DDFA module 8, capstone option #3

For your last module, you're going to be building your own model, from scratch! But don't worry, you'll be getting a *little* bit of help: you have three options of datasets to choose from (plus, you can of course always ask any questions of the TA).

Option 3: Time series ARIMA modeling using hourly energy consumption

If you choose this option, you will be building an ARIMA time-series model. We'll be using publicly available data from Kaggle (a data science competition website with freely available datasets, and even some competitions for money! Check it out).

The data is on hourly energy consumption and has been provided by a regional utility company in the United States.

The dataset

You'll be using a dataset of hourly energy usage from an energy company in the midwest. You can read more about the dataset here. We have provided you the dataset for ComEd, the energy provider here in Chicago. If you'd like to test your skills, you can download datasets for the other providers from the Kaggle site.

The data is provided at an hourly frequency. The frequency of the model you build is up to you. You may want to try building several models, each at a different frequency. For example, you can roll up the hourly data to the average daily data or the average monthly usage data. What does the monthly time series look like? What patterns do you see?

Your assignment

You should import the csv dataset mentioned above. You'll then want to carry out an entire end-to-end data science workflow. This includes:

- Import the data
- Explore the data using summary tables and charts/graphs (use `matplotlib` and `Pandas`)
- Try plotting the energy consumption data at different frequencies: data is provided at an hourly frequency, but try rolling up to daily or monthly average; what patterns do you see?
- Apply at least one machine learning algorithm to the dataset (e.g. ARIMA)

- Check how well your model is working; how do we do this for time-series models? Try a train/test split

Hints

Your mission, should you choose to accept it, is to use the provided dataset to build an ARIMA time-series forecasting model. Here are some things you should know:

- There is no outcome/dependent variable as this is a time-series task
- You're welcome to use either `statsmodels` or to try Facebook Prophet. Remember, we didn't learn Facebook Prophet code, but we did cover the package briefly during webinar 2. You should be able to use what you know now to follow along with the above tutorial, but apply it to our energy data instead
- Kaggle allows users to post their solution notebooks to the site for each dataset, and it's perfectly fine to check out others' solutions for inspiration as you work on your own solution. Of course, you can't just copy/paste others' work and claim it as your own! Don't cheat, but you can take hints if you're stuck

Get started!

In [204…

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
```

Retrieving the data from excel

In [207…

```
# Read the csv file and recast the index to datetime format

comed_df = pd.read_csv('COMED_hourly.csv', infer_datetime_format=True, header=0, in

comed_df.index = pd.to_datetime(comed_df.index)
```

```
/tmp/ipykernel_553/2449921600.py:3: FutureWarning: The argument 'infer_datetime_form
at' is deprecated and will be removed in a future version. A strict version of it is
now the default, see https://pandas.pydata.org/pdeps/0004-consistent-to-datetime-par
sing.html. You can safely remove this argument.
  comed_df = pd.read_csv('COMED_hourly.csv', infer_datetime_format=True, header=0, i
ndex_col=0, names=['Datetime', 'COMED_MW'])
```

Exploratory Data Analysis

Summary Statistics

First I am running Summary Statistics, to get an understanding of any errors in the data, column types, and distribution pulling means, std. deviation, maximums and minimums trying to spot problems.

In [210…
```
# Ran Describe() to give me an idea of the distribution of the dataset.
comed_df.describe()
```

Out[210…

	COMED_MW
count	66497.000000
mean	11420.152112
std	2304.139517
min	7237.000000
25%	9780.000000
50%	11152.000000
75%	12510.000000
max	23753.000000

In [212…
```
# Shape provides the structure of the dataset, two columns 66K rows.
comed_df.shape
```

Out[212…
```
(66497, 1)
```

In [214…
```
# Here info() shows the data type, we have a float.
comed_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 66497 entries, 2011-12-31 01:00:00 to 2018-01-02 00:00:00
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   COMED_MW  66497 non-null  float64
dtypes: float64(1)
memory usage: 1.0 MB
```

In [216…
```
# I am trying to spot any missing data or NaNs using isnull()
comed_df.isnull().sum()
```

Out[216…
```
COMED_MW    0
dtype: int64
```

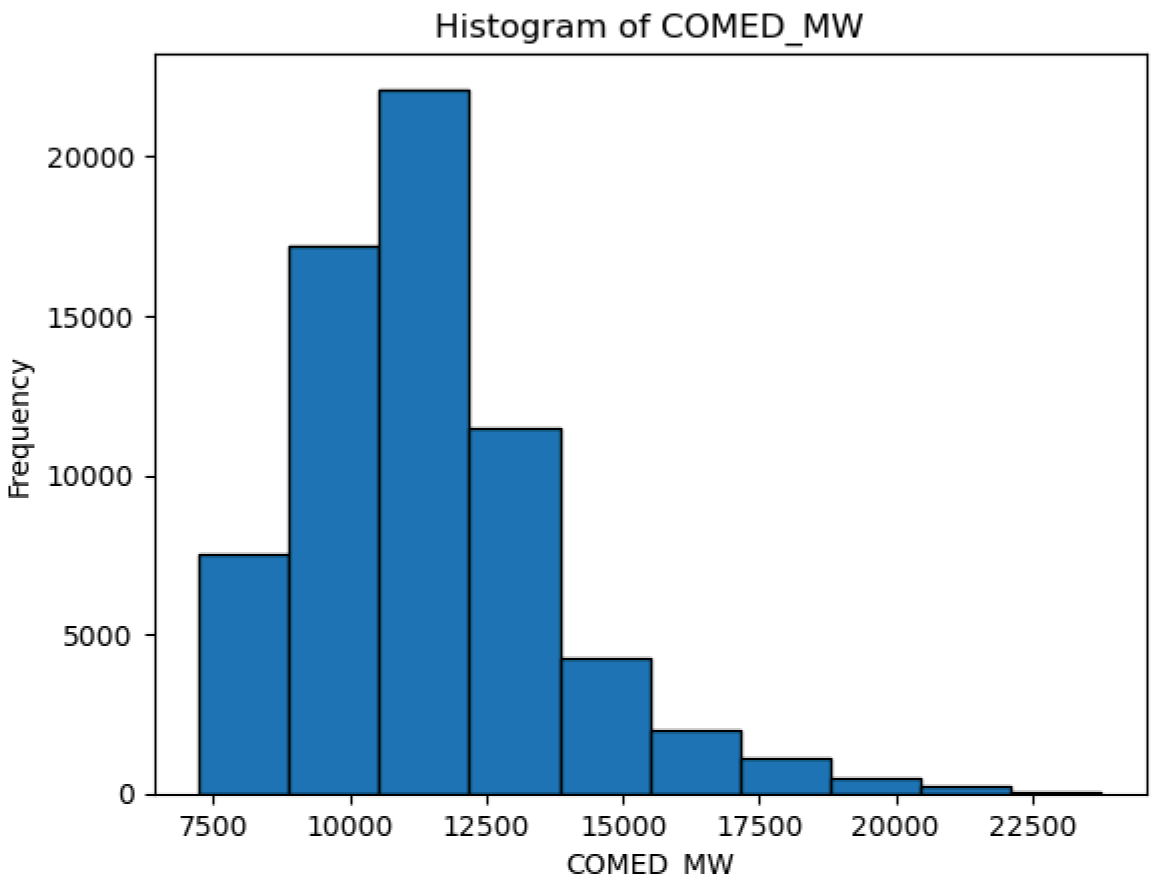
Visualizations

Now I am ploting data with plotly. This will help me visually understand the characteristic of the dataset.

Histogram

In [220…

```
# Create a histogram of the 'COMED_MW' column
plt.hist(comed_df['COMED_MW'], bins=10, edgecolor='black')

# Add title and labels
plt.title('Histogram of COMED_MW')
plt.xlabel('COMED_MW')
plt.ylabel('Frequency')

# Show the plot
plt.show()
```



In [221…

```
from scipy.stats import skew, kurtosis

# Calculate skewness
skewness = skew(comed_df['COMED_MW'])

# Calculate kurtosis
kurt = kurtosis(comed_df['COMED_MW'])

print(f'Skewness: {skewness}')
print(f'Kurtosis: {kurt}')
```

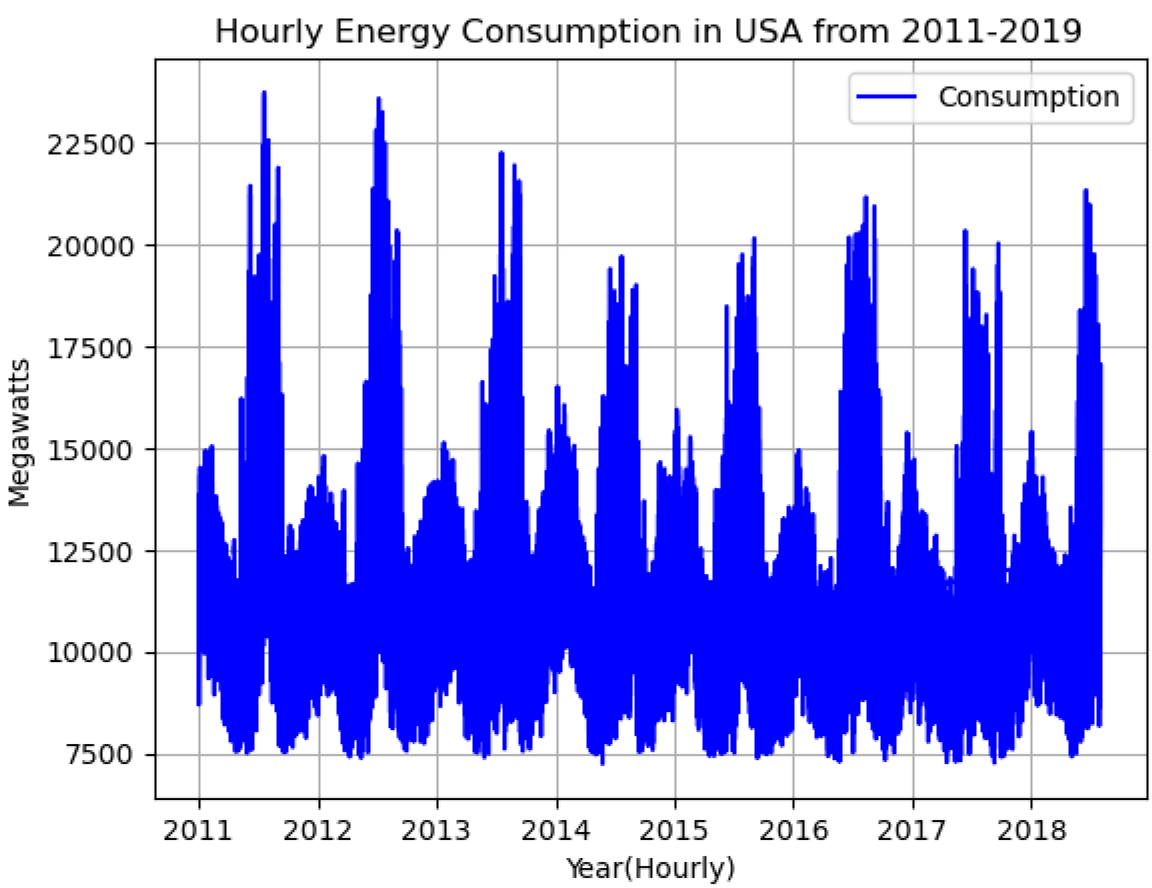
```
Skewness: 1.1618202009869127
Kurtosis: 2.1668230508395414
```

- The dataset has a Right or positive skew, meaning that there are very high values in our data pulling the mean upward.

Line Charts

In [226…

```
# Plotted the hourly data
plt.plot(comed_df.index, comed_df['COMED_MW'], label='Consumption', color='blue')

plt.title('Hourly Energy Consumption in USA from 2011-2019')
plt.xlabel('Year(Hourly)')
plt.ylabel('Megawatts')
plt.legend()
plt.grid()
plt.show()
```



- It seems like daily is too much data to run ARIMA with, data is to bulked together though not stationary.
- I think this is because the dataset contains hours.

In [229…

```
comed_df.head(24)
```

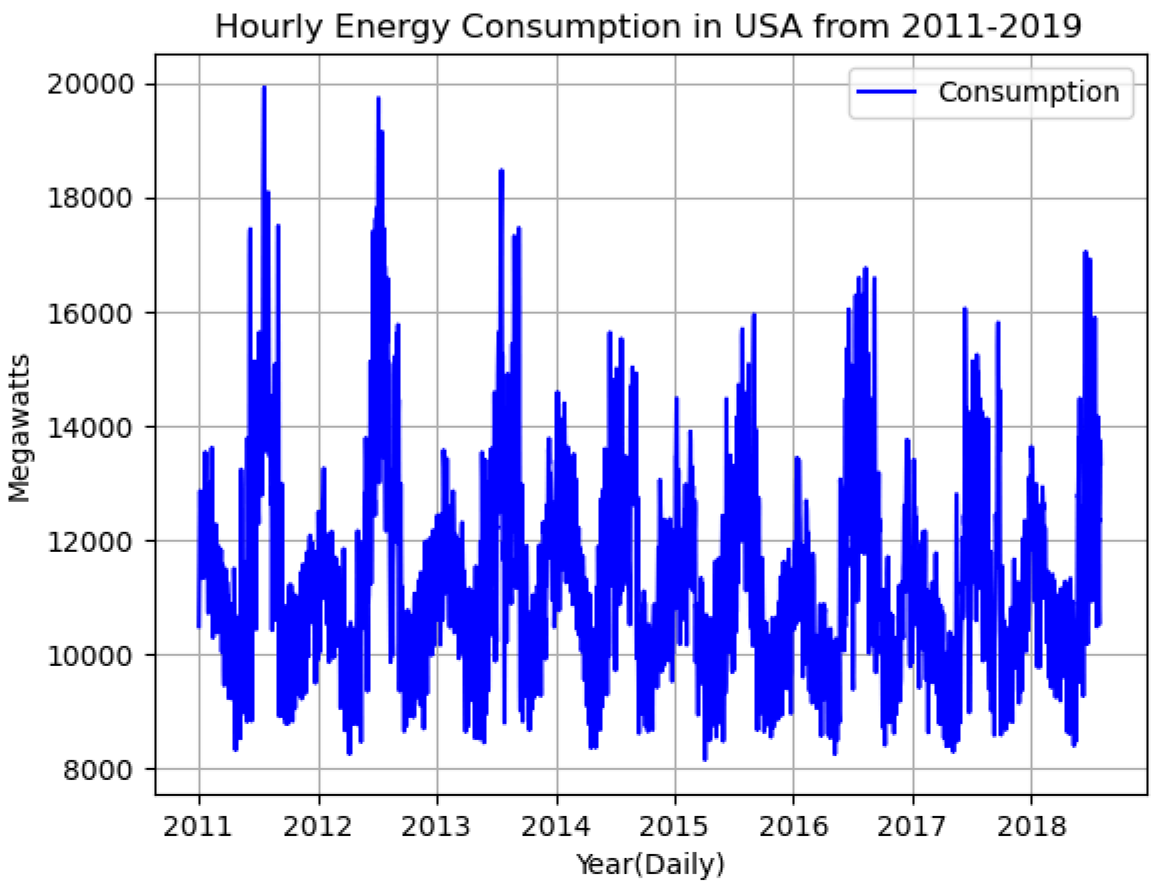
Out[229…

	COMED_MW
Datetime	
2011-12-31 01:00:00	9970.0
2011-12-31 02:00:00	9428.0
2011-12-31 03:00:00	9059.0
2011-12-31 04:00:00	8817.0
2011-12-31 05:00:00	8743.0
2011-12-31 06:00:00	8735.0
2011-12-31 07:00:00	8993.0
2011-12-31 08:00:00	9363.0
2011-12-31 09:00:00	9545.0
2011-12-31 10:00:00	9676.0
2011-12-31 11:00:00	9937.0
2011-12-31 12:00:00	10139.0
2011-12-31 13:00:00	10326.0
2011-12-31 14:00:00	10359.0
2011-12-31 15:00:00	10293.0
2011-12-31 16:00:00	10240.0
2011-12-31 17:00:00	10416.0
2011-12-31 18:00:00	11225.0
2011-12-31 19:00:00	11907.0
2011-12-31 20:00:00	11812.0
2011-12-31 21:00:00	11542.0
2011-12-31 22:00:00	11149.0
2011-12-31 23:00:00	10855.0
2012-01-01 00:00:00	10335.0

In [231…

```
# TODO create a new dataframe with the data at a daily frequency
comed_df_daily = comed_df.resample('D').mean()

# Plotted the hourly data
plt.plot(comed_df_daily.index, comed_df_daily['COMED_MW'], label='Consumption', col

plt.title('Hourly Energy Consumption in USA from 2011-2019')
plt.xlabel('Year(Daily)')
```

```
plt.ylabel('Megawatts')
plt.legend()
plt.grid()
plt.show()
```



In [232…

```
comed_df_daily.head(6)
```

Out[232…

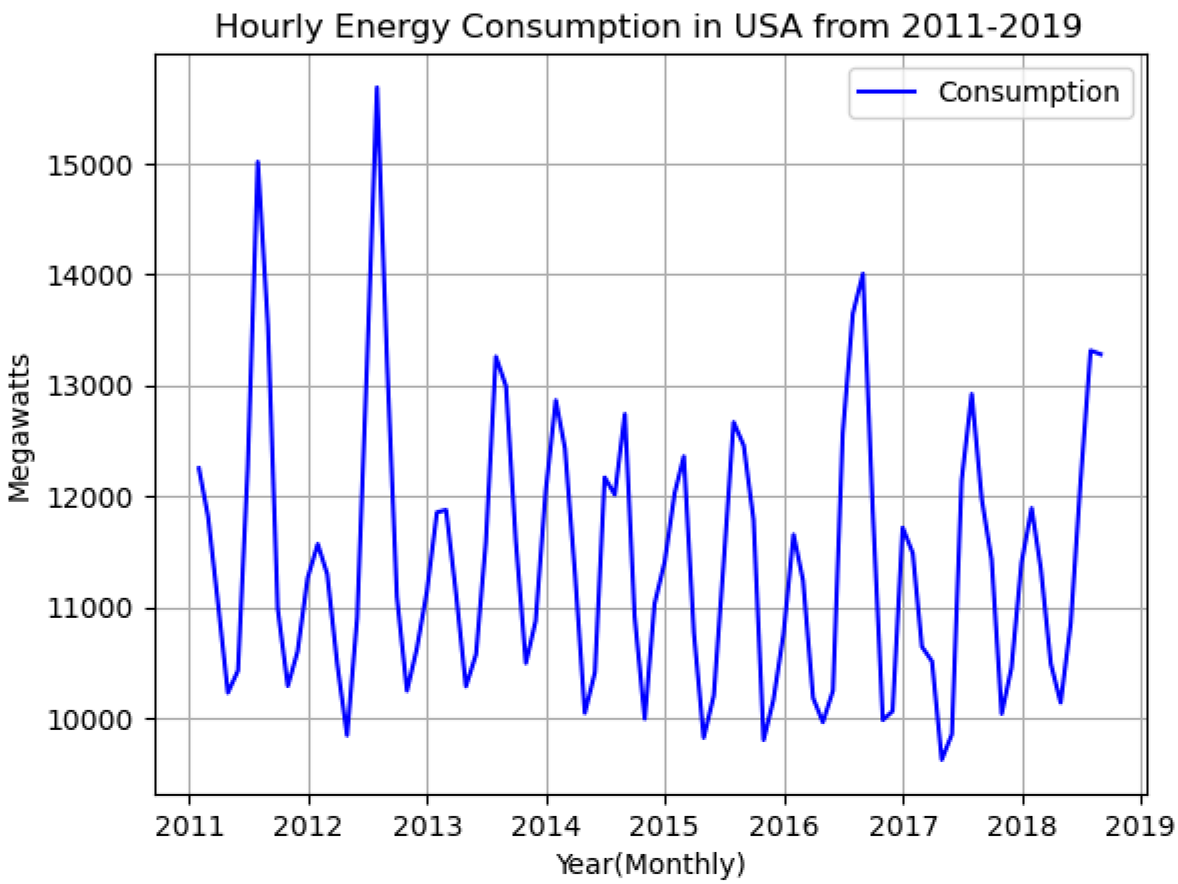
Datetime	COMED_MW
2011-01-01	10485.304348
2011-01-02	11255.708333
2011-01-03	12117.625000
2011-01-04	12499.750000
2011-01-05	12855.166667
2011-01-06	12572.250000

- Daily data provides a good spread, but it might be easier to plot a trend line on monthly data. We do see volatility, thus I don't think variance is constant (homoscedasticity).

In [236…

```
# TODO create a new dataframe with the data at a monthly frequency
comed_df_monthly = comed_df.resample('M').mean()
```

```
# TODO plot the monthly data
plt.plot(comed_df_monthly.index, comed_df_monthly['COMED_MW'], label='Consumption',

plt.title('Hourly Energy Consumption in USA from 2011-2019')
plt.xlabel('Year(Monthly)')
plt.ylabel('Megawatts')
plt.legend()
plt.grid()
plt.show()
```



- Monthly provides a nicer spread.

In [239…

```
comed_df_monthly.head()
```

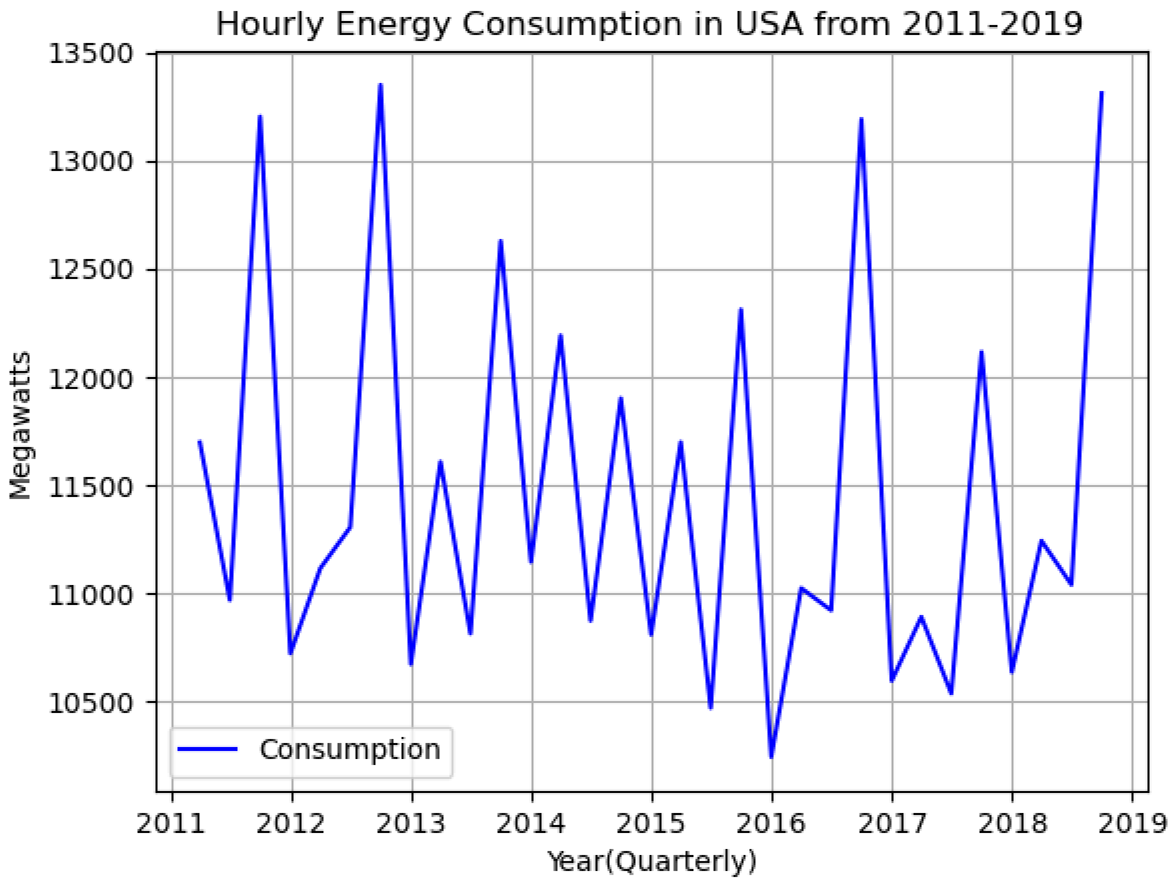
Out[239…

	COMED_MW
Datetime	
2011-01-31	12252.029610
2011-02-28	11820.212798
2011-03-31	11028.816958
2011-04-30	10229.681944
2011-05-31	10429.517473

- I guess the only problem with monthly data is interpretability. We will come to the customer telling them that we can project the monthly average consumption per month, but the average is for 1 hour.
- I think it would be better to sum up the values into total MW consumption per month to see if we can project total consumption of every month. That is more intuitive.

In [242…

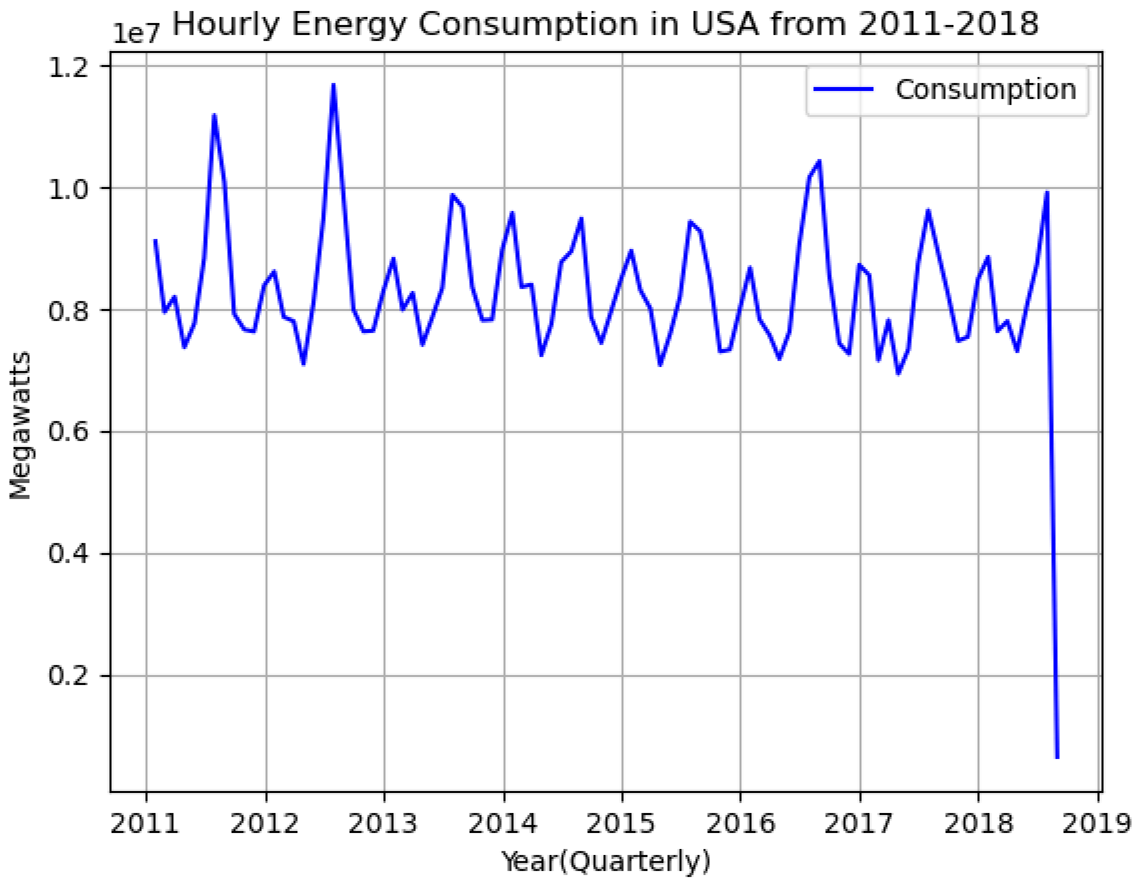
```
# TODO create a new dataframe with the data at a monthly frequency
comed_df_quarterly = comed_df.resample('Q').mean()

# TODO plot the monthly data
plt.plot(comed_df_quarterly.index, comed_df_quarterly['COMED_MW'], label='Consumpti

plt.title('Hourly Energy Consumption in USA from 2011-2019')
plt.xlabel('Year(Quarterly)')
plt.ylabel('Megawatts')
plt.legend()
plt.grid()
plt.show()
```



- When we see it quarterly we see a lot of volatility. I think we summarized too much, I would stay monthly.

In [244…

```
# Now let's try the monthly total
monthly_consumption = comed_df.resample('M').sum()

# Plot the monthly data
plt.plot(monthly_consumption.index, monthly_consumption['COMED_MW'], label='Consump

plt.title('Hourly Energy Consumption in USA from 2011-2018')
plt.xlabel('Year(Quarterly)')
plt.ylabel('Megawatts')
plt.legend()
plt.grid()
plt.show()
```



In [245…

```
monthly_consumption.tail(10)
```

Out[245…

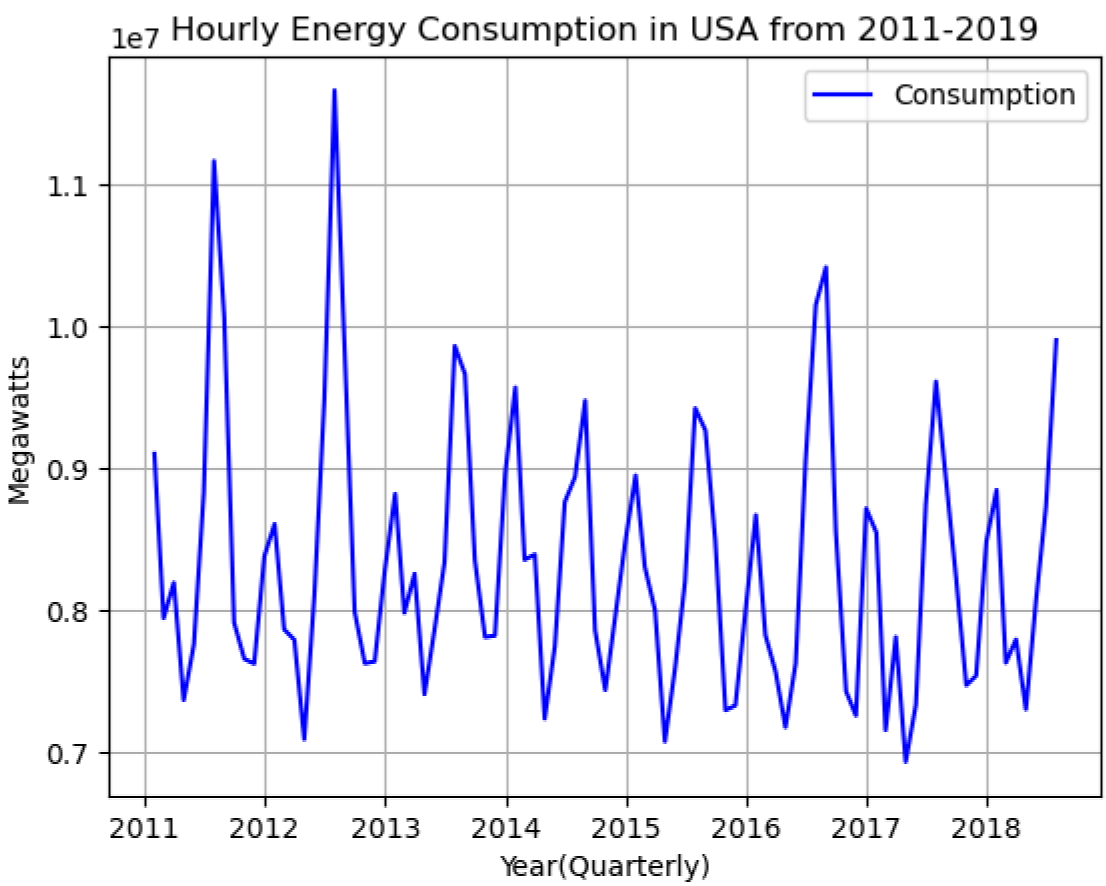
	COMED_MW
Datetime	
2017-11-30	7538269.0
2017-12-31	8483690.0
2018-01-31	8847943.0
2018-02-28	7628265.0
2018-03-31	7794444.0
2018-04-30	7301136.0
2018-05-31	8074566.0
2018-06-30	8731995.0
2018-07-31	9904590.0
2018-08-31	650666.0

- The last month doesn't seem to be complete let's remove it.

In [248…
```
monthly_consumption_comp = monthly_consumption[monthly_consumption.index <= '2018-0

print(monthly_consumption_comp.tail())
```

```
              COMED_MW
Datetime
2018-03-31  7794444.0
2018-04-30  7301136.0
2018-05-31  8074566.0
2018-06-30  8731995.0
2018-07-31  9904590.0
```

In [252…
```
# Let's plot again.
plt.plot(monthly_consumption_comp.index, monthly_consumption_comp['COMED_MW'], labe

plt.title('Hourly Energy Consumption in USA from 2011-2019')
plt.xlabel('Year(Quarterly)')
plt.ylabel('Megawatts')
plt.legend()
plt.grid()
plt.show()
```



- The shape did change it looks now as if it always come down to ,7 at some point around February or March, probably because we are changing seasons, and people don't use ACs during that period probably.
- What I don't like is that I am picking up seasonaility as I transformed into Monthly Totals

Decomposition

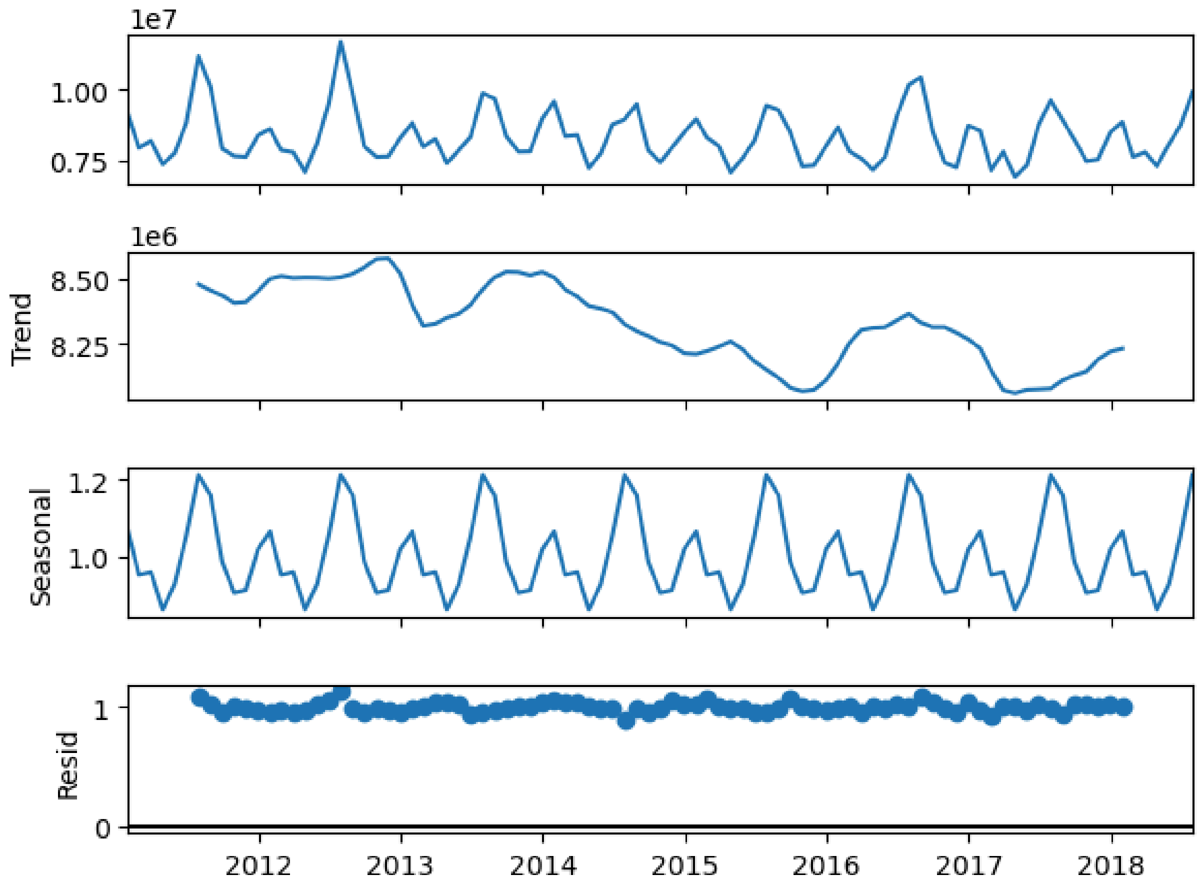
First with Monthly Totals Data

In [255…

```
# import
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [258…

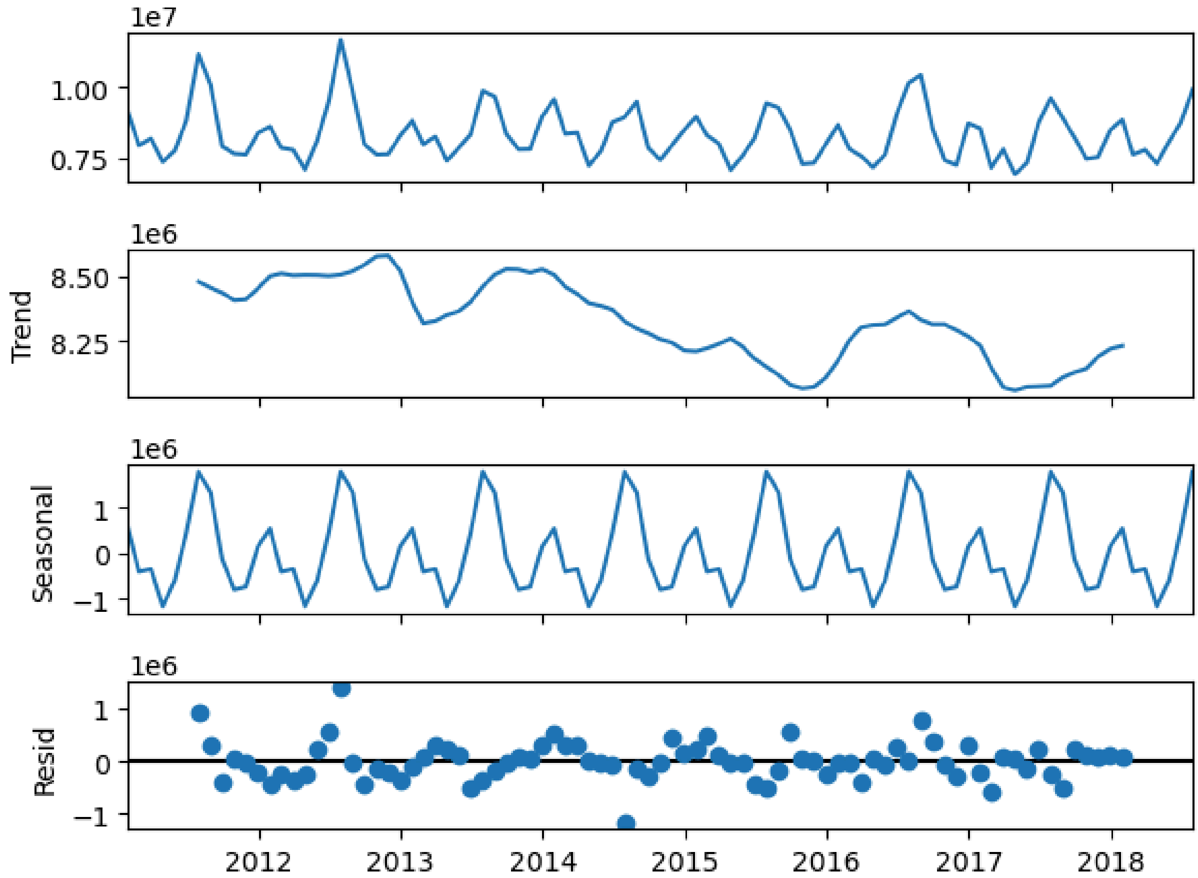
```
#Performing a multiplicative decomposition
# With the monthly data
# I assign the plot to a variable so it does not print twice
monthly_sum_decomposition = seasonal_decompose(monthly_consumption_comp, model="mul
fig = monthly_sum_decomposition.plot()
```



- A mean residual close to 1 in multiplicative is good
- I can see a trend though, so it needs to be differentiated

In [260…

```
#Performing an additive decomposition
# With the monthly data
# I assign the plot to a variable so it does not print twice
monthly_sum_decomposition2 = seasonal_decompose(monthly_consumption_comp, model="ad
fig = monthly_sum_decomposition2.plot()
```



- A mean residual close to 0 in additive is good
- I can see a trend though, so it needs to be differentiated

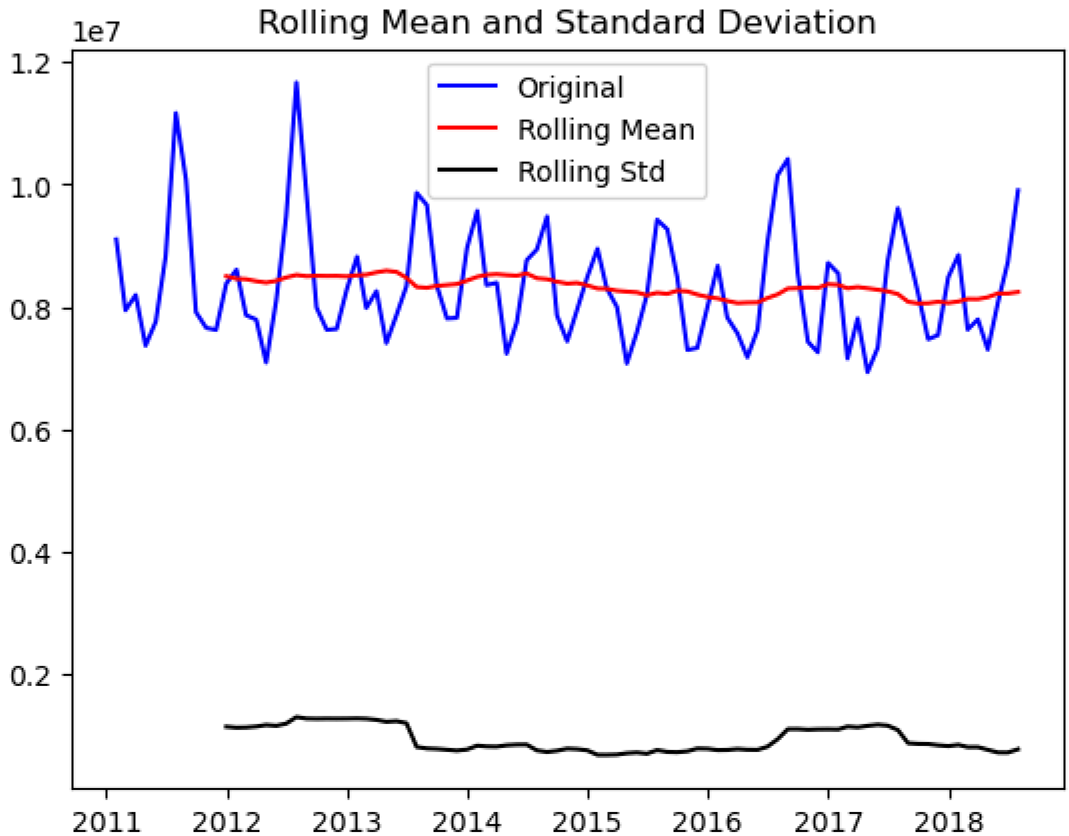
In [262…

```
# define ADF test function
import statsmodels.tsa.stattools as ts
def dftest(timeseries):
    dftest = ts.adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','Lags Used'
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
    #Determine rolling statistics
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, 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 and Standard Deviation')
    plt.show(block=False)
```

In [263…

```
dftest(monthly_consumption_comp)
```

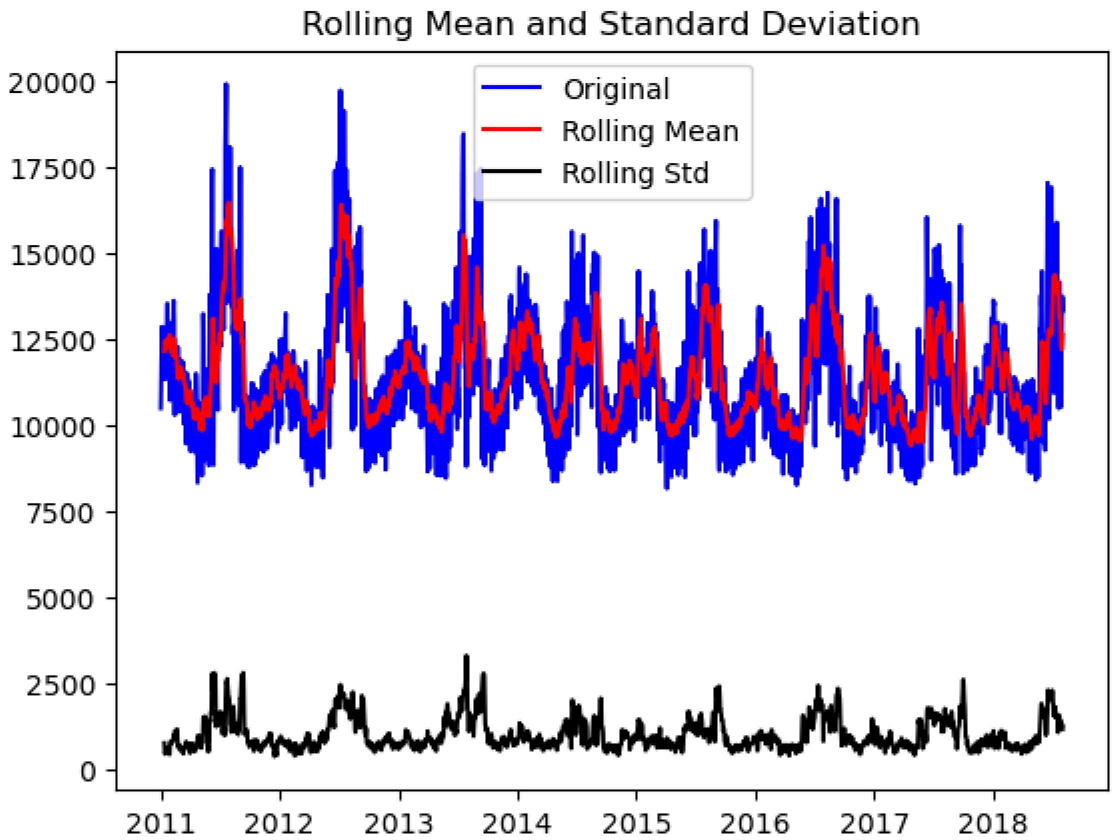
```
Test Statistic          -1.774090
p-value                  0.393399
Lags Used               12.000000
Observations Used       78.000000
Critical Value (1%)     -3.517114
Critical Value (5%)     -2.899375
Critical Value (10%)    -2.586955
dtype: float64
```



In [264…

```
dftest(comed_df_daily)
```

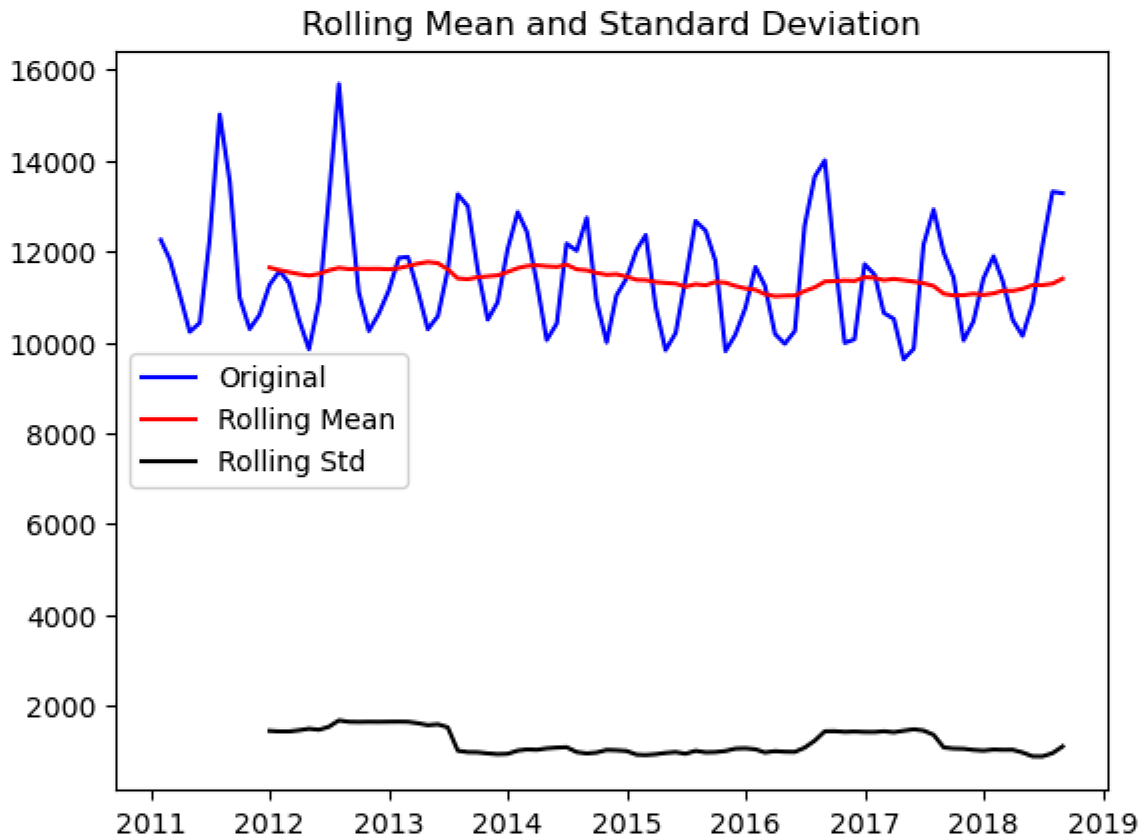
```
Test Statistic         -5.825466e+00
p-value                 4.089167e-07
Lags Used               2.800000e+01
Observations Used       2.743000e+03
Critical Value (1%)    -3.432736e+00
Critical Value (5%)    -2.862594e+00
Critical Value (10%)   -2.567331e+00
dtype: float64
```



In [265…

```
dftest(comed_df_monthly)
```

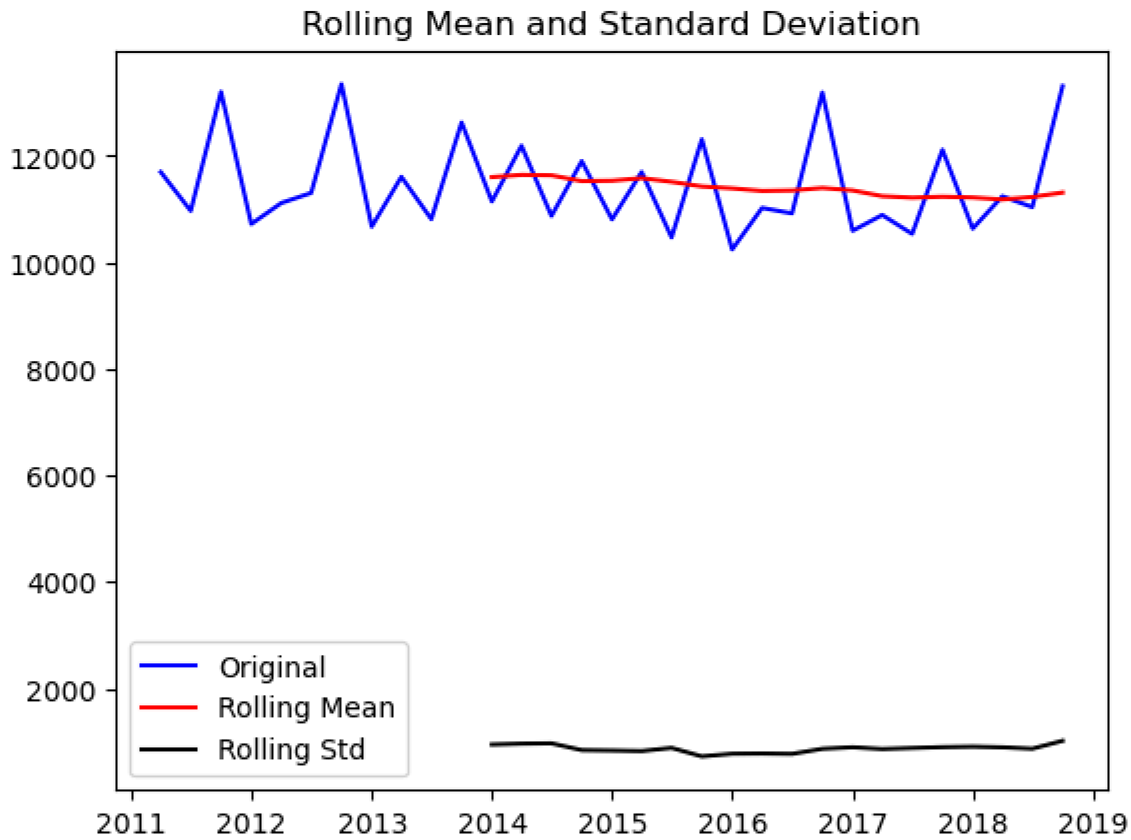
```
Test Statistic          -1.785990
p-value                  0.387439
Lags Used               11.000000
Observations Used       80.000000
Critical Value (1%)     -3.514869
Critical Value (5%)     -2.898409
Critical Value (10%)    -2.586439
dtype: float64
```



In [266…

```
dftest(comed_df_quarterly)
```

```
Test Statistic          -2.015695
p-value                  0.279734
Lags Used                4.000000
Observations Used       26.000000
Critical Value (1%)     -3.711212
Critical Value (5%)     -2.981247
Critical Value (10%)    -2.630095
dtype: float64
```



- We need to use differencing in all scenarios since the data is stationary.
- The test failed to acheive a p-value of less that .05 so we failed to reject the null hypthesis of non-stationarity.

Differencing

First the monthly average consumption time series.

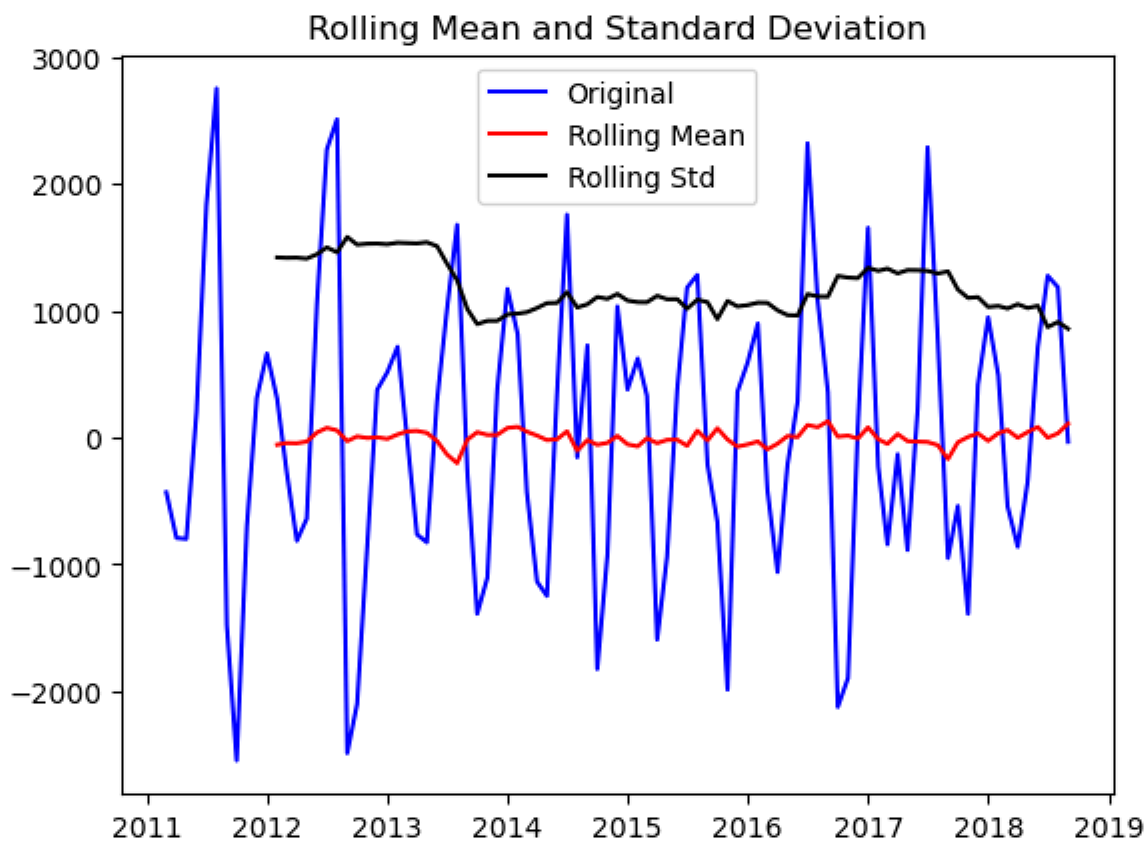
In [270…

```
# todo: create new df with monthly data with a first difference (use .diff())
# remember to drop first row (see code hint above)
comed_df_monthly_diff = comed_df_monthly.diff()[1:]
```

In [272…

```
# todo: check whether monthly differenced series is stationary
dftest(comed_df_monthly_diff)
```

```
Test Statistic          -7.708800e+00
p-value                  1.280200e-11
Lags Used                1.000000e+01
Observations Used        8.000000e+01
Critical Value (1%)     -3.514869e+00
Critical Value (5%)     -2.898409e+00
Critical Value (10%)    -2.586439e+00
dtype: float64
```

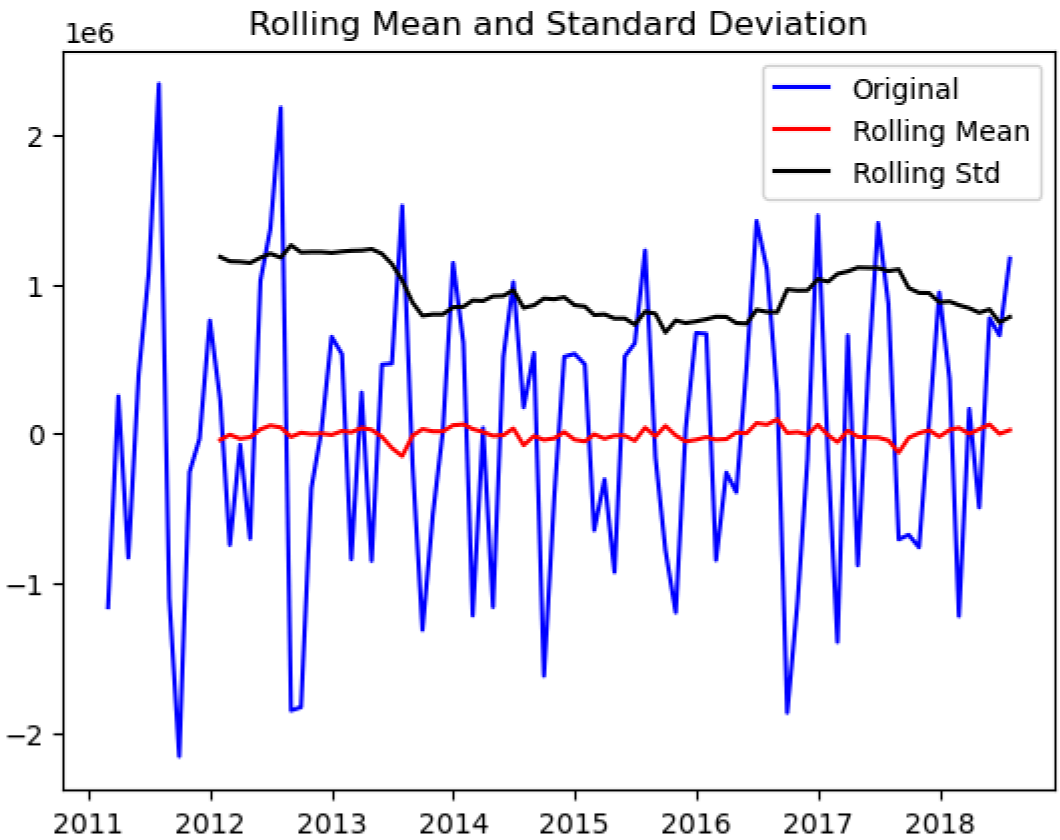


Second the monthly sum total time series.

In [274…
```
# todo: create new df with monthly data with a first difference (use .diff())
# remember to drop first row (see code hint above)
monthly_consumption_comp_diff = monthly_consumption_comp.diff()[1:]
```

In [275…
```
# todo: check whether monthly differenced series is stationary
dftest(monthly_consumption_comp_diff)
```

```
Test Statistic          -5.072519
p-value                  0.000016
Lags Used               11.000000
Observations Used       78.000000
Critical Value (1%)     -3.517114
Critical Value (5%)     -2.899375
Critical Value (10%)    -2.586955
dtype: float64
```



- Now they are both looking much better. They also look very similar even though one is taking monthly averages and the other making a month total.
- Oh but note that monthly averages didn't pass the Dickey Fuller test.

ACF and PACF Plots

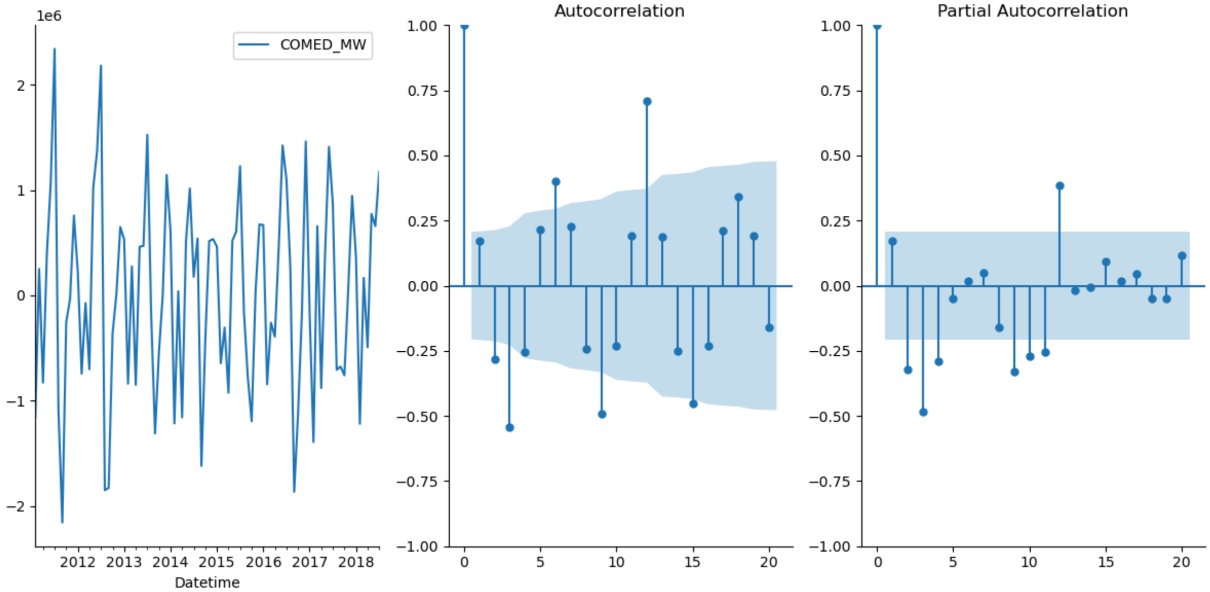
In [278…

```
import statsmodels.tsa.api as smt
import seaborn as sns

def acf_pacf_plots(data, lags=None):
    # create the structure for the plots
    layout = (1, 3)
    raw  = plt.subplot2grid(layout, (0, 0))
    acf  = plt.subplot2grid(layout, (0, 1))
    pacf = plt.subplot2grid(layout, (0, 2))

    # create the actual plots
    data.plot(ax=raw, figsize=(12, 6))
    smt.graphics.plot_acf(data, lags=lags, ax=acf)
    smt.graphics.plot_pacf(data, lags=lags, ax=pacf)
    sns.despine()
    plt.tight_layout()
```

In [279…

```
# Run the ACF and PACF plots of our monthly differenced data
# (remember to use the differenced data, as that is what's stationary!)
```

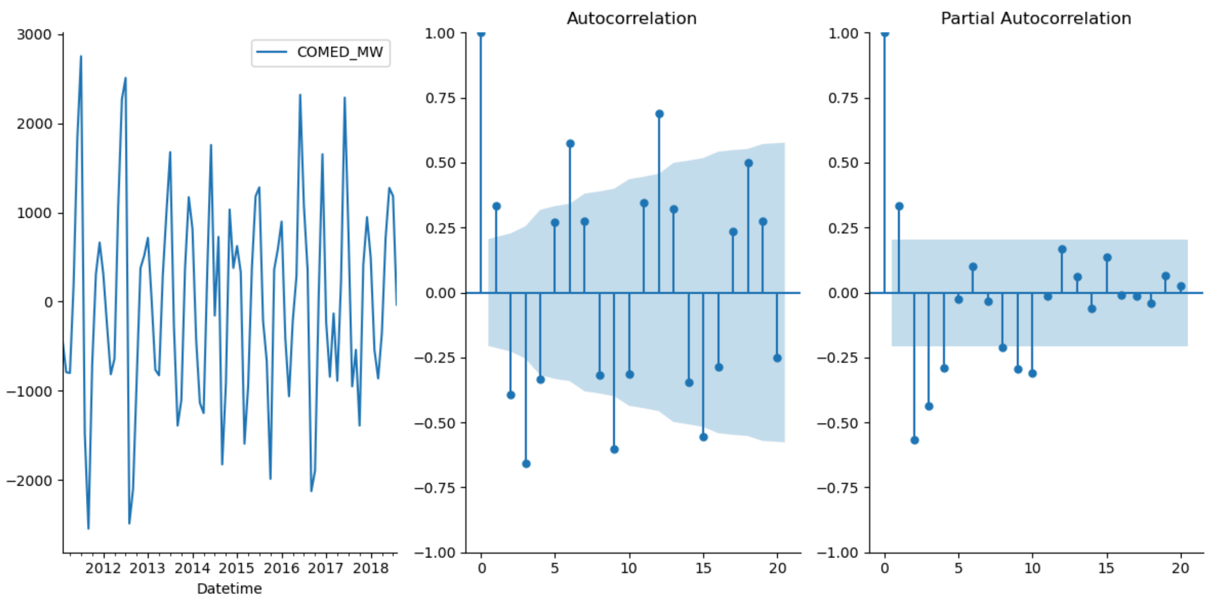
```
# use the helper function above (acf_pacf_plots())
acf_pacf_plots(monthly_consumption_comp_diff)
```



- Based on the PACF Plot, there are 5 lags over the blue area, we are diferencing 1 time. This is for the monthly total consumption.

In [283…

```
# Run the ACF and PACF plots of our monthly differenced data
# (remember to use the differenced data, as that is what's stationary!)
# use the helper function above (acf_pacf_plots())
acf_pacf_plots(comed_df_monthly_diff)
```



- Based on the PACF Plot, there are 5 lags over the blue area, we are diferencing 1 time. This is for the monthly average hourly consumption.

ARIMA Model for Average Monthly Hour Energy Consumption

In [287…

```
# import
from statsmodels.tsa.arima.model import ARIMA
```

In [288…

```
# Using ARIMA to fit an AR model
AR = ARIMA(comed_df_monthly.COMED_MW,order=(5,1,0)).fit()

AR.summary()
```

Out[288…

SARIMAX Results

Dep. Variable:	COMED_MW	**No. Observations:**	92
Model:	ARIMA(5, 1, 0)	**Log Likelihood**	-737.839
Date:	Sat, 16 Nov 2024	**AIC**	1487.678
Time:	07:18:14	**BIC**	1502.743
Sample:	01-31-2011	**HQIC**	1493.756
	- 08-31-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0327	0.068	0.481	0.631	-0.101	0.166
ar.L2	-0.3514	0.079	-4.423	0.000	-0.507	-0.196
ar.L3	-0.3202	0.089	-3.598	0.000	-0.495	-0.146
ar.L4	-0.2764	0.074	-3.736	0.000	-0.421	-0.131
ar.L5	0.0248	0.091	0.272	0.786	-0.154	0.204
sigma2	5.63e+05	7.33e+04	7.678	0.000	4.19e+05	7.07e+05

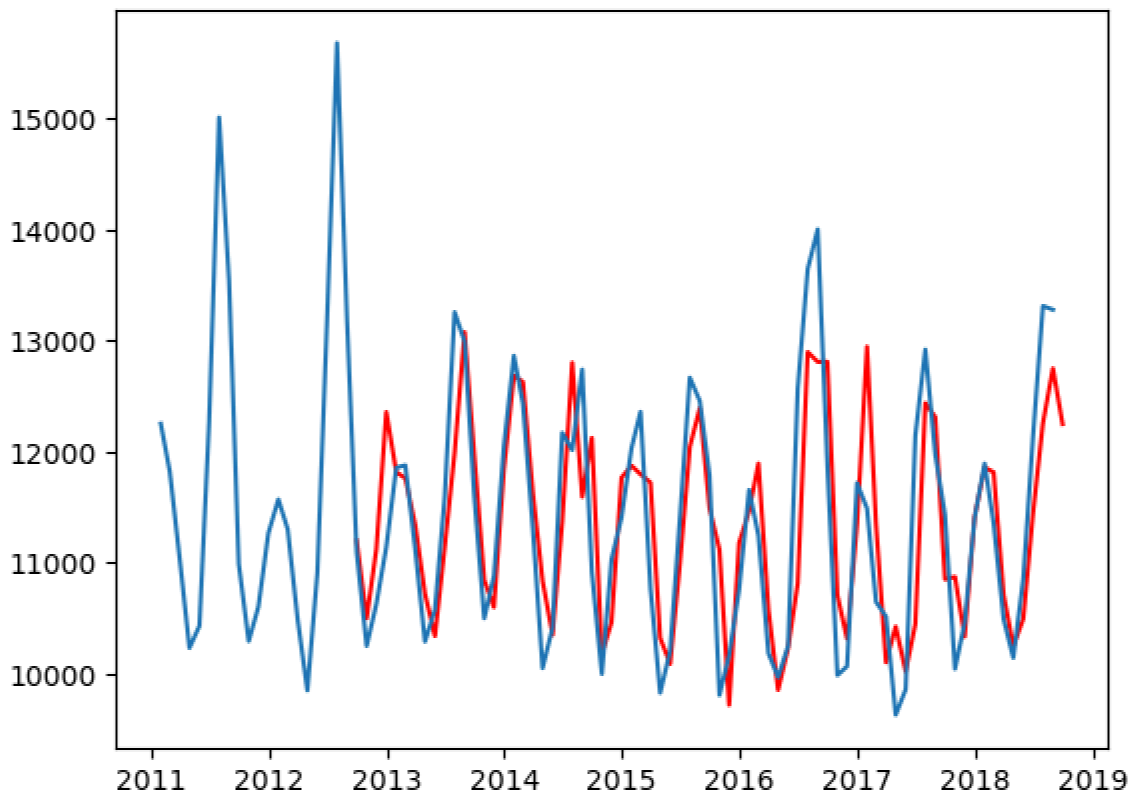
Ljung-Box (L1) (Q):	0.85	**Jarque-Bera (JB):**	4.87
Prob(Q):	0.36	**Prob(JB):**	0.09
Heteroskedasticity (H):	0.55	**Skew:**	0.53
Prob(H) (two-sided):	0.11	**Kurtosis:**	3.39

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [289…

```
# Created the in_sample series using AR.predict
in_sample = AR.predict(start=20, end=len(comed_df_monthly), dynamic=False)

# Plotted the actual and in-sample prediction on the same plot
plt.plot(in_sample, color='red')

plt.plot(comed_df_monthly);
```

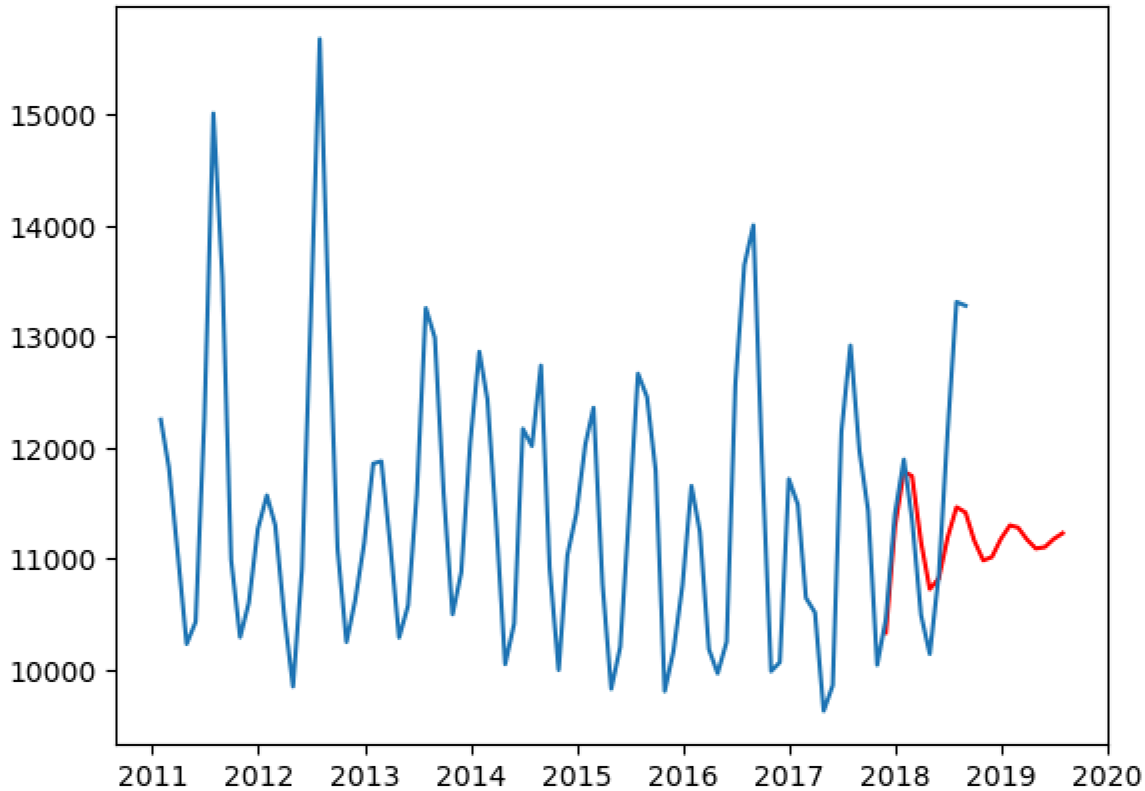


In [291…

```
# todo: out-of-sample prediction series using AR.predict
out_of_sample = AR.predict(start=len(comed_df_monthly)-10, end=len(comed_df_monthly
```

In [292…

```
# todo: plot the actual and out-of-sample on the same plot
# use a different color for out of sample so you can tell them apart (for example,

# todo: plot the actual and in-sample prediction on the same plot
plt.plot(out_of_sample, color='red')

plt.plot(comed_df_monthly);
```



In [294…

```
# import
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [297…

```
comed_df_monthly.shape
```

Out[297…

```
(92, 1)
```

In [298…

```
92*.3
```

Out[298…

```
27.599999999999998
```

In [301…

```
# todo: create train and test series
test = comed_df_monthly[:28]
train = comed_df_monthly[28:]
print(len(train))
print(len(test))
```

```
64
28
```

In [302…

```
# todo: use ARIMA().fit() to fit on the TRAIN dataset only
# remember, use MONTHLY data and use I = 1 to use a first-difference, aka stationar

AR2 = ARIMA(train,order=(5,1,0)).fit()

print(AR.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                COMED_MW   No. Observations:                   92
Model:                  ARIMA(5, 1, 0)   Log Likelihood                -737.839
Date:                 Sat, 16 Nov 2024   AIC                           1487.678
Time:                         07:18:15   BIC                           1502.743
Sample:                     01-31-2011   HQIC                          1493.756
                          - 08-31-2018
Covariance Type:                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0327      0.068      0.481      0.631      -0.101       0.166
ar.L2         -0.3514      0.079     -4.423      0.000      -0.507      -0.196
ar.L3         -0.3202      0.089     -3.598      0.000      -0.495      -0.146
ar.L4         -0.2764      0.074     -3.736      0.000      -0.421      -0.131
ar.L5          0.0248      0.091      0.272      0.786      -0.154       0.204
sigma2       5.63e+05   7.33e+04      7.678      0.000    4.19e+05    7.07e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.85   Jarque-Bera (JB):                 4.87
Prob(Q):                              0.36   Prob(JB):                         0.09
Heteroskedasticity (H):               0.55   Skew:                             0.53
Prob(H) (two-sided):                  0.11   Kurtosis:                         3.39
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

In [303…

```
test_pred2 = AR2.predict(start=0, end=len(train)-1, dynamic=True)
len(test_pred2)
```

Out[303…

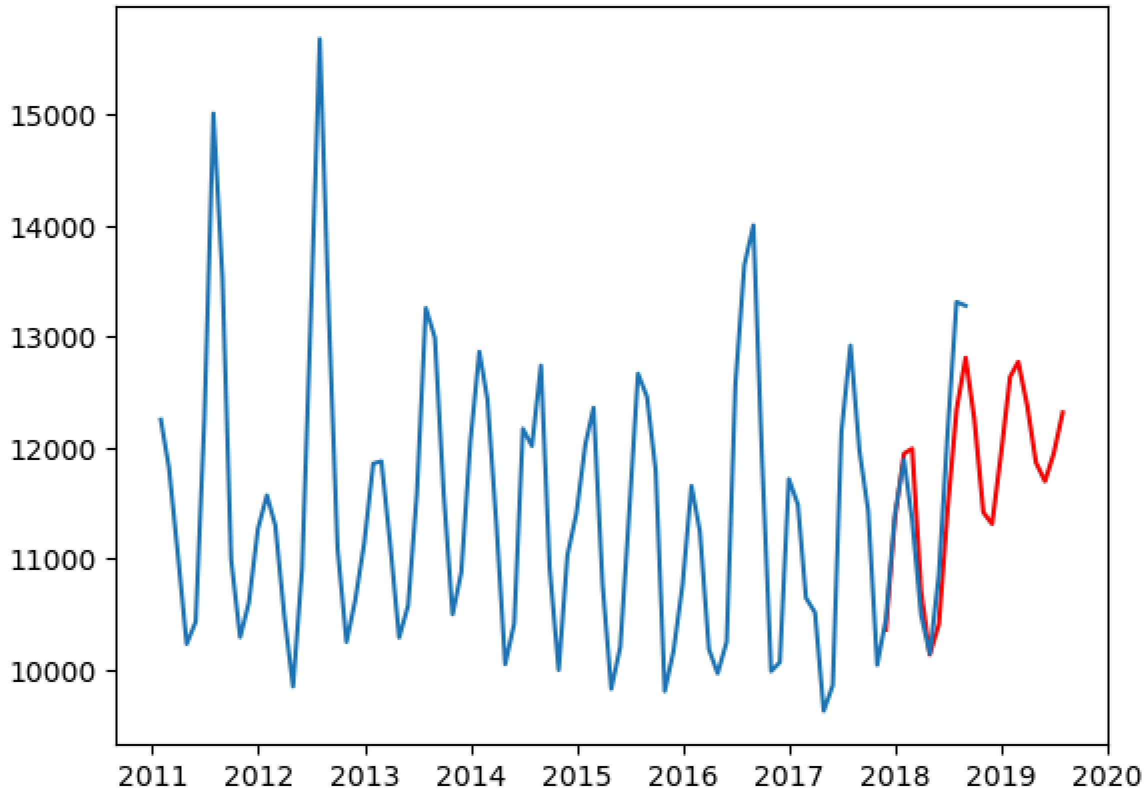
```
64
```

In [304…

```
# todo: use your fitted model from above, plus .predict(), to predict on the traini
# start = len(train)
# end = len(train) + ?

test_predict = AR2.predict(start=len(train)-10, end=len(train)+10, dynamic=False)
```

In [307…

```
# todo: plot test and test_pred from the above cell on the same plot
plt.plot(test_predict, color='red')

plt.plot(comed_df_monthly);
```



Now with Prophet

In [310…
```
#conda install -c conda-forge prophet
```

In [311…
```
# pip install prophet
```

In [312…
```
comed_df_monthly
```

Out[312…

Datetime	COMED_MW
2011-01-31	12252.029610
2011-02-28	11820.212798
2011-03-31	11028.816958
2011-04-30	10229.681944
2011-05-31	10429.517473
...	...
2018-04-30	10140.466667
2018-05-31	10852.911290
2018-06-30	12127.770833
2018-07-31	13312.620968
2018-08-31	13278.897959

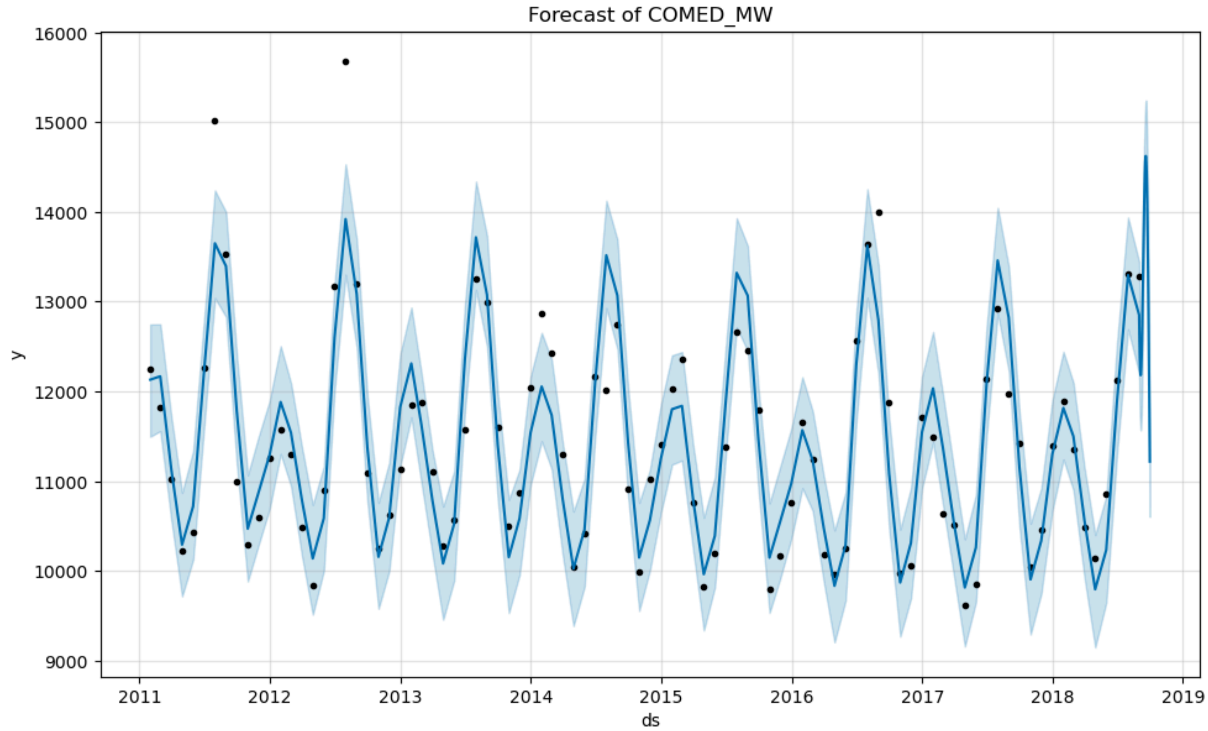
92 rows × 1 columns

In [314…

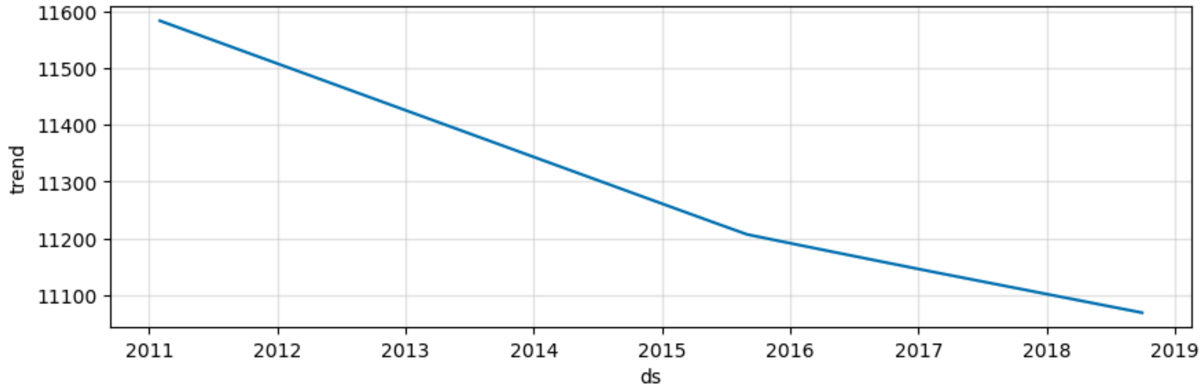
```
from prophet import Prophet

# Assuming comed_df_monthly has 'Datetime' as the index

# Reset the index to make 'Datetime' a regular column
comed_df_monthly_prophet = comed_df_monthly.reset_index()

# Rename columns to match Prophet's requirements
prophet_df = comed_df_monthly_prophet.rename(columns={'Datetime': 'ds', 'COMED_MW':

# Ensure the 'ds' column is in datetime format
prophet_df['ds'] = pd.to_datetime(prophet_df['ds'])

# Sort the dataframe by date
prophet_df = prophet_df.sort_values('ds')

# If you have any missing values, you might want to handle them
prophet_df = prophet_df.dropna()

# Now you can proceed with creating and fitting the Prophet model
model = Prophet()
model.fit(prophet_df)

# Generate future dates for forecasting
future_dates = model.make_future_dataframe(periods=30)  # 30 months into the future

# Make predictions
forecast = model.predict(future_dates)
```

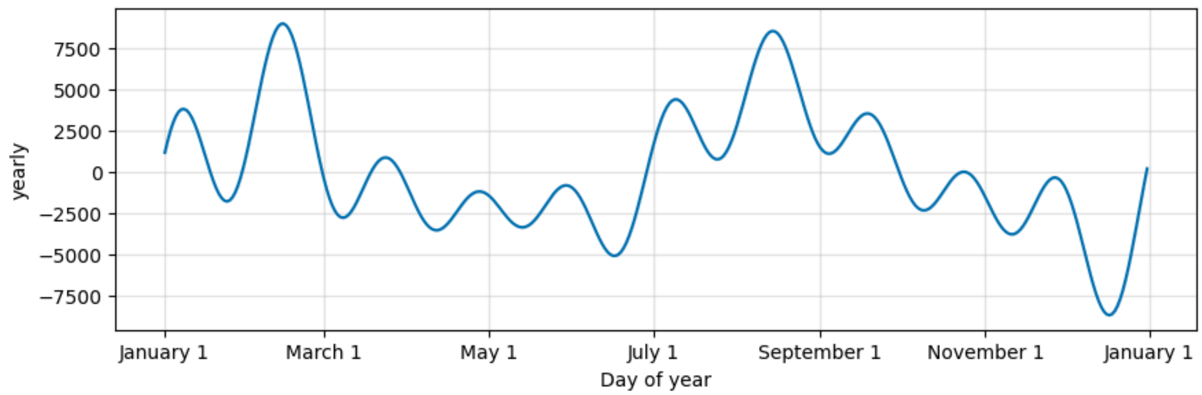
```
07:18:16 - cmdstanpy - INFO - Chain [1] start processing
07:18:16 - cmdstanpy - INFO - Chain [1] done processing
```

In [316…
```
fig1 = model.plot(forecast)
plt.title('Forecast of COMED_MW')
plt.show()
```



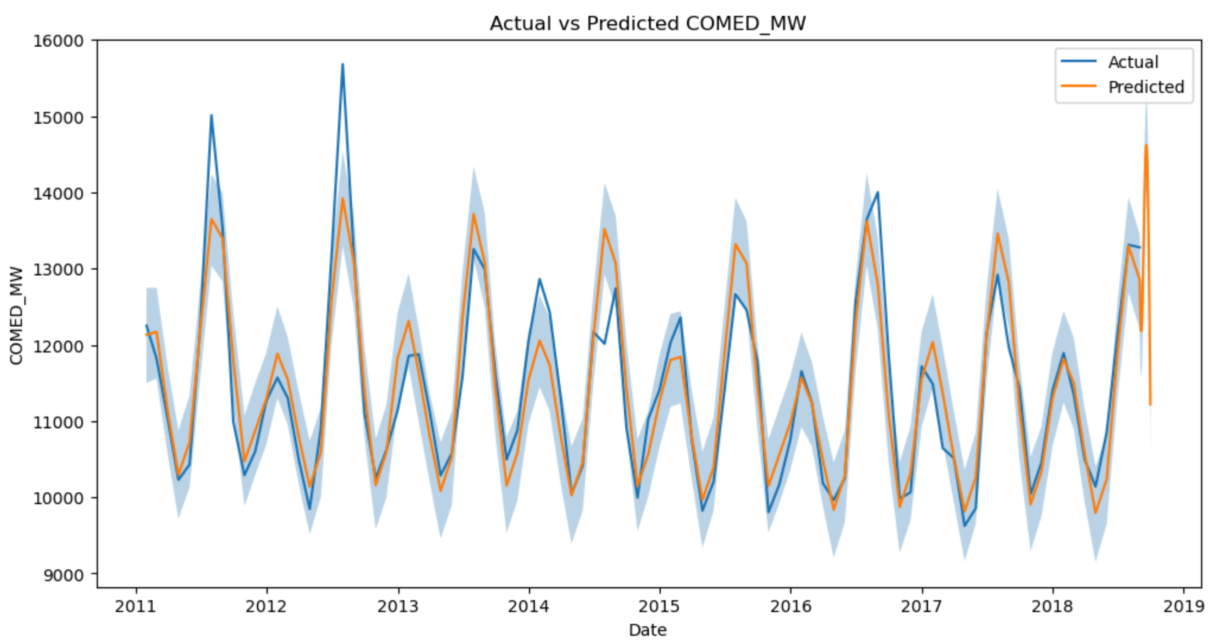
In [318…
```
fig2 = model.plot_components(forecast)
plt.show()
```





In [319…

```
plt.figure(figsize=(12,6))
plt.plot(prophet_df['ds'], prophet_df['y'], label='Actual')
plt.plot(forecast['ds'], forecast['yhat'], label='Predicted')
plt.fill_between(forecast['ds'], forecast['yhat_lower'], forecast['yhat_upper'], al
plt.legend()
plt.title('Actual vs Predicted COMED_MW')
plt.xlabel('Date')
plt.ylabel('COMED_MW')
plt.show()
```



In [322…

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Merge actual and predicted values
results = pd.merge(prophet_df, forecast[['ds', 'yhat']], on='ds')

# Calculate metrics
mae = mean_absolute_error(results['y'], results['yhat'])
rmse = np.sqrt(mean_squared_error(results['y'], results['yhat']))
mape = np.mean(np.abs((results['y'] - results['yhat']) / results['y'])) * 100

print(f'Mean Absolute Error: {mae:.2f}')
print(f'Root Mean Square Error: {rmse:.2f}')
print(f'Mean Absolute Percentage Error: {mape:.2f}%')
```

```
Mean Absolute Error: 340.78
Root Mean Square Error: 471.65
Mean Absolute Percentage Error: 2.88%
```

Now lets try Prophet on the Monthly Consumption Total Data

In [329…

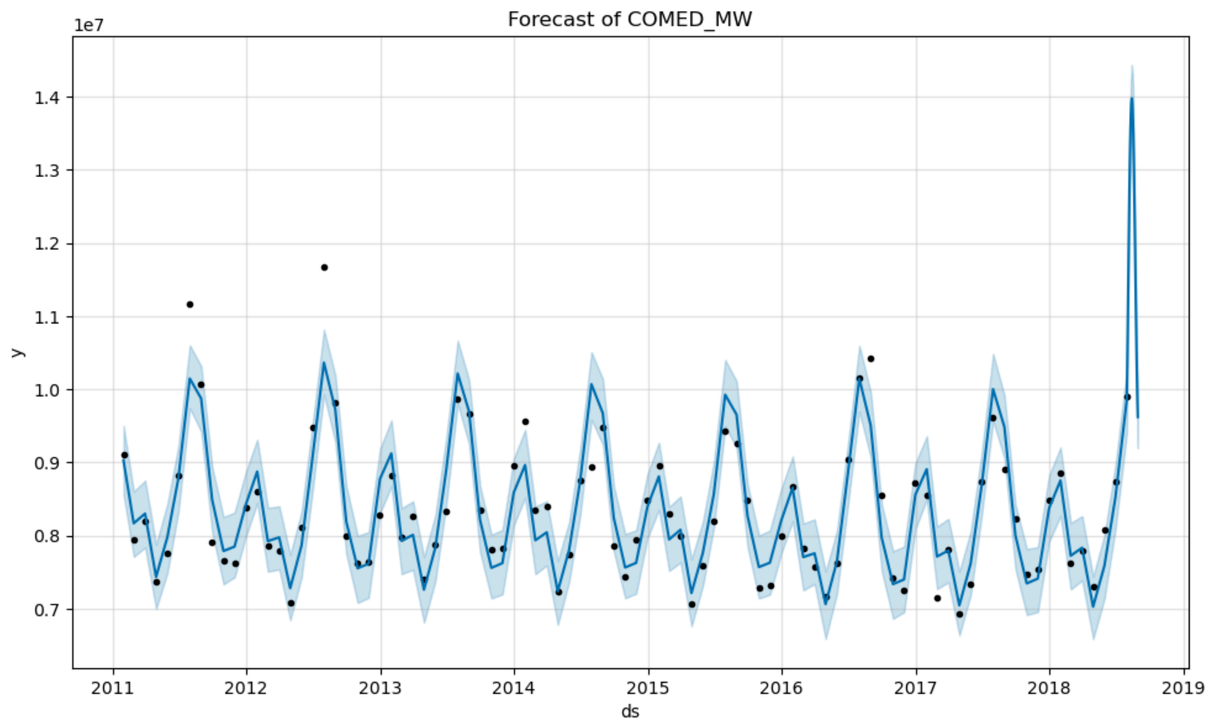
```
from prophet import Prophet

# Assuming comed_df_monthly has 'Datetime' as the index

# Reset the index to make 'Datetime' a regular column
monthly_consumption_prophet = monthly_consumption_comp.reset_index()

# Rename columns to match Prophet's requirements
prophet_df = monthly_consumption_prophet.rename(columns={'Datetime': 'ds', 'COMED_M

# Ensure the 'ds' column is in datetime format
prophet_df['ds'] = pd.to_datetime(prophet_df['ds'])

# Sort the dataframe by date
prophet_df = prophet_df.sort_values('ds')

# If you have any missing values, you might want to handle them
prophet_df = prophet_df.dropna()

# Now you can proceed with creating and fitting the Prophet model
model = Prophet()
model.fit(prophet_df)

# Generate future dates for forecasting
future_dates = model.make_future_dataframe(periods=30)  # 30 months into the future

# Make predictions
forecast = model.predict(future_dates)
```

```
07:18:18 - cmdstanpy - INFO - Chain [1] start processing
07:18:19 - cmdstanpy - INFO - Chain [1] done processing
```

In [330…

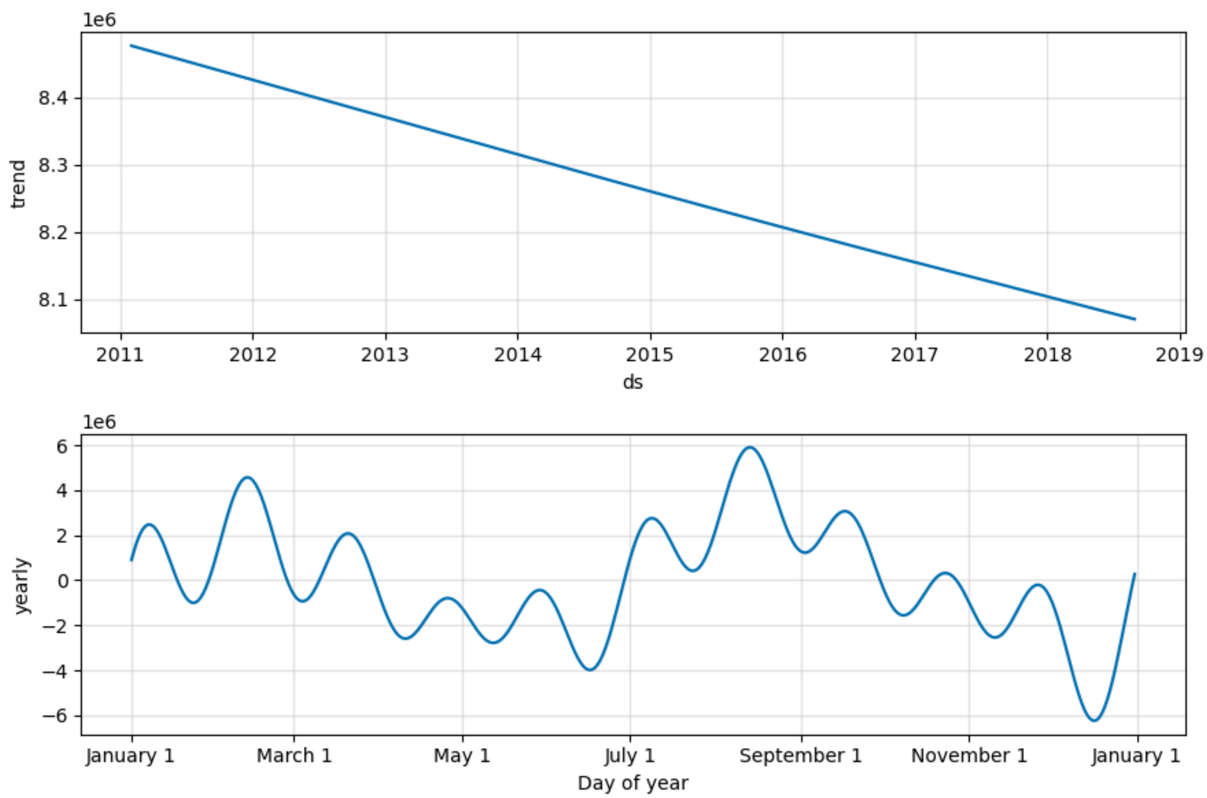
```
fig1 = model.plot(forecast)
plt.title('Forecast of COMED_MW')
```

```
plt.show()
```



In [334…

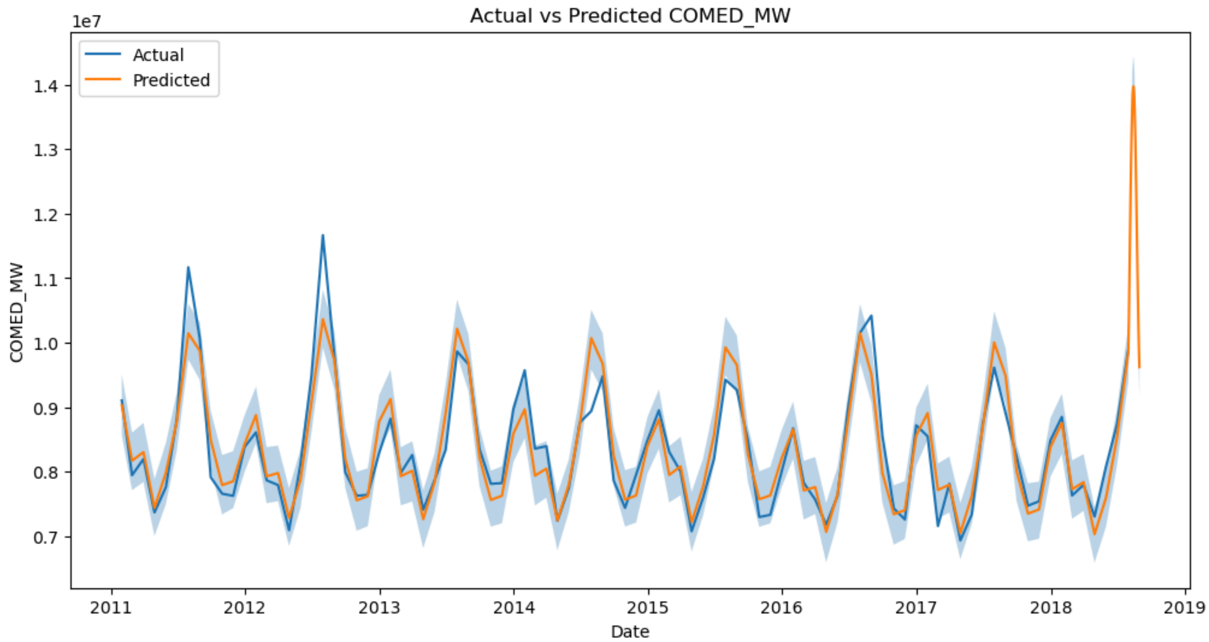
```
fig2 = model.plot_components(forecast)
plt.show()
```



In [335…

```
plt.figure(figsize=(12,6))
plt.plot(prophet_df['ds'], prophet_df['y'], label='Actual')
plt.plot(forecast['ds'], forecast['yhat'], label='Predicted')
plt.fill_between(forecast['ds'], forecast['yhat_lower'], forecast['yhat_upper'], al
```

```
plt.legend()
plt.title('Actual vs Predicted COMED_MW')
plt.xlabel('Date')
plt.ylabel('COMED_MW')
plt.show()
```



In [339...

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Merge actual and predicted values
results = pd.merge(prophet_df, forecast[['ds', 'yhat']], on='ds')

# Calculate metrics
mae = mean_absolute_error(results['y'], results['yhat'])
rmse = np.sqrt(mean_squared_error(results['y'], results['yhat']))
mape = np.mean(np.abs((results['y'] - results['yhat']) / results['y'])) * 100

print(f'Mean Absolute Error: {mae:.2f}')
print(f'Root Mean Square Error: {rmse:.2f}')
print(f'Mean Absolute Percentage Error: {mape:.2f}%')
```

```
Mean Absolute Error: 250157.16
Root Mean Square Error: 346818.53
Mean Absolute Percentage Error: 2.91%
```

- The Mean Absolute Percentage Error of the Total Monthly Consumption sum is similar to the Average Monthly Hourly Consumption.
- The Mean Absolute Error is much bigger in the total consumption sum, but this is only because scale. I am using monthly consumption which is in the millions of Megawatts, and Average Monthly Hourly consumption is in tens of thousands because it's Megawatt per hour.

Conclusion

- Facebook Prophet has tremendous capabilities for forecasting similar to ARIMA model, using lags, and even correcting for holidays, etc. With Prophet we don't have to worry about the integration either.
- Prophet was much easier to use, though it's like a black box, it's hard to explain the trend. What I would recommend is to run ARIMA models first, and then Prophet, and see if they are similar.
- If they are similar I would use ARIMA to explain how time series forecasts work in the analysis we made, and then let them know that Prophet accounts for more things like holidays, and corrects for other seasonal events.
- What is great is to see that both ARIMA and Prophet where consistent projecting the 10 and 30 day forecast. Prophet looked cleaner though and sticked better to the limits within the series. ARIMA tends to go out of bounds when Dynamic is set to "True".
- Average Monthly Hour Consumption gave slightly better Mean Absolute Percentage Error, but they were both around the same level.
- When modeling with ARIMA I want to know what to do when I reach zero, I researched that I can substitute negative amounts with 0 post model fitting and plotting, if I am modeling a value that can't be zero. I wonder if Prophet has limits that would identify that my time series has no negative values.
- This would be specially useful for modeling financial data like when trying to predict yearly revenue.