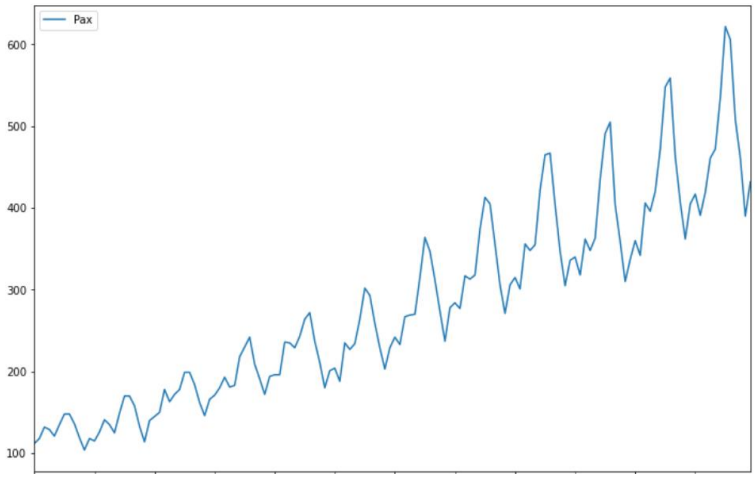
Plotting the time series

In [13]:
```
#Increase the figure size
from pylab import rcParams
rcParams['figure.figsize'] = 12, 8
df1.plot()
plt.show()
```

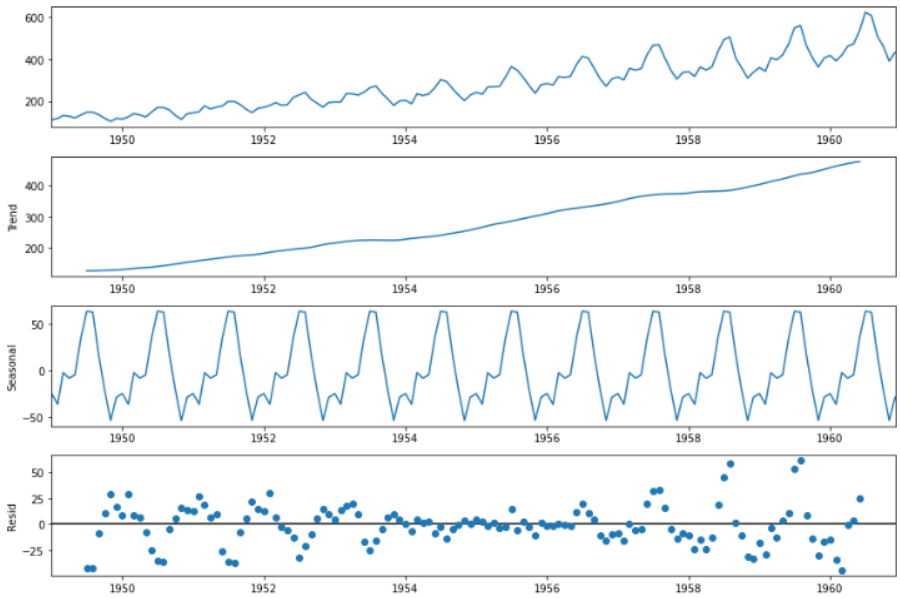


Plotting the decomposition Additive

We see an increasing trend and seasonality which is not constant in nature.

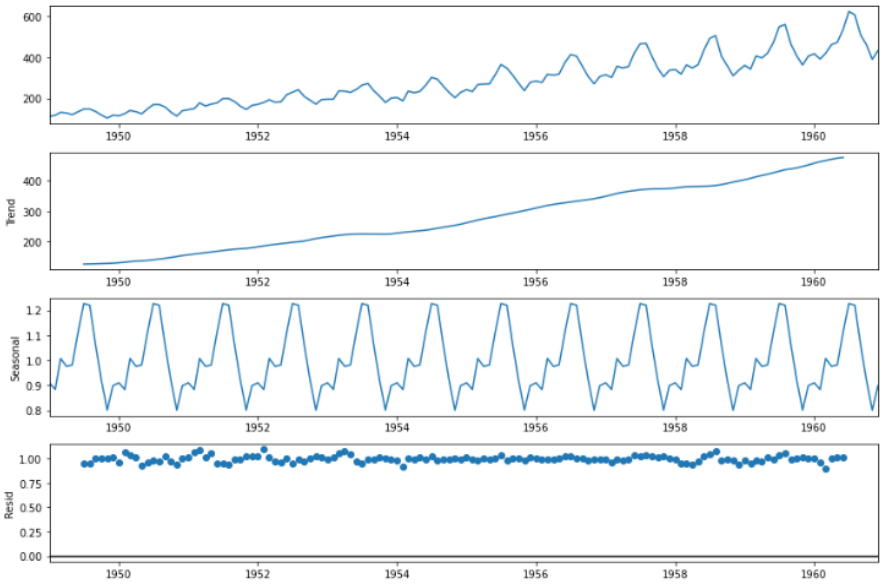
In [14]:
```
#Decompose the time series additively
df1_add_decompose = seasonal_decompose(df1, model = 'additive', period = 12)
```

In [15]:
```
df1_add_decompose.plot()
plt.show()
```



Plotting the decomposition Multiplicative and also doing log if required. If residual is above 1 series is multiplicative.

In [16]:
```
##Decompose the time series multiplicatively
df1_mul_decompose = seasonal_decompose(df1, model = "multiplicative")
df1_mul_decompose.plot()
plt.show()
```



In [17]:
```
#Let's try to do log transformation
df1_log = df1.copy()
```

In [18]:
```
df1_log['Pax'] = np.log(df1)
```

In [19]:
```
df1_log.Pax
```

Out[19]:
```
Year-Month
1949-01-01    4.718499
```

Log Transformation

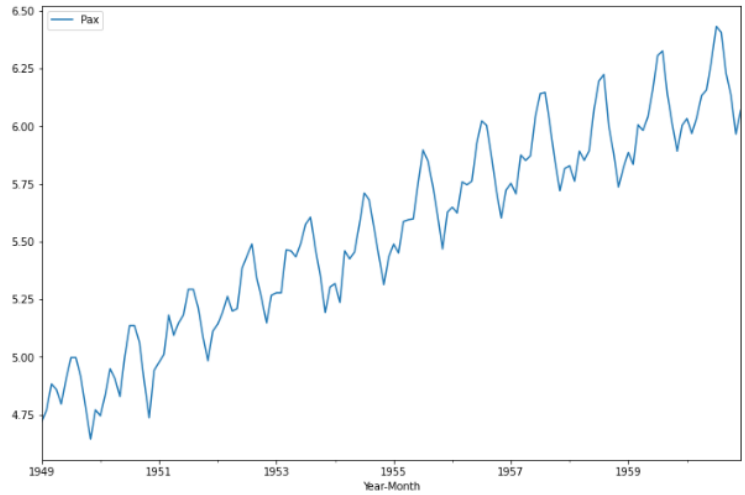
In [17]:
```
#Let's try to do log transformation
df1_log = df1.copy()
```

In [18]:
```
df1_log['Pax'] = np.log(df1)
```

In [19]:
```
df1_log.Pax
```

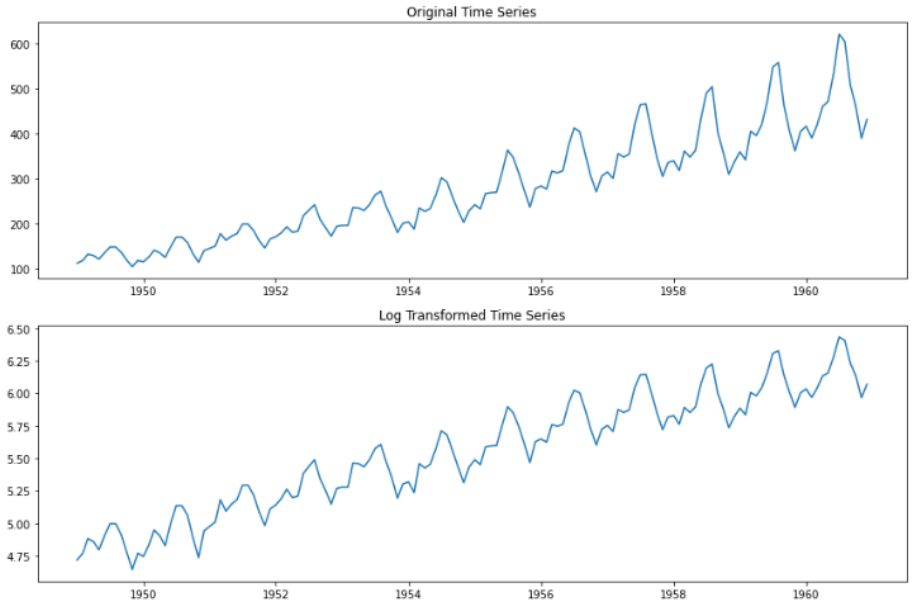
Out[19]:
```
Year-Month
1949-01-01    4.718499
1949-02-01    4.770685
1949-03-01    4.882802
1949-04-01    4.859812
1949-05-01    4.795791
                 ...
1960-08-01    6.406880
1960-09-01    6.230481
1960-10-01    6.133398
1960-11-01    5.966147
1960-12-01    6.068426
Name: Pax, Length: 144, dtype: float64
```

In [20]:
```
#Visualize the log transformed series
df1_log.plot()
plt.show()
```



Comparing Original and Log Transformation

In [21]:
```
#Compare with the original series
plt.subplot(2,1,1)
plt.title('Original Time Series')
plt.plot(df1)

plt.subplot(2,1,2)
plt.title('Log Transformed Time Series')
plt.plot(df1_log)
plt.tight_layout()
```



Unclear Seasonality in plot

Example 2

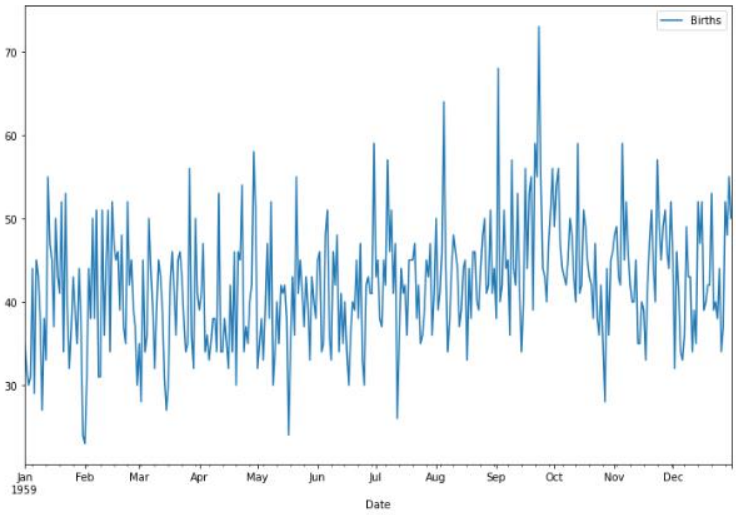
In [23]:
```
#Read the data
df2 = pd.read_csv('daily-total-female-births-E.csv', parse_dates = ['Date'], index_col = 'Date')
```

In [24]:
```
df2.head()
```

Out[24]:

	Births
Date	
1959-01-01	35
1959-01-02	32
1959-01-03	30
1959-01-04	31
1959-01-05	44

In [25]:
```
#Visualise the time series
df2.plot()
plt.show()
```

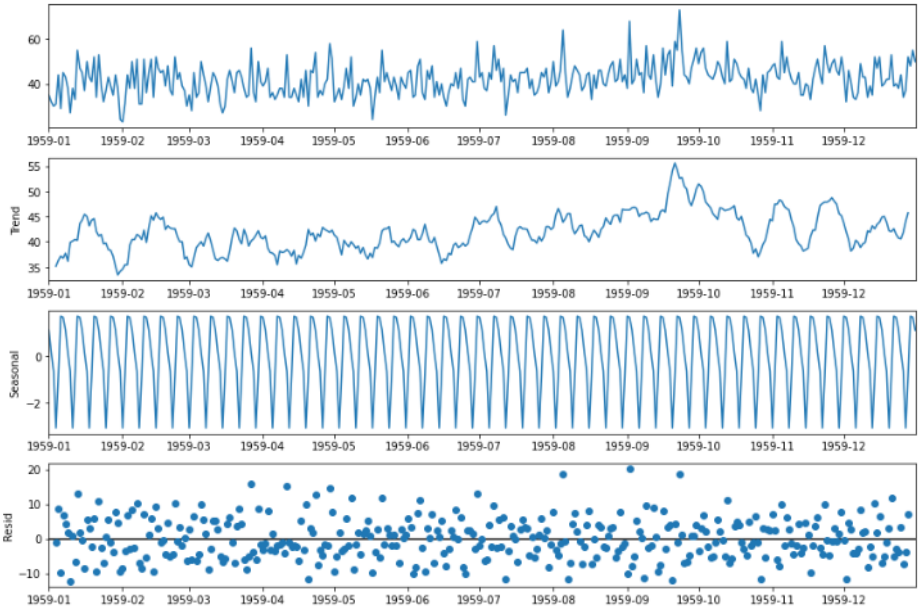


Doesn't show very clear trend and seasonality.

Comparing Trend, Seasonality and Residual for **Additive** – Added to understand Observed.

In [26]:
```
#Additive decomposition
df2_add_decompose = seasonal_decompose(df2, model = 'additive')
```

In [27]:
```
df2_add_decompose.plot()
plt.show()
```



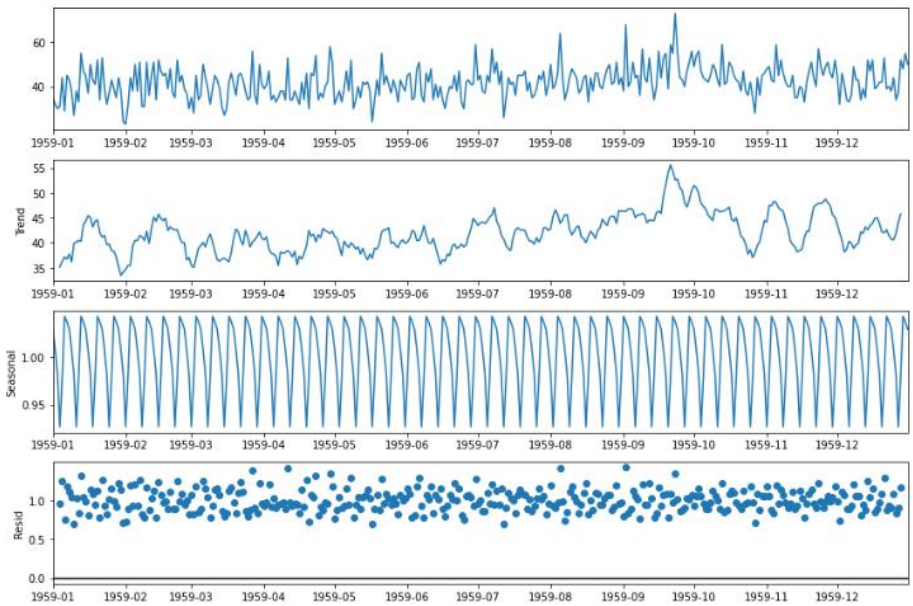
In [28]:
```
#Let's inspect each component
df2_add_decompose.trend
```

Out[28]:
```
Date
1959-01-01          NaN
1959-01-02          NaN
1959-01-03          NaN
1959-01-04    35.142857
1959-01-05    36.285714
                 ...
1959-12-27    44.000000
1959-12-28    45.714286
1959-12-29          NaN
1959-12-30          NaN
```

```
In [28]: #Let's inspect each component
         df2_add_decompose.trend

Out[28]: Date
         1959-01-01          NaN
         1959-01-02          NaN
         1959-01-03          NaN
         1959-01-04    35.142857
         1959-01-05    36.285714
                          ...
         1959-12-27    44.000000
         1959-12-28    45.714286
         1959-12-29          NaN
         1959-12-30          NaN
         1959-12-31          NaN
         Name: trend, Length: 365, dtype: float64

In [29]: df2_add_decompose.seasonal

Out[29]: Date
         1959-01-01    1.105812
         1959-01-02    0.142226
         1959-01-03   -0.656093
         1959-01-04   -3.077608
         1959-01-05   -0.868817
                          ...
         1959-12-27   -3.077608
         1959-12-28   -0.868817
         1959-12-29    1.702450
         1959-12-30    1.652030
         1959-12-31    1.105812
         Name: seasonal, Length: 365, dtype: float64

In [30]: df2_add_decompose.resid

Out[30]: Date
         1959-01-01         NaN
         1959-01-02         NaN
         1959-01-03         NaN
         1959-01-04   -1.065249
         1959-01-05    8.583102
                          ...
         1959-12-27   -3.922392
         1959-12-28    7.154531
         1959-12-29         NaN
         1959-12-30         NaN
         1959-12-31         NaN
         Name: resid, Length: 365, dtype: float64

In [31]: #Since this is an additive model:
         #Observed = Trend + Seasonal + Irregular should hold true
         35.142857 -3.077608 -1.065249

Out[31]: 31.0
```

Comparing Trend, Seasonality and Residual for Multiplicative– Multiplied to understand Observed.

```
In [34]: #Multiplicative decomposition
         df2_mul_decompose = seasonal_decompose(df2, model = 'multiplicative')
         df2_mul_decompose.plot()
         plt.show()
```



In [36]:
```
#Let's inspect each component
df2_mul_decompose.trend
```

Out[36]:
```
Date
1959-01-01         NaN
1959-01-02         NaN
1959-01-03         NaN
1959-01-04   35.142857
1959-01-05   36.285714
                ...
1959-12-27   44.000000
1959-12-28   45.714286
1959-12-29         NaN
1959-12-30         NaN
1959-12-31         NaN
Name: trend, Length: 365, dtype: float64
```

In [37]:
```
df2_mul_decompose.seasonal
```

Out[37]:
```
Date
1959-01-01   1.028509
1959-01-02   1.007079
1959-01-03   0.981494
1959-01-04   0.926726
1959-01-05   0.977911
                ...
1959-12-27   0.926726
1959-12-28   0.977911
1959-12-29   1.041987
1959-12-30   1.036295
1959-12-31   1.028509
Name: seasonal, Length: 365, dtype: float64
```

In [38]:
```
df2_mul_decompose.resid
```

Out[38]:
```
Date
1959-01-01        NaN
1959-01-02        NaN
1959-01-03        NaN
1959-01-04   0.951861
1959-01-05   1.239989
                ...
1959-12-27   0.907398
1959-12-28   1.163194
1959-12-29        NaN
1959-12-30        NaN
1959-12-31        NaN
Name: resid, Length: 365, dtype: float64
```

In [39]:
```
#Since this is a multiplicative model:
#Observed = Trend*Seasonality*Irregular
35.14*.93*.952
```

Out[39]:
```
31.1115504
```

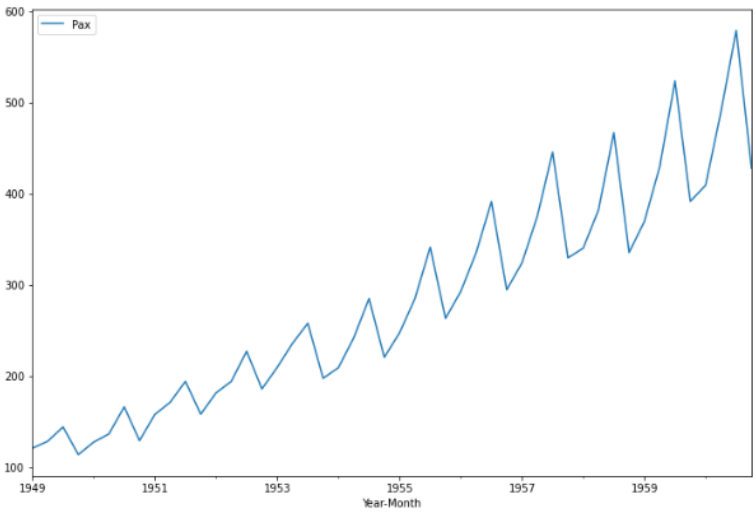
Converting from Monthly to Quarterly

Downsampling

In [40]:
```
#Let's change the monthly series to quarterly. This would require aggregation.
df1_q = df1.resample('Q').mean()
```

In [41]:
```
df1_q.plot()
```

Out[41]:
```
<AxesSubplot:xlabel='Year-Month'>
```



Converting from Monthly to Daily

Upsampling

In [42]:
```
#Let's change the monthly series to daily.
df1_d = df1.resample('D').ffill()
```

In [43]:
```
df1_d['1949-02']
```

```
C:\Users\graje\AppData\Local\Temp\ipykernel_23688\2013041239.py:1: FutureWarning: Indexing a DataFrame with a datetimelike inde
x using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `fr
ame.loc[string]` instead.
  df1_d['1949-02']
```

Out[43]:

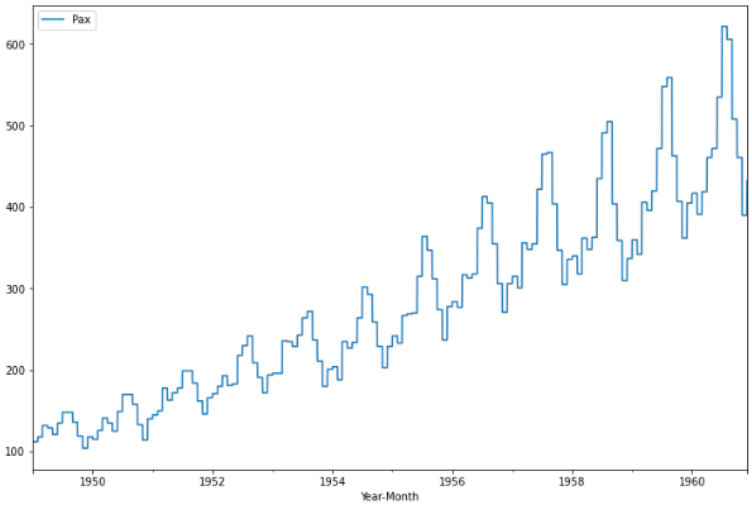
Year-Month	Pax
1949-02-01	118
1949-02-02	118
1949-02-03	118
1949-02-04	118
1949-02-05	118
1949-02-06	118
1949-02-07	118
1949-02-08	118
1949-02-09	118
1949-02-10	118
1949-02-11	118
1949-02-12	118
1949-02-13	118
1949-02-14	118
1949-02-15	118
1949-02-16	118
1949-02-17	118
1949-02-18	118
1949-02-19	118
1949-02-20	118
1949-02-21	118
1949-02-22	118
1949-02-23	118
1949-02-24	118
1949-02-25	118
1949-02-26	118

Converting from Monthly to Hourly

1949-02-23	118
1949-02-24	118
1949-02-25	118
1949-02-26	118
1949-02-27	118
1949-02-28	118

In [44]:
```
df1_d.plot()
```

Out[44]: <AxesSubplot:xlabel='Year-Month'>



In [45]:
```
#Let's change the monthly series to hourly.
df1_h = df1.resample('H').interpolate()
```

In [46]:
```
df1_h
```

Out[46]:

Year-Month	Pax
1949-01-01 00:00:00	112.000000
1949-01-01 01:00:00	112.008065

Converting from Monthly to Hourly

In [45]:
```
#Let's change the monthly series to hourly.
df1_h = df1.resample('H').interpolate()
```

In [46]:
```
df1_h
```

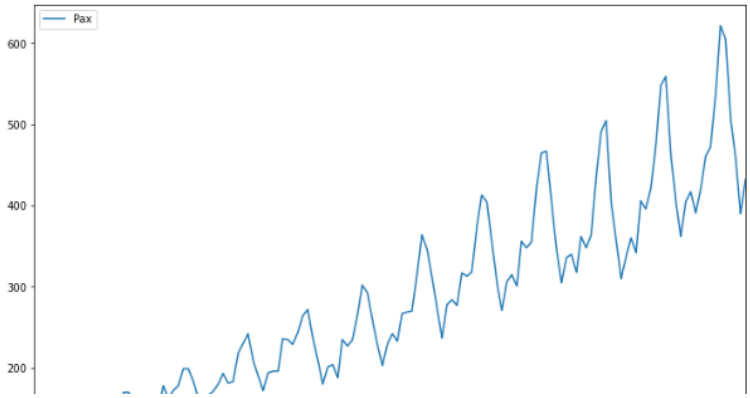
Out[46]:

Year-Month	Pax
1949-01-01 00:00:00	112.000000
1949-01-01 01:00:00	112.008065
1949-01-01 02:00:00	112.016129
1949-01-01 03:00:00	112.024194
1949-01-01 04:00:00	112.032258
...	...
1960-11-30 20:00:00	431.766667
1960-11-30 21:00:00	431.825000
1960-11-30 22:00:00	431.883333
1960-11-30 23:00:00	431.941667
1960-12-01 00:00:00	432.000000

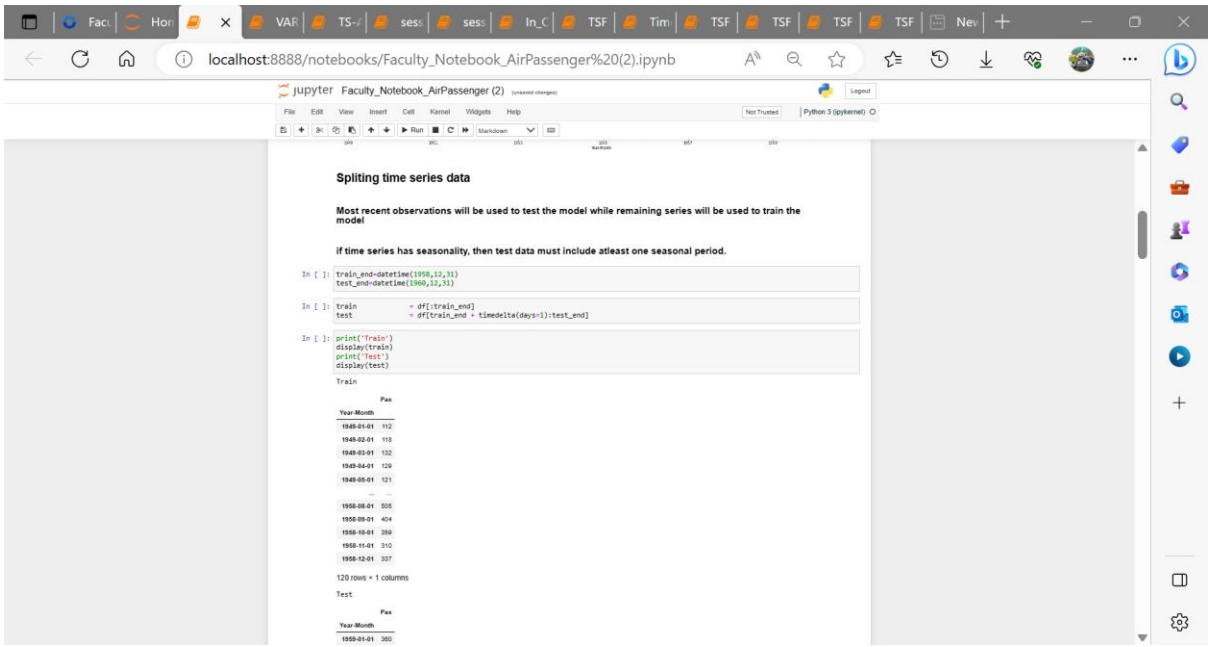
104449 rows × 1 columns

In [47]:
```
df1_h.plot()
```

Out[47]:
```
<AxesSubplot:xlabel='Year-Month'>
```



Splitting data into test and training



Spliting time series data

Most recent observations will be used to test the model while remaining series will be used to train the model

if time series has seasonality, then test data must include atleast one seasonal period.

```
In [ ]: train_end=datetime(1958,12,31)
        test_end=datetime(1960,12,31)
```

```
In [ ]: train      = df[:train_end]
        test       = df[train_end + timedelta(days=1):test_end]
```

```
In [ ]: print('Train')
        display(train)
        print('Test')
        display(test)
```

Train

Year-Month	Pax
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121
...	...
1958-08-01	505
1958-09-01	404
1958-10-01	359
1958-11-01	310
1958-12-01	337

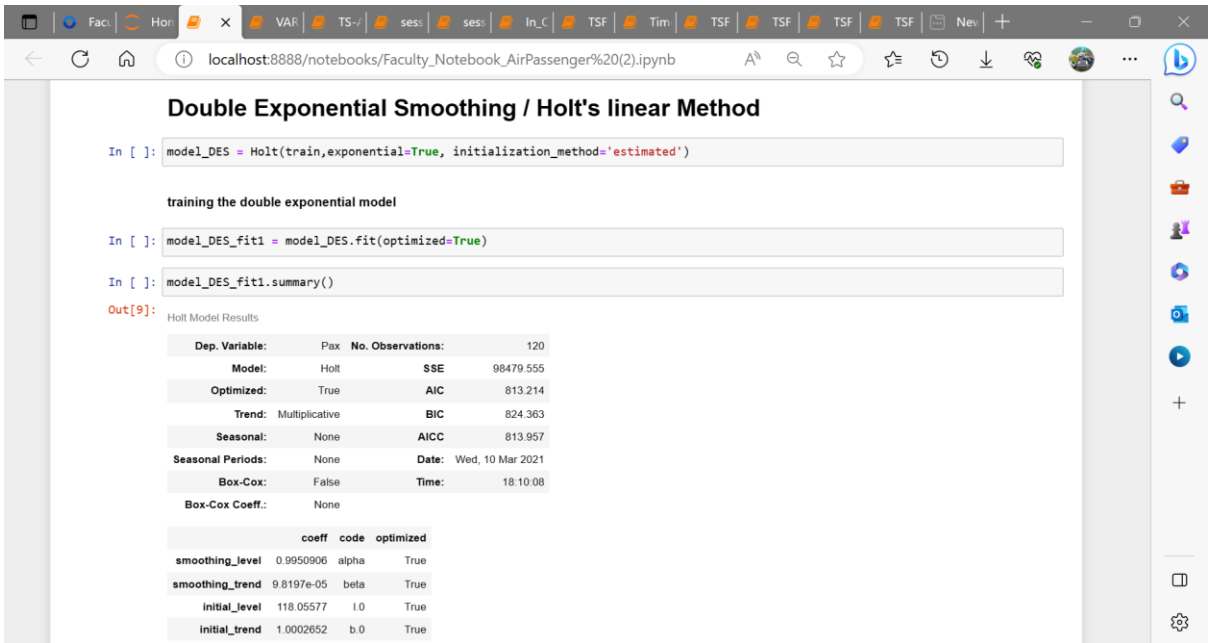
120 rows × 1 columns

Test

Year-Month	Pax
1959-01-01	360

DES – Double Exponential smoothing – Holt's Linear Method

`



Double Exponential Smoothing / Holt's linear Method

```
In [ ]: model_DES = Holt(train,exponential=True, initialization_method='estimated')
```

training the double exponential model

```
In [ ]: model_DES_fit1 = model_DES.fit(optimized=True)
```

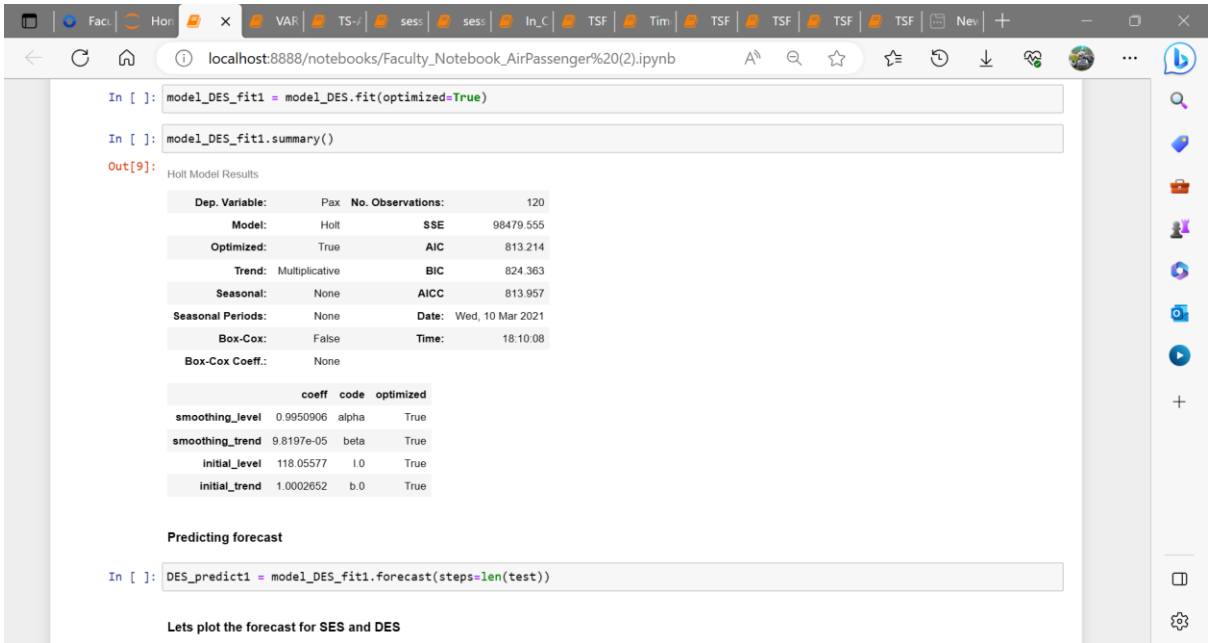
```
In [ ]: model_DES_fit1.summary()
```

Out[9]:

Holt Model Results

Dep. Variable:	Pax	No. Observations:	120
Model:	Holt	SSE	98479.555
Optimized:	True	AIC	813.214
Trend:	Multiplicative	BIC	824.363
Seasonal:	None	AICC	813.957
Seasonal Periods:	None	Date:	Wed, 10 Mar 2021
Box-Cox:	False	Time:	18:10:08
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.9950906	alpha	True
smoothing_trend	9.8197e-05	beta	True
initial_level	118.05577	l.0	True
initial_trend	1.0002652	b.0	True



```
In [ ]: model_DES_fit1 = model_DES.fit(optimized=True)

In [ ]: model_DES_fit1.summary()
```

Out[9]:

Holt Model Results

Dep. Variable:	Pax	**No. Observations:**	120
Model:	Holt	**SSE**	98479.555
Optimized:	True	**AIC**	813.214
Trend:	Multiplicative	**BIC**	824.363
Seasonal:	None	**AICC**	813.957
Seasonal Periods:	None	**Date:**	Wed, 10 Mar 2021
Box-Cox:	False	**Time:**	18:10:08
Box-Cox Coeff.:	None		

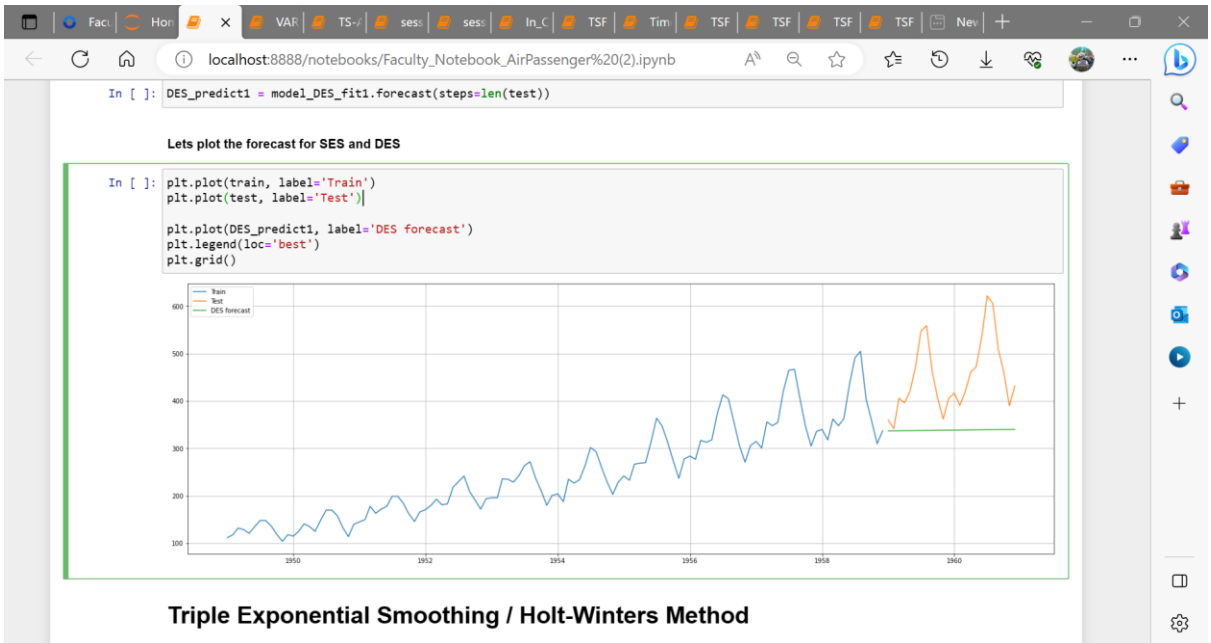
	coeff	code	optimized
smoothing_level	0.9950906	alpha	True
smoothing_trend	9.8197e-05	beta	True
initial_level	118.05577	l.0	True
initial_trend	1.0002652	b.0	True

Predicting forecast

```
In [ ]: DES_predict1 = model_DES_fit1.forecast(steps=len(test))
```

Lets plot the forecast for SES and DES

SES and DES



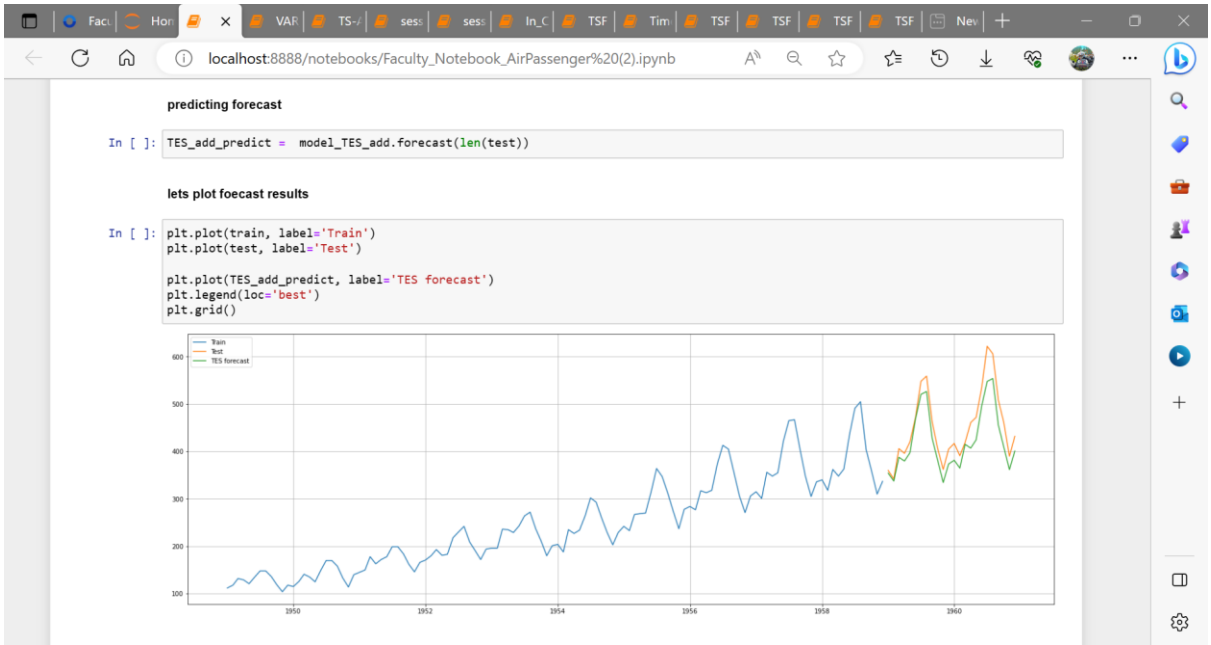
```
In [ ]: DES_predict1 = model_DES_fit1.forecast(steps=len(test))
```

Lets plot the forecast for SES and DES

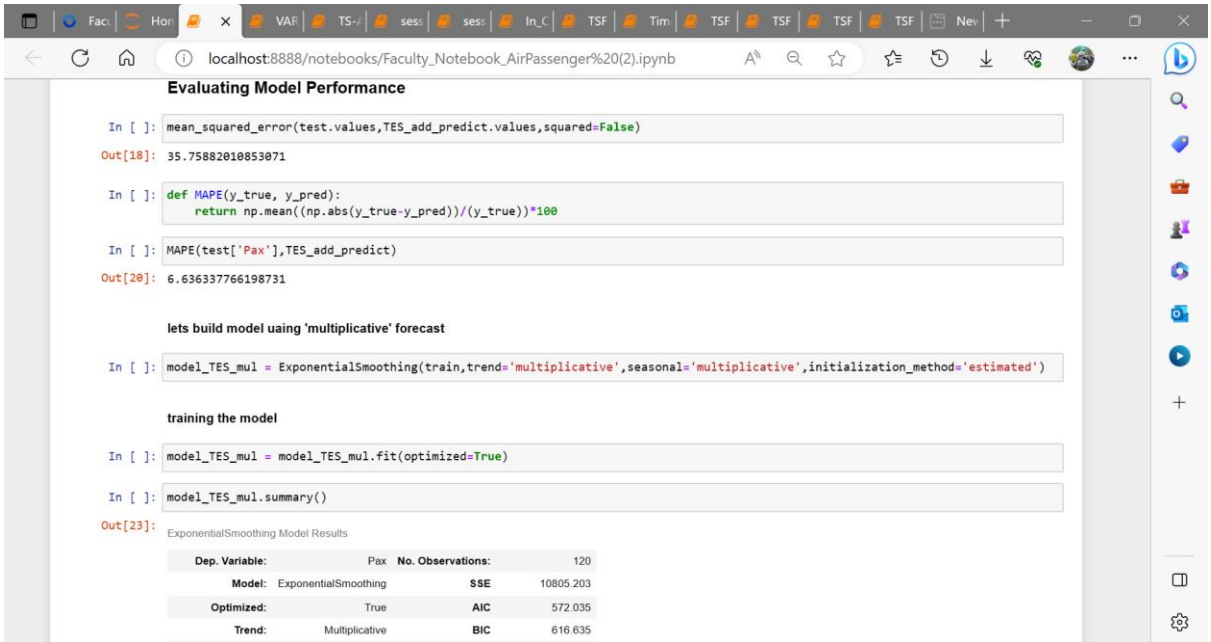
```
In [ ]: plt.plot(train, label='Train')
        plt.plot(test, label='Test')

        plt.plot(DES_predict1, label='DES forecast')
        plt.legend(loc='best')
        plt.grid()
```


Triple Exponential Smoothing / Holt-Winters Method

TES - Triple Exponential Smoothing / Holt-Winters Method



Evaluating Model / TES and Mul Forecast



localhost:8888/notebooks/Faculty_Notebook_AirPassenger%20(2).ipynb

predicting forecast

In [ ]:
```
TES_mul_predict =  model_TES_mul.forecast(len(test))
```

lets plot foecast results for H-W model with multiplicative seasonality

In [ ]:
```
plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(TES_mul_predict, label='TES forecast')
plt.legend(loc='best')
plt.grid()
```

localhost:8888/notebooks/Faculty_Notebook_AirPassenger%20(2).ipynb

Evaluating Model Performance

In [ ]:
```
mean_squared_error(test.values,TES_mul_predict.values,squared=False)
```

Out[26]:
```
13.87915336998583
```

Defining Mean Absolute Percentage error

In [ ]:
```
def MAPE(y_true, y_pred):
    return np.mean((np.abs(y_true-y_pred))/(y_true))*100
```

Mean Absolute Percentage Error for simple forecasting model

In [ ]:
```
MAPE(test['Pax'],TES_mul_predict)
```

Out[28]:
```
2.813305003034416
```

forecasting

In [ ]:
```
model_TES_mul = ExponentialSmoothing(df,trend='multiplicative',seasonal='multiplicative',initialization_method='estimated')
```

In [ ]:
```
model_TES_mul = model_TES_mul.fit(optimized=True)
```

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

Out[31]:
```
ExponentialSmoothing Model Results
```

localhost:8888/notebooks/Faculty_Notebook_AirPassenger%20(2).ipynb

initial_seasons.6	1.2019271	s.6	True
initial_seasons.7	1.1888389	s.7	True
initial_seasons.8	1.0783485	s.8	True
initial_seasons.9	0.9337215	s.9	True
initial_seasons.10	0.8188590	s.10	True
initial_seasons.11	0.9485993	s.11	True

In [ ]:
```
TES_mul_predict =  model_TES_mul.forecast(12)
```

In [ ]:
```
plt.plot(df, label='data')
plt.plot(TES_mul_predict, label='TES forecast')
plt.legend(loc='best')
plt.grid()
```

localhost:8888/notebooks/Faculty_Notebook_AirPassenger%20(2).ipynb

Triple Exponential Smoothing / Holt-Winters Method

lets build model using 'additive' seasonality

In [ ]:
```
model_TES_add = ExponentialSmoothing(train,trend='additive',seasonal='additive',initialization_method='estimated')
```

training the model

In [ ]:
```
model_TES_add = model_TES_add.fit(optimized=True)
```

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

Out[15]:

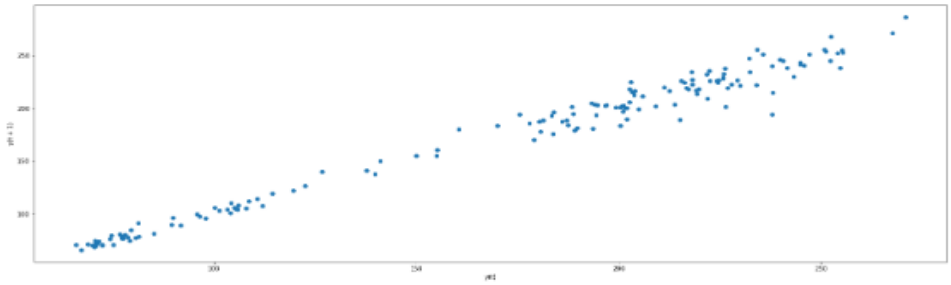
ExponentialSmoothing Model Results

Dep. Variable:	Pax	No. Observations:	120
Model:	ExponentialSmoothing	SSE	16033.589
Optimized:	True	AIC	619.394
Trend:	Additive	BIC	663.994
Seasonal:	Additive	AICC	626.166
Seasonal Periods:	12	Date:	Wed, 10 Mar 2021
Box-Cox:	False	Time:	18:10:31
Box-Cox Coeff.:	None		

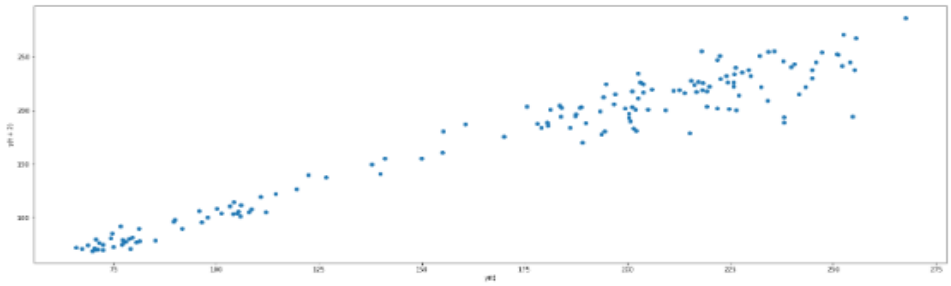
	coeff	code	optimized
smoothing_level	0.2367681	alpha	True
smoothing_trend	5.7672e-10	beta	True

Plotting Lags 1 , 2 and 3

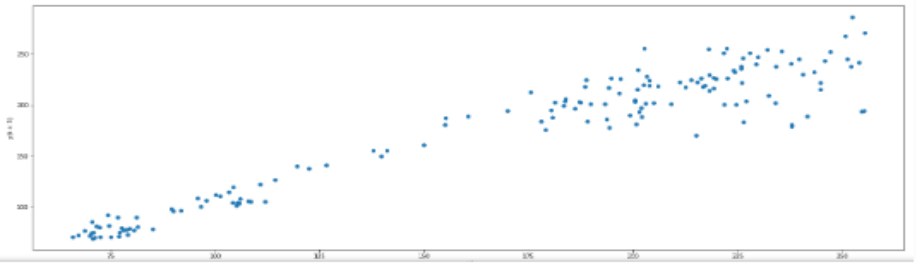
```
In [ ]: pd.plotting.lag_plot(df, lag=1)
Out[13]: <AxesSubplot:xlabel='y(t)', ylabel='y(t + 1)'>
```



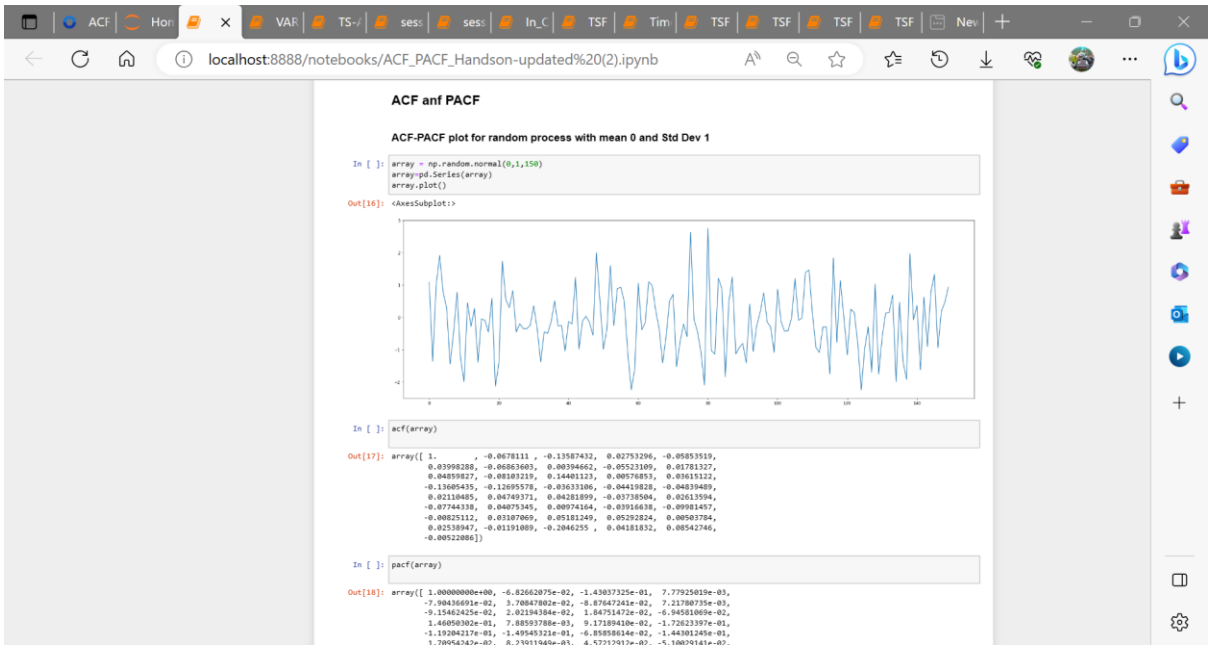
```
In [ ]: pd.plotting.lag_plot(df, lag=2)
Out[14]: <AxesSubplot:xlabel='y(t)', ylabel='y(t + 2)'>
```

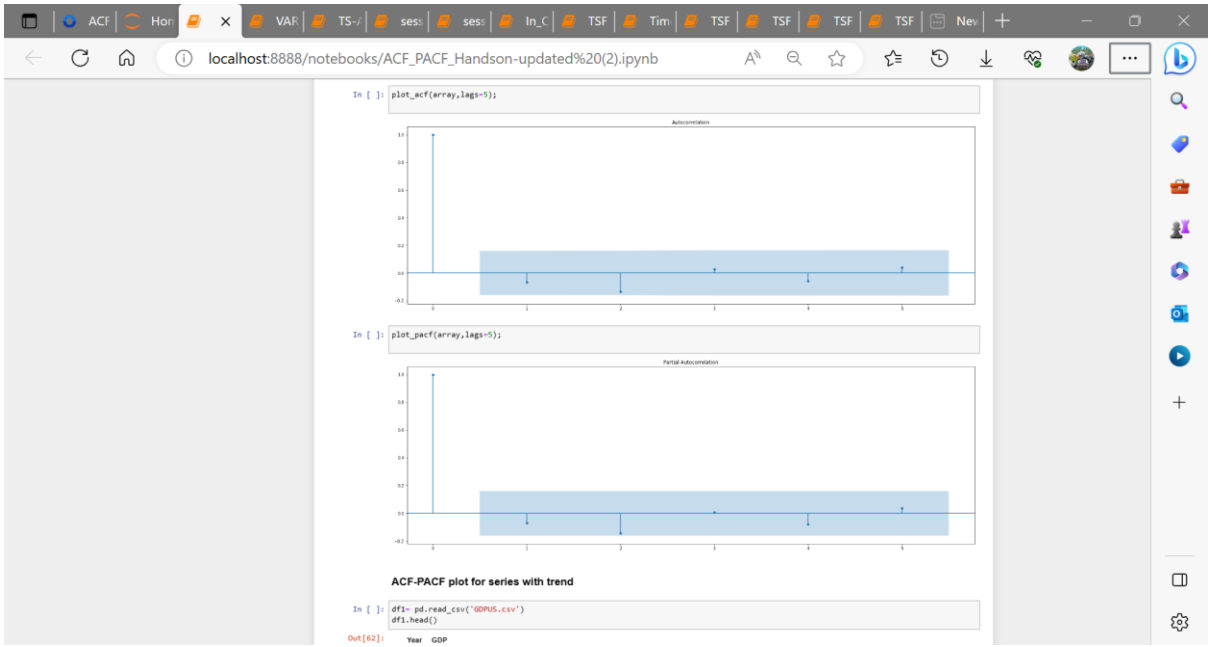


```
In [ ]: pd.plotting.lag_plot(df, lag=3)
Out[15]: <AxesSubplot:xlabel='y(t)', ylabel='y(t + 3)'>
```

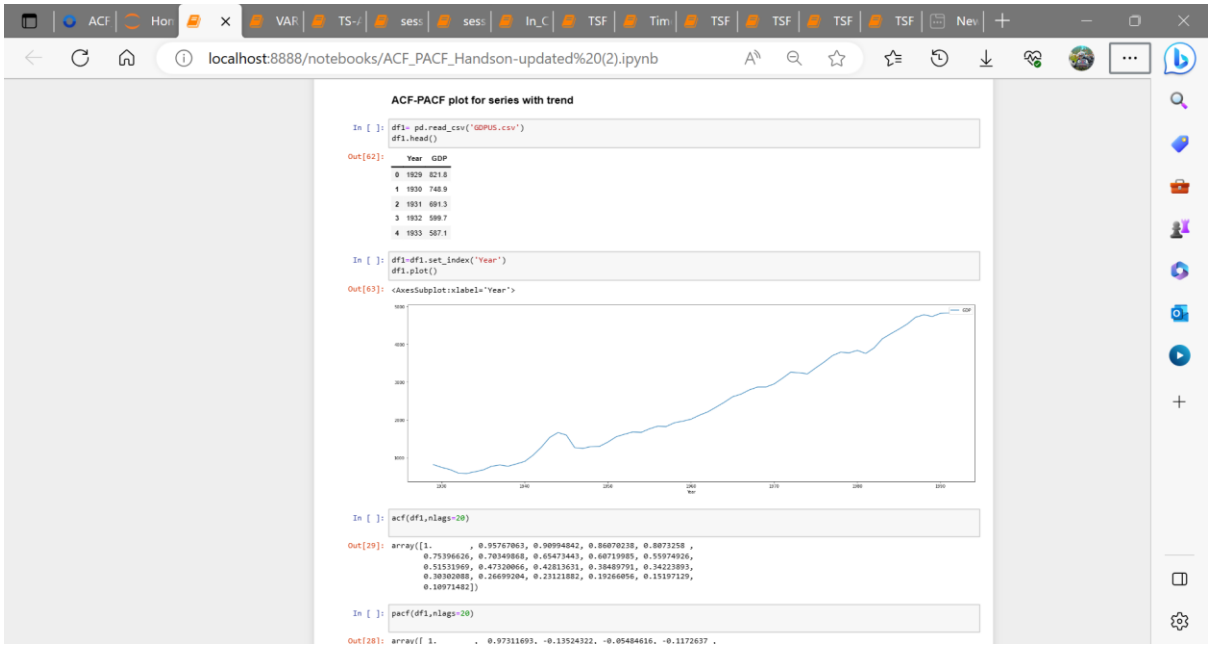


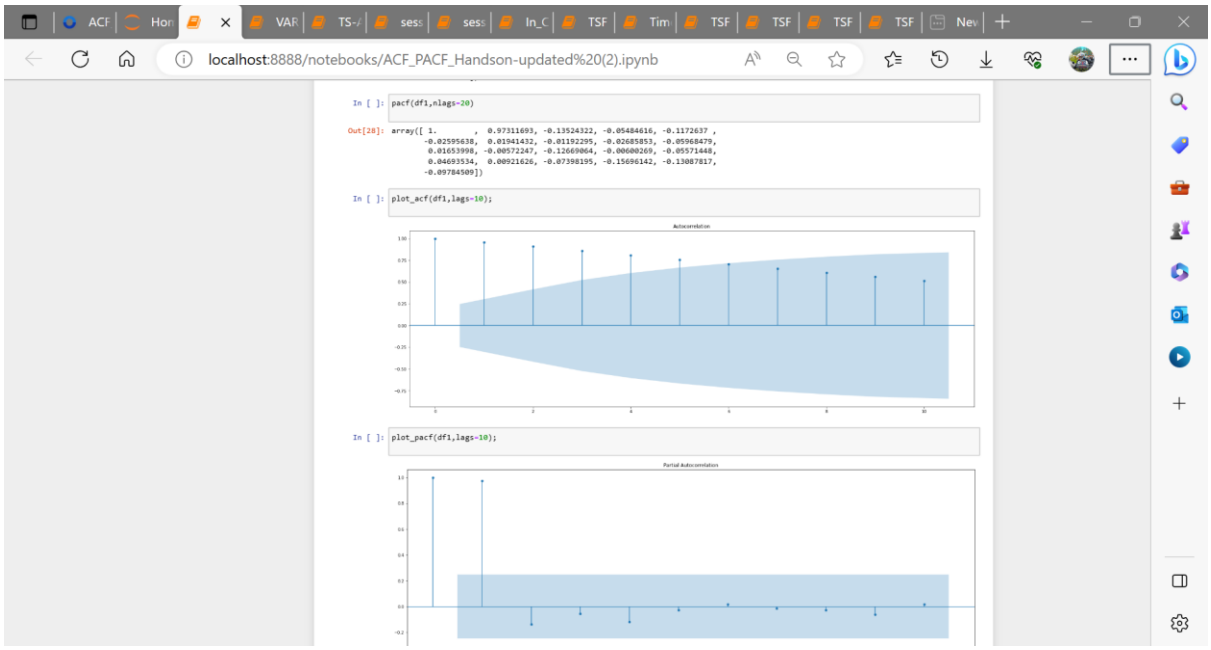
Random ACF, PACF





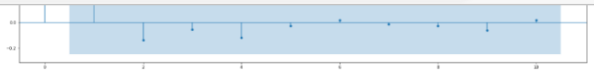
ACF with Series and Trend


localhost:8888/notebooks/ACF_PACF_Handson-updated%20(2).ipynb
ACF-PACF plot for series with trend
In [ ]: df1= pd.read_csv('GDPUS.csv')
df1.head()
Out[62]:
Year GDP
0 1929 821.8
1 1930 748.9
2 1931 691.3
3 1932 589.7
4 1933 587.1
In [ ]: df1=df1.set_index('Year')
df1.plot()
Out[63]: <AxesSubplot:xlabel='Year'>
In [ ]: acf(df1,nlags=20)
Out[29]: array([1. , 0.95767063, 0.90994842, 0.86070238, 0.8073258 ,
0.75396626, 0.70349868, 0.65473443, 0.60719985, 0.55974926,
0.51531969, 0.47320066, 0.42813631, 0.38489791, 0.34223893,
0.30302088, 0.26699204, 0.23121882, 0.19266056, 0.15197129,
0.10971482])
In [ ]: pacf(df1,nlags=20)
Out[28]: array([1. , 0.97311693, -0.13524322, -0.05484616, -0.1172637 ,


localhost:8888/notebooks/ACF_PACF_Handson-updated%20(2).ipynb
In [ ]: pacf(df1,nlags=20)
Out[28]: array([1. , 0.97311693, -0.13524322, -0.05484616, -0.1172637 ,
-0.02595638, 0.01941432, -0.01192295, -0.02685853, -0.05968479,
0.01653998, -0.00572247, -0.12669064, -0.00600269, -0.05571448,
0.04693534, 0.00921626, -0.07398195, -0.15696142, -0.13087817,
-0.09784509])
In [ ]: plot_acf(df1,lags=10);
Autocorrelation
In [ ]: plot_pacf(df1,lags=10);
Partial Autocorrelation

Seasonal

localhost:8888/notebooks/ACF_PACF_Handson-updated%20(2).ipynb



ACF and PACF for seasonal data

In [ ]:
```
df2= pd.read_csv('AirTemp.csv')
df2.head()
```

Out[33]:

	Year	Month	AvgTemp
0	1920	Jan	40.6
1	1920	Feb	40.8
2	1920	Mar	44.4
3	1920	Apr	46.7
4	1920	May	54.1

In [ ]:
```
df2.tail()
```

Out[34]:

	Year	Month	AvgTemp
235	1939	Aug	61.8
236	1939	Sep	58.2
237	1939	Oct	46.7
238	1939	Nov	46.6
239	1939	Dec	37.8

In [ ]:
```
date=pd.date_range(start='01/01/1920',end='31/12/1939',freq='M')
```

In [ ]:
```
df2['Month']=date
df2=df2.drop('Year',axis=1)
df2=df2.set_index('Month')
```

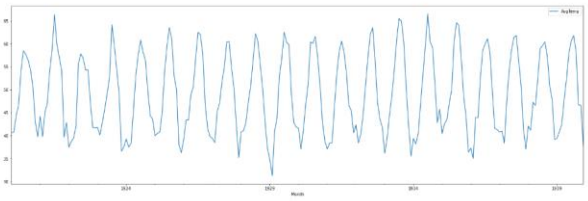
In [ ]:
```
df2.plot()
```

Out[39]:
```
<AxesSubplot:xlabel='Month'>
```

localhost:8888/notebooks/ACF_PACF_Handson-updated%20(2).ipynb

In [ ]:
```
df2.plot()
```

Out[39]:
```
<AxesSubplot:xlabel='Month'>
```



In [ ]:
```
acf(df2)
```

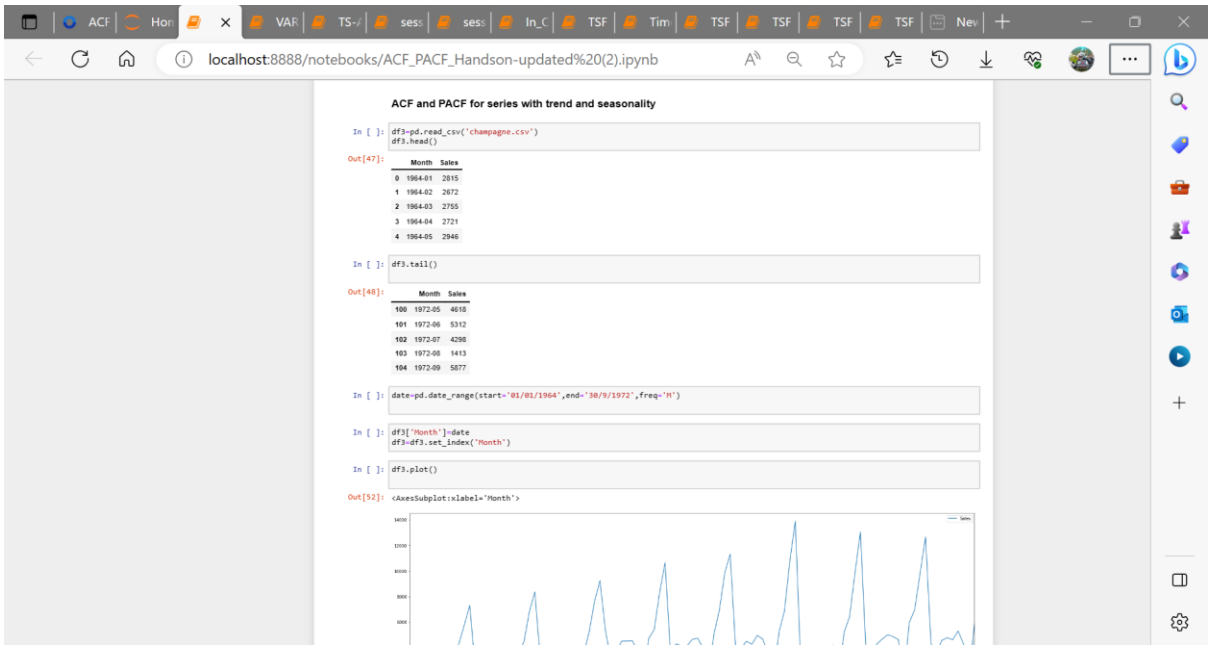
Out[40]:
```
array([ 1.        ,  0.80790526,  0.45248675, -0.01751263, -0.46420604,
       -0.76994762, -0.87586799, -0.75631615, -0.44503049, -0.01012765,
        0.4287941 ,  0.76499405,  0.88418607,  0.76992388,  0.43521273,
       -0.01110211, -0.44055642, -0.73441497, -0.84416434, -0.72553741,
       -0.43139643, -0.01523368,  0.41976175,  0.73227741,  0.8660013 ,
        0.7274997 ,  0.40601206, -0.02150236, -0.43011936, -0.71382109,
       -0.79664795, -0.6866535 , -0.40649436, -0.01605604,  0.38012585,
        0.69100116,  0.8097235 ,  0.70242215,  0.39449846, -0.00218798,
       -0.40009647])
```

In [ ]:
```
pacf(df2)
```

Out[41]:
```
array([ 1.        ,  0.81128562, -0.59065522, -0.57230108, -0.38994045,
       -0.23943022, -0.28545341, -0.27998011, -0.21257999,  0.0131737 ,
        0.14094934,  0.31502474,  0.21896989,  0.25542659,  0.05184143,
       -0.06315817, -0.08001226,  0.03759164, -0.18472002, -0.12794006,
       -0.23702625, -0.06083874,  0.22273062,  0.22081597,  0.51347757,
        0.16349955,  0.03036024, -0.20450648, -0.4756613 , -0.90449509,
       -6.09840056,  1.13090207,  0.89330965,  4.64842046, -1.22270799,
       -0.54471989, -0.32306872, -0.18700824, -0.0381359 ,  0.35999596,
        0.29658174])
```

In [ ]:
```
plot_acf(df2,lags=15);
```

With Series and Trend and Seasonality



ACF and PACF for series with trend and seasonality

In [ ]:
```
df3=pd.read_csv('champagne.csv')
df3.head()
```

Out[47]:

	Month	Sales
0	1964-01	2815
1	1964-02	2672
2	1964-03	2755
3	1964-04	2721
4	1964-05	2946

In [ ]:
```
df3.tail()
```

Out[48]:

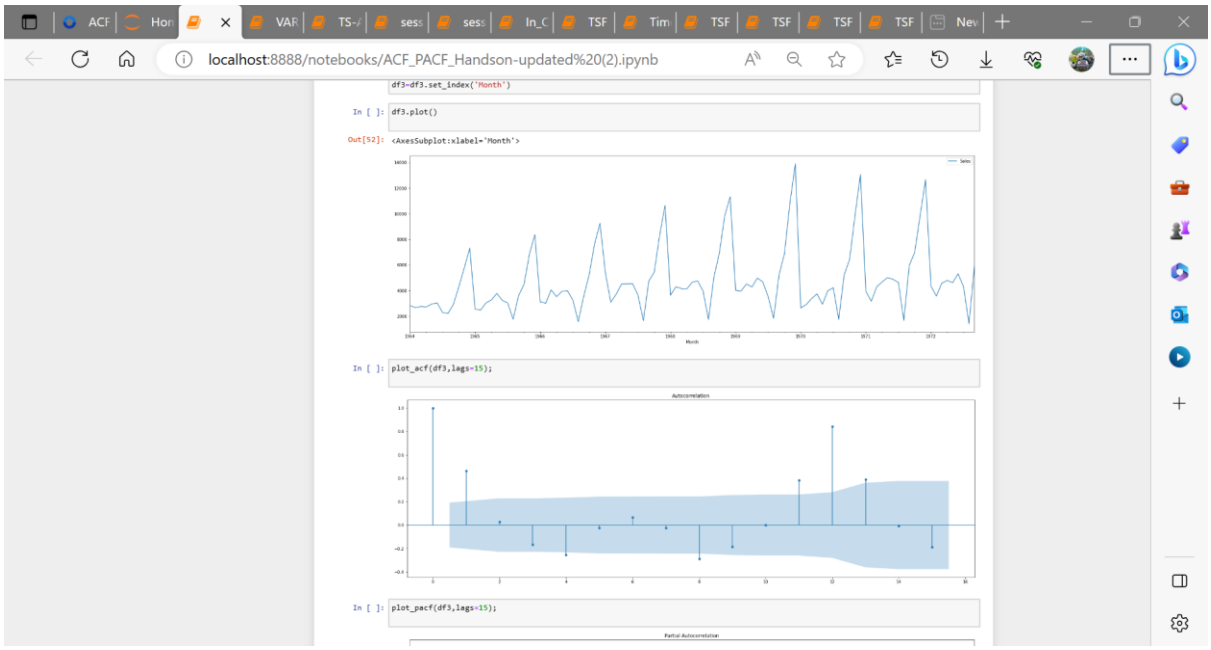
	Month	Sales
100	1972-05	4618
101	1972-06	5312
102	1972-07	4298
103	1972-08	1413
104	1972-09	5877

In [ ]:
```
date=pd.date_range(start='01/01/1964',end='30/9/1972',freq='M')
```

In [ ]:
```
df3['Month']=date
df3=df3.set_index('Month')
```

In [ ]:
```
df3.plot()
```

Out[52]: <AxesSubplot:xlabel='Month'>

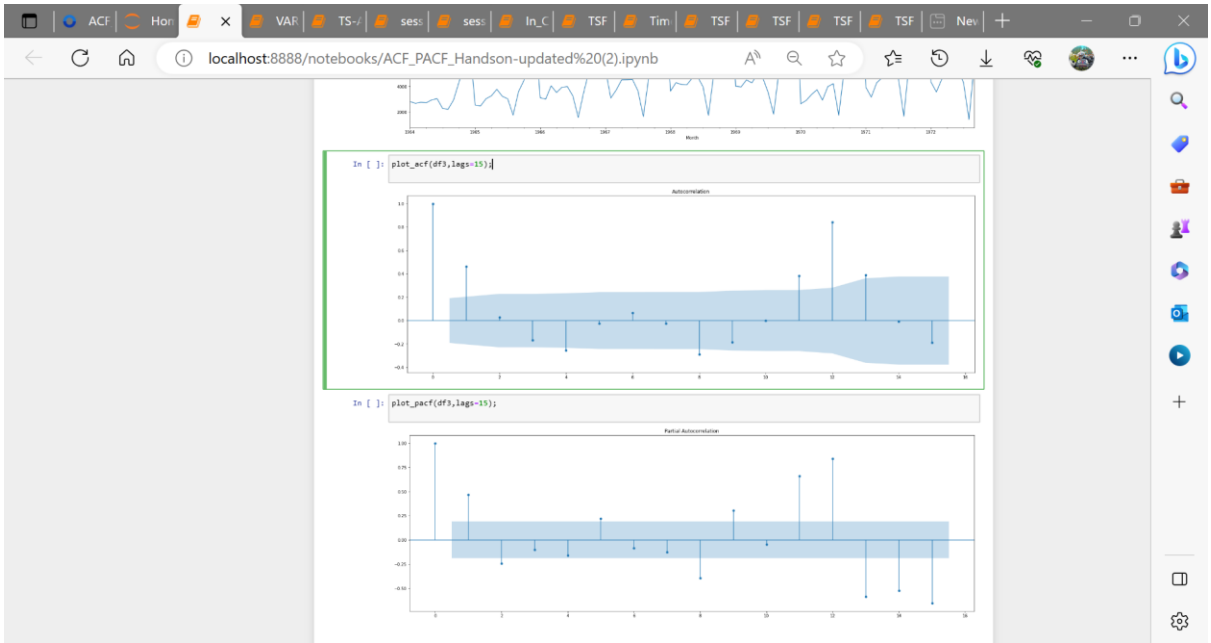


In [ ]:
```
df3.plot()
```

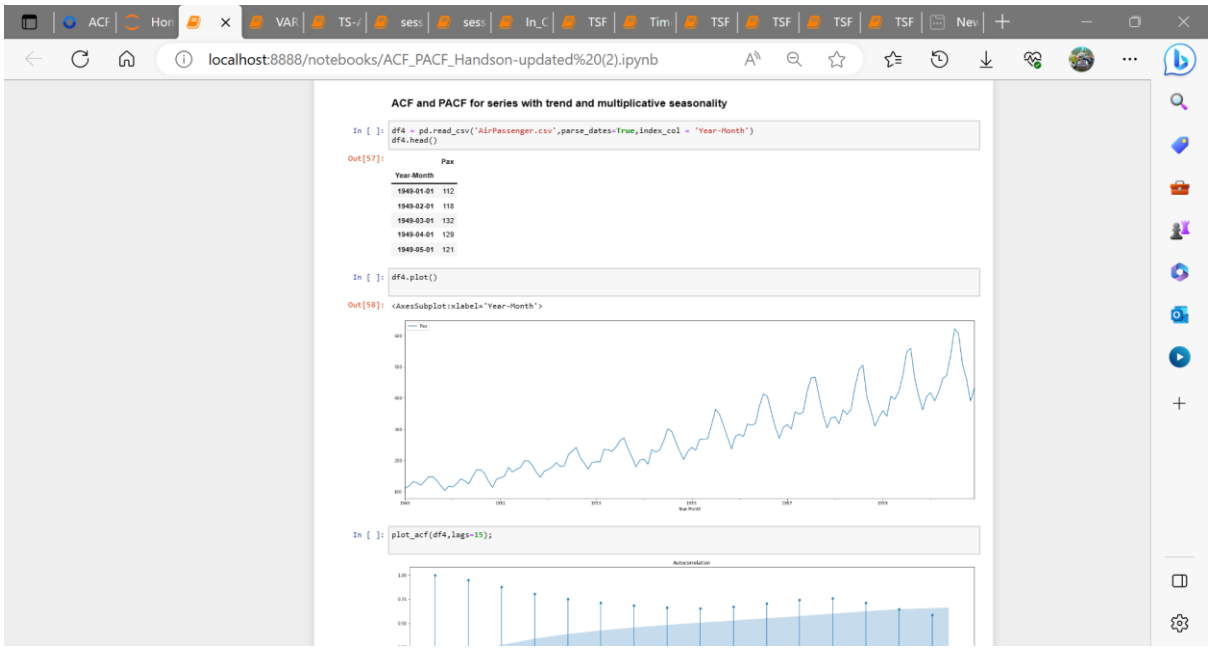
Out[52]: <AxesSubplot:xlabel='Month'>

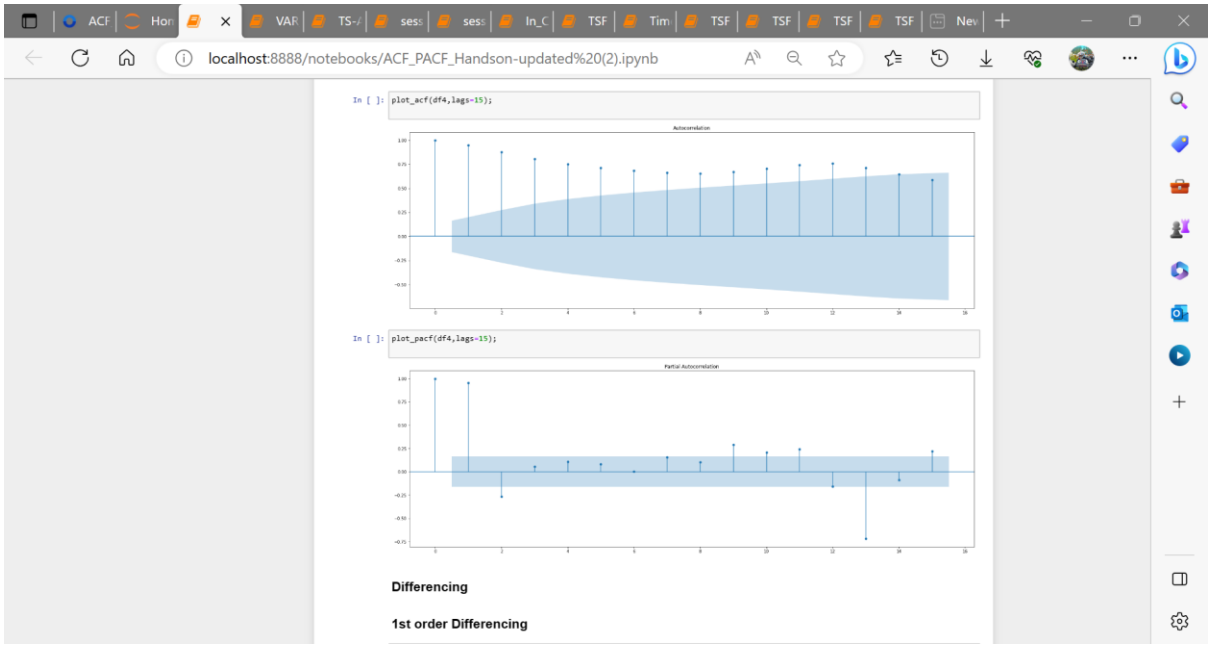
In [ ]:
```
plot_acf(df3,lags=15);
```

In [ ]:
```
plot_pacf(df3,lags=15);
```

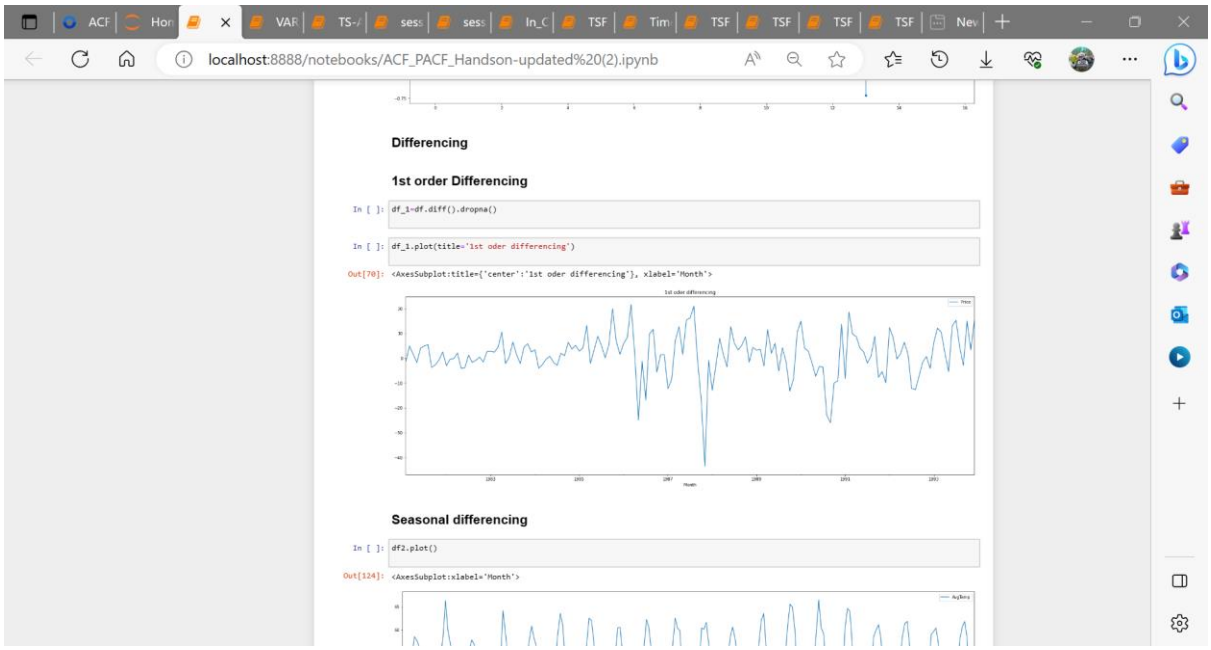


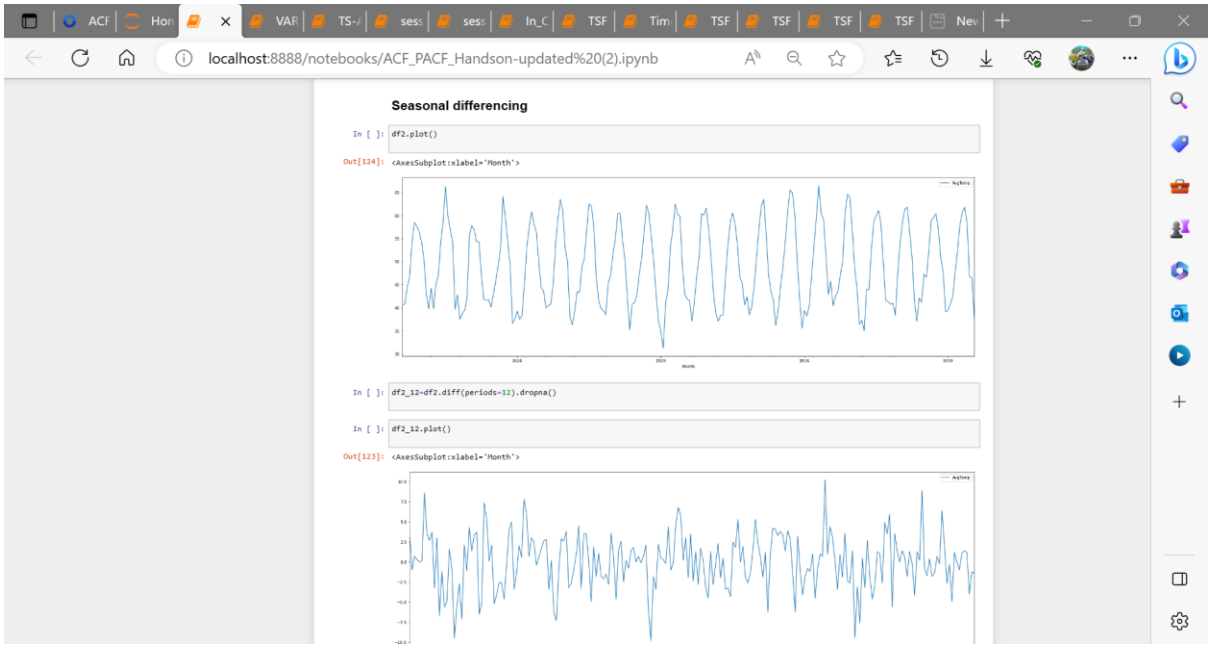
With Series and Trend and Multiplicative seasonalirt





Differencing





Seasonal differencing

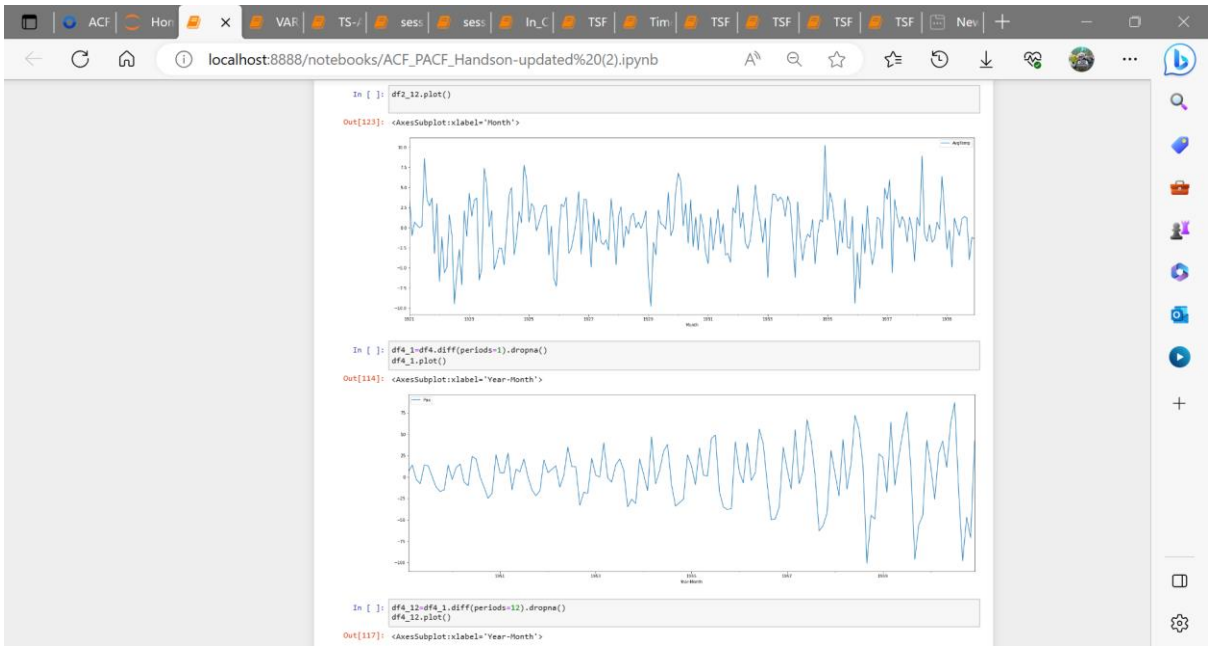
In [ ]:
```
df2.plot()
```

Out[124]: <AxesSubplot:xlabel='Month'>

In [ ]:
```
df2_12=df2.diff(periods=12).dropna()
```

In [ ]:
```
df2_12.plot()
```

Out[123]: <AxesSubplot:xlabel='Month'>



In [ ]:
```
df2_12.plot()
```

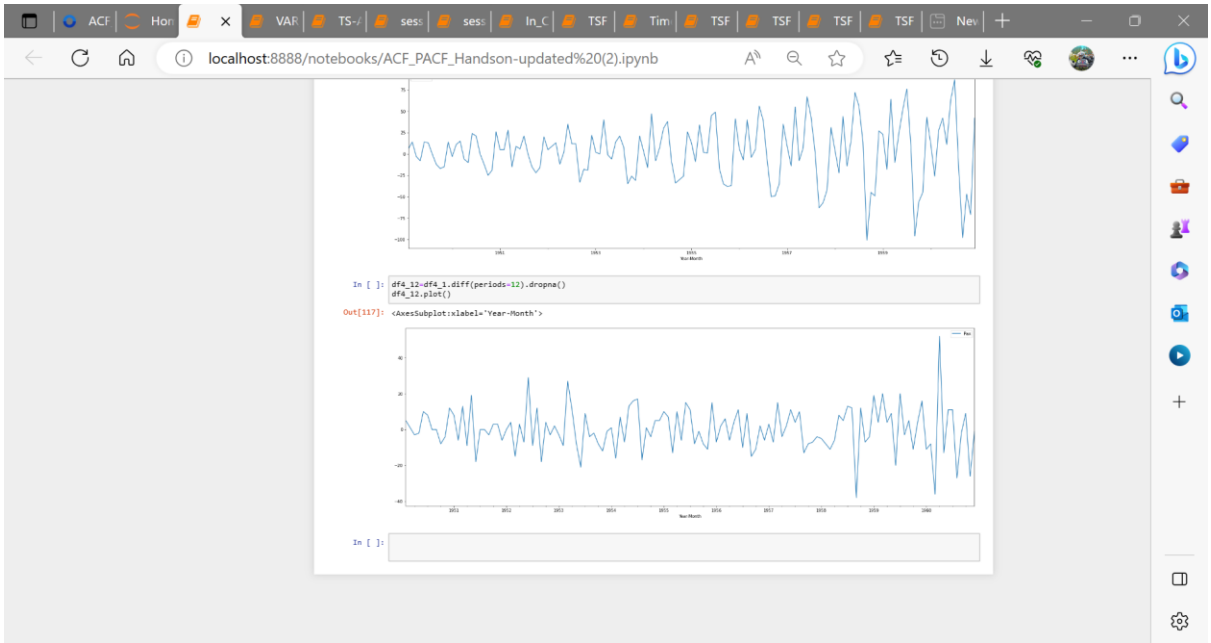
Out[123]: <AxesSubplot:xlabel='Month'>

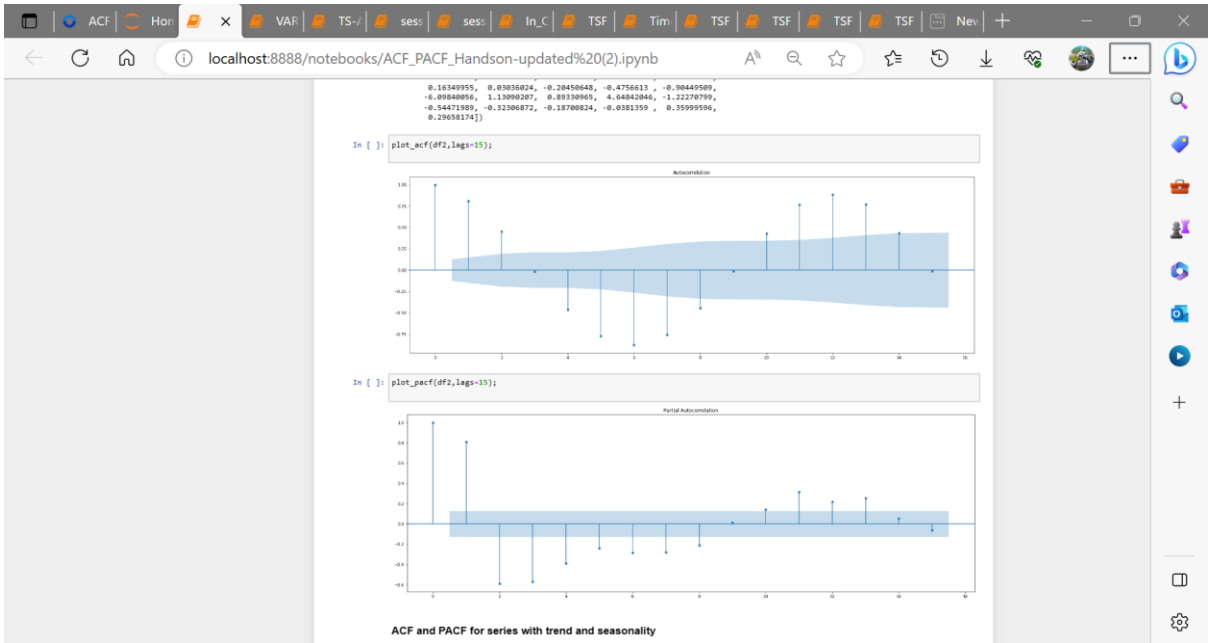
In [ ]:
```
df4_1=df4.diff(periods=1).dropna()
df4_1.plot()
```

Out[114]: <AxesSubplot:xlabel='Year-Month'>

In [ ]:
```
df4_12=df4_1.diff(periods=12).dropna()
df4_12.plot()
```

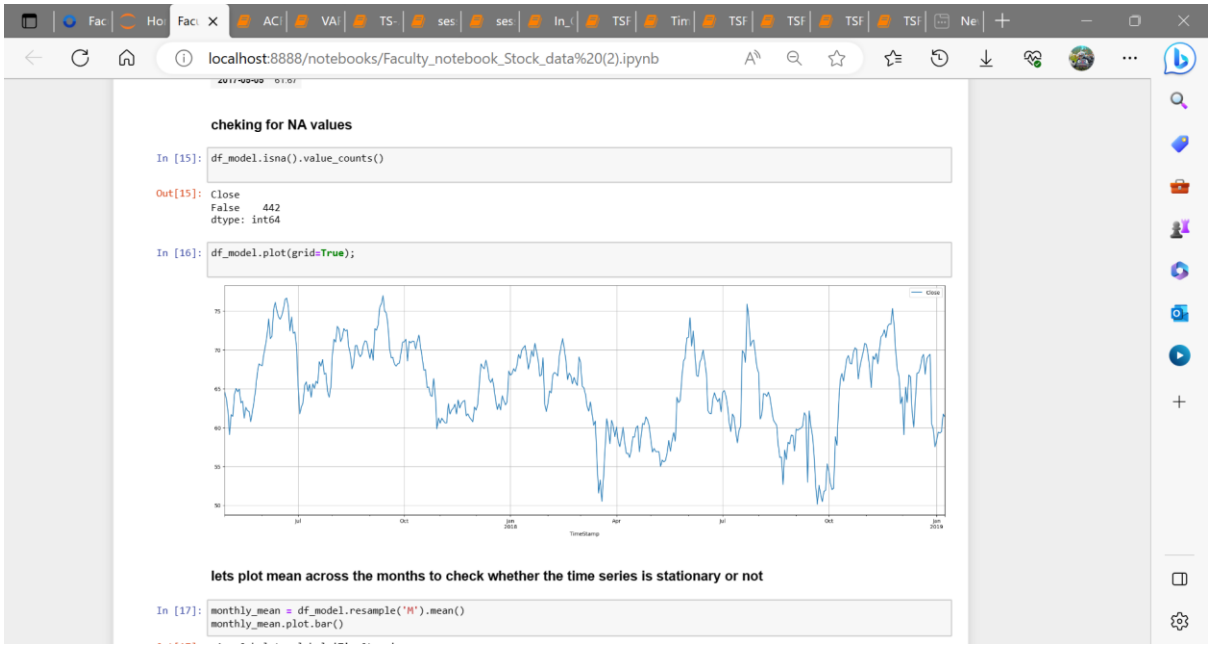
Out[117]: <AxesSubplot:xlabel='Year-Month'>



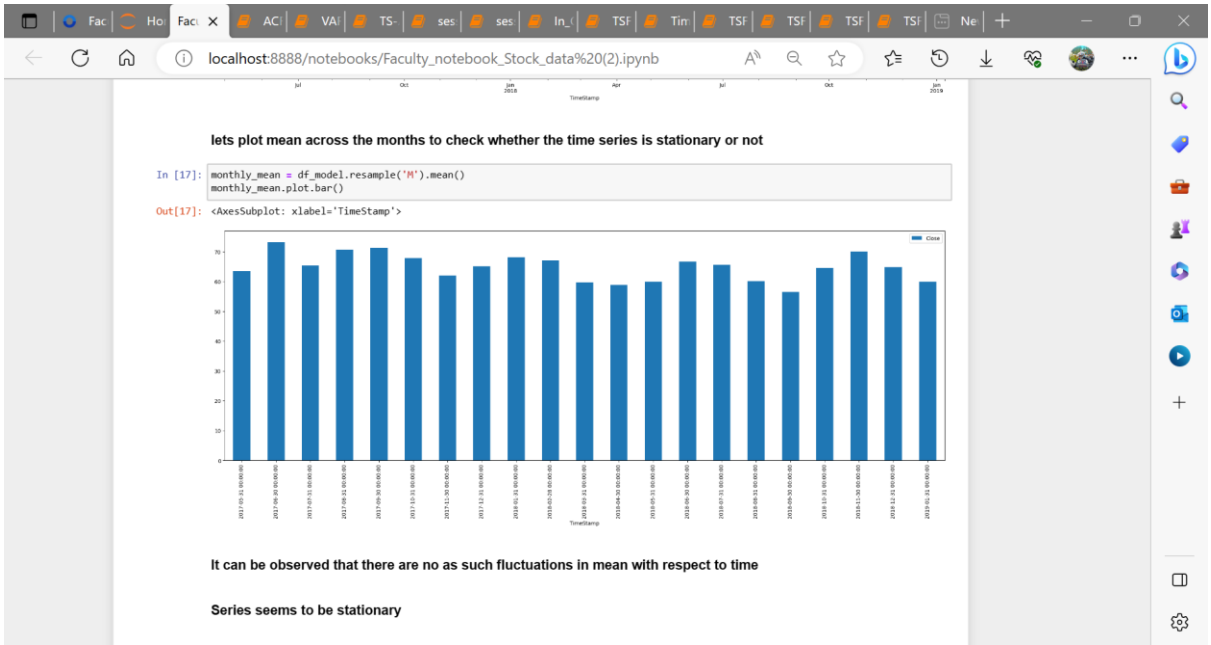


ACF and PACF for series with trend and seasonality

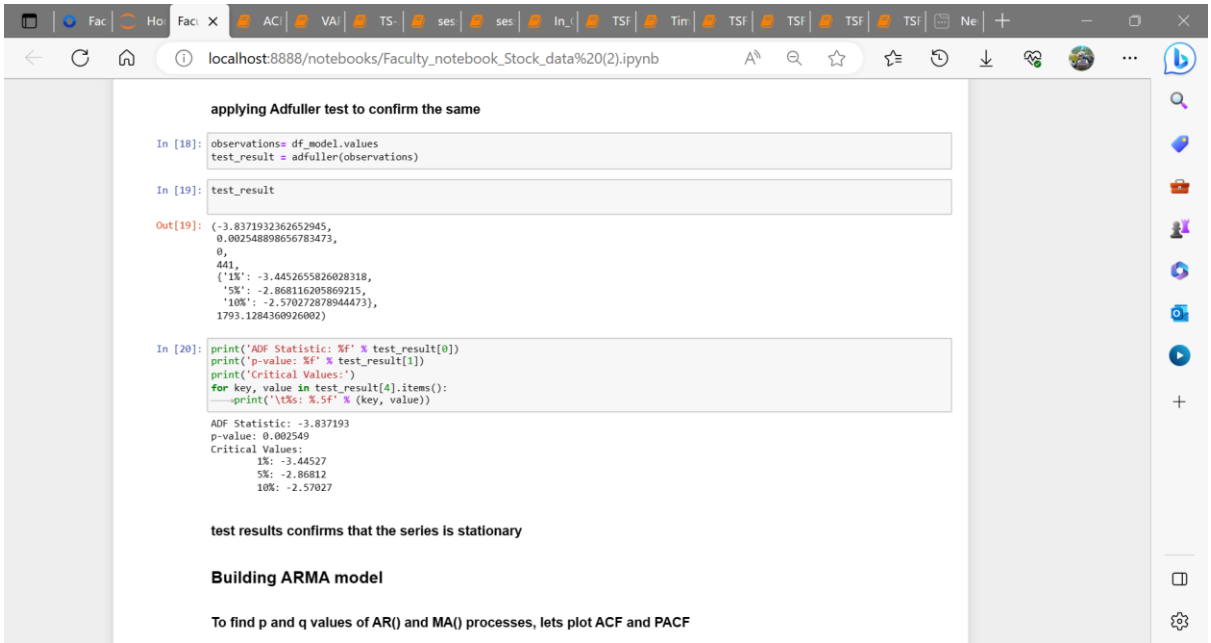
Checking for Null values



Stationary as mean is ok



Adfuller Test



applying Adfuller test to confirm the same

In [18]:
```
observations= df_model.values
test_result = adfuller(observations)
```

In [19]:
```
test_result
```

Out[19]:
```
(-3.8371932362652945,
 0.002548898656783473,
 0,
 441,
 {'1%': -3.4452655826028318,
  '5%': -2.868116205869215,
  '10%': -2.570272878944473},
 1793.1284360926002)
```

In [20]:
```
print('ADF Statistic: %f' % test_result[0])
print('p-value: %f' % test_result[1])
print('Critical Values:')
for key, value in test_result[4].items():
    print('\t%s: %.5f' % (key, value))
```

```
ADF Statistic: -3.837193
p-value: 0.002549
Critical Values:
	1%: -3.44527
	5%: -2.86812
	10%: -2.57027
```

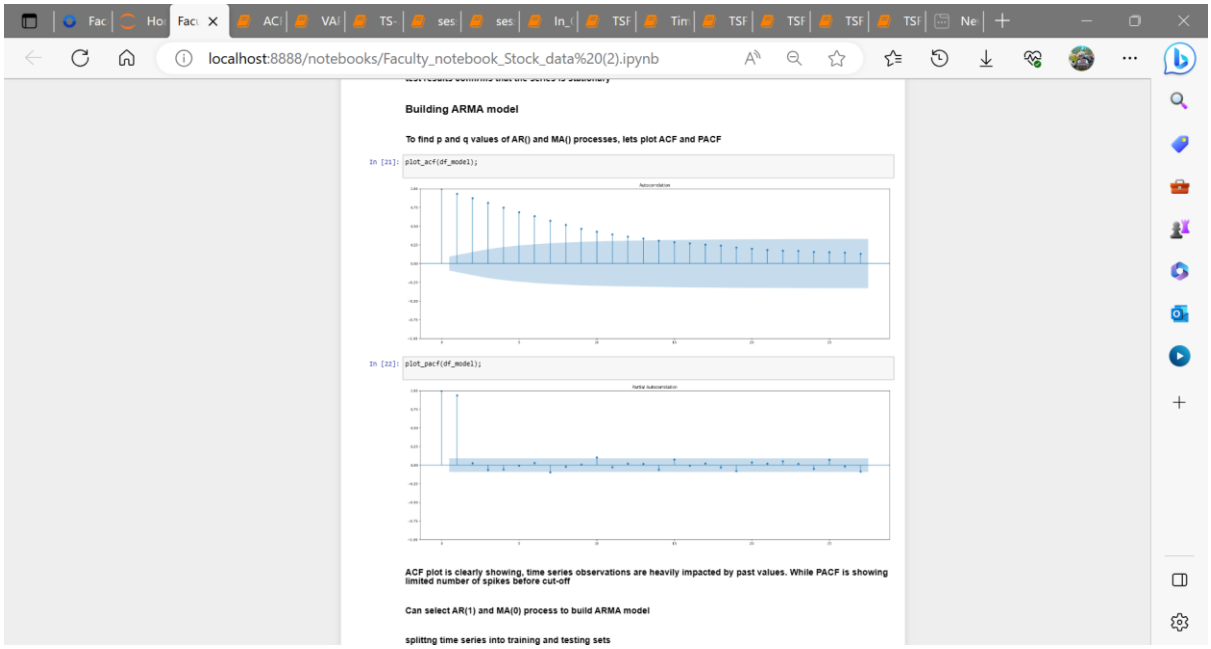
test results confirms that the series is stationary

Building ARMA model

To find p and q values of AR() and MA() processes, lets plot ACF and PACF

-3.83 list less than 1. 5. 10 % values

ARMA



Building ARMA model

To find p and q values of AR() and MA() processes, lets plot ACF and PACF

In [21]:
```
plot_acf(df_model);
```

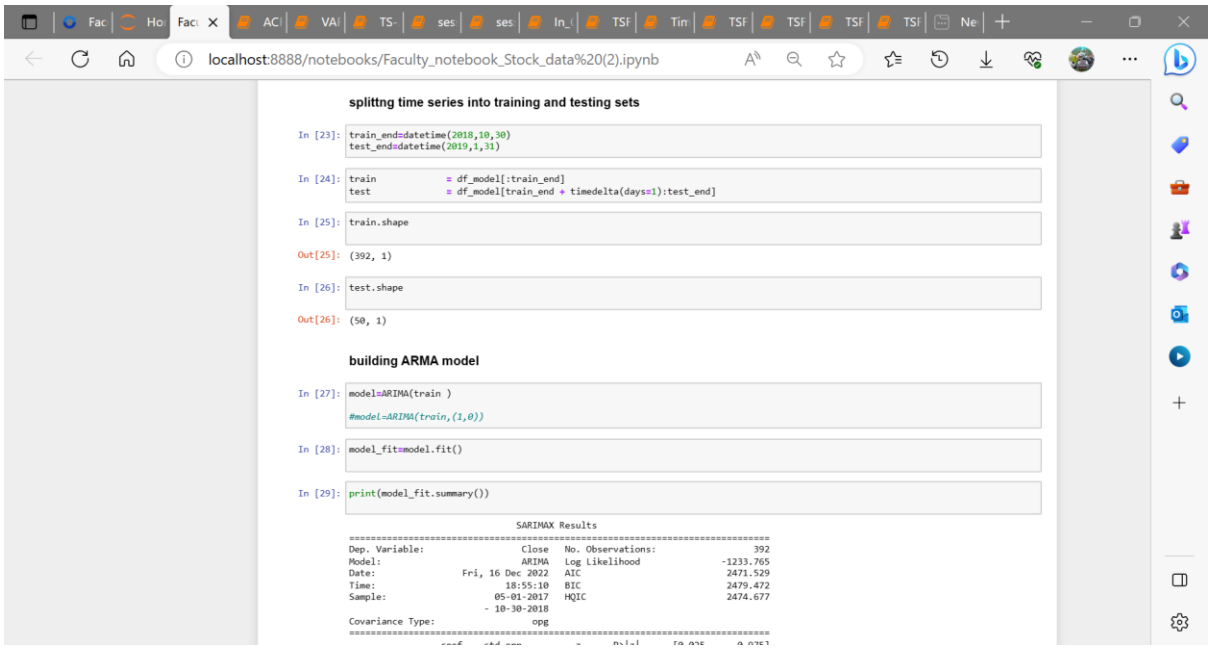
In [22]:
```
plot_pacf(df_model);
```

ACF plot is clearly showing, time series observations are heavily impacted by past values. While PACF is showing limited number of spikes before cut-off

Can select AR(1) and MA(0) process to build ARMA model

splittng time series into training and testing sets

SPLITTING TS to Train and test



splittng time series into training and testing sets

In [23]:
```
train_end=datetime(2018,10,30)
test_end=datetime(2019,1,31)
```

In [24]:
```
train              = df_model[:train_end]
test               = df_model[train_end + timedelta(days=1):test_end]
```

In [25]:
```
train.shape
```

Out[25]: (392, 1)

In [26]:
```
test.shape
```

Out[26]: (50, 1)

building ARMA model

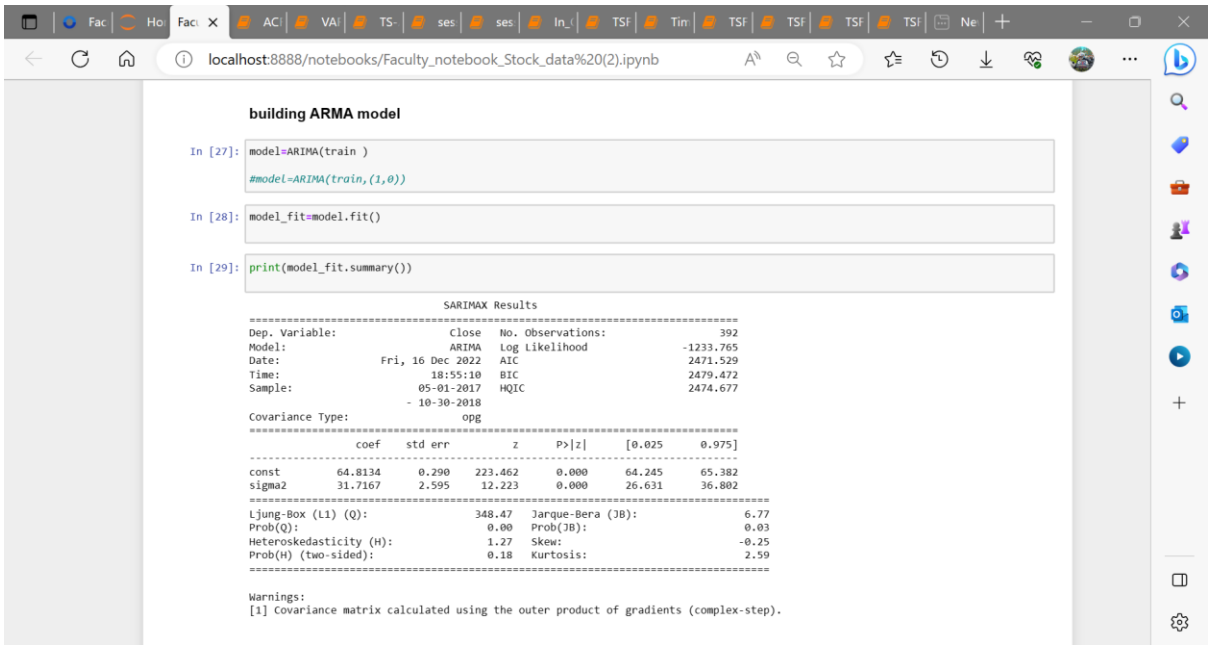
In [27]:
```
model=ARIMA(train )

#model=ARIMA(train,(1,0))
```

In [28]:
```
model_fit=model.fit()
```

In [29]:
```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                  392
Model:                          ARIMA   Log Likelihood               -1233.765
Date:                Fri, 16 Dec 2022   AIC                           2471.529
Time:                        18:55:10   BIC                           2479.472
Sample:                    05-01-2017   HQIC                          2474.677
                         - 10-30-2018
Covariance Type:                  opg
==============================================================================
```

Build ARMA



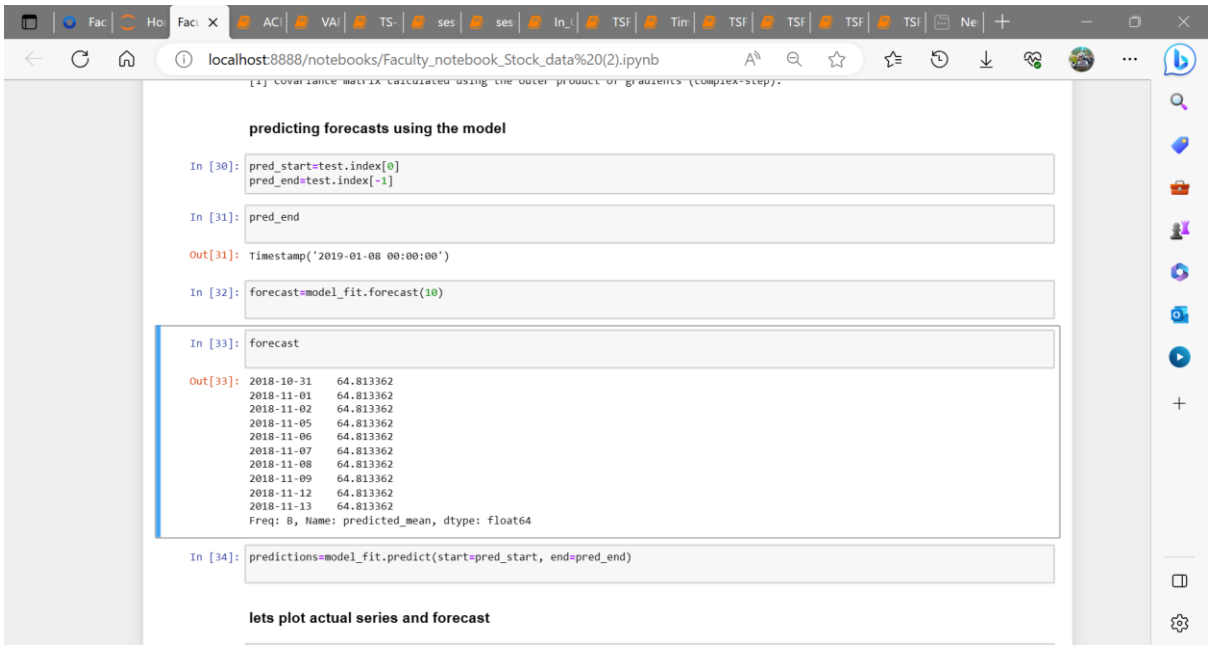
building ARMA model

In [27]:
```
model=ARIMA(train )

#model=ARIMA(train,(1,0))
```

In [28]:
```
model_fit=model.fit()
```

In [29]:
```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                  392
Model:                          ARIMA   Log Likelihood               -1233.765
Date:                Fri, 16 Dec 2022   AIC                           2471.529
Time:                        18:55:10   BIC                           2479.472
Sample:                    05-01-2017   HQIC                          2474.677
                         - 10-30-2018
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         64.8134      0.290    223.462      0.000      64.245      65.382
sigma2        31.7167      2.595     12.223      0.000      26.631      36.802
===================================================================================
Ljung-Box (L1) (Q):                 348.47   Jarque-Bera (JB):                 6.77
Prob(Q):                              0.00   Prob(JB):                         0.03
Heteroskedasticity (H):               1.27   Skew:                            -0.25
Prob(H) (two-sided):                  0.18   Kurtosis:                         2.59
===================================================================================

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



predicting forecasts using the model

In [30]:
```
pred_start=test.index[0]
pred_end=test.index[-1]
```

In [31]:
```
pred_end
```

Out[31]:
```
Timestamp('2019-01-08 00:00:00')
```

In [32]:
```
forecast=model_fit.forecast(10)
```

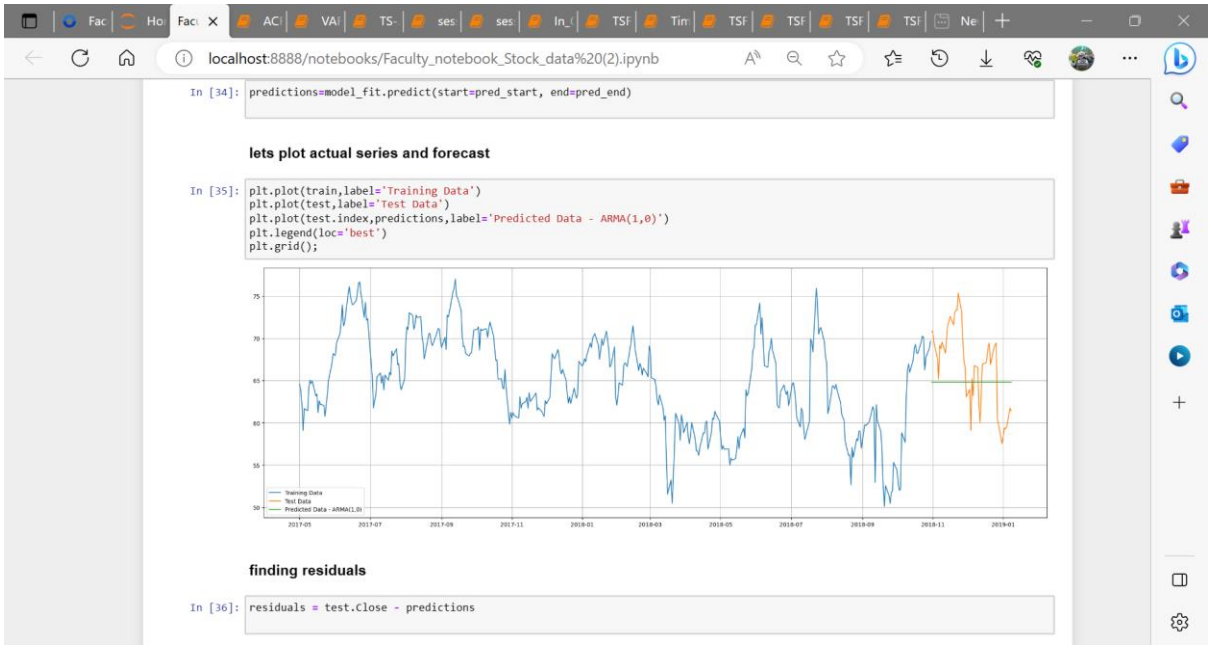
In [33]:
```
forecast
```

Out[33]:
```
2018-10-31    64.813362
2018-11-01    64.813362
2018-11-02    64.813362
2018-11-05    64.813362
2018-11-06    64.813362
2018-11-07    64.813362
2018-11-08    64.813362
2018-11-09    64.813362
2018-11-12    64.813362
2018-11-13    64.813362
Freq: B, Name: predicted_mean, dtype: float64
```

In [34]:
```
predictions=model_fit.predict(start=pred_start, end=pred_end)
```

lets plot actual series and forecast

forecasting



In [34]:
```
predictions=model_fit.predict(start=pred_start, end=pred_end)
```

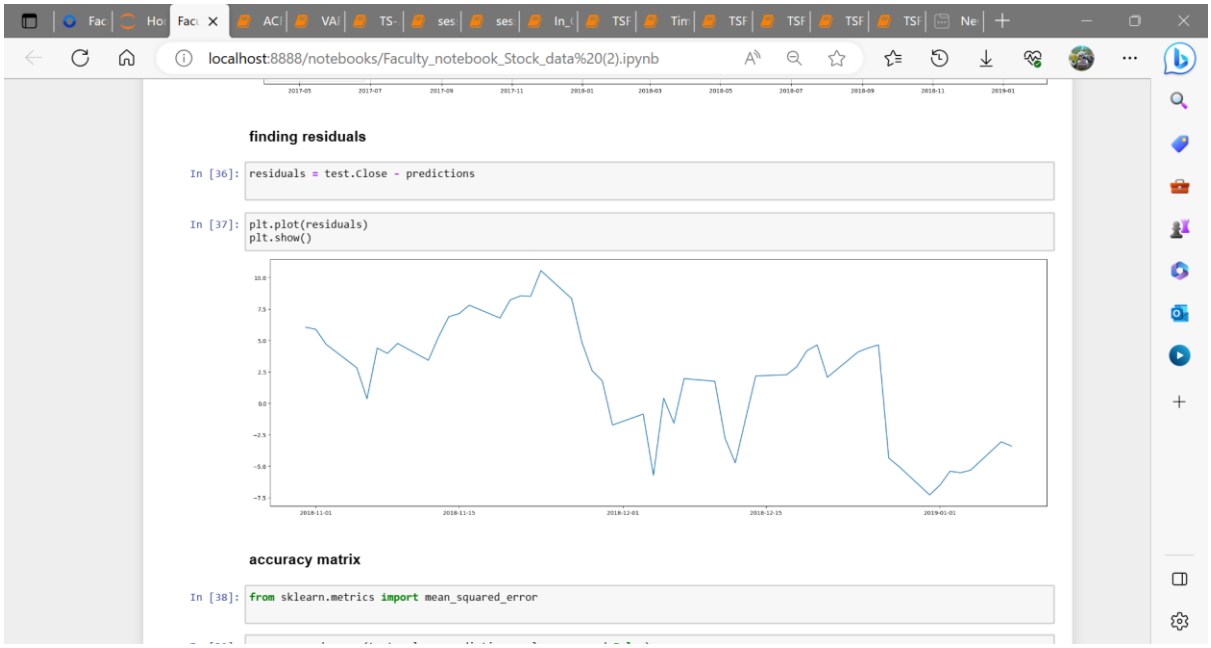
lets plot actual series and forecast

In [35]:
```
plt.plot(train,label='Training Data')
plt.plot(test,label='Test Data')
plt.plot(test.index,predictions,label='Predicted Data - ARMA(1,0)')
plt.legend(loc='best')
plt.grid();
```

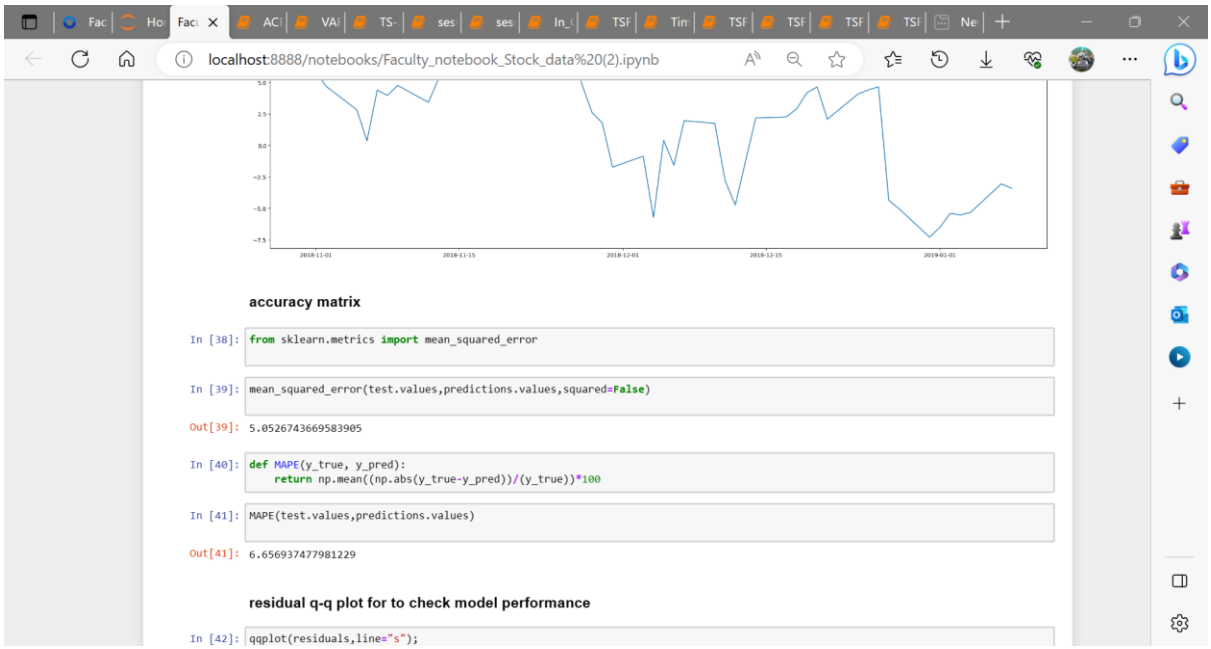

finding residuals

In [36]:
```
residuals = test.Close - predictions
```

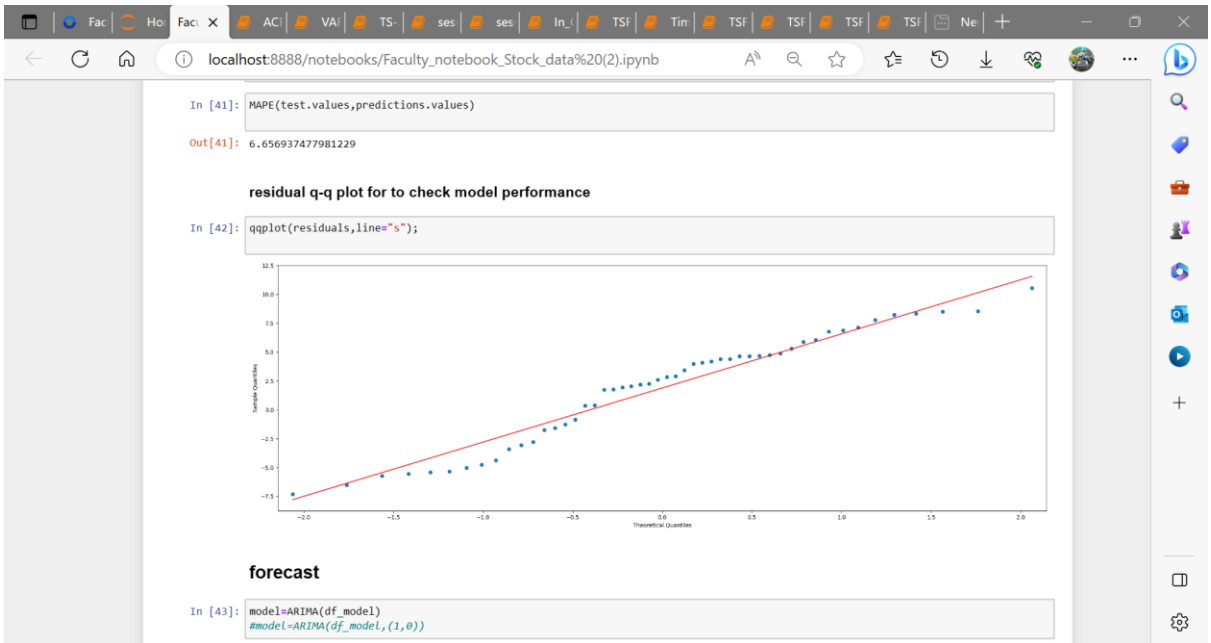
Finding Residuals



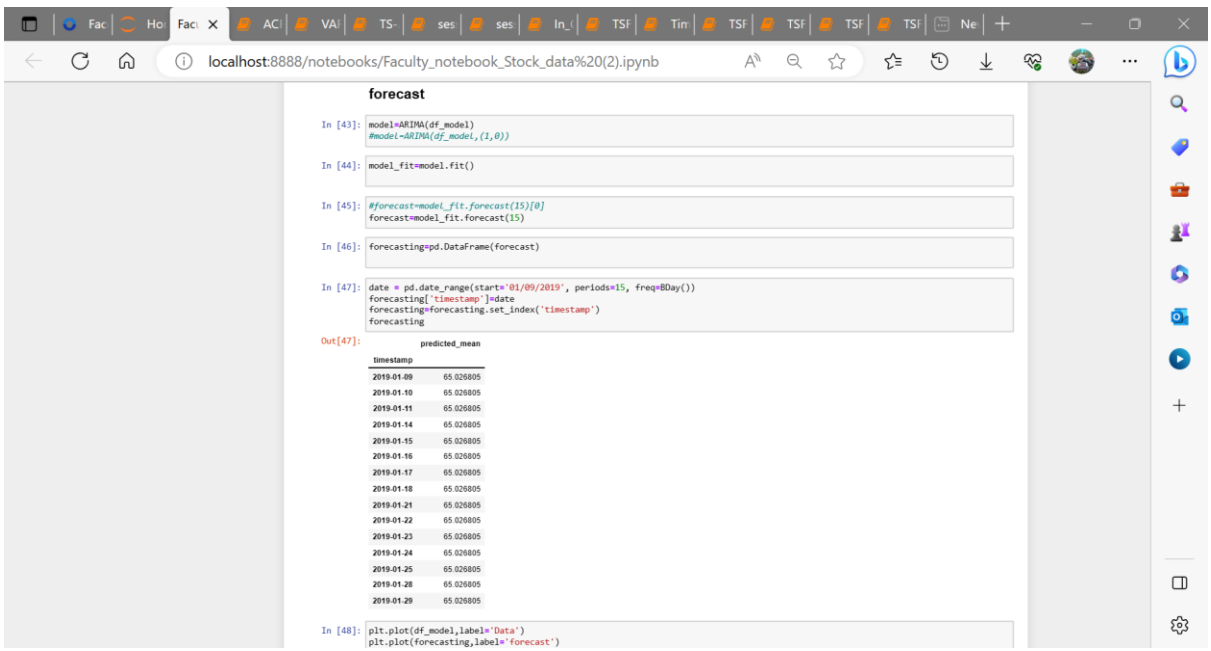
Accuraccy Matrix



Residual q-q plot



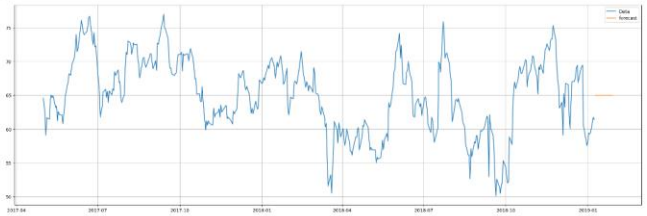
Forecast on model built



localhost:8888/notebooks/Faculty_notebook_Stock_data%20(2).ipynb

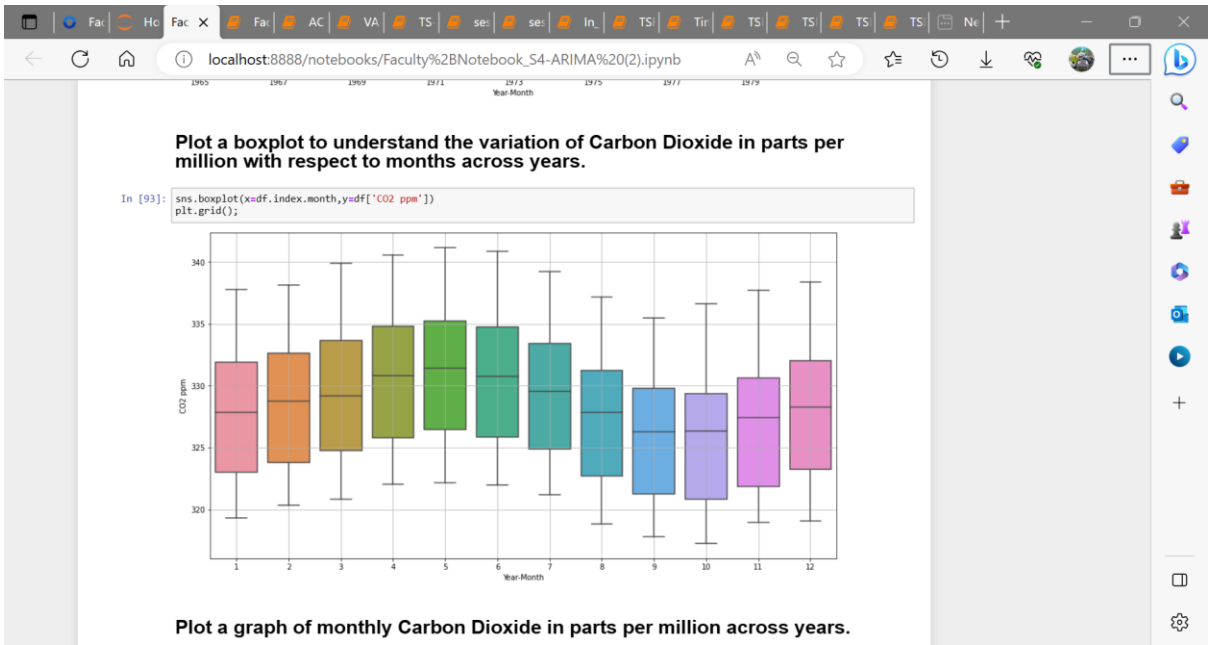
2019-01-14	65.026805
2019-01-15	65.026805
2019-01-16	65.026805
2019-01-17	65.026805
2019-01-18	65.026805
2019-01-21	65.026805
2019-01-22	65.026805
2019-01-23	65.026805
2019-01-24	65.026805
2019-01-25	65.026805
2019-01-28	65.026805
2019-01-29	65.026805

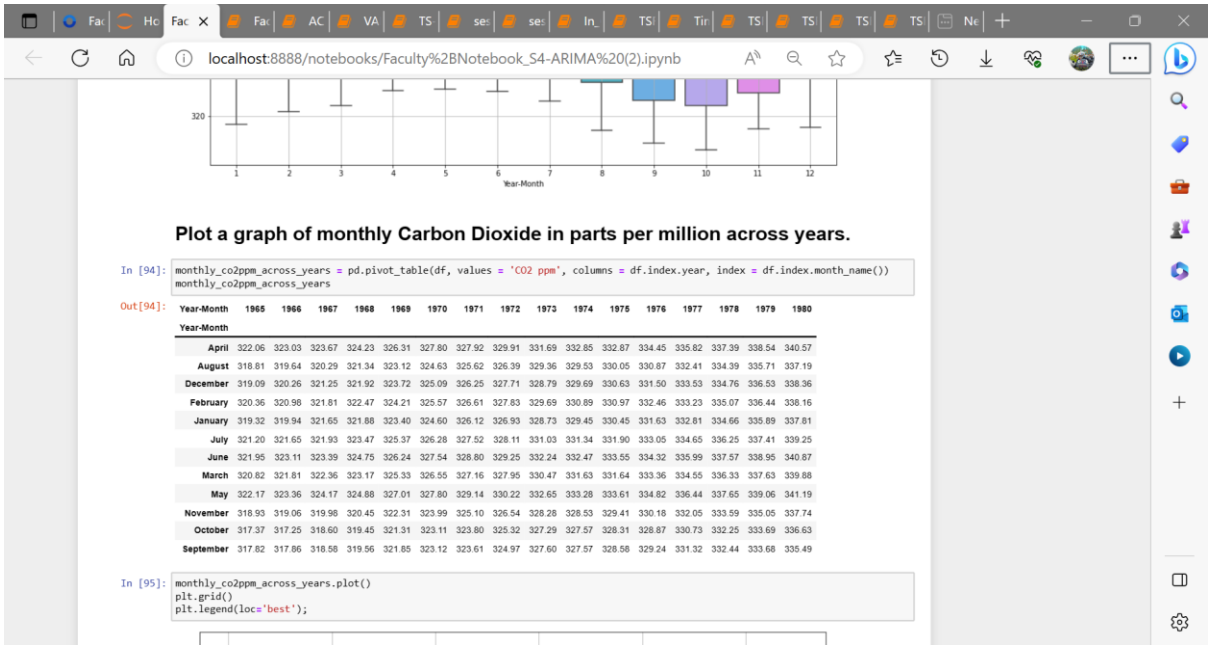
In [48]:
```
plt.plot(df_model,label='Data')
plt.plot(forecasting,label='forecast')
plt.legend(loc='best')
plt.grid();
```



End

Box plot



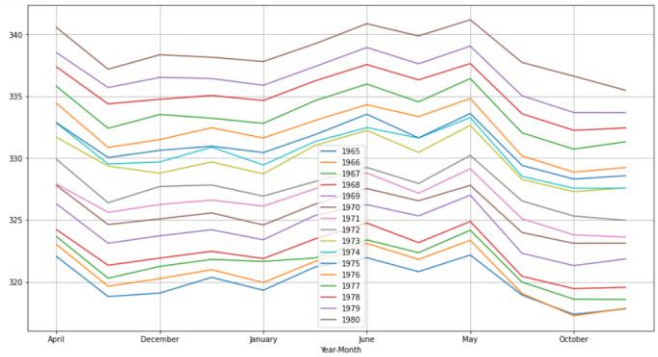


Year-Month	1965	1966	1967	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980
April	322.06	323.03	323.67	324.23	326.31	327.80	327.92	329.91	331.69	332.85	332.87	334.45	335.82	337.39	338.54	340.57
August	318.81	319.64	320.29	321.34	323.12	324.63	325.62	326.39	329.36	329.53	330.05	330.87	332.41	334.39	335.71	337.19
December	319.09	320.26	321.25	321.92	323.72	325.09	326.25	327.71	328.79	329.69	330.63	331.50	333.53	334.76	336.53	338.36
February	320.36	320.98	321.81	322.47	324.21	325.57	326.61	327.83	329.69	330.89	330.97	332.46	333.23	335.07	336.44	338.16
January	319.32	319.94	321.65	321.88	323.40	324.60	326.12	326.93	328.73	329.45	330.45	331.63	332.81	334.66	335.89	337.81
July	321.20	321.65	321.93	323.47	325.37	326.28	327.52	328.11	331.03	331.34	331.90	333.05	334.65	336.25	337.41	339.25
June	321.95	323.11	323.39	324.75	326.24	327.54	328.80	329.25	332.24	332.47	333.55	334.32	335.99	337.57	338.95	340.87
March	320.82	321.81	322.36	323.17	325.33	326.55	327.16	327.95	330.47	331.63	331.64	333.36	334.55	336.33	337.63	339.88
May	322.17	323.36	324.17	324.88	327.01	327.80	329.14	330.22	332.65	333.28	333.61	334.82	336.44	337.65	339.06	341.19
November	318.93	319.06	319.98	320.45	322.31	323.99	325.10	326.54	328.28	328.53	329.41	330.18	332.05	333.59	335.05	337.74
October	317.37	317.25	318.60	319.45	321.31	323.11	323.80	325.32	327.29	327.57	328.31	328.87	330.73	332.25	333.69	336.63
September	317.82	317.86	318.58	319.56	321.85	323.12	323.61	324.97	327.60	327.57	328.58	329.24	331.32	332.44	333.68	335.49

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

May	322.17	323.36	324.17	324.88	327.01	327.80	329.14	330.22	332.65	333.28	333.61	334.82	336.44	337.65	339.06	341.19
November	318.93	319.06	319.98	320.45	322.31	323.99	325.10	326.54	328.28	328.53	329.41	330.18	332.05	333.59	335.05	337.74
October	317.37	317.25	318.60	319.45	321.31	323.11	323.80	325.32	327.29	327.57	328.31	328.87	330.73	332.25	333.69	336.63
September	317.82	317.86	318.58	319.56	321.85	323.12	323.61	324.97	327.60	327.57	328.58	329.24	331.32	332.44	333.68	335.49

In [95]:
```
monthly_co2ppm_across_years.plot()
plt.grid()
plt.legend(loc='best');
```

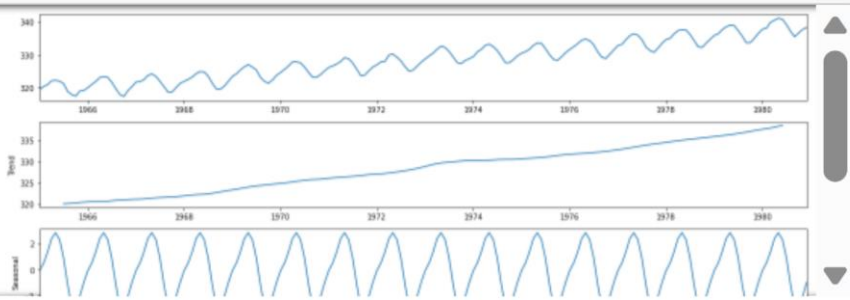


Decompose the Time Series to understand the various components.

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

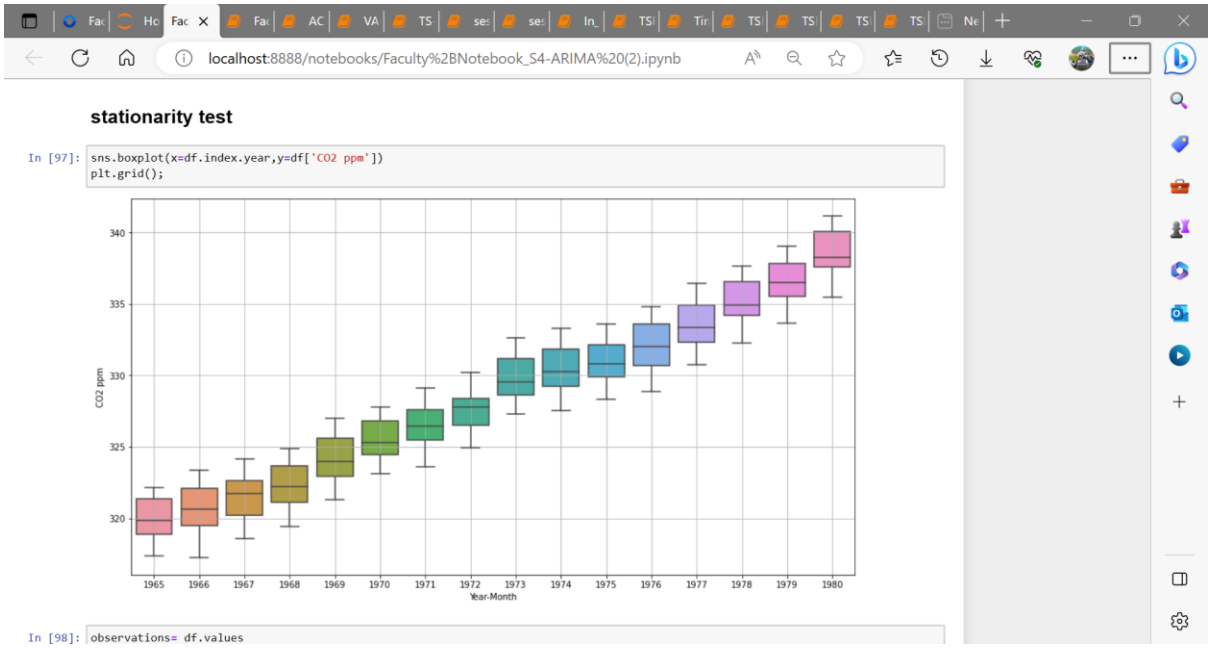
Decompose the Time Series to understand the various components.

In [96]:
```
decomposition = seasonal_decompose(df,model='additive')
decomposition.plot();
```

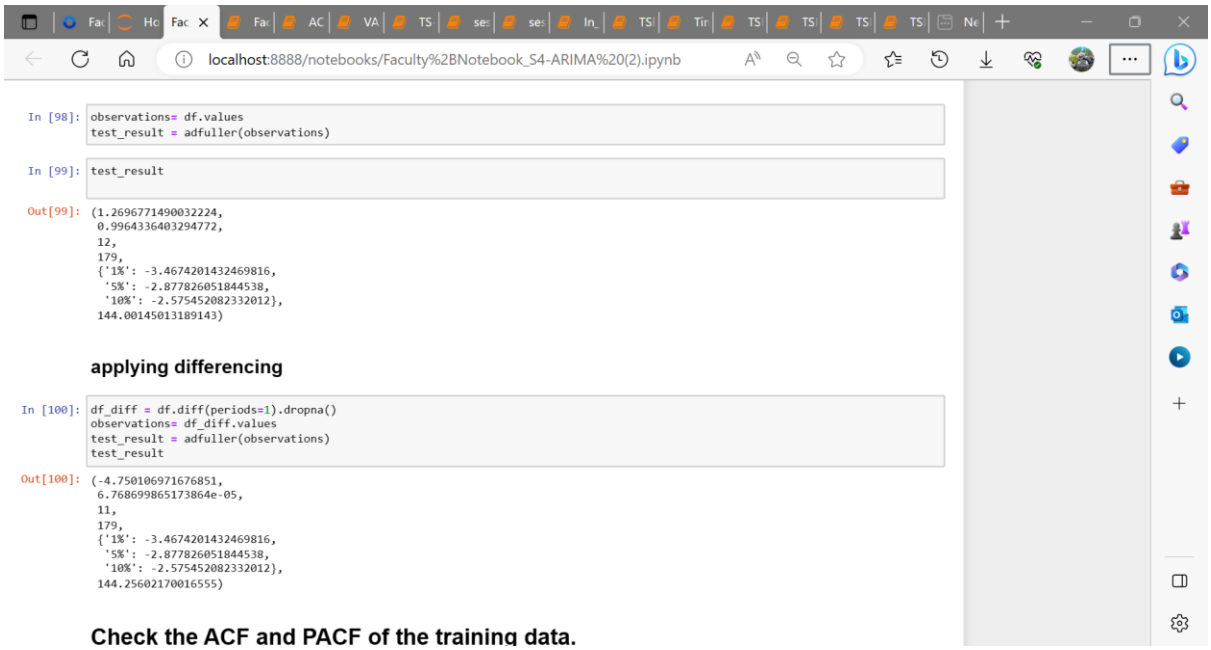


stationarity test

In [97]:
```
sns.boxplot(x=df.index.year,y=df['CO2 ppm'])
plt.grid();
```



Note after applying differencing the value has reduced from 1.26 to -4. Now the TS is stationary



Add inference for Null hypothesis

Selecting model with lowest AKAIke Information Criteria

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [106]:
```
train              = df[:train_end]
test               = df[train_end + timedelta(days=1):test_end]
```

Selecting an order of ARIMA model for data with the lowest Akaike Information Criteria (AIC).

In [107]:
```
import itertools
p = q = range(0, 4)
d= range(1,2)
pdq = list(itertools.product(p, d, q))
print('parameter combinations for the Model')
for i in range(1,len(pdq)):
    print('Model: {}'.format(pdq[i]))
```

```
parameter combinations for the Model
Model: (0, 1, 1)
Model: (0, 1, 2)
Model: (0, 1, 3)
Model: (1, 1, 0)
Model: (1, 1, 1)
Model: (1, 1, 2)
Model: (1, 1, 3)
Model: (2, 1, 0)
Model: (2, 1, 1)
Model: (2, 1, 2)
Model: (2, 1, 3)
Model: (3, 1, 0)
Model: (3, 1, 1)
Model: (3, 1, 2)
Model: (3, 1, 3)
```

In [108]:
```
df0bj1 = pd.DataFrame(columns=['param', 'AIC'])
df0bj1
```

Out[108]:

param	AIC

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

```
Model: (3, 1, 3)
```

In [108]:
```
dfObj1 = pd.DataFrame(columns=['param', 'AIC'])
dfObj1
```

Out[108]:

param	AIC

In [109]:
```
for param in pdq:
            try:
                mod = ARIMA(train, order=param)
                results_Arima = mod.fit()
                print('ARIMA{} - AIC:{}'.format(param, results_Arima.aic))
                dfObj1 = dfObj1.append({'param':param, 'AIC': results_Arima.aic}, ignore_index=True)

            except:
                continue
```

```
ARIMA(0, 1, 0) - AIC:521.6866087381227
ARIMA(0, 1, 1) - AIC:427.2089971236799
ARIMA(0, 1, 2) - AIC:386.71116114025926
ARIMA(0, 1, 3) - AIC:370.1183497636364
ARIMA(1, 1, 0) - AIC:412.155005106317
ARIMA(1, 1, 1) - AIC:394.2494988510245
ARIMA(1, 1, 2) - AIC:379.5240045466983
ARIMA(1, 1, 3) - AIC:372.0013093529383
ARIMA(2, 1, 0) - AIC:373.2794540954644
ARIMA(2, 1, 1) - AIC:332.32841192201295
ARIMA(2, 1, 2) - AIC:333.9885262034079
ARIMA(2, 1, 3) - AIC:336.23080880420673
ARIMA(3, 1, 0) - AIC:356.01656397685235
ARIMA(3, 1, 1) - AIC:334.0121904477379
ARIMA(3, 1, 2) - AIC:335.5236233647461
ARIMA(3, 1, 3) - AIC:337.0482811392801
```

In [110]:
```
dfObj1.sort_values(by=['AIC'])
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb


```
ARIMA(3, 1, 1) - AIC:334.0121904477379
ARIMA(3, 1, 2) - AIC:335.5236233647461
ARIMA(3, 1, 3) - AIC:337.0482811392801
```

In [110]:
```
dfObj1.sort_values(by=['AIC'])
```


Out[110]:

	param	AIC
9	(2, 1, 1)	332.328412
10	(2, 1, 2)	333.988526
13	(3, 1, 1)	334.012190
14	(3, 1, 2)	335.523623
11	(2, 1, 3)	336.230809
15	(3, 1, 3)	337.048281
12	(3, 1, 0)	356.016564
3	(0, 1, 3)	370.118350
7	(1, 1, 3)	372.001309
8	(2, 1, 0)	373.279454
6	(1, 1, 2)	379.524005
2	(0, 1, 2)	386.711161
5	(1, 1, 1)	394.249499
4	(1, 1, 0)	412.155005
1	(0, 1, 1)	427.208997
0	(0, 1, 0)	521.686609

In [111]:
```
model = ARIMA(train, order=(2,1,3))

results_Arima = model.fit()

print(results_Arima.summary())
```

Build Model

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

```
1  (0, 1, 1)  427.208997
0  (0, 1, 0)  521.686609
```

In [111]:
```
model = ARIMA(train, order=(2,1,3))

results_Arima = model.fit()

print(results_Arima.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                 CO2 ppm   No. Observations:                  168
Model:                ARIMA(2, 1, 3)   Log Likelihood                -162.115
Date:                Fri, 16 Dec 2022   AIC                            336.231
Time:                        21:00:27   BIC                            354.939
Sample:                    01-01-1965   HQIC                           343.824
                         - 12-01-1978
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.4422      0.089     16.157      0.000       1.267       1.617
ar.L2         -0.7611      0.089     -8.548      0.000      -0.936      -0.587
ma.L1         -0.7110      0.124     -5.745      0.000      -0.954      -0.468
ma.L2          0.0986      0.120      0.820      0.412      -0.137       0.334
ma.L3         -0.1859      0.102     -1.816      0.069      -0.387       0.015
sigma2         0.4028      0.045      8.871      0.000       0.314       0.492
===================================================================================
Ljung-Box (L1) (Q):                   0.79   Jarque-Bera (JB):                 0.91
Prob(Q):                              0.37   Prob(JB):                         0.63
Heteroskedasticity (H):               0.70   Skew:                             0.17
Prob(H) (two-sided):                  0.19   Kurtosis:                         3.10
===================================================================================

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

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

Predict on the Test Set using this model and evaluate the model on the test set using RMSE and MAPE

In [112]:
```
pred_start=test.index[0]
pred_end=test.index[-1]
```

In [113]:
```
ARIMA_predictions=results_Arima.predict(start=pred_start, end=pred_end)
```

In [114]:
```
ARIMA_predictions
```

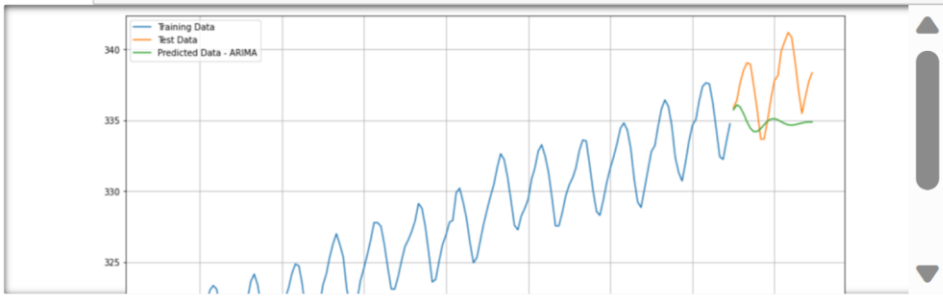
Out[114]:
```
1979-01-01    335.746732
1979-02-01    336.092913
1979-03-01    335.948025
1979-04-01    335.475580
1979-05-01    334.904519
1979-06-01    334.440561
1979-07-01    334.206121
1979-08-01    334.221162
1979-09-01    334.421299
1979-10-01    334.698478
1979-11-01    334.945881
1979-12-01    335.091701
1980-01-01    335.113686
1980-02-01    335.034401
1980-03-01    334.903325
1980-04-01    334.774642
1980-05-01    334.688827
1980-06-01    334.663017
1980-07-01    334.691111
1980-08-01    334.751274
1980-09-01    334.816653
1980-10-01    334.865149
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

```
1980-12-01    334.877505
Freq: MS, Name: predicted_mean, dtype: float64
```

In [115]:
```
ARIMA_pred=ARIMA_predictions.cumsum()
#ARIMA_pred
```

In [116]:
```
plt.plot(train,label='Training Data')
plt.plot(test,label='Test Data')
plt.plot(test.index,ARIMA_predictions,label='Predicted Data - ARIMA')
plt.legend(loc='best')
plt.grid();
```



In [117]:
```
test.columns
```

Out[117]:
```
Index(['CO2 ppm'], dtype='object')
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [117]:
```
test.columns
```

Out[117]:
```
Index(['CO2 ppm'], dtype='object')
```

In [118]:
```
test.head()
```

Out[118]:

Year-Month	CO2 ppm
1979-01-01	335.89
1979-02-01	336.44
1979-03-01	337.63
1979-04-01	338.54
1979-05-01	339.06

In [119]:
```
ARIMA_predictions.head()
```

Out[119]:
```
1979-01-01    335.746732
1979-02-01    336.092913
1979-03-01    335.948025
1979-04-01    335.475580
1979-05-01    334.904519
Freq: MS, Name: predicted_mean, dtype: float64
```

In [120]:
```
ARIMA_predictions_df = pd.DataFrame(ARIMA_predictions)
```

In [121]:
```
ARIMA_predictions_df.head()
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [121]:
```
ARIMA_predictions_df.head()
```

Out[121]:

	predicted_mean
1979-01-01	335.746732
1979-02-01	336.092913
1979-03-01	335.948025
1979-04-01	335.475580
1979-05-01	334.904519

In [122]:
```
ARIMA_predictions_df['predicted_mean']
```

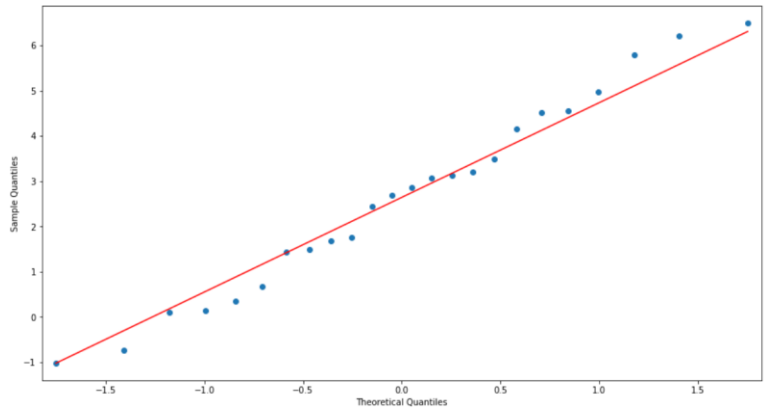
Out[122]:
```
1979-01-01    335.746732
1979-02-01    336.092913
1979-03-01    335.948025
1979-04-01    335.475580
1979-05-01    334.904519
1979-06-01    334.440561
1979-07-01    334.206121
1979-08-01    334.221162
1979-09-01    334.421299
1979-10-01    334.698478
1979-11-01    334.945881
1979-12-01    335.091701
1980-01-01    335.113686
1980-02-01    335.034401
1980-03-01    334.903325
1980-04-01    334.774642
1980-05-01    334.688827
1980-06-01    334.663017
1980-07-01    334.691111
1980-08-01    334.751274
1980-09-01    334.816653
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

```
1980-12-01    334.877505
Freq: MS, Name: predicted_mean, dtype: float64
```

In [123]:
```
residuals = test['CO2 ppm'] - ARIMA_predictions_df['predicted_mean']
```

In [124]:
```
qqplot(residuals,line="s");
```



localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [125]:
```
from sklearn.metrics import  mean_squared_error

rmse = mean_squared_error(test['CO2 ppm'],ARIMA_predictions_df['predicted_mean'], squared=False)

print(rmse)
```

```
3.371770506606991
```

In [126]:
```
def MAPE(y_true, y_pred):
    return np.mean((np.abs(y_true-y_pred))/(y_true))*100
```

In [127]:
```
mape=MAPE(test['CO2 ppm'].values,ARIMA_predictions_df['predicted_mean'].values)
print(mape)
```

```
0.8234293516746082
```

SARIMA Model

we will find the model parameters based on AIC criteria. Parameters will be generated using combination for the given range.

In [128]:
```
import itertools
p = q = range(0, 3)
d= range(1,2)
pdq = list(itertools.product(p, d, q))

model_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Model...')
print('Model: {}{}'.format(pdq[1], model_pdq[1]))
print('Model: {}{}'.format(pdq[1], model_pdq[2]))
print('Model: {}{}'.format(pdq[2], model_pdq[3]))
print('Model: {}{}'.format(pdq[2], model_pdq[4]))
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

SARIMA Model

we will find the model parameters based on AIC criteria. Parameters will be generated using combination for the given range.

In [128]:
```
import itertools
p = q = range(0, 3)
d= range(1,2)
pdq = list(itertools.product(p, d, q))

model_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Model...')
print('Model: {}{}'.format(pdq[1], model_pdq[1]))
print('Model: {}{}'.format(pdq[1], model_pdq[2]))
print('Model: {}{}'.format(pdq[2], model_pdq[3]))
print('Model: {}{}'.format(pdq[2], model_pdq[4]))
```

```
Examples of parameter combinations for Model...
Model: (0, 1, 1)(0, 1, 1, 12)
Model: (0, 1, 1)(0, 1, 2, 12)
Model: (0, 1, 2)(1, 1, 0, 12)
Model: (0, 1, 2)(1, 1, 1, 12)
```

Creating an empty Dataframe with column names only where the model and AIC scores will be saved

In [129]:
```
dfObj2 = pd.DataFrame(columns=['param','seasonal', 'AIC'])
dfObj2
```

Out[129]:

param	seasonal	AIC

In [130]:
```
import statsmodels.api as sm
for param in pdq:
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

Creating an empty Dataframe with column names only where the model and AIC scores will be saved

In [129]:
```
df0bj2 = pd.DataFrame(columns=['param','seasonal', 'AIC'])
df0bj2
```

Out[129]:

param	seasonal	AIC

In [130]:
```
import statsmodels.api as sm
for param in pdq:
    for param_seasonal in model_pdq:
        mod = sm.tsa.statespace.SARIMAX(train,
                                        order=param,
                                        seasonal_order=param_seasonal,
                                        enforce_stationarity=False,
                                        enforce_invertibility=False)

        results_SARIMA = mod.fit()
        print('SARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results_SARIMA.aic))
        df0bj2 = df0bj2.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
```

```
SARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:169.5026390080524
SARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:76.60024513106538
SARIMA(0, 1, 0)x(0, 1, 2, 12)12 - AIC:63.668610704536704
SARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:101.77574384550147
SARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:73.06883156314025
SARIMA(0, 1, 0)x(1, 1, 2, 12)12 - AIC:63.39376878404886
SARIMA(0, 1, 0)x(2, 1, 0, 12)12 - AIC:78.32111628951127
SARIMA(0, 1, 0)x(2, 1, 1, 12)12 - AIC:64.7713520537352
SARIMA(0, 1, 0)x(2, 1, 2, 12)12 - AIC:64.0921131510954
SARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:165.13652038660632
SARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:66.2925068981281
SARIMA(0, 1, 1)x(0, 1, 2, 12)12 - AIC:62.33158586070941
SARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:98.4577031121515
SARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:64.95051921007558
SARIMA(0, 1, 1)x(1, 1, 2, 12)12 - AIC:60.48473972992478
SARIMA(0, 1, 1)x(2, 1, 0, 12)12 - AIC:76.1432098087794
SARIMA(0, 1, 1)x(2, 1, 1, 12)12 - AIC:59.415580794585054
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

```
SARIMA(2, 1, 2)x(0, 1, 2, 12)12 - AIC:65.89563726918 89
SARIMA(2, 1, 2)x(1, 1, 0, 12)12 - AIC:95.99223454495859
SARIMA(2, 1, 2)x(1, 1, 1, 12)12 - AIC:69.15503979581301
SARIMA(2, 1, 2)x(1, 1, 2, 12)12 - AIC:63.402971100707774
SARIMA(2, 1, 2)x(2, 1, 0, 12)12 - AIC:81.78476331678657
SARIMA(2, 1, 2)x(2, 1, 1, 12)12 - AIC:69.68148164786038
SARIMA(2, 1, 2)x(2, 1, 2, 12)12 - AIC:66.50054911853431
```

Sorting parameters for best AIC score

In [131]:
```
df0bj2.sort_values(by=['AIC'])
```

Out[131]:

	param	seasonal	AIC
29	(1, 1, 0)	(0, 1, 2, 12)	59.094377
16	(0, 1, 1)	(2, 1, 1, 12)	59.415581
32	(1, 1, 0)	(1, 1, 2, 12)	59.504680
14	(0, 1, 1)	(1, 1, 2, 12)	60.484740
56	(2, 1, 0)	(0, 1, 2, 12)	60.946225
...	...	...	...
18	(0, 1, 2)	(0, 1, 0, 12)	165.522024
63	(2, 1, 1)	(0, 1, 0, 12)	165.632730
27	(1, 1, 0)	(0, 1, 0, 12)	166.711967
54	(2, 1, 0)	(0, 1, 0, 12)	167.606179
0	(0, 1, 0)	(0, 1, 0, 12)	169.502639

81 rows × 3 columns

In [132]:
```
model = sm.tsa.statespace.SARIMAX(train,
                                  order=(1,1,0),
                                  seasonal_order=(1,1,2,12),
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [132]:
```
model = sm.tsa.statespace.SARIMAX(train,
                                  order=(1,1,0),
                                  seasonal_order=(1,1,2,12),
                                  )
model_Sarima = model.fit()
print(model_Sarima.summary())
```

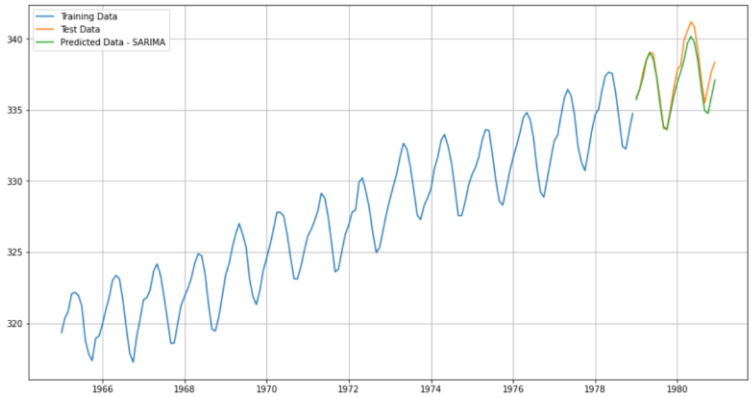
```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                  CO2 ppm   No. Observations:                  168
Model:             SARIMAX(1, 1, 0)x(1, 1, [1, 2], 12)   Log Likelihood                 -42.065
Date:                                   Fri, 16 Dec 2022   AIC                             94.131
Time:                                           21:01:35   BIC                            109.348
Sample:                                       01-01-1965   HQIC                           100.312
                                            - 12-01-1978
Covariance Type:                                     opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2664      0.072     -3.699      0.000      -0.408      -0.125
ar.S.L12      -0.6905      0.613     -1.126      0.260      -1.893       0.512
ma.S.L12      -0.3735      5.793     -0.064      0.949     -11.728      10.981
ma.S.L24      -0.6217      3.922     -0.159      0.874      -8.308       7.065
sigma2         0.0817      0.479      0.171      0.865      -0.858       1.021
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                 0.97
Prob(Q):                              0.87   Prob(JB):                         0.61
Heteroskedasticity (H):               0.55   Skew:                             0.02
Prob(H) (two-sided):                  0.03   Kurtosis:                         2.61
===================================================================================

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

In [133]:
```
SARIMA_predictions=model_Sarima.predict(start=pred_start, end=pred_end)
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [133]:
```
SARIMA_predictions=model_Sarima.predict(start=pred_start, end=pred_end)
```

In [134]:
```
plt.plot(train,label='Training Data')
plt.plot(test,label='Test Data')
plt.plot(test.index,SARIMA_predictions,label='Predicted Data - SARIMA')
plt.legend(loc='best')
plt.grid();
```



localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

finding RSMA and MAPE

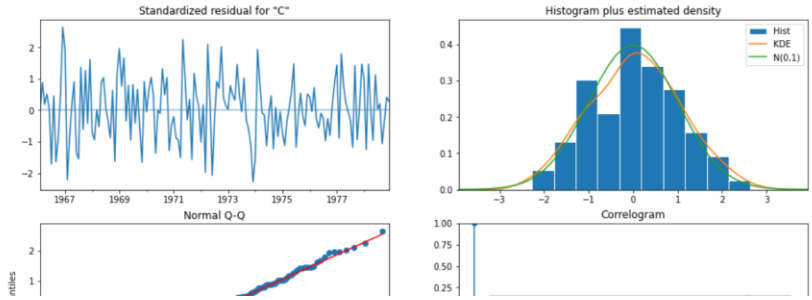
In [135]:
```
from sklearn.metrics import  mean_squared_error
rmse = mean_squared_error(test['CO2 ppm'],SARIMA_predictions, squared=False)
print(rmse)
```

```
0.8213789499910458
```

In [136]:
```
mape = MAPE(test['CO2 ppm'],SARIMA_predictions)
print(mape)
```

```
0.18461311230296018
```

In [137]:
```
model_Sarima.plot_diagnostics(figsize=(16, 8))
plt.show()
```



localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

fitting model on whole data

In [138]:
```
model = sm.tsa.statespace.SARIMAX(df,
                                  order=(1,1,0),
                                  seasonal_order=(1,1,2,12),
                                  enforce_stationarity=False,
                                  enforce_invertibility=False)
model_Sarima = model.fit()
print(model_Sarima.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                CO2 ppm   No. Observations:                  192
Model:             SARIMAX(1, 1, 0)x(1, 1, [1, 2], 12)   Log Likelihood                -39.857
Date:                                  Fri, 16 Dec 2022   AIC                             89.714
Time:                                          21:01:36   BIC                            104.899
Sample:                                      01-01-1965   HQIC                            95.882
                                           - 12-01-1980
Covariance Type:                                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2668      0.085     -3.120      0.002      -0.434      -0.099
ar.S.L12      -0.5808      0.223     -2.603      0.009      -1.018      -0.144
ma.S.L12      -0.3017      0.240     -1.258      0.208      -0.771       0.168
ma.S.L24      -0.4384      0.218     -2.010      0.044      -0.866      -0.011
sigma2         0.0944      0.009     10.987      0.000       0.078       0.111
===================================================================================
Ljung-Box (L1) (Q):                   0.06   Jarque-Bera (JB):                14.27
Prob(Q):                              0.81   Prob(JB):                         0.00
Heteroskedasticity (H):               1.13   Skew:                             0.38
Prob(H) (two-sided):                  0.68   Kurtosis:                         4.28
===================================================================================

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

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

Forecast with confidence interval

In [139]:
```
forecast = model_Sarima.forecast(steps=24)
```

In [140]:
```
forecast
```

Out[140]:
```
1981-01-01    339.589037
1981-02-01    340.167598
1981-03-01    341.227873
1981-04-01    342.311254
1981-05-01    342.846994
1981-06-01    342.507077
1981-07-01    341.130712
1981-08-01    339.208836
1981-09-01    337.517373
1981-10-01    337.486304
1981-11-01    338.716824
1981-12-01    339.895359
1982-01-01    340.994987
1982-02-01    341.593296
1982-03-01    342.692636
1982-04-01    343.751765
1982-05-01    344.284442
1982-06-01    343.940750
1982-07-01    342.554429
1982-08-01    340.615463
1982-09-01    338.924181
1982-10-01    338.987696
1982-11-01    340.205041
1982-12-01    341.323481
Freq: MS, Name: predicted_mean, dtype: float64
```

In [141]:
```
pred95 = model_Sarima.get_forecast(steps=24)
```

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

Freq: MS, Name: predicted_mean, dtype: float64

In [141]:
```
pred95 = model_Sarima.get_forecast(steps=24)
```

In [142]:
```
pred95=pred95.conf_int()
```

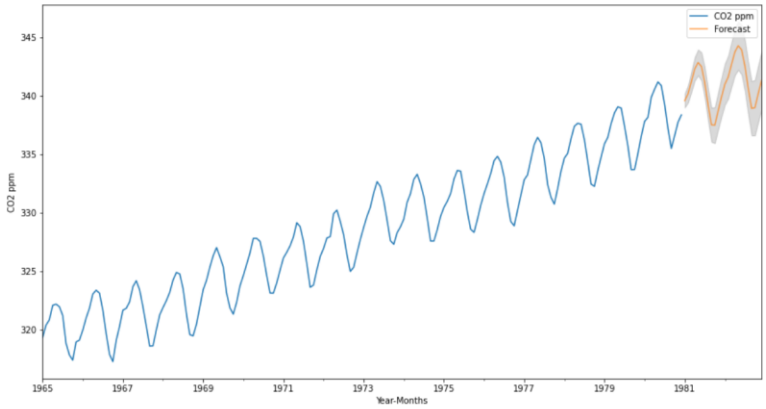
In [143]:
```
pred95
```

Out[143]:

	lower CO2 ppm	upper CO2 ppm
1981-01-01	338.986505	340.191568
1981-02-01	339.420455	340.914740
1981-03-01	340.337379	342.118367
1981-04-01	341.302865	343.319642
1981-05-01	341.731827	343.962161
1981-06-01	341.294816	343.719339
1981-07-01	339.828498	342.432926
1981-08-01	337.822512	340.595159
1981-09-01	336.051755	338.982991
1981-10-01	335.945468	339.027141
1981-11-01	337.104273	340.329374
1981-12-01	338.214151	341.576567
1982-01-01	339.227026	342.762948
1982-02-01	339.748190	343.438402
1982-03-01	340.772170	344.613102
1982-04-01	341.759119	345.744410
1982-05-01	342.222055	346.346829

localhost:8888/notebooks/Faculty%2BNotebook_S4-ARIMA%20(2).ipynb

In [144]:
```
axis = df.plot(label='Observed', figsize=(15, 8))
forecast.plot(ax=axis, label='Forecast', alpha=0.7)
axis.fill_between(forecast.index, pred95['lower CO2 ppm'], pred95['upper CO2 ppm'], color='k', alpha=.15)
axis.set_xlabel('Year-Months')
axis.set_ylabel('CO2 ppm')
plt.legend(loc='best')
plt.show()
```



Session 5

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

```
warnings.filterwarnings("ignore")
```

- Store id
- Day of Week
- Date
- Customers: the number of customers on a given day.(Target Variable)
- Open: an indicator for whether the store was open: 0 = closed, 1 = open.
- Promo: indicates whether a store is running a promo on that day.
- StateHoliday: indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. This has value as "0", "a", "b", "c"
- SchoolHoliday: indicates if the (Store, Date) was affected by the closure of public schools.

In [ ]:
```
Store50 = pd.read_excel("Store_50.xlsx", parse_dates = True, index_col = 'Date')
```

In [ ]:
```
Store50.head()
```

Out[3]:

	Store	Day Of Week	Customers	Open	Promotion	State Holiday	School Holiday
Date							
2016-05-31	1044	1	884	1	1	0	0
2016-05-31	1041	1	1032	1	1	0	1
2016-05-31	1036	1	1070	1	1	0	0
2016-05-31	1047	1	2043	1	1	0	0
2016-05-31	1012	1	1239	1	1	0	0

In [ ]:
```
print("The number of rows: ",Store50.shape[0], "\n""The number of columns: ",Store50.shape[1])
```

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

In [ ]:
```
print("The number of rows: ",Store50.shape[0], "\n""The number of columns: ",Store50.shape[1])
```

```
The number of rows:  43326
The number of columns:  7
```

In [ ]:
```
## Checking the number of stores for which we have the data

Store50['Store'].unique()
```

Out[5]:
```
array([1044, 1041, 1036, 1047, 1012, 1037, 1027, 1005, 1011, 1004, 1043,
       1001, 1048, 1030, 1035, 1013, 1029, 1042, 1017, 1032, 1024, 1006,
       1007, 1002, 1019, 1000, 1009, 1023, 1016, 1050, 1031, 1010, 1034,
       1039, 1033, 1045, 1015, 1022, 1040, 1046, 1021, 1028, 1008, 1025,
       1014, 1020, 1038, 1003, 1018, 1026, 1049], dtype=int64)
```

In [ ]:
```
## Selecting the store 1044 only for analysis.
## Here, we are also reversing the data as we need to contiguous data in the ascending order for a Time Series Analysis.

Store50_1044 = Store50[Store50.Store == 1044]. sort_index(ascending=True)
```

In [ ]:
```
Store50_1044.head()
```

Out[7]:

	Store	Day Of Week	Customers	Open	Promotion	State Holiday	School Holiday
Date							
2014-01-01	1044	2	0	0	0	a	1
2014-01-02	1044	5	473	1	0	0	0
2014-01-03	1044	5	483	1	0	0	0

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (unsaved changes)

In [ ]:
```
Store50_1044.head()
```

Out[7]:

	Store	Day Of Week	Customers	Open	Promotion	State Holiday	School Holiday
Date							
2014-01-01	1044	2	0	0	0	a	1
2014-01-02	1044	5	473	1	0	0	0
2014-01-03	1044	5	483	1	0	0	0
2014-01-04	1044	1	0	0	0	b	1
2014-01-05	1044	3	0	0	1	a	0

In [ ]:
```
Store50_1041 = Store50[Store50.Store == 1041]. sort_index(ascending=True)
Store50_1041.head()
```

Out[8]:

	Store	Day Of Week	Customers	Open	Promotion	State Holiday	School Holiday
Date							
2014-01-01	1041	2	0	0	0	a	1
2014-01-02	1041	5	742	1	0	0	0
2014-01-03	1041	5	864	1	0	0	0
2014-01-04	1041	1	0	0	0	b	1
2014-01-05	1041	3	0	0	1	a	0

In [ ]:
```
df = pd.DataFrame(columns=['Store_1044','Store_1041'])
df
```

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

In [ ]:
```
df = pd.DataFrame(columns=['Store_1044','Store_1041'])
df
```

Out[9]:

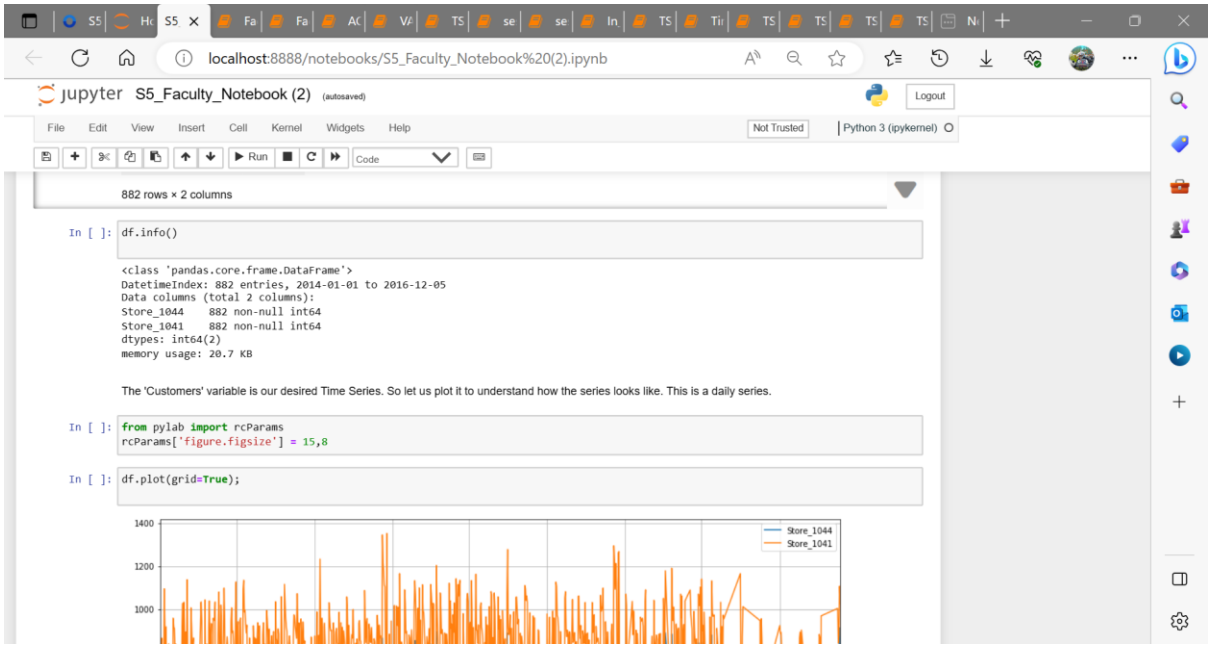
Store_1044	Store_1041

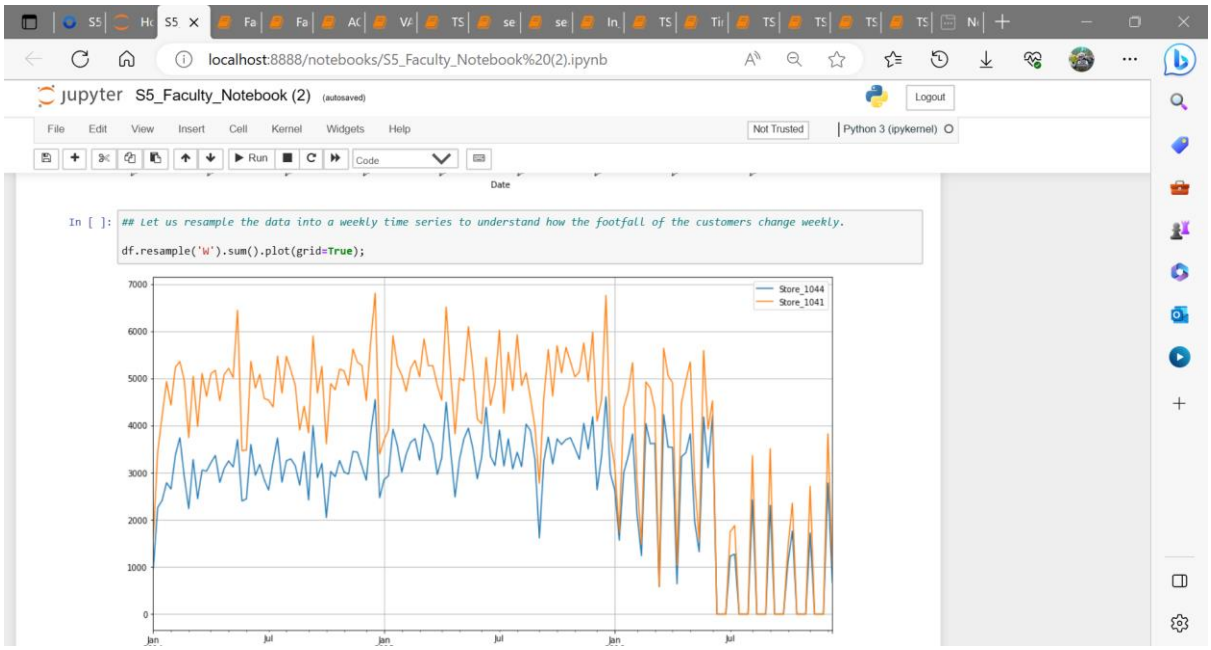
In [ ]:
```
## Putting the two desired Time Series together in one dataframe

df['Store_1044'] = Store50_1044['Customers']
df['Store_1041'] = Store50_1041['Customers']
df
```

Out[11]:

	Store_1044	Store_1041
Date		
2014-01-01	0	0
2014-01-02	473	742
2014-01-03	483	864
2014-01-04	0	0
2014-01-05	0	0
2014-01-06	498	518
2014-01-07	684	1095
2014-01-08	539	909
2014-01-09	0	0
2014-01-10	542	917
2014-01-11	0	0

In [ ]:
```
df.info()
```


localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb
jupyter S5_Faculty_Notebook (2) (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Logout
Run
Code
882 rows × 2 columns
In [ ]: df.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 882 entries, 2014-01-01 to 2016-12-05
Data columns (total 2 columns):
Store_1044 882 non-null int64
Store_1041 882 non-null int64
dtypes: int64(2)
memory usage: 20.7 KB
The 'Customers' variable is our desired Time Series. So let us plot it to understand how the series looks like. This is a daily series.
In [ ]: from pylab import rcParams
rcParams['figure.figsize'] = 15,8
In [ ]: df.plot(grid=True);
1400
1200
1000
Store_1044
Store_1041


localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb
jupyter S5_Faculty_Notebook (2) (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Logout
Run
Code
Date
In [ ]: ## Let us resample the data into a weekly time series to understand how the footfall of the customers change weekly.
df.resample('W').sum().plot(grid=True);
7000
6000
5000
4000
3000
2000
1000
0
Jan
Jul
Jan
Jul
Jan
Jul
Store_1044
Store_1041

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

We see that the Store_1041 has a higher footfall of customers as compared to the Store_1044.

In [ ]:
```
df.index.year.unique()
```

Out[16]:
```
Int64Index([2014, 2015, 2016], dtype='int64', name='Date')
```

Split the data into training and test. The test data starts from 2016 and onwards.

In [ ]:
```
train=df[df.index.year !=2016]
test=df[df.index.year ==2016]
```

In [ ]:
```
print('First 5 rows of the training data')
display(train.head())
print('Last 5 rows of the training data')
display(train.tail())
print('First 5 rows of the test data')
display(test.head())
print('Last 5 rows of the test data')
display(test.tail())
```

```
First 5 rows of the training data
```

Date	Store_1044	Store_1041
2014-01-01	0	0
2014-01-02	473	742
2014-01-03	483	864

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [ ]:
```
print('First 5 rows of the training data')
display(train.head())
print('Last 5 rows of the training data')
display(train.tail())
print('First 5 rows of the test data')
display(test.head())
print('Last 5 rows of the test data')
display(test.tail())
```

```
First 5 rows of the training data
```

Date	Store_1044	Store_1041
2014-01-01	0	0
2014-01-02	473	742
2014-01-03	483	864
2014-01-04	0	0
2014-01-05	0	0

```
Last 5 rows of the training data
```

Date	Store_1044	Store_1041
2015-12-27	537	535
2015-12-28	0	0
2015-12-29	667	806
2015-12-30	668	765

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Let us check the number of rows and columns of the training and test set.

In [ ]:
```
print(train.shape)
print(test.shape)
```

```
(730, 2)
(152, 2)
```

Checking whether the training data is stationary.

In [ ]:
```
## Importing the Augmented Dickey-Fuller test from the statsmodels library

from statsmodels.tsa.stattools import adfuller
```

In [ ]:
```
## Defning a function
def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries,regression='ct')#running the adf test on the input time series
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    ## creating a series to format the output
    for key,value in dftest[4].items():##running a for loop to format the critical values of the test statistic
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

In [ ]:
```
adf_test(train['Store_1044'])
```

```
Results of Dickey-Fuller Test:
```

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

```
    for key,value in dftest[4].items():##running a for loop to format the critical values of the test statistic
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

In [ ]:
```
adf_test(train['Store_1044'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -6.690539e+00
p-value                          6.994789e-08
#Lags Used                       2.000000e+01
Number of Observations Used      7.090000e+02
Critical Value (1%)             -3.971596e+00
Critical Value (5%)             -3.416700e+00
Critical Value (10%)            -3.130705e+00
dtype: float64
```

In [ ]:
```
adf_test(train['Store_1041'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -6.147834
p-value                          0.000001
#Lags Used                      20.000000
Number of Observations Used    709.000000
Critical Value (1%)             -3.971596
Critical Value (5%)             -3.416700
Critical Value (10%)            -3.130705
dtype: float64
```

Series is stationary and hence no need for differentiation.

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Vector Auto-Regressive

Now, we are going to build a Vector Auto-Regressive Model. In this model, we will be using the lagged or past values of a particular Time Series to predict the other Time Series.

In this case, we have two stationary Time Series and we will be using these to build a VAR model.

Before we code up this model, let us understand how this model is interpreted and how the equation of this model looks like for two time-series.

The following is the example of a two-dimensional VAR(1) model.

$$y_{1,t} = c_1 + \phi_{11,1}\, y_{1,t-1} + \phi_{12,1}\, y_{2,t-1} + e_{1,t}$$

$$y_{2,t} = c_2 + \phi_{21,1}\, y_{1,t-1} + \phi_{22,1}\, y_{2,t-1} + e_{2,t}$$

Here e is the error term (white noise).

We can include the moving average terms to this kind of models as well. We can think of VAR models to AR models extended to multiple Time Series.

But we need to very careful in incorporating the number of variables in a VAR model and also be wary of the order of the VAR model. A complicated model by definition would work well on the training data but not so much on the test data. Introducing more time series would also include the error terms from those time series for the prediction of another time series.

VAR models are very powerful in a sense that it gives us an idea about the causation property a bit as well. If we get an improvement in the prediction accuracy by using a VAR model instead of an AR model, we can say that the past values of another time series aids in the prediction of our desired time

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Now, we are going to build a Vector Auto-Regressive Model. In this model, we will be using the lagged or past values of a particular Time Series to predict the other Time Series.

In this case, we have two stationary Time Series and we will be using these to build a VAR model.

Before we code up this model, let us understand how this model is interpreted and how the equation of this model looks like for two time-series.

The following is the example of a two-dimensional VAR(1) model.

$$y_{1,t} = c_1 + \phi_{11,1}\, y_{1,t-1} + \phi_{12,1}\, y_{2,t-1} + e_{1,t}$$

$$y_{2,t} = c_2 + \phi_{21,1}\, y_{1,t-1} + \phi_{22,1}\, y_{2,t-1} + e_{2,t}$$

Here e is the error term (white noise).

We can include the moving average terms to this kind of models as well. We can think of VAR models to AR models extended to multiple Time Series.

But we need to very careful in incorporating the number of variables in a VAR model and also be wary of the order of the VAR model. A complicated model by definition would work well on the training data but not so much on the test data. Introducing more time series would also include the error terms from those time series for the prediction of another time series.

VAR models are very powerful in a sense that it gives us an idea about the causation property a bit as well. If we get an improvement in the prediction accuracy by using a VAR model instead of an AR model, we can say that the past values of another time series aids in the prediction of our desired time series.

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Build a VAR model.

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

In [ ]:
```
## We need to convert these variables into float as that is the input that statsmodels api takes

train['Store_1044'] = train['Store_1044'].astype('float64')
train['Store_1041'] = train['Store_1041'].astype('float64')
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 730 entries, 2014-01-01 to 2015-12-31
Data columns (total 2 columns):
Store_1044    730 non-null float64
Store_1041    730 non-null float64
dtypes: float64(2)
memory usage: 17.1 KB
```

In [ ]:
```
## We are running this iteration upto 7 days since this is a daily data and we have seen that the customer footfall
## of Store 1044 has a seasonality of 7.

for i in range(1,8):
    model = sm.tsa.VARMAX(train,order=(i,0),trend='c')
    model_result = model.fit()
    print('Order =',i)
    print('AIC:',model_result.aic)
```

```
Order = 1
AIC: 19230.833991125226
```

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [ ]:
```
## We are running this iteration upto 7 days since this is a daily data and we have seen that the customer footfall
## of Store 1044 has a seasonality of 7.

for i in range(1,8):
    model = sm.tsa.VARMAX(train,order=(i,0),trend='c')
    model_result = model.fit()
    print('Order =',i)
    print('AIC:',model_result.aic)
```

```
Order = 1
AIC: 19230.833991125226
Order = 2
AIC: 19181.933268520806
Order = 3
AIC: 19185.015107959807
Order = 4
AIC: 19149.910545855873
Order = 5
AIC: 19143.80985901448
Order = 6
AIC: 19118.19579946341
Order = 7
AIC: 19051.762446638415
```

In [ ]:
```
model = sm.tsa.VARMAX(train,order=(7,0),trend='c')
model_result = model.fit()
model_result.summary()
```

Out[28]:

Statespace Model Results

Dep. Variable:	['Store_1044', 'Store_1041']	**No. Observations:**	730

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
Order = 6
AIC: 19118.19579946341
Order = 7
AIC: 19051.762446638415
```

In [ ]:
```
model = sm.tsa.VARMAX(train,order=(7,0),trend='c')
model_result = model.fit()
model_result.summary()
```

Out[28]:

Statespace Model Results

Dep. Variable:	['Store_1044', 'Store_1041']	No. Observations:	730
Model:	VAR(7)	Log Likelihood	-9492.881
	+ intercept	AIC	19051.762
Date:	Tue, 16 Feb 2021	BIC	19203.333
Time:	17:59:32	HQIC	19110.239
Sample:	01-01-2014		
	- 12-31-2015		
Covariance Type:	opg		

Ljung-Box (Q):	115.84, 85.90	Jarque-Bera (JB):	65.83, 1688.68
Prob(Q):	0.00, 0.00	Prob(JB):	0.00, 0.00
Heteroskedasticity (H):	1.33, 1.65	Skew:	-0.70, -1.07
Prob(H) (two-sided):	0.03, 0.00	Kurtosis:	3.44, 10.13

In [ ]:
```
pred = model_result.forecast(steps=len(test))
pred
```

Out[29]:

	Store_1044	Store_1041

localhost:8888/notebooks/S5_Faculty_Notebook%20(2).ipynb

jupyter S5_Faculty_Notebook (2) (autosaved)

File Edit View Insert Cell Kernel Widgets Help

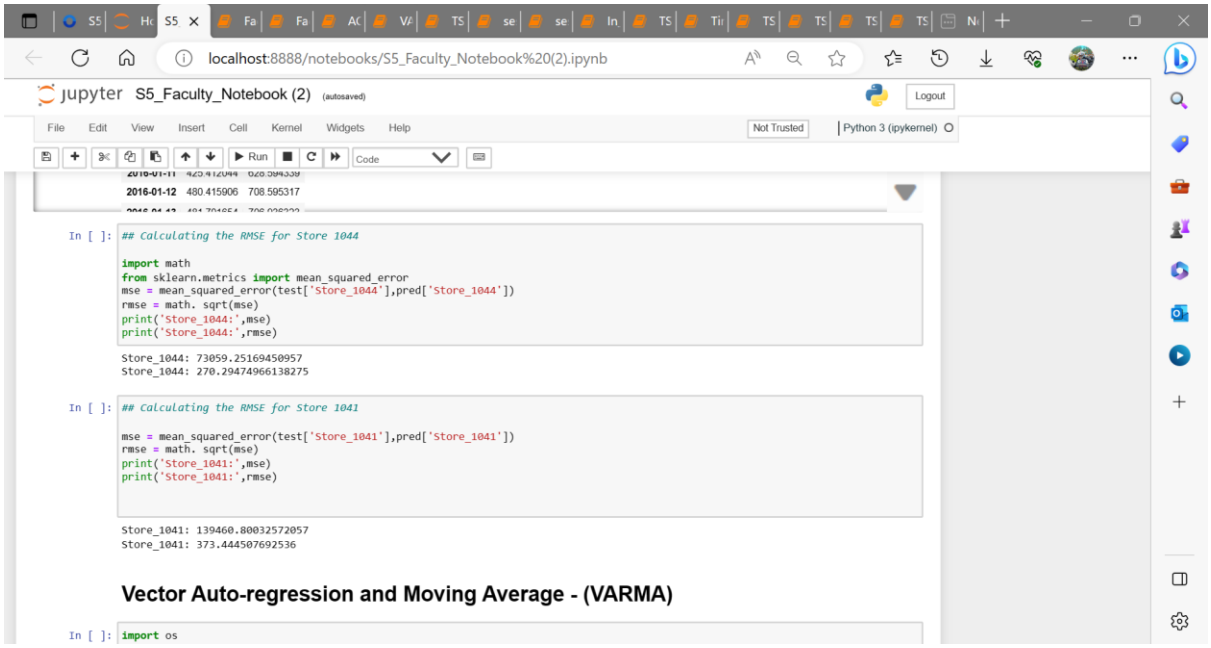
Prob(Q):	0.00, 0.00	Prob(JB):	0.00, 0.00
Heteroskedasticity (H):	1.33, 1.65	Skew:	-0.70, -1.07
Prob(H) (two-sided):	0.03, 0.00	Kurtosis:	3.44, 10.13

In [ ]:
```
pred = model_result.forecast(steps=len(test))
pred
```

Out[29]:

	Store_1044	Store_1041
2016-01-01	396.938653	513.361758
2016-01-02	469.559427	649.430068
2016-01-03	539.414903	692.643263
2016-01-04	328.202395	482.117648
2016-01-05	521.092410	735.864244
2016-01-06	520.307164	723.240300
2016-01-07	482.554076	653.105674
2016-01-08	438.052011	625.273907
2016-01-09	488.794689	707.800341
2016-01-10	499.999911	707.234569
2016-01-11	425.412044	628.594339
2016-01-12	480.415906	708.595317

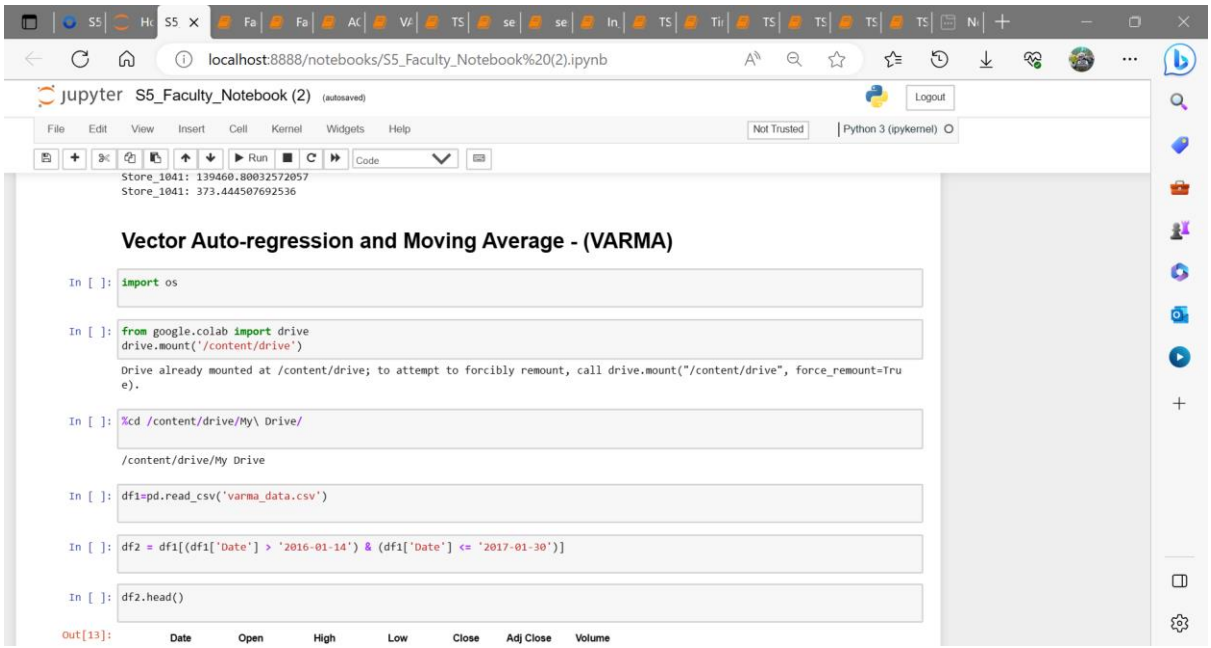
In [ ]:
```
## Calculating the RMSE for Store 1044

import math
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(test['Store_1044'],pred['Store_1044'])
```



```
2016-01-12  480.415906  708.595317

In [ ]: ## Calculating the RMSE for Store 1044

        import math
        from sklearn.metrics import mean_squared_error
        mse = mean_squared_error(test['Store_1044'],pred['Store_1044'])
        rmse = math. sqrt(mse)
        print('Store_1044:',mse)
        print('Store_1044:',rmse)

        Store_1044: 73059.25169450957
        Store_1044: 270.29474966138275

In [ ]: ## Calculating the RMSE for Store 1041

        mse = mean_squared_error(test['Store_1041'],pred['Store_1041'])
        rmse = math. sqrt(mse)
        print('Store_1041:',mse)
        print('Store_1041:',rmse)

        Store_1041: 139460.80032572057
        Store_1041: 373.444507692536
```

Vector Auto-regression and Moving Average - (VARMA)

```
In [ ]: import os
```



```
Store_1041: 139460.80032572057
Store_1041: 373.444507692536
```

Vector Auto-regression and Moving Average - (VARMA)

```
In [ ]: import os

In [ ]: from google.colab import drive
        drive.mount('/content/drive')

        Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
        e).

In [ ]: %cd /content/drive/My\ Drive/

        /content/drive/My Drive

In [ ]: df1=pd.read_csv('varma_data.csv')

In [ ]: df2 = df1[(df1['Date'] > '2016-01-14') & (df1['Date'] <= '2017-01-30')]

In [ ]: df2.head()

Out[13]:
```

Date	Open	High	Low	Close	Adj Close	Volume

Varma not doing as collab will be required