ARIMA AND SARIMA ON SEOSONAL DATA AND VECTOR AUTO REGRESSION

TIME SERIES MODELS ARIMA AND SARIMA ON CHOCLATE DATA SET

Import Necessary libraries.

```
!pip install statsmodels==0.13.2

import pandas as pd
from pandas import datetime

from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import acf

import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates

from sklearn.metrics import mean_squared_error
from math import sqrt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
import numpy as np
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: statsmodels==0.13.2 in /usr/local/lib/python3.7/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from statsmodels==0.13.2) (1.21.6)
```

```
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages (from statsmodels==0.13.2) (21.3)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (from statsmodels==0.13.2) (0.5.2)
Requirement already satisfied: scipy>=1.3 in /usr/local/lib/python3.7/dist-packages (from statsmodels==0.13.2) (1.4.1)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.7/dist-packages (from statsmodels==0.13.2) (1.3.5)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=21.3->statsm
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25->statsmodels==0.13.2)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25->statsmodels
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.2->statsmodels==0.13.2) (1.15.0)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: FutureWarning: The pandas.datetime class is deprecated and will
  """
```

Loading Choclate sales data set.

```
choclate_sales_data=pd.read_csv('Choclate_Sales.csv')
```

Understanding Choclate sales data set.

```
choclate_sales_data.head()
```

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

Setting index as Date and converting it into date time.

```
choclate_sales_data['Date'] = pd.to_datetime(choclate_sales_data['Date'])

choclate_sales_data=choclate_sales_data.set_index(choclate_sales_data['Date'])
```

```
choclate_sales_data.head()
```

	Date	Choclate_Sales
Date		
1964-01-01	1964-01-01	2815
1964-02-01	1964-02-01	2672
1964-03-01	1964-03-01	2755
1964-04-01	1964-04-01	2721
1964-05-01	1964-05-01	2946

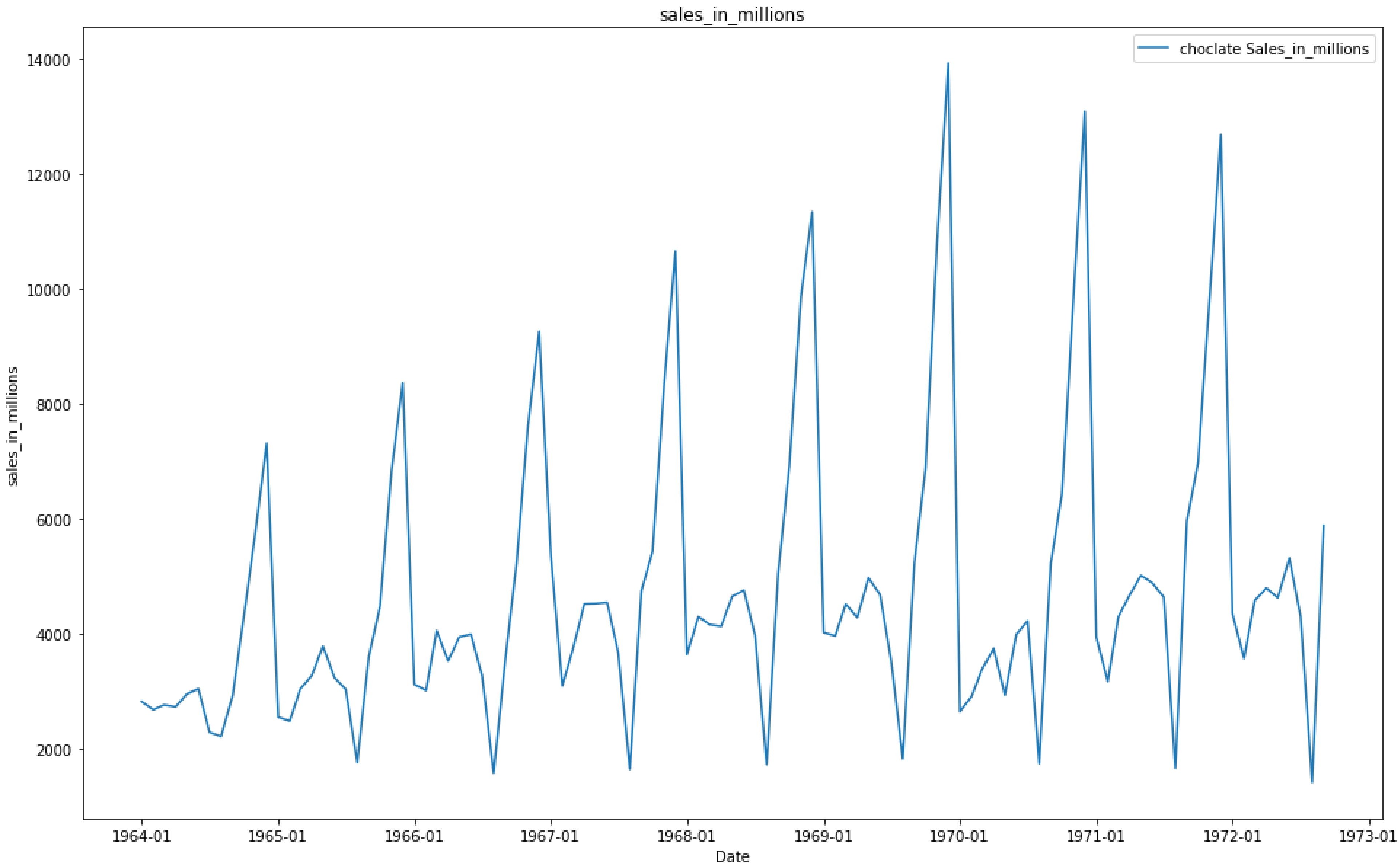
Drop all null values.

```
choclate_sales_data=choclate_sales_data.dropna()
```

Plot for choclate sales in millions

```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(choclate_sales_data['Date'], choclate_sales_data['Choclate_Sales'], label="choclate Sales_in_millions")


ax.set(xlabel="Date", ylabel="sales_in_millions",
       title="sales_in_millions")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
```

```
ax.xaxis.set_major_formatter(DateFormatter("19%y-%m"))

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

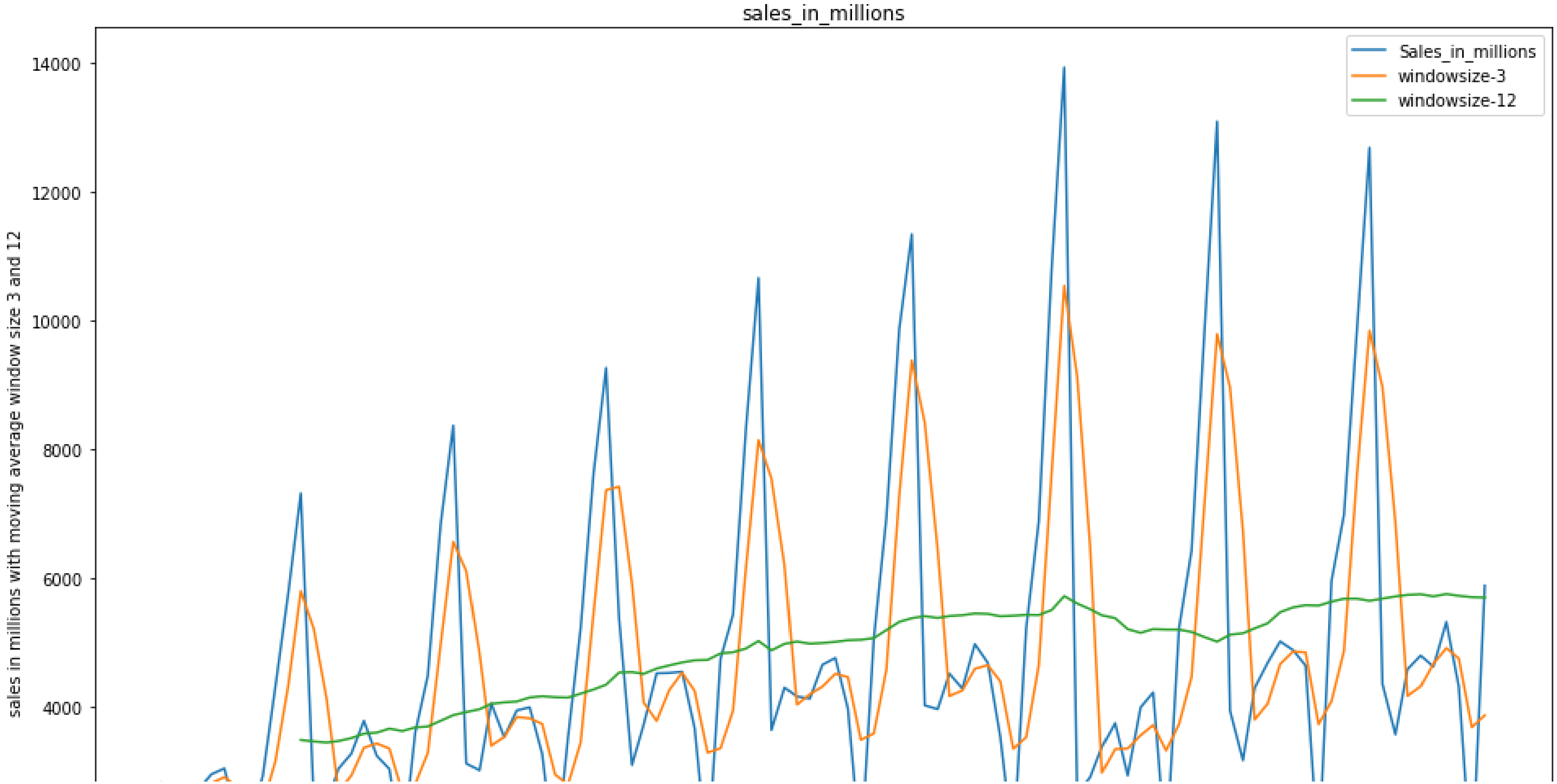


plot for choclate sales in millions from 1964 to 1967 to understand data in year

```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(choclate_sales_data['Date']['1964':'1966'], choclate_sales_data['Choclate_Sales']['1964':'1966'], label="choclate Sales_in_m


ax.set(xlabel="Date", ylabel="sales_in_millions from 1964 to 1967 ",
       title="sales_in_millions")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
ax.xaxis.set_major_formatter(DateFormatter("19%y-%m"))

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

plot for sales in millions with moving average window size 3 and 12

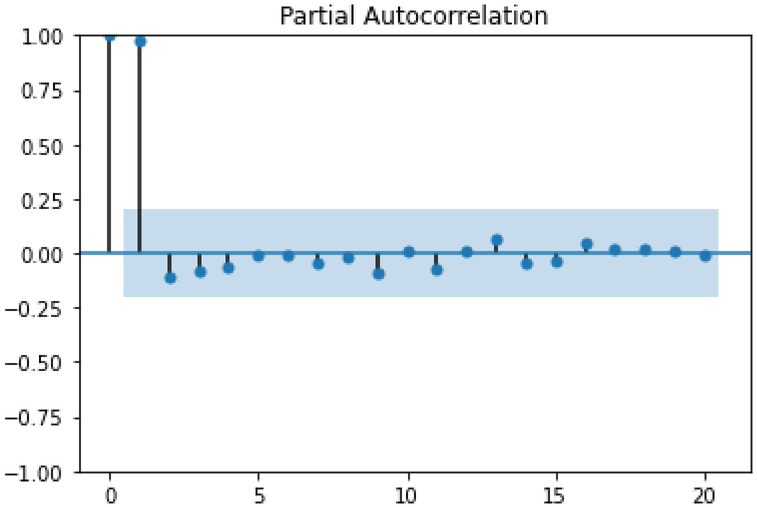
```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(choclate_sales_data['Date'], choclate_sales_data['Choclate_Sales'], label="Sales_in_millions")
ax.plot(choclate_sales_data['Date'], choclate_sales_data['Choclate_Sales'].rolling( window=3).mean(), label="windowsize-3")
ax.plot(choclate_sales_data['Date'], choclate_sales_data['Choclate_Sales'].rolling( window=12).mean(), label="windowsize-12")

ax.set(xlabel="Date", ylabel="sales in millions with moving average window size 3 and 12",
       title="sales_in_millions")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
ax.xaxis.set_major_formatter(DateFormatter("19%y-%m"))
plt.legend()
plt.show()
```

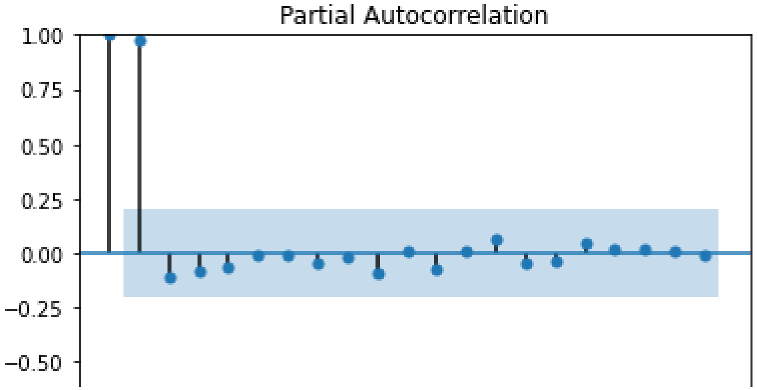


pacf plot for choclate sales with rolling window size as 12.

```
xt=choclate_sales_data['Choclate_Sales'].rolling( window=12).mean().dropna()
plot_pacf(xt)
```

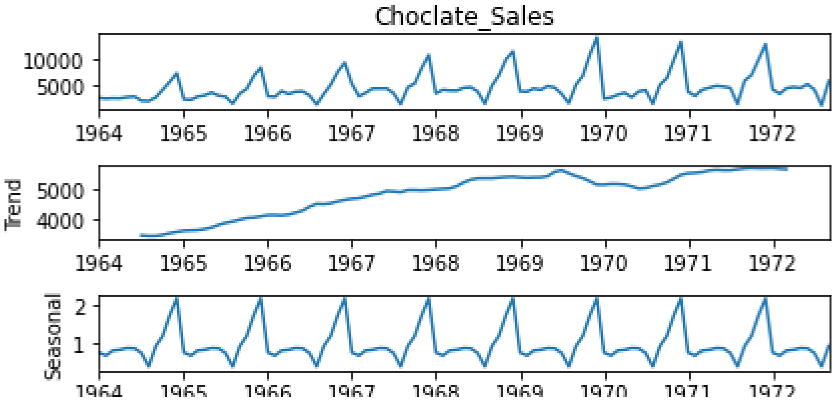
```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353: FutureWarning: The default method 'yw' can produce
  FutureWarning,
```





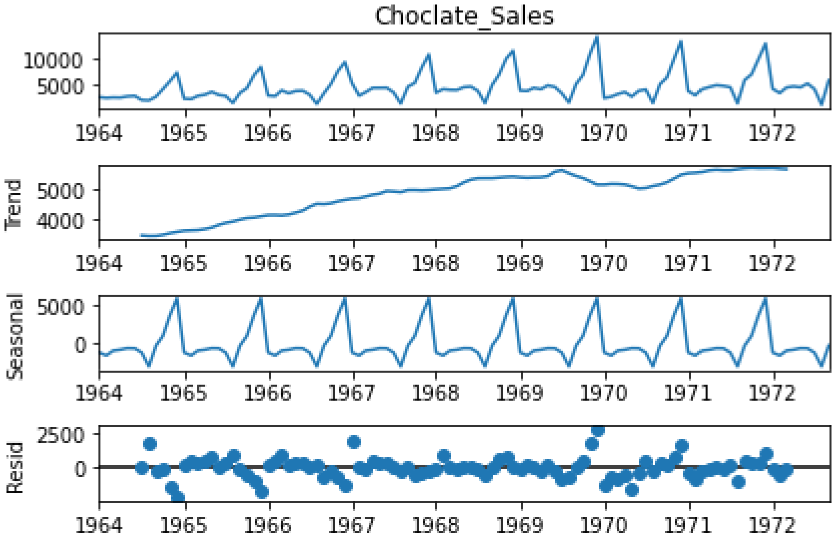
plot for multiplicative seasonal decompose

```
output = sm.tsa.seasonal_decompose(choclate_sales_data['Choclate_Sales'],period=12,model="multiplicative")
fig_multiplicative = output.plot()
```



plot for additive seasonal decompose

```
output_add= sm.tsa.seasonal_decompose(choclate_sales_data['Choclate_Sales'],period=12,model="additive")
fig_additive = output_add.plot()
```



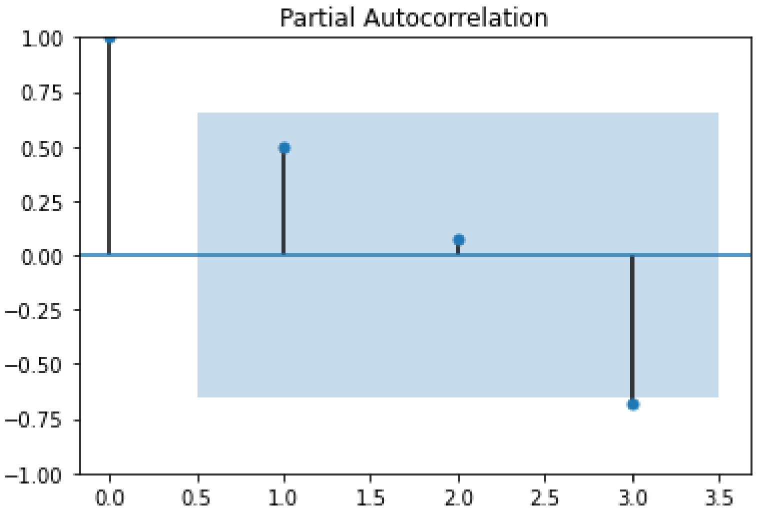
calculating sesonal average for 12 months

```
choclate_sales_seosonal_average = choclate_sales_data.groupby([choclate_sales_data.index.year]).mean()
```

pacf plot for seasonal average

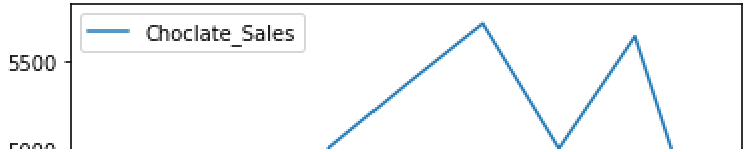
```
plot_pacf(choclate_sales_seosonal_average,lags=3);
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353: FutureWarning: The default method 'yw' can produce
  FutureWarning,
```




```
choclate_sales_seosonal_average.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f820f0e4a10>
```



Dickey fuller test

```
#Ho: It is non stationary
#H1: It is stationary
def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['p-value','#Lags Used']
    print(labels[0]+' : '+str(result[1]))
    print(labels[1]+' : '+str(result[2]))
    if result[1] <= 0.05:
        print("Data has no unit root and is stationary")
    else:
        print("Data is non-stationary ")
```

Dickey fuller test for orginal dataset

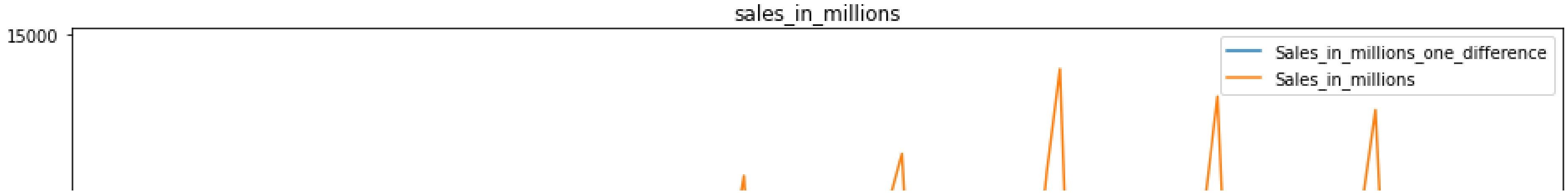
```
adfuller_test(choclate_sales_data['Choclate_Sales'])
```

```
p-value : 0.363915771660247
#Lags Used : 11
Data is non-stationary
```

Plot for 1st difference for choclate sales

```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(choclate_sales_data['Date'][1:],choclate_sales_data['Choclate_Sales'].diff().dropna(), label="Sales_in_millions_one_differen
```

```
ax.plot(choclate_sales_data['Date'][1:], choclate_sales_data['Choclate_Sales'][1:], label="Sales_in_millions")


ax.set(xlabel="Date", ylabel="sales_in_millions vs one_difference sales in millions",
       title="sales_in_millions")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
ax.xaxis.set_major_formatter(DateFormatter("19%y-%m"))
plt.legend()
plt.show()
```




```
adfuller_test(choclate_sales_data['Choclate_Sales'].diff().dropna().dropna())
```

```
p-value : 2.519620447387081e-10
#Lags Used : 11
Data has no unit root and is stationary
```

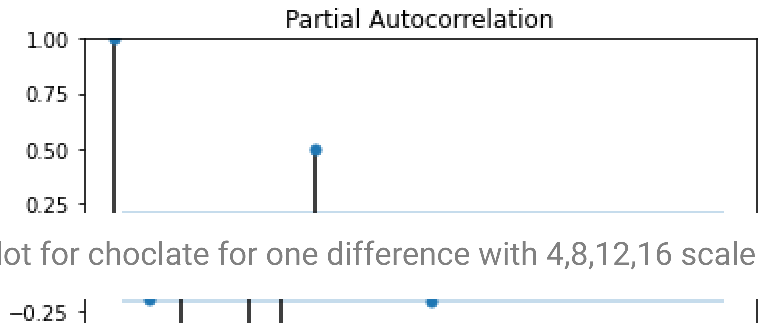
Spliting the data to test and train

```
dup_choclate_data=choclate_sales_data
size = int(len(dup_choclate_data) * 0.80)
```

```
train, test = dup_choclate_data[0:size], dup_choclate_data[size:len(dup_choclate_data)]
```

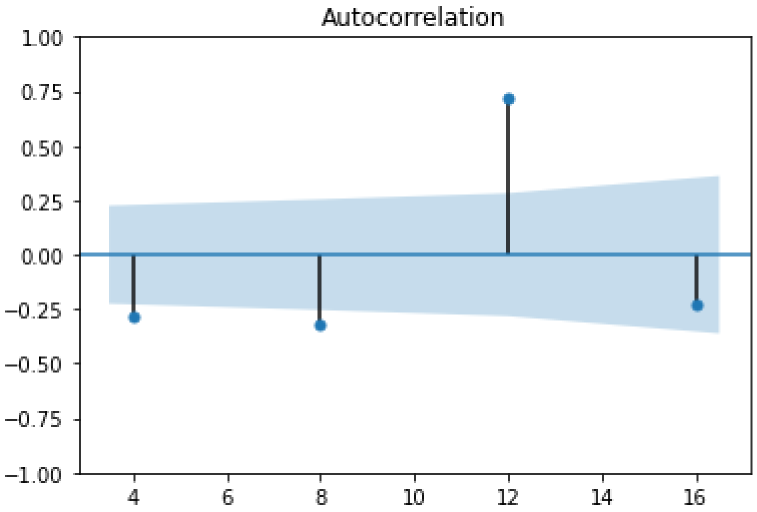
pacf for training dataset with one difference

```
plot_pacf(train['Choclate_Sales'].diff().dropna(),lags=36,method='ywm');
```



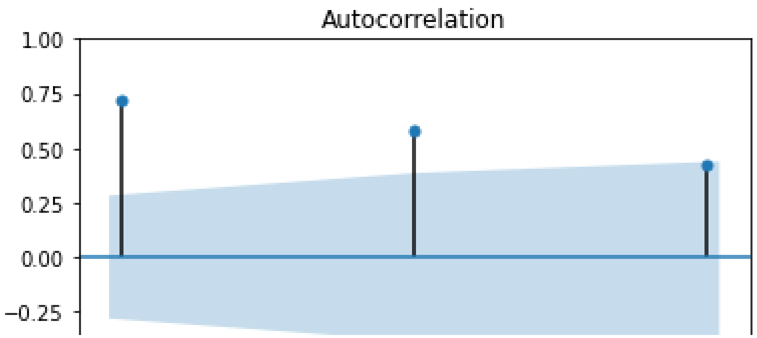
Acf plot for choclate for one difference with 4,8,12,16 scale

```
plot_acf(train['Choclate_Sales'].diff().dropna(),lags=[4,8,12,16]);
```



Acf plot for choclate for one difference with 12,24,36 scale

```
plot_acf(train['Choclate_Sales'].diff().dropna(),lags=[12,24,36]);
```



plotted pacf and acf plots on log data with 1 difference to understand data.Nono important information found

Double-click (or enter) to edit

setting index as date for test

```
test=test.set_index(test['Date'])
```

```
train_series=train['Choclate_Sales'].squeeze()
test_series=test['Choclate_Sales'].squeeze()
train_series_nd=train_series.values
test_series_nd=test_series.values
```

identifying p and q values based on lowest aic value for p(4) and q(4) combinations

```
best_aic=0
flag=0
for p in range(4):
  for q in range(4):
    model = ARIMA(train_series, order=(p ,1, q))
    model_fit = model.fit()
    aic=model_fit.aic
```

```
        if((aic>=best_aic) & (flag==0)):
            best_aic=aic
            flag=1
            best_p=p
            best_q=q
        if((aic<best_aic) & (flag==1)):
          best_aic=aic
          best_p=p
          best_q=q
    print('Best p and q combination for 1st difference arima model with low aic score')
    print(f'best p value {best_p}')
    print(f'best q value {best_q}')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was pro
  self._init_dates(dates, freq)
Best p and q combination for 1st difference arima model with low aic score
best p value 3
best q value 3
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:606: ConvergenceWarning: Maximum Likelihood optimization fa
  ConvergenceWarning)
```

```
#by above output p=3 and q=3
model = ARIMA(train_series, order=(3 ,1, 3))
model_fit = model.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:606: ConvergenceWarning: Maximum Likelihood optimization faile
  ConvergenceWarning)
```

```
forecast = model_fit.forecast(steps=21)
print(forecast)
```

```
1971-01-01    9058.386866
1971-02-01    4714.064332
1971-03-01    2589.389986
1971-04-01    1713.965706
1971-05-01    2505.223530
1971-06-01    3756.979106
1971-07-01    5243.936422
1971-08-01    6251.499297
1971-09-01    6795.980563
1971-10-01    6774.568354
1971-11-01    6456.942564
1971-12-01    5984.178085
1972-01-01    5581.065090
1972-02-01    5315.168609
1972-03-01    5234.645823
1972-04-01    5287.489956
1972-05-01    5422.182361
1972-06-01    5565.913668
1972-07-01    5678.528293
1972-08-01    5734.938787
1972-09-01    5740.055671
Freq: MS, Name: predicted_mean, dtype: float64
```

```
start = "1971-01-01"
end = "1972-09-01"
new_weeks=pd.date_range(start, end, freq='MS')
new_weeks
```

```
DatetimeIndex(['1971-01-01', '1971-02-01', '1971-03-01', '1971-04-01',
```

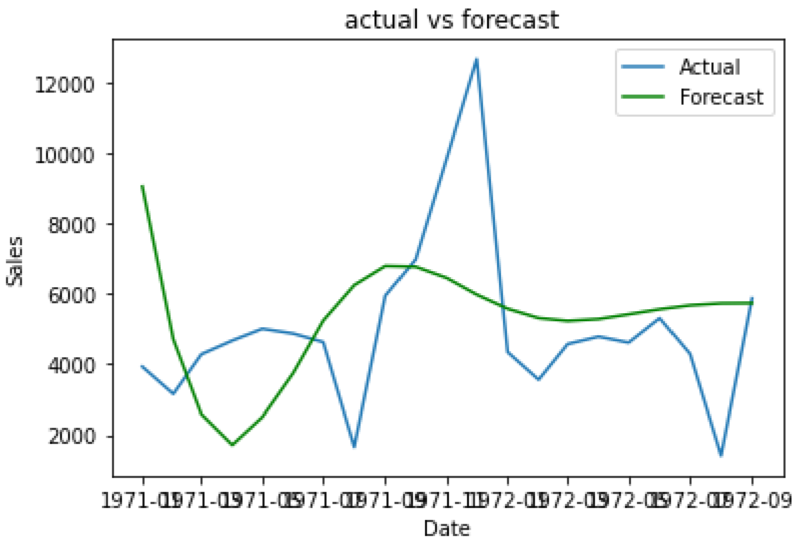
```
               '1971-05-01', '1971-06-01', '1971-07-01', '1971-08-01',
               '1971-09-01', '1971-10-01', '1971-11-01', '1971-12-01',
               '1972-01-01', '1972-02-01', '1972-03-01', '1972-04-01',
               '1972-05-01', '1972-06-01', '1972-07-01', '1972-08-01',
               '1972-09-01'],
              dtype='datetime64[ns]', freq='MS')
```

```
test_series
```

```
Date
1971-01-01     3934
1971-02-01     3162
1971-03-01     4286
1971-04-01     4676
1971-05-01     5010
1971-06-01     4874
1971-07-01     4633
1971-08-01     1659
1971-09-01     5951
1971-10-01     6981
1971-11-01     9851
1971-12-01    12670
1972-01-01     4348
1972-02-01     3564
1972-03-01     4577
1972-04-01     4788
1972-05-01     4618
1972-06-01     5312
1972-07-01     4298
1972-08-01     1413
1972-09-01     5877
Name: Choclate_Sales, dtype: int64
```

```
plt.plot(test_series, label="Actual")
plt.plot(new_weeks,forecast, color='green', label="Forecast")
plt.title(" actual vs forecast")
plt.xlabel("Date")
plt.ylabel("Sales")
```

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



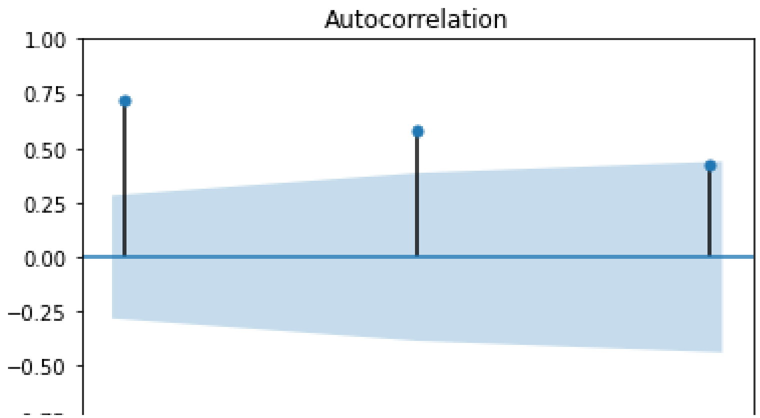

```
rmse = sqrt(mean_squared_error(forecast, test_series))
print(rmse)
print('ARIMA(3,1,3) RMSE: %.2f' % rmse)
```

```
2698.0304589000175
ARIMA(3,1,3) RMSE: 2698.03
```

Implementation of SARIMA

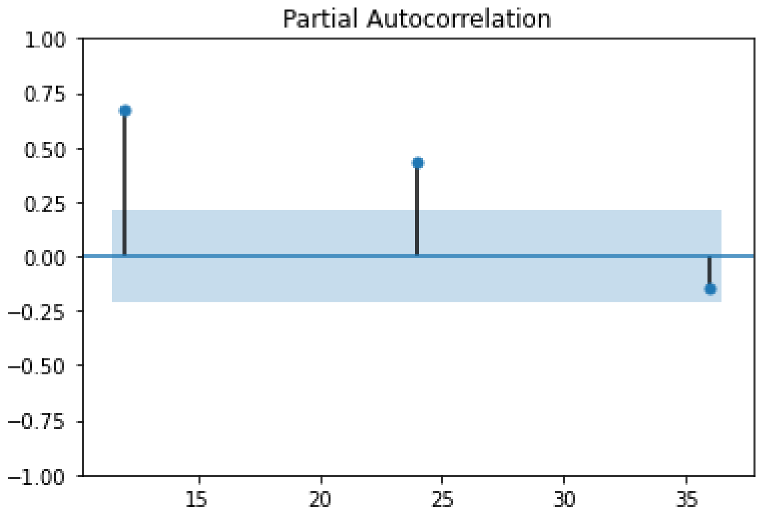
Considering 2 seasons 12 months and 4 months for 12 months below are pacf and acf plots

```
plot_acf(train['Choclate_Sales'].diff().dropna(),lags=[12,24,36]);
```




```
plot_pacf(train['Choclate_Sales'].diff().dropna(),lags=[12,24,36]);
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353: FutureWarning: The default method 'yw' can produce
  FutureWarning,
```



from above plots considering p=2 and q=2 for seasanol order 12

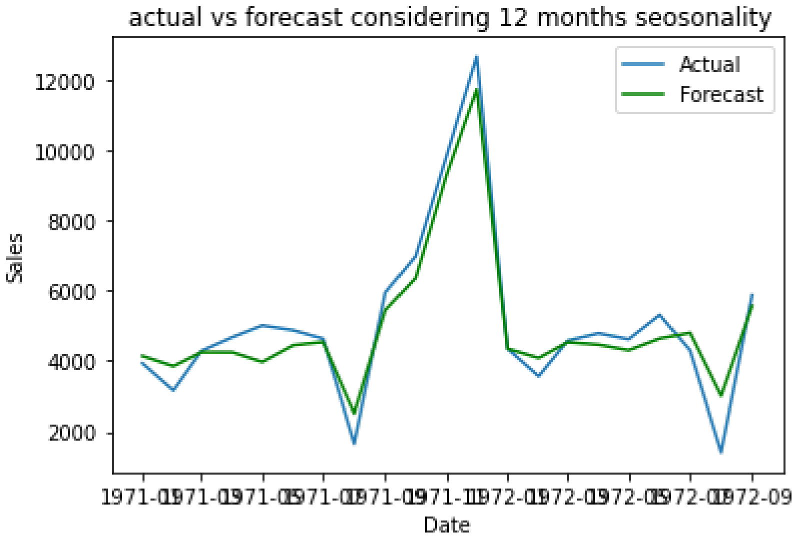
```
model_sarima=sm.tsa.statespace.SARIMAX(train_series,order=(3, 1, 3),seasonal_order=(2,0,2,12))
result_sarima=model_sarima.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py:997: UserWarning: Non-stationary starting seasonal
  warn('Non-stationary starting seasonal autoregressive'
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py:1009: UserWarning: Non-invertible starting seasona
  warn('Non-invertible starting seasonal moving average'
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:606: ConvergenceWarning: Maximum Likelihood optimization faile
  ConvergenceWarning)
```

```
forecast_sarima = result_sarima.forecast(steps=21)
print(forecast_sarima)
```

```
1971-01-01     4142.816256
1971-02-01     3850.772006
1971-03-01     4250.090705
1971-04-01     4248.090334
1971-05-01     3971.164830
1971-06-01     4447.504663
1971-07-01     4539.913665
1971-08-01     2515.877630
1971-09-01     5445.728632
1971-10-01     6363.176618
1971-11-01     9309.462009
1971-12-01    11732.190434
1972-01-01     4348.820337
1972-02-01     4084.143830
1972-03-01     4533.284592
1972-04-01     4462.727741
1972-05-01     4305.565106
1972-06-01     4639.195061
1972-07-01     4802.182737
1972-08-01     3006.430406
1972-09-01     5569.534639
Freq: MS, Name: predicted_mean, dtype: float64
```

```
plt.plot(test_series, label="Actual")
plt.plot(new_weeks,forecast_sarima, color='green', label="Forecast")
plt.title(" actual vs forecast considering 12 months seosonality")
plt.xlabel("Date")
plt.ylabel("Sales")
plt.legend()
plt.show()
```

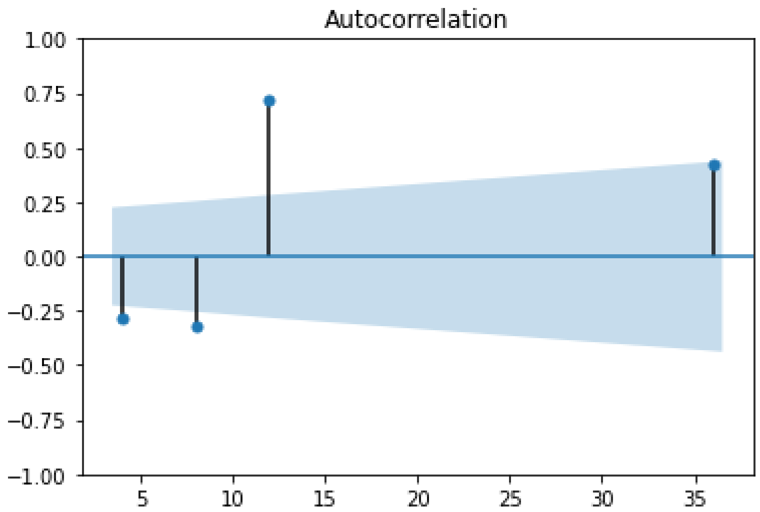



```
rmse = sqrt(mean_squared_error(forecast_sarima, test_series))
print(rmse)
print('SRIMA(3,1,3,2,0,2,12) RMSE: %.2f' % rmse)
```

```
629.9071923819017
SRIMA(3,1,3,2,0,2,12) RMSE: 629.91
```

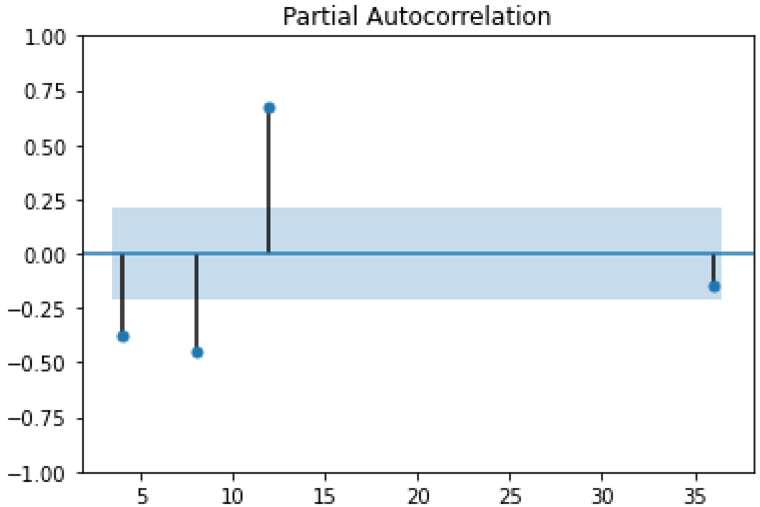
SARIMA FOR 4 months pacf and acf plots

```
plot_acf(train['Choclate_Sales'].diff().dropna(),lags=[4,8,12,36]);
```




```
plot_pacf(train['Choclate_Sales'].diff().dropna(),lags=[4,8,12,36]);
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353: FutureWarning: The default method 'yw' can produce
  FutureWarning,
```



from above plots considering p=3 and q=3 for 4 month seasonal arima

```
model_sarima=sm.tsa.statespace.SARIMAX(train_series,order=(3, 1, 3),seasonal_order=(3,0,3,4))
result_sarima=model_sarima.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py:997: UserWarning: Non-stationary starting seasonal
  warn('Non-stationary starting seasonal autoregressive'
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:606: ConvergenceWarning: Maximum Likelihood optimization faile
  ConvergenceWarning)
```

```
forecast_sarima = result_sarima.forecast(steps=21)
```

```
plt.plot(test_series, label="Actual")
plt.plot(new_weeks,forecast_sarima, color='green', label="Forecast")
plt.title(" actual vs forecast considering 4 month seosonality")
plt.xlabel("Date")
plt.ylabel("Sales")
plt.legend()
plt.show()
```


```
rmse = sqrt(mean_squared_error(forecast_sarima, test_series))
print(rmse)
print('SRIMA(3,1,3,3,0,3,4) RMSE: %.2f' % rmse)
```

```
804.1211825891829
SRIMA(3,1,3,3,0,3,4) RMSE: 804.12
```

12 months sesanolity gave good results with low RMSE

```
#End-of-sarima and end of arima
```

Implementation of vector auto regression

```
from statsmodels.tsa.stattools  import   grangercausalitytests
```

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.varmax import VARMAX
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from tqdm import tqdm_notebook
from itertools import product
```

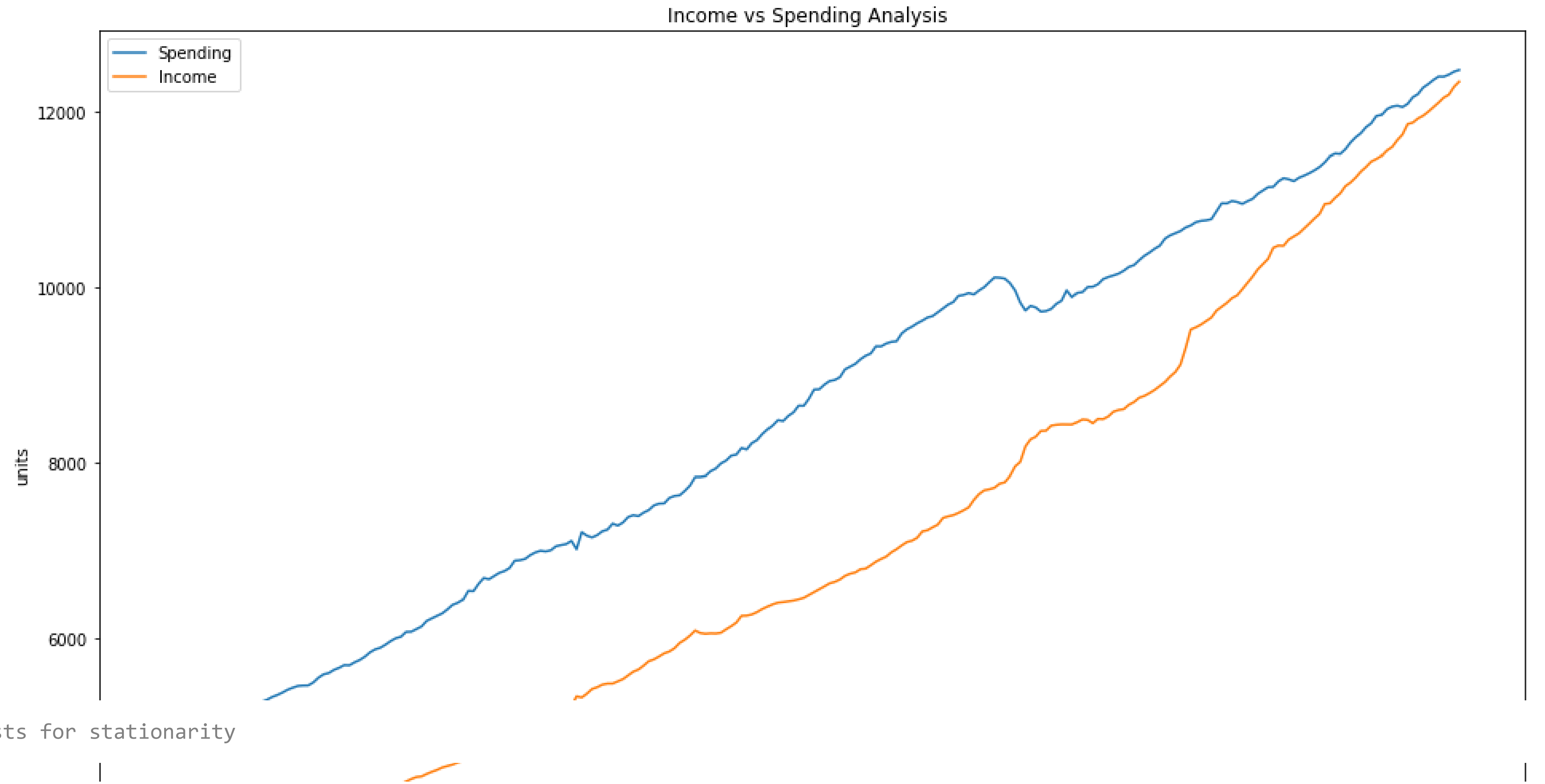
Importing income vs spending data set

```
data_income_vs_spending=pd.read_csv('Income_vs_Spending.csv',parse_dates=True)
data_income_vs_spending=data_income_vs_spending.set_index(data_income_vs_spending['Date'])
```

```
data_income_vs_spending.head()
```

Date	Date	Spending	Income
01-01-1995	01-01-1995	4851.2	3492.4
02-01-1995	02-01-1995	4850.8	3489.9
03-01-1995	03-01-1995	4885.4	3491.1
04-01-1995	04-01-1995	4890.2	3499.2
05-01-1995	05-01-1995	4933.1	3524.2

```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(data_income_vs_spending['Date'], data_income_vs_spending['Spending'], label="Spending")

ax.plot(data_income_vs_spending['Date'], data_income_vs_spending['Income'], label="Income")

ax.set(xlabel="Date", ylabel="units",
       title="Income vs Spending Analysis ")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
#ax.xaxis.set_major_formatter(DateFormatter("%y-%m"))
plt.legend()
plt.show()
```




```
# Tests for stationarity
```

```
adfuller_test(data_income_vs_spending['Income'])
```

```
p-value : 1.0
#Lags Used : 4
Data is non-stationary
```

```
adfuller_test(data_income_vs_spending['Spending'])
```

```
p-value : 0.9693009944428312
#Lags Used : 3
```

```
Data is non-stationary
```

```
difference_1_Income=data_income_vs_spending['Income'].diff().dropna()
adfuller_test(difference_1_Income.diff().dropna())
```

```
p-value : 4.7606749312953e-10
#Lags Used : 14
Data has no unit root and is stationary
```

```
difference_1_Spending=data_income_vs_spending['Spending'].diff().dropna()
adfuller_test(difference_1_Spending.diff().dropna())
```

```
p-value : 2.6878999679871547e-14
#Lags Used : 8
Data has no unit root and is stationary
```

granger causality test whether spending cause Income?

```
granger_spending = grangercausalitytests(data_income_vs_spending[['Income','Spending']], 8)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.6439  , p=0.4231  , df_denom=248, df_num=1
ssr based chi2 test:   chi2=0.6517  , p=0.4195  , df=1
likelihood ratio test: chi2=0.6509  , p=0.4198  , df=1
parameter F test:         F=0.6439  , p=0.4231  , df_denom=248, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=3.0760  , p=0.0479  , df_denom=245, df_num=2
ssr based chi2 test:   chi2=6.2776  , p=0.0433  , df=2
likelihood ratio test: chi2=6.2001  , p=0.0450  , df=2
parameter F test:         F=3.0760  , p=0.0479  , df_denom=245, df_num=2

Granger Causality
```

```
number of lags (no zero) 3
ssr based F test:         F=3.2731  , p=0.0218  , df_denom=242, df_num=3
ssr based chi2 test:   chi2=10.1034 , p=0.0177  , df=3
likelihood ratio test: chi2=9.9038  , p=0.0194  , df=3
parameter F test:         F=3.2731  , p=0.0218  , df_denom=242, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=2.6203  , p=0.0357  , df_denom=239, df_num=4
ssr based chi2 test:   chi2=10.8760 , p=0.0280  , df=4
likelihood ratio test: chi2=10.6443 , p=0.0309  , df=4
parameter F test:         F=2.6203  , p=0.0357  , df_denom=239, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=3.7026  , p=0.0030  , df_denom=236, df_num=5
ssr based chi2 test:   chi2=19.3761 , p=0.0016  , df=5
likelihood ratio test: chi2=18.6537 , p=0.0022  , df=5
parameter F test:         F=3.7026  , p=0.0030  , df_denom=236, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=3.0533  , p=0.0068  , df_denom=233, df_num=6
ssr based chi2 test:   chi2=19.3419 , p=0.0036  , df=6
likelihood ratio test: chi2=18.6192 , p=0.0049  , df=6
parameter F test:         F=3.0533  , p=0.0068  , df_denom=233, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=2.7799  , p=0.0086  , df_denom=230, df_num=7
ssr based chi2 test:   chi2=20.7285 , p=0.0042  , df=7
likelihood ratio test: chi2=19.8981 , p=0.0058  , df=7
parameter F test:         F=2.7799  , p=0.0086  , df_denom=230, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=2.5548  , p=0.0110  , df_denom=227, df_num=8
ssr based chi2 test:   chi2=21.9692 , p=0.0050  , df=8
likelihood ratio test: chi2=21.0358 , p=0.0071  , df=8
parameter F test:         F=2.5548  , p=0.0110  , df_denom=227, df_num=8
```

Test for Income granger cause spending?

```
granger_spending = grangercausalitytests(data_income_vs_spending[['Spending', 'Income']], 8)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.5856  , p=0.4448  , df_denom=248, df_num=1
ssr based chi2 test:   chi2=0.5927  , p=0.4414  , df=1
likelihood ratio test: chi2=0.5920  , p=0.4416  , df=1
parameter F test:         F=0.5856  , p=0.4448  , df_denom=248, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.6461  , p=0.5250  , df_denom=245, df_num=2
ssr based chi2 test:   chi2=1.3186  , p=0.5172  , df=2
likelihood ratio test: chi2=1.3151  , p=0.5181  , df=2
parameter F test:         F=0.6461  , p=0.5250  , df_denom=245, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.6772  , p=0.1725  , df_denom=242, df_num=3
ssr based chi2 test:   chi2=5.1771  , p=0.1593  , df=3
likelihood ratio test: chi2=5.1240  , p=0.1629  , df=3
parameter F test:         F=1.6772  , p=0.1725  , df_denom=242, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=1.5428  , p=0.1905  , df_denom=239, df_num=4
ssr based chi2 test:   chi2=6.4034  , p=0.1710  , df=4
likelihood ratio test: chi2=6.3222  , p=0.1763  , df=4
parameter F test:         F=1.5428  , p=0.1905  , df_denom=239, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.4318  , p=0.2135  , df_denom=236, df_num=5
ssr based chi2 test:   chi2=7.4925  , p=0.1865  , df=5
likelihood ratio test: chi2=7.3811  , p=0.1938  , df=5
```

```
parameter F test:         F=1.4318  , p=0.2135  , df_denom=236, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=1.5207  , p=0.1721  , df_denom=233, df_num=6
ssr based chi2 test:   chi2=9.6330  , p=0.1410  , df=6
likelihood ratio test: chi2=9.4492  , p=0.1498  , df=6
parameter F test:         F=1.5207  , p=0.1721  , df_denom=233, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=1.7036  , p=0.1090  , df_denom=230, df_num=7
ssr based chi2 test:   chi2=12.7027 , p=0.0797  , df=7
likelihood ratio test: chi2=12.3844 , p=0.0886  , df=7
parameter F test:         F=1.7036  , p=0.1090  , df_denom=230, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=1.4916  , p=0.1612  , df_denom=227, df_num=8
ssr based chi2 test:   chi2=12.8263 , p=0.1180  , df=8
likelihood ratio test: chi2=12.5006 , p=0.1302  , df=8
parameter F test:         F=1.4916  , p=0.1612  , df_denom=227, df_num=8
```

Performing 2nd order differencing

```
income_vs_spend_diff1=data_income_vs_spending[['Income','Spending']].diff().dropna()
income_vs_spend_diff2=income_vs_spend_diff1.diff().dropna()
```

```
train_length=len(income_vs_spend_diff2)*0.95
test_length=len(income_vs_spend_diff2)*0.05
print(train_length)
print(test_length)
```

```
237.5
12.5
```

```
data_income_vs_spending.index
```

```
Index(['01-01-1995', '02-01-1995', '03-01-1995', '04-01-1995', '05-01-1995',
       '06-01-1995', '07-01-1995', '08-01-1995', '09-01-1995', '10-01-1995',
       ...
       '03-01-2015', '04-01-2015', '05-01-2015', '06-01-2015', '07-01-2015',
       '08-01-2015', '09-01-2015', '10-01-2015', '11-01-2015', '12-01-2015'],
      dtype='object', name='Date', length=252)
```

splititng data into test and train

```
train=income_vs_spend_diff2[:237]
test=data_income_vs_spending[239:]
```

```
train.head()
len(train)
len(test)
```

```
13
```

building var model

```
model_1 = VAR(train)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
```

```
sorted_order=model_1.select_order(maxlags=20)
print(sorted_order.summary())
```

```
 VAR Order Selection (* highlights the minimums)
==================================================
       AIC         BIC         FPE         HQIC
```

```
--------------------------------------------------
0        14.89       14.92   2.924e+06       14.90
1        14.31       14.41   1.644e+06       14.35
2        14.09       14.24   1.313e+06       14.15
3        13.98      14.19*   1.173e+06       14.06
4        13.95       14.23   1.147e+06       14.07
5        13.91       14.25   1.101e+06      14.05*
6        13.92       14.32   1.109e+06       14.08
7        13.91       14.38   1.101e+06       14.10
8       13.85*       14.38  1.039e+06*       14.07
9        13.87       14.46   1.054e+06       14.11
10       13.90       14.55   1.088e+06       14.16
11       13.94       14.65   1.129e+06       14.22
12       13.96       14.74   1.163e+06       14.28
13       13.96       14.80   1.157e+06       14.30
14       13.99       14.89   1.196e+06       14.36
15       13.98       14.94   1.178e+06       14.37
16       13.98       15.01   1.187e+06       14.40
17       14.00       15.09   1.212e+06       14.44
18       14.01       15.17   1.227e+06       14.48
19       14.01       15.22   1.222e+06       14.50
20       14.04       15.32   1.268e+06       14.56
--------------------------------------------------
```

from above information considering lowest BIC value at lag p =3

```
train
```

	Income	Spending
Date		
03-01-1995	3.7	35.0
04-01-1995	6.9	-29.8
05-01-1995	16.9	38.1
06-01-1995	-0.3	1.5
07-01-1995	-6.2	-51.7
...	...	...

```
var= VAR(train)
model_var = var.fit(3)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provid
  self._init_dates(dates, freq)
```

```
print(model_var.summary())
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Sun, 26, Jun, 2022
Time:                     19:11:21
--------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                    14.0573
Nobs:                     234.000    HQIC:                   13.9339
Log likelihood:          -2270.58    FPE:                1.03569e+06
AIC:                      13.8505    Det(Omega_mle):         976399.
--------------------------------------------------------------------
Results for equation Income
==============================================================================
```

```
                    coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const                  0.330041         1.817568            0.182           0.856
L1.Income             -0.605007         0.065226           -9.276           0.000
L1.Spending           -0.100599         0.049710           -2.024           0.043
L2.Income             -0.393490         0.071568           -5.498           0.000
L2.Spending           -0.194326         0.059854           -3.247           0.001
L3.Income             -0.045411         0.066016           -0.688           0.492
L3.Spending           -0.176303         0.049017           -3.597           0.000
==============================================================================

Results for equation Spending
==============================================================================
                    coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const                  0.091217         2.379735            0.038           0.969
L1.Income              0.236518         0.085400            2.770           0.006
L1.Spending           -0.844391         0.065085          -12.974           0.000
L2.Income              0.181278         0.093704            1.935           0.053
L2.Spending           -0.526596         0.078367           -6.720           0.000
L3.Income              0.126827         0.086435            1.467           0.142
L3.Spending           -0.217811         0.064178           -3.394           0.001
==============================================================================

Correlation matrix of residuals
              Income  Spending
Income      1.000000 -0.214690
Spending   -0.214690  1.000000
```

```
predicted_values = model_var.forecast(y=train.values[-8:],steps=13)
predicted_values
```

```
array([[ 8.93322304e+00,  1.95505315e+01],
       [-1.84502833e+00, -6.77706274e+00],
       [ 2.33948966e+00,  2.10265318e+00],
       [-3.10642115e+00, -1.02195474e+00],
       [ 2.26169256e+00,  7.78395148e-01],
```

```
       [-1.72608697e-01, -2.17363148e-01],
       [-2.63639703e-01,  6.26422272e-02],
       [ 3.53463989e-01,  1.76440563e-01],
       [ 2.36170176e-01, -2.94938810e-02],
       [ 1.76803407e-02,  9.60614628e-02],
       [ 1.75323691e-01,  7.90274864e-02],
       [ 1.84870234e-01,  5.49505755e-02],
       [ 1.10581592e-01,  6.00283336e-02]])
```

```
idx = pd.date_range(start='12-01-2014   ',periods=13,freq='MS')
predictions = pd.DataFrame(predicted_values,index=idx,columns=['DF2_Spending','DF2_Income'])
```

```
predictions
```

DF2_Spending DF2_Income

Reverting back 2nd order differenced terms

```
predictions['DF1_Spending']= (data_income_vs_spending['Spending'][-14]-data_income_vs_spending['Spending'][-15])+predictions['DF2_Spe
predictions['Spending_prediction'] = data_income_vs_spending['Spending'][-14]+ predictions['DF1_Spending'].cumsum()
```

2015-03-01 -3.106421 -1.021955

```
predictions['DF1_Income']= (data_income_vs_spending['Income'][-14]-data_income_vs_spending['Income'][-15])+predictions['DF2_Income']
predictions['Income_prediction'] = data_income_vs_spending['Income'][-14]+ predictions['DF1_Income'].cumsum()
```

2015-05-01 -0.172609 -0.217363

```
test
```

	Date	Spending	Income

```
predictions.index=test.index
```

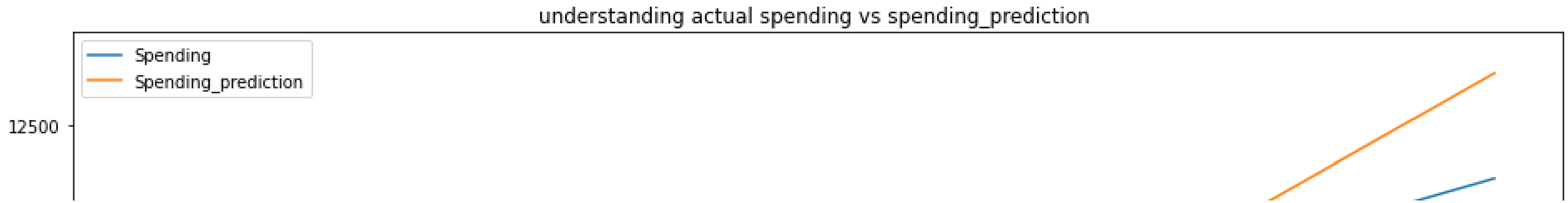
12-01-2014	12-01-2014	12062.0	11670.1

```
predictions
```

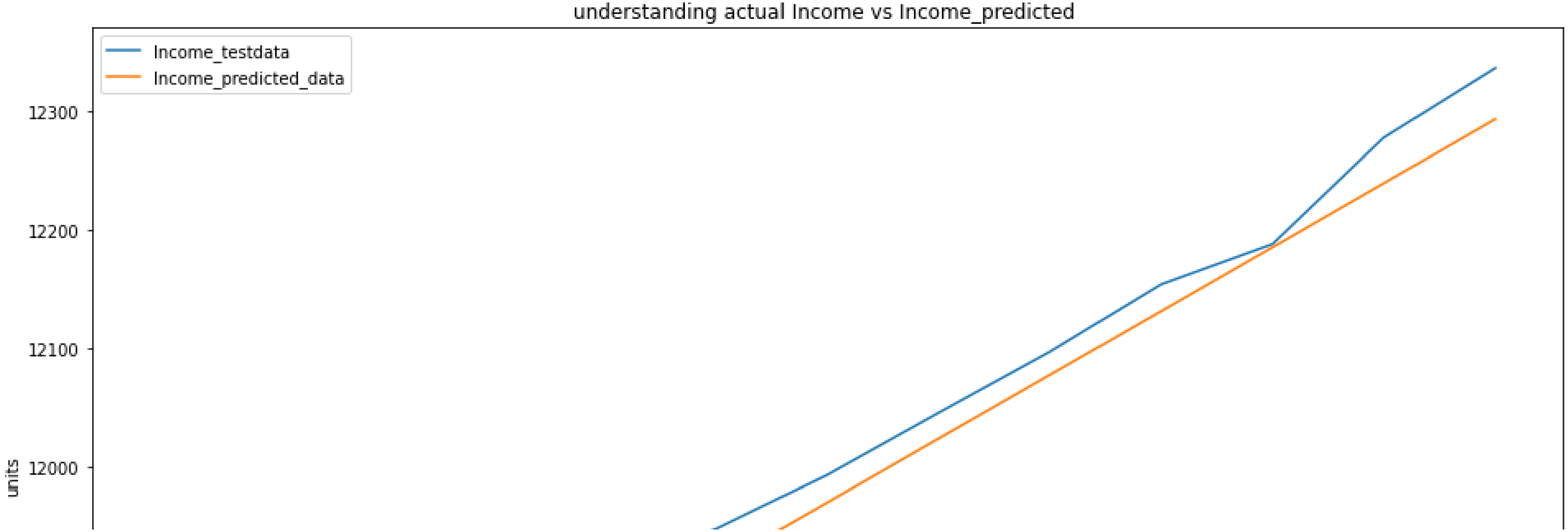
Date	DF2_Spending	DF2_Income	DF1_Spending	Spending_prediction	DF1_Income	Income_prediction
12-01-2014	8.933223	19.550532	37.333223	12088.733223	58.650532	11650.150532
01-01-2015	-1.845028	-6.777063	35.488195	12124.221418	51.873469	11702.024000
02-01-2015	2.339490	2.102653	37.827684	12162.049102	53.976122	11756.000122
03-01-2015	-3.106421	-1.021955	34.721263	12196.770365	52.954167	11808.954290
04-01-2015	2.261693	0.778395	36.982956	12233.753321	53.732562	11862.686852
05-01-2015	-0.172609	-0.217363	36.810347	12270.563668	53.515199	11916.202051
06-01-2015	-0.263640	0.062642	36.546707	12307.110376	53.577841	11969.779893
07-01-2015	0.353464	0.176441	36.900171	12344.010547	53.754282	12023.534175
08-01-2015	0.236170	-0.029494	37.136342	12381.146889	53.724788	12077.258963
09-01-2015	0.017680	0.096061	37.154022	12418.300910	53.820850	12131.079813
10-01-2015	0.175324	0.079027	37.329346	12455.630256	53.899877	12184.979690
11-01-2015	0.184870	0.054951	37.514216	12493.144472	53.954828	12238.934517
12-01-2015	0.110582	0.060028	37.624797	12530.769269	54.014856	12292.949373

```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(predictions.index,test['Spending'], label="Spending")
```

```
ax.plot(predictions.index, predictions['Spending_prediction'], label="Spending_prediction")


ax.set(xlabel="Date", ylabel="units",
       title="understanding actual spending vs spending_prediction ")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
#ax.xaxis.set_major_formatter(DateFormatter("%y-%m"))
plt.legend()
plt.show()
```




```
fig, ax = plt.subplots(figsize=(16,10))

ax.plot(predictions.index,test['Income'], label="Income_testdata")

ax.plot(predictions.index, predictions['Income_prediction'], label="Income_predicted_data")


ax.set(xlabel="Date", ylabel="units",
       title="understanding actual Income vs Income_predicted ")

#ax.xaxis.set_major_locator(mdates.WeekdayLocator(interval=2))
#ax.xaxis.set_major_formatter(DateFormatter("%y-%m"))
plt.legend()
plt.show()
```




```
rmse = sqrt(mean_squared_error(test['Income'], predictions['Income_prediction']))
print(rmse)
print('Income predictions RMSE: %.2f' % rmse)
```

```
42.36340280152077
Income predictions RMSE: 42.36
```

```
rmse = sqrt(mean_squared_error(test['Spending'], predictions['Spending_prediction']))
print(rmse)
print('Spending predictions RMSE: %.2f' % rmse)
```

```
43.34542973530263
Spending predictions RMSE: 43.35
```

Based on my analysis based on granger causality test which was performed for 3 lags I came to conclusion that Chi square test result for income granger cause spending p value is greater than 0.05 at lag 3 so that means it rejects null hypothesis of alpha=0 and concludes there is dependency of income for predicting spending and. For spending granger cause income test p values at lag 3 is less than 0.05 accepting null hypothesis “spending does not granger cause income” so spending does not have much impact in income prediction.

✓ 0s completed at 1:11 PM