Econ 430 Homework 3

December 2, 2020

0.1 Wong Jing Hei (Koby) UID: 805643634

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import copy
import statsmodels.api as sm
import scipy.stats
from pylab import rcParams
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from  statsmodels.regression.linear_model import OLS
from statsmodels.tools import add_constant
import seaborn as sns
from itertools import cycle
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import kpss
from sklearn.metrics import mean_squared_error
%matplotlib inline
rcParams['figure.figsize'] = 15, 10
energy = pd.read_csv("energy.csv",index_col=[0])
```

Section I. Introduction

Over the past few decades, the average lifestyle in the US has grown more and more modernized. With the popularization and more widespread access of electricity and fuel, these technologies generated an increased demand for energy production. Just in 2019 alone, the US Energy Information Administration reported that 101.04 quadrillion BTU of fuel/energy and 4.12 trillion kilowatthours of electricity was produced. As a result of this, I want to analyze the overall trend of energy production in the US over the past few decades and observe whether there are any inherent trends in seasonality that can be observed. Using data found on Kaggle, I analyze the trend and seasonality of the industrial production index on electric and gas utilities from 1950 to 2018. The index measures relative production of energy over the years, setting 2012 as the default year with an index of 100.

[2]:
```
#Limiting the datset to consider dates from 1950 to 2018
energy=energy[132:960]
energy.head()
```

[2]:
```
            IPG2211A2N
DATE
1950-01-01      9.0934
1950-02-01      8.8867
1950-03-01      8.9384
1950-04-01      8.8350
1950-05-01      8.8092
```

[3]:
```
energy.tail()
```

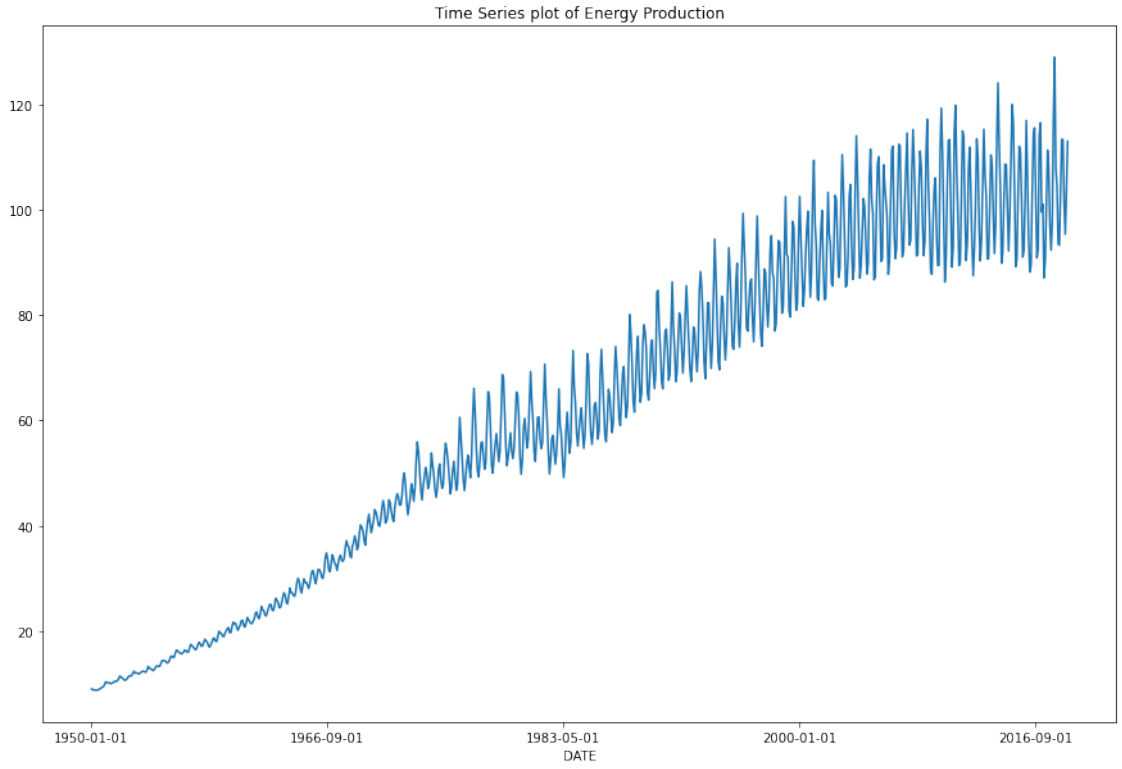
[3]:
```
            IPG2211A2N
DATE
2018-08-01    113.2758
2018-09-01    101.5656
2018-10-01     95.3203
2018-11-01    103.5750
2018-12-01    112.9498
```

Section II Part 1. Modelling and Forecasting Trend

1a. Show a time-series plot of your data.

[4]:
```
energy.IPG2211A2N.plot()
plt.title('Time Series plot of Energy Production')
```

[4]:
```
Text(0.5, 1.0, 'Time Series plot of Energy Production')
```

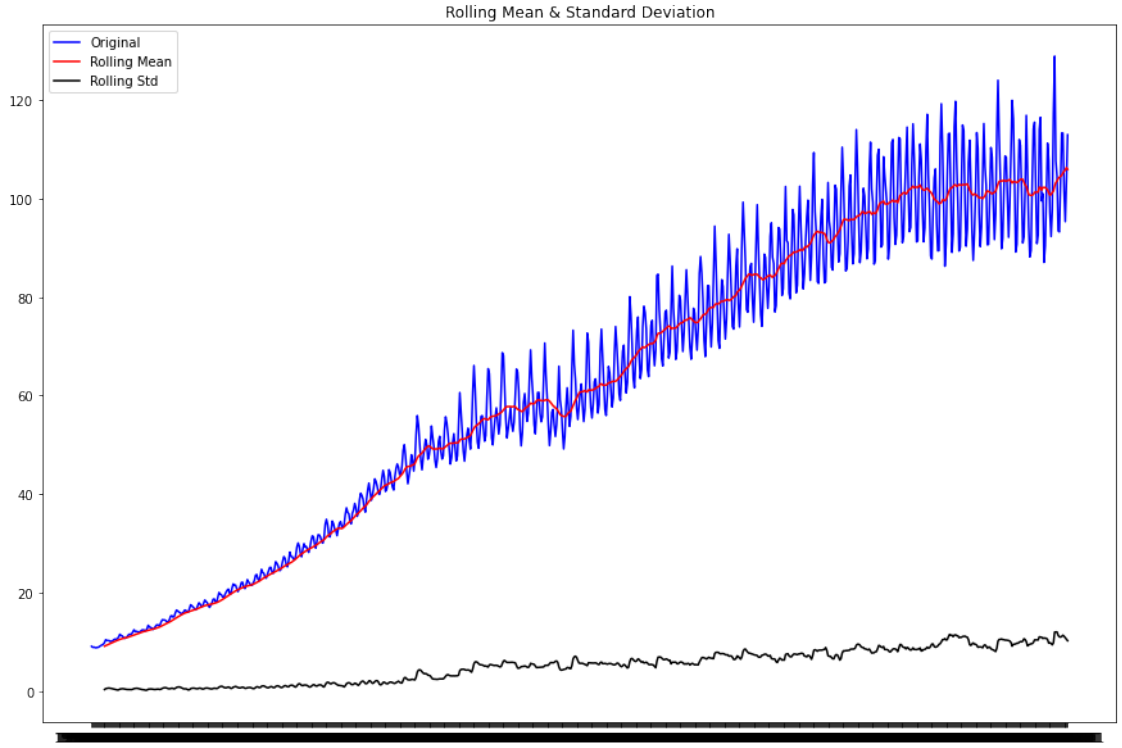


There is a clear trend with features of seasonality observed in the diagrams above.

1b. Does your plot in (a) suggest that the data are covariance stationary? Explain your answer.

[5]:
```
#Defining and plotting the rolling mean and rolling standard deviation of the␣
 ↪time series for reference
rolmean = energy.rolling(window=12).mean() #window size 12 denotes 12 months,␣
 ↪giving rolling mean at yearly level
rolstd = energy.rolling(window=12).std()
```

[6]:
```
rcParams['figure.figsize'] = 15, 10
orig = plt.plot(energy, color='blue', label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```



No. The trend of the rolling mean is clearly going upwards and is not centered around a certain value, with obvious seasonality observed each year, although the standard deviation seem to remain pretty constant over time. Hence, the model is not stationary, as one of the two "rolling" curves is

not parallel to the x axis.

[7]:
```
#Conducting a KPSS test
result = kpss(energy.IPG2211A2N.values, regression='c')
print('\nKPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[3].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
warnings.filterwarnings('ignore')
```

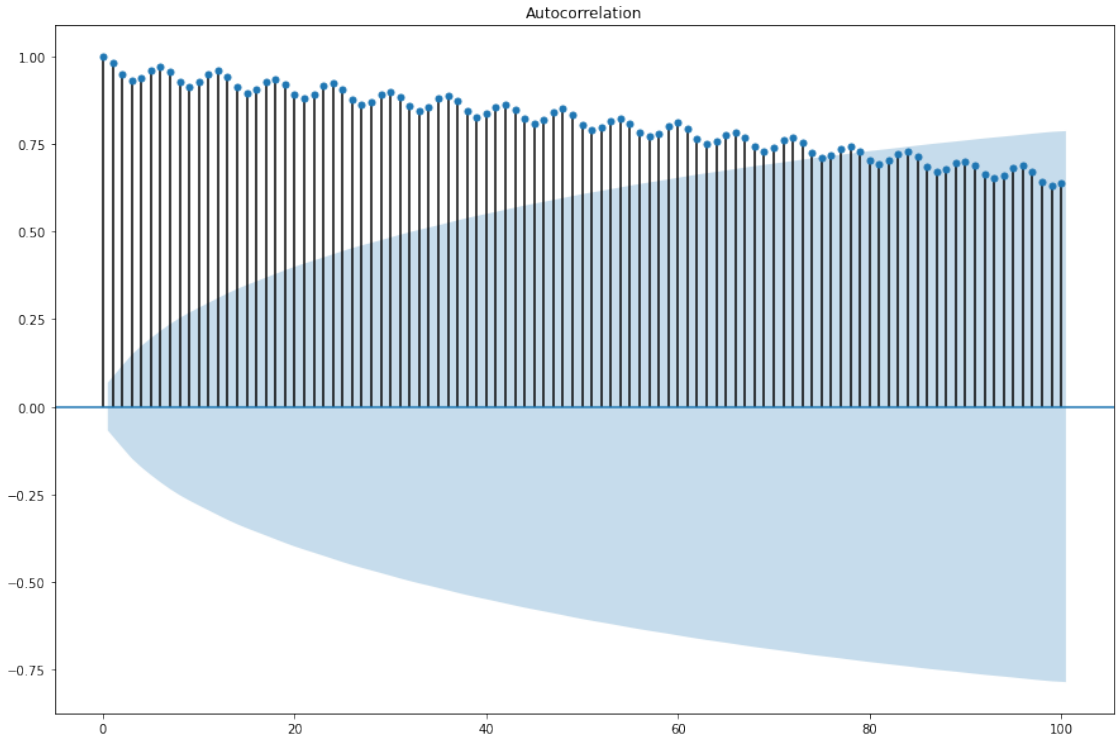
```
KPSS Statistic: 3.848177
p-value: 0.010000
Critial Values:
   10%, 0.347
Critial Values:
   5%, 0.463
Critial Values:
   2.5%, 0.574
Critial Values:
   1%, 0.739
```

```
C:\Users\Koby\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:1661:
FutureWarning: The behavior of using lags=None will change in the next release.
Currently lags=None is the same as lags='legacy', and so a sample-size lag
length is used. After the next release, the default will change to be the same
as lags='auto' which uses an automatic lag length selection method. To silence
this warning, either use 'auto' or 'legacy'
  warn(msg, FutureWarning)
C:\Users\Koby\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:1685:
InterpolationWarning: p-value is smaller than the indicated p-value
  warn("p-value is smaller than the indicated p-value", InterpolationWarning)
```

The null hypothesis that the series is stationary is also rejected based on the KPSS test, further proof that the plot in (a) is not covariance stationary.

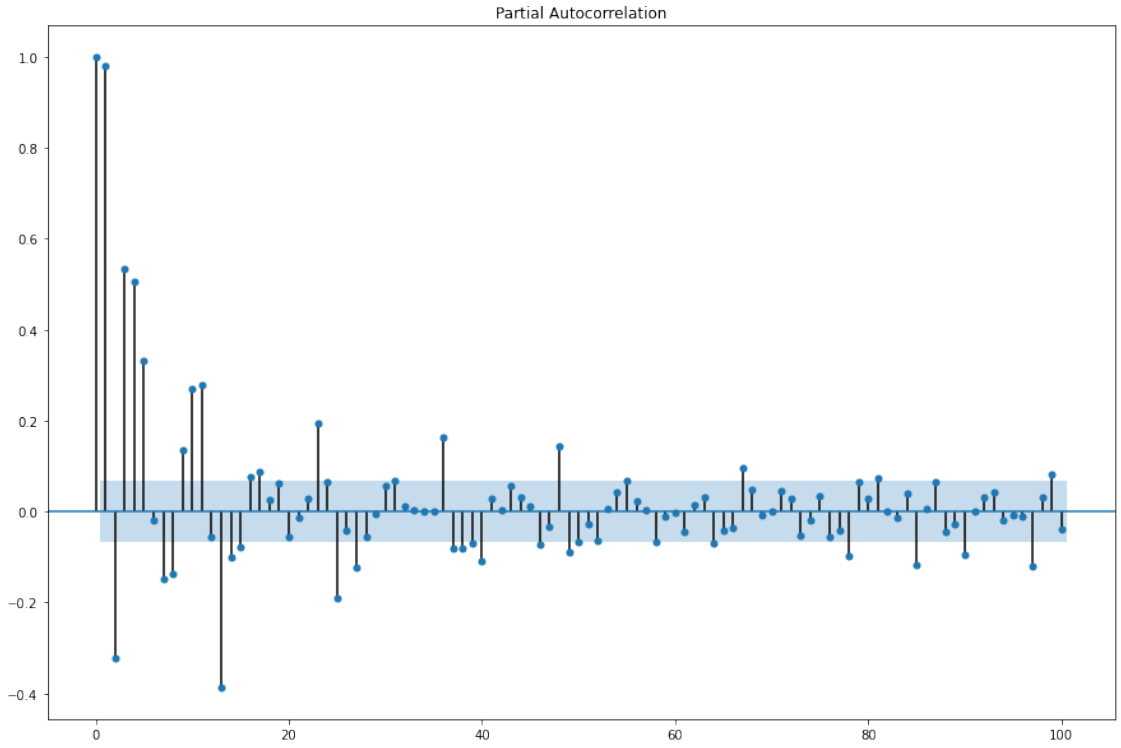
1c. Plot and discuss the ACF and PACF of your data.

[8]:
```
acf_plot = plot_acf(energy.IPG2211A2N, lags=100)
```



From looking at the ACF plot, we can find it very slowly decaying even after 100 lags, which matches with the expected trend due to the non-stationarity of data found in the plot of 1a.

[9]:
```
pacf_plot2 = plot_pacf(energy.IPG2211A2N, lags=100)
```



The PACF plot indicates terms that are initially significant in the first few lags, then stop being significant as the number of lags increase. This suggests that there are autoregressive terms in the model.

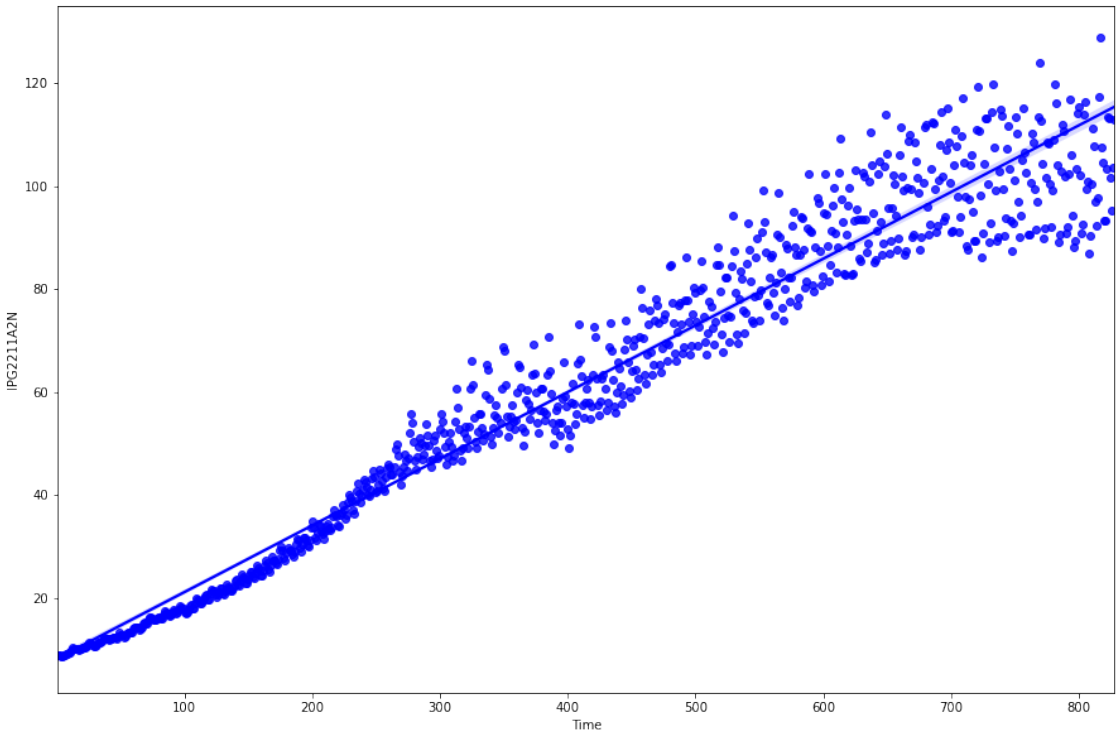
1d. Fit a linear and nonlinear (e.g., polynomial, exponential, quadratic + periodic, etc.) model to your series. In one window, show both figures of the original times series plot with the respective fit.

As the original times series exhibits slight features of a quadratic function, I will attempt to fit a quadratic model to the series.

[10]:
```
test=pd.Series(range(1,len(energy)+1))
test.values
energy["Time"]=test.values
energy["Quad"]=np.square(test.values)
reglinear=OLS(energy.IPG2211A2N,add_constant(energy["Time"])).fit()
regquad=OLS(energy.IPG2211A2N,add_constant(energy["Time"]+energy["Quad"])).fit()
```

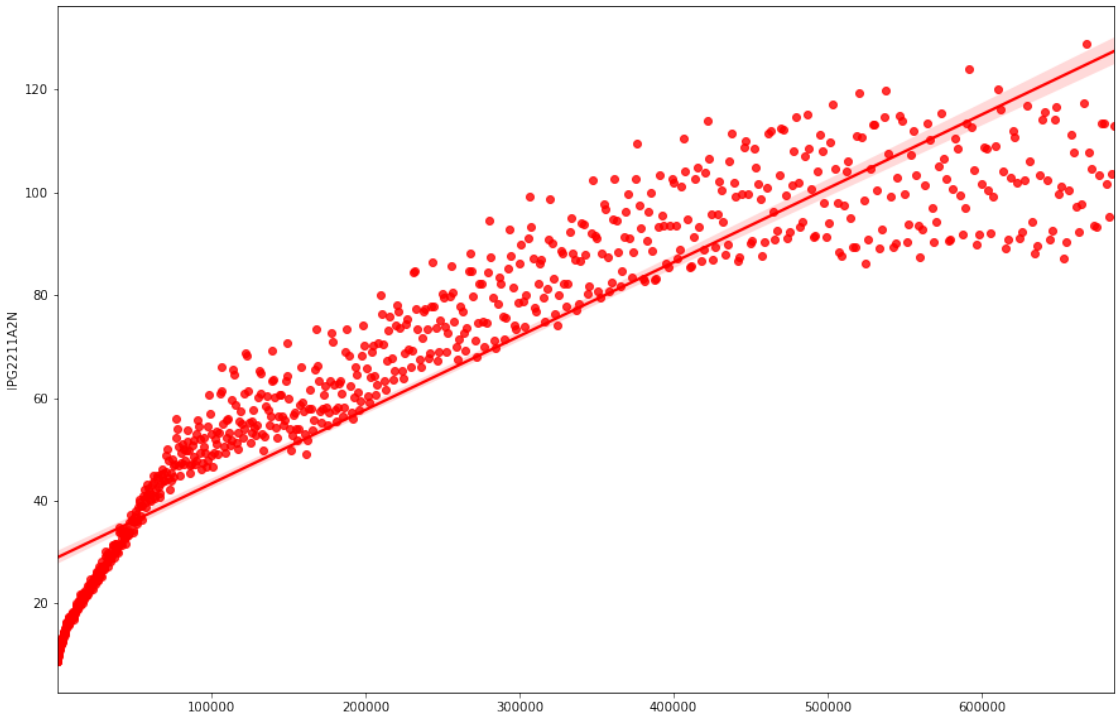
[11]:
```
sns.regplot(x=energy["Time"],y=energy.IPG2211A2N, color="blue")
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ff75dbffa0>
```



[12]:
```
sns.regplot(x=energy["Time"]+energy["Quad"],y=energy.IPG2211A2N, color="red")
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1ff7574f6a0>

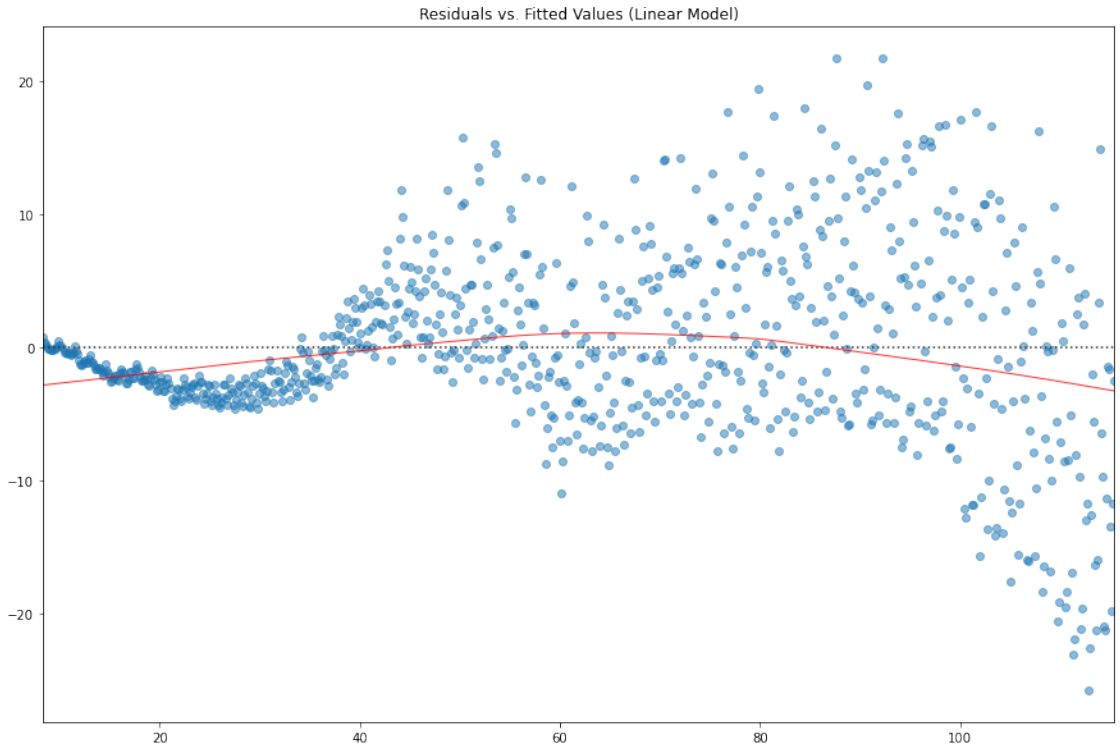


1e. For each model, plot the respective residuals vs. fitted values and discuss your observations

[13]:
```
#Defining variables of time and regressions
test=pd.Series(range(1,len(energy)+1))
test.values
energy["Time"]=test.values
energy["Quad"]=np.square(test.values)
reglinear=OLS(energy.IPG2211A2N,add_constant(energy["Time"])).fit()
X=energy[["Time", "Quad"]]
regquad=OLS(energy.IPG2211A2N,add_constant(X)).fit()
```

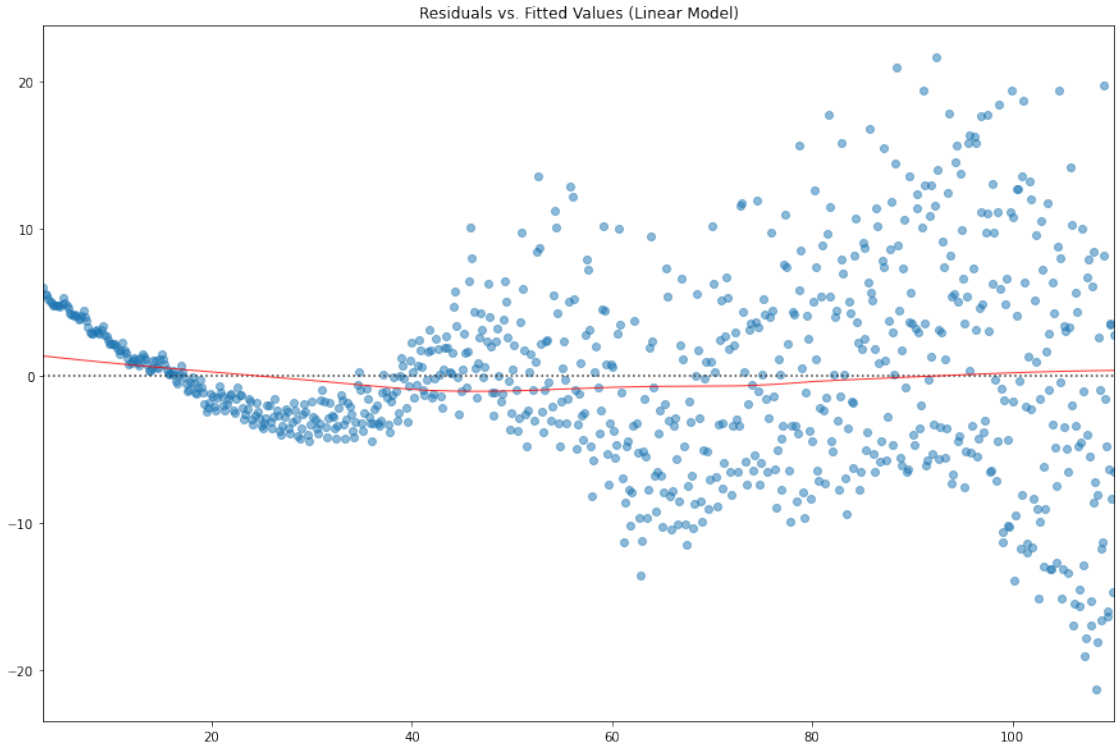
[14]:
```
#Plotting the residuals vs fitted values graph
sns.residplot(x=reglinear.fittedvalues,y=reglinear.resid,lowess=True,
                          scatter_kws={'alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
plt.title('Residuals vs. Fitted Values (Linear Model)')
```

[14]:
```
Text(0.5, 1.0, 'Residuals vs. Fitted Values (Linear Model)')
```



[15]:
```
sns.residplot(x=regquad.fittedvalues,y=regquad.resid,lowess=True,
                          scatter_kws={'alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
plt.title('Residuals vs. Fitted Values (Linear Model)')
```

[15]:
```
Text(0.5, 1.0, 'Residuals vs. Fitted Values (Linear Model)')
```



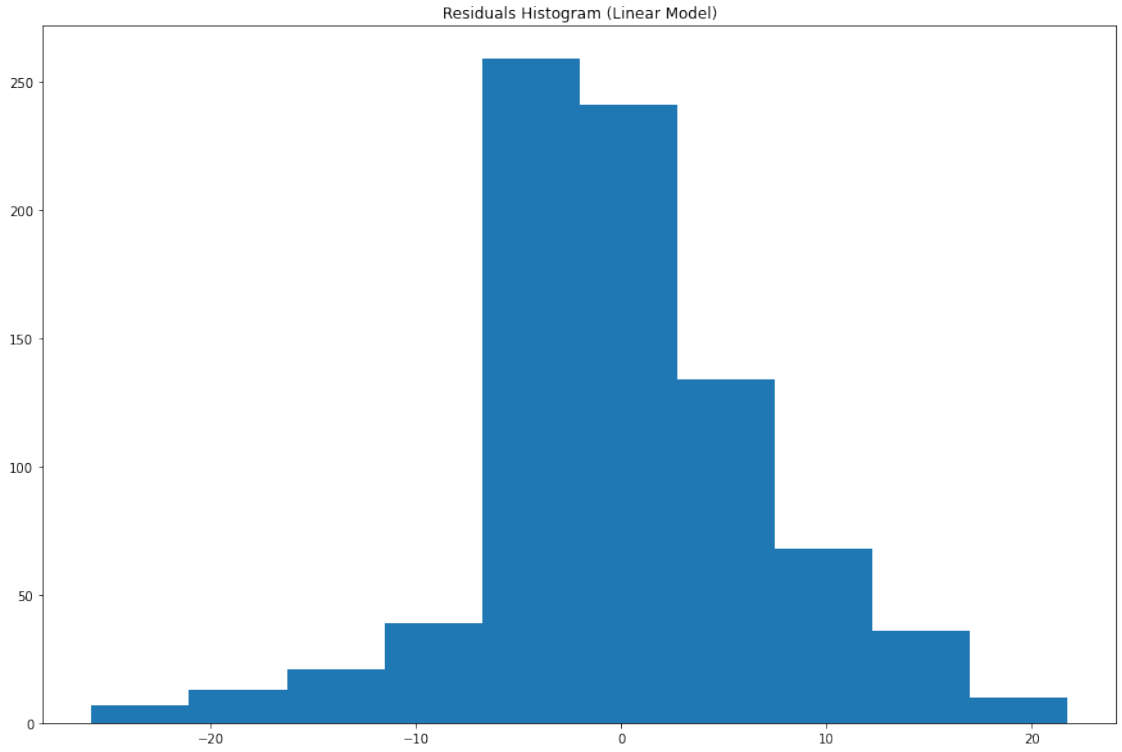
The residuals of both models are more or less randomly distributed and deviate closely around 0. Just from looking at the graphs, it is hard to identify which model is better. More diagnostic tests are needed.

In the quadratic model, there is a clear trend within the residuals and does not deviate around zero as closely as seen in the graph. This implies that the trend is already linear in nature, and does not require transformation.

1f. For each model, plot a histogram of the residuals and discuss your observations.

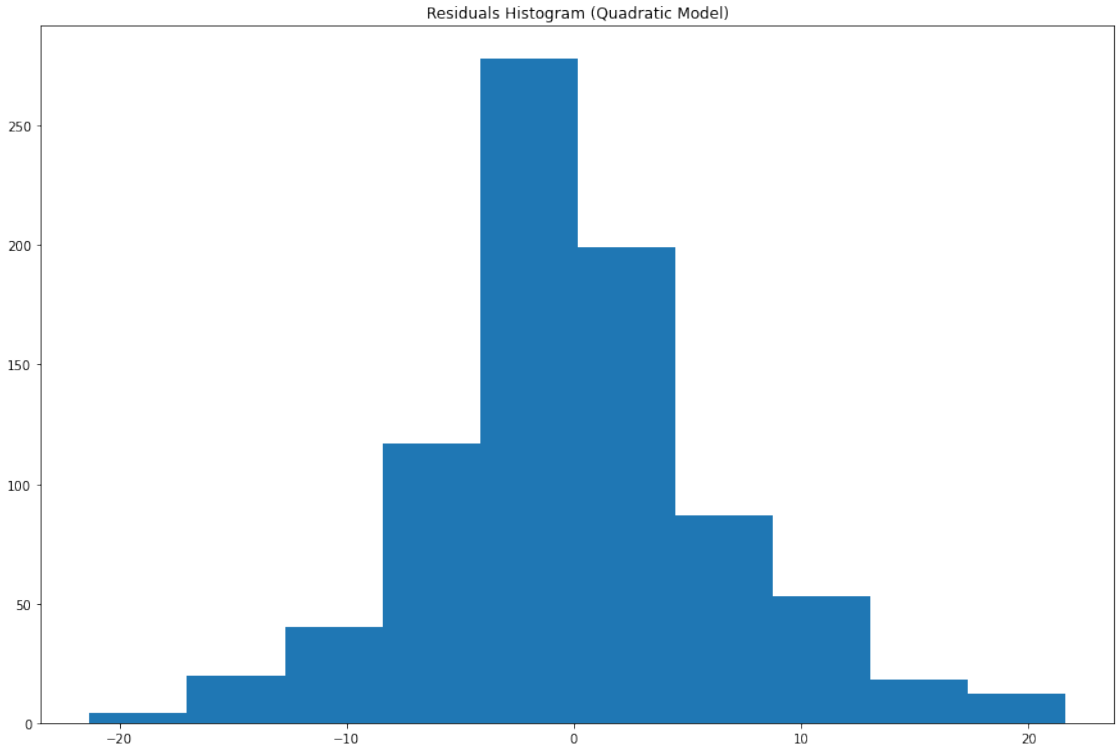
[16]:
```
a=plt.hist(reglinear.resid)
plt.title('Residuals Histogram (Linear Model)')
```

[16]:
```
Text(0.5, 1.0, 'Residuals Histogram (Linear Model)')
```



[17]:
```
a=plt.hist(regquad.resid)
plt.title('Residuals Histogram (Quadratic Model)')
```

[17]:
```
Text(0.5, 1.0, 'Residuals Histogram (Quadratic Model)')
```



The histogram of residuals in the quadratic model is slightly more normally distributed and more skewed towards zero than the ones in the linear model, which seem to be slightly skewed to the right. This implies that the assumption of normality is more likely to be held in the quadratic model than the linear model. However, the difference is small, and more diagnostics is still needed to test which model is better.

1g. For each model, discuss the associated diagnostic statistics (R^2, t—distribution, F—distribution, etc.)

[18]:
```
reglinear.summary()
```

[18]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             IPG2211A2N   R-squared:                       0.952
Model:                            OLS   Adj. R-squared:                  0.952
Method:                 Least Squares   F-statistic:                 1.636e+04
Date:                Tue, 01 Dec 2020   Prob (F-statistic):               0.00
Time:                        18:16:09   Log-Likelihood:                -2780.5
No. Observations:                 828   AIC:                             5565.
Df Residuals:                     826   BIC:                             5574.
Df Model:                           1
Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          8.2518      0.484     17.040      0.000       7.301       9.202
Time           0.1294      0.001    127.902      0.000       0.127       0.131
==============================================================================
Omnibus:                       23.591   Durbin-Watson:                   0.704
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.110
Skew:                          -0.018   Prob(JB):                     2.93e-12
Kurtosis:                       4.240   Cond. No.                         958.
==============================================================================

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

[19]:
```
regquad.summary()
```

[19]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             IPG2211A2N   R-squared:                       0.957
Model:                            OLS   Adj. R-squared:                  0.957
Method:                 Least Squares   F-statistic:                     9277.
Date:                Tue, 01 Dec 2020   Prob (F-statistic):               0.00
Time:                        18:16:09   Log-Likelihood:                -2730.2
No. Observations:                 828   AIC:                             5466.
Df Residuals:                     825   BIC:                             5481.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.9767      0.685      4.345      0.000       1.632       4.321
Time           0.1676      0.004     43.909      0.000       0.160       0.175
Quad         -4.6e-05   4.46e-06    -10.318      0.000   -5.47e-05   -3.72e-05
==============================================================================
Omnibus:                       24.512   Durbin-Watson:                   0.795
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               32.497
Skew:                           0.307   Prob(JB):                     8.78e-08
Kurtosis:                       3.752   Cond. No.                     9.23e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
```

```
specified.
[2] The condition number is large, 9.23e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

The R squared statistic in the quadratic regression is slightly larger than the statistic in the linear regression model. This implies that the proportion of the variance for a energy produced that's explained by the time variables is better explained in the quadratic model. The t statistic for the quadratic term also implies that it is significant at the 0.1% significance level. However, comparison of the F statistic implies that the overall significance of the regression is stronger in the linear model than the quadratic one, with higher t statistics in the time varaibles as well, though this could be artificially increased due to small amount of explanatory variables in the regression.

1h. Select a trend model using AIC and one using BIC (show the values obtained from each criterion). Do the selected models agree?

[20]:
```
names = ['linear_model', 'quadratic_model']
models = [reglinear, regquad]
df = pd.DataFrame({name:[getattr(mod, 'aic'), getattr(mod, 'bic')] for name,␣
↪mod in zip(names,models)}).T
df.columns = ['AIC', 'BIC']
df
```

[20]:

	AIC	BIC
linear_model	5564.912316	5574.350343
quadratic_model	5466.409003	5480.566042

The AIC and BIC values are both smaller in the quadratic model, suggsting the the quadratic model is the ideal model for trend. Hence the selected models agree.

1i. Use your preferred model to forecast h-steps (at least 16) ahead. Your forecast should include the respective uncertainty prediction interval. Depending on your data, h will be in days, months, years, etc.

Since previous tests indicate that the quadratic model is the better fit for the model, I will choose the quadratic model for further analysis.

[21]:
```
#Manually calculating forecast errors and prediction intervals to match with␣
↪forecasts in a dataframe
df = pd.DataFrame()
energy["X"]=1
X=energy[["X","Time","Quad"]]
X=np.asarray(X)
X=X.reshape((828,3))
ypred=regquad.predict(X)
yactual=np.asarray(energy["IPG2211A2N"])
RSS=mean_squared_error(yactual,ypred)*828
RSE=(RSS/regquad.df_resid)**0.5
meant=np.mean(energy["Time"])
sdt=np.std(energy["Time"])
```

```
meanq=np.mean(energy["Quad"])
sdq=np.std(energy["Quad"])
for i in range(len(energy)+1,len(energy)+1+16):
    predictions=regquad.predict([[1,i,i**2]])
    f_e=(RSE)*(1+1/828+(i-meant)/((sdt**2)*(828-1))+(i**2-meanq)/
 ↪((sdq**2)*(828-1)))**0.5
    p_iu=scipy.stats.t.ppf(1 - 0.05/ 2, regquad.df_resid)*f_e+(regquad.
 ↪predict([[1,i,i**2]]))
    p_il=-scipy.stats.t.ppf(1 - 0.05/ 2, regquad.df_resid)*f_e+(regquad.
 ↪predict([[1,i,i**2]]))
    df=df.append([[predictions,p_il,p_iu]],ignore_index=True)
df.rename(columns={0: 'Prediction',1:"Lower Prediction Interval",2:"Upper␣
 ↪Prediction Interval"},
          index={0:"2019-01-01",1:"2019-02-01",2:"2019-03-01",3:"2019-04-01",4:
 ↪"2019-05-01",5:"2019-06-01",6:"2019-07-01",7:"2019-08-01",8:"2019-09-01",9:
 ↪"2019-10-01",10:"2019-11-01",11:"2019-12-01",12:"2020-01-01",13:
 ↪"2020-02-01",14:"2020-03-01",15:"2020-04-01"})
```

[21]:

	Prediction	Lower Prediction Interval	Upper Prediction Interval
2019-01-01	[110.28753999311134]	[97.41331928621338]	[123.1617607000093]
2019-02-01	[110.37880699351943]	[97.50458615023837]	[123.2530278368005]
2019-03-01	[110.4699819961628]	[97.59576101649827]	[123.34420297582733]
2019-04-01	[110.56106500104144]	[97.68684388499308]	[123.4352861170898]
2019-05-01	[110.65205600815537]	[97.77783475572281]	[123.52627726058793]
2019-06-01	[110.74295501750456]	[97.86873362868741]	[123.6171764063217]
2019-07-01	[110.83376202908903]	[97.95954050388694]	[123.70798355429112]
2019-08-01	[110.92447704290879]	[98.05025538132139]	[123.79869870449619]
2019-09-01	[111.01510005896382]	[98.14087826099075]	[123.8893218569369]
2019-10-01	[111.10563107725413]	[98.23140914289499]	[123.97985301161327]
2019-11-01	[111.19607009777971]	[98.32184802703415]	
2019-12-01	[111.28641712054059]	[98.41219491340823]	
2020-01-01	[111.37667214553673]	[98.50244980201721]	
2020-02-01	[111.46683517276813]	[98.59261269286108]	
2020-03-01	[111.55690620223484]	[98.68268358593988]	
2020-04-01	[111.64688523393681]	[98.77266248125358]	

```
2019-11-01      [124.07029216852527]
2019-12-01      [124.16063932767295]
2020-01-01      [124.25089448905625]
2020-02-01      [124.34105765267518]
2020-03-01       [124.4311288185298]
2020-04-01      [124.52110798662005]
```

The above table illustrates the forecast and 95% prediction intervals 16 months ahead.

Section II Part 2. Modeling and Forecasting Seasonality

2a. Construct and test (by looking at the diagnostic statistics) a model with a full set of seasonal dummies.

[22]:
```
#Creating dummy variables for each respctive month and adding it to the␣
 ↪original time series dataframe.
i = iter(["Jan", "Feb",␣
 ↪"March","Apr","May","June","July","Aug","Sept","Oct","Nov","Dec"])
energy['Month'] = energy.index.map(dict(zip(energy.index, cycle(i))))
date=pd.get_dummies(energy["Month"])
energy = pd.concat([energy, date], axis=1)
```

[23]:
```
#Constructing a model with only seasonal dummies.
X=energy[["Jan", "Feb",␣
 ↪"March","Apr","May","June","July","Aug","Sept","Oct","Nov", "Dec"]]
regseason=OLS(energy.IPG2211A2N,X).fit()
```

[24]:
```
regseason.summary()
```

[24]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             IPG2211A2N   R-squared:                       0.020
Model:                            OLS   Adj. R-squared:                  0.006
Method:                 Least Squares   F-statistic:                     1.477
Date:                Tue, 01 Dec 2020   Prob (F-statistic):              0.135
Time:                        18:16:09   Log-Likelihood:                -4028.9
No. Observations:                 828   AIC:                             8082.
Df Residuals:                     816   BIC:                             8138.
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Jan           69.9622      3.808     18.373      0.000      62.488      77.437
Feb           65.4621      3.808     17.191      0.000      57.988      72.937
March         61.6258      3.808     16.184      0.000      54.151      69.100
```

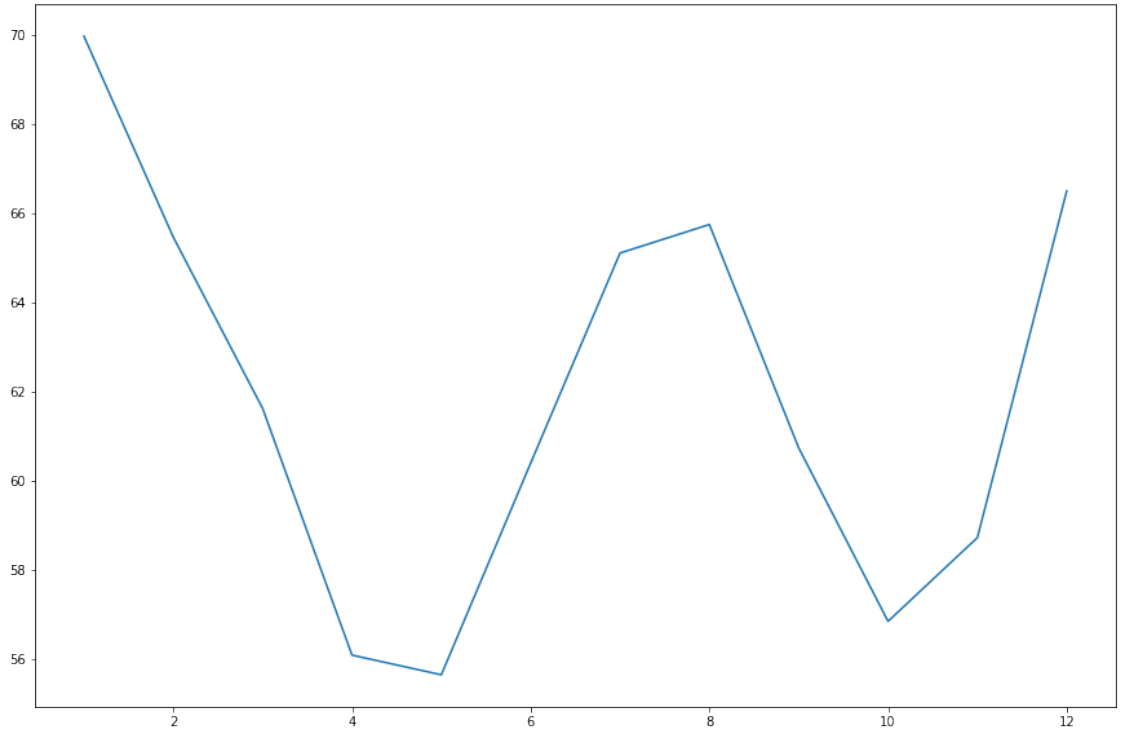
```
Apr            56.0992      3.808     14.732      0.000      48.625      63.574
May            55.6587      3.808     14.617      0.000      48.184      63.133
June           60.4011      3.808     15.862      0.000      52.927      67.876
July           65.1050      3.808     17.097      0.000      57.631      72.580
Aug            65.7444      3.808     17.265      0.000      58.270      73.219
Sept           60.7410      3.808     15.951      0.000      53.267      68.215
Oct            56.8563      3.808     14.931      0.000      49.382      64.331
Nov            58.7317      3.808     15.424      0.000      51.257      66.206
Dec            66.4990      3.808     17.463      0.000      59.025      73.974
==============================================================================
Omnibus:                      771.610   Durbin-Watson:                   0.014
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               55.746
Skew:                          -0.156   Prob(JB):                     7.85e-13
Kurtosis:                       1.768   Cond. No.                         1.00
==============================================================================

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

All the seasonal dummy variables are significant at the 0.1% significance level. Just by looking at the coefficients, a W shaped seasonal trend can be estimated, which matches with the seasonal plot plotted in 1a. The r squared value and F statistic of this regression is understandably low, as it omits the trend line of the model and only includes seasonality.

2b. Plot the estimated seasonal factors and interpret your plot.

[25]:
```
plt.plot([1, 2, 3, 4,5,6,7,8,9,10,11,12], [regseason.fittedvalues[0],regseason.
↪fittedvalues[1],regseason.fittedvalues[2],regseason.
↪fittedvalues[3],regseason.fittedvalues[4],regseason.
↪fittedvalues[5],regseason.fittedvalues[6],regseason.
↪fittedvalues[7],regseason.fittedvalues[8],regseason.
↪fittedvalues[9],regseason.fittedvalues[10],regseason.fittedvalues[11]])
```

[25]:
```
[<matplotlib.lines.Line2D at 0x1ff76df8880>]
```



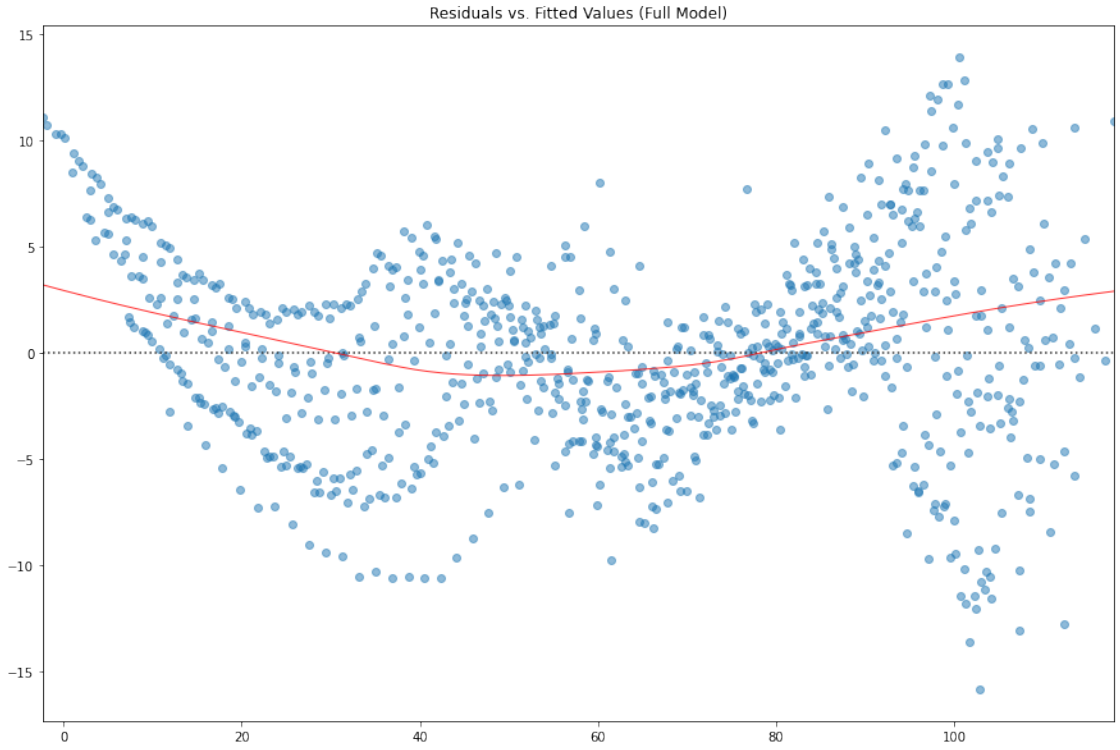
The seasonal trend follows a W shaped pattern, with peaks at January and August. This is to be expected, as energy consumption increases during summer and winter (Air conditioning and heating respectively). This will be matched with an increase in energy production, hence producing this seasonal trend.

2c. In order to improve your model, add the trend model from problem 1 to your seasonal model. We will refer to this model as the full model. For the full model, plot the respective residuals vs. fitted values and discuss your observations.

[26]:
```
X=energy[["Feb", "March","Apr","May","June","July","Aug","Sept","Oct","Nov",␣
 ↪"Dec","Time","Quad"]]
regfull=OLS(energy.IPG2211A2N,add_constant(X)).fit()
```

[27]:
```
sns.residplot(x=regfull.fittedvalues,y=regfull.resid,lowess=True,
                          scatter_kws={'alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
plt.title('Residuals vs. Fitted Values (Full Model)')
```

[27]:
```
Text(0.5, 1.0, 'Residuals vs. Fitted Values (Full Model)')
```



The residuals vs fitted model for the full model is much more randomly distributed and is more centered around zero than the graphs seen in part 1e. This unsurprisingly implies that the full model that accounts for both seasonality and trend is a better fit than a model that only accounts for trend.

2d. Interpret the respective summary statistics including the error metrics of your full model.

[28]:
```
regfull.summary()
```

[28]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             IPG2211A2N   R-squared:                       0.978
Model:                            OLS   Adj. R-squared:                  0.978
Method:                 Least Squares   F-statistic:                     2777.
Date:                Tue, 01 Dec 2020   Prob (F-statistic):               0.00
Time:                        18:16:10   Log-Likelihood:                -2457.9
No. Observations:                 828   AIC:                             4944.
Df Residuals:                     814   BIC:                             5010.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
```

```
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          11.7105      0.737     15.879      0.000      10.263      13.158
Feb            -4.6301      0.809     -5.726      0.000      -6.217      -3.043
March          -8.5963      0.809    -10.631      0.000     -10.183      -7.009
Apr           -14.2528      0.809    -17.626      0.000     -15.840     -12.666
May           -14.8229      0.809    -18.331      0.000     -16.410     -13.236
June          -10.2102      0.809    -12.627      0.000     -11.797      -8.623
July           -5.6357      0.809     -6.970      0.000      -7.223      -4.049
Aug            -5.1258      0.809     -6.339      0.000      -6.713      -3.539
Sept          -10.2586      0.809    -12.686      0.000     -11.846      -8.671
Oct           -14.2725      0.809    -17.650      0.000     -15.860     -12.685
Nov           -12.5263      0.809    -15.490      0.000     -14.114     -10.939
Dec            -4.8880      0.809     -6.045      0.000      -6.475      -3.301
Time            0.1677      0.003     60.631      0.000       0.162       0.173
Quad       -4.601e-05   3.23e-06    -14.244      0.000   -5.23e-05   -3.97e-05
==============================================================================
Omnibus:                        1.668   Durbin-Watson:                   0.634
Prob(Omnibus):                  0.434   Jarque-Bera (JB):                1.614
Skew:                          -0.011   Prob(JB):                        0.446
Kurtosis:                       3.215   Cond. No.                     3.83e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.83e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

The month of January is omitted in the regression to avoid multicollinearity. Condition number is large probably due to the square term, increasing the statistic for multicollinearity. As seen the in the t statistics, all variables, including the coefficient is significant at the 0.1% significance level. This is reflected in the F statsitic of the regression, which suggests that the overall significance of the regression is significant at the 0.1% significance level as well. The R squared value of the regression is also very high at 0.978.

The AIC and BIC values of the full regression is lower than the ones in part 1h. This suggests that the full model is a superior model. The skew and Kurtosis is close to their ideal values of 0 and 3 respectively, and indicates that the errors are normally distributed.

[29]:
```
#Finding the mean squared error
X2=energy[["X","Feb",␣
 ↪"March","Apr","May","June","July","Aug","Sept","Oct","Nov",␣
 ↪"Dec","Time","Quad"]]
X2=np.asarray(X2)
X2=X2.reshape((828,14))
y_pred=regfull.predict(X2)
```

```
y_actual=np.asarray(energy["IPG2211A2N"])
mean_squared_error(y_actual,y_pred)
```

[29]: 22.17650152406903

The mean squared error of this regression is 22.177 (3 d.p.), and is relatively large. This could be due to the small amounts of variation/seasonality observed in the initial parts of the time series. However, given that the other summary statistics of the model holds strong, this model will remain decent at forecasting energy production, especially since seasonality seems to become stronger as time increases.

2e. Use the full model to forecast h-steps (at least 16) ahead. Your forecast should include the respective prediction interval.

[30]:
```
#Manually calculating the forecast error and prediction intervals and fitting␣
 ↪it into a dataframe with predictions.
df = pd.DataFrame()
RSS=mean_squared_error(y_actual,y_pred)*828
RSE=(RSS/regfull.df_resid)**0.5
meant=np.mean(energy["Time"])
sdt=np.std(energy["Time"])
meanq=np.mean(energy["Quad"])
sdq=np.std(energy["Quad"])
mfeb=np.mean(energy["Feb"])
sfeb=np.std(energy["Feb"])
mmarch=np.mean(energy["March"])
smarch=np.std(energy["March"])
mapril=np.mean(energy["Apr"])
sapril=np.std(energy["Apr"])
mmay=np.mean(energy["May"])
smay=np.std(energy["May"])
mjune=np.mean(energy["June"])
sjune=np.std(energy["June"])
mjuly=np.mean(energy["July"])
sjuly=np.std(energy["July"])
maug=np.mean(energy["Aug"])
saug=np.std(energy["Aug"])
msept=np.mean(energy["Sept"])
ssept=np.std(energy["Sept"])
moct=np.mean(energy["Oct"])
soct=np.std(energy["Oct"])
mnov=np.mean(energy["Nov"])
snov=np.std(energy["Nov"])
mdec=np.mean(energy["Dec"])
sdec=np.std(energy["Dec"])
for i in range(len(energy)+1,len(energy)+1+16):
    if i%12==2:
        feb=1
```

```
    else:
        feb=0
    if i%12==3:
        march=1
    else:
        march=0
    if i%12==4:
        april=1
    else:
        april=0
    if i%12==5:
        may=1
    else:
        may=0
    if i%12==6:
        june=1
    else:
        june=0
    if i%12==7:
        july=1
    else:
        july=0
    if i%12==8:
        aug=1
    else:
        aug=0
    if i%12==9:
        sept=1
    else:
        sept=0
    if i%12==10:
        oct=1
    else:
        oct=0
    if i%12==11:
        nov=1
    else:
        nov=0
    if i%12==0:
        dec=1
    else:
        dec=0
    predictions=regfull.
↪predict([[1,feb,march,april,may,june,july,aug,sept,oct,nov,dec,i,i**2]])
```

```
    f_e=(RSE)*(1+1/828+(i-meant)/((sdt**2)*(828-1))+(i**2-meanq)/
 ↪((sdq**2)*(828-1))+(feb-mfeb)/((sfeb**2)*(828-1))+(march-mmarch)/
 ↪((smarch**2)*(828-1))+(april-mapril)/((sapril**2)*(828-1))+(may-mmay)/
 ↪((smay**2)*(828-1))+(june-mjune)/((sjune**2)*(828-1))+(july-mjuly)/
 ↪((sjuly**2)*(828-1))+(aug-maug)/((saug**2)*(828-1))+(sept-msept)/
 ↪((ssept**2)*(828-1))+(oct-moct)/((soct**2)*(828-1))+(nov-mnov)/
 ↪((snov**2)*(828-1))+(dec-mdec)/((sdec**2)*(828-1)))**0.5
    p_iu=scipy.stats.t.ppf(1 - 0.05/ 2, regfull.df_resid)*f_e+regfull.
 ↪predict([[1,feb,march,april,may,june,july,aug,sept,oct,nov,dec,i,i**2]])
    p_il=-scipy.stats.t.ppf(1 - 0.05/ 2, regfull.df_resid)*f_e+regfull.
 ↪predict([[1,feb,march,april,may,june,july,aug,sept,oct,nov,dec,i,i**2]])
    df=df.append([[predictions,p_il,p_iu]],ignore_index=True)
df.rename(columns={0: 'Prediction',1:"Lower Prediction Interval",2:"Upper␣
 ↪Prediction Interval"},
          index={0:"2019-01-01",1:"2019-02-01",2:"2019-03-01",3:"2019-04-01",4:
 ↪"2019-05-01",5:"2019-06-01",6:"2019-07-01",7:"2019-08-01",8:"2019-09-01",9:
 ↪"2019-10-01",10:"2019-11-01",11:"2019-12-01",12:"2020-01-01",13:
 ↪"2020-02-01",14:"2020-03-01",15:"2020-04-01"})
```

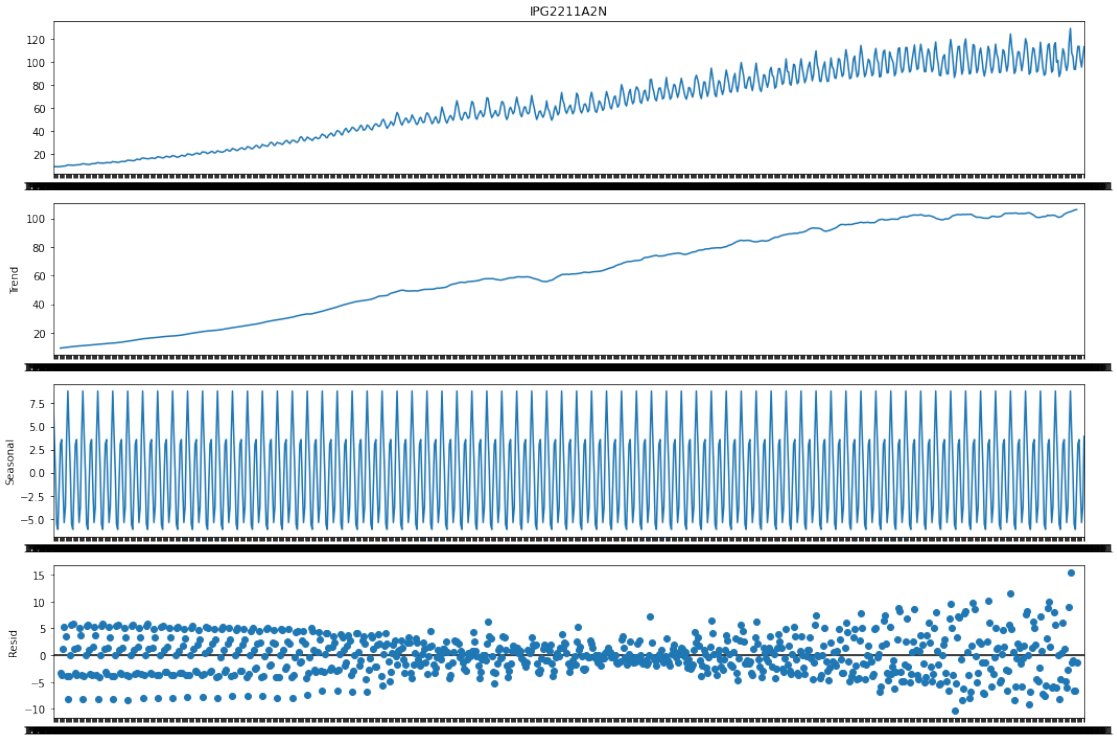
[30]:

	Prediction	Lower Prediction Interval	Upper Prediction Interval
2019-01-01	[119.08832687157727]	[109.8277563690124]	[128.34889737414213]
2019-02-01	[114.54957895162678]	[105.21502180534266]	[123.88413609791091]
2019-03-01	[110.67462813312733]	[101.34007088808738]	[120.00918537816727]
2019-04-01	[105.1093034015816]	[95.77474605778558]	[114.44386074537762]
2019-05-01	[104.63020475699413]	[95.29564731444175]	[113.96476219954651]
2019-06-01	[109.33395103994272]	[99.99939349863372]	[118.66850858125171]
2019-07-01	[113.99924369970252]	[104.66468605963664]	[123.3338013397684]
2019-08-01	[114.59994505511474]	[105.26538731629171]	[123.93450279393778]
2019-09-01	[109.55790583081688]	[100.22334799323643]	[118.89246366839733]
2019-10-01	[105.63454921521445]	[96.2999912788763]	
2019-11-01	[107.47134187497352]	[98.13678383987742]	
2019-12-01	[115.20000554922858]	[105.86544741537425]	
2020-01-01	[120.17834083217247]	[110.91776913505524]	
2020-02-01	[115.63848872734556]	[106.30393039597398]	
2020-03-01	[111.7624337239697]	[102.4278752938391]	
2020-04-01	[106.19600480754758]	[96.86144627865768]	

```
2019-10-01      [114.96910715155259]
2019-11-01      [116.80589991006961]
2019-12-01      [124.53456368308291]
2020-01-01       [129.4389125292897]
2020-02-01      [124.97304705871714]
2020-03-01      [121.09699215410032]
2020-04-01      [115.53056333643748]
```

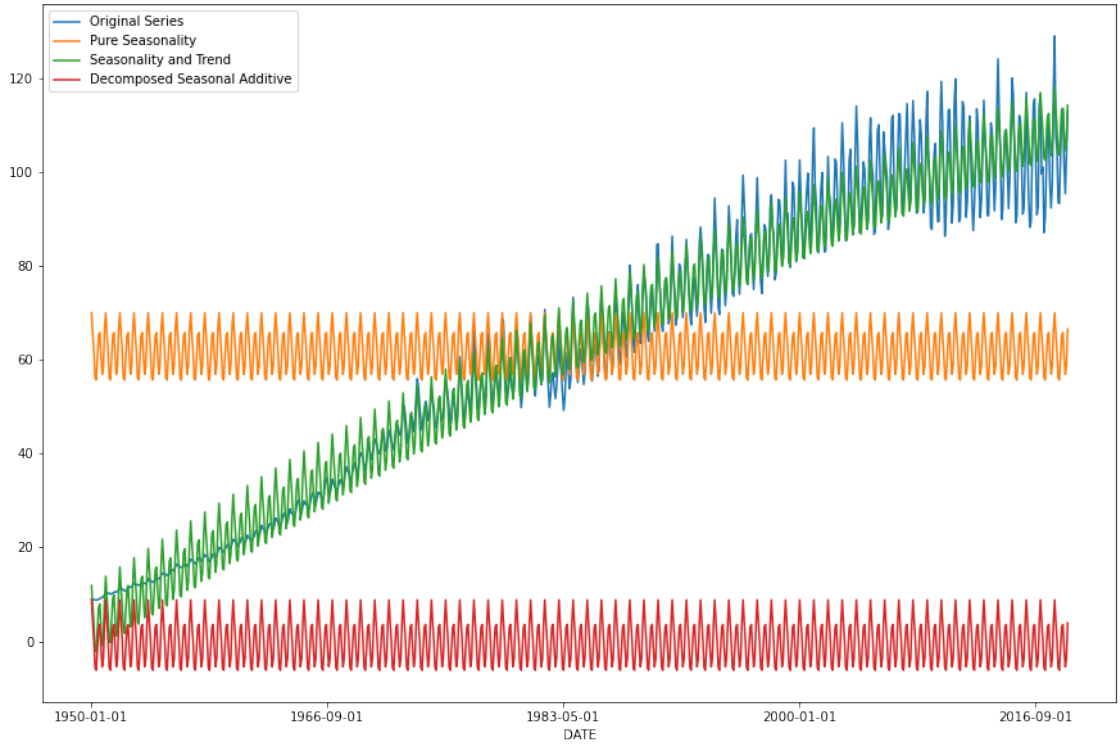
2f. Plot the STL decomposition plot, and based on it, choose between an additive and multiplicative seasonal adjustment. Perform the correction, and plot the seasonally adjusted series. Based on this adjustment, would your trend model from problem 1 still be appropriate? Explain your answer in detail.

[31]:
```
result = seasonal_decompose(energy.IPG2211A2N, model='additive', freq=12)␣
 ↪#Every cycle should end every 12 months
result.plot()
plt.show()
```



[32]:
```
energy.IPG2211A2N.plot(label="Original Series")
plt.plot(regseason.predict(),label="Pure Seasonality")
plt.plot(regfull.predict(), label = "Seasonality and Trend")
plt.plot(result.seasonal, label="Decomposed Seasonal Additive")
plt.legend(loc='best')
```

[32]: <matplotlib.legend.Legend at 0x1ff77086e80>



My predicted forecasts (Green series) are very similar with the actual data (Blue series) especially towards the end of the time series as the data in the beginning does not show strong trends of seasonality. As the adjustment is additive, this implies that my trend model from problem 1 is still appropriate, especially given how well fit the combined dataset is to the actual data.

Section III. Conclusions and Future Work (state your conclusion regarding your final model and forecast, and provide some insight as to how it could be improved).

Analysis of this time series found a quadratic trend relationship between energy production and time, and W shaped seasonality with peaks during the summer and winter. This matches with the expectations, as energy consumption and hence production is expected to increase during winter and summer due to the use of heating and air conditioning respectively in most common households. The final model forecast seems to match really well with the actual data in the series, though more accurate forecasts could potentially be made if a full ARIMA model is used to forecast the time series and account for the initial low seasonality observed in the data.

Section IV. References

- Board of Governors of the Federal Reserve System (US), Industrial Production: Electric and gas utilities [IPG2211A2N], retrieved from Kaggle; https://www.kaggle.com/sadeght/industrial-production-electric-and-gas-utilities, Nov 28, 2020.
- U.S Energy Information Administration, "US Energy Facts Explained", retrieved online; https://www.eia.gov/energyexplained/us-energy-facts/data-and-statistics.php, Dec 1, 2020.