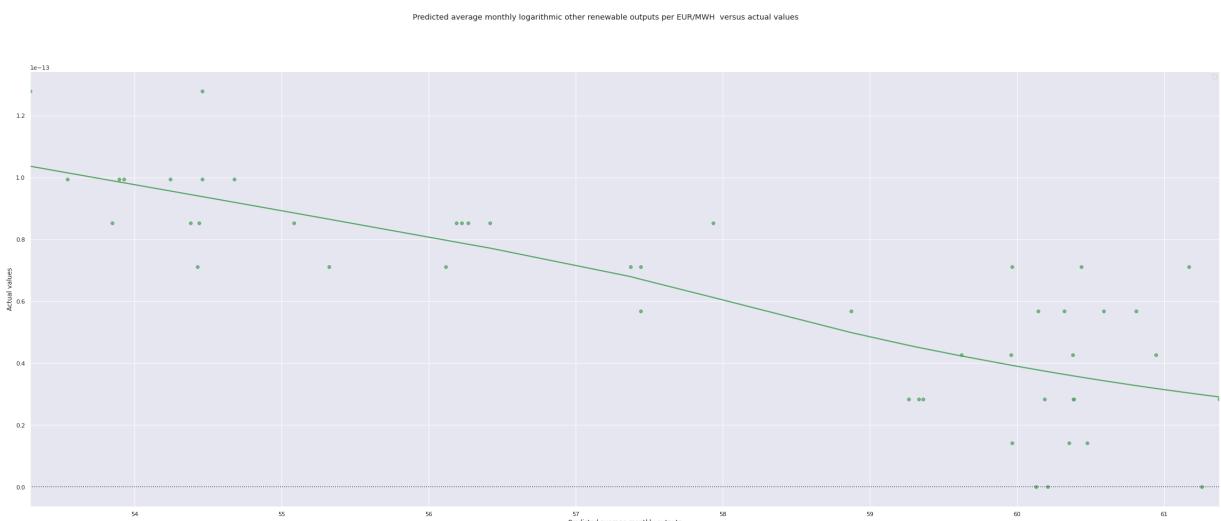


As one can observe this residual plot, one may notice the slight humps in the observations, which form a nonlinear pattern. However, the residuals are spread out. This indicates homoscedasticity, a lack of bias and constant variance. The green dots represent the respective observations while the dotted 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 outputs")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#")
sns.residplot(x = Otherrenewable_Logpred, y = other_renewable, lowess = True)
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c48a73d0>



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.

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  
[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,  
 32.61866666666666,  
 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,  
50.40073611111113,  
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(io2,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:	Sun, 09 Oct 2022	Prob (F-statistic):	0.0706			
Time:	06:05:03	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:

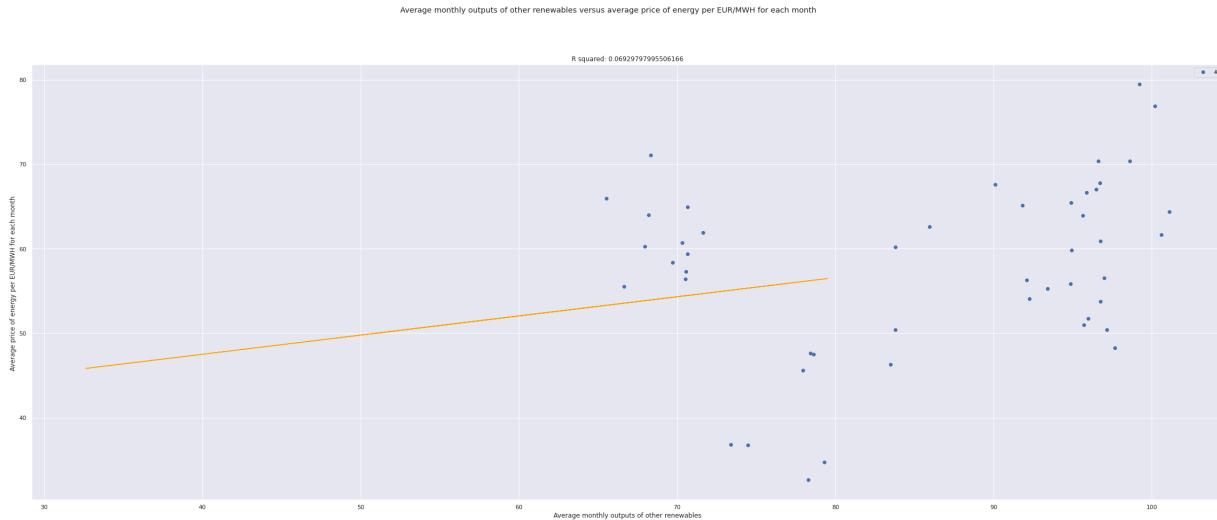
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In []:

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

# OLS linear Scatterplot plt.suptitle("Average monthly outputs to price of energy per MWh")
plt.plot(other_renewable,io2, "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 average price of energy per MWh')
plt.ylabel('Average price of energy per EUR/MWH for each month ')
plt.xlabel('Average monthly outputs of other renewables')
plt.show()
```

There **is** a very weak **and** positive correlation between the output **and** their a



```
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 = modelotherrenewablerereg.get_influence()

standardized_residualsOther = influenceOtherreg.resid_studentized_internal

print(standardized_residualsOther)
```

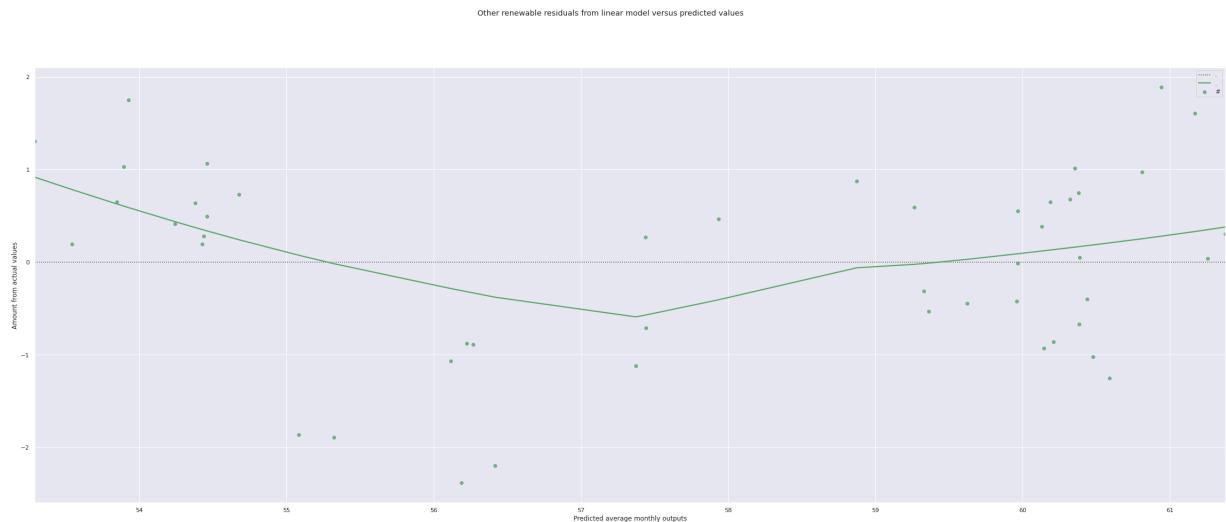
```
[ 1.07173466  0.20008828  0.20431036  0.42119401  0.29189581  1.3148206
 1.76196653  1.03882106  0.65920866  0.50527028  0.64896446  0.7364449
 -1.06284522 -1.88251627 -1.85515545 -2.37656007 -2.18832498 -1.11552308
 -0.88359491 -0.86915788 -0.70670097  0.27552976  0.46669786  0.87724287
 1.88928248 -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

plt.suptitle("Other renewable residuals from linear model versus predicted v
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Amount from actual values")
```

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

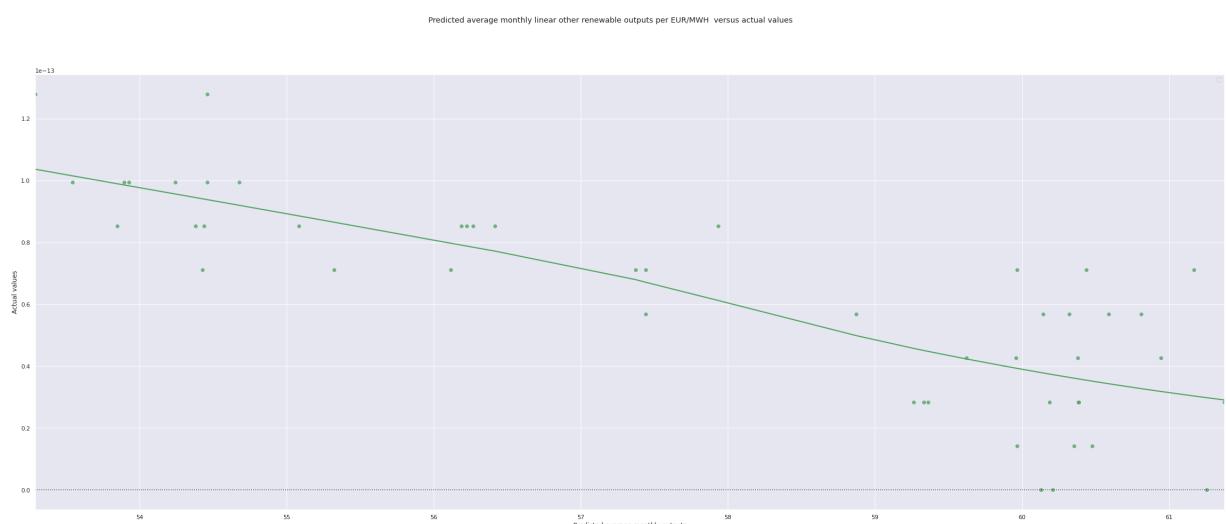
```
Out[ ]: <matplotlib.legend.Legend at 0x7f81c47a8890>
```



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 E")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = predictionsotherrenewable, y = other_renewable, lowess = 1
#Predicted OLS average monthly linear values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c4736490>
```



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.

This is the quadratic model used for the average monthly other renewable output versus the 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,io2,2))
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, io2)
lin2 = LinearRegression()
lin2.fit(X_poly, io2)
Otherrenewable_Quad = sm.OLS(io2, otherrenewable2).fit()

# OLS Predicted Quadratic values
Otherrenewable_ypred = Otherrenewable_Quad.predict(otherrenewable2)

#OLS Quadratic Summary Table
Otherrenewable_Quad.summary()

#Quadratic Scatterplots
polyline = np.linspace(start = 0, stop =100 , num = 100)
plt.plot(polyline, modelOtherrenewablequad(polyline))
plt.scatter(other_renewable , io2, color = 'blue')
plt.title("R squared : 0.350")
plt.suptitle('Quadratic for average monthly prices of energy per EUR/MWH ver')
plt.xlabel('Average monthly outputs of other renewables')
plt.ylabel('Average monthly prices of energy per EUR/MWH')
plt.show()

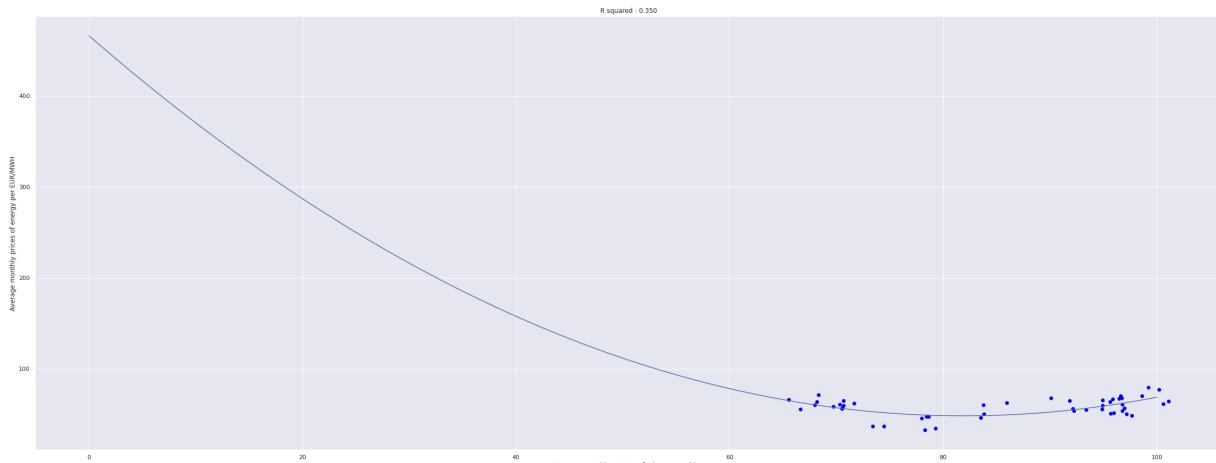
influenceOtherrenewableQuad = Otherrenewable_Quad.get_influence() #Quadratic

standardized_residualsOtherrenewableQuad = influenceOtherrenewableQuad.resid
```

```
print(standardized_residualsOtherrenewableQuad)
```

$$0.06228 x^2 - 10.2 x + 465.9$$

Quadratic for average monthly prices of energy per EUR/MWh versus average monthly outputs of other renewables



```
[ 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 quadratic 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:	Sun, 09 Oct 2022	Prob (F-statistic):	6.08e-05			
Time:	06:05:05	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]
```

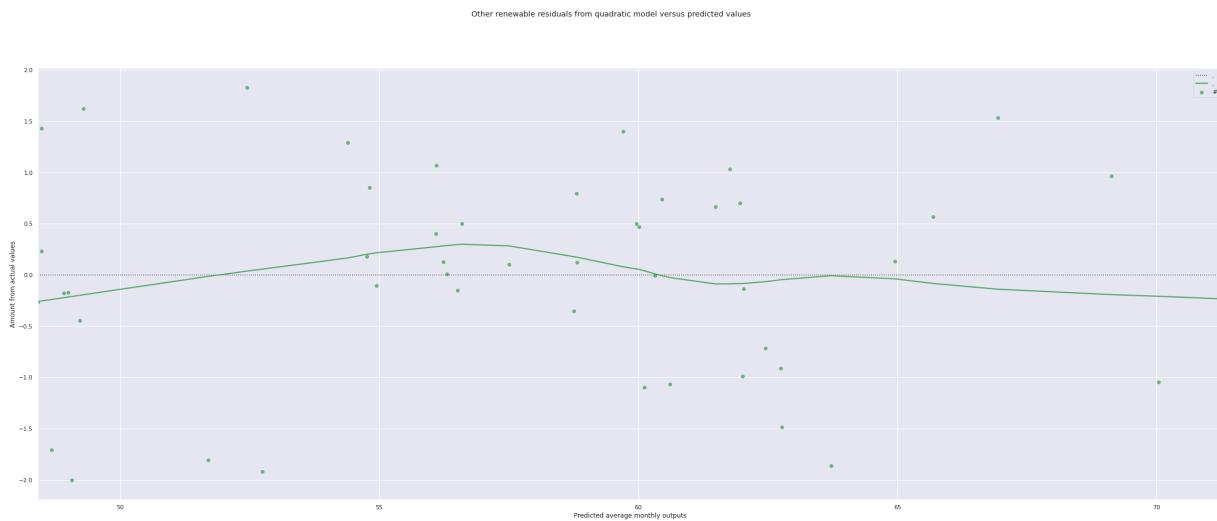
In []:

```
#Predicted average monthly OLS quadratic values versus residuals
sns.residplot(x = Otherrenewable_ypred, y = standardized_residualsOtherrenewable)

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[]:



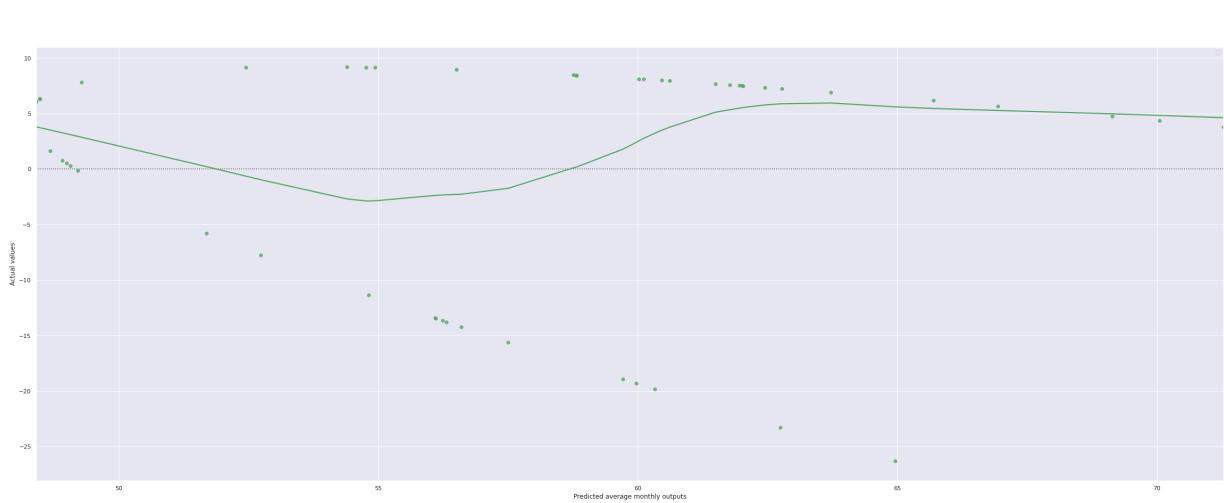
In []:

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 outputs per month")
plt.xlabel("Predicted average monthly outputs")
plt.ylabel("Actual values")
plt.legend(..#)
sns.residplot(x = Otherrenewable_ypred, y = other_renewable, lowess = True,
#Predicted OLS average monthly quadratic values versus actual values
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c45ad850>
```



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
dfotherrenewable = ({'Price':[64.9490188172043, 56.38385416666667, 55.522462
    "Other_Renewable" : [70.66030013642565, 70.51488095238095, 66.6325706594885
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")
    else:
        print("insufficient evidence against the null hypothesis(Ho), fail to reject the null hypothesis")
```

```
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 Other Renewable Ratio")  
plt.ylabel('Autocorrelations')  
plt.xlabel('Lags')  
  
plt.show  
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function  
plot_pacf(df_otherrenewable["Ratio"])  
plt.suptitle("Partial autocorrelations of Other Renewable Ratio")  
plt.ylabel('Partial autocorrelations')  
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(4)  
df_otherrenewable.head()
```

```

{'Price': [64.9490188172043, 56.38385416666667, 55.52246298788695, 58.354083
33333335, 57.29405913978494, 65.97490277777777, 71.0720430107527, 63.998064
51612903, 60.25479166666667, 59.40676510067113, 60.72679166666667, 61.901760
752688176, 45.57872311827957, 36.75208333333333, 36.818008075370116, 32.6186
6666666666, 34.69137096774194, 46.26631944444444, 47.502016129032256, 47.602
33870967742, 50.40559722222222, 60.18242953020134, 62.58105555555556, 67.595
13440860215, 79.49208333333334, 59.83779761904762, 50.95989232839838, 51.717
91666666666, 53.772620967741936, 56.25822222222222, 55.25258064516129, 54.08
432795698924, 55.81655555555556, 63.92528859060402, 65.43065277777778, 65.15
127688172043, 56.511975806451616, 60.877098214285716, 48.279717362045766, 5
0.40073611111111, 61.63376344086022, 64.34813888888888, 67.78344086021505, 7
0.36391129032258, 76.91404166666666, 70.36221476510067, 67.04260752688172, 6
6.6235138888889], 'Other_Renewable': [70.66030013642565, 70.51488095238095,
66.63257065948856, 69.70055710306407, 70.56586021505376, 65.51182197496523,
68.3252688172043, 68.1760752688172, 67.96666666666667, 70.66935483870968, 7
0.3125, 71.62314939434724, 77.95430107526882, 74.45402298850574, 73.40242261
103634, 78.275, 79.28225806451613, 83.48333333333333, 78.6271870794078, 78.4
3010752688173, 83.79166666666667, 83.78120805369127, 85.95694444444445, 90.0
9005376344086, 99.20430107526882, 94.90922619047619, 95.6850605652759, 95.97
083333333333, 96.74596774193549, 92.10555555555555, 93.39650537634408, 92.24
731182795699, 94.87222222222222, 95.6268456375839, 94.90138888888889, 91.815
86021505376, 96.98252688172043, 96.75744047619048, 97.64199192462988, 97.151
38888888889, 100.59005376344086, 101.10972222222222, 96.71736204576042, 96.6
1559139784946, 100.19444444444444, 98.61744966442953, 95.88172043010752, 96.
4625], 'Dates': ['2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '201
5-06', '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '20
16-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-07', '2
016-08', '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	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

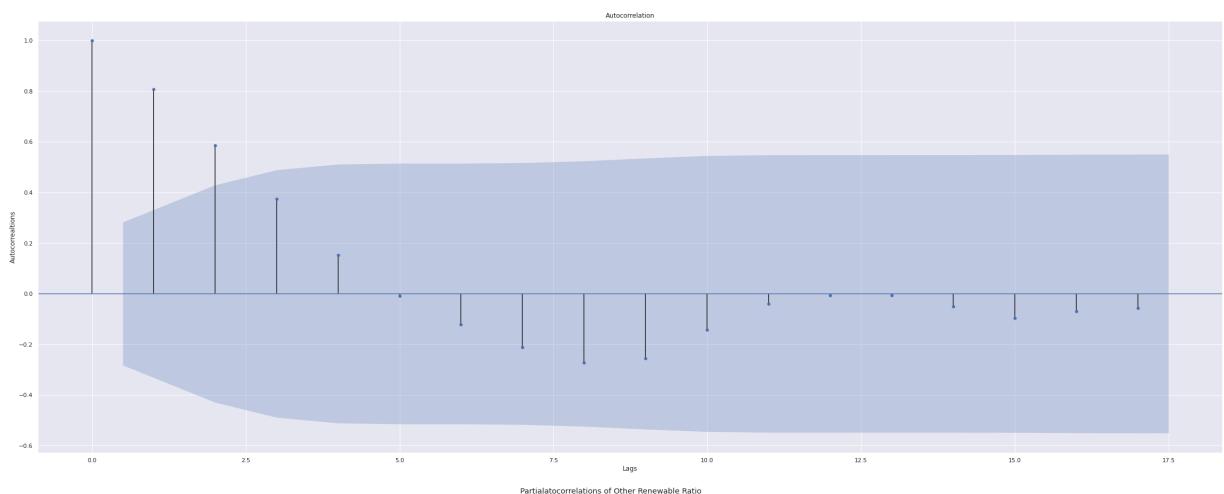
23	67.595134	90.090054	2016-12
24	79.492083	99.204301	2017-01
25	59.837798	94.909226	2017-02
26	50.959892	95.685061	2017-03
27	51.717917	95.970833	2017-04
28	53.772621	96.745968	2017-05
29	56.258222	92.105556	2017-06
30	55.252581	93.396505	2017-07
31	54.084328	92.247312	2017-08
32	55.816556	94.872222	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, 1.1944418131789
818, 1.2316435818046778, 0.9929809551311909, 0.9613522550191385, 1.065283392
306891, 1.1279877464793602, 1.1895842959796394, 1.1578497409504178, 1.157045
4301049409, 1.7103221797805228, 2.025844965392141, 1.9936554541672733, 2.399
6995585349907, 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, 1.45041176971292
97, 1.4092718456115882, 1.7161411452658597, 1.5893898249815874, 2.0224226084
925134, 1.9275787693797464, 1.6320608729330732, 1.571292099011756, 1.4268582
53555079, 1.3730844352755214, 1.3026807884920595, 1.40156829903176, 1.430160
967287293, 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

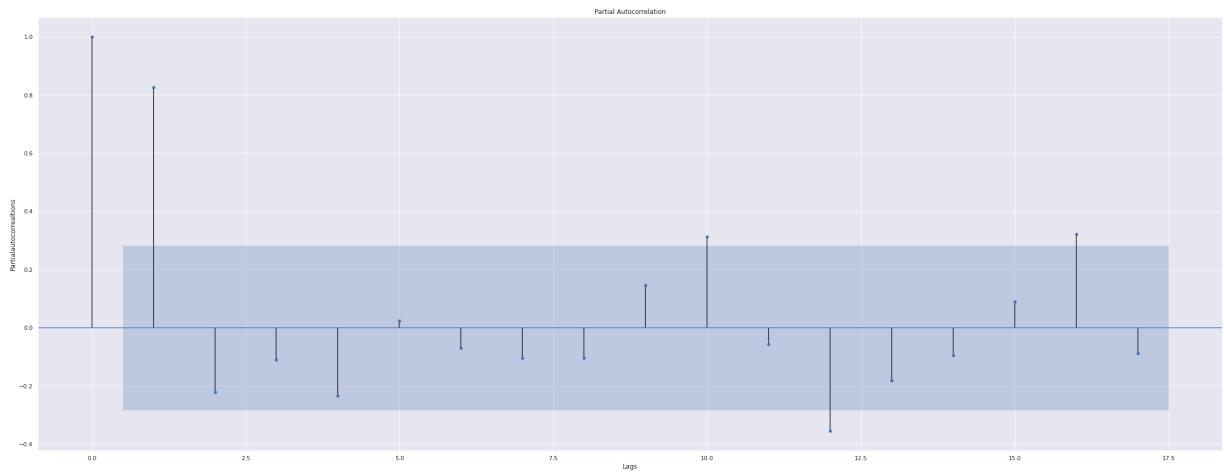
Out[]:

	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

Autocorrelations of Other Renewable Ratio



Partial Autocorrelations of Other Renewable Ratio



In []: `Otherrenewable_Ratio_Autocorrelations = sm.tsa.acf(df_otherrenewable["Ratio"]
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 * n * 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 the average monthly prices of energy per EUR/MWH.

```
In [ ]: df_otherrenewable.describe(include = 'all') # Description table of Dataframe
```

Out[]:

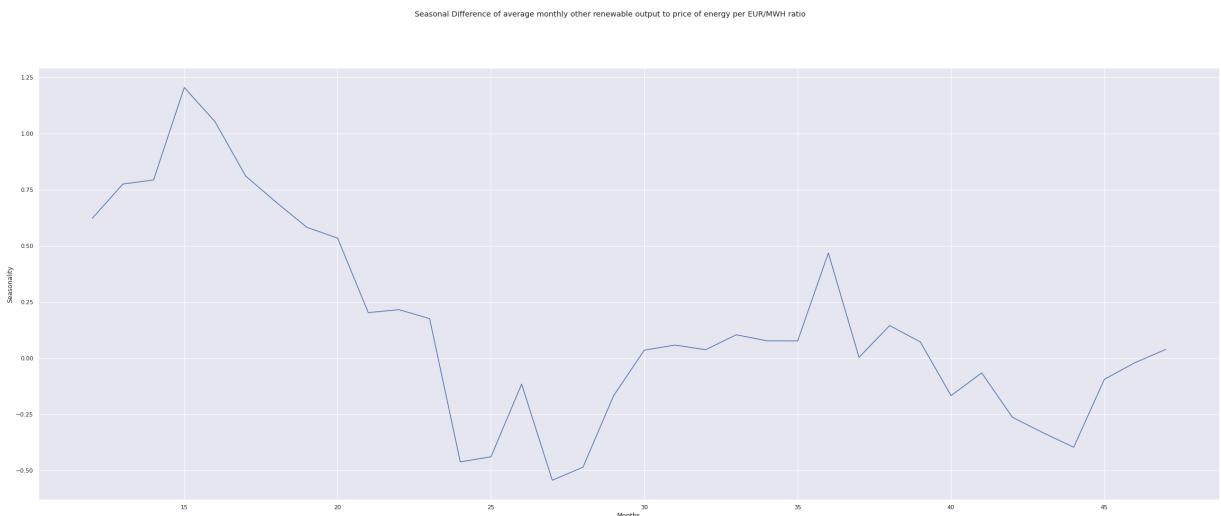
	Price	Other_Renewable	Dates	Ratio	First Difference Ratio	Seasonal Difference Ratio
count	48.000000	48.000000	48	48.000000	47.000000	36.000000
unique	NaN	NaN	48	NaN	NaN	NaN
top	NaN	NaN	2015-01	NaN	NaN	NaN
freq	NaN	NaN	1	NaN	NaN	NaN
mean	57.859848	85.633141	NaN	1.524915	0.007658	0.145119
std	10.320573	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")
plt.ylabel('Seasonality')
plt.xlabel('Months')

df_otherrenewable['Seasonal Difference Ratio'].plot() # Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c44509d0>



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) )

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

adfuller_test(df_otherrenewable["Other_Renewable"])

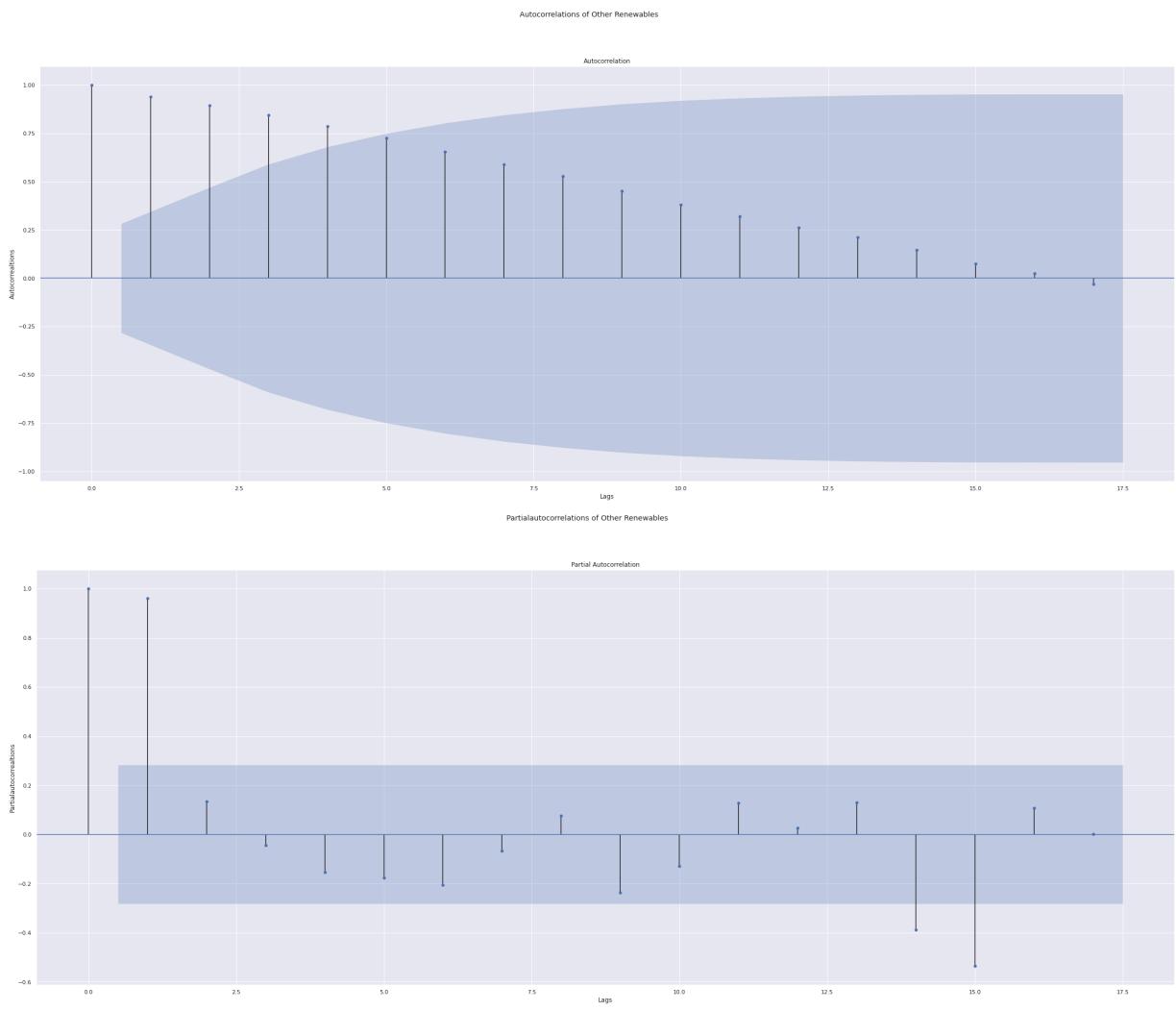
test_result=adfuller(df_otherrenewable["Other_Renewable"])

from statsmodels.graphics.tsaplots import plot_acf # Autocorrelation Plot
plot_acf(df_otherrenewable["Other_Renewable"])
plt.suptitle("Autocorrelations of Other Renewables")
plt.ylabel('Autocorrelations')
plt.xlabel('Lags')

plt.show
from statsmodels.graphics.tsaplots import plot_pacf # Partial autocorrelation Function
plot_pacf(df_otherrenewable["Other_Renewable"])
plt.suptitle("Partialautocorrelations of Other Renewables")
plt.ylabel('Partialautocorrelations')
plt.xlabel('Lags')
plt.show
otherrenewable_Autocorrelations = sm.tsa.acf(df_otherrenewable["Other_Renewable"])
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 * n_lags * p.log10(nobs)), nobs - 1) after 0.12 is released. Set the number of lags to a non-integer to silence this warning.
FutureWarning,
```



```
In [ ]: Otherrenewable_Autocorrelations = sm.tsa.acf(df_otherrenewable["Other_Renewables"])
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 -0.38835355 -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['Seasonal Difference']=df_otherrenewable["Other_Renewable"]
df_otherrenewable.head()

plt.suptitle("Seasonal Difference of average monthly Other Renewable output")
plt.ylabel('Seasonality')
plt.xlabel('Months')
df_otherrenewable['Seasonal Difference'].plot() # Seasonality Plot
```

Out[]: <matplotlib.axes._subplots.AxesSubplot at 0x7f81c42b5250>

