PRACTICAL N0 – 1

Aim: Arima used for acf and pacf

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

#Load time Series data
#load built-in AirPassengers dataset
url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
df=pd.read_csv(url, parse_dates=['Month'],index_col='Month')
df_columns=['Passengers']
df.head()
```

Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
```

```
[ ] #Load time Series data
    #load built-in AirPassengers dataset
    url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
    df=pd.read_csv(url, parse_dates=['Month'],index_col='Month')
    df_columns=['Passengers']
    df.head()
```

Month	Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

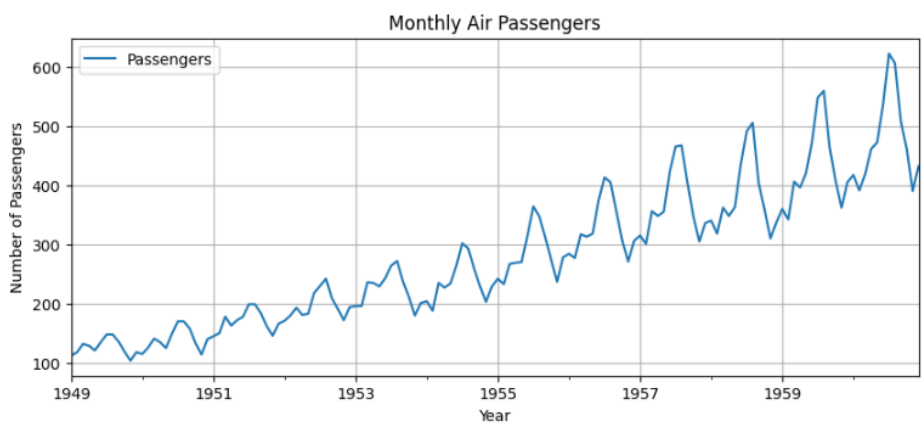
Code:

```
#plot the time series
df.plot(title='Monthly Air Passengers',figsize=(10,4))
plt.ylabel("Number of Passengers")
plt.xlabel("Year")
```

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

Output:

```
#plot the time series
df.plot(title='Monthly Air Passengers',figsize=(10,4))
plt.ylabel("Number of Passengers")
plt.xlabel("Year")
plt.grid()
plt.show()
```



Code:

```
#test for stationary(adf)
def test_stationarity(series):
  result = adfuller(series)
  print(f'ADF statistics: {result[0]:.4f}')
  print(f'p-value: {result[1]:.4f}')
  if result[1] <= 0.05:
    print('Stationary')
  else:
    print('Not Stationary')
test_stationarity(df['Passengers'])
```

Output:

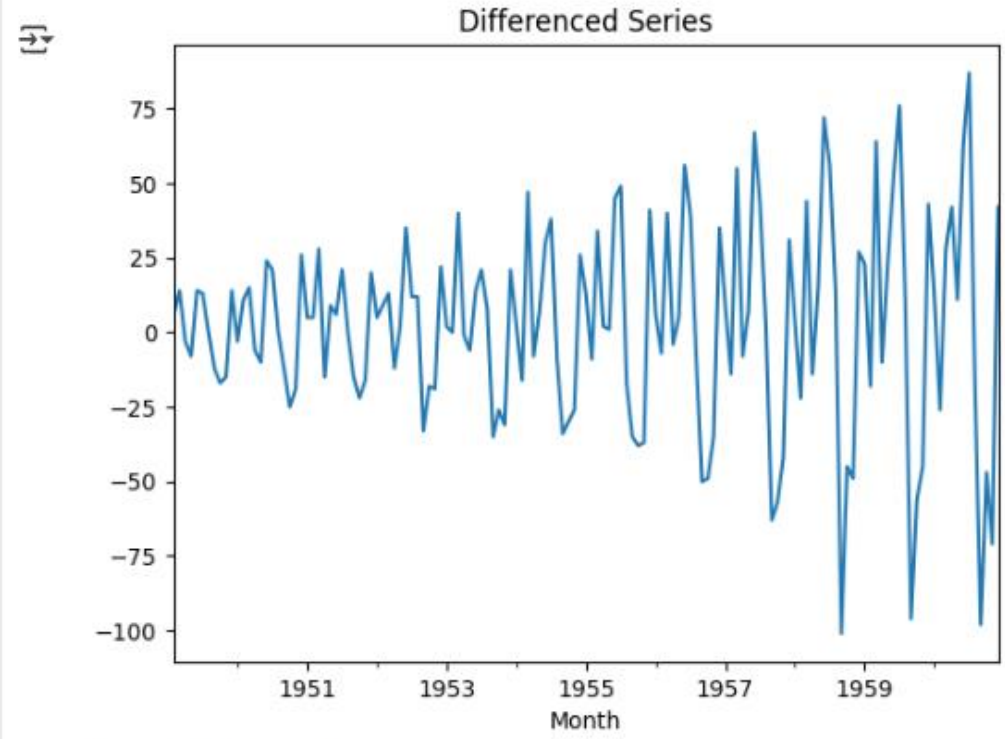
```
[ ]  #test for stationary(adf)
     def test_stationarity(series):
       result = adfuller(series)
       print(f'ADF statistics: {result[0]:.4f}')
       print(f'p-value: {result[1]:.4f}')
       if result[1] <= 0.05:
         print('Stationary')
       else:
         print('Not Stationary')
     test_stationarity(df['Passengers'])

     ADF statistics: 0.8154
     p-value: 0.9919
     Not Stationary
```

Code:

```
#make stationary (Differencing)
```

```
df_diff = df['Passengers'].diff().dropna()
df_diff.plot(title='Differenced Series')
plt.show()

test_stationarity(df_diff)
```

Output:

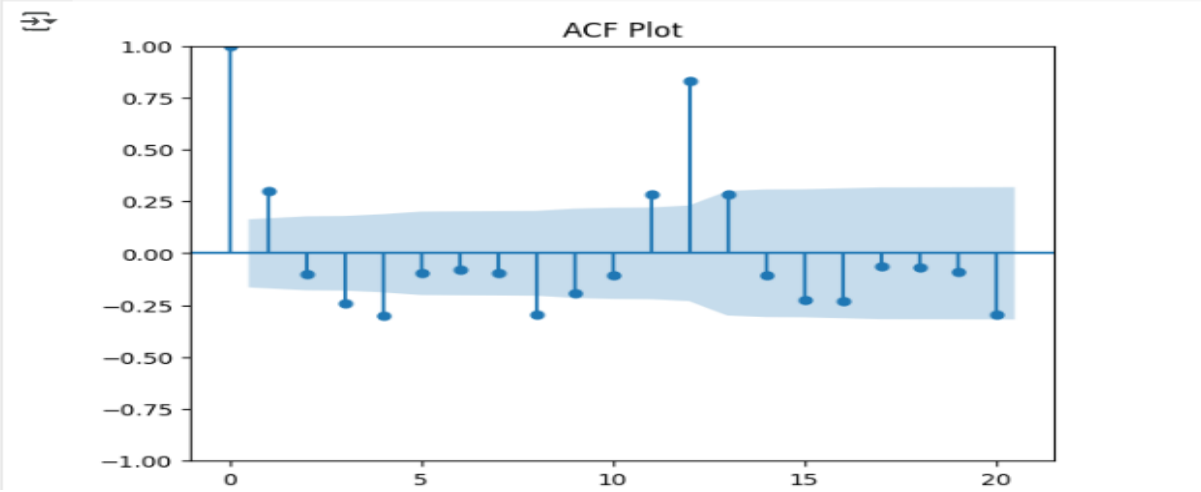


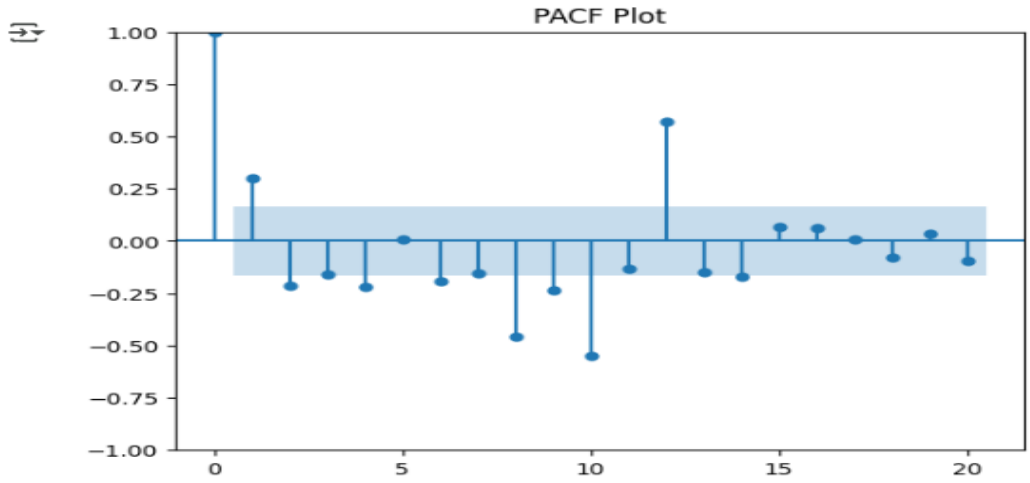

```
ADF statistics: -2.8293
p-value: 0.0542
Not Stationary
```

Code:

```
#plot Acf and pacf
plot_acf(df_diff,lags=20)
plt.title("ACF Plot")
plt.show()

plot_pacf(df_diff,lags=20)
plt.title("PACF Plot")
plt.show()
```

Output:

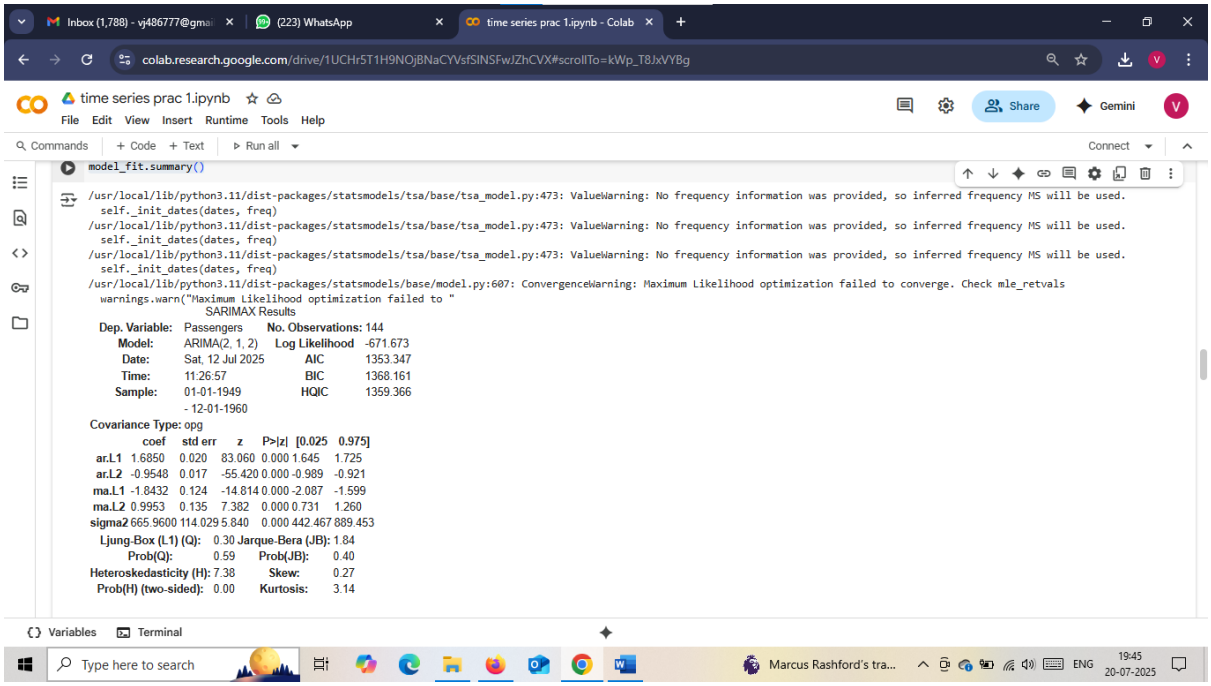




Code:

```
#Arima
model=ARIMA(df['Passengers'],order=(2,1,2))
model_fit=model.fit()
model_fit.summary()
```

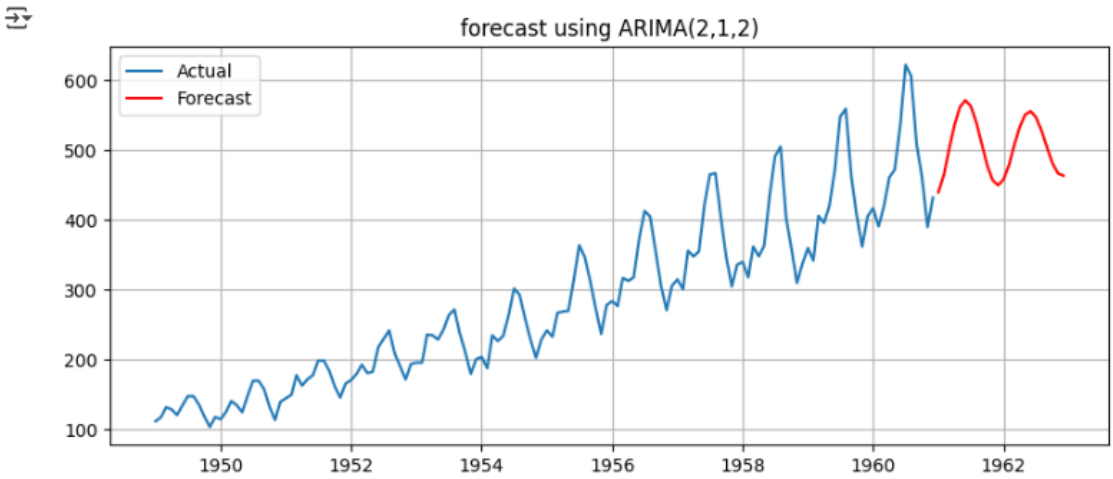
Output:



Code:

```
#forecast and plot
forecast = model_fit.forecast(steps=24)
forecast.index =pd.date_range(start=df.index[-1] +
pd.DateOffset(months=1), periods=24, freq='MS')

plt.figure(figsize=(10, 4))
plt.plot(df['Passengers'], label='Actual')
plt.plot(forecast, label='Forecast', color='red')
plt.title("forecast using ARIMA(2,1,2)")
plt.legend()
plt.grid()
plt.show()
```

Output:



Code:

```
#evaluate forecast(train/test)

#train
train=df['Passengers'][:-24]
test=df['Passengers'][-24:]

model=ARIMA(train,order=(2,1,2))
model_fit=model.fit()
preds = model_fit.forecast(steps=24)

#evaluate
mse=mean_squared_error(test,preds)
print(f'Mean squared error: {mse:.2f}')
```

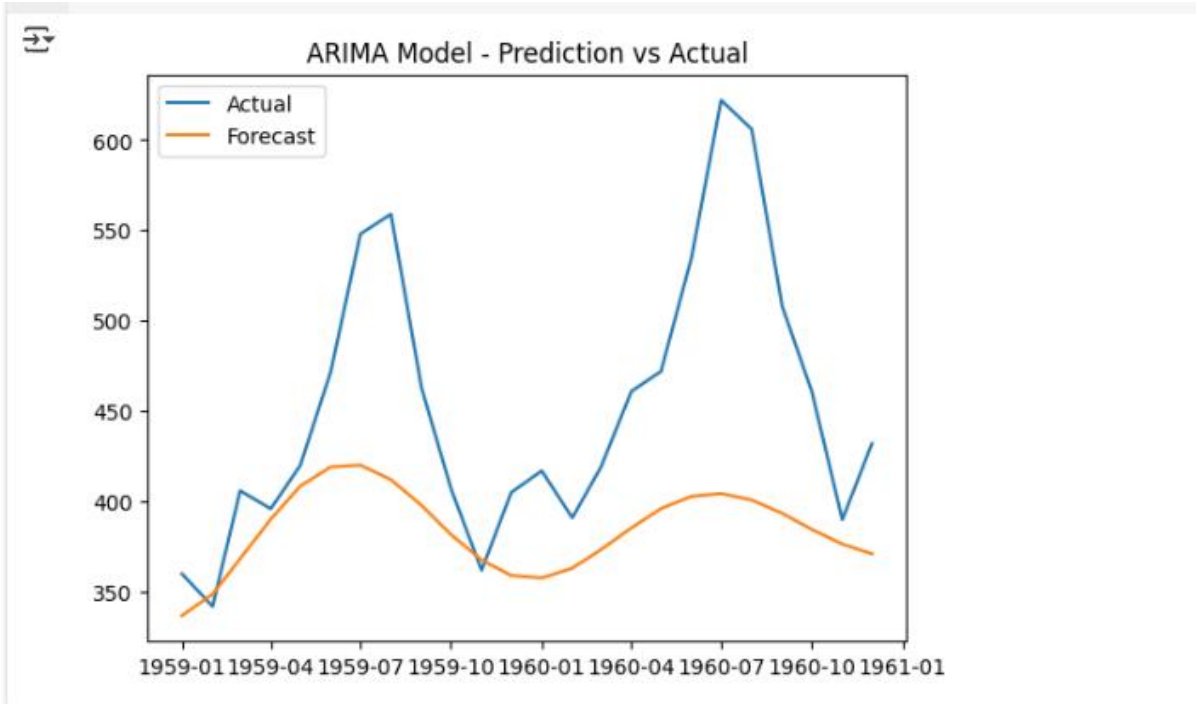
Output:

```
#evaluate
mse=mean_squared_error(test,preds)
print(f'Mean squared error: {mse:.2f}')

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
Mean squared error: 8225.46
```

Code:

```
plt.plot(test.index,test.values,label='Actual')
plt.plot(test.index,preds,label='Forecast')
plt.title("ARIMA Model - Prediction vs Actual")
plt.legend()
plt.show()
```

Output:

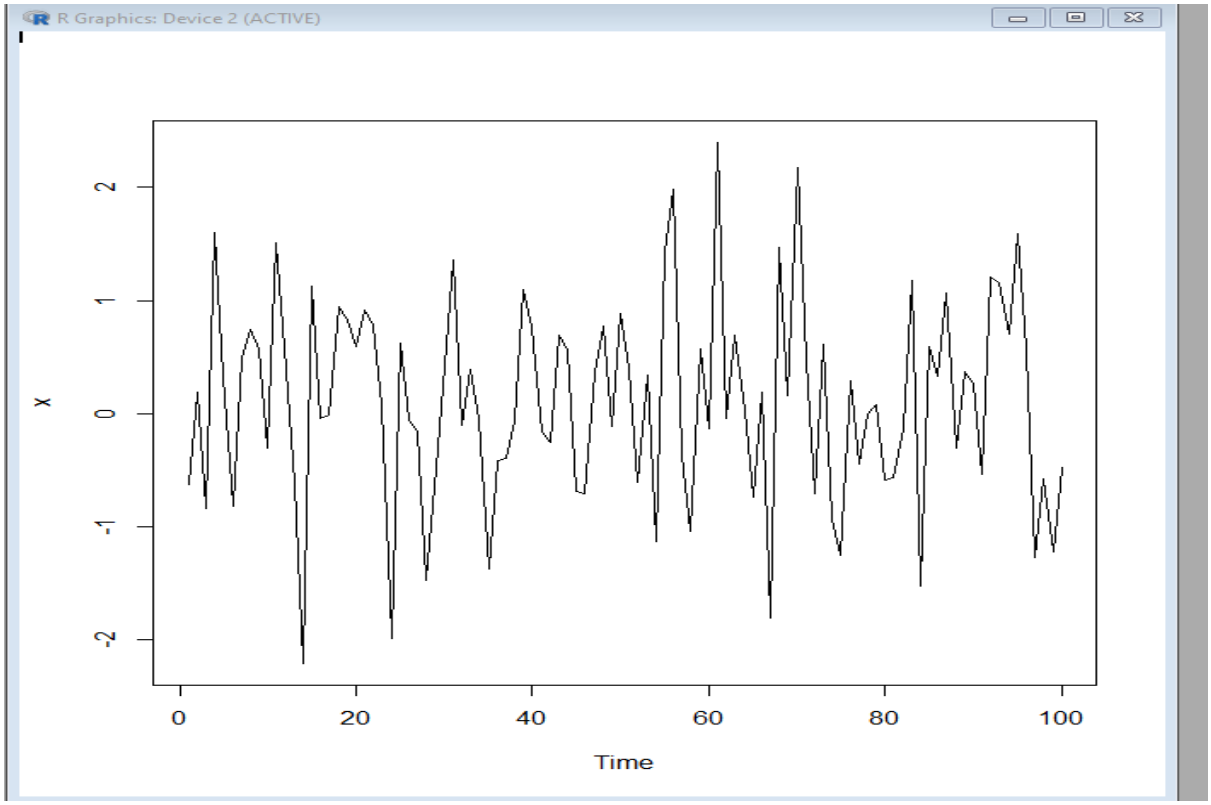


#Used in r studio

Code:

```
set.seed(1)

x<-rnorm(100,mean=0,sd=1)

plot(x,type='l',xlab="Time")
```

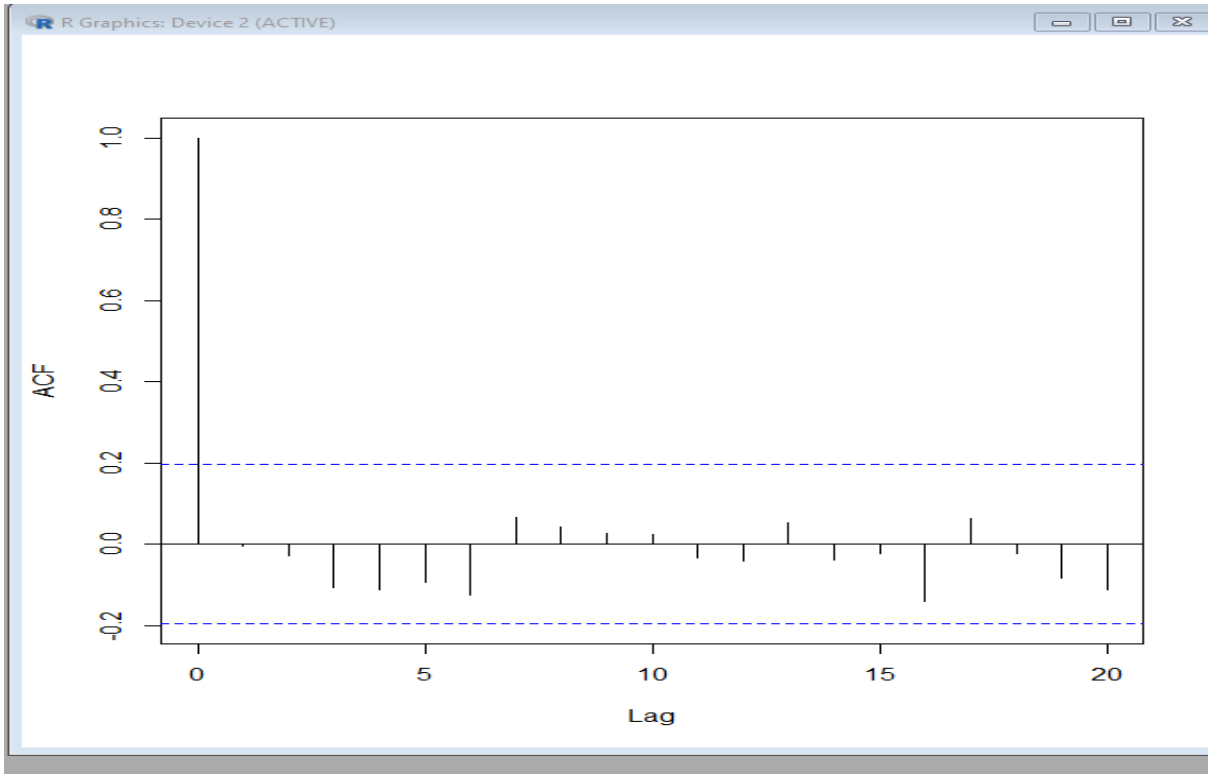
Output:



Code:

```
acf(x,main="")
```

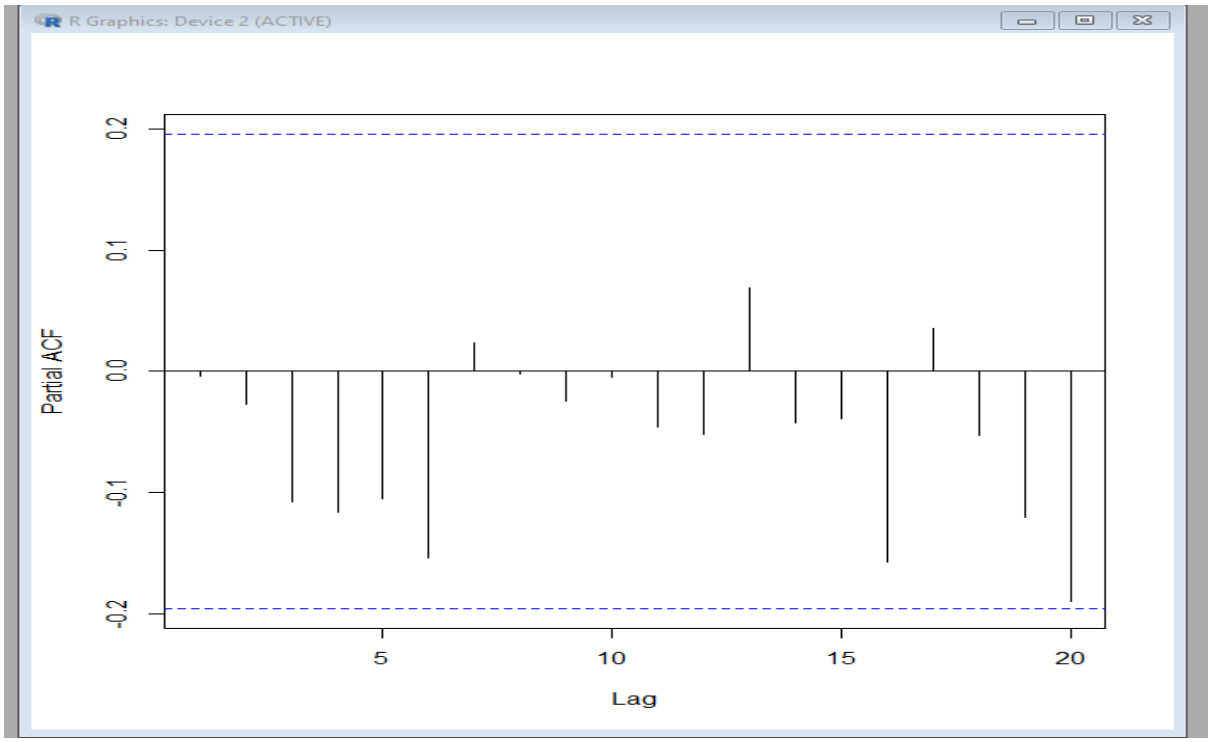
Output:



Code:

```
pacf(x,main="")
```

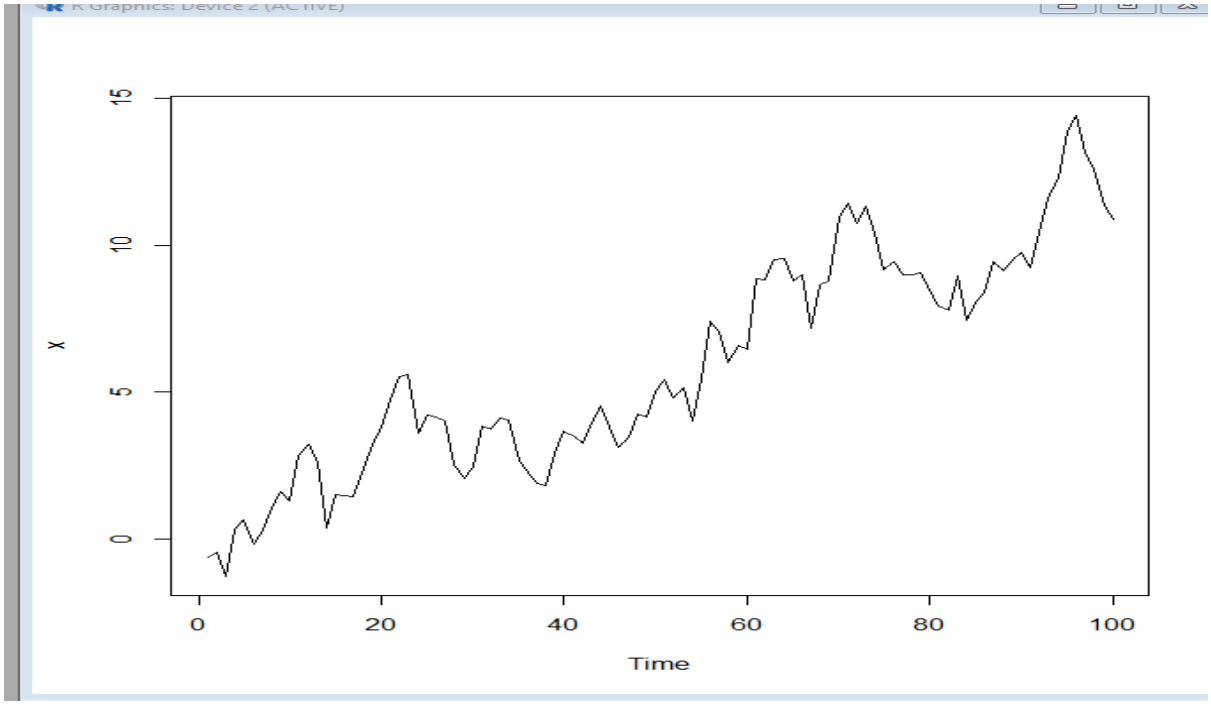
Output:



Code:

```
set.seed(1)
x<-w<-rnorm(100,mean=0,sd=1)
for(t in 2:100)x[t]<-x[t-1]+w[t]
plot(x,type='l',xlab="Time")
```

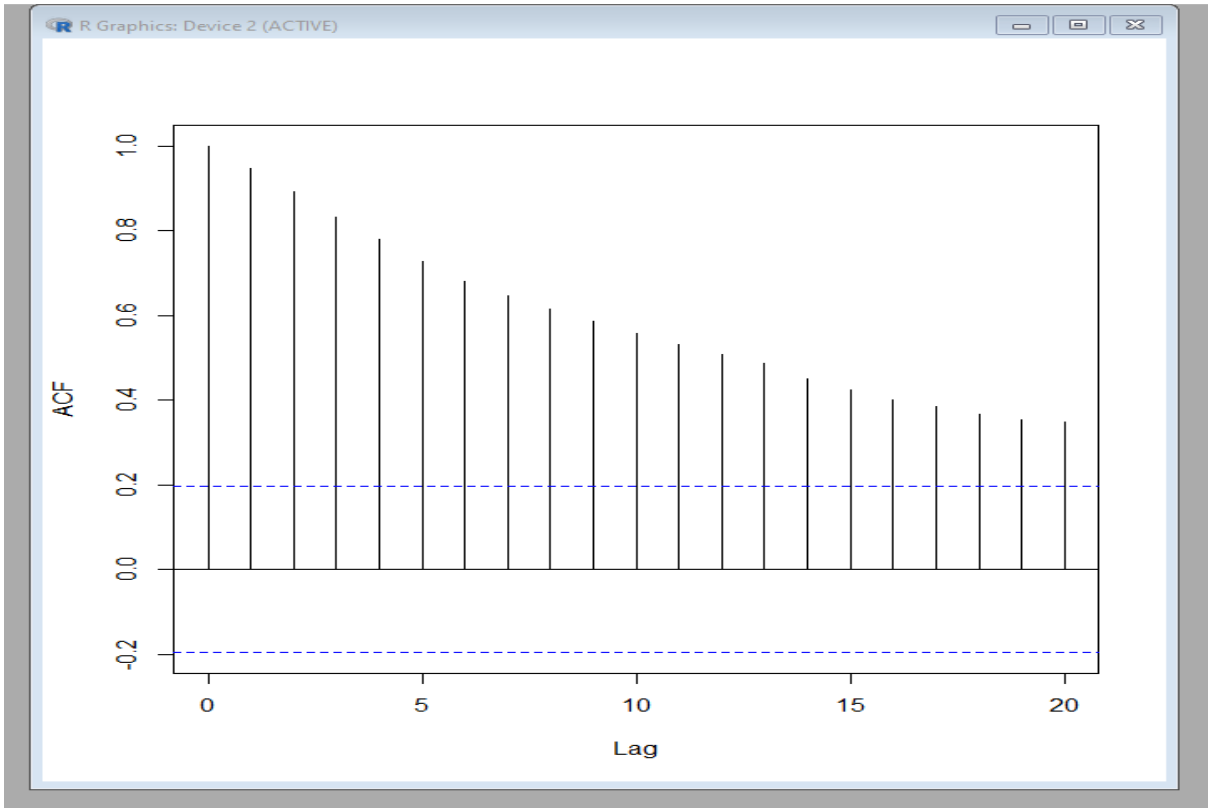
Output:



Code:

```
acf(x,main="")
```

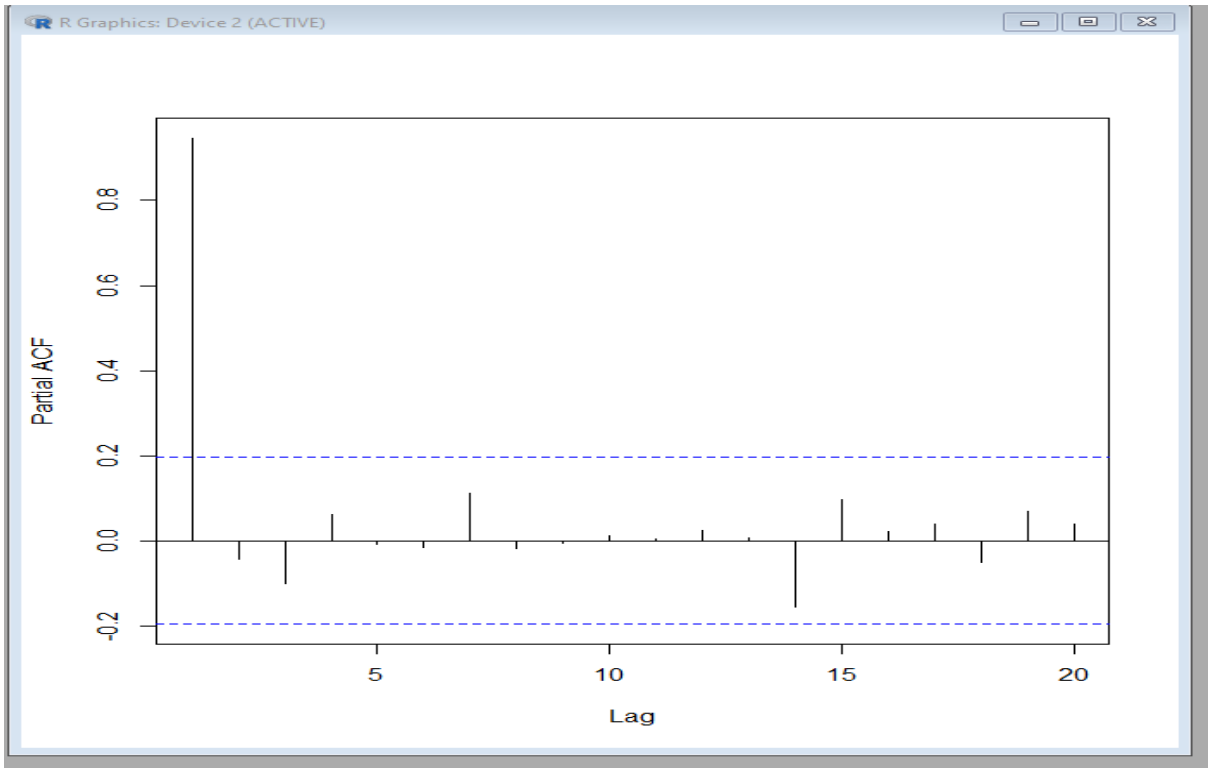
Output:



Code:

```
pacf(x,main="")
```

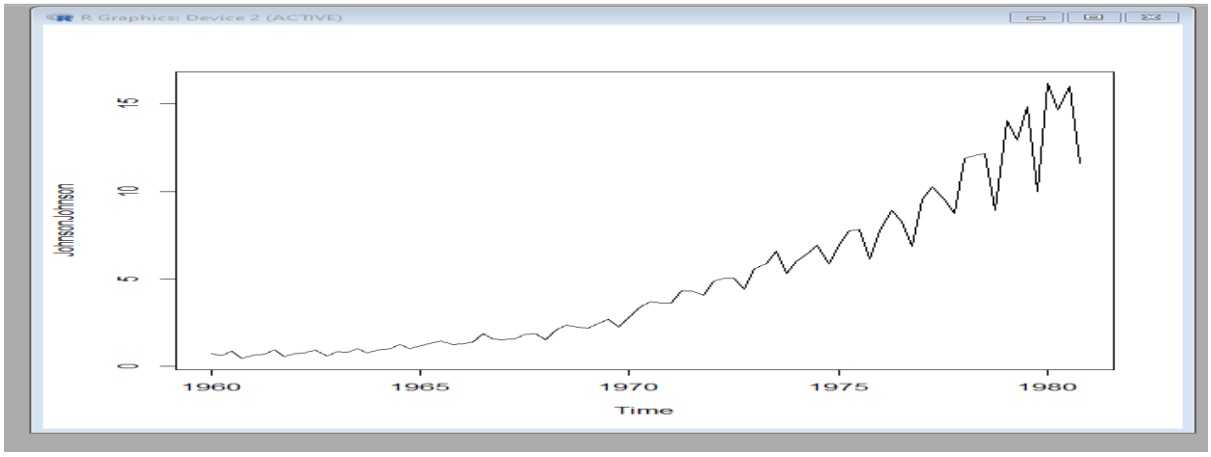
Output:



Code:

```
library(datasets)
plot(JohnsonJohnson)
```

Output:



PRACTICAL N0 – 2

Aim: Linear regression

Code:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

data_set = pd.read_csv('Salary_Data.csv')
print(data_set)
```

Output:

```
#prac3 Linear Regression
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

data_set = pd.read_csv('Salary_Data.csv')
print(data_set)
```

```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
```

Code:

```
x=data_set.iloc[:,:-1].values
y=data_set.iloc[:,1].values
print(y)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=1/3,random
_state=0)
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(x_train,y_train)
```

Output:

```
x=data_set.iloc[:,:-1].values
y=data_set.iloc[:,1].values
print(y)
```

```
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
  57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
  81363.  93940.  91738.  98273. 101302. 113812. 109431. 105582. 116969.
 112635. 122391. 121872.]
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=1/3,random_state=0)
```

```
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(x_train,y_train)
```

```
LinearRegression
LinearRegression()
```

Code:

```
_pred=regressor.predict(x_test)
x_pred=regressor.predict(x_train)
print(y_pred)
print(x_pred)
```

Output:

```
y_pred=regressor.predict(x_test)
x_pred=regressor.predict(x_train)
print(y_pred)
print(x_pred)
```

```
[ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]
[ 53919.42532909  74480.49870396  56723.20806202  68872.93323808
 103452.92027763  90368.60085726  38965.91742009 124948.58789682
  54854.0195734   47377.2656189   81957.25265845  82891.84690277
  61396.17928358  56723.20806202 110929.67423213  45508.07713028
  37096.72893147  93172.3835902   72611.31021533  64199.96201652]
```

Code:

```
mtp.scatter(x_train,y_train,color="green")
mtp.plot(x_train,x_pred,color="red")
mtp.title("Salary vs Experience(Training DataSet)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

Output:

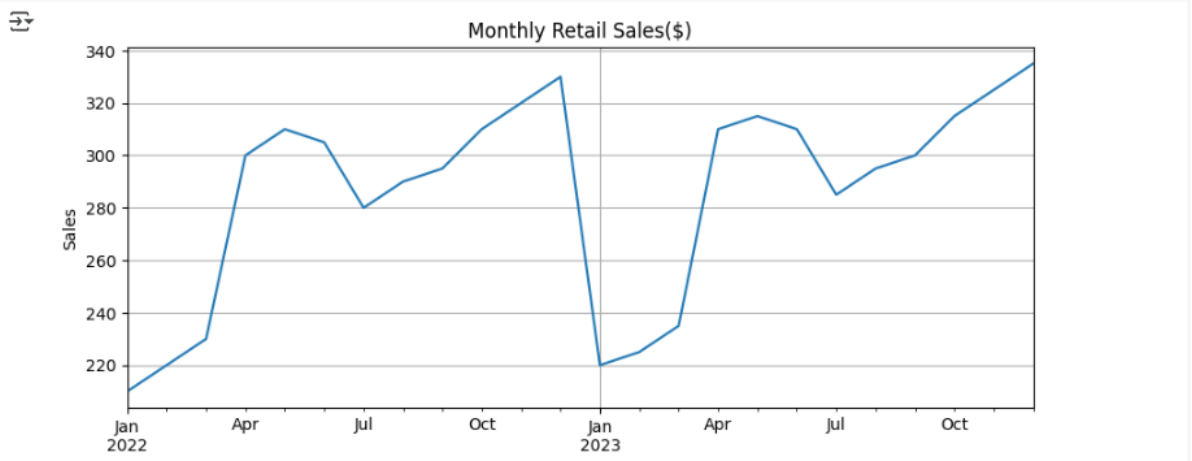
PRACTICAL N0 – 3

Aim: Exponential Smoothing using fit holt winter additive

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
#load or Define data
#load and prepare data
sales=[210,220,230,300,310,305,280,290,295,310,320,330,220,225,235,310,
315,310,285,295,300,315,325,335]
dates=pd.date_range(start='2022-01-01',periods=len(sales),freq='MS')
df = pd.Series(sales,index=dates)
df.plot(title='Monthly Retail Sales($)',figsize=(10,4))
plt.ylabel("Sales")
plt.grid()
plt.show()
```

Output:

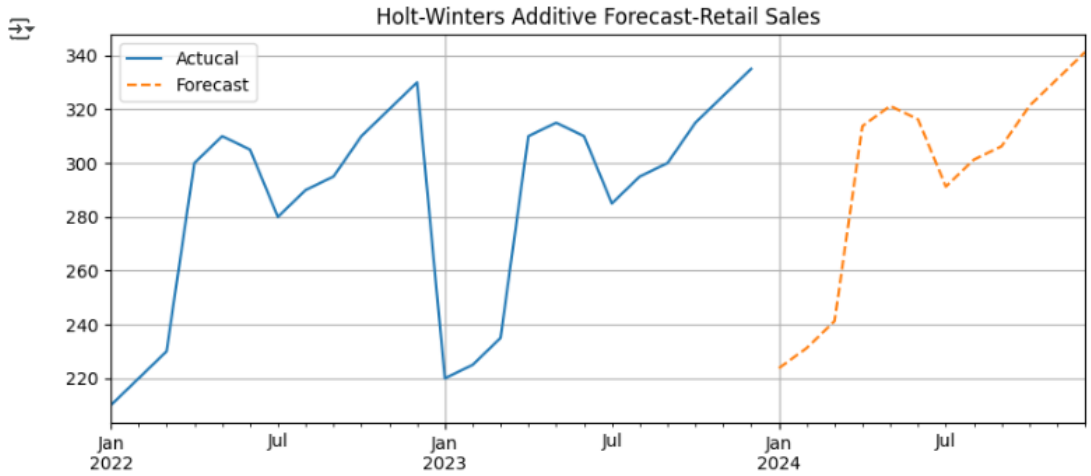


Code:

```
#Create models
#fit holt-winters Additive
model =
ExponentialSmoothing(df,trend='add',seasonal='add',seasonal_periods=12)
fit=model.fit()
#forecast next 12 month
forecast=fit.forecast(12)
#plot actucal & forecasted values
#plot actual + forecast
plt.figure(figsize=(10,4))
df.plot(label='Actucal')
forecast.plot(label='Forecast',linestyle='--')
plt.title("Holt-Winters Additive Forecast-Retail Sales")
```

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

Output:



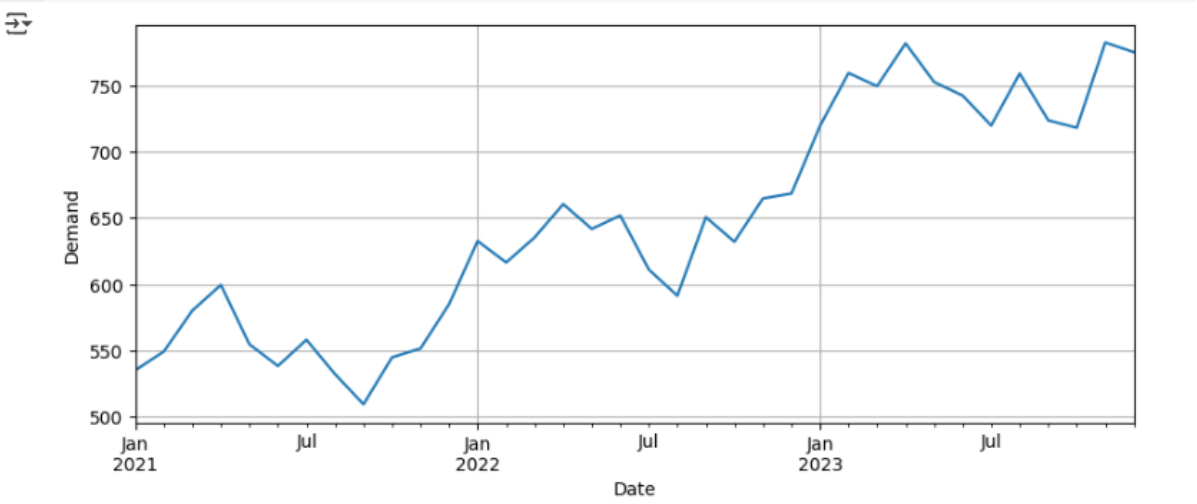
#using different method

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
#Simutaled Electronics demand data
#monthly demand over 3 year(36month)
np.random.seed(42)
months=pd.date_range(start='2021-01-01',periods=36,freq='MS')
#Trend+seasonality+noise
demand=(
    500+#base demand
    np.linspace(0,300,36)+ #upward trend
    50*np.sin(2*np.pi*months.month/12)+ #seasonality(1-yearcycle)
    np.random.normal(0,20,36) #random noise
  )
df=pd.Series(demand,index=months)
df.name='Monthly Electronics Demand'
plt.figure(figsize=(10,4))
df.plot()
df=pd.Series(demand,index=months)
df.name = ('Monthly Electronics Demand(units)')
plt.xlabel('Date')
plt.ylabel('Demand')
plt.grid()
plt.show()
```

Output:

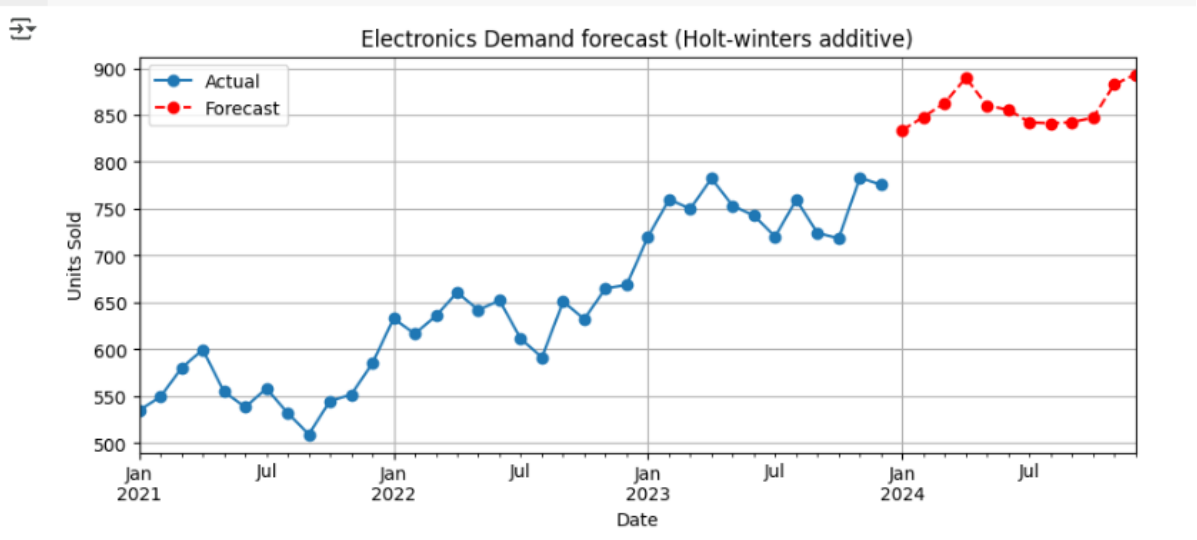
```
plt.show()
```



Code:

```
#apply holt-winter forecasting(additive)
model=ExponentialSmoothing(df,trend='add',seasonal='add',seasonal_periods=12)
fit=model.fit()
#forecast for 12 months
forecast=fit.forecast(12)
#plotting actucal and forecasting demand
plt.figure(figsize=(10,4))
df.plot(label='Actual',marker='o')
forecast.plot(label='Forecast',marker='o',linestyle='--',color='red')
plt.title('Electronics Demand forecast (Holt-winters additive)')
plt.xlabel('Date')
plt.ylabel("Units Sold")
plt.legend()
plt.grid()
plt.show()
```

Output:

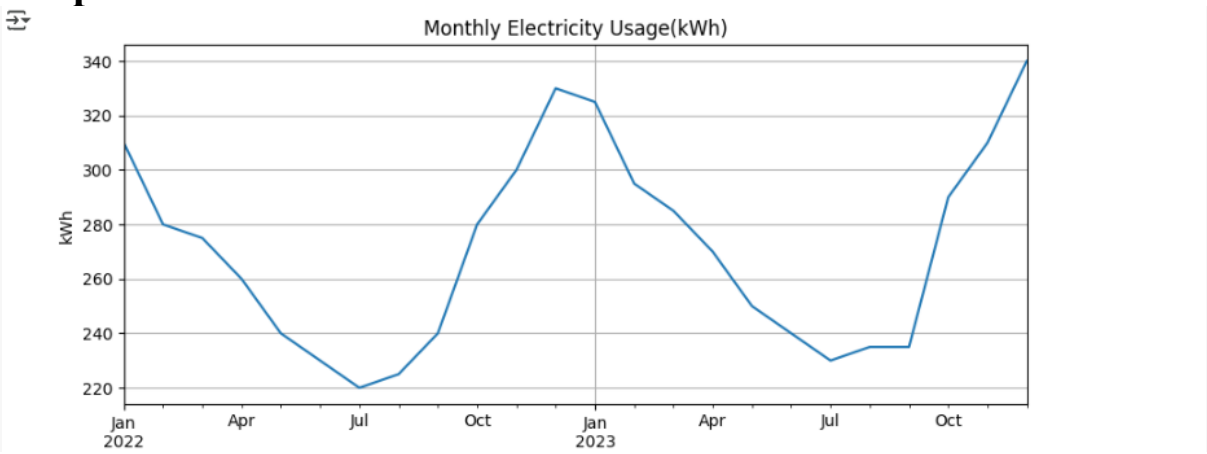


#using multiplicative

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
#energy consumption(multiplicative)
#predict monthly electricity usage
#synthetic but realistic kwh values for monthly electricity
energy=[310,280,275,260,240,230,220,225,240,280,300,330,325,295,285,270
,250,240,230,235,235,290,310,340]
dates=pd.date_range(start='2022-01-01',periods=len(energy),freq='MS')
df2=pd.Series(energy,index=dates)
df2.plot(title='Monthly Electricity Usage(kWh)',figsize=(10,4))
plt.ylabel('kWh')
plt.grid()
plt.show()
```

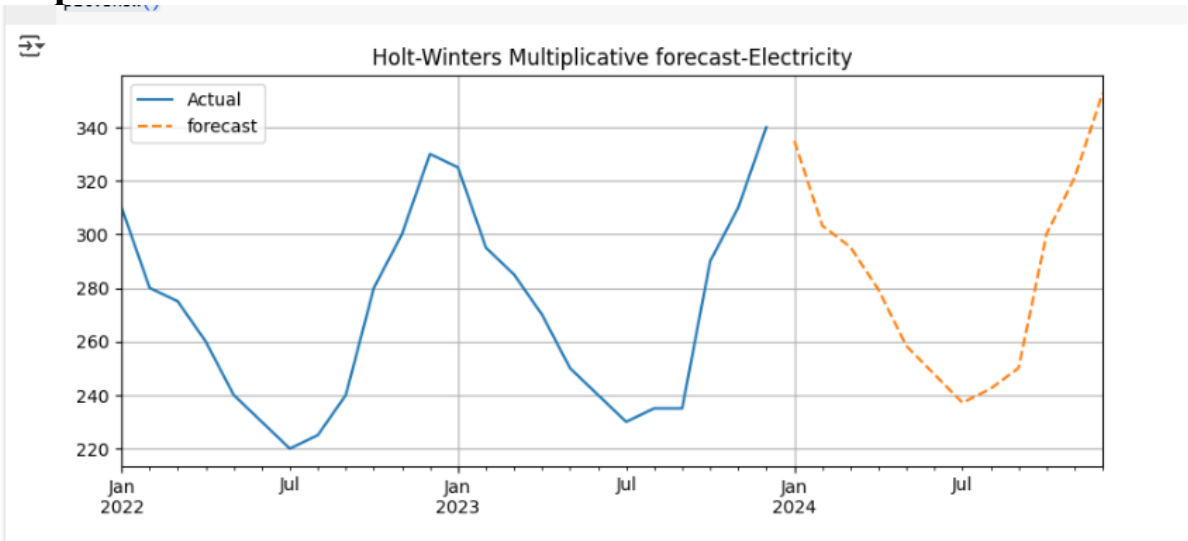
Output:



Code:

```
#Holt Winters Multiplicative
model2=ExponentialSmoothing(df2,trend='add',seasonal='mul',seasonal_per
iods=12)
fit2=model2.fit()
#forecast
forecast2=fit2.forecast(12)

#plot electricity usage
plt.figure(figsize=(10,4))
df2.plot(label='Actual')
forecast2.plot(label='forecast',linestyle='--')
plt.title("Holt-Winters Multiplicative forecast-Electricity")
plt.grid()
plt.legend()
plt.show()
```

Output:



#using different dataset

Code:

```
#prac 7
#holt-winter on kashmir Tourism data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
#monthly tourist data
np.random.seed(10)
months=pd.date_range(start='2019-01-01',periods=60,freq='MS')
#base+trend+seasonal variation(e.g high in may-july and dec-jan)
base=10000
trend=np.linspace(0,5000,60)
seasonal_pattern=np.sin(2*np.pi*months.month/12)*2000 #seasonal effect

noise=np.random.normal(0,800,60) #random variation
visitors = base+trend+seasonal_pattern+noise
df = pd.Series(visitors, index=months)
df.name='Monthly Visitors(Kashmir)'
#plot actual data
plt.figure(figsize=(10,4))
df.plot()
plt.title("Monthly Tourist Visitors-Kashmir Valley")
plt.xlabel("Date")
plt.ylabel('Visitors')
plt.grid()
plt.show()
```

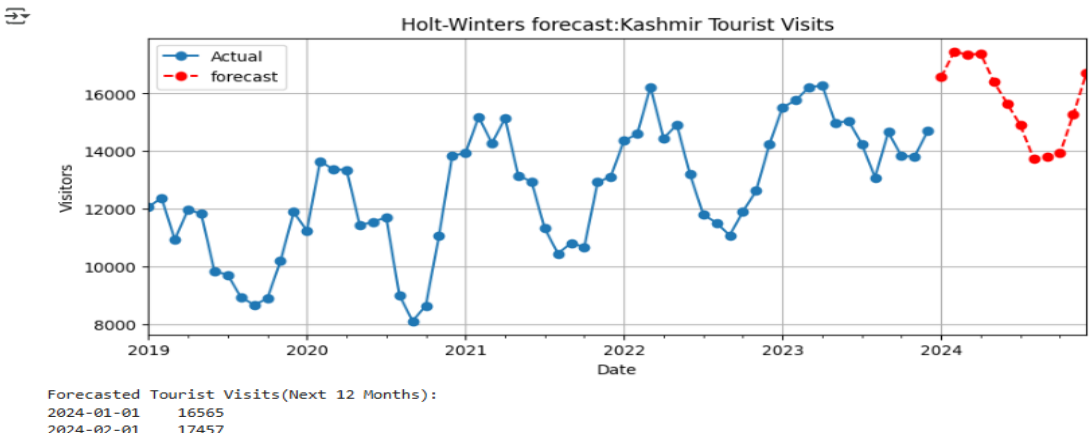
Output:


```
#apply holty-Winters(additive seasonality)
```

Code:

```
model=ExponentialSmoothing(df,trend='add',seasonal='add',seasonal_perio
ds=12)
#12 months per seasonal cycle
fit=model.fit()
#forecast next 12 months tourist
forecast=fit.forecast(12)
#plot actual vs forecast data
plt.figure(figsize=(10,4))
df.plot(label='Actual',marker='o')
forecast.plot(label='forecast',marker='o',linestyle='--',color='red')
plt.title("Holt-Winters forecast:Kashmir Tourist Visits")
plt.xlabel("Date")
plt.ylabel("Visitors")
plt.grid()
plt.legend()
plt.show()
print("Forecasted Tourist Visits(Next 12 Months):")
print(forecast.round(0).astype(int))
```

Output:



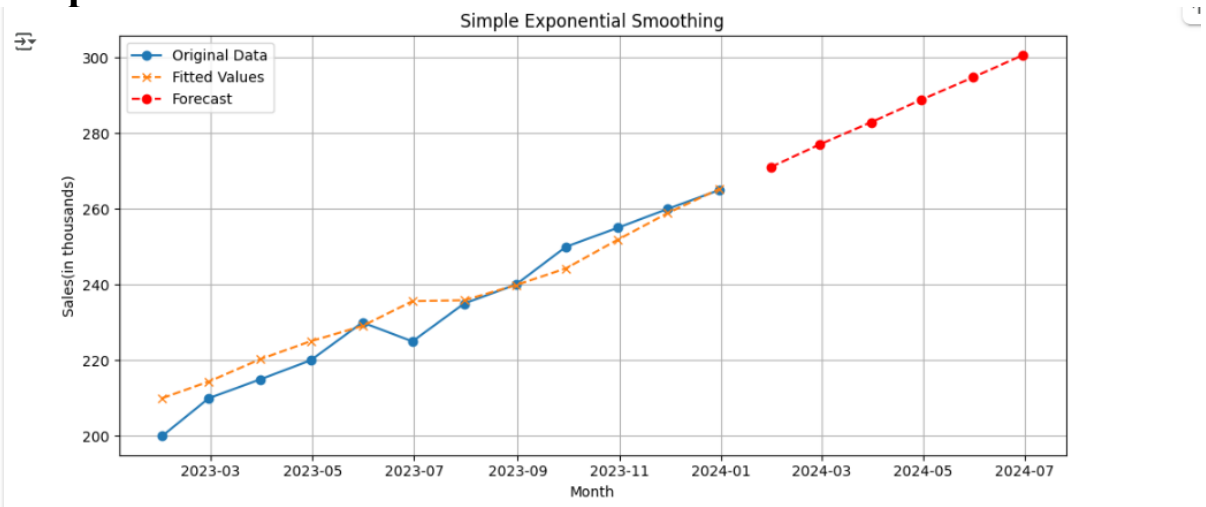
PRACTICAL N0 – 4

Aim: Simple Exponential Smoothing

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import SimpleExpSmoothing,Holt
#create or load dataset
#monthly sales data(in thousand)
sales_data=[200,210,215,220,230,225,235,240,250,255,260,265]

#create a time index
dates=pd.date_range(start='2023-01-
01',periods=len(sales_data),freq='M')

#create a time series object
sales_series=pd.Series(sales_data,index=dates)
#apply ses
#create the ses model
model=Holt(sales_series)
#fit the model alpha=0.4,0.3,0.2
fit=model.fit(smoothing_level=0.4,optimized='False')

# forecast for next 6 month
forecast=fit.forecast(6)
#visualize result
#plotting the original data,fitted values and forecast
plt.figure(figsize=(10,5))
plt.plot(sales_series,label='Original Data',marker='o')
plt.plot(fit.fittedvalues,label='Fitted Values',linestyle='--
',marker='x')
plt.plot(forecast,label='Forecast',linestyle='--
',marker='o',color='red')

plt.title('Simple Exponential Smoothing')
plt.xlabel('Month')
plt.ylabel('Sales(in thousands)')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

Output:



Code:

```
#print forecast values
print("Forecasted sales(next 6 month)")
print(forecast.round(2))
```

Output:

```
#print forecast values
print("Forecasted sales(next 6 month)")
print(forecast.round(2))
```

```
Forecasted sales(next 6 month)
2024-01-31    271.10
2024-02-29    277.01
2024-03-31    282.92
2024-04-30    288.84
2024-05-31    294.75
2024-06-30    300.67
Freq: ME, dtype: float64
```

#using csv file

Code:

```
#program
#csv files
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
#load dataset
df=pd.read_csv('electricity_usage.csv',parse_dates=['Date'],index_col='
Date')
print(df.head())
#checking optional values
print(df.isnull().sum())
#create time series object
series =df['Consumption']
#apply ses
model=Holt(series)
```

```
fit=model.fit(smoothing_level=0.4,optimized='False')
#forecast next 6 month
forecast=fit.forecast(6)

#print forecast values
print("Next 6 month forecast:")
print(forecast)
```

Output:

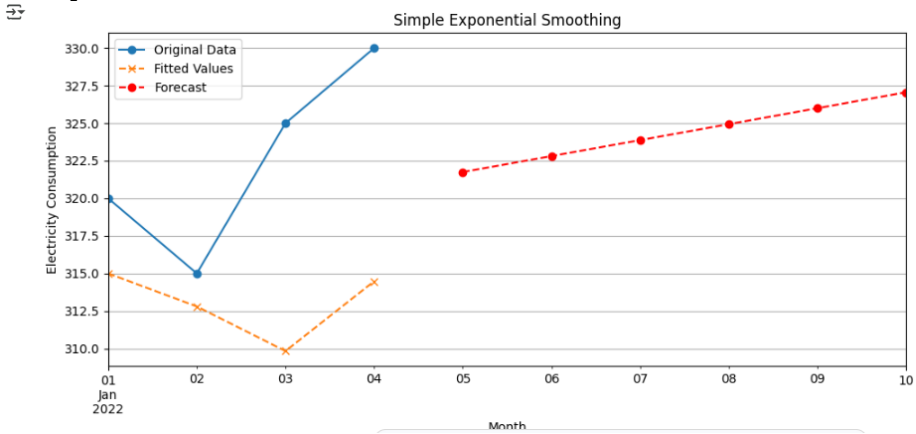
```
#forecast next 6 month
forecast=fit.forecast(6)

#print forecast values
print("Next 6 month forecast:")
print(forecast)
```

```
Next 6 month forecast:
2022-01-05    321.749235
2022-01-06    322.811622
2022-01-07    323.874010
2022-01-08    324.936397
2022-01-09    325.998784
2022-01-10    327.061171
Freq: D, dtype: float64
```

Code:

```
#plot the result
plt.figure(figsize=(10,5))
series.plot(label='Original Data',marker='o')
fit.fittedvalues.plot(label='Fitted Values',linestyle='--',marker='x')
forecast.plot(label='Forecast',linestyle='--',marker='o',color='red')

plt.title('Simple Exponential Smoothing')
plt.xlabel('Month')
plt.ylabel('Electricity Consumption')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

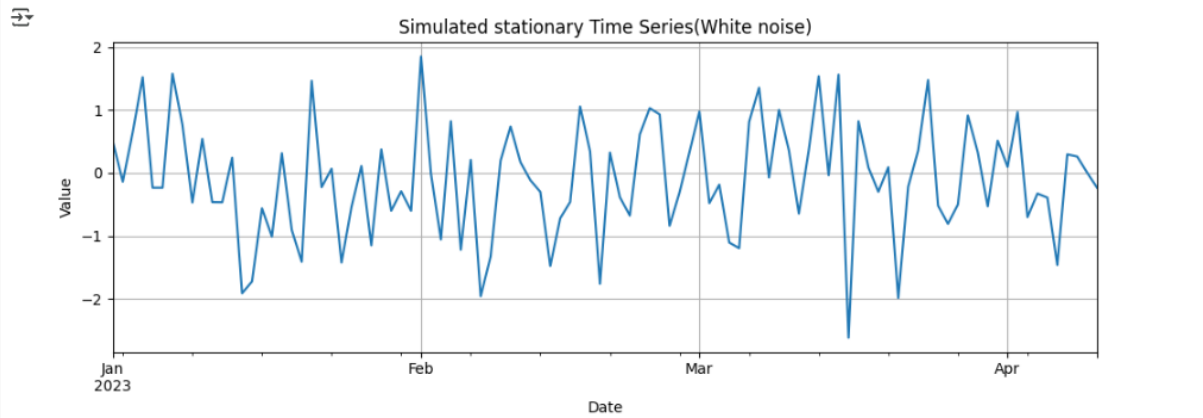
Output:



Code:

```
#staionary time series example
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
#set seed for reproducibility
np.random.seed(42)
#build model
data = np.random.normal(loc=0,scale=1,size=100)
dates=pd.date_range(start='2023-01-01',periods=100,freq='D')
series=pd.Series(data,index=dates)
#plot stationary data series
plt.figure(figsize=(10,4))
series.plot(title='Simulated stationary Time Series(White noise)')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid()
plt.tight_layout()
plt.show()
```

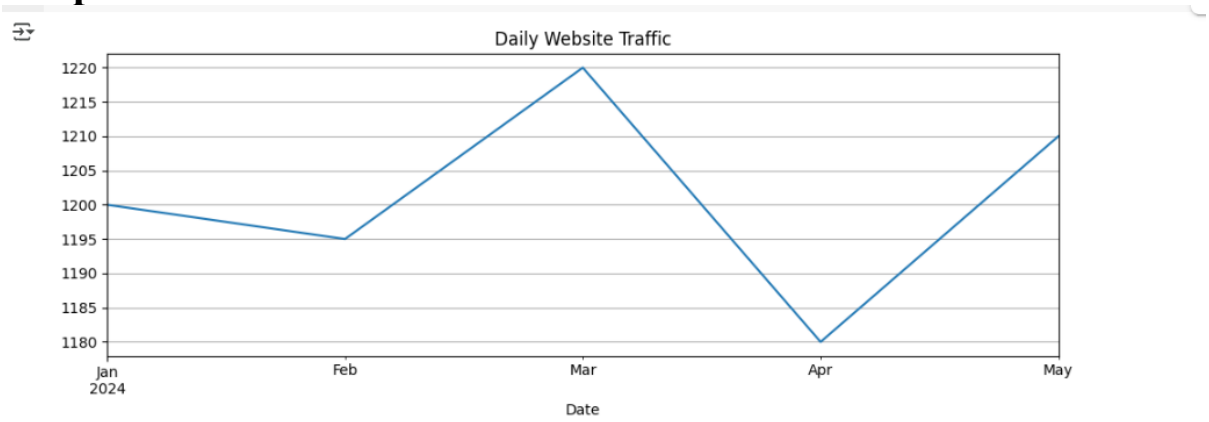
Output:



Code:

```
#check whether stationary or non stationary
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
df=pd.read_csv('website_traffic.csv',parse_dates=['Date'],index_col='Da
te')
df['Visits'].plot(figsize=(10,4),title="Daily Website Traffic")
plt.grid()
plt.tight_layout()
plt.show()
```

Output:



Code:

```
#perform augmented Dickey-Fuller test
result=adfuller(df['Visits'])
print("Augmented Dickey-Fuller Test:",result[0])
print("p-value:",result[1])
```

Output:

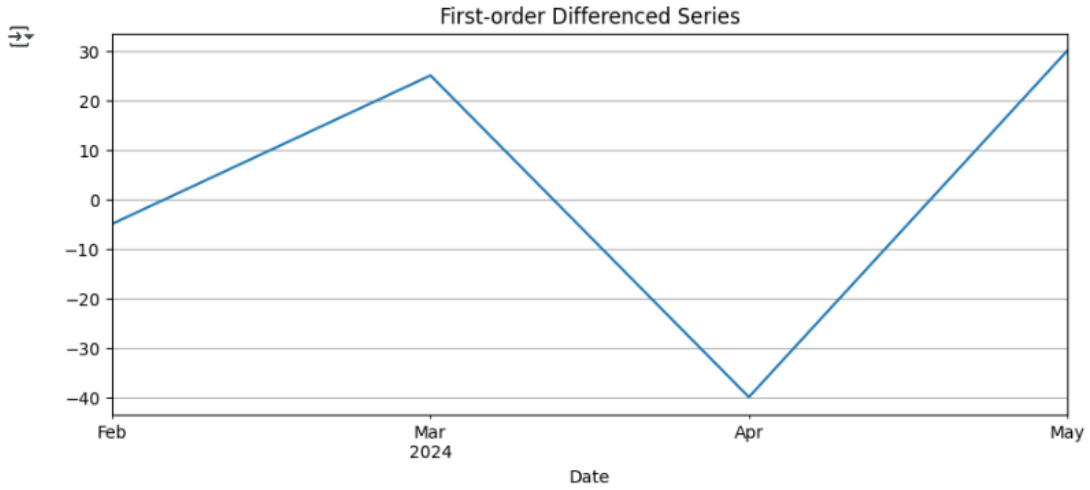
```
#perform augmented Dickey-Fuller test
result=adfuller(df['Visits'])
print("Augmented Dickey-Fuller Test:",result[0])
print("p-value:",result[1])
```

```
Augmented Dickey-Fuller Test: -4.109375445408512
p-value: 0.0009345359045134847
```

Code:

```
if result[1]<0.05:
  print("The series is stationary")
else:
  print("The series is non-stationary.Applying differencing...")
#first order differencing
df_diff=df['Visits'].diff().dropna()
#plot differenced series
df_diff.plot(figsize=(10,4),title="First-order Differenced Series")
plt.grid()
plt.show()
```

Output:



Code:

```
#re-run adf test on differenced series
result_diff= adfuller(df_diff)
print("Augmented Dickey-Fuller Test:",result_diff[0])
print("p-value after decreasing:",result_diff[1])
if result_diff[1]<0.05:
  print("The differenced series is now stationary")
else:
  print("The differenced series is now non-stationary")
```

Output:

```
#re-run adf test on differenced series
result_diff= adfuller(df_diff)
print("Augmented Dickey-Fuller Test:",result_diff[0])
print("p-value after decreasing:",result_diff[1])
```

```
Augmented Dickey-Fuller Test: -3.5388175323270006
p-value after decreasing: 0.007044379192708437
```

```
[ ] if result_diff[1]<0.05:
      print("The differenced series is now stationary")
    else:
      print("The differenced series is now non-stationary")
```

```
The differenced series is now stationary
```

PRACTICAL N0 – 5

Aim: Simple Exponential Smoothing holt(in rstudio) & Exponential and moving average smoothing-Holts forecasting

Code:

```
install.packages("forecast")
```

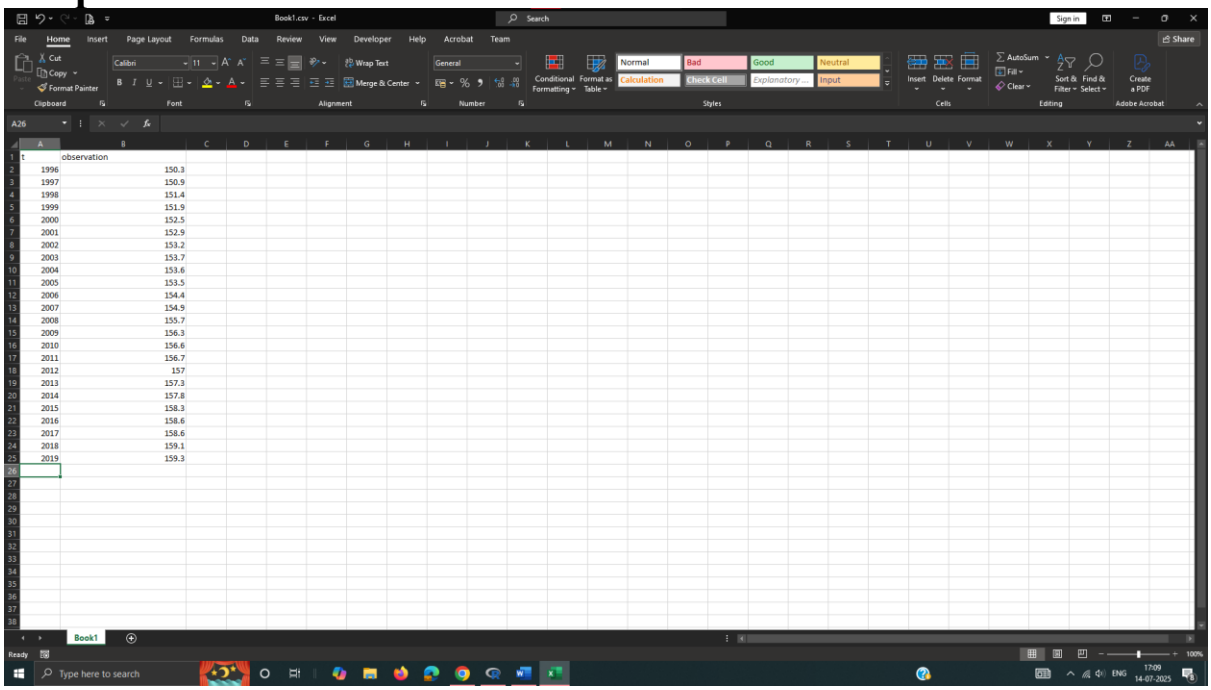
Output:

```
> install.packages("forecast")
Warning in install.packages("forecast") :
  'lib = "C:/Program Files/R/R-4.5.1/library"' is not writable
--- Please select a CRAN mirror for use in this session ---
also installing the dependencies 'utf8', 'farver', 'labeling', 'R6', 'RColorBrewer', 'viridisLite', 'pillar', 'pkgconfig', 'xts', 'TTR', 'curl',

trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/utf8_1.2.6.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/farver_2.1.2.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/labeling_0.4.3.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/R6_2.6.1.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/RColorBrewer_1.1-3.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/viridisLite_0.4.2.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/pillar_1.11.0.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/pkgconfig_2.0.3.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/xts_0.14.1.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/TTR_0.24.4.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/curl_6.4.0.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/cli_3.6.5.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/glue_1.8.0.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/gtable_0.3.6.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/isoband_0.2.7.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/lifecycle_1.0.4.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/rlang_1.1.6.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/scales_1.4.0.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/tibble_3.3.0.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/vctrs_0.6.5.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/quadprog_1.5-8.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/quantmod_0.4.28.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/jsonlite_2.0.0.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/colorspace_2.1-1.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/fracdiff_1.5-3.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/generics_0.1.4.zip'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/ggplot2_3.5.2.zip'
```

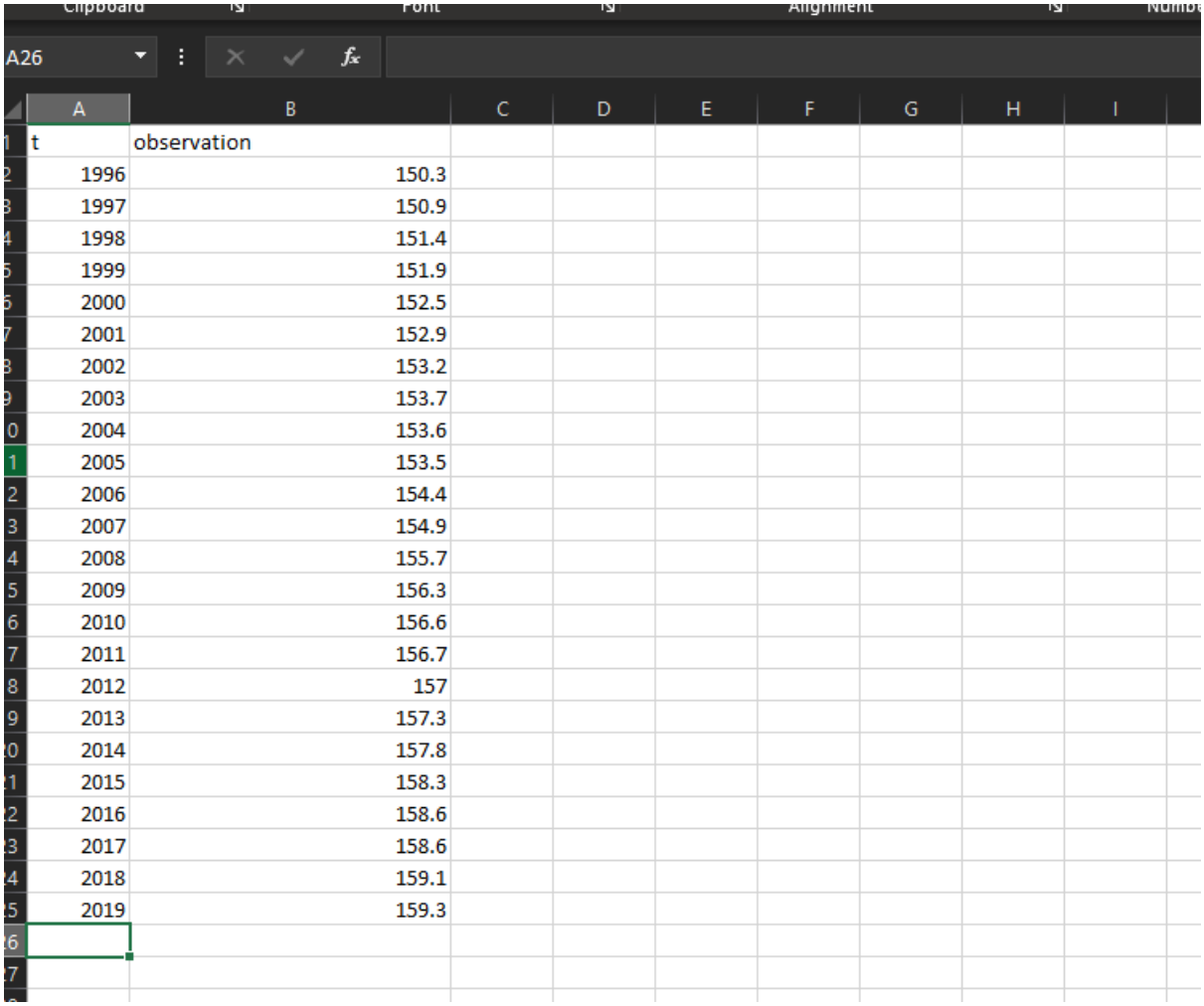
Code:

```
library(forecast)
```

Output:



t	observation
1996	150.3
1997	150.9
1998	151.4
1999	151.9
2000	152.5
2001	152.9
2002	153.2
2003	153.7
2004	153.6
2005	153.5
2006	154.4
2007	154.9
2008	155.7
2009	156.3
2010	156.6
2011	156.7
2012	157
2013	157.3
2014	157.8
2015	158.3
2016	158.6
2017	158.6
2018	159.1
2019	159.3



	t	observation
2	1996	150.3
3	1997	150.9
4	1998	151.4
5	1999	151.9
6	2000	152.5
7	2001	152.9
8	2002	153.2
9	2003	153.7
10	2004	153.6
11	2005	153.5
12	2006	154.4
13	2007	154.9
14	2008	155.7
15	2009	156.3
16	2010	156.6
17	2011	156.7
18	2012	157
19	2013	157.3
20	2014	157.8
21	2015	158.3
22	2016	158.6
23	2017	158.6
24	2018	159.1
25	2019	159.3

Code:

```
data<-read.csv('C:/Users/Admin/Desktop/Book1.csv')
data
```

Output:

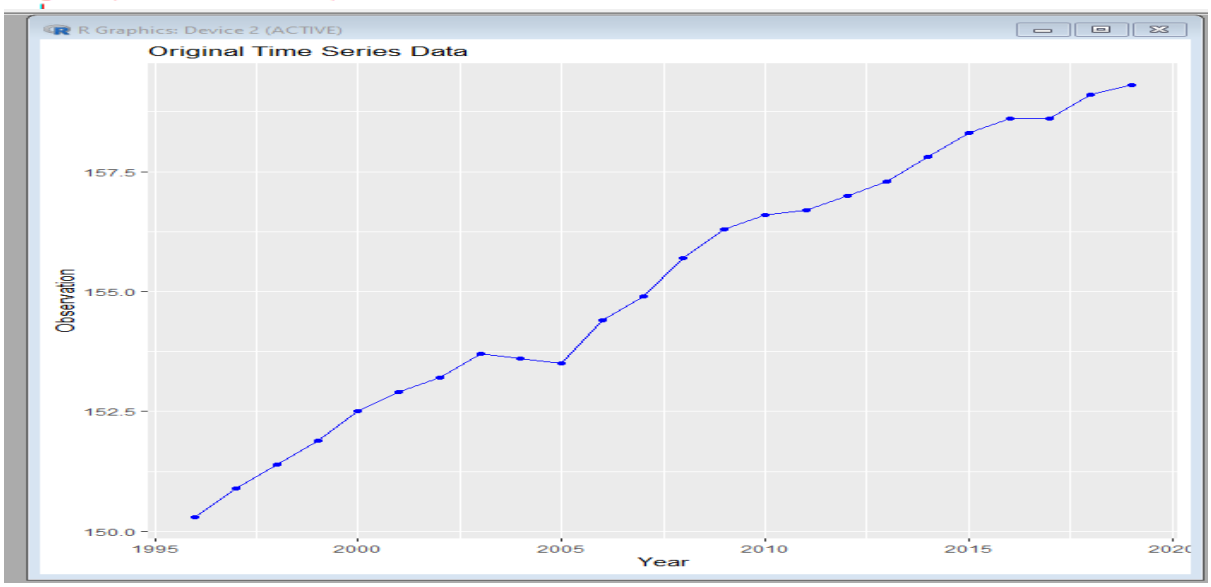
```
> data<-read.csv('C:/Users/Admin/Desktop/Book1.csv')
> data
    t observation
1  1996        150.3
2  1997        150.9
3  1998        151.4
4  1999        151.9
5  2000        152.5
6  2001        152.9
7  2002        153.2
8  2003        153.7
9  2004        153.6
10 2005        153.5
11 2006        154.4
12 2007        154.9
13 2008        155.7
14 2009        156.3
15 2010        156.6
16 2011        156.7
17 2012        157.0
18 2013        157.3
19 2014        157.8
20 2015        158.3
21 2016        158.6
22 2017        158.6
23 2018        159.1
24 2019        159.3
```

Code:

```
library(ggplot2)
ggplot(data,aes(x=t,y=observation))+
 geom_line(color="blue")+
geom_point(color="blue")+
 ggtitle("Original Time Series Data")+
 xlab("Year")+
 ylab("Observation")
```

Output:

```
> ggplot(data,aes(x=t,y=observation))+
+ geom_line(color="blue")+
+ geom_point(color="blue")+
+ ggtitle("Original Time Series Data")+
+ xlab("Year")+
+ ylab("Observation")
```



Code:

```
ses_model<-ses(data$observation, alpha=0.3,initial="simple",h=1)
ses_fitted<-fitted(ses_model)
 ses_fitted
```

Output:

```
> ses_model<-ses(data$observation, alpha=0.3,initial="simple",h=1)
> ses_fitted<-fitted(ses_model)
> ses_fitted
Time Series:
Start = 1
End = 24
Frequency = 1
 [1] 150.3000 150.3000 150.4800 150.7560 151.0992 151.5194 151.9336 152.3135
 [9] 152.7295 152.9906 153.1434 153.5204 153.9343 154.4640 155.0148 155.4904
[17] 155.8533 156.1973 156.5281 156.9097 157.3268 157.7087 157.9761 158.3133
```

Code:

```
holt_model<-
holt(data$observation,aplha=0.3,beta=0.2,initial="simple",h=1)
holt_fitted<-fitted(holt_model)
 holt_fitted
```

Output:

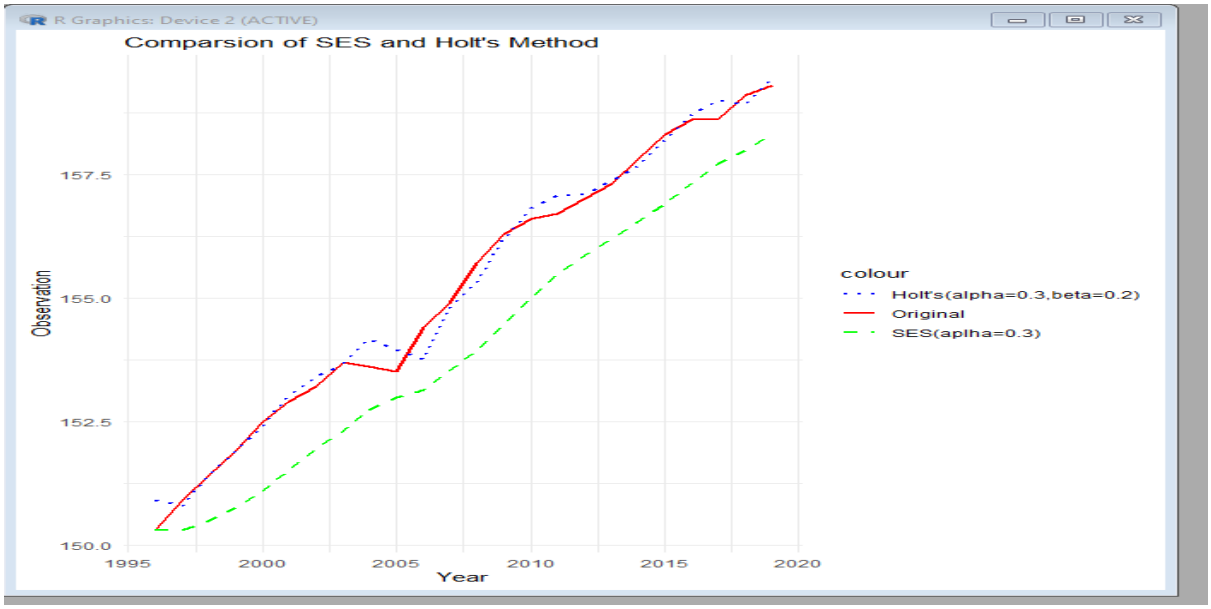
```
> holt_model<-holt(data$observation,aplha=0.3,beta=0.2,initial="simple",h=1)
> holt_fitted<-fitted(holt_model)
> holt_fitted
Time Series:
Start = 1
End = 24
Frequency = 1
 [1] 150.9000 150.7800 151.4040 151.9032 152.4026 153.0220 153.3976 153.6581
 [9] 154.1665 153.9532 153.7626 154.7900 155.3120 156.1896 156.8117 157.0694
[17] 157.0955 157.3764 157.6611 158.1889 158.7111 158.9889 158.9111 159.4489
```

Code:

```
ggplot()+
 geom_line(aes(x=data$t,
y=data$observation,color="Original"),size=1)+
geom_line(aes(x=data$t, y=ses_fitted,
color="SES(aplha=0.3)"),linetype="dashed",size=1)+
 geom_line(aes(x=data$t,
y=holt_fitted,color="Holt's(alpha=0.3,beta=0.2)"),linetype="dotted",
size=1)+
 ggtitle("Comparsion of SES and Holt's Method")+
 xlab("Year")+
 ylab("Observation")+
 scale_color_manual(values=c("blue","red","green"))+
 theme_minimal()
```

Output:

```
> ggplot()+
+  geom_line(aes(x=data$t, y=data$observation,color="Original"),size=1)+
+ geom_line(aes(x=data$t, y=ses_fitted, color="SES(aplha=0.3)"),linetype="dashe$
+ geom_line(aes(x=data$t, y=holt_fitted,color="Holt's(alpha=0.3,beta=0.2)"),lin$
+ ggtitle("Comparsion of SES and Holt's Method")+
+ xlab("Year")+
+ ylab("Observation")+
+ scale_color_manual(values=c("blue","red","green"))+
+ theme_minimal()
```



#Different dataset

Code:

```
years<-1991:2003
 co2_concentration<-
c(355.62,356.36,357.1,358.86,360.9,362.58,363.84,366.58,368.3,369
.47,371.03,373.61,357.61)
 ts_co2<-ts(co2_concentration,start=min(years),frequency=1)
 ts_co2
```

Output:

```
> years<-1991:2003
> co2_concentration<-c(355.62,356.36,357.1,358.86,360.9,362.58,363.84,366.58,368.3,369.47,371.03,373.61,357.6
> ts_co2<-ts(co2_concentration,start=min(years),frequency=1)
> ts_co2
Time Series:
Start = 1991
End = 2003
Frequency = 1
 [1] 355.62 356.36 357.10 358.86 360.90 362.58 363.84 366.58 368.30 369.47
[11] 371.03 373.61 357.61
```

Code:

```
moving_avg_manual<-function(data,window_size){
n<-length(data)
 ma<-rep(NA,n)
 for(i in seq(window_size,n)){
```

```
ma[i]<-mean(data[(i-window_size+1):i])
}
return(ma)
}
window_size<-3
ma_co2<-moving_avg_manual(co2_concentration,window_size)
ma_co2
```

Output:

```
> moving_avg_manual<-function(data,window_size){
+ n<-length(data)
+ ma<-rep(NA,n)
+ for(i in seq(window_size,n)){
+ ma[i]<-mean(data[(i-window_size+1):i])
+ }
+ return(ma)
+ }
> window_size<-3
> ma_co2<-moving_avg_manual(co2_concentration,window_size)
> ma_co2
 [1]       NA       NA 356.3600 357.4400 358.9533 360.7800 362.4400 364.3333
 [9] 366.2400 368.1167 369.6000 371.3700 367.4167
>
```

Code:

```
forecast_2004<-tail(ma_co2,1)
cat("Forecasted co2 concentration for 2004 using 3-year Moving Average:",forecast_2004,"\n")
```

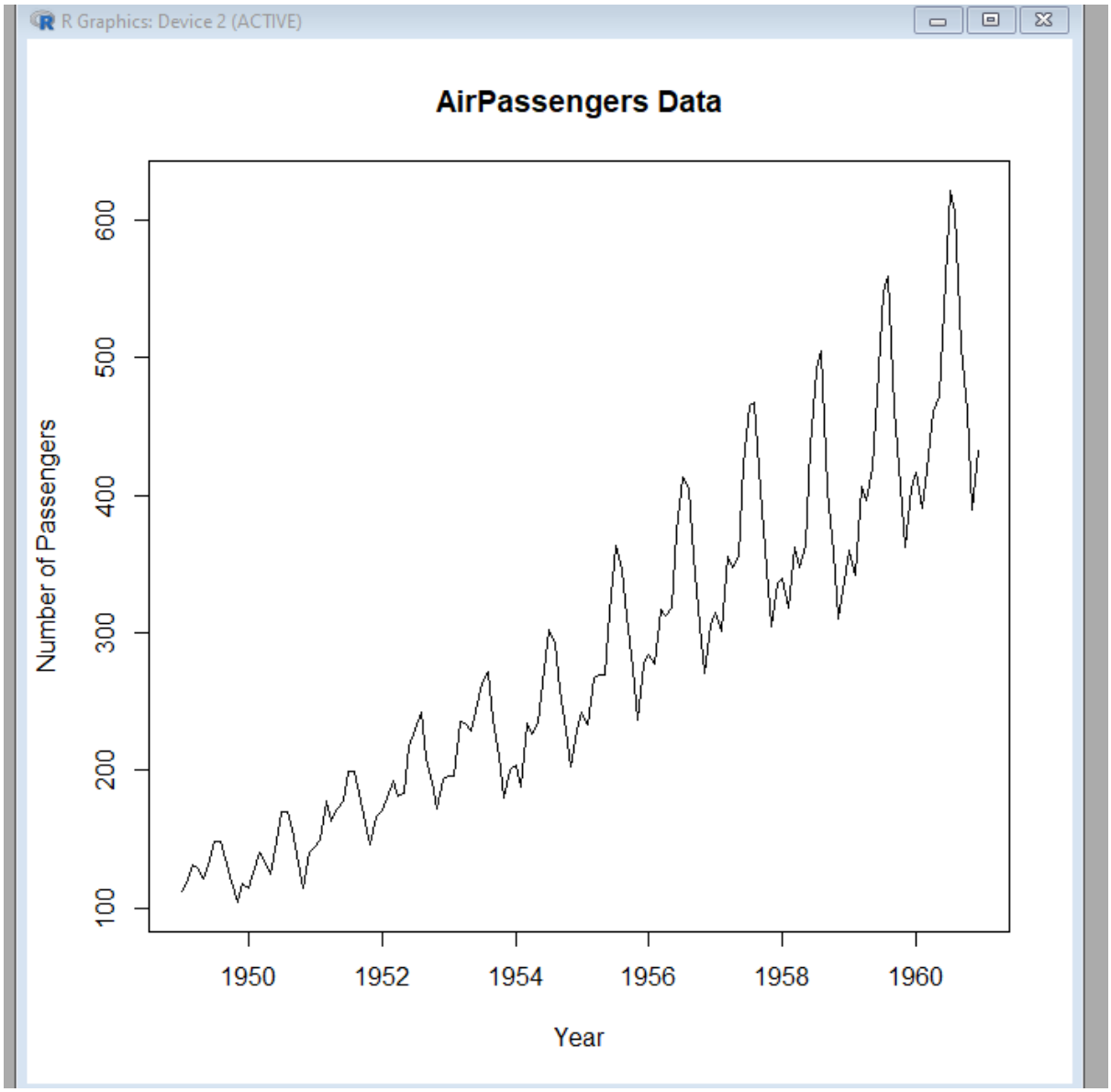
Output:

```
> forecast_2004<-tail(ma_co2,1)
> cat("Forecasted co2 concentration for 2004 using 3-year Moving Average:",forecast_2004,"\n")
Forecasted co2 concentration for 2004 using 3-year Moving Average: 367.4167
```

Code:

```
insall.packages("forecast")

library(forecast)

 data("AirPassengers")

plot(AirPassengers,main="AirPassengers Data",ylab="Number of Passengers",xlab="Year")
```

Output:



#Exponential and moving average smoothing-Holts forecasting

Code:

```
data<-
c(150.3,150.9,151.4,151.9,152.5,152.9,153.2,153.7,153.6,153.5,154.4,154.9,155.7,156.3,156.6,156.7,157,157.3,157.8,158.3,158.6,158.6,159.1,159.3)

time_series_data<-ts(data,start=1996,frequency=1)

#A)simple exponential smoothing aplha=0.3

#load necessary package

library(forecast)
```

Output:

```
> #Exponential and moving average smoothing-Holts forecasting
> data<-c(150.3,150.9,151.4,151.9,152.5,152.9,153.2,153.7,153.6,153.5,154.4,154.9,155.7,156.3,156.6,156.7,157,157.3,157.8,158.3,158.6,158.6,159.1,159.3)
> time_series_data<-ts(data,start=1996,frequency=1)
> #A)simple exponential smoothing aplha=0.3
> #load necessary package
> library(forecast)
Registered S3 method overwritten by 'quantmod':
  method           from
  as.zoo.data.frame zoo
```

Code:

```
ses_model<-ses(time_series_data,aplha=0.3,h=1)

summary(ses_model)
```

Output:

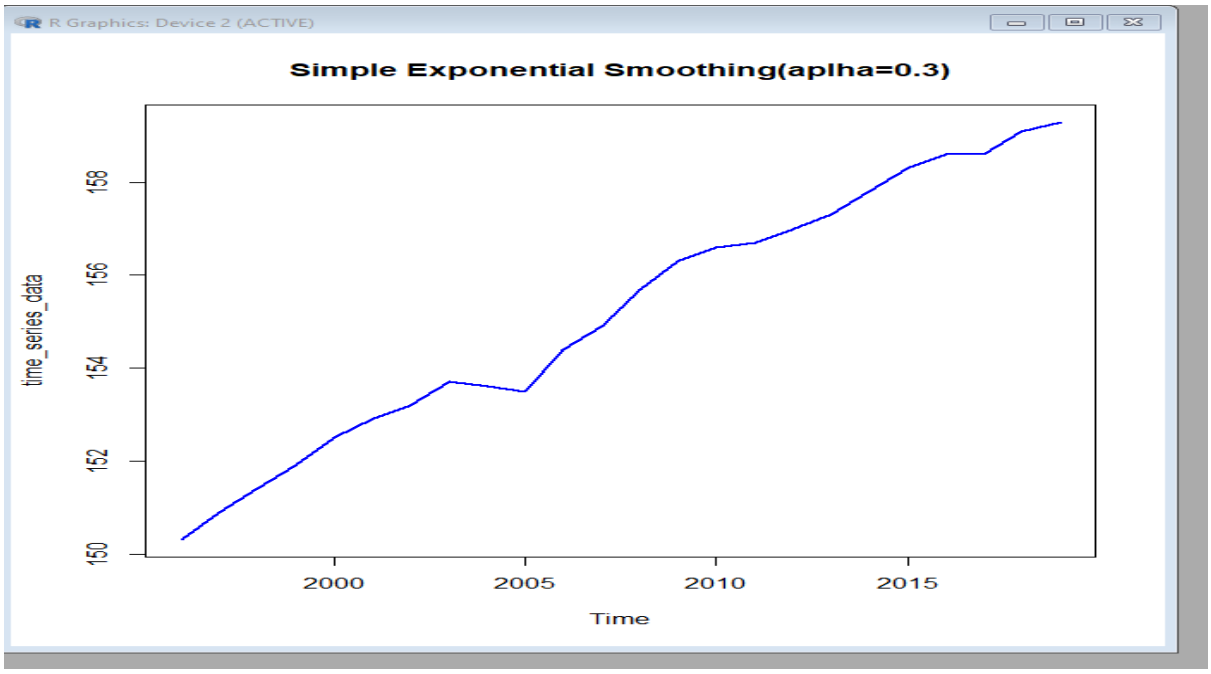
```
> ses_model<-ses(time_series_data,aplha=0.3,h=1)
> summary(ses_model)

Forecast method: Simple exponential smoothing

Model Information:
Simple exponential smoothing

Call:
ses(y = time_series_data, h = 1, aplha = 0.3)

  Smoothing parameters:
    alpha = 0.9999

  Initial states:
    l = 150.3001

  sigma:  0.4749

     AIC     AICc      BIC
44.43717 45.63717 47.97134

Error measures:
                    ME      RMSE       MAE       MPE      MAPE      MASE
Training set 0.3750322 0.4546386 0.3917047 0.2419044 0.2527626 0.9584264
                   ACF1
Training set 0.0627605

Forecasts:
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2020          159.3 158.6914 159.9085 158.3693 160.2307
```

Code:

#plot the ses model

plot(time_series_data,main="Simple Exponential Smoothing(aplha=0.3)",col="blue",lwd=2)

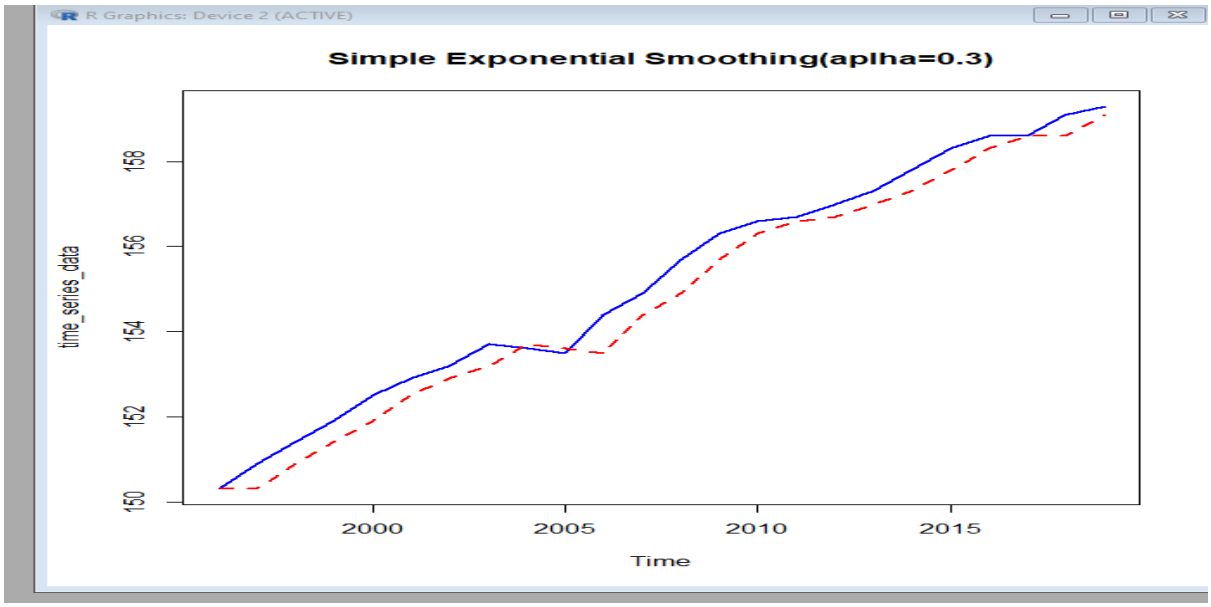
Output:



Code:

lines(fitted(ses_model),col="red",lty=2,lwd=2)

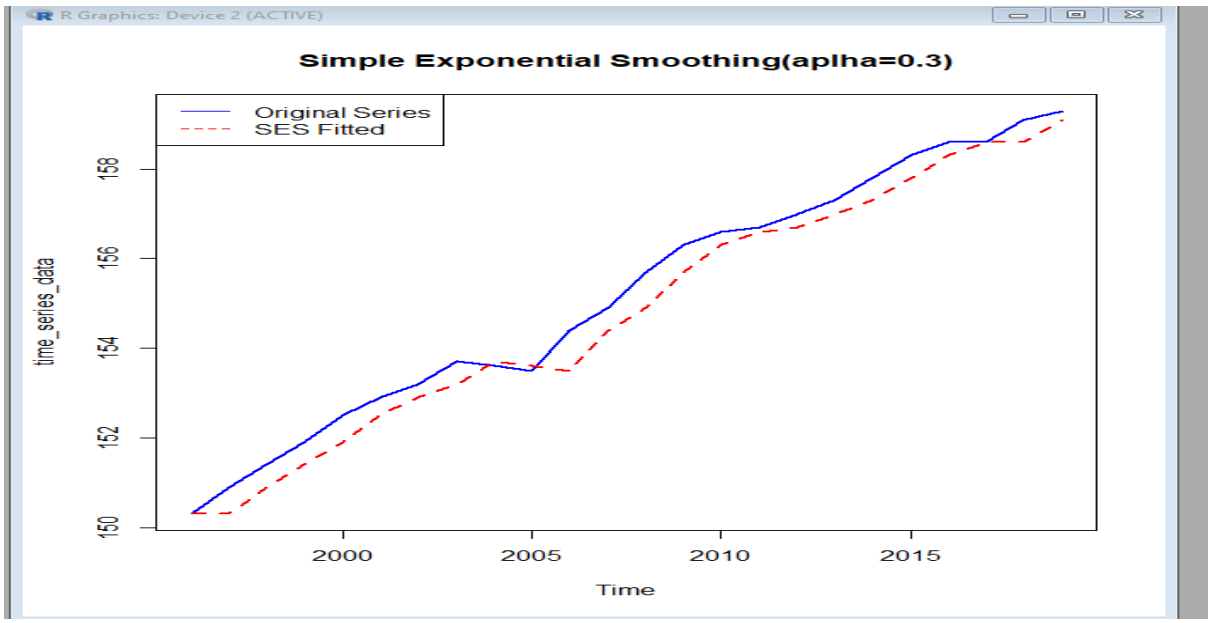
Output:



Code:

```
legend("topleft",legend=c("Original Series","SES
Fitted"),col=c("blue","red"),lty=1:2)
```

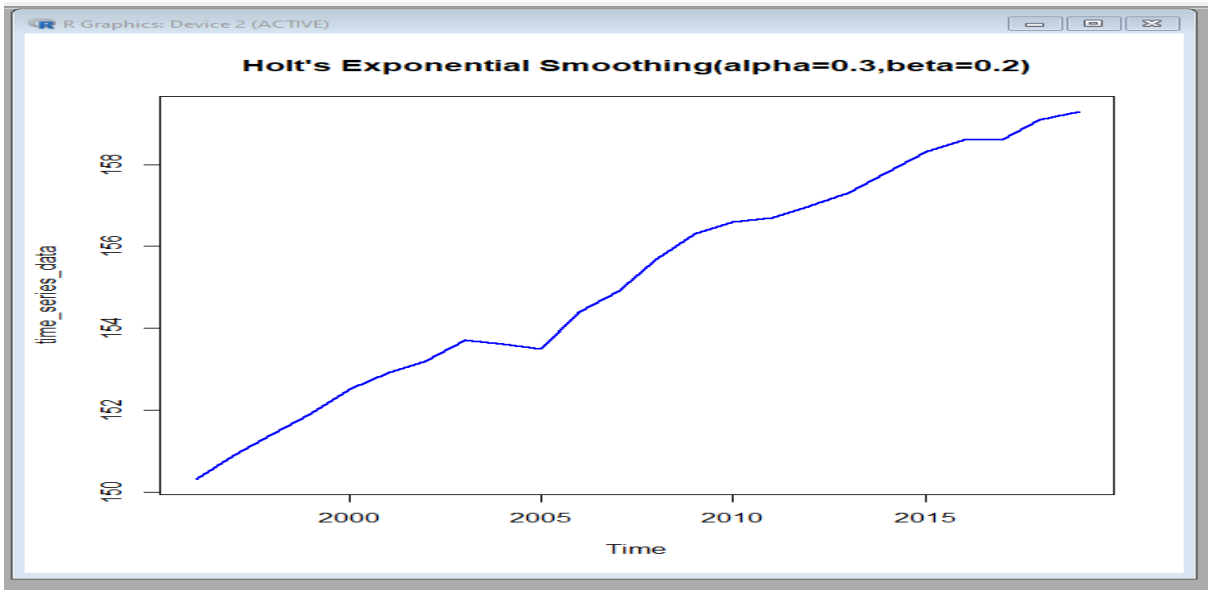
Output:



Code:

```
#plot the holt's model

 plot(time_series_data,main="Holt's Exponential
Smoothing(alpha=0.3,beta=0.2)",col="blue",lwd=2)
```

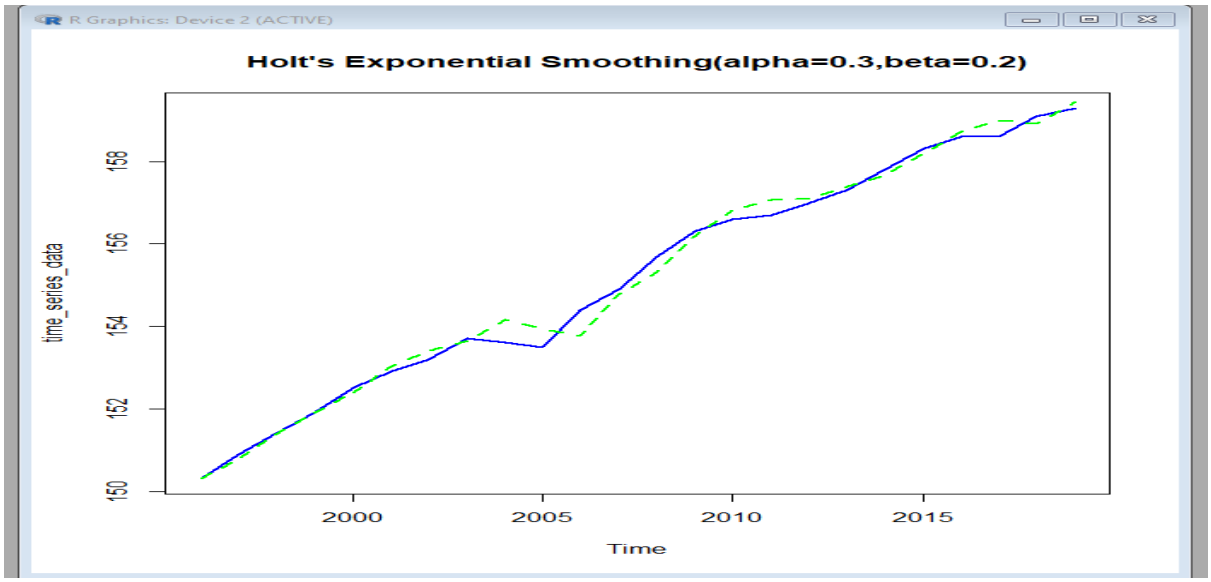
Output:



Code:

```
lines(fitted(holt_model),col="green",lty=2,lwd=2)
```

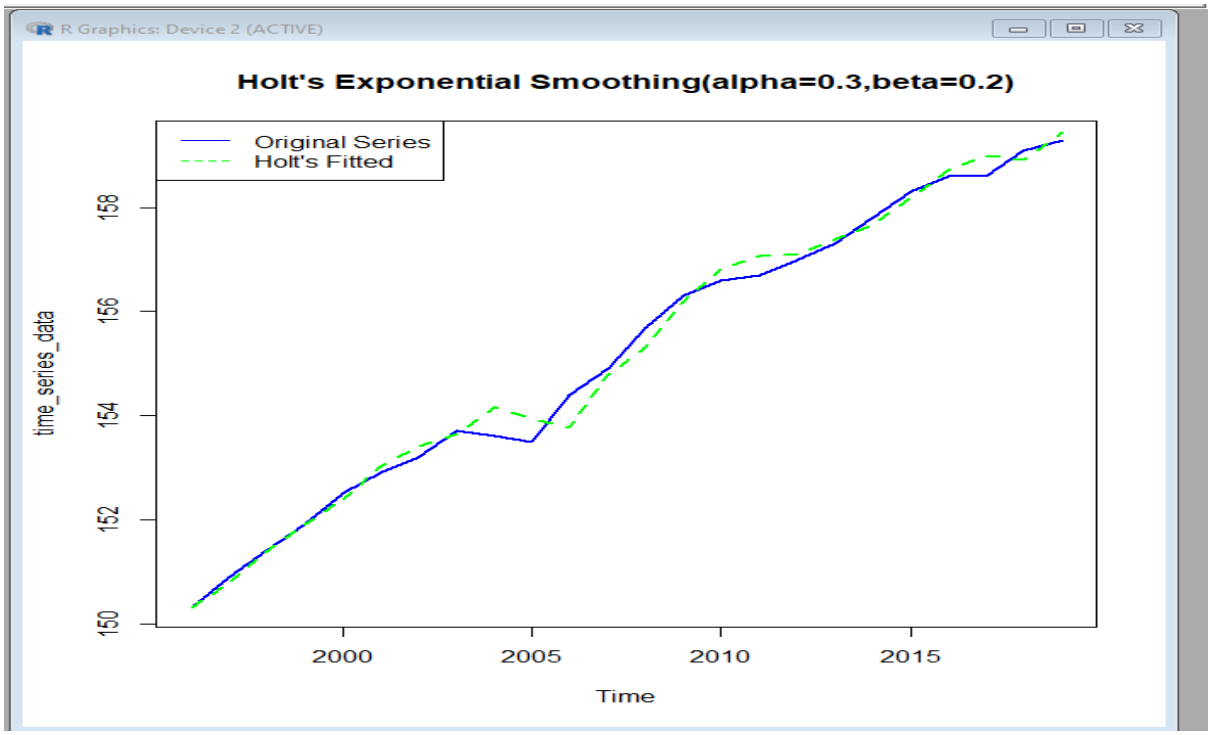
Output:



Code:

```
legend("topleft",legend=c("Original Series","Holt's
Fitted"),col=c("blue","green"),lty=1:2)
```

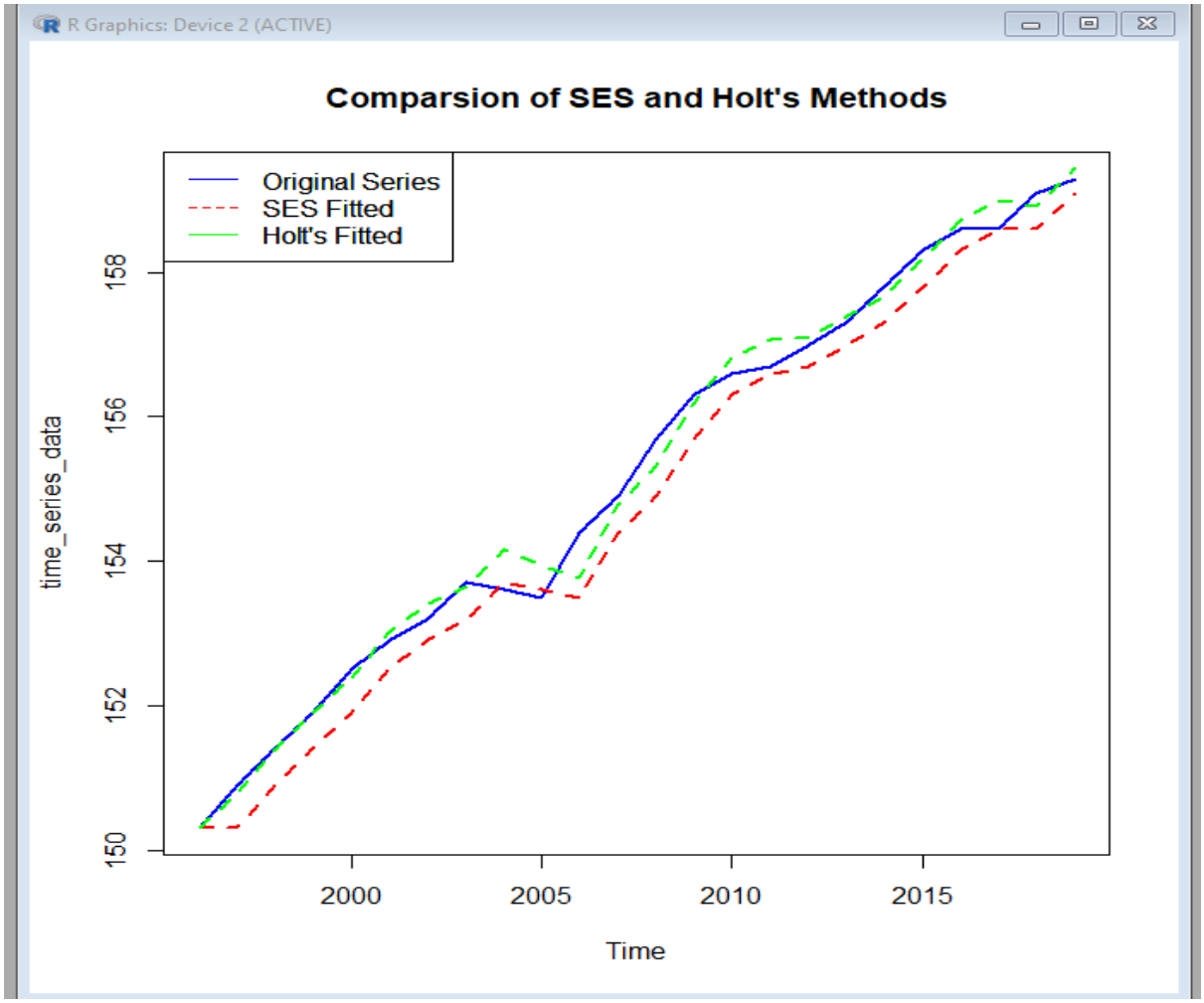
Output:



Code:

```
#plotting both model

 plot(time_series_data,main="Comparsion of SES and Holt's Methods",col="blue",lwd=2)

 lines(fitted(ses_model),col="red",lty=2,lwd=2)

lines(fitted(holt_model),col="green",lty=2,lwd=2)

 legend("topleft",legend=c("Original Series","SES Fitted","Holt's Fitted"),col=c("blue","red","green"),lty=1:2)
```

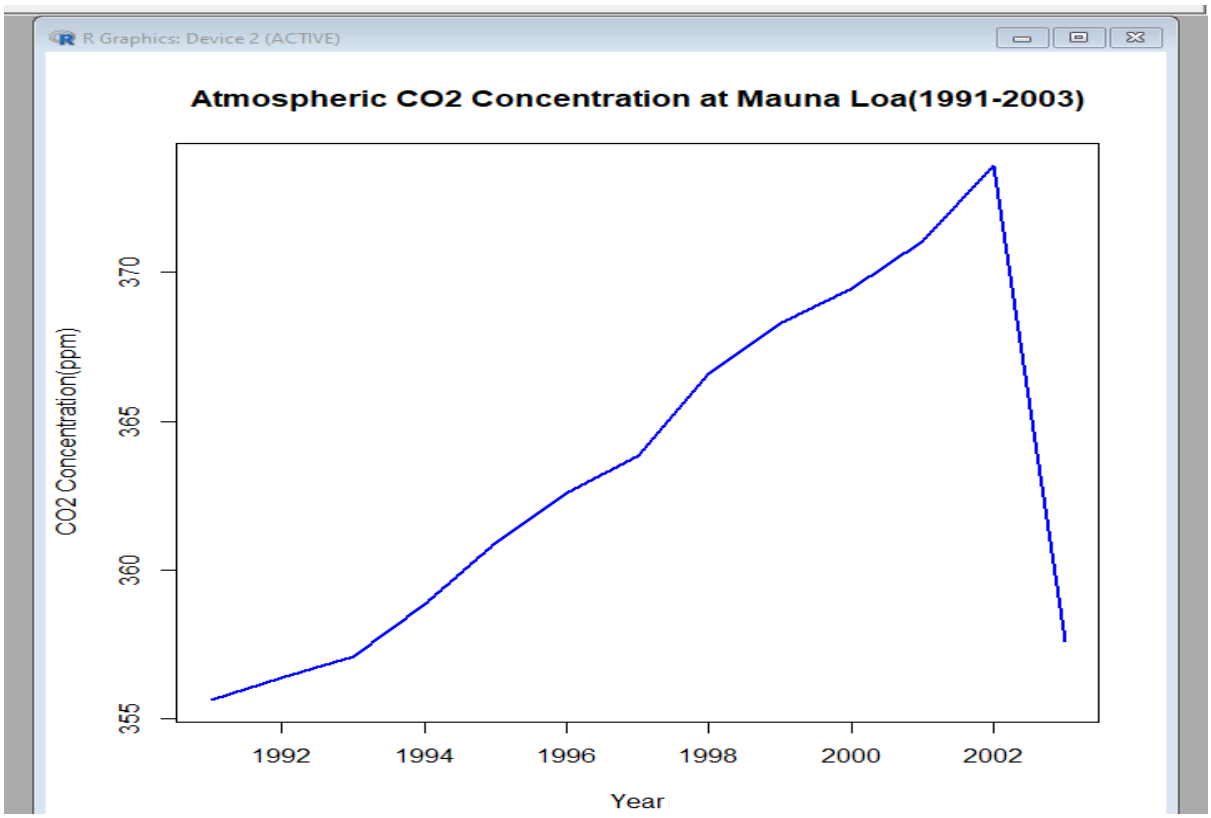
Output:



Code:

```
#Exponential and moving average smoothing 3-years

#given data

co2_concentration<-
c(355.62,356.36,357.1,358.86,360.9,362.58,363.84,366.58,368.3,369
.47,371.03,373.61,357.61)

#create a time series object for the co2 concentration data

co2_ts<-ts(co2_concentration,start=1991,frequency=1)

#plotting the time series

plot(co2_ts,main="Atmospheric CO2 Concentration at Mauna
Loa(1991-2003)",xlab="Year",ylab="CO2
Concentration(ppm)",col="blue",lwd=2)
```

Output:



Code:

```
#Calculate 3-year moving average
 moving_average_3<-filter(co2_ts,rep(1/3,3),sides=2)
 #forecast the value for 2004
 #using the last three values(2001,2002,2003)to forecast for 2004
 forecast_2004<-mean(co2_concentration[11:13])
 forecast_2004
```

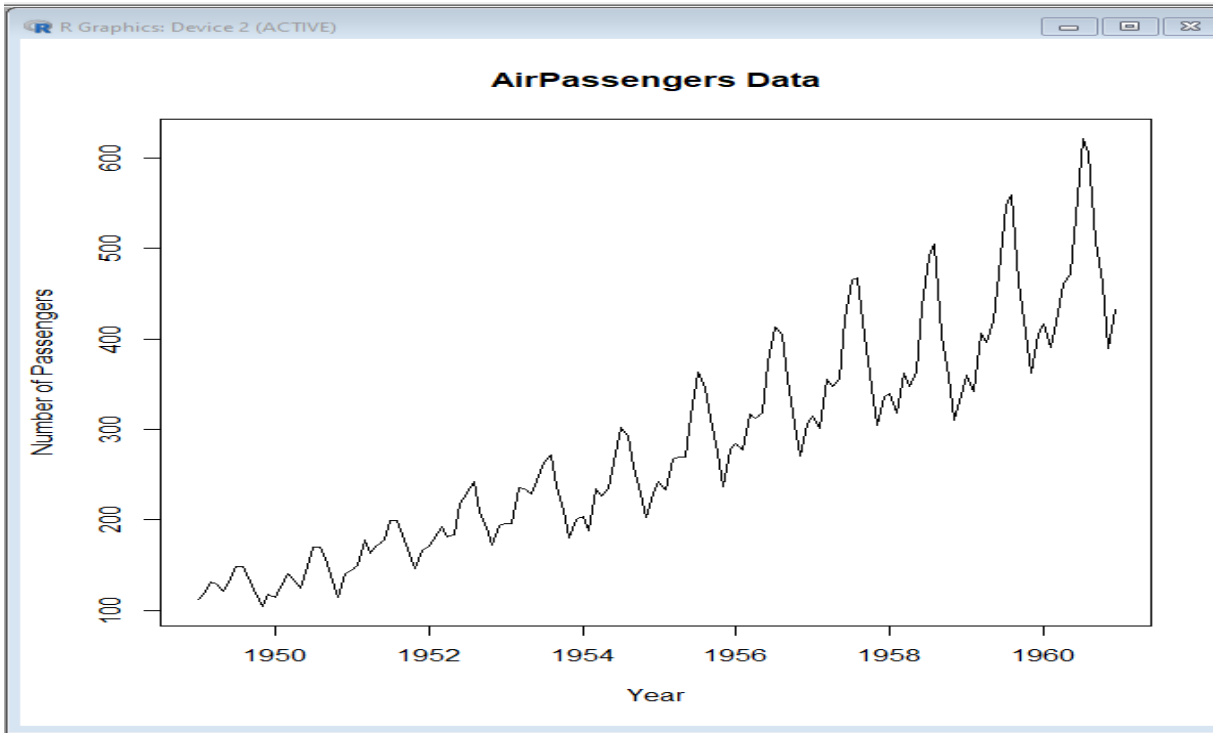
Output:

```
> #Calculate 3-year moving average
> moving_average_3<-filter(co2_ts,rep(1/3,3),sides=2)
> #forecast the value for 2004
> #using the last three values(2001,2002,2003)to forecast for 2004
> forecast_2004<-mean(co2_concentration[11:13])
> forecast_2004
[1] 367.4167
```

Code:

```
#Exponential and moving average smoothing for holt forecasting
 library(stats)
 data("AirPassengers")
 #plot the time series
 plot(AirPassengers,main="AirPassengers
Data",xlab="Year",ylab="Number of Passengers")
```

Output:



Code:

```
#apply holt-winters model (multiplicative)
 hw_model<-HoltWinters(AirPassengers,seasonal="multiplicative")
#summary
summary(hw_model)
```

Output:

```
> summary(hw_model)
            Length Class  Mode
fitted      528    mts    numeric
x           144    ts     numeric
alpha         1    -none- numeric
beta          1    -none- numeric
gamma         1    -none- numeric
coefficients 14    -none- numeric
seasonal      1    -none- character
SSE           1    -none- numeric
call          3    -none- call
```

Code:

```
#forecast for the next 12 months
forecast<-predict(hw_model,n.ahead=12)
forecast
```

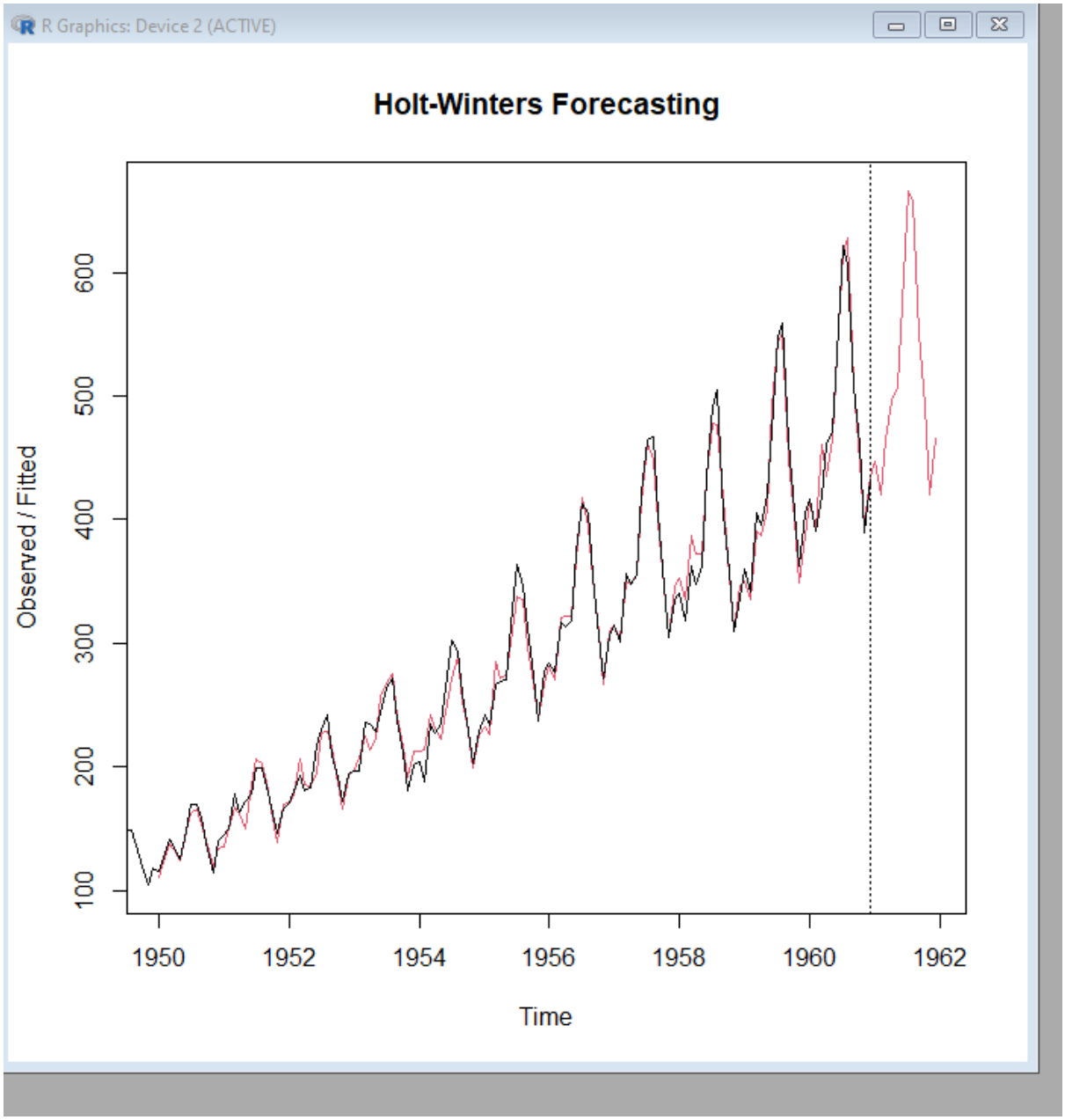
Output:

```
> #forecast for the next 12 months
> forecast<-predict(hw_model,n.ahead=12)
> forecast
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1961 447.0559 419.7123 464.8671 496.0839 507.5326 575.4509 666.5923 657.9137
          Sep      Oct      Nov      Dec
1961 550.3088 492.9853 420.2073 465.6345
> |
```

Code:

#plot the original series and forecast

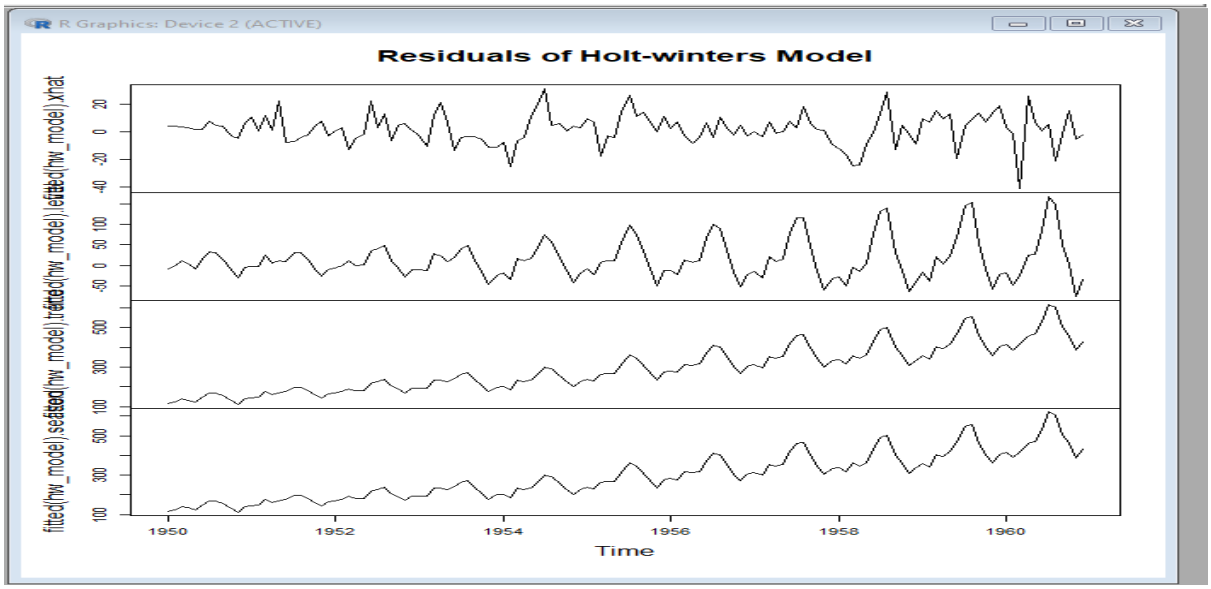
plot(hw_model,forecast,main="Holt-Winters Forecasting")



Code:

```
#plot the original series and forecast
residuals<-AirPassengers-fitted(hw_model)
 plot(residuals,main="Residuals of Holt-winters Model")
```

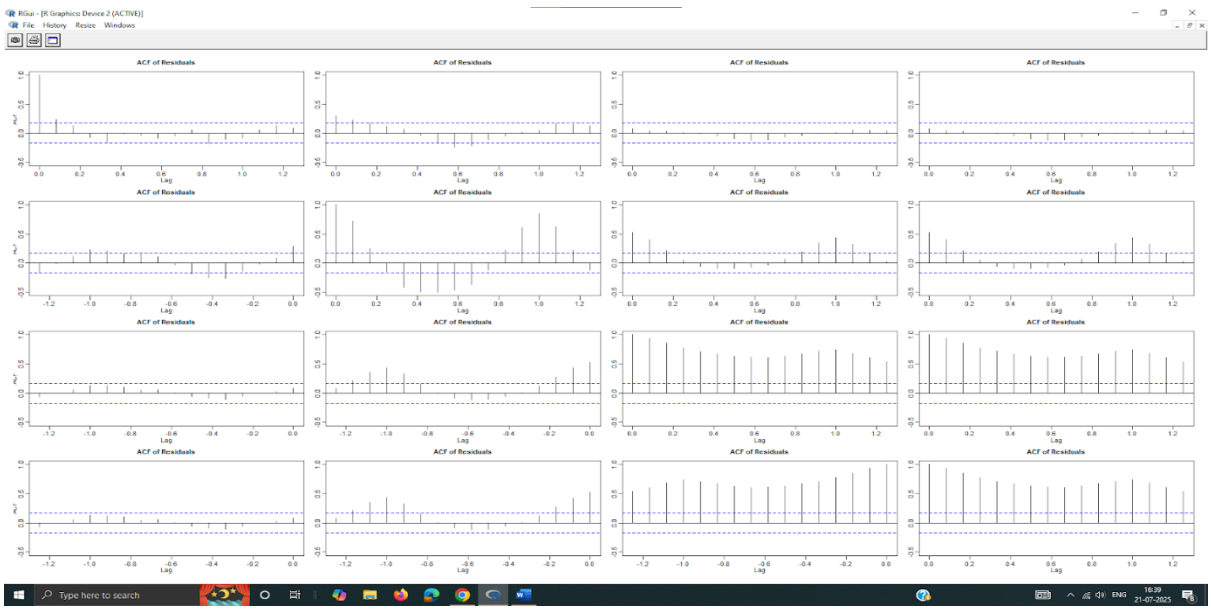
Output:



Code:

```
#check for autocorrelation
acf(residuals,main="ACF of Residuals")
```

Output:



PRACTICAL N0 – 6

Aim: Autocorrelation & Autoregression using Arima

Code:

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
data=[100,105,102,107,110,112,115,117]
s = pd.Series(data)
#model
model = ARIMA(s,order=(1,0,1))
result = model.fit()
#summary
print(result.summary())
```

Output:

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
```

```
data=[100,105,102,107,110,112,115,117]
s = pd.Series(data)
```

```
#model
model = ARIMA(s,order=(1,0,1))
result = model.fit()
```

```
#summary
print(result.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                    8
Model:                 ARIMA(1, 0, 1)   Log Likelihood                -21.936
Date:                Sat, 26 Jul 2025   AIC                             51.872
Time:                        10:46:48   BIC                             52.190
Sample:                             0   HQIC                            49.729
                                  - 8
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

Code:

```
#forecacst
future=result.forecast(steps=3)
print("Next 3 prediction:",future.tolist())
```

Output:

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

```
#forecacst
future=result.forecast(steps=3)
print("Next 3 prediction:",future.tolist())
```

```
Next 3 prediction: [115.95972016043233, 114.96894031537057, 114.10923549133534]
```

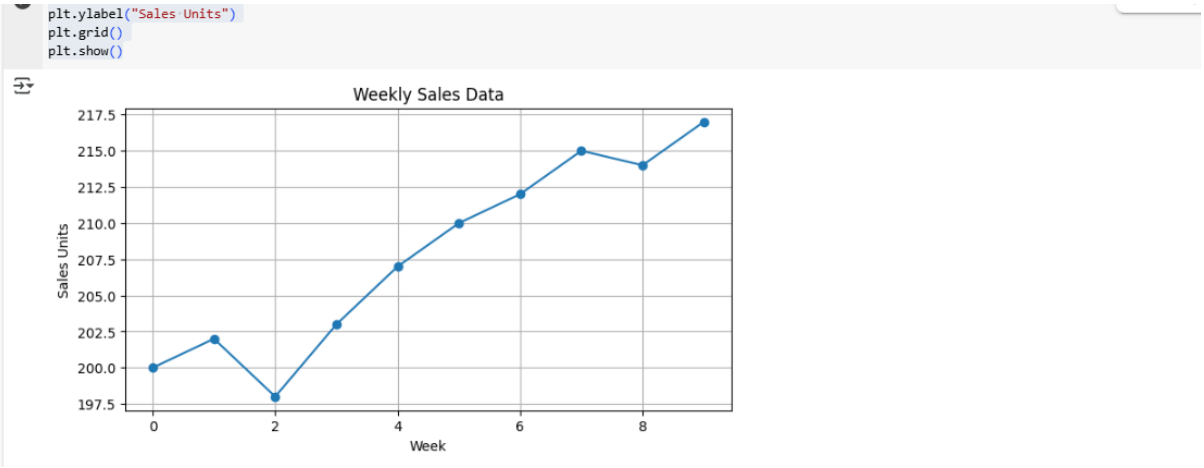
```
#next datasets
```

#Using different dataset

Code:

```
#next datasets
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
#load data
data=[200,202,198,203,207,210,212,215,214,217]
series=pd.Series(data)
#plot time series
plt.figure(figsize=(8,4))
plt.plot(series,marker='o')
plt.title("Weekly Sales Data")
plt.xlabel("Week")
plt.ylabel("Sales Units")
plt.grid()
plt.show()
```

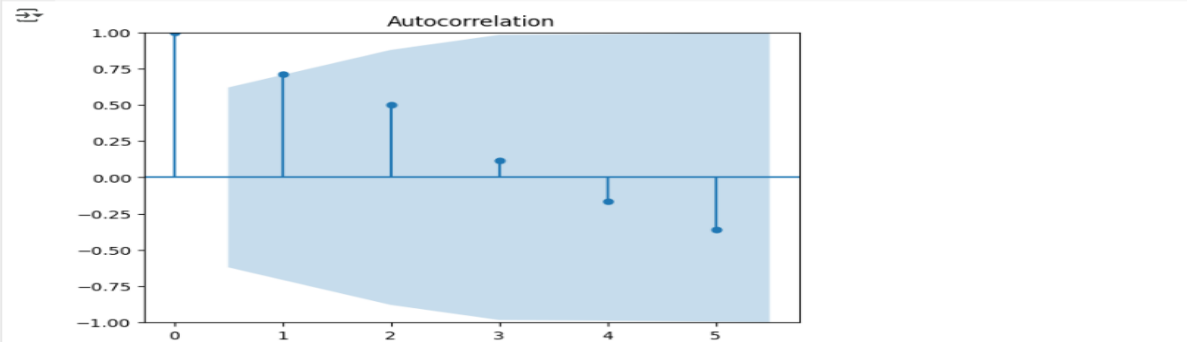
Output:

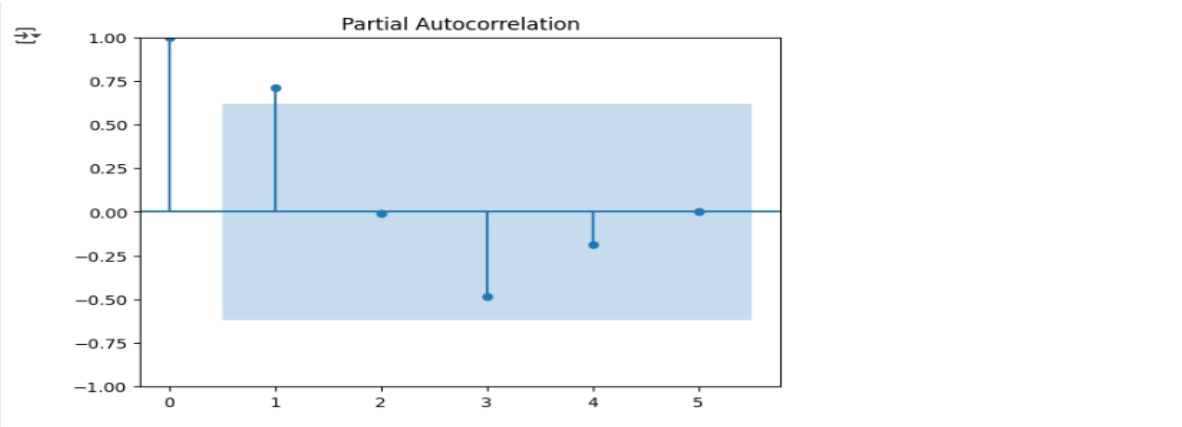


Code:

```
#ACF & PACF
plot_acf(series,lags=5)
plt.show()
plot_pacf(series,lags=5)
plt.show()
```

Output:



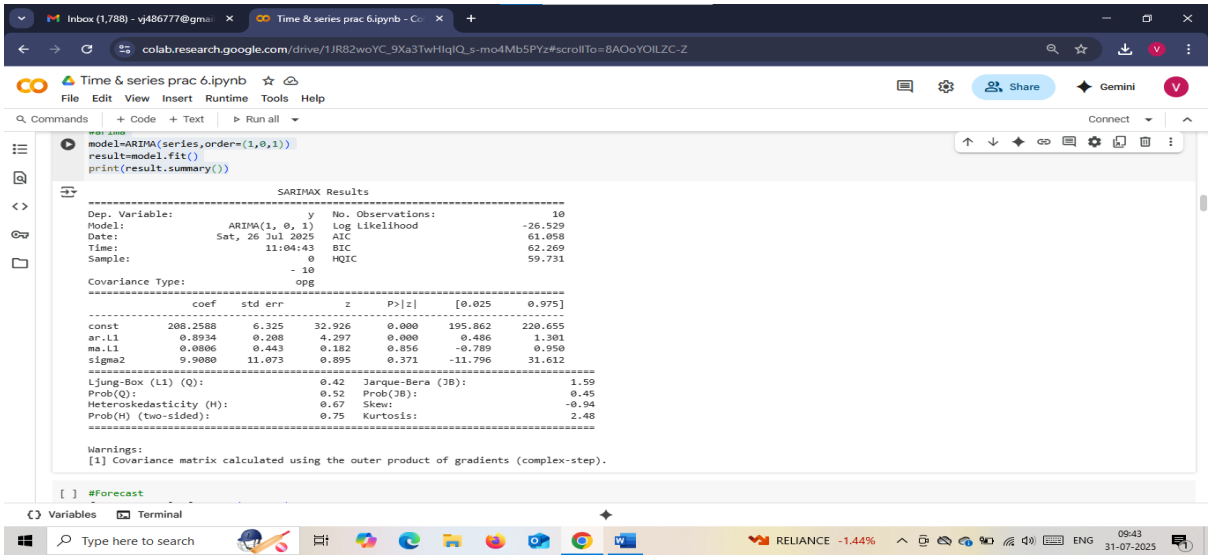



```
[ ]  #arima
     model=ARIMA(series,order=(1,0,1))
```

Code:

```
#arima
model=ARIMA(series,order=(1,0,1))
result=model.fit()
print(result.summary())
```

Output:



Code:

```
#Forecast
forecast=result.forecast(steps=3)
print("\nForecast for next 3 weeks:")
print(forecast)
```

Output:

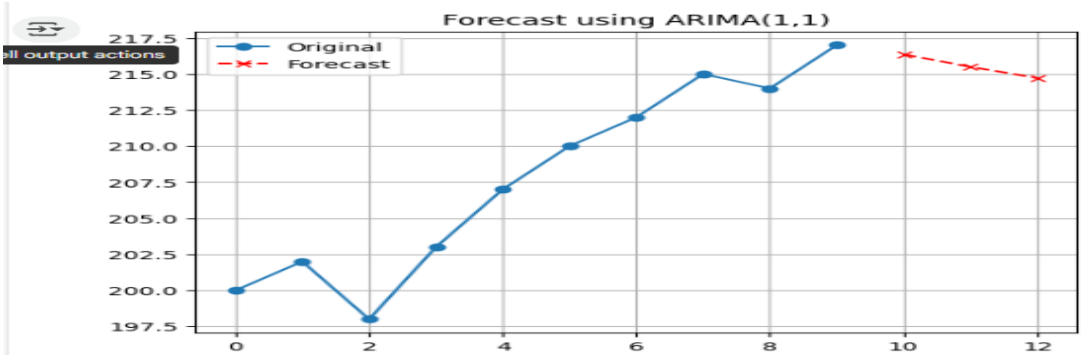
```
#Forecast
forecast=result.forecast(steps=3)
print("\nForecast for next 3 weeks:")
print(forecast)
```

```
Forecast for next 3 weeks:
10    216.362951
11    215.499264
12    214.727623
Name: predicted_mean, dtype: float64
```

Code:

```
#plot forecast
plt.plot(series,label='Original',marker='o')
forecast_index=range(len(series),len(series)+3)
plt.plot(forecast_index,forecast,label='Forecast',marker='x',linestyle=
'--',color='red')
plt.legend()
plt.title("Forecast using ARIMA(1,1)")
plt.grid()
plt.show()
print(result.summary())
```

Output:



```
[ ] print(result.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   10
Model:                 ARIMA(1, 0, 1)   Log Likelihood                -26.529
Date:                Sat, 26 Jul 2025   AIC                             61.058
Time:                        11:22:53   BIC                             62.269
Sample:                             0   HQIC                            59.731
                                  - 10
```

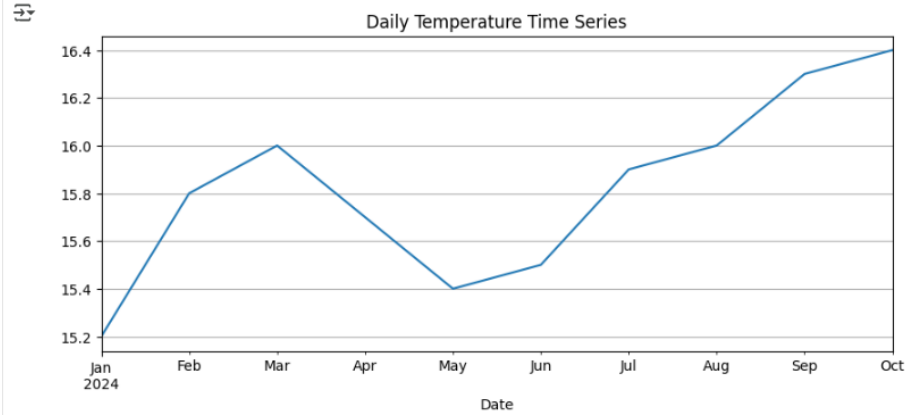
#next dataset from csv file

Code:

```
##next dataset from csv file
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.stattools import acf
#load dataset
df=pd.read_csv("temperature_data.csv",parse_dates=['Date'],index_col='D
ate')
#visualize the data
df['Temperature'].plot(title='Daily Temperature Time
Series',figsize=(10,4))
plt.grid()
plt.show()
```

Output:

```
#visualize the data
df['Temperature'].plot(title='Daily Temperature Time Series',figsize=(10,4))
plt.grid()
plt.show()
```



Code:

```
#calculate and plot autocovariance manually
def autocovariance(x,lag):
  x_mean=np.mean(x)
  n=len(x)
  return np.sum((x[:n-lag]-x_mean)*(x[lag:]-x_mean))/n
#print autocovariance for lags 1 to 5
for lag in range(1,6):
  cov=autocovariance(df['Temperature'].values,lag)
  print(f"Autocovariance for lag {lag}: {cov:4f}")
```

Output:

```
[ ] #calculate and plot autocovariance manually
    def autocovariance(x,lag):
      x_mean=np.mean(x)
      n=len(x)
      return np.sum((x[:n-lag]-x_mean)*(x[lag:]-x_mean))/n
```

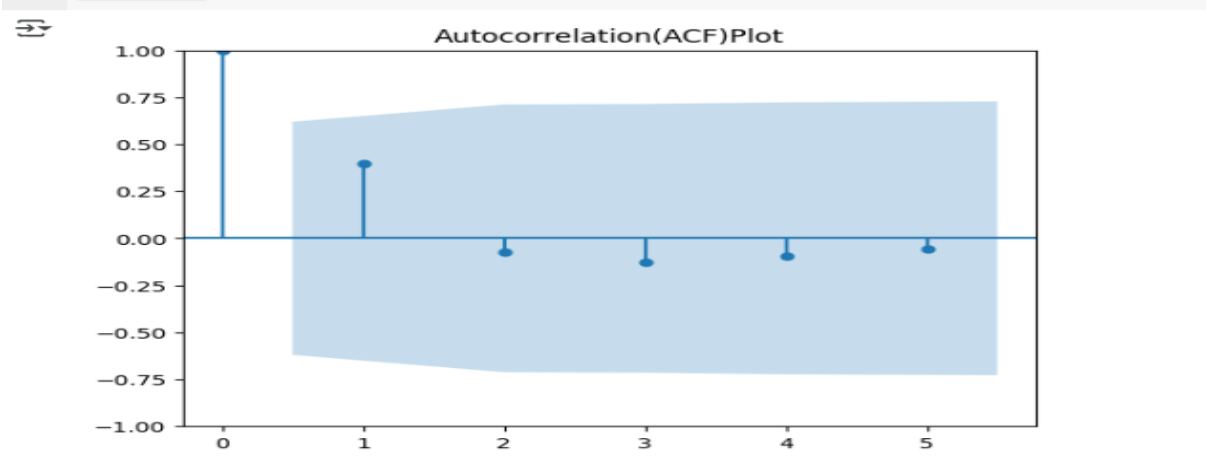
```
#print autocovariance for lags 1 to 5
for lag in range(1,6):
  cov=autocovariance(df['Temperature'].values,lag)
  print(f"Autocovariance for lag {lag}: {cov:4f}")
```

```
Autocovariance for lag 1: 0.052560
Autocovariance for lag 2: -0.009480
Autocovariance for lag 3: -0.016720
Autocovariance for lag 4: -0.012760
Autocovariance for lag 5: -0.007200
```

Code:

```
#plot autocorrelation function (ACF)
from statsmodels.graphics.tsaplots import plot_acf
#ACF
plot_acf(df['Temperature'],lags=5)
plt.title("Autocorrelation(ACF)Plot")
plt.show()
```

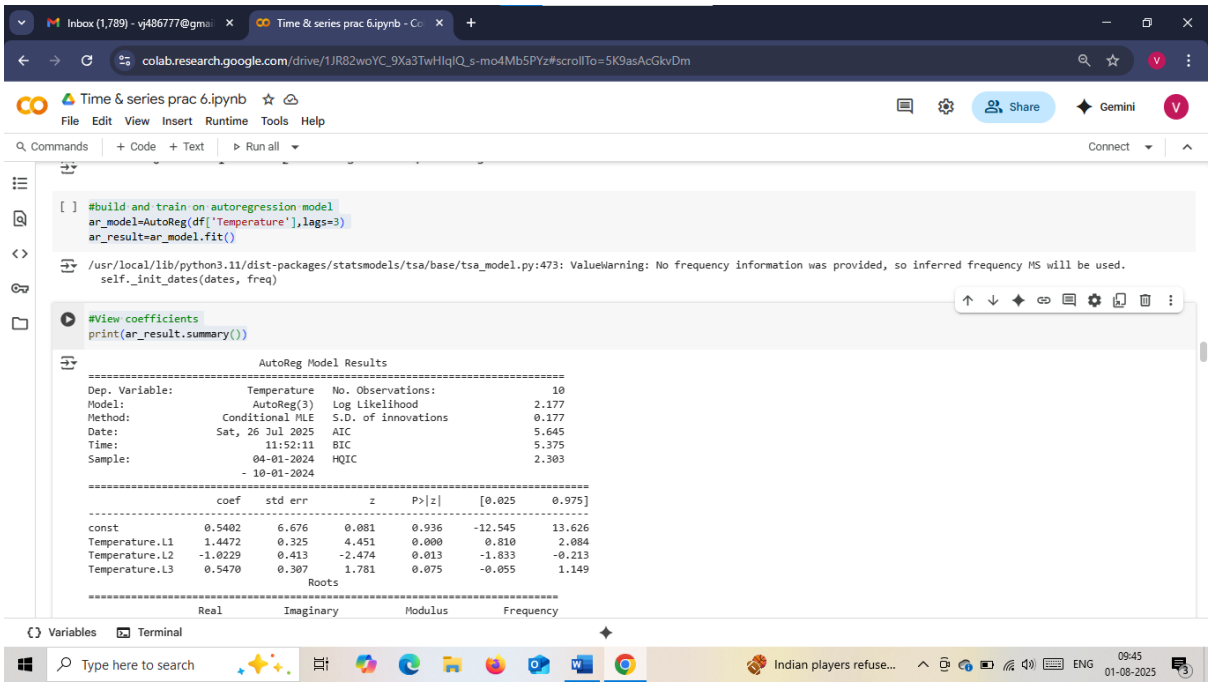
Output:



Code:

```
#build and train on autoregression model
ar_model=AutoReg(df['Temperature'],lags=3)
ar_result=ar_model.fit()
#View coefficients
print(ar_result.summary())
```

Output:



Code:

```
#make prediction and plot
#for next 10 values
forecast=ar_result.predict(start=len(df),end=len(df)+9,dynamic=False)
print(forecast)
```

Output:

```
#make prediction and plot
#for next 10 values
forecast=ar_result.predict(start=len(df),end=len(df)+9,dynamic=False)
print(forecast)

2024-11-01    16.352744
2024-12-01    16.346167
2025-01-01    16.439687
2025-02-01    16.555907
2025-03-01    16.624839
2025-04-01    16.656870
2025-05-01    16.696286
2025-06-01    16.758271
2025-07-01    16.825176
2025-08-01    16.880156
Freq: MS, dtype: float64
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/deterministic.py:308: UserWarning: Only PeriodIndexes, DatetimeIndexes w
  fcast_index = self._extend_index(index, steps, forecast_index)
```

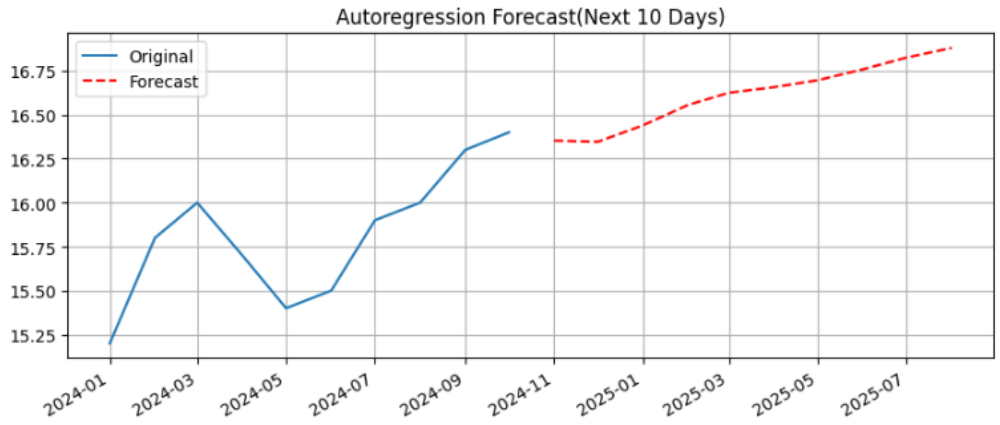
Code:

```
#plot original data and forecast
plt.figure(figsize=(10,4))
plt.plot(df['Temperature'],label='Original')
forecast.plot(label='Forecast',linestyle='--',color='red')
plt.title('Autoregression Forecast(Next 10 Days)')
plt.legend()
plt.grid()
plt.show()
```

Output:

```
plt.title('Autoregression Forecast(Next 10 Days)')
plt.legend()
plt.grid()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/pandas/plotting/_matplotlib/core.py:975: UserWarning: This axis already has a conver
  return ax.plot(*args, **kwds)
```




```
#program 1 project next2 years big billon days sales of flipkart using
ARMA(1)
```

Code:

```
#Hypothetical dataset(2014-2023)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

#load data
df=pd.read_csv("Hypodataset.csv")
#Create time series
sales = df['BBD Sales(in Crores)']
years = df['Year']
sales_series=pd.Series(data=sales.values,index=years)
print("Sales Data:\n",sales_series)
```

Output:

```
[ ] #program 1 project next2 years big billon days sales of flipkart using ARMA(1)
    #Hypothetical dataset(2014-2023)
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from statsmodels.tsa.arima.model import ARIMA
    from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

    #load data
    df=pd.read_csv("Hypodataset.csv")
```

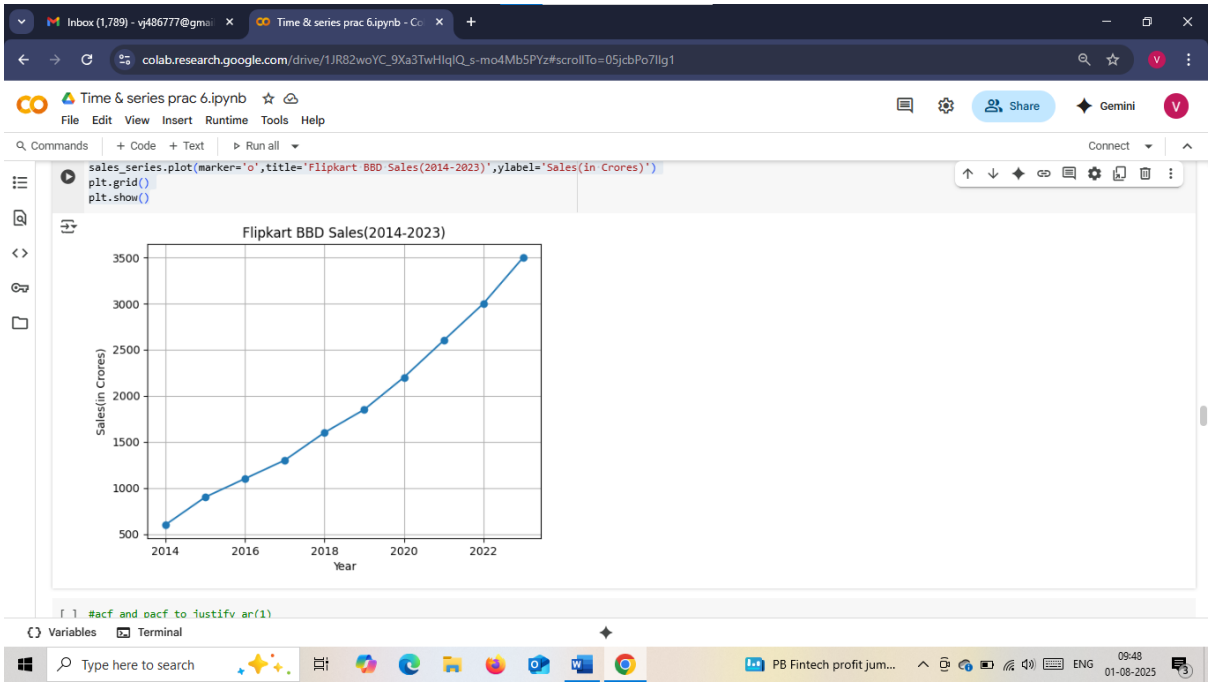
```
#Create time series
sales = df['BBD Sales(in Crores)']
years = df['Year']
sales_series=pd.Series(data=sales.values,index=years)
print("Sales Data:\n",sales_series)
```

```
Sales Data:
 Year
2014     600
2015     900
2016    1100
2017    1300
2018    1600
2019    1850
2020    2200
2021    2600
```

Code:

```
#plot the time series
sales_series.plot(marker='o',title='Flipkart BBD Sales(2014-2023)',ylabel='Sales(in Crores)')
plt.grid()
plt.show()
```

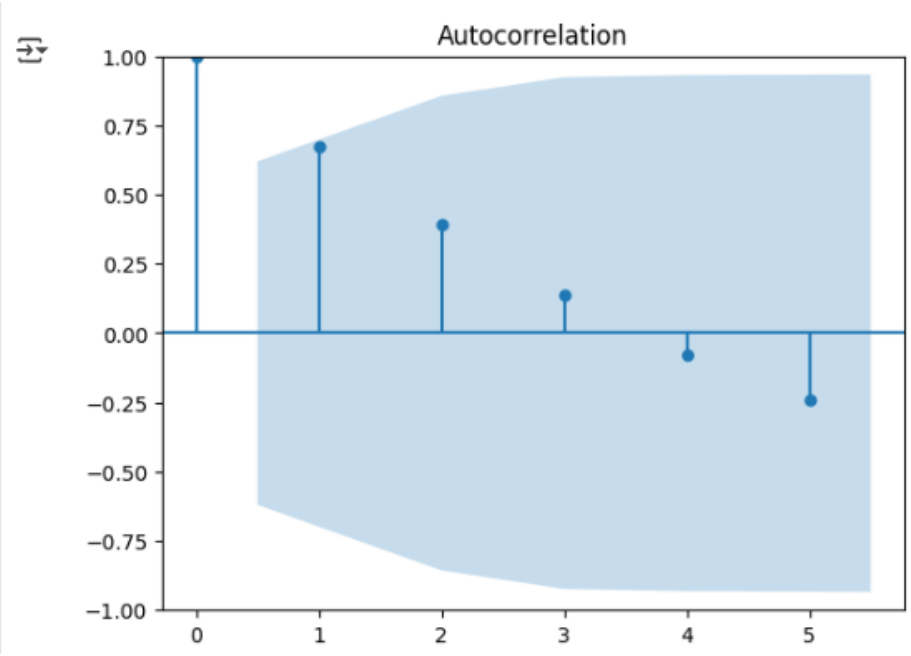
Output:

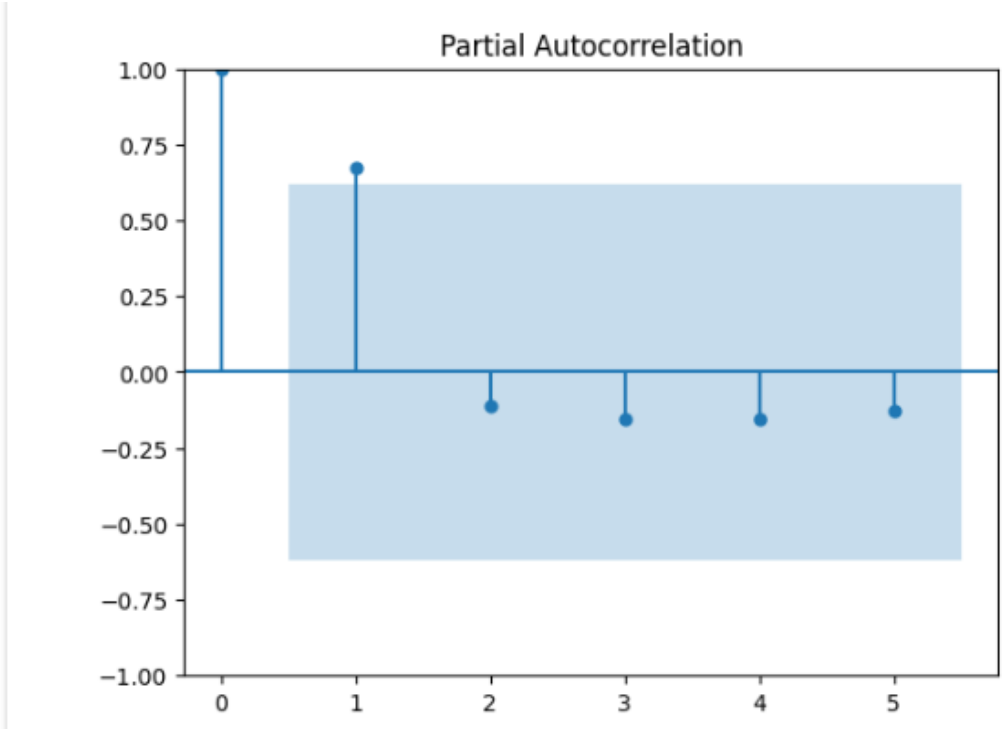


Code:

```
#acf and pacf to justify ar(1)
plot_acf(sales_series)
plt.show()
plot_pacf(sales_series)
plt.show()
```

Output:

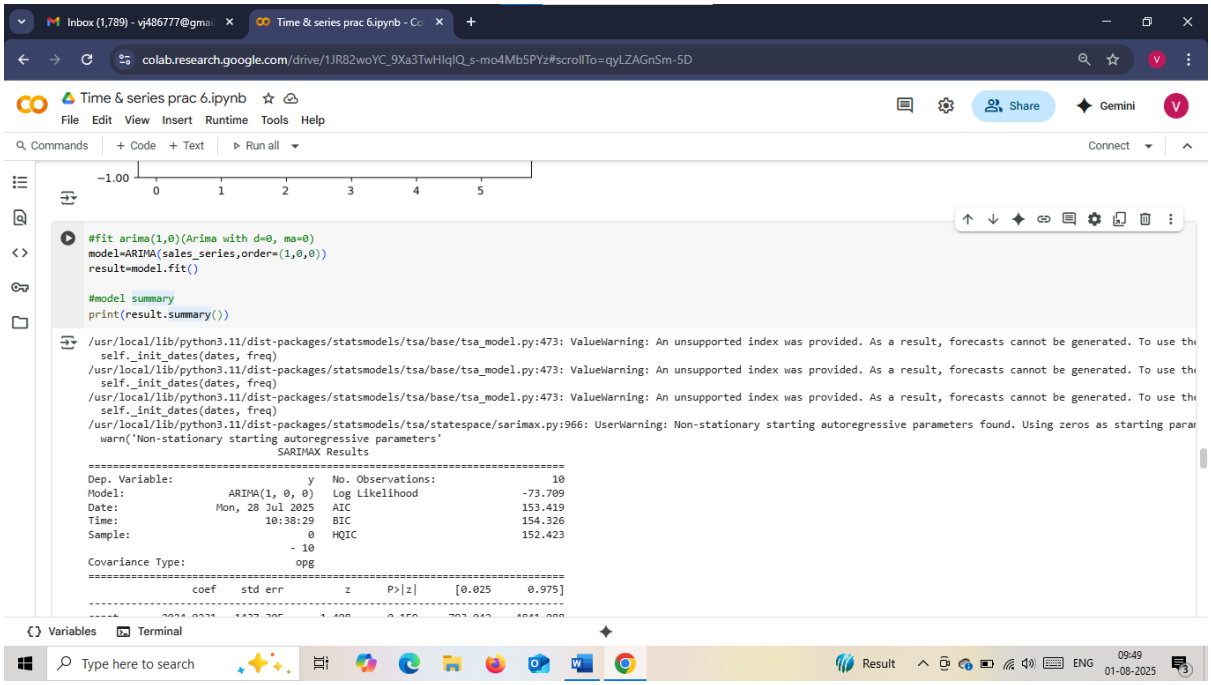




Code:

```
#fit arima(1,0)(Arima with d=0, ma=0)
model=ARIMA(sales_series,order=(1,0,0))
result=model.fit()

#model summary
print(result.summary())
```

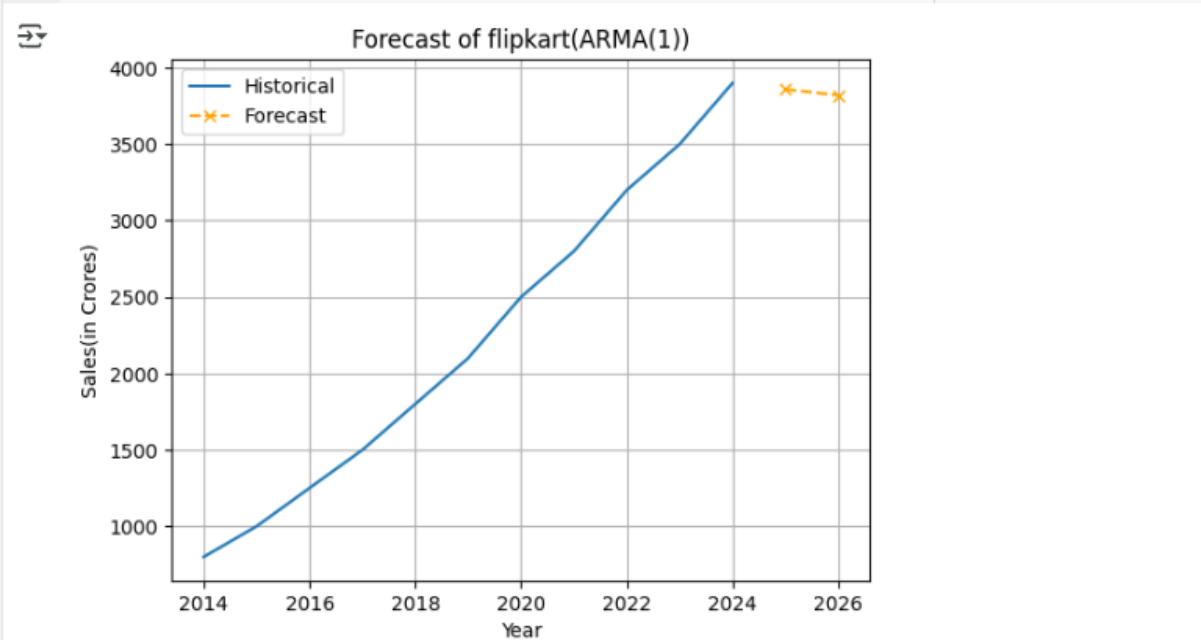
Output:



Code:

```
forecast=result.forecast(steps=2)
forecast_years=pd.date_range(start='2024',periods=2,freq='YE')
plt.plot(sales_series,label='Historical')
plt.plot(forecast_years,forecast,label='Forecast',marker='x',linestyle=
'--',color='orange')
plt.title('Forecast of flipkart(ARMA(1))')
plt.ylabel('Sales(in Crores)')
plt.xlabel('Year')
plt.legend()
plt.grid()
plt.show()
print("Forecasted Sales for 2024:", forecast.iloc[0])
print("Forecasted Sales for 2025:", forecast.iloc[1])
```

Output:




```
print("Forecasted Sales for 2024:", forecast.iloc[0])
print("Forecasted Sales for 2025:", forecast.iloc[1])
```

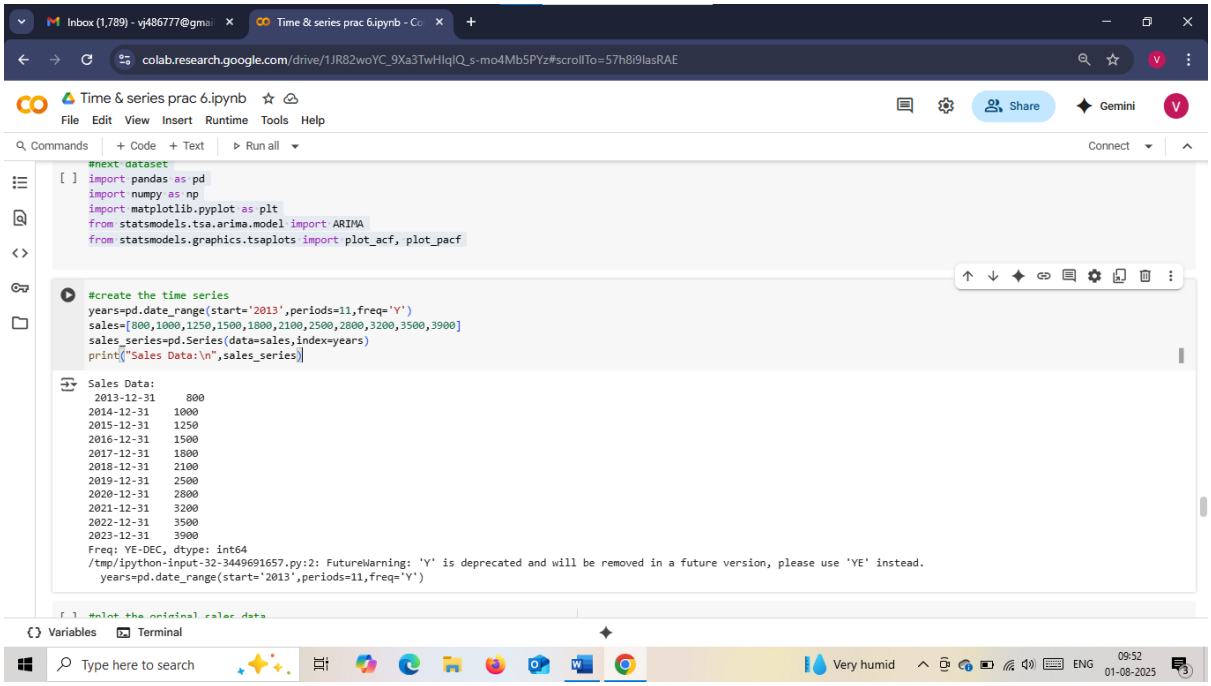
```
Forecasted Sales for 2024: 3859.5327484533027
Forecasted Sales for 2025: 3820.0366142882617
```

#Next dataset

Code:

```
#next dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
#create the time series
years=pd.date_range(start='2013',periods=11,freq='Y')
sales=[800,1000,1250,1500,1800,2100,2500,2800,3200,3500,3900]
sales_series=pd.Series(data=sales,index=years)
print("Sales Data:\n",sales_series)
```

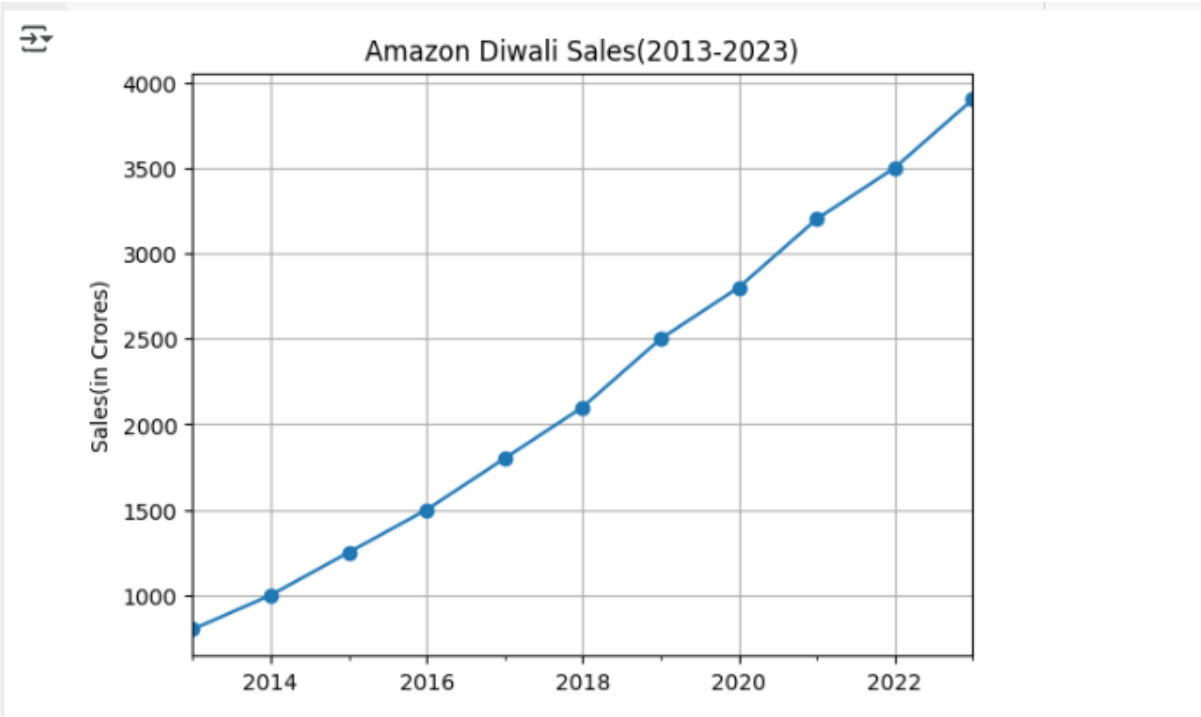
Output:



Code:

```
#plot the original sales data
sales_series.plot(marker='o',title='Amazon Diwali Sales(2013-
2023)',ylabel='Sales(in Crores)')
plt.grid()
plt.show()
```

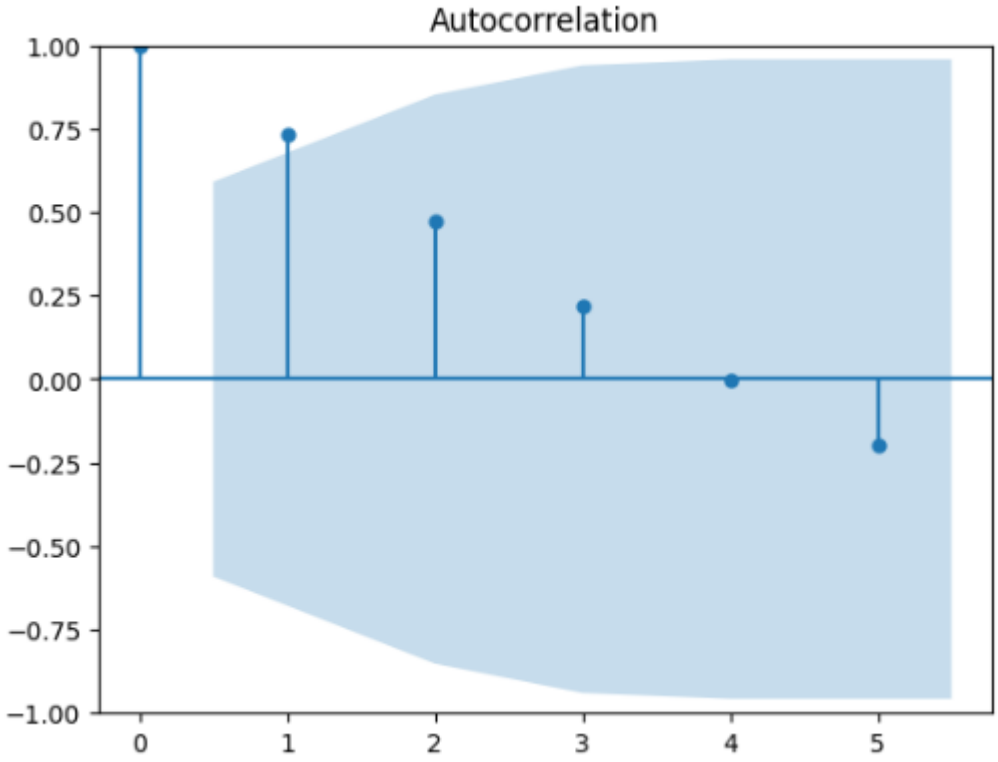
Output:

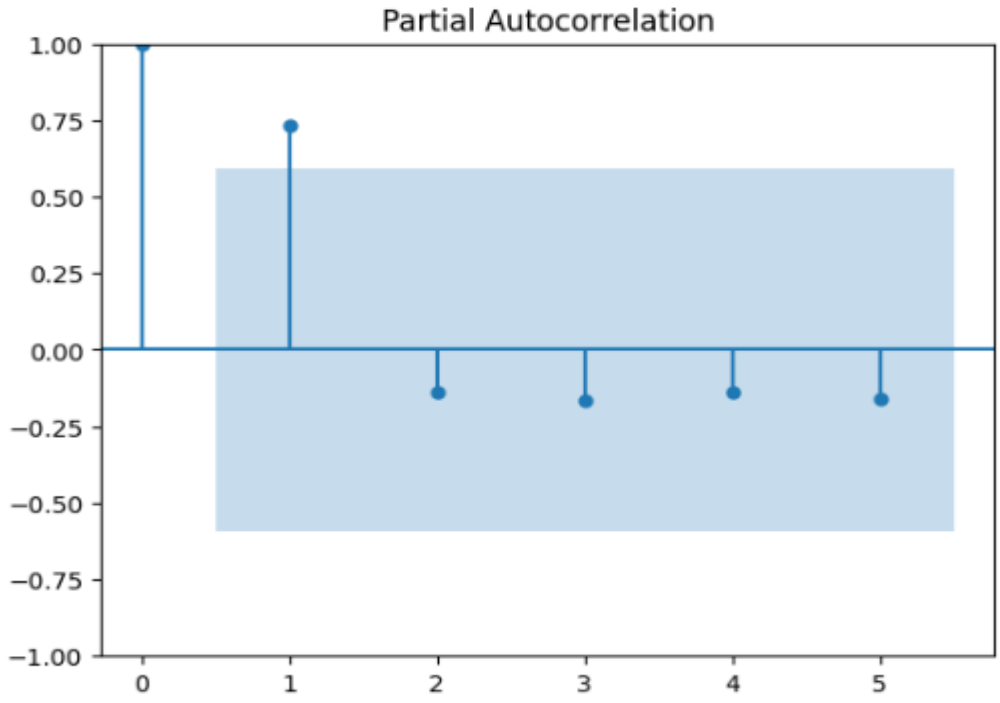


Code:

```
#acf and pacf
plot_acf(sales_series)
plot_pacf(sales_series)
plt.show()
```

Output:





Code:

```
#fit arma(1,0)
model=ARIMA(sales_series,order=(1,0,0))
result=model.fit()
#model summary
print(result.summary())
```

Output:

```
[ ] #fit arma(1,0)
    model=ARIMA(sales_series,order=(1,0,0))
    result=model.fit()
```

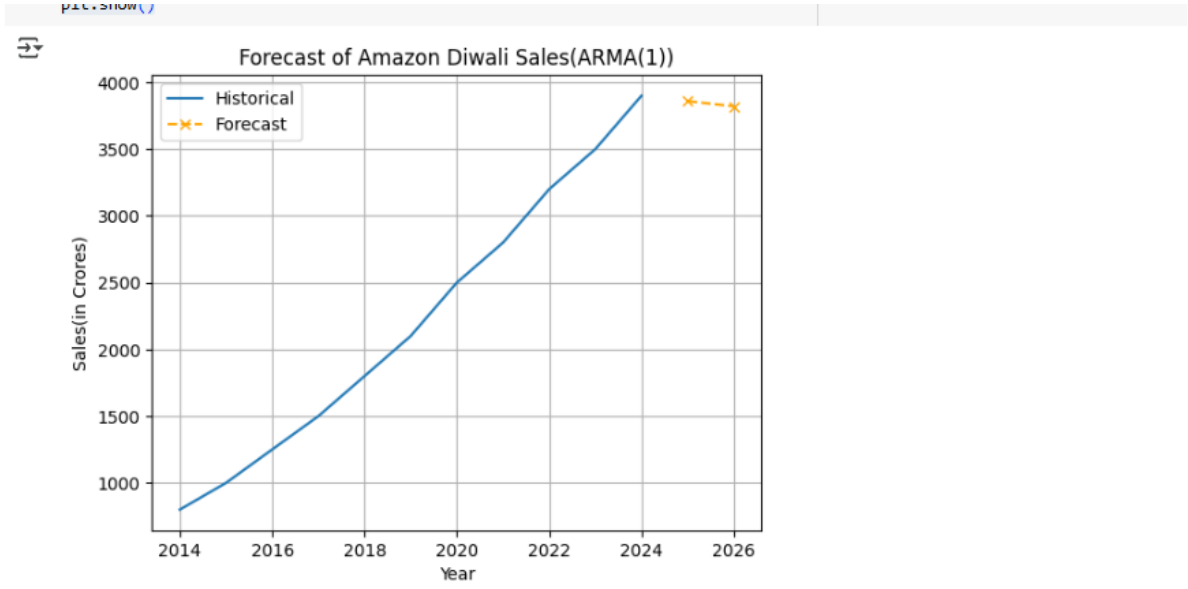
```
#model summary
print(result.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   11
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -80.457
Date:                Mon, 28 Jul 2025   AIC                            166.914
Time:                        11:08:04   BIC                            168.108
Sample:                    12-31-2013   HQIC                           166.161
                         - 12-31-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       2213.6967   1445.975      1.531      0.126    -620.362    5047.756
ar.L1          0.9760      0.201      4.863      0.000       0.583       1.369
sigma2      1.003e+05   2.06e+05      0.487      0.626   -3.03e+05    5.04e+05
==============================================================================
```

Code:

```
#forecast nest 2 years
forecast=result.forecast(steps=2)
forecast_years=pd.date_range(start='2024',periods=2,freq='YE')
#plot historical and forecasted values
plt.plot(sales_series,label='Historical')
plt.plot(forecast_years,forecast,label='Forecast',marker='x',linestyle=
'--',color='orange')
plt.title('Forecast of Amazon Diwali Sales(ARMA(1))')
plt.ylabel('Sales(in Crores)')
plt.xlabel('Year')
plt.legend()
plt.grid()
plt.show()
print("Forecasted Amazon Diwali Sales:")
for year,value in zip(forecast_years,forecast):
  print(f"{year}: {value:.2f}Crores")
```

Output:




```
print("Forecasted Amazon Diwali Sales:")
for year,value in zip(forecast_years,forecast):
  print(f"{year}: {value:.2f}Crores")
```

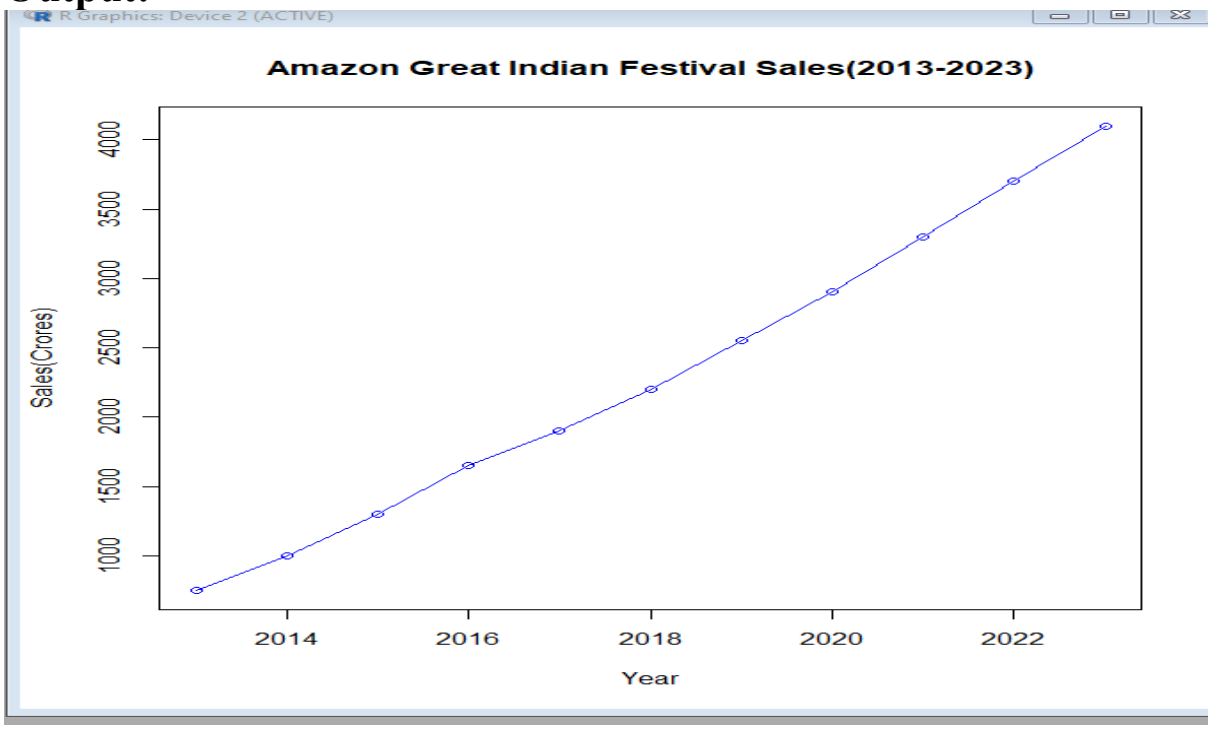
```
Forecasted Amazon Diwali Sales:
2024-12-31 00:00:00: 3859.53Crores
2025-12-31 00:00:00: 3820.04Crores
```

#Same program in r-studio

Code:

```
#load required libraries in r
library(forecast)
library(tseries)

#create time series object(from 2013,frequency=1 sice yearly data)
sales_ts<-
ts(c(750,1000,1300,1650,1900,2200,2550,2900,3300,3700,4100),start
=2013,frequency=1)
#plot the time series
plot(sales_ts,main="Amazon Great Indian Festival Sales(2013-
2023)",ylab="Sales(Crores)",xlab="Year",type="o",col="blue")
```

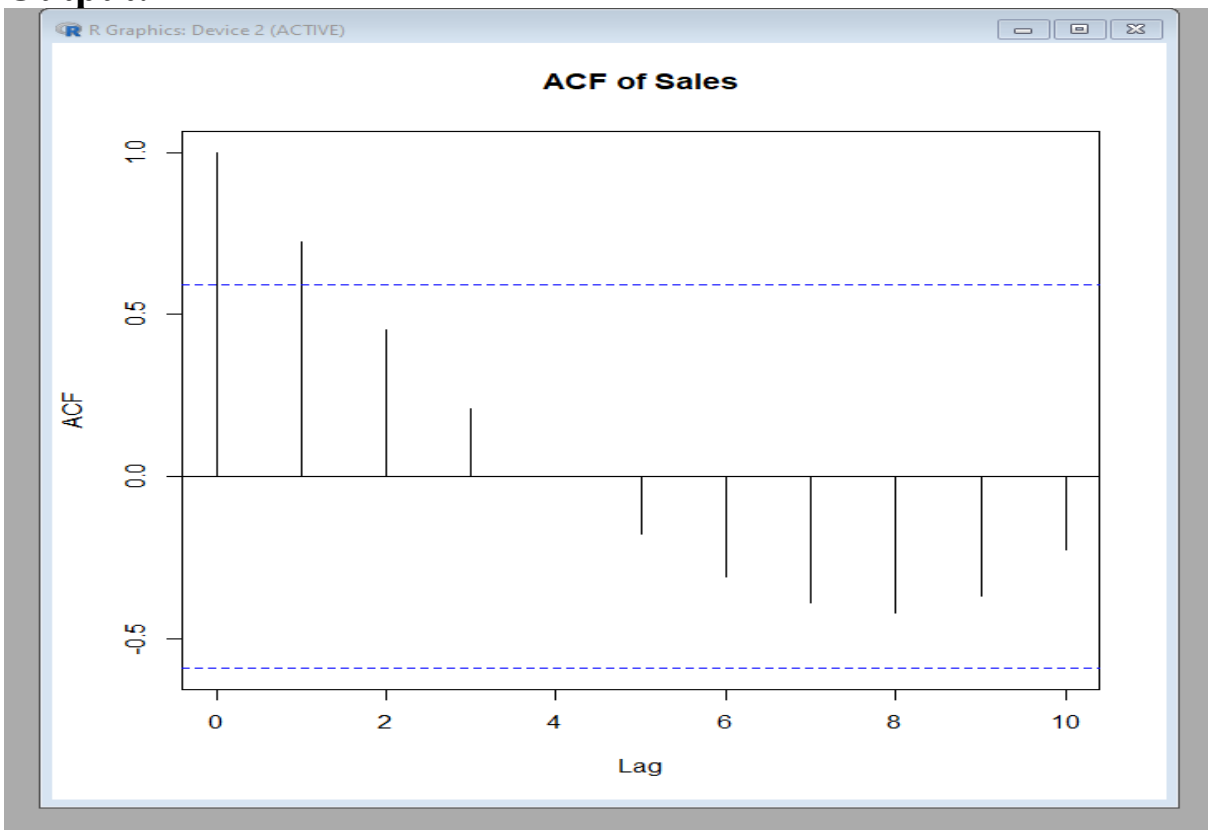
Output:



Code:

```
#check acf and pacf
acf(sales_ts,main="ACF of Sales")
```

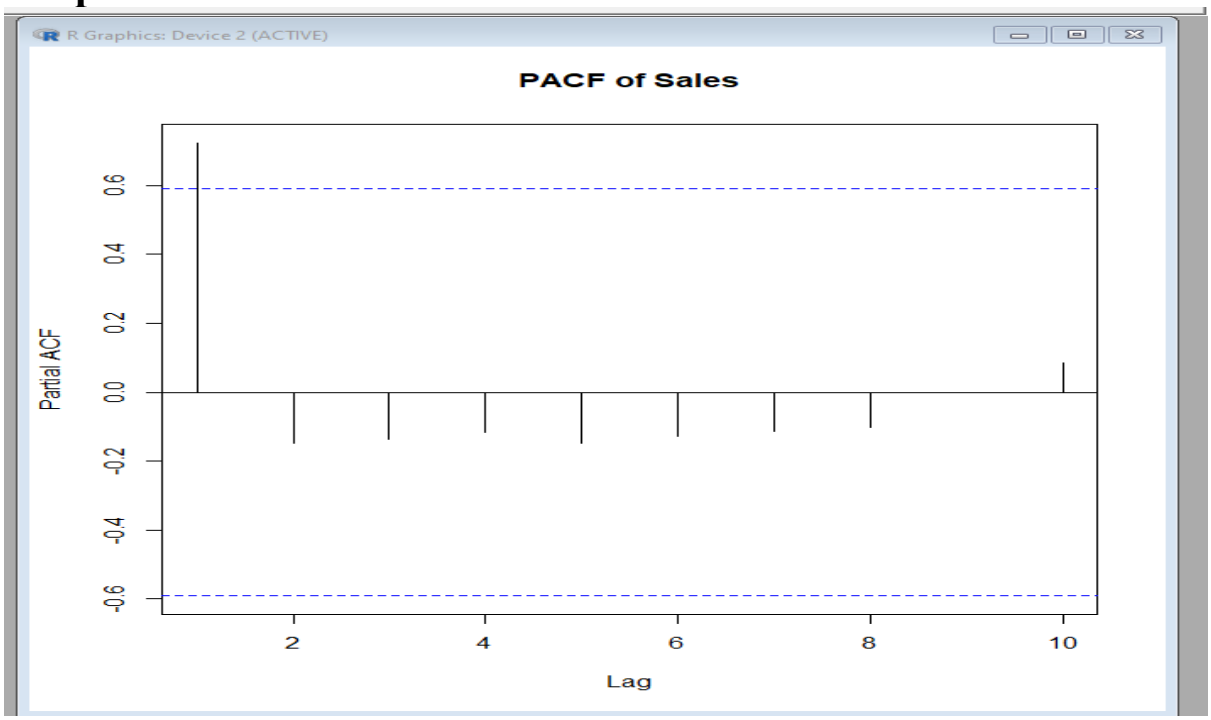
Output:



Code:

```
pacf(sales_ts,main="PACF of Sales")
```

Output:



Code:

```
arma_model<-Arima(sales_ts,order=c(1,0,1))
summary(arma_model)
```

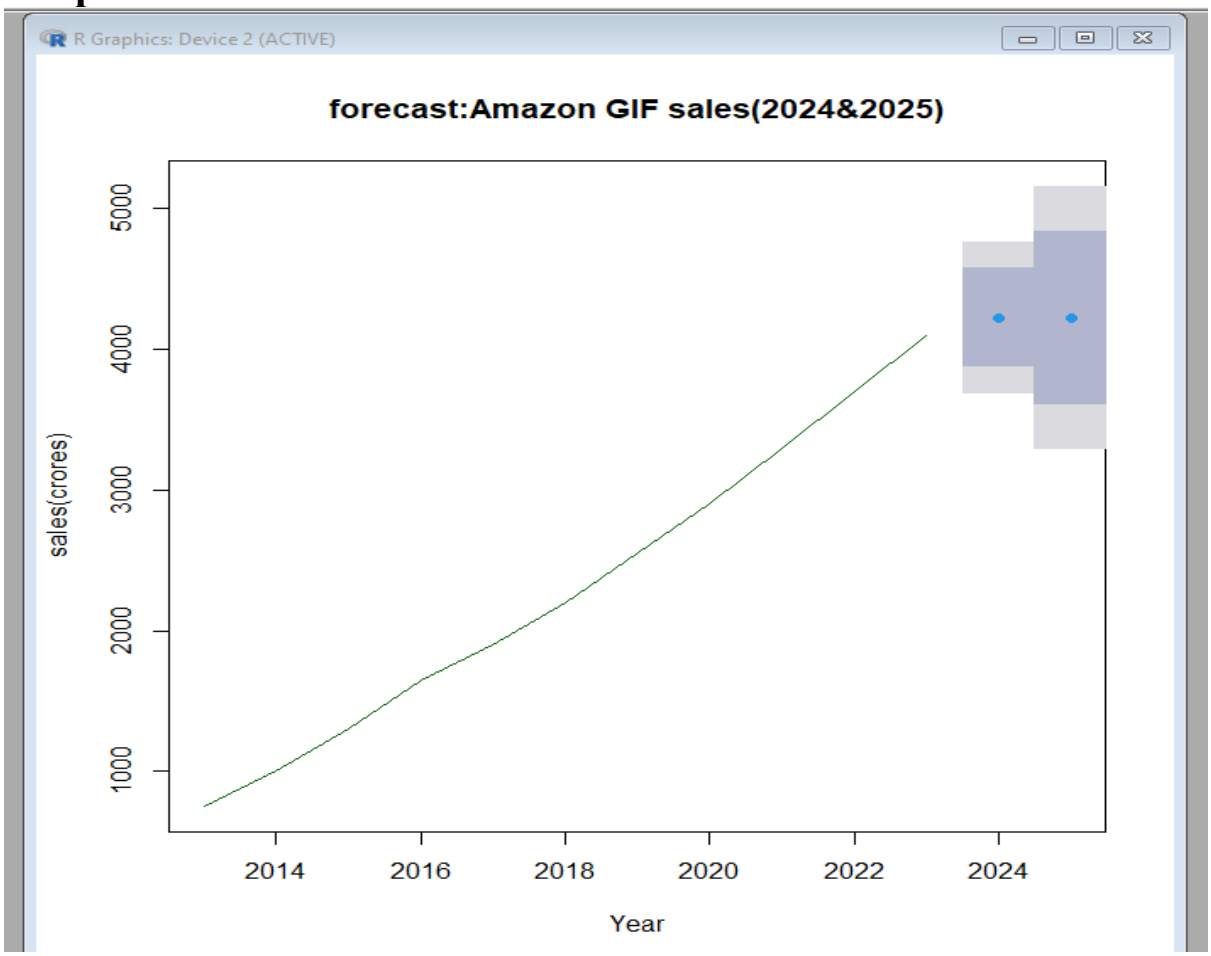
Output:

```
> summary(arma_model)
Series: sales_ts
ARIMA(1,0,1) with non-zero mean

Coefficients:
         ar1     ma1       mean
      0.9995  0.4353   3161.804
s.e.  0.0055  0.3880  12442.605

sigma^2 = 75024:  log likelihood = -79.48
AIC=166.95   AICc=173.62   BIC=168.54

Training set error measures:
                    ME     RMSE      MAE      MPE     MAPE      MASE       ACF1
Training set 213.668 233.5873 223.7415 9.653652 10.99679 0.6678851 0.02578363
```

Code:

```
forecast_result<-forecast(arma_model,h=2)
#plot the forecast
plot(forecast_result,main="forecast:Amazon GIF
sales(2024&2025)",xlab="Year",ylab="sales(crores)"
,col="darkgreen")
```

Output:



Code:

print(forecast_result)

Output:

```
>
> print(forecast_result)
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2024       4222.182 3871.158 4573.206 3685.338 4759.027
2025       4221.606 3607.715 4835.497 3282.741 5160.471
>
```

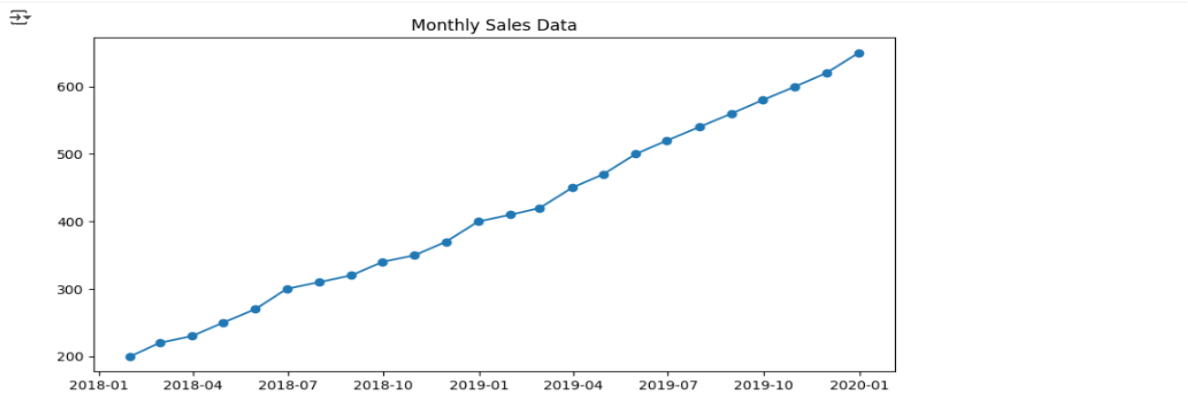
PRACTICAL N0 –7

Aim: Using Arima for forecasting

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.arima.model import ARIMA
#load dataset
#monthly sales
data=pd.DataFrame({'Month':pd.date_range(start='2018-
01',periods=24,freq='ME'),'Sales':[200,220,230,250,270,300,310,320,340,
350,370,400,410,420,450,470,500,520,540,560,580,600,620,650]})
data.set_index('Month',inplace=True)
#plot data
plt.figure(figsize=(10,5))
plt.plot(data['Sales'],marker='o')
plt.title("Monthly Sales Data")
plt.show()
```

Output:



Code:

```
#Check Stationary(ADF test)
def check_stationarity(series):
    result=adfuller(series)
    print("ADF Statistic:", result[0])
    print("p-value:",result[1])
    if result[1]<=0.05:
      print("Stationary")
    else:
      print("Non-Stationary")

check_stationarity(data['Sales'])
```

Output:

```
check_stationarity(data['Sales'])
```

```
ADF Statistic: 1.3551050643521871
p-value: 0.9969003347519994
Non-Stationary
```

Code:

```
#Difference to stationarity
data_diff=data['Sales'].diff().dropna()
check_stationarity(data_diff)
```

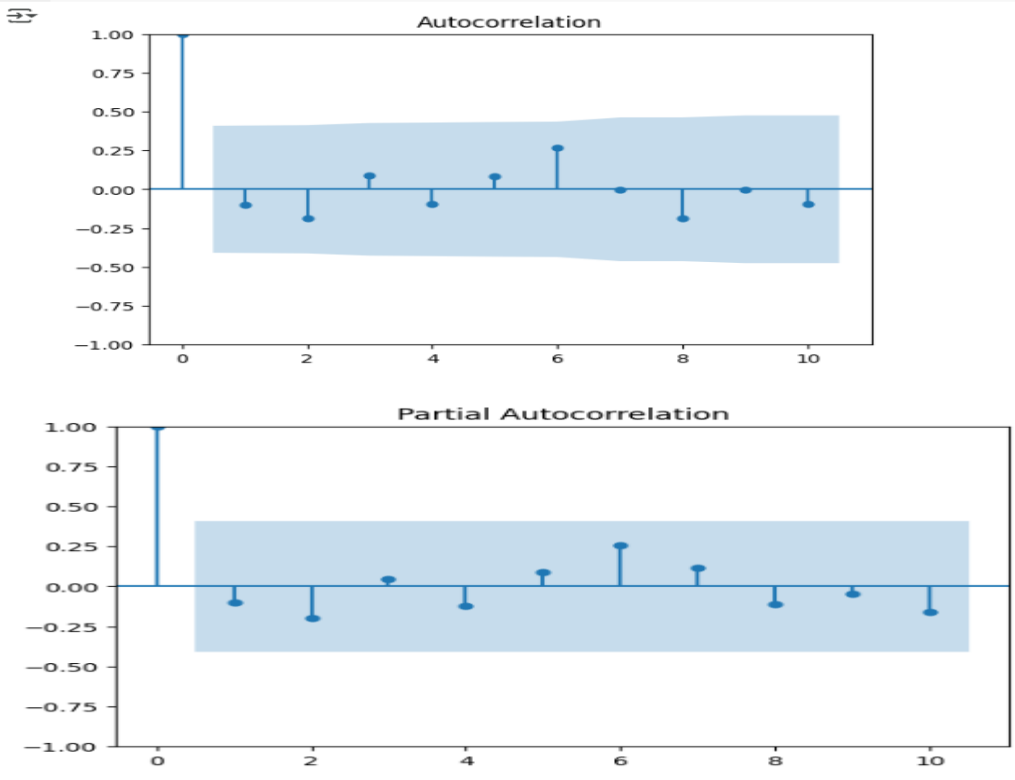
Output:

```
ADF Statistic: -3.8658943604089964
p-value: 0.0023006870324156995
Stationary
```

Code:

```
plot_acf(data_diff,lags=10)    #ARIMA(1,1,1) fromacf and pacf
plot_pacf(data_diff,lags=10)
plt.show()
```

Output:



Code:

```
#fit ARIMA Model
model=ARIMA(data['Sales'],order=(1,1,1))
model_fit=model.fit()
print(model_fit.summary())
```

Output:

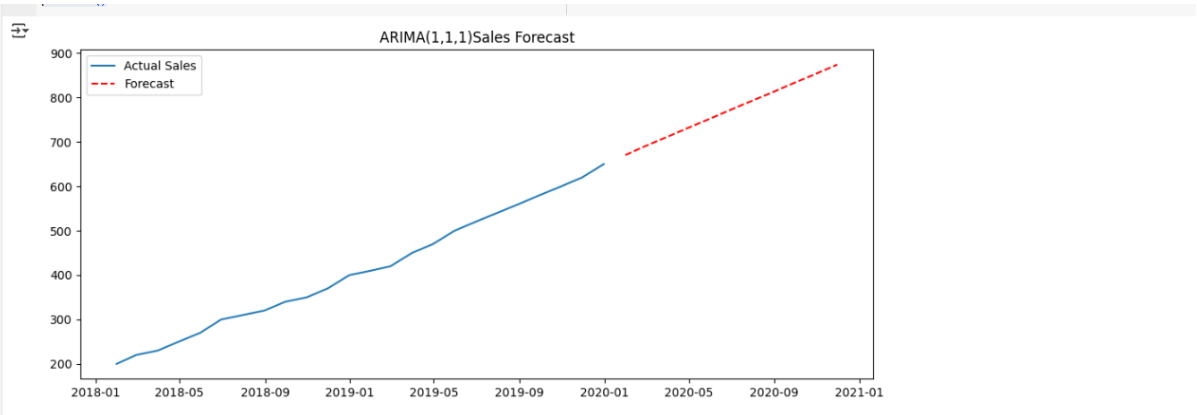
```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                   24
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -80.102
Date:                Wed, 06 Aug 2025   AIC                            166.203
Time:                        18:37:38   BIC                            169.610
Sample:                    01-31-2018   HQIC                           167.060
                         - 12-31-2019
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9995      0.005    203.702      0.000       0.990       1.009
ma.L1         -0.9137      0.355     -2.571      0.010      -1.610      -0.217
sigma2        52.6164     23.772      2.213      0.027       6.025      99.208
===================================================================================
Ljung-Box (L1) (Q):                   0.86   Jarque-Bera (JB):                 0.96
Prob(Q):                              0.35   Prob(JB):                         0.62
Heteroskedasticity (H):               0.60   Skew:                             0.06
Prob(H) (two-sided):                  0.49   Kurtosis:                         2.01
===================================================================================
```

Code:

```
#forecast future Sales
forecast = model_fit.forecast(steps=12)
future_index=pd.date_range(start=data.index[-1],periods=12,freq='ME')
forecast_df=pd.DataFrame({'Forecast':forecast},index=future_index)

plt.figure(figsize=(12,5))
plt.plot(data['Sales'],label='Actual Sales')
plt.plot(forecast_df['Forecast'],label='Forecast',color='red',linestyle
='--')
plt.title("ARIMA(1,1,1)Sales Forecast")
plt.legend()
plt.show()
```

Output:



Code:

```
#forecast Airline Passengers
url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
df=pd.read_csv(url, parse_dates=['Month'], index_col='Month')
df.head()
```

Output:

Month	Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

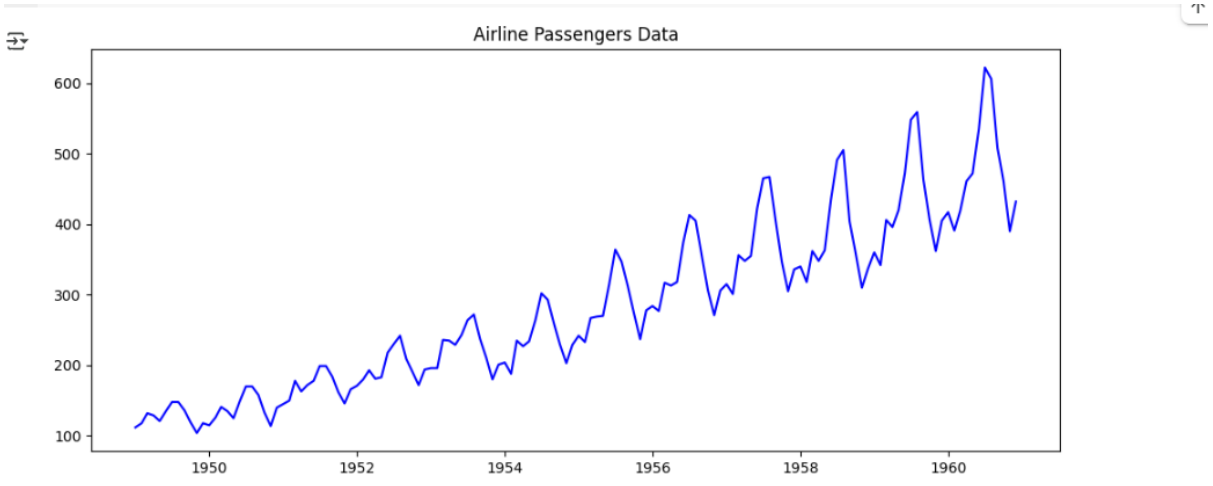
Next steps: Generate code with df | View recommended plots

```
[12] #plot Series
     plt.figure(figsize=(12,5))
```

Code:

```
#plot Series
plt.figure(figsize=(12,5))
plt.plot(df['Passengers'],color='blue')
plt.title("Airline Passengers Data")
plt.show()
```

Output:



Code:

```
#fit Arima model
model=ARIMA(df['Passengers'],order=(2,1,2))
model_fit=model.fit()
print(model_fit.summary())
```

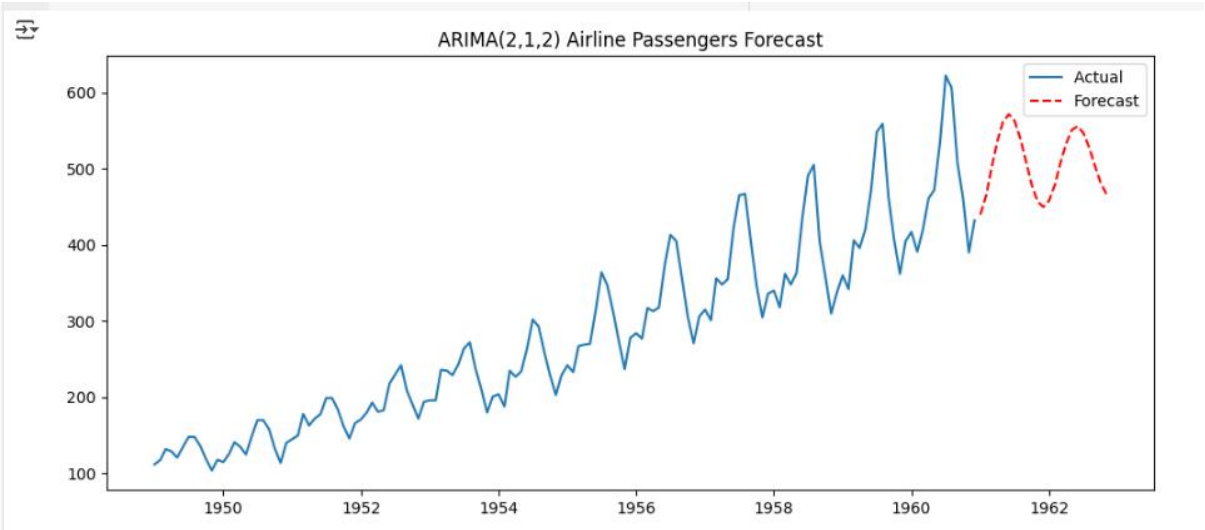
Output:

```
  self._init_dates(dates, freq)
                               SARIMAX Results
==============================================================================
Dep. Variable:             Passengers   No. Observations:                  144
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -671.673
Date:                Wed, 06 Aug 2025   AIC                           1353.347
Time:                        18:37:39   BIC                           1368.161
Sample:                    01-01-1949   HQIC                          1359.366
                         - 12-01-1960
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.6850      0.020     83.060      0.000       1.645       1.725
ar.L2         -0.9548      0.017    -55.420      0.000      -0.989      -0.921
ma.L1         -1.8432      0.124    -14.814      0.000      -2.087      -1.599
ma.L2          0.9953      0.135      7.382      0.000       0.731       1.260
sigma2       665.9600    114.029      5.840      0.000     442.467     889.453
===================================================================================
Ljung-Box (L1) (Q):                   0.30   Jarque-Bera (JB):                 1.84
```

Code:

```
#forecast
forecast = model_fit.forecast(steps=24)
future_index=pd.date_range(start=df.index[-1],periods=24,freq='MS')
forecast_df=pd.DataFrame({'Forecast':forecast},index=future_index)

plt.figure(figsize=(12,5))
plt.plot(df['Passengers'],label='Actual ')
plt.plot(forecast_df['Forecast'],label='Forecast',color='red',linestyle
='--')
plt.title("ARIMA(2,1,2) Airline Passengers Forecast")
plt.legend()
plt.show()
```

Output:

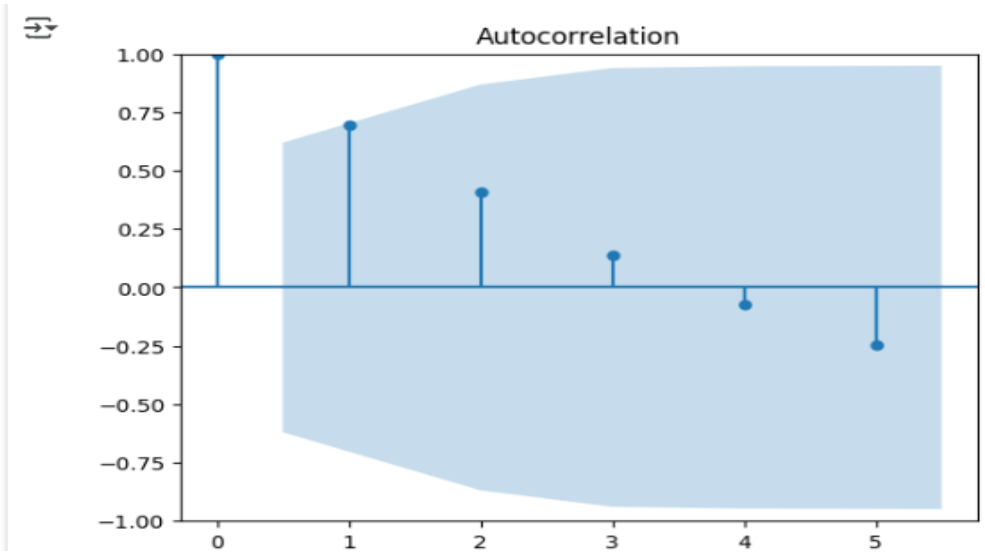


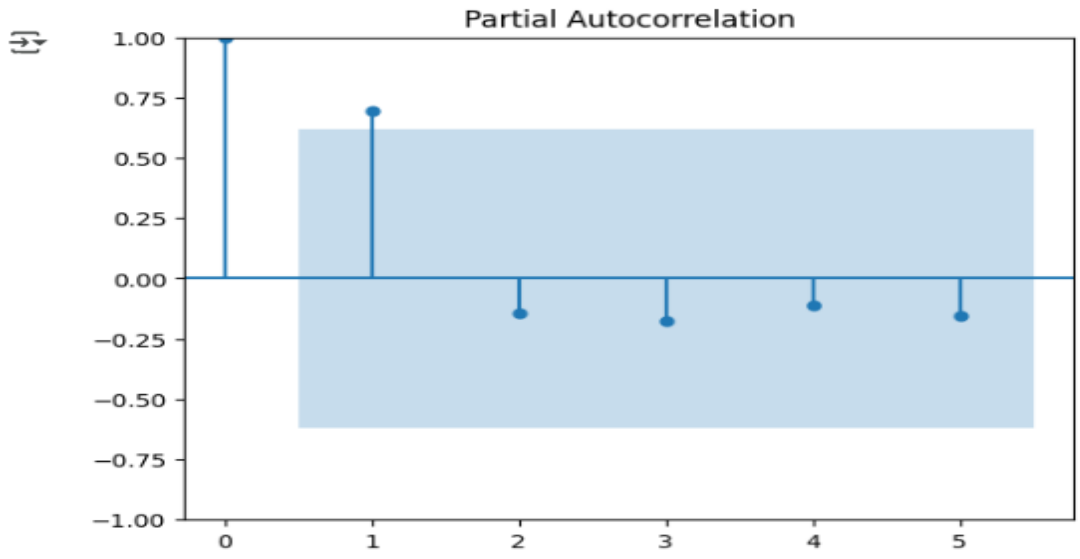
#ARMA

Code:

```
#ARMA model forecast (2,0)
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
#load series
data=[100,104,107,111,115,118,120,124,127,130]
series=pd.Series(data)
#ACF and PACF plots
plot_acf(series,lags=5)
plt.show()
plot_pacf(series,lags=5)
plt.show()
```

Output:





Code:

```
#fit ARMA(2,0) model
model=ARIMA(series,order=(2,0,0))
model_fit=model.fit()
print(model_fit.summary())
```

Output:

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   10
Model:                 ARIMA(2, 0, 0)   Log Likelihood                -18.670
Date:                Wed, 06 Aug 2025   AIC                             45.340
Time:                        18:37:39   BIC                             46.550
Sample:                             0   HQIC                            44.012
                                  - 10
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        112.5162     39.524      2.847      0.004      35.050     189.983
ar.L1          1.9128      0.262      7.291      0.000       1.399       2.427
ar.L2         -0.9369      0.233     -4.025      0.000      -1.393      -0.481
sigma2         1.1114      0.832      1.336      0.182      -0.520       2.742
===================================================================================
Ljung-Box (L1) (Q):                   4.22   Jarque-Bera (JB):                 1.03
Prob(Q):                              0.04   Prob(JB):                         0.60
Heteroskedasticity (H):               1.83   Skew:                             0.69
Prob(H) (two-sided):                  0.63   Kurtosis:                         2.25
===================================================================================

Warnings:
```

Code:

```
#forecast next 3 values
forecast=model_fit.forecast(steps=3)
print("\nforecast for next 3 days:",forecast.tolist())
```

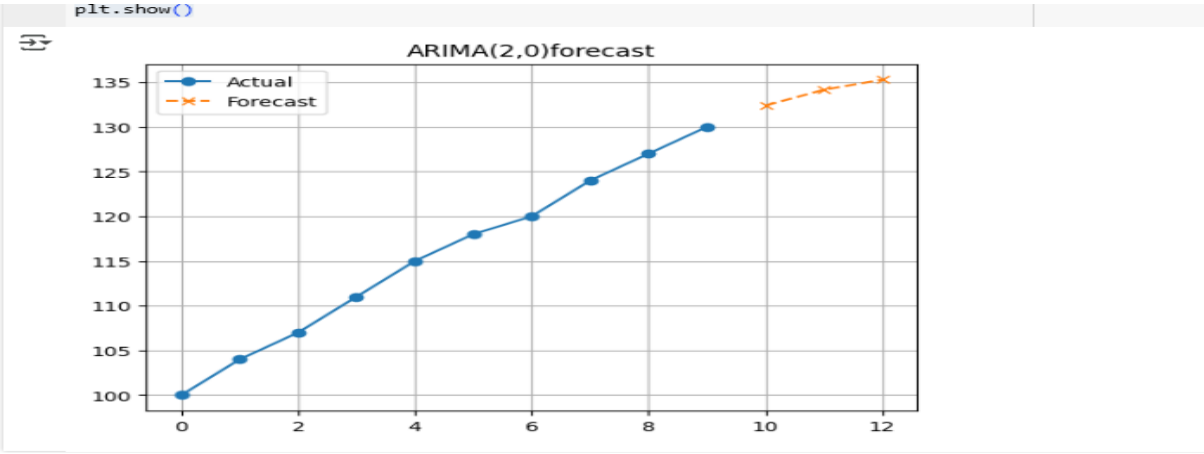
Output:

```
#forecast next 3 values
forecast=model_fit.forecast(steps=3)
print("\nforecast for next 3 days:",forecast.tolist())

forecast for next 3 days: [132.3893208087086, 134.1488641347984, 135.27591924000663]
```

Code:

```
#plot
plt.plot(series,label="Actual",marker='o')
plt.plot(range(len(series),len(series)+3),forecast,label="Forecast",mar
ker='x',linestyle='--')
plt.title("ARIMA(2,0)forecast")
plt.legend()
plt.grid()
plt.show()
```

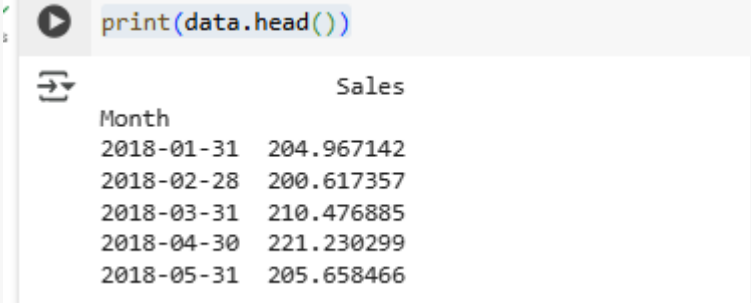
Output:



Code:

```
pip install pandas numpy matplotlib statsmodels pmdarima

#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
#from pmdarima import auto_arima
#create
np.random.seed(42)
months=pd.date_range(start='2018-01-01',periods=60,freq='ME')
sales=200+np.arange(60)*2+np.random.normal(0,10,60) #upward trend +
noise
data=pd.DataFrame({'Month':months,'Sales':sales})
data.set_index('Month',inplace=True)
print(data.head())
```

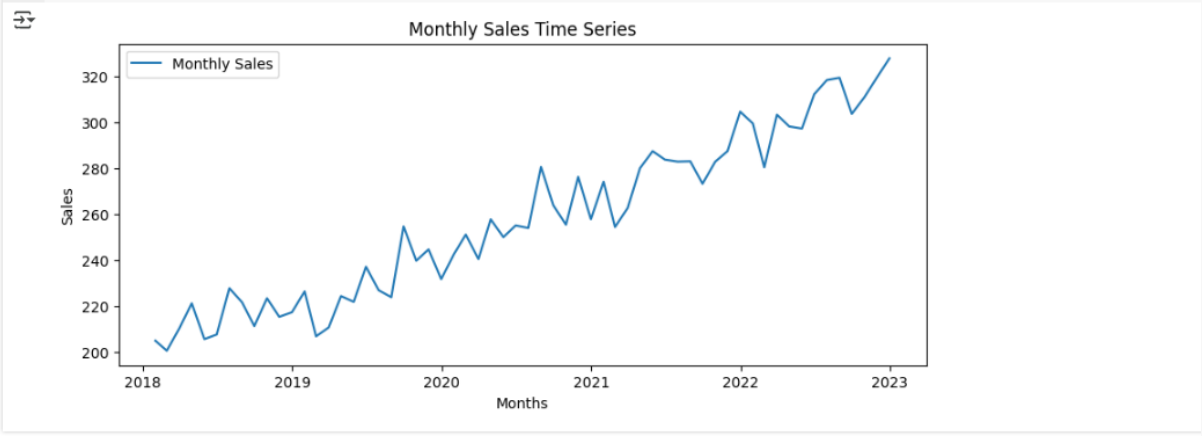
Output:



Code:

```
#visualize the data
plt.figure(figsize=(10,4))
plt.plot(data['Sales'],label='Monthly Sales')
plt.title("Monthly Sales Time Series")
plt.xlabel("Months")
plt.ylabel("Sales")
plt.legend()
plt.show()
```

Output:



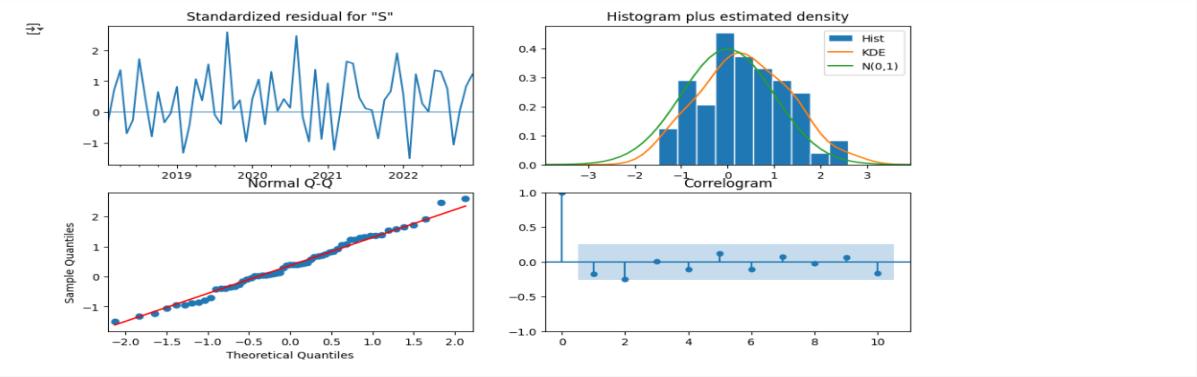
Code:

```
#check stationarity
result=adfuller(data['Sales'])
print("ADF Statistic:",result[0])
print("p-value:",result[1])
if result[1]<=0.05:
  print("Data is Stationary")
else:
  print("Data is Non-Stationary")
model=ARIMA(data['Sales'],order=(1,1,1))
model_fit=model.fit()
print(model_fit.summary())
model_fit.plot_diagnostics(figsize=(12,6))
plt.show()
```

Output:

```
ADF Statistic: 1.2914496798697157
p-value: 0.9965606304178265
Data is Non-Stationary
```

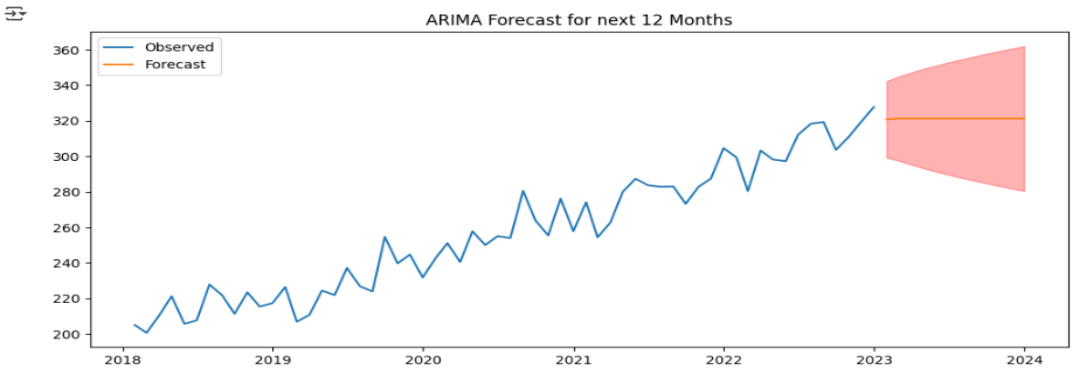
```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                      Sales   No. Observations:                   60
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -225.403
Date:                Wed, 06 Aug 2025   AIC                            456.805
Time:                        18:37:50   BIC                            463.038
Sample:                    01-31-2018   HQIC                           459.238
                         - 12-31-2022
Covariance Type:                  opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
ar.L1         -0.0771      0.293     -0.264      0.792      -0.650       0.496
ma.L1         -0.4760      0.260     -1.828      0.068      -0.986       0.034
sigma2       121.1655     24.103      5.027      0.000      73.924     168.407
==========================================================================================
Ljung-Box (L1) (Q):                   1.83   Jarque-Bera (JB):                 0.67
Prob(Q):                              0.18   Prob(JB):                         0.72
Heteroskedasticity (H):               0.95   Skew:                             0.14
Prob(H) (two-sided):                  0.90   Kurtosis:                         2.56
==========================================================================================

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



Code:

```
forecast = model_fit.get_forecast(steps=12)
forecast_df=forecast.summary_frame()
#plot
plt.figure(figsize=(12,5))
plt.plot(data['Sales'],label='Observed')
plt.plot(forecast_df['mean'],label='Forecast')
plt.fill_between(forecast_df.index,forecast_df['mean_ci_lower'],forecas
t_df['mean_ci_upper'],color='red',alpha=0.3)
plt.title("ARIMA Forecast for next 12 Months")
plt.legend()
plt.show()
```

Output:



#R-studio code

Code:

```
#forecast monthly sales usingg arima model

 install.packages("forecast")

install.packages("tseries")

library(forecast)

library(tseries)

#create or load forecast

sales<-
c(200,220,230,250,270,300,310,320,340,350,370,400,410,420,450,4
70,500,520,540,560,580,600,620,650)

#convert numeric vector to time series

sales_ts<-ts(sales,start=c(2022,1),frequency=12)
```

Output:

```
> #forecast monthly sales usingg arima model
> install.packages("forecast")
Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.5'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
Warning: failed to download mirrors file (cannot open URL 'https://cran.r-project.org/CRAN_mirrors.csv'); using local file 'C:/PROGRA~1/R/R-45~1.1/doc/CRAN_mirrors.csv'
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/forecast_8.24.0.zip'
Content type 'application/zip' length 1920158 bytes (1.8 MB)
downloaded 1.8 MB

package 'forecast' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Admin\AppData\Local\Temp\RtmpQxs6dM\downloaded_packages
Warning message:
In download.file(url, destfile = f, quiet = TRUE) :
  URL 'https://cran.r-project.org/CRAN_mirrors.csv': status was 'SSL connect error'
> install.packages("tseries")
Installing package into 'C:/Users/Admin/AppData/Local/R/win-library/4.5'
(as 'lib' is unspecified)
trying URL 'https://cloud.r-project.org/bin/windows/contrib/4.5/tseries_0.10-58.zip'
Content type 'application/zip' length 387008 bytes (377 KB)
downloaded 377 KB

package 'tseries' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Admin\AppData\Local\Temp\RtmpQxs6dM\downloaded_packages
> library(forecast)
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
> library(tseries)

    'tseries' version: 0.10-58

    'tseries' is a package for time series analysis and computational
    finance.

    See 'library(help="tseries")' for details.

> #create or load forecast
> sales<-c(200,220,230,250,270,300,310,320,340,350,370,400,410,420,450,470,500,520,540,560,580,600,620,650)
> #convert numeric vector to time series
> sales_ts<-ts(sales,start=c(2022,1),frequency=12)
```

Code:

```
#display the first value

print(sales_ts)
```

Output:

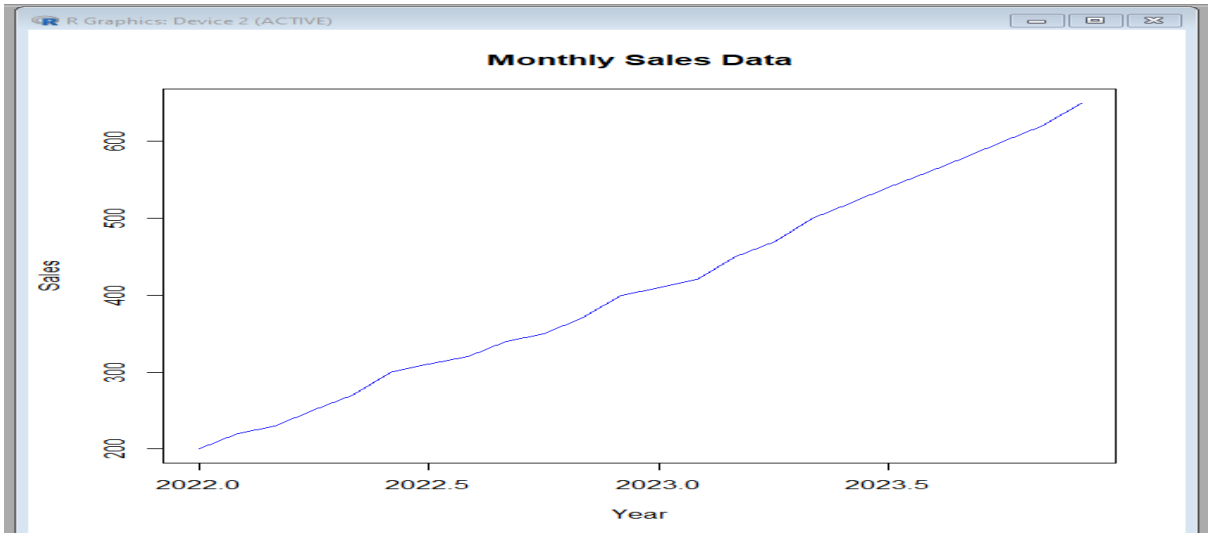
```
> #display the first value
> print(sales_ts)
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2022 200 220 230 250 270 300 310 320 340 350 370 400
2023 410 420 450 470 500 520 540 560 580 600 620 650
```

Code:

#plot the series

plot(sales_ts,main="Monthly Sales Data",xlab="Year",ylab="Sales",col="blue")

Output:



Code:

#adf

adf.test(sales_ts)

Output:

```
> #adf
> adf.test(sales_ts)

        Augmented Dickey-Fuller Test

data:  sales_ts
Dickey-Fuller = -0.72851, Lag order = 2, p-value = 0.956
alternative hypothesis: stationary
```

Code:

```
best_model<-auto.arima(sales_ts)
print(best_model)
```

Output:

```
> best_model<-auto.arima(sales_ts)
> print(best_model)
Series: sales_ts
ARIMA(0,1,0) with drift

Coefficients:
        drift
      19.5652
s.e.   1.4392

sigma^2 = 49.8:  log likelihood = -77.07
AIC=158.13   AICc=158.73   BIC=160.41
```

Code:

```
#forecast for next 12 months
forecast_sales<-forecast(best_model,h=12)
print(forecast_sales)
```

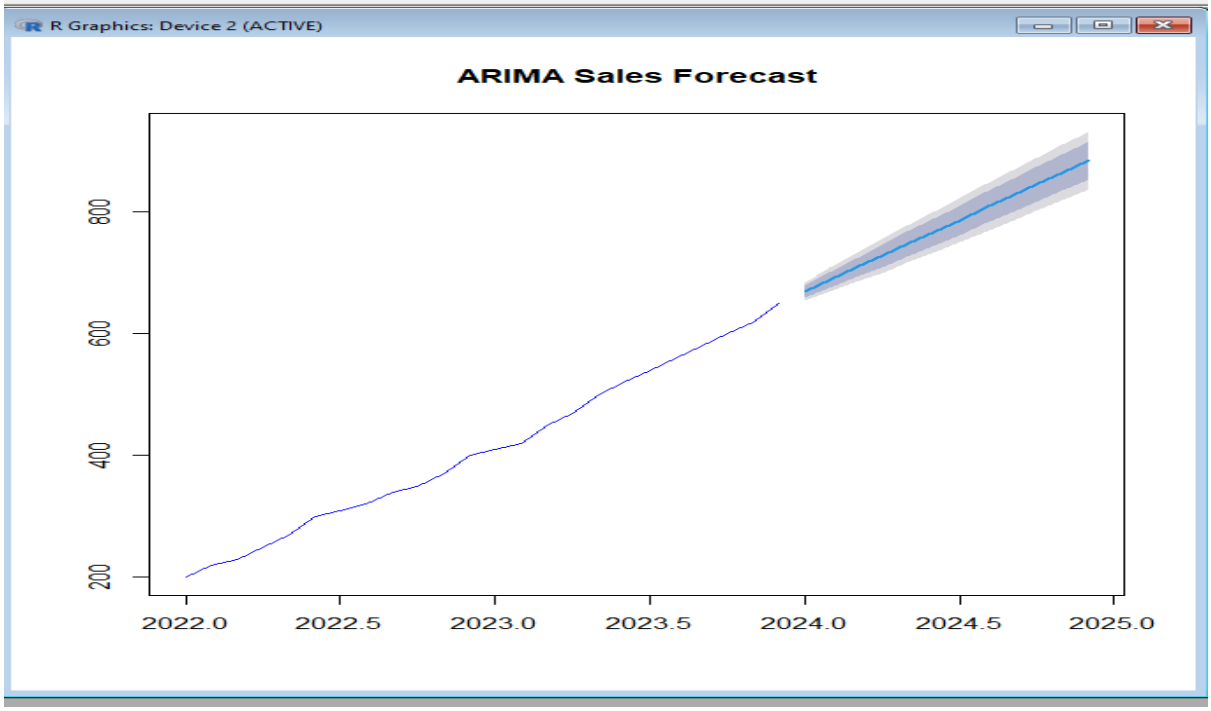
Output:

```
> #forecast for next 12 months
> forecast_sales<-forecast(best_model,h=12)
> print(forecast_sales)
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Jan 2024       669.5652 660.5211 678.6094 655.7334 683.3970
Feb 2024       689.1304 676.3401 701.9208 669.5693 708.6916
Mar 2024       708.6957 693.0307 724.3606 684.7382 732.6531
Apr 2024       728.2609 710.1726 746.3492 700.5972 755.9245
May 2024       747.8261 727.6028 768.0494 716.8972 778.7550
Jun 2024       767.3913 745.2378 789.5448 733.5104 801.2722
Jul 2024       786.9565 763.0280 810.8851 750.3609 823.5521
Aug 2024       806.5217 780.9410 832.1024 767.3994 845.6441
Sep 2024       826.0870 798.9545 853.2194 784.5915 867.5824
Oct 2024       845.6522 817.0521 874.2523 801.9121 889.3923
Nov 2024       865.2174 835.2214 895.2134 819.3424 911.0924
Dec 2024       884.7826 853.4528 916.1124 836.8678 932.6975
```

Code:

```
#plot forecast
plot(forecast_sales,main="ARIMA Sales Forecast",col="blue")
```

Output:



Code:

#Arima on real dataset(Airline Passenger)

library(forecast)

library(tseries)

#load dataset

url<-
https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv

df<-read.csv(url)

head(df)

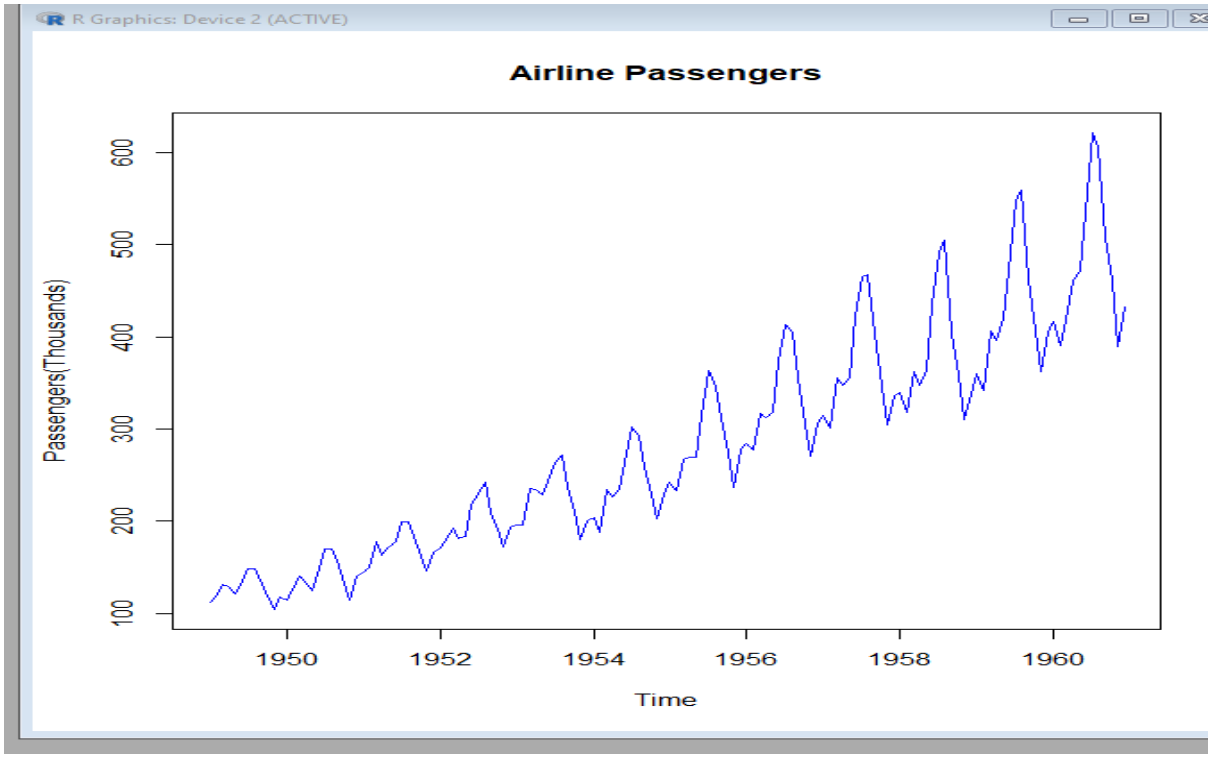
Output:

```
> df<-read.csv(url)
> head(df)
     Month Passengers
1 1949-01         112
2 1949-02         118
3 1949-03         132
4 1949-04         129
5 1949-05         121
6 1949-06         135
>
```

Code:

```
#step 2 convert to time series
 air_ts<-ts(df$Passengers,start=c(1949,1),frequency=12)
 #plot the data
 plot(air_ts,main="Airline
Passengers",ylab="Passengers(Thousands)",col="blue")
```

Output:



Code:

```
#check stationarity
 adf.test(air_ts)
```

Output:

```
> #check stationarity
> adf.test(air_ts)

        Augmented Dickey-Fuller Test

data:  air_ts
Dickey-Fuller = -7.3186, Lag order = 5, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(air_ts) : p-value smaller than printed p-value
```

Code:

#fit arima model

model_air<-auto.arima(air_ts)

summary(model_air)

Output:

```
> #fit arima model
>
> model_air<-auto.arima(air_ts)
> summary(model_air)
Series: air_ts
ARIMA(2,1,1)(0,1,0)[12]

Coefficients:
          ar1      ar2      ma1
      0.5960   0.2143  -0.9819
s.e.  0.0888   0.0880   0.0292

sigma^2 = 132.3:  log likelihood = -504.92
AIC=1017.85   AICc=1018.17   BIC=1029.35

Training set error measures:
                    ME     RMSE      MAE       MPE     MAPE      MASE         ACF1
Training set 1.3423 10.84619 7.86754 0.420698 2.800458 0.245628 -0.00124847
> |
```

Code:

#forecast next 24 months

forecast_air<-forecast(model_air,h=24)

print(forecast_air)

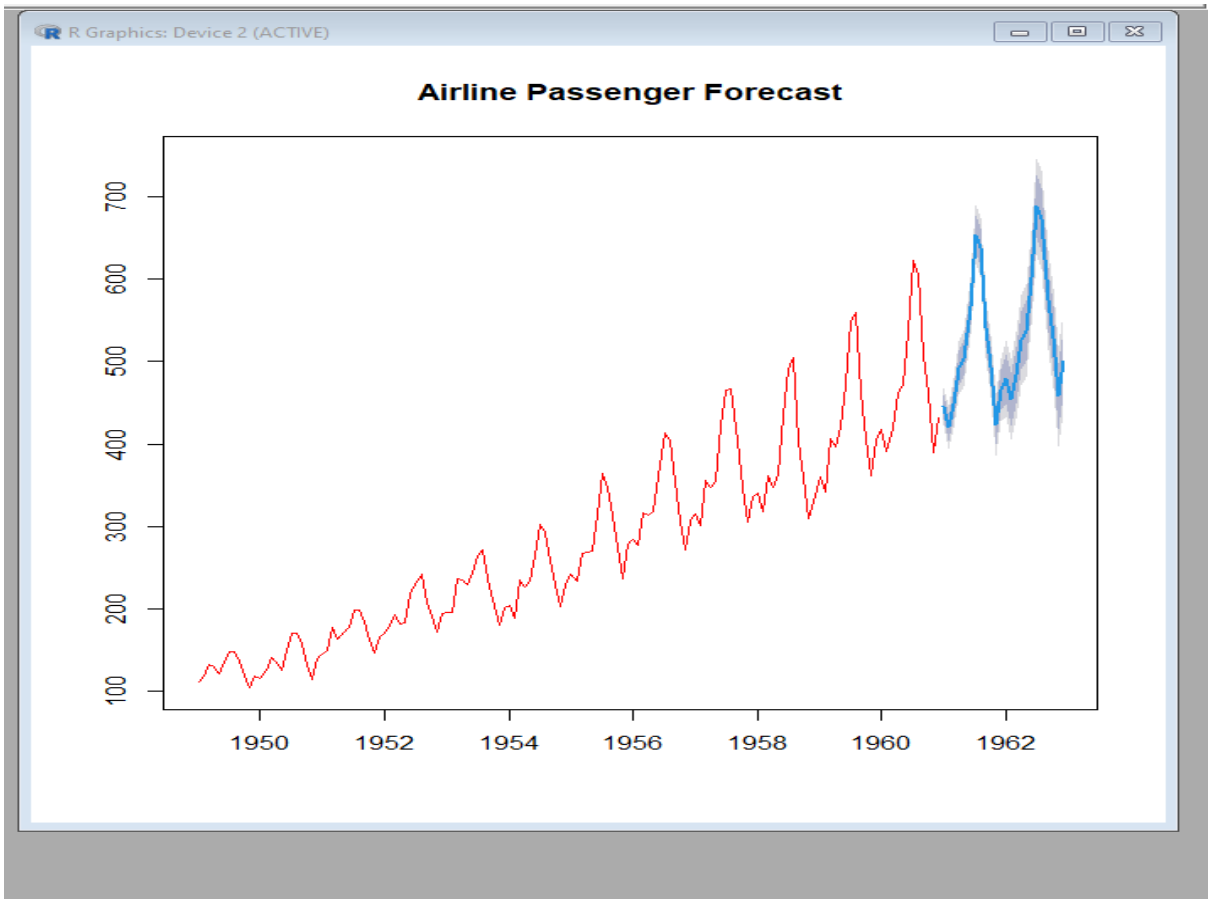
Output:

```
> #forecast next 24 months
> forecast_air<-forecast(model_air,h=24)
> print(forecast_air)
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Jan 1961       445.6349 430.8903 460.3795 423.0851 468.1847
Feb 1961       420.3950 403.0907 437.6993 393.9304 446.8596
Mar 1961       449.1983 429.7726 468.6240 419.4892 478.9074
Apr 1961       491.8399 471.0270 512.6529 460.0092 523.6707
May 1961       503.3945 481.5559 525.2330 469.9953 536.7937
Jun 1961       566.8624 544.2637 589.4612 532.3007 601.4242
Jul 1961       654.2602 631.0820 677.4383 618.8122 689.7081
Aug 1961       638.5975 614.9704 662.2246 602.4630 674.7320
Sep 1961       540.8837 516.9028 564.8647 504.2081 577.5594
Oct 1961       494.1266 469.8624 518.3909 457.0177 531.2356
Nov 1961       423.3327 398.8381 447.8273 385.8715 460.7939
Dec 1961       465.5076 440.8229 490.1923 427.7556 503.2596
Jan 1962       479.2908 448.9986 509.5831 432.9629 525.6188
Feb 1962       454.1768 421.7184 486.6353 404.5359 503.8178
Mar 1962       483.0870 448.7343 517.4396 430.5491 535.6248
Apr 1962       525.8193 490.1122 561.5263 471.2101 580.4284
May 1962       537.4507 500.6863 574.2151 481.2244 593.6770
Jun 1962       600.9839 563.3924 638.5754 543.4927 658.4752
Jul 1962       688.4370 650.1834 726.6907 629.9331 746.9410
Aug 1962       672.8213 634.0292 711.6134 613.4940 732.1487
Sep 1962       575.1475 535.9102 614.3847 515.1393 635.1557
Oct 1962       528.4242 488.8131 568.0352 467.8443 589.0040
Nov 1962       457.6590 417.7293 497.5886 396.5918 518.7261
Dec 1962       499.8582 459.6529 540.0634 438.3695 561.3468
```

Code:

```
plot(forecast_air,main="Airline Passenger Forecast",col="red")
```

Output:



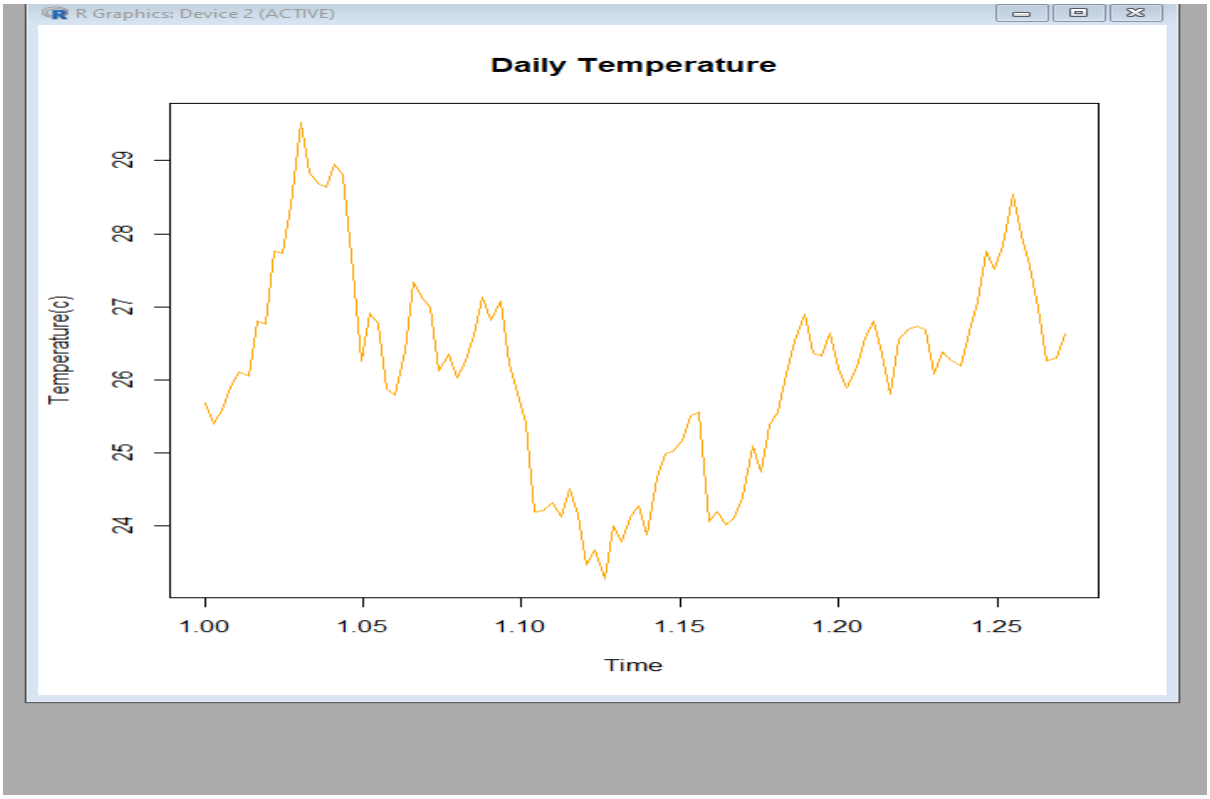
Code:

```
#example 3
 #Arima on daily temperature(synthetic data)
 #create dataset
 set.seed(42)
 days<-seq(as.Date("2022-01-01"),by="day",length.out=100)
 temperature<-25+cumsum(rnorm(100,0,0.5))#random noise
 #create time series
 temp_ts<-ts(temperature,frequency=365)
```

#plot

plot(temp_ts,main="Daily Temperature",ylab="Temperature(c)",col="orange")

Output:



Code:

#fit arima model

model_temp<-auto.arima(temp_ts)

summary(model_temp)

Output:

```
> model_temp<-auto.arima(temp_ts)
> summary(model_temp)
Series: temp_ts
ARIMA(0,1,0)

sigma^2 = 0.2666:  log likelihood = -75.04
AIC=152.08   AICc=152.12   BIC=154.68

Training set error measures:
                     ME      RMSE       MAE         MPE     MAPE MASE      ACF1
Training set 0.00965947 0.5137773 0.4064194 0.01774295 1.554414  NaN 0.0641011
```

Code:

```
#forecast for next 30 days
 forecast_temp<-forecast(model_temp,h=30)
 print(forecast_temp)
```

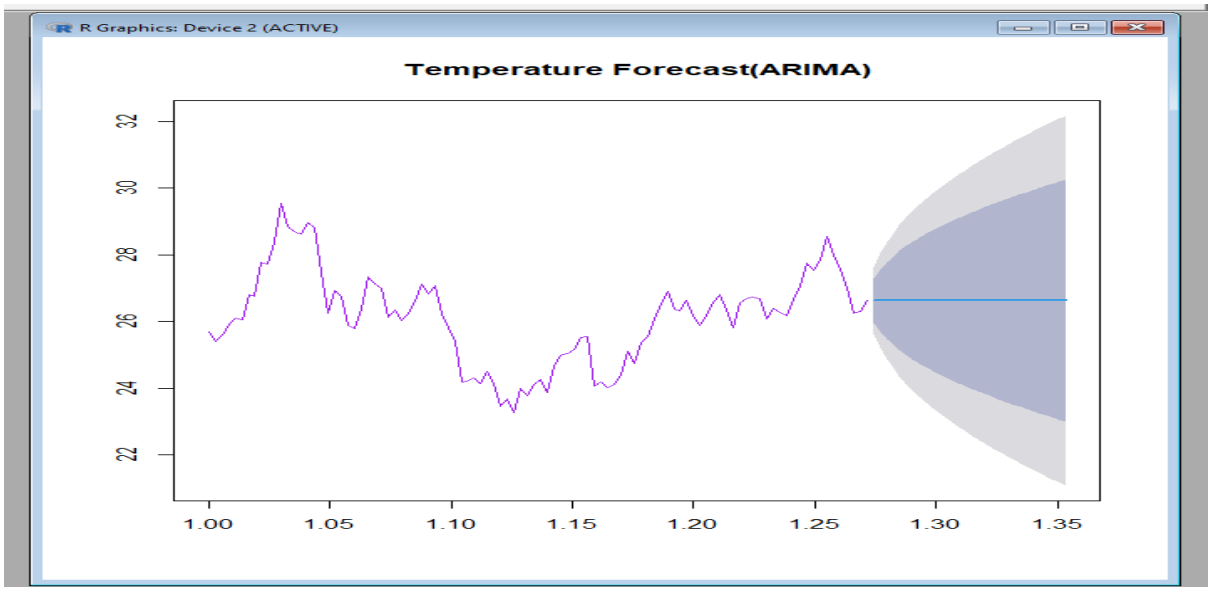
Output:

```
> #forecast for next 30 days
> forecast_temp<-forecast(model_temp,h=30)
> print(forecast_temp)
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
1.273973       26.62574 25.96399 27.28749 25.61368 27.63780
1.276712       26.62574 25.68989 27.56160 25.19447 28.05701
1.279452       26.62574 25.47956 27.77192 24.87280 28.37868
1.282192       26.62574 25.30224 27.94924 24.60162 28.64986
1.284932       26.62574 25.14602 28.10546 24.36271 28.88877
1.287671       26.62574 25.00479 28.24669 24.14672 29.10477
1.290411       26.62574 24.87492 28.37656 23.94809 29.30339
1.293151       26.62574 24.75403 28.49745 23.76321 29.48827
1.295890       26.62574 24.64049 28.61099 23.58957 29.66191
1.298630       26.62574 24.53311 28.71838 23.42533 29.82615
1.301370       26.62574 24.43097 28.82051 23.26912 29.98236
1.304110       26.62574 24.33337 28.91811 23.11987 30.13161
1.306849       26.62574 24.23977 29.01171 22.97671 30.27477
1.309589       26.62574 24.14970 29.10178 22.83897 30.41252
1.312329       26.62574 24.06280 29.18868 22.70606 30.54542
1.315068       26.62574 23.97874 29.27274 22.57751 30.67397
1.317808       26.62574 23.89728 29.35420 22.45292 30.79856
1.320548       26.62574 23.81818 29.43330 22.33194 30.91954
1.323288       26.62574 23.74124 29.51024 22.21428 31.03720
1.326027       26.62574 23.66631 29.58517 22.09968 31.15180
1.328767       26.62574 23.59323 29.65826 21.98791 31.26357
1.331507       26.62574 23.52186 29.72962 21.87877 31.37271
1.334247       26.62574 23.45210 29.79938 21.77208 31.47940
1.336986       26.62574 23.38385 29.86764 21.66769 31.58379
1.339726       26.62574 23.31700 29.93449 21.56545 31.68603
1.342466       26.62574 23.25147 30.00001 21.46524 31.78624
1.345205       26.62574 23.18719 30.06429 21.36693 31.88455
1.347945       26.62574 23.12409 30.12739 21.27043 31.98105
1.350685       26.62574 23.06211 30.18937 21.17564 32.07584
1.353425       26.62574 23.00119 30.25029 21.08247 32.16901
```

Code:

```
#plot
plot(forecast_temp,main="Temperature
Forecast(ARIMA)",col="purple")
```

Output:



PRACTICAL N0 –8

Aim: Using Sarima for forecasting

Code:

```
#SARIMA with clear seasonal pattern and upward trenf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
#generate data
#Generate data
np.random.seed(0)
date_range=pd.date_range(start='2020-01-01',periods=48,freq='M')
trend=np.linspace(100,200,48)
seasonality=20*np.sin(2*np.pi*date_range.month/12)
noise=np.random.normal(0,5,48)
data=trend+seasonality+noise

df=pd.DataFrame({'Sales':data},index=date_range)
df
```

Output:

```
seasonality=20*np.sin(2*np.pi*date_range.month/12)
noise=np.random.normal(0,5,48)
data=trend+seasonality+noise

df=pd.DataFrame({'Sales':data},index=date_range)
df
```

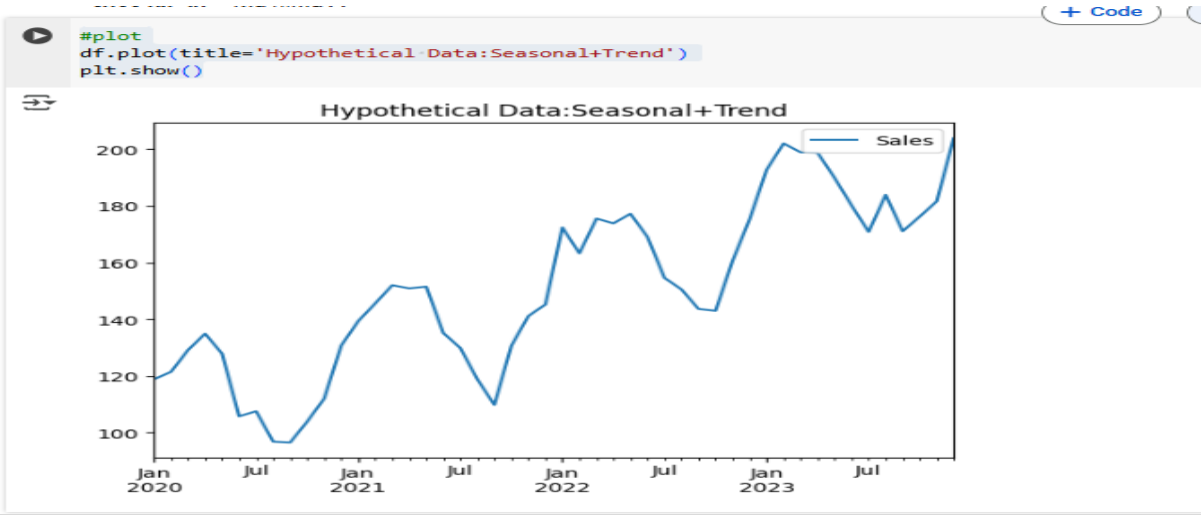
```
/tmp/ipython-input-955236531.py:4: FutureWarning: 'M' is deprecated and will be re
  date_range=pd.date_range(start='2020-01-01',periods=48,freq='M')
```

	Sales
2020-01-31	118.820262
2020-02-29	121.448954
2020-03-31	129.149009
2020-04-30	134.907953
2020-05-31	127.848428

Code:

```
#plot
df.plot(title='Hypothetical Data:Seasonal+Trend')
plt.show()
```

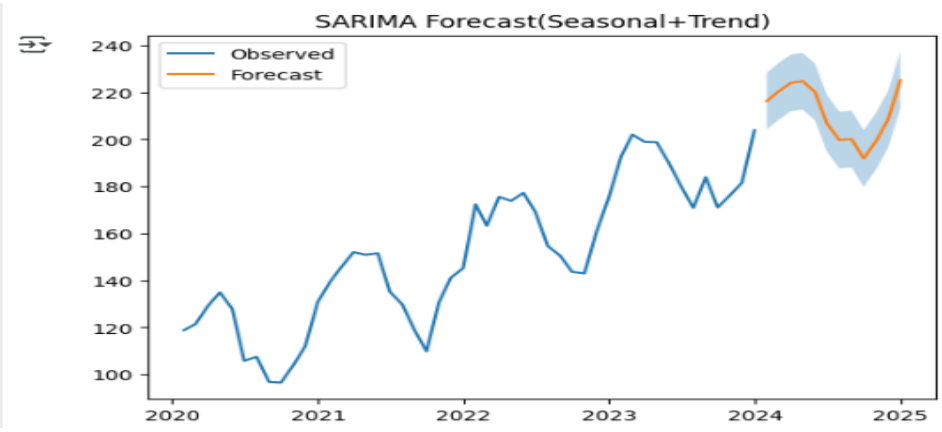
Output:



Code:

```
#fit sarima
model=SARIMAX(df['Sales'],order=(1,1,1),seasonal_order=(1,1,1,12))
result=model.fit()
#forecast
forecast=result.get_forecast(steps=12)
forecast_df=forecast.summary_frame()
plt.plot(df,label='Observed')
plt.plot(forecast_df['mean'],label='Forecast')
plt.fill_between(forecast_df.index,forecast_df['mean_ci_lower'],forecas
t_df['mean_ci_upper'],alpha=0.3)
plt.title('SARIMA Forecast(Seasonal+Trend)')
plt.legend()
plt.show()
```

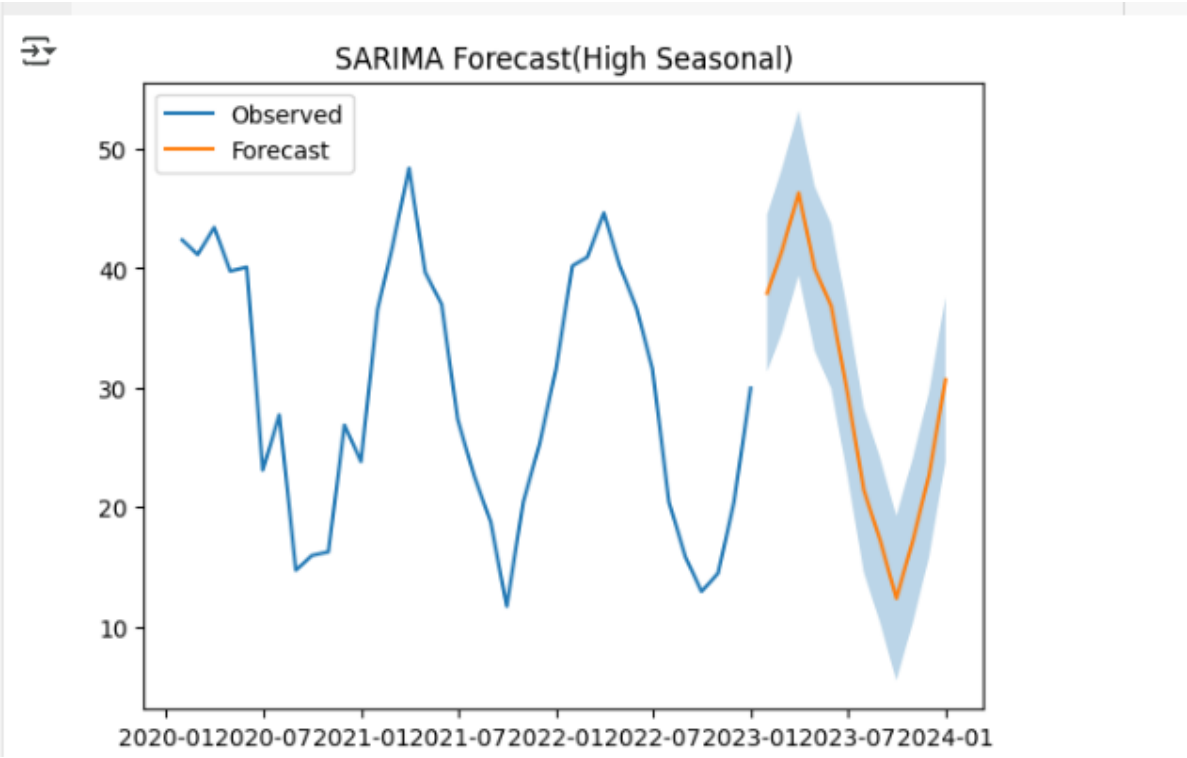
Output:



Code:

```
#example2 with high seasonality trend sysnthic data no trend
np.random.seed(1)
dates=pd.date_range(start='2020-01-01',periods=36,freq='M')
seasonality=30+15*np.sin(2*np.pi*dates.month/12)
noise=np.random.normal(0,3,36)
values=seasonality+noise
df=pd.DataFrame({'Values':values},index=dates)
df
#fit Sarima
model=SARIMAX(df['Values'],order=(1,0,0),seasonal_order=(1,1,0,12))
res=model.fit()
#forecast
fcast=res.get_forecast(steps=12)
fcast_df=fcast.summary_frame()
plt.plot(df,label='Observed')
plt.plot(fcast_df['mean'],label='Forecast')
plt.fill_between(fcast_df.index,fcast_df['mean_ci_lower'],fcast_df['mea
n_ci_upper'],alpha=0.3)
plt.title('SARIMA Forecast(High Seasonal)')
plt.legend()
plt.show()
```

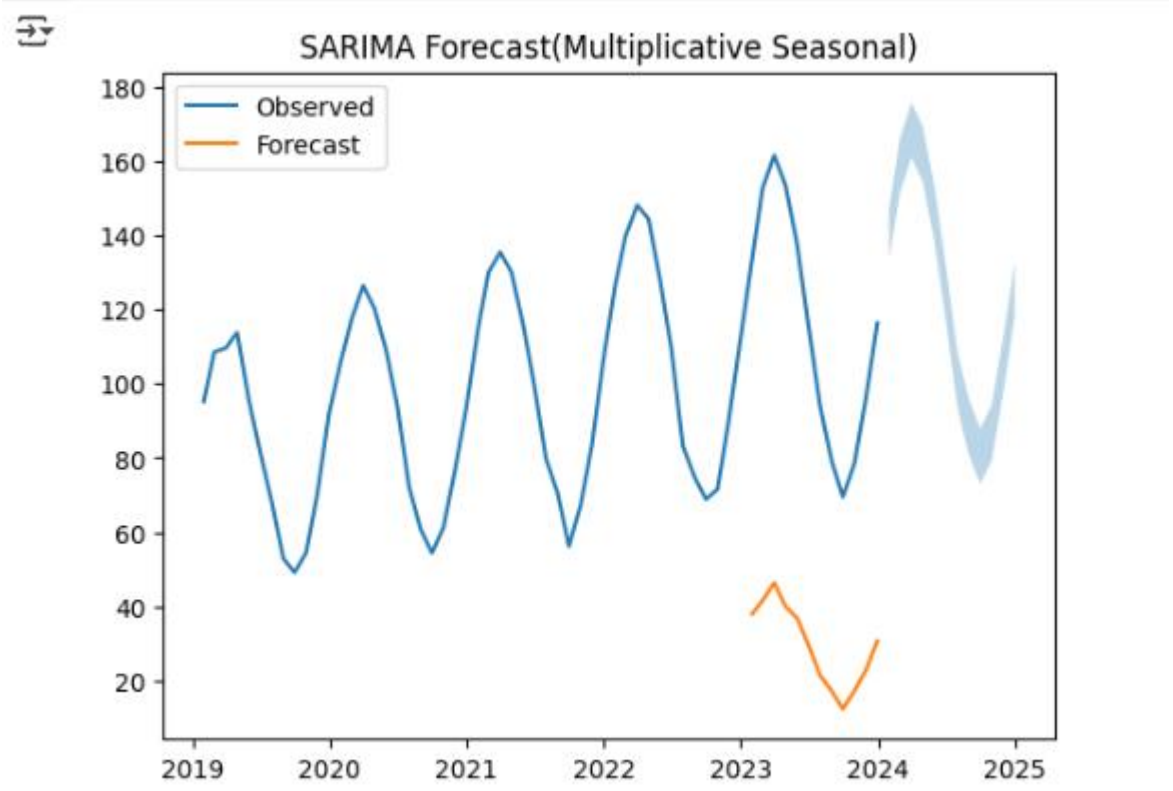
Output:



Code:

```
#example 3 Sarima with multiplicative seasonality and weak trend
np.random.seed(2)
months=pd.date_range(start='2019-01',periods=60,freq='M')
trend=np.linspace(80,120,60)
seasonal_factor=1+0.4*np.sin(2*np.pi*months.month/12)
noise=np.random.normal(0,2,60)
values=trend*seasonal_factor+noise
df=pd.DataFrame({'Value':values},index=months)
#fit sarima
model=SARIMAX(df['Value'],order=(1,1,1),seasonal_order=(1,1,1,12))
res=model.fit()
#forecast
pred=res.get_forecast(steps=12)
pred_df=pred.summary_frame()
#plot
plt.plot(df,label='Observed')
plt.plot(fcast_df['mean'],label='Forecast')
plt.fill_between(pred_df.index,pred_df['mean_ci_lower'],pred_df['mean_c
i_upper'],alpha=0.3)
plt.title('SARIMA Forecast(Multiplicative Seasonal)')
plt.legend()
plt.show()
```

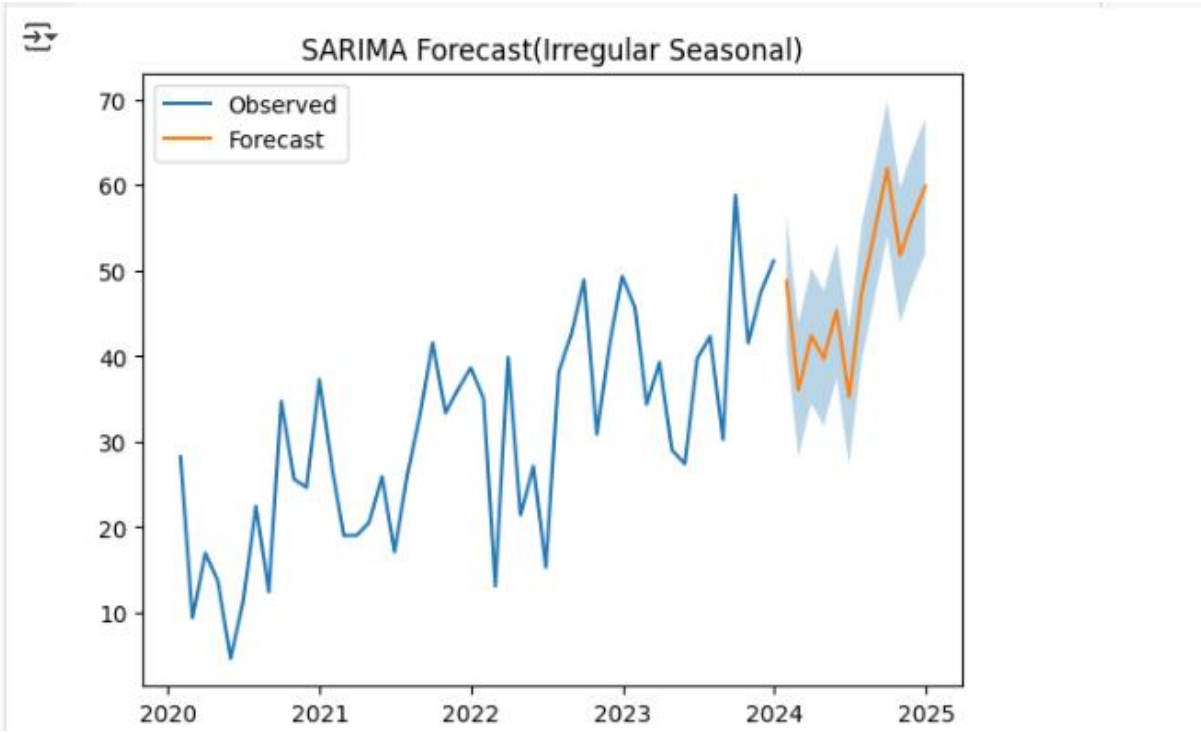
Output:



Code:

```
#example 4  SARIMA with irregular Seasonal Pattern
np.random.seed(3)
months=pd.date_range(start='2020-01',periods=48,freq='M')
pattern=np.random.choice([10,20,30],size=12,replace=True)
seasonality=np.tile(pattern,4)
trend=0.5*np.arange(48)
noise=np.random.normal(0,5,48)
values=trend+seasonality+noise
df=pd.DataFrame({'Value':values},index=months)
#fit sarima
model=SARIMAX(df['Value'],order=(1,1,1),seasonal_order=(2,1,0,12))
res=model.fit()
#forecast
future=res.get_forecast(steps=12)
future_df=future.summary_frame()
plt.plot(df,label='Observed')
plt.plot(future_df['mean'],label='Forecast')
plt.fill_between(future_df.index,future_df['mean_ci_lower'],future_df['
mean_ci_upper'],alpha=0.3)
plt.title('SARIMA Forecast(Irregular Seasonal)')
plt.legend()
plt.show()
```

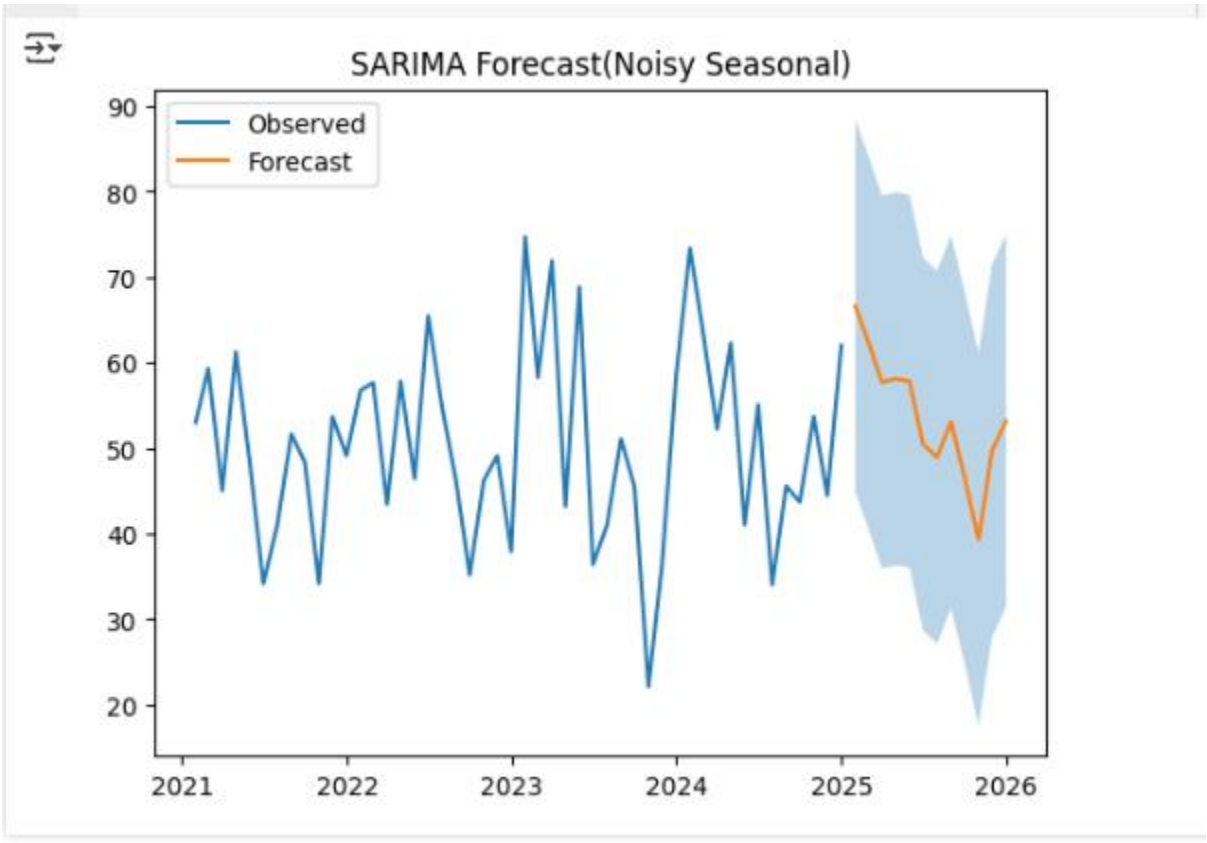
Output:



Code:

```
#example 5: Sarima with weak seasonality and strong noise
np.random.seed(4)
months=pd.date_range(start='2021-01',periods=48,freq='M')
seasonal=5*np.sin(2*np.pi*months.month/12)
noise=np.random.normal(0,10,48)
values=50+seasonal+noise
df=pd.DataFrame({'Value':values},index=months)

#fit sarima
model=SARIMAX(df['Value'],order=(0,1,1),seasonal_order=(1,1,1,12))
res=model.fit()
#forecast
forecast=res.get_forecast(steps=12)
forecast_df=forecast.summary_frame()
plt.plot(df,label='Observed')
plt.plot(forecast_df['mean'],label='Forecast')
plt.fill_between(forecast_df.index,forecast_df['mean_ci_lower'],forecas
t_df['mean_ci_upper'],alpha=0.3)
plt.title('SARIMA Forecast(Noisy Seasonal)')
plt.legend()
plt.show()
```

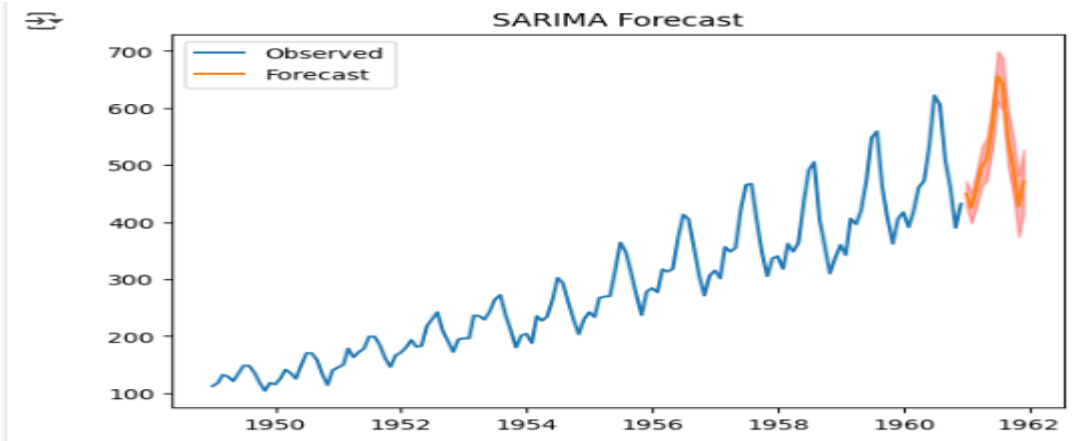
Output:



Code:

```
#example 6
#SARIMA on airline passengers dataset(seasonal monthly data)
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
#from pmdarima import auto_arima
from statsmodels.tsa.stattools import adfuller
#load dataset
url='https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv'
data=pd.read_csv(url,parse_dates=['Month'],index_col='Month')
data.columns=['Passengers']
#plot
data.plot(title='Airline Passengers')
plt.show()
#adf
result =adfuller(data['Passengers'])
print("ADF p-value:",result[1])
#fit Sarima
sarima=SARIMAX(data['Passengers'],order=(1,1,1),seasonal_order=(1,1,1,12))
sarima_fit=sarima.fit()
print(sarima_fit.summary())
#forecast
forecast=sarima_fit.get_forecast(steps=12)
forecast_df=forecast.summary_frame()
plt.plot(data,label='Observed')
plt.plot(forecast_df['mean'],label='Forecast')
plt.fill_between(forecast_df.index,forecast_df['mean_ci_lower'],forecast_df['mean_ci_upper'],color='r',alpha=0.3)
plt.title('SARIMA Forecast')
plt.legend()
plt.show()
```

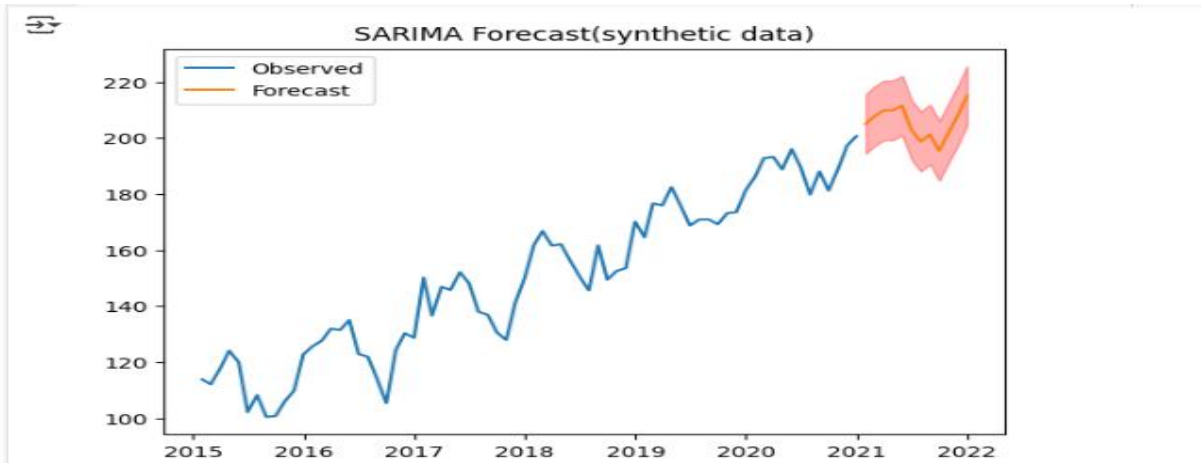
Output:



Code:

```
#example 7 Sarima on synthetic Seasonal sales data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
#create synthetic data
np.random.seed(0)
months=pd.date_range(start='2015-01',periods=72,freq='M')
seasonal=10*np.sin(2*np.pi*months.month/12)
trend=np.linspace(100,200,72)
noise=np.random.normal(0,5,72)
sales=trend+seasonal+noise
df=pd.DataFrame({'Sales':sales},index=months)
df
#plot
df.plot(title='Synthetic Seasonal Data')
plt.show()
#sarima model
model=SARIMAX(df['Sales'],order=(1,1,1),seasonal_order=(1,1,1,12))
result=model.fit()
#Forecast
#forecast
forecast=result.get_forecast(steps=12)
forecast_df=forecast.summary_frame()

#plot
plt.plot(df,label='Observed')
plt.plot(forecast_df['mean'],label='Forecast')
plt.fill_between(forecast_df.index,forecast_df['mean_ci_lower'],forecas
t_df['mean_ci_upper'],color='r',alpha=0.3)
plt.title('SARIMA Forecast(synthetic data)')
plt.legend()
plt.show()
```

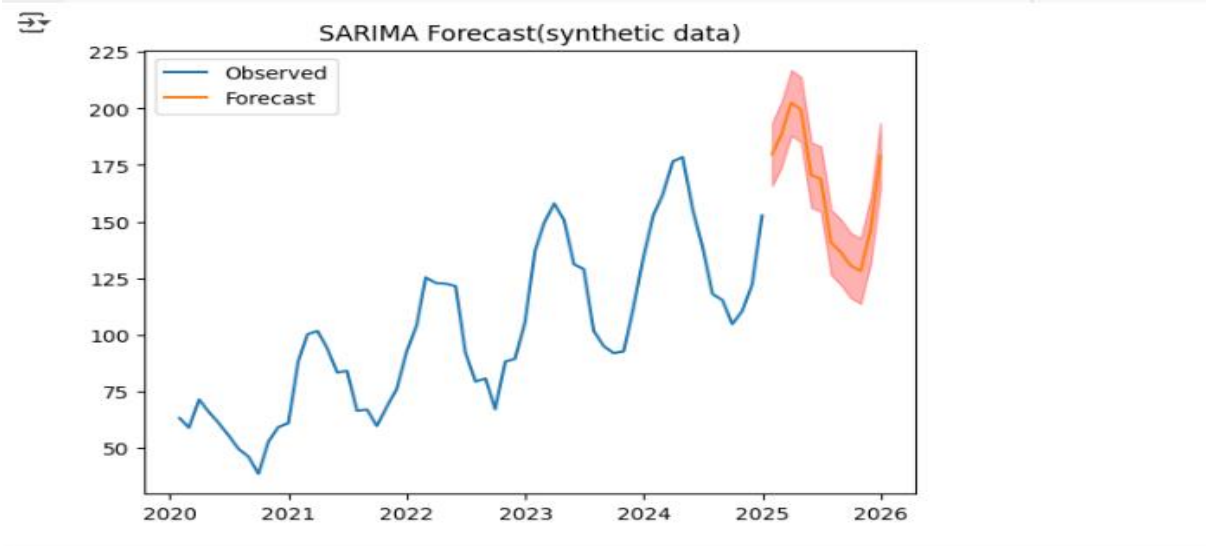
Output:



Code:

```
#sarima with multiplicative Seasonal component synthetic
#multiplicative seasonality
months=pd.date_range(start='2020-01',periods=60,freq='M')
seasonal=1+0.3*np.sin(2*np.pi*months.month/12)
trend=np.linspace(50,150,60)
noise=np.random.normal(0,5,60)
sales=trend*seasonal+noise
df=pd.DataFrame({'Sales':sales},index=months)
df

#plot
df.plot(title='Synthetic  Multiplicative Seasonal Data')
plt.show()
#sarima model
model=SARIMAX(df['Sales'],order=(1,1,1),seasonal_order=(1,1,1,12))
result=model.fit()
#forecast
forecast=result.get_forecast(steps=12)
forecast_df=forecast.summary_frame()
#plot
plt.plot(df,label='Observed')

plt.plot(forecast_df['mean'],label='Forecast')
plt.fill_between(forecast_df.index,forecast_df['mean_ci_lower'],forecas
t_df['mean_ci_upper'],color='r',alpha=0.3)
plt.title('SARIMA Forecast(synthetic data)')
plt.legend()
plt.show()
```

Output:



PRACTICAL N0 –9

Aim: Aic, Bic & Fpe

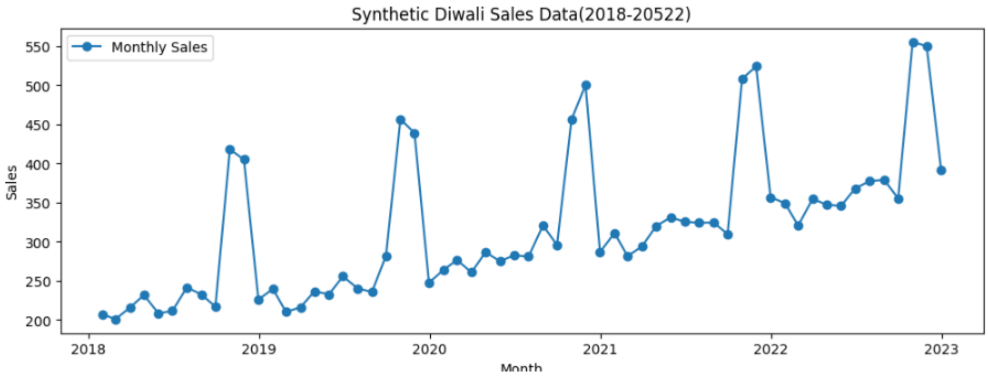
Code:

```
from inspect import modulesbyfile
#example diwali sales aic,bic,fpe
#step1:generate dataset6
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.random.seed(42)
months=pd.date_range(start='2018-01-01',periods=60,freq='M')
#base trend=noise
sales=200+np.arange(60)*3+np.random.normal(0,15,60)
#add diwali effect:higher sales in oct-nov
for i,m in enumerate(months):
  if m.month in [10,11]:
    sales[i]+=100+np.random.randint(50,100)
data=pd.DataFrame({'Month':months,'Sales':sales})
data.set_index('Month',inplace=True)

#plot dataset
plt.figure(figsize=(12,4))
plt.plot(data['Sales'],marker='o',label="Monthly Sales")
plt.title("Synthetic Diwali Sales Data(2018-20522)")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.legend()
plt.show()

#unity function for aic, bic,fpe
from statsmodels.tsa.arima.model import ARIMA
def evaluate_arima(series,order):
  model=ARIMA(series,order=order)
  fit=model.fit()
  n_obs=fit.nobs
  sigma2=fit.mse
  k=fit.df_model+1
  fpe=sigma2*(n_obs+k)/(n_obs-k)
  return{"Order":order,"AIC":fit.aic,"BIC":fit.bic,"FPE":fpe}

#case1: ARIMA(1,1,1)
result1=evaluate_arima(data['Sales'],(1,1,1))
print("case1:",result1)

#case2: ARIMA(2,1,2)
result2=evaluate_arima(data['Sales'],(2,1,2))
```

```
print("case2:",result2)

#case3: ARIMA(3,1,2)
result3=evaluate_arima(data['Sales'],(3,1,2))
print("case3:",result3)

#case4: ARIMA(2,0,2)
result4=evaluate_arima(data['Sales'],(2,0,2))
print("case4:",result4)

#case5: ARIMA(1,1,2)
result5=evaluate_arima(data['Sales'],(1,1,2))
print("case5:",result5)

#compare all results
results=pd.DataFrame([result1,result2,result3,result4,result5])

print("\nComparsion of Model (AIC,BIC,FPE):")
print(results.sort_values(by="AIC"))
```

Output:

```
/tmp/ipython-input-3378351328.py:8: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  months=pd.date_range(start='2018-01-01',periods=60,freq='M')
```




```
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency ME will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency ME will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency ME will be used.
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency ME will be used.
  self._init_dates(dates, freq)
case1: {'Order': (1, 1, 1), 'AIC': np.float64(675.7912864544597), 'BIC': np.float64(682.0238987861769), 'FPE': np.float64(6373.020392061551)}
case2: {'Order': (2, 1, 2), 'AIC': np.float64(662.0576160724877), 'BIC': np.float64(672.4453032920163), 'FPE': np.float64(5194.986934089465)}
case3: {'Order': (3, 1, 2), 'AIC': np.float64(664.0541013405347), 'BIC': np.float64(676.519326003969), 'FPE': np.float64(5373.086869940933)}
case4: {'Order': (2, 0, 2), 'AIC': np.float64(675.1588481844228), 'BIC': np.float64(687.7249155577554), 'FPE': np.float64(4679.675364491733)}
case5: {'Order': (1, 1, 2), 'AIC': np.float64(660.066182924016), 'BIC': np.float64(668.3763326996389), 'FPE': np.float64(5021.906342064552)}

Comparsion of Model (AIC,BIC,FPE):
       Order         AIC         BIC          FPE
4  (1, 1, 2)  660.066183  668.376333  5021.906342
1  (2, 1, 2)  662.057616  672.445303  5194.986934
2  (3, 1, 2)  664.054101  676.519326  5373.086870
3  (2, 0, 2)  675.158848  687.724916  4679.675364
0  (1, 1, 1)  675.791286  682.023899  6373.020392
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency ME will be used.
  self._init_dates(dates, freq)
```

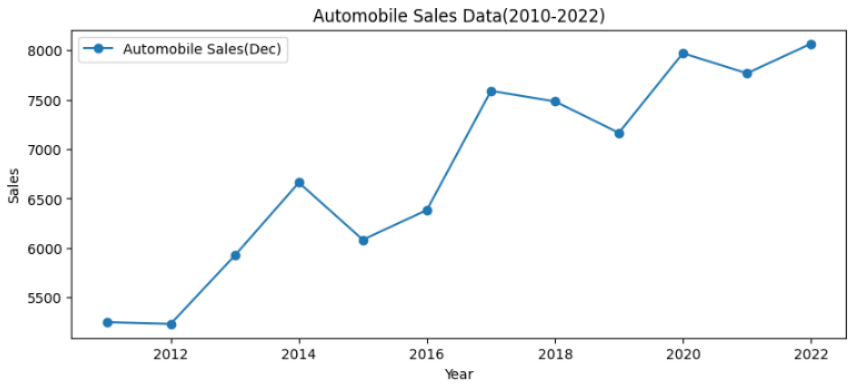
Code:

```
#example2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.random.seed(42)
#years from 2010 to 2022
years=pd.date_range(start='2010-12-01',end='2022-12-01',freq="12M")
#simulated automobile sales with upward trend+noise
sales=5000+(np.arange(len(years))*300)+np.random.normal(0,500,len(years
))
data=pd.DataFrame ({"Year":years,"Auto_Sales":sales})
data.set_index("Year",inplace=True)
#plot the dataset
plt.figure(figsize=(10,4))
plt.plot(data.index,data["Auto_Sales"],marker='o',label="Automobile
Sales(Dec)")
plt.title("Automobile Sales Data(2010-2022)")
plt.xlabel("Year")
plt.ylabel("Sales")
plt.legend()
plt.show()

#define function
from statsmodels.tsa.arima.model import ARIMA
def evaluate_arima(series,order):
  model=ARIMA(series,order=order)
  fit=model.fit()
  n_obs=fit.nobs
  sigma2=fit.mse
  k=fit.df_model+1
  fpe=sigma2*(n_obs+k)/(n_obs-k)
  return{"Order":order,"AIC":fit.aic,"BIC":fit.bic,"FPE":fpe}
#case1: ARIMA(1,1,1)
result1=evaluate_arima(data['Auto_Sales'],(1,1,1))
print("case1:",result1)

#case2: ARIMA(2,1,1)
result2=evaluate_arima(data['Auto_Sales'],(2,1,1))
print("case2:",result2)

#case3: ARIMA(2,1,2)
result3=evaluate_arima(data['Auto_Sales'],(2,1,2))
print("case3:",result3)

#case4: ARIMA(3,1,2)
result4=evaluate_arima(data['Auto_Sales'],(3,1,2))
print("case4:",result4)
```

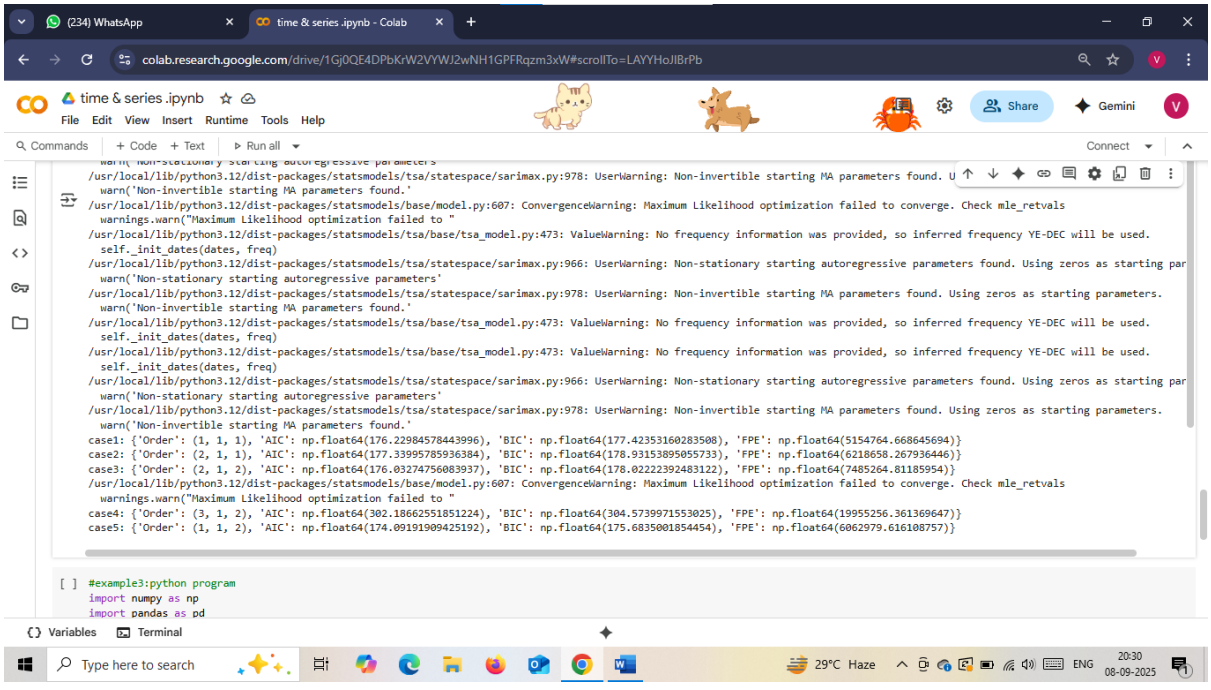
```
#case5: ARIMA(1,1,2)
result5=evaluate_arima(data['Auto_Sales'],(1,1,2))
print("case5:",result5)
#compare all results
results=pd.DataFrame([result1,result2,result3,result4,result5])

print("\nComparsion of Model (AIC,BIC,FPE):")
print(results.sort_values(by="AIC"))
```

Output:

```
/tmp/ipython-input-2600439064.py:7: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  years=pd.date_range(start='2010-12-01',end='2022-12-01',freq="12M")
```




```
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred freque
```



Code:

```
#example3:python program
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
#synthetic data
np.random.seed(0)
n = 80
data=pd.Series(50+0.5*np.arange(n)+np.random.normal(0,2,n))
#fit arima
model=ARIMA(data,order=(2,1,2))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)
print("AIC",fit.aic)
print("BIC",fit.bic)
print("FPE",fpe)
```

Output:

```
print("BIC",fit.bic)
print("FPE",fpe)
```

```
AIC 370.58181675910527
BIC 382.4290560214404
FPE 48.06658289120959
```

Code:

```
#Synthetic time series
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
#generate synthetic trend data
np.random.seed(42)
n = 80
series = pd.Series(100+0.5*np.arange(n)+np.random.normal(0,2,n))
#fit arima(2,1,2)
model=ARIMA(series,order=(2,1,2))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)
```

```
print("Synthetic Example")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)
```

Output:

```
Synthetic Example
AIC: 368.4220658620451
BIC: 380.2693051243802
FPE: 154.42050581263337
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: U
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: U
  warn('Non-invertible starting MA parameters found.'
```

Code:

```
import statsmodels.api as sm

data=pd.read_csv('AirPassengers.csv')
data['Month']=pd.to_datetime(data['Month'])
data.set_index('Month',inplace=True)

#fit arima
model=ARIMA(data['#Passengers'],order=(2,1,1))
fit=model.fit()

#compute fpe
n_obs=fit.nobs

print("\nAirline Passengers Example")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)
```

Output:

```

Airline Passengers Example
AIC: 1378.3383195987703
BIC: 1390.18969811981
FPE: 154.42050581263337
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py
  self._init_dates(dates, freq)
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/base/tsa_model.py
  self._init_dates(dates, freq)
```

Code:

```
#Alternate method for mutliple orders
orders=[(1,1,1),(2,1,1),(3,1,2)]
results=[]

for order in orders:
    model=ARIMA(data['#Passengers'],order=order)
    fit=model.fit()
    n_obs=fit.nobs
    sigma2=fit.mse
    k=fit.df_model+1
    fpe=sigma2*(n_obs+k)/(n_obs-k)
    results.append((order,fit.aic,fit.bic,fpe))

import pandas as pd
df=pd.DataFrame(results,columns=["Order(p,q,d)","AIC","BIC","FPE"])
print(df)
```

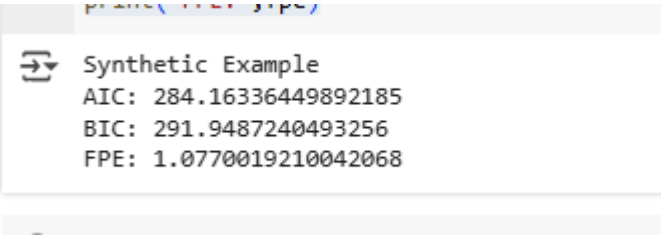
Output:

```
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/statespace/sari
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.12/dist-packages/statsmodels/tsa/statespace/sari
  warn('Non-invertible starting MA parameters found.'
  Order(p,q,d)          AIC          BIC          FPE
0    (1, 1, 1)  1394.682505  1403.571039  1102.526488
1    (2, 1, 1)  1378.338320  1390.189698   992.972382
2    (3, 1, 2)  1377.086307  1394.863374   986.024100
```

Code:

```
#example 4
#generate data
np.random.seed(0)
n=100
steps=np.random.normal(0,1,n)
series=pd.Series(np.cumsum(steps))
#fit arima
model=ARIMA(series,order=(1,1,1))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)
print("Synthetic Example")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)
```

Output:

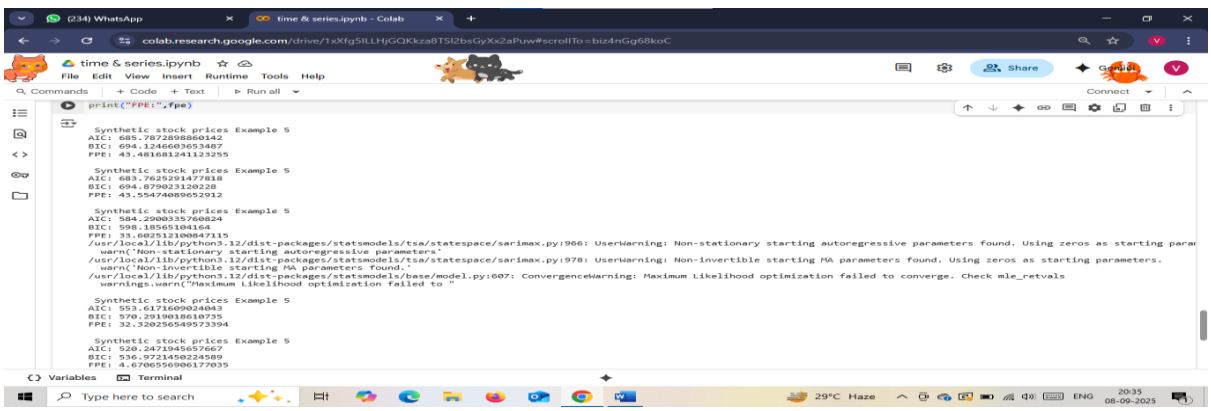


Code:

```
#Example 5

np.random.seed(10)
n=120
time=np.arange(n)
series=50+10*np.sin(2*np.pi*time/12)+np.random.normal(0,2,n)
series=pd.Series(series)

#fit arima
model=ARIMA(series,order=(1,1,1))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)

print("\n Synthetic stock prices Example 5")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)

model=ARIMA(series,order=(1,1,2))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)

print("\n Synthetic stock prices Example 5")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)

model=ARIMA(series,order=(2,1,2))
fit=model.fit()
#compute FPE
```

```
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)

print("\n Synthetic stock prices Example 5")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)

model=ARIMA(series,order=(3,1,2))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)

print("\n Synthetic stock prices Example 5")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)

model=ARIMA(series,order=(2,0,2))
fit=model.fit()
#compute FPE
n_obs=fit.nobs
sigma2=fit.mse
k=fit.df_model+1
fpe=sigma2*(n_obs+k)/(n_obs-k)

print("\n Synthetic stock prices Example 5")
print("AIC:",fit.aic)
print("BIC:",fit.bic)
print("FPE:",fpe)
```

Output:



PRACTICAL N0 –10

Aim: ARCH & GARCH

Code:

```
pip install yfinance
pip install arch
 #example1ARCH(1):Model on S & P 500 Returns
 #step 1
 import yfinance as yf
 import pandas as pd
 import numpy as np
 import matplotlib.pyplot as plt
 from arch import arch_model

 #step2
 #load S&P 500 daily data
 data=yf.download("^GSPC",start="2020-01-01",end="2025-01-01")
 returns=100*data['Close'].pct_change().dropna()
#step 3:Fit arch(1)model
model=arch_model(returns,vol='ARCH',p=1)
results=model.fit(disp='off')

print("Examole 1 - ARCH(1) on S&P 500 Returns")
print(results.summary())
```

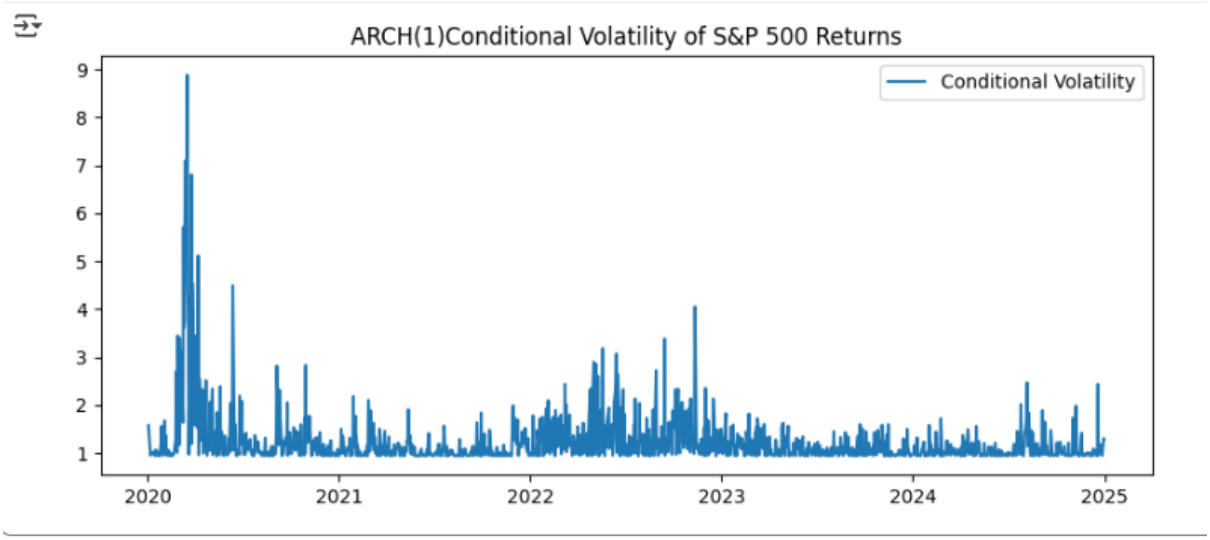
Output:

```
Examole 1 - ARCH(1) on S&P 500 Returns
                     Constant Mean - ARCH Model Results
==============================================================================
Dep. Variable:                  ^GSPC   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                       ARCH   Log-Likelihood:               -1974.89
Distribution:                  Normal   AIC:                           3955.79
Method:            Maximum Likelihood   BIC:                           3971.19
                                        No. Observations:                 1257
Date:                Sat, Sep 13 2025   Df Residuals:                     1256
Time:                        10:21:59   Df Model:                            1
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.1347  3.299e-02      4.083  4.455e-05 [7.003e-02,  0.199]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.8862  8.363e-02     10.597  3.091e-26 [  0.722,  1.050]
alpha[1]       0.5308      0.123      4.319  1.567e-05 [  0.290,  0.772]
==============================================================================

Covariance estimator: robust
```

Code:

```
#step 5: plot conditional volatility
plt.figure(figsize=(10,4))
plt.plot(results.conditional_volatility,label='Conditional Volatility')
plt.title('ARCH(1)Conditional Volatility of S&P 500 Returns')
plt.legend()
plt.show()
```

Output:

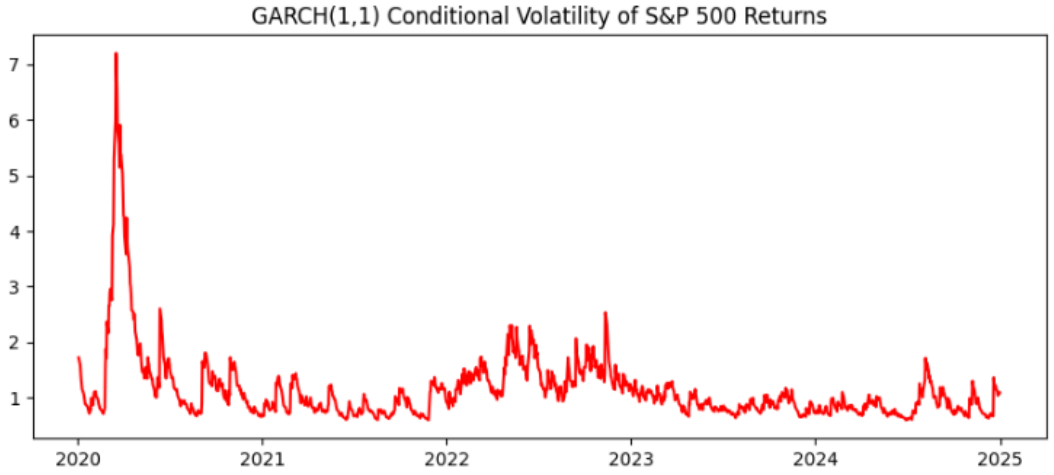


Code:

```
#example2:GARCH(1,1) model on S&P 500 returns
#step 1: fit garch(1,1) model
model=arch_model(returns,vol='GARCH',p=1,q=1)
results=model.fit(disp='off')

print("\nExample 2 - GARCH(1,1) on S&P 500 Returns")
print(results.summary())
#step 2: plot volatility
plt.figure(figsize=(10,4))
plt.plot(results.conditional_volatility,color='red')
plt.title('GARCH(1,1) Conditional Volatility of S&P 500 Returns')
plt.show()
```

Output:

```
Example 2 - GARCH(1,1) on S&P 500 Returns
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                  ^GSPC   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -1832.95
Distribution:                  Normal   AIC:                           3673.91
Method:            Maximum Likelihood   BIC:                           3694.45
                                        No. Observations:                 1257
Date:                Sat, Sep 13 2025   Df Residuals:                     1256
Time:                        10:29:36   Df Model:                            1
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.0869  2.554e-02      3.404  6.651e-04 [3.687e-02,  0.137]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0540  1.945e-02      2.775  5.515e-03 [1.586e-02,9.210e-02]
alpha[1]       0.1560  3.458e-02      4.512  6.423e-06   [8.826e-02,  0.224]
beta[1]        0.8073  3.939e-02     20.494  2.431e-93     [  0.730,  0.885]
==============================================================================

Covariance estimator: robust
```



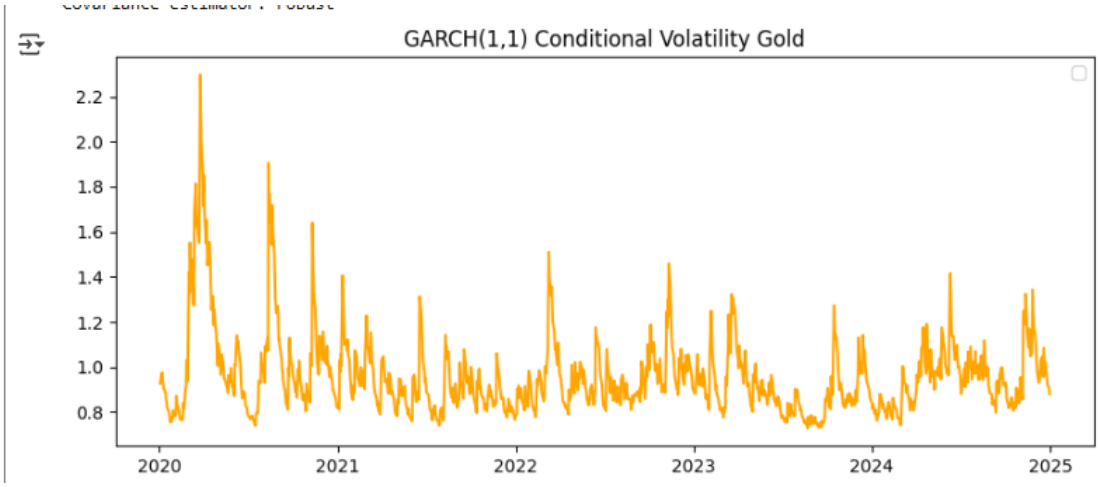
Code:

```
#Example 3: GARCH(1,1) on gold prices
#Step 1: Download gold prices data(GLD ETF)
gold=yf.download('GLD',start='2020-01-01',end='2025-01-01')
gold_returns=100*gold['Close'].pct_change().dropna()

#Step 2: Fit GARCH(1,1) model
model=arch_model(gold_returns,vol='GARCH',p=1,q=1)
results=model.fit(disp='off')

print("\nExample 3 - GARCH(1,1) on Gold Returns")
```

```
print(results.summary())
#step 3
plt.figure(figsize=(10,4))
plt.plot(results.conditional_volatility,color='orange')
plt.title('GARCH(1,1) Conditional Volatility Gold')
plt.legend()
plt.show()
```

Output:

```
Example 3 - GARCH(1,1) on Gold Returns
                  Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                    GLD   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -1714.30
Distribution:                  Normal   AIC:                           3436.61
Method:            Maximum Likelihood   BIC:                           3457.15
                                        No. Observations:                 1257
Date:                Sun, Sep 14 2025   Df Residuals:                     1256
Time:                        12:52:46   Df Model:                            1
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.0305  2.636e-02      1.158      0.247 [-2.113e-02,8.222e-02]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0817  2.423e-02      3.372  7.450e-04 [3.423e-02,  0.129]
alpha[1]       0.0890  2.563e-02      3.474  5.135e-04 [3.880e-02,  0.139]
beta[1]        0.8255  3.686e-02     22.399 4.048e-111   [  0.753,  0.898]
==============================================================================

Covariance estimator: robust
```

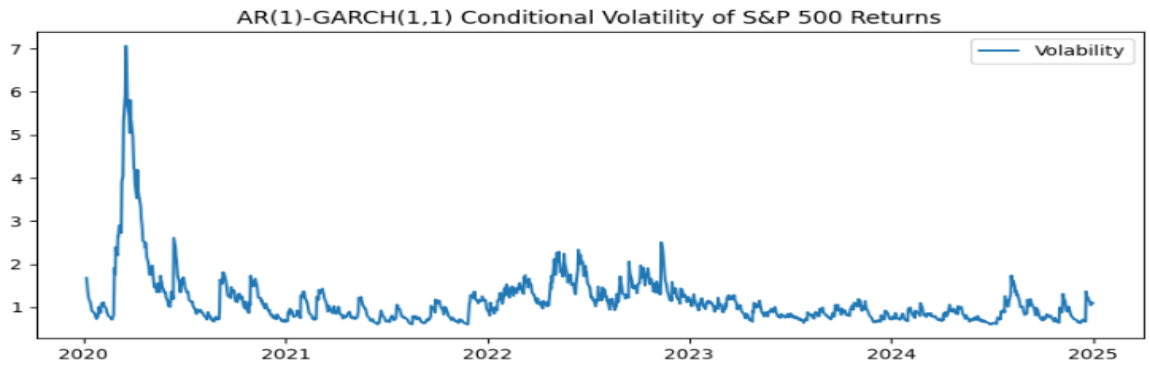


Code:

```
#Example 4: GARCH with AR(1) Mean(ARIMA + GARCH)
#step1 :fit AR(1)-GARCH(1,1)
model=arch_model(returns,mean='AR',lags=1,vol='GARCH',p=1,q=1)
results=model.fit(disp='off')

print("\nExample 4 - AR(1)-GARCH(1,1) on S&P 500 Returns")
print(results.summary())
#step 2:
plt.figure(figsize=(10,4))
plt.plot(results.conditional_volatility,label='Volability')
plt.title('AR(1)-GARCH(1,1) Conditional Volatility of S&P 500 Returns')
plt.legend()
plt.show()
```

Output:

```
Example 4 - AR(1)-GARCH(1,1) on S&P 500 Returns
                           AR - GARCH Model Results
==============================================================================
Dep. Variable:                  ^GSPC   R-squared:                       0.011
Mean Model:                        AR   Adj. R-squared:                  0.010
Vol Model:                      GARCH   Log-Likelihood:               -1831.10
Distribution:                  Normal   AIC:                           3672.20
Method:            Maximum Likelihood   BIC:                           3697.88
                                        No. Observations:                 1256
Date:                Sun, Sep 14 2025   Df Residuals:                     1254
Time:                        12:52:46   Df Model:                            2
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
Const          0.0914  2.610e-02      3.502  4.611e-04    [4.025e-02,  0.143]
^GSPC[1]      -0.0344  3.160e-02     -1.090      0.276 [-9.637e-02,2.749e-02]
                              Volatility Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0540  1.935e-02      2.792  5.236e-03 [1.611e-02,9.198e-02]
alpha[1]       0.1560  3.508e-02      4.447  8.716e-06   [8.724e-02,  0.225]
beta[1]        0.8073  3.959e-02     20.391  1.998e-92     [  0.730,  0.885]
==============================================================================

Covariance estimator: robust
```



Code:

```
#example 5: comparing GARCH(1,1) vs GARCH(2,1)
#step 1:
models=[(1,1),(2,1)]
results=[]
#step2:
for p,q in models:
  model=arch_model(returns,vol='GARCH',p=p,q=q)
  fit = model.fit(disp='off')
  results.append({"Model":f"GARCH({p},{q})",
                  "AIC":fit.aic,
                  "BIC":fit.bic,
                  "LogLik":fit.loglikelihood})
#step 3:
df=pd.DataFrame(results)
print("\nExample 5 - Model Comparsion")
print(df)
```

Output:

```
Example 5 - Model Comparsion
        Model          AIC          BIC       LogLik
0  GARCH(1,1)  3673.907096  3694.453029 -1832.953548
1  GARCH(2,1)  3675.500570  3701.182986 -1832.750285
```

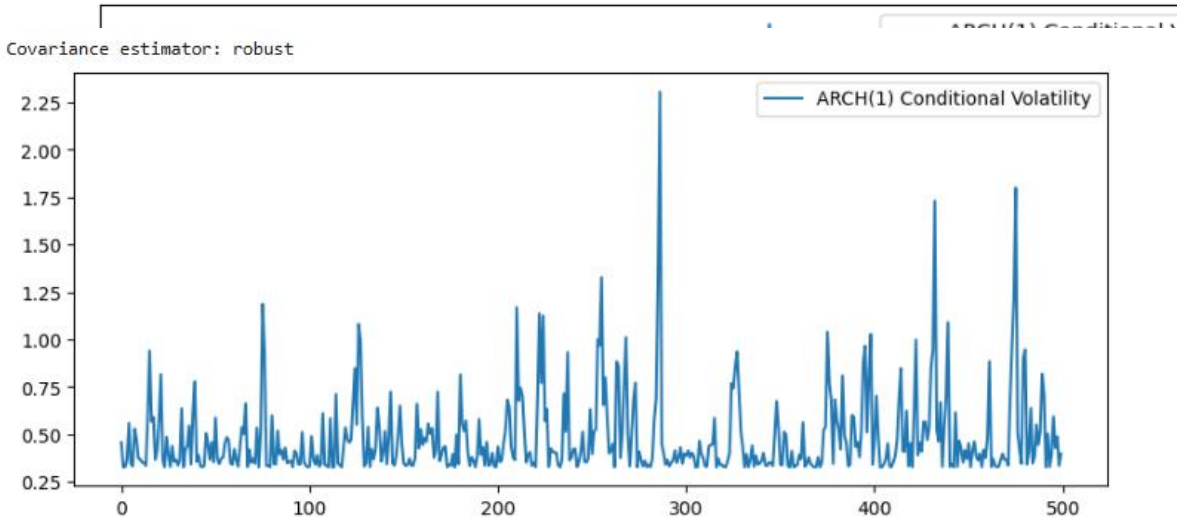
Code:

```
#program 6:Simulate & fit ARCH(1)Model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from arch import arch_model

#step 1: simulate synthetic data(volatility Clustering)
np.random.seed(42)
n= 500
eps = np.random.normal(size=n)
vol=np.zeros(n)
ret=np.zeros(n)

alpha0,alpha1 = 0.1,0.8 #ARCH(1) parameters

for t in range(1,n):
  vol[t]=np.sqrt(alpha0+alpha1*ret[t-1]**2)
  ret[t]=vol[t]*eps[t]
series=pd.Series(ret)
```

```
#step2:fit arch(1) model
arch_fit=arch_model(series,vol='ARCH',p=1).fit(disp='off')
print("\nExample 6 - ARCH(1) Synthetic Data")
print(arch_fit.summary())

#step 3: plot volability
plt.figure(figsize=(10,4))
plt.plot(arch_fit.conditional_volatility,label='ARCH(1) Conditional
Volatility')
plt.legend()
plt.show()
```

Output:

```
Example 6 - ARCH(1) Synthetic Data
                       Constant Mean - ARCH Model Results
==============================================================================
Dep. Variable:                    None   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                       ARCH   Log-Likelihood:               -310.574
Distribution:                  Normal   AIC:                           627.147
Method:            Maximum Likelihood   BIC:                           639.791
                                        No. Observations:                  500
Date:                Sun, Sep 14 2025   Df Residuals:                      499
Time:                        12:52:47   Df Model:                            1
                                 Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
mu         -6.0859e-03  1.577e-02     -0.386      0.700 [-3.700e-02,2.483e-02]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.1070  1.226e-02      8.728  2.586e-18 [8.296e-02,  0.131]
alpha[1]       0.6451  8.886e-02      7.260  3.869e-13   [  0.471,  0.819]
============================================================================

Covariance estimator: robust
```

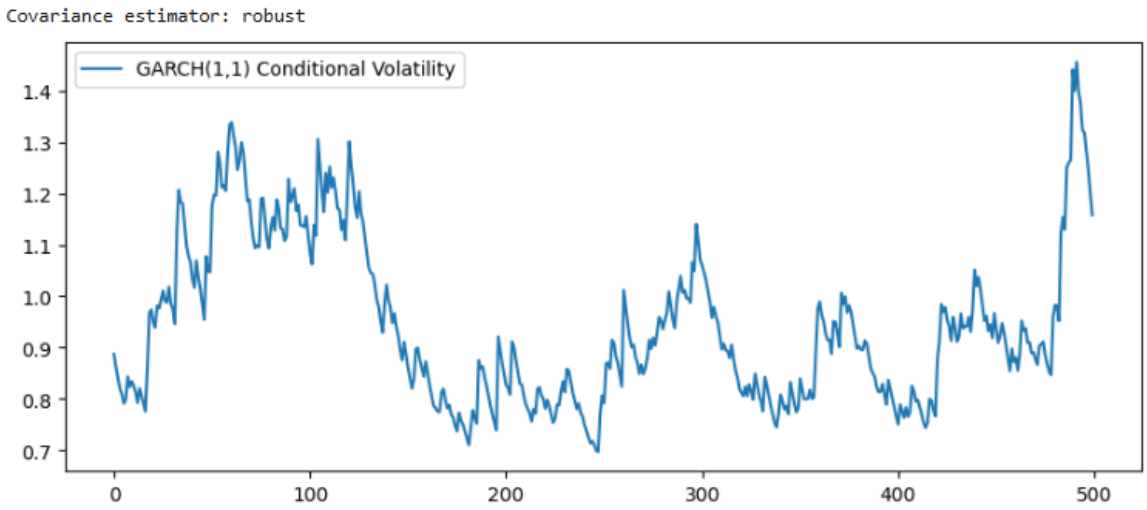


Code:

```
#program 7:Simulate & fit GARCH(1,1) model
#step 1:
np.random.seed(123)
n= 500
eps = np.random.normal(size=n)
ret=np.zeros(n)
vol=np.zeros(n)

omega,alpha,beta=0.05,0.1,0.85 #GARCH(1,1)
for t in range(1,n):
    vol[t]=np.sqrt(omega+alpha*ret[t-1]**2+beta*vol[t-1]**2)
    ret[t]=vol[t]*eps[t]
series = pd.Series(ret)

#step2 fit garch(1,1)
garch_fit=arch_model(series,vol='GARCH',p=1,q=1).fit(disp='off')
print("\nExample 7 - GARCH(1,1) Synthetic Data")
print(garch_fit.summary())

plt.figure(figsize=(10,4))
plt.plot(garch_fit.conditional_volatility,label='GARCH(1,1) Conditional
Volatility')
plt.legend()
plt.show()
```

Output:

```
Example 7 - GARCH(1,1) Synthetic Data
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                  None   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -672.793
Distribution:                  Normal   AIC:                           1353.59
Method:            Maximum Likelihood   BIC:                           1370.44
                                        No. Observations:                  500
Date:                Sun, Sep 14 2025   Df Residuals:                      499
Time:                        12:52:48   Df Model:                            1
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
------------------------------------------------------------------------------
mu            -0.0400  4.004e-02     -1.000      0.317 [ -0.118,3.845e-02]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0295  1.539e-02      1.917  5.528e-02 [-6.656e-04,5.965e-02]
alpha[1]       0.0620  1.687e-02      3.677  2.361e-04  [2.897e-02,9.511e-02]
beta[1]        0.9068  2.421e-02     37.451 5.853e-307      [  0.859,  0.954]
==============================================================================
```



Code:

```
#example 8 Compare arch(1) vs garch(1,1)
#step1: fit arch(1)
arch_fit=arch_model(series,vol='ARCH',p=1).fit(disp='off')
#step2: compare aic/bic
print("\nProgram 3- Model Comparsion on Same Synthetic Data")
print("ARCH(1)-AIC:",arch_fit.aic,"BIC:",arch_fit.bic)
print("GARCH(1,1)-AIC:",garch_fit.aic,"BIC:",garch_fit.bic)
```

Output:

```
Program 3- Model Comparsion on Same Synthetic Data
ARCH(1)-AIC: 1375.1455766213467 BIC: 1387.7894009166132
GARCH(1,1)-AIC: 1353.5850430802452 BIC: 1370.443475473934
```

Code:

```
#program9:AR(1)-GARCH(1,1)(Mean+volatility)
np.random.seed(99)
n= 500
eps = np.random.normal(size=n)
ret=np.zeros(n)
vol=np.zeros(n)

phi=0.3 # AR(1) parameter
omega,alpha,beta=0.05,0.1,0.85 # GARCH(1,1) parameters: omega, alpha_1,
beta_1
for t in range(1,n):
  vol[t]=np.sqrt(omega+alpha*ret[t-1]**2+beta*vol[t-1]**2)
  ret[t]=phi*ret[t-1]+vol[t]*eps[t]

series=pd.Series(ret)
```

```
#step2: fit ar(1) - garch(1,1)
ar_garch_fit=arch_model(series,mean='AR',lags=1,vol='GARCH',p=1,q=1).fi
t(disp='off')
print("\nExample 8 - AR(1)-GARCH(1,1) on Synthetic Data")
print(ar_garch_fit.summary())
```

Output:

```
Example 8 - AR(1)-GARCH(1,1) on Synthetic Data
                        AR - GARCH Model Results
==============================================================================
Dep. Variable:                  None   R-squared:                       0.082
Mean Model:                       AR   Adj. R-squared:                  0.080
Vol Model:                    GARCH   Log-Likelihood:                -679.645
Distribution:                Normal   AIC:                            1369.29
Method:           Maximum Likelihood   BIC:                            1390.35
                                       No. Observations:                  499
Date:                Sun, Sep 14 2025   Df Residuals:                      497
Time:                        12:52:48   Df Model:                            2
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
------------------------------------------------------------------------------
Const          0.0228  3.958e-02      0.575      0.565 [-5.480e-02,  0.100]
None[1]        0.2812  4.366e-02      6.440  1.192e-10    [  0.196,  0.367]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0623  2.864e-02      2.173  2.976e-02 [6.111e-03,  0.118]
alpha[1]       0.1163  3.305e-02      3.519  4.327e-04 [5.153e-02,  0.181]
beta[1]        0.8226  5.141e-02     16.001  1.248e-57   [  0.722,  0.923]
==============================================================================
```

Code:

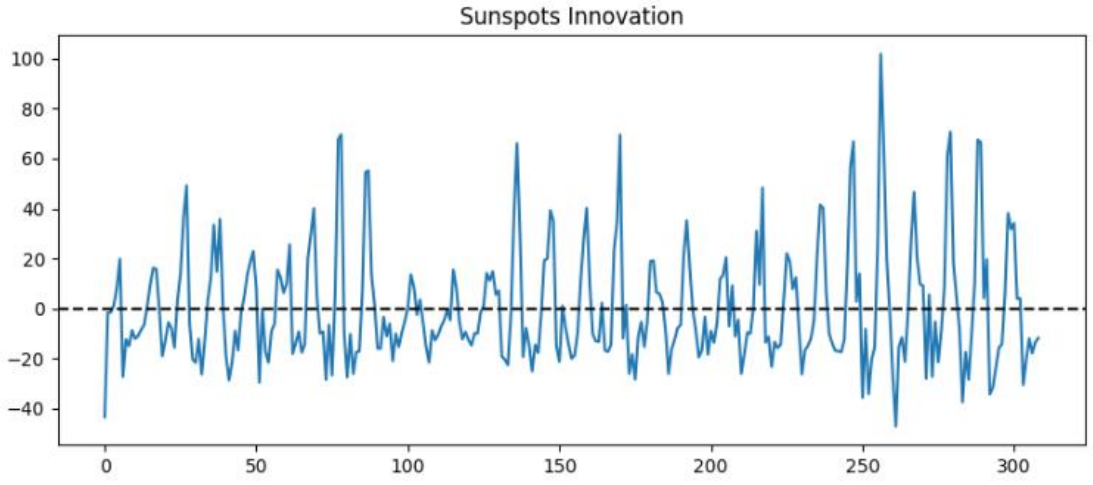
```
#program 1:Innovation from ar(1)model(Sunspots)
import matplotlib.pyplot as plt
from statsmodels.datasets import sunspots
from statsmodels.tsa.arima.model import ARIMA
#LOAD data
data = sunspots.load_pandas().data['SUNACTIVITY']
#fit ar(1)
model=ARIMA(data,order=(1,0,0)).fit()
innovations=model.resid

print("===Program 1 Output ===")
print(model.summary())
print("\nFirst 10 innovations:\n",innovations.head(10))

#plot
plt.figure(figsize=(10,4))
plt.plot(innovations)
plt.axhline(0,linestyle='--',color='black')
```

```
plt.title('Sunspots Innovation')
plt.show()
```

Output:

```
===Program 1 Output ===
                               SARIMAX Results
==============================================================================
Dep. Variable:            SUNACTIVITY   No. Observations:                  309
Model:                 ARIMA(1, 0, 0)   Log Likelihood               -1406.585
Date:                Sun, 14 Sep 2025   AIC                           2819.169
Time:                        12:52:50   BIC                           2830.369
Sample:                             0   HQIC                          2823.647
                                  - 309
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         48.3963      9.949      4.865      0.000      28.897      67.895
ar.L1          0.8244      0.036     23.011      0.000       0.754       0.895
sigma2       524.5565     39.103     13.415      0.000     447.916     601.197
===================================================================================
Ljung-Box (L1) (Q):                 100.01   Jarque-Bera (JB):               135.37
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.81   Skew:                             1.29
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.96
===================================================================================

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

First 10 innovations:
```

```
1    -1.619376
2    -1.565917
3     1.311966
4     8.541001
5    19.823496
6   -27.313821
7   -12.405539
8   -14.985728
9    -8.741493
dtype: float64
```

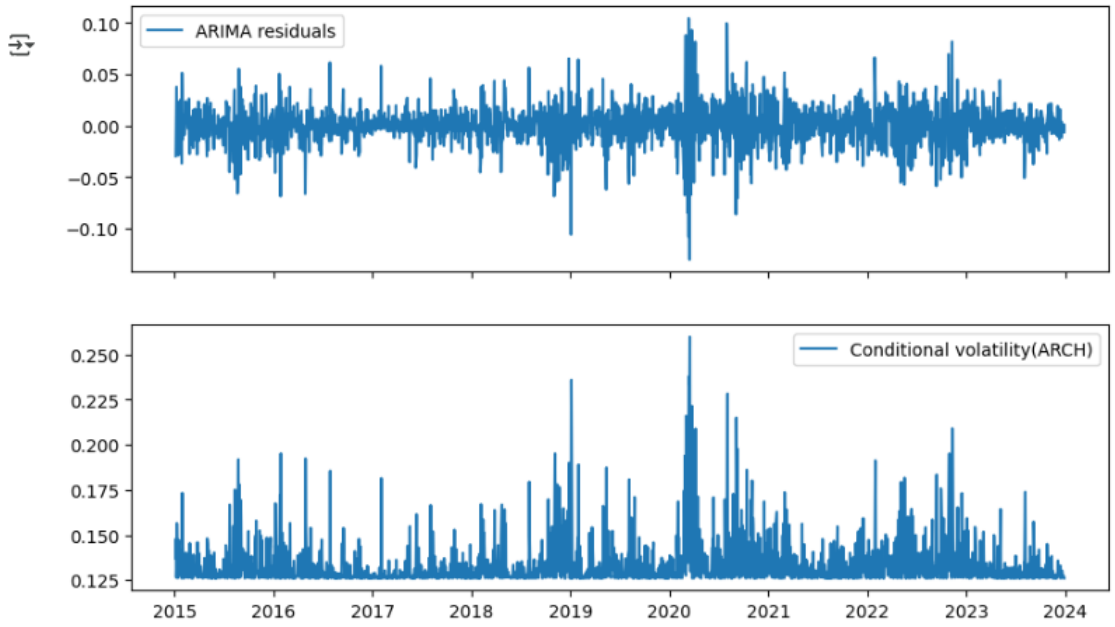


Code:

```
!pip install arch
import numpy as np
import pandas as pd
import yfinance as yf
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
from typing_extensions import dataclass_transform
import matplotlib.pyplot as plt
#Step1: load data
ticker="AAPL"
data=yf.download(ticker,start="2015-01-01",end="2023-12-
31",progress=False)
data=data['Close'].dropna()

#step2: compute return
returns=np.log(data).diff().dropna()

#step 3: fit arima model on return
arima_model_obj = ARIMA(returns, order=(1, 0, 0))
arima_res = arima_model_obj.fit()

#step 4: fit arch model on arima residuals
arch_model_obj=arch_model(arima_res.resid,vol="ARCH",p=1,o=0,q=0,dist='
normal')
arch_res=arch_model_obj.fit(update_freq=5)
print(arch_res.summary())

#step 5:forecast
horizon = 5
arima_fore = arima_res.get_forecast(steps=horizon)
vol_fore = arch_res.forecast(horizon=horizon)
print("Mean forecast:\n",arima_fore.predicted_mean.tail(horizon))
print("Vol forecast:\n",vol_fore.variance.tail(horizon))

#Step6: plot
fig,ax=plt.subplots(2,1,figsize=(10,6),sharex=True)
ax[0].plot(arima_res.resid,label='ARIMA residuals')
ax[0].legend()
ax[1].plot(np.sqrt(arch_res.conditional_volatility),label='Conditional
volatility(ARCH)')
ax[1].legend()
plt.show()
```

Output:

```
Function evaluations: 43
Gradient evaluations: 6
                     Constant Mean - ARCH Model Results
==============================================================================
Dep. Variable:                    None   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                       ARCH   Log-Likelihood:                5926.25
Distribution:                  Normal   AIC:                           -11846.5
Method:            Maximum Likelihood   BIC:                           -11829.3
                                        No. Observations:                 2263
Date:                Mon, Sep 15 2025   Df Residuals:                     2262
Time:                        11:26:19   Df Model:                            1
                                 Mean Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
mu         5.3491e-04  3.955e-04      1.352      0.176 [-2.403e-04,1.310e-03]
                           Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega      2.5262e-04  1.759e-05     14.364  8.735e-47 [2.181e-04,2.871e-04]
alpha[1]       0.2512  5.939e-02      4.229  2.344e-05     [  0.135,  0.368]
============================================================================

Covariance estimator: robust
Mean forecast:
 2263    0.001385
```

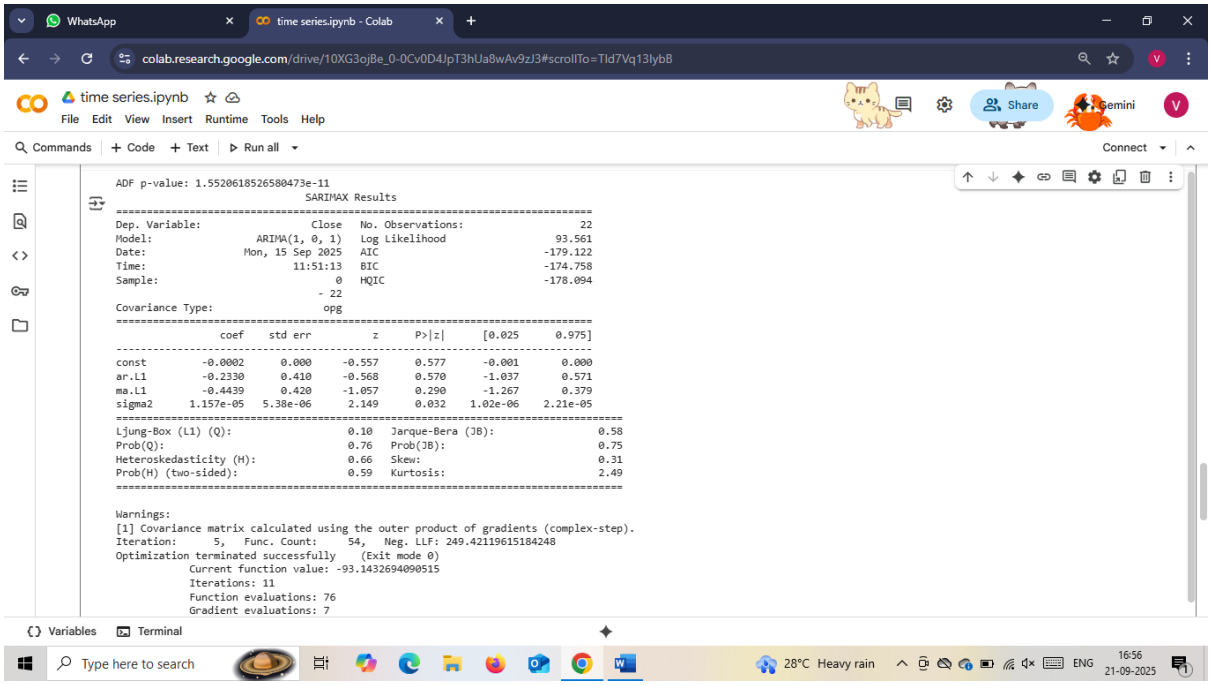


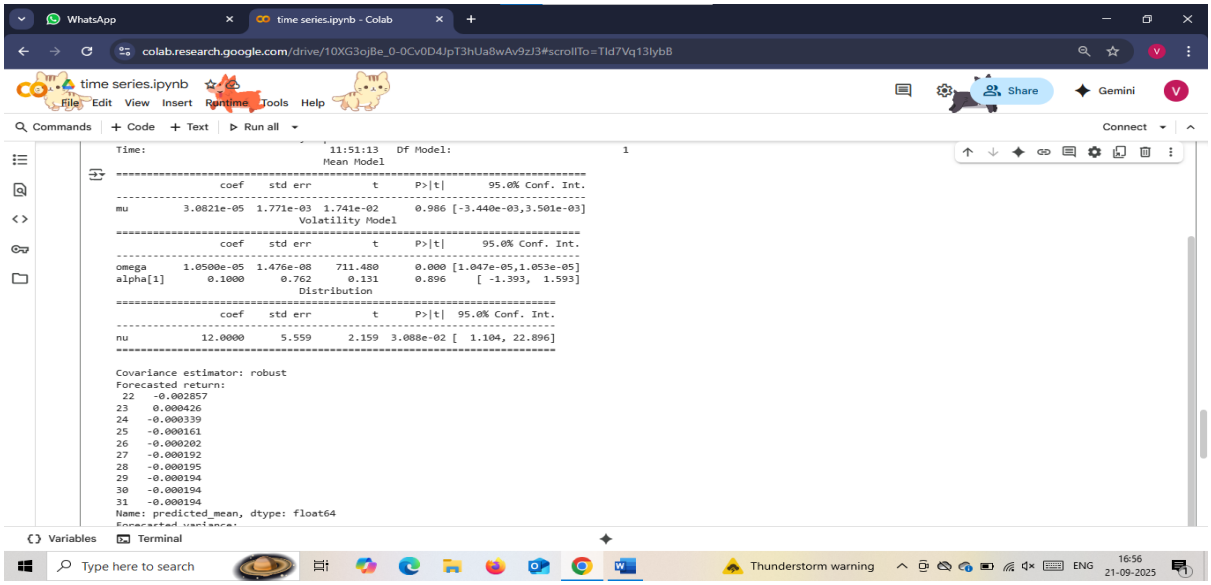
Code:

```
#example 2: arima+arch on exchange rates of us dollars
import pandas as pd
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
import matplotlib.pyplot as plt
```

```
#step 1: Load some exchange rate data
data=pd.read_csv('EURUSD_daily.csv')
data['Date'] = pd.to_datetime(data['Date'])
data = data.set_index('Date')
close=data['Close'].dropna()

#step 2: compute return
returns=np.log(close).diff().dropna()

#step 3: stationary check
adf=adfuller(returns)
print("ADF p-value:",adf[1])

#step 4: fit arima on return
model_arima=ARIMA(returns,order=(1,0,1))
res_arima=model_arima.fit()
print(res_arima.summary())

#step 5:ARCH(1) on residuals
am=arch_model(res_arima.resid,vol='ARCH',p=1,o=0,q=0,dist='t')
#use t distribution
res_arch=am.fit(update_freq=5)
print(res_arch.summary())
#Step6: forecast
steps=10
mean_fore=res_arima.get_forecast(steps=steps)
vol_fore=res_arch.forecast(horizon=steps)
print("Forecasted return:\n",mean_fore.predicted_mean.tail(steps))
print("Forecasted variance:\n",vol_fore.variance.tail(steps))
```

Output:

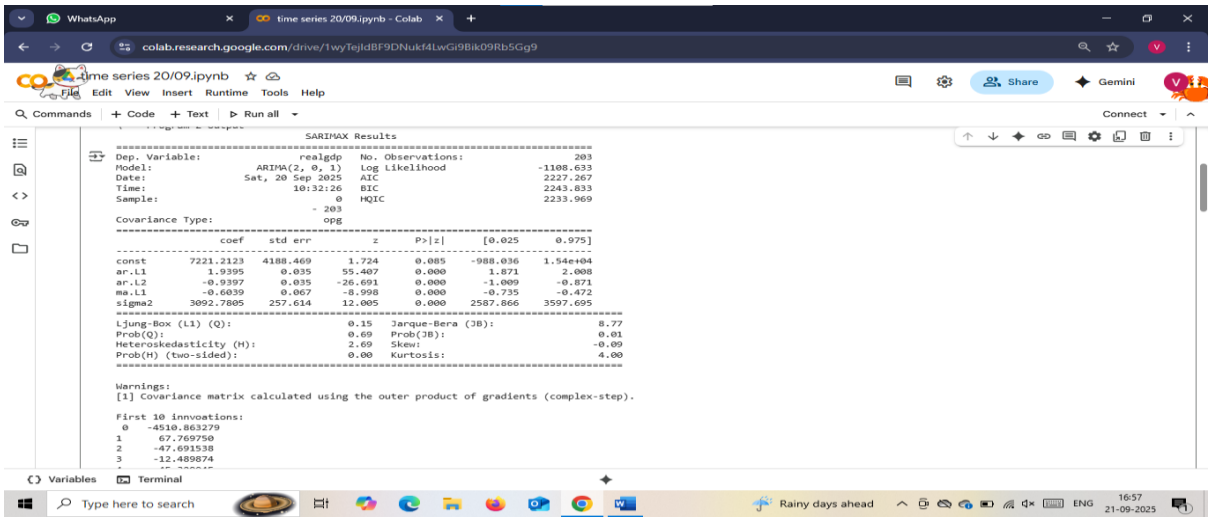




Code:

```
#program2 innovation from arma(2,1) (real GDP)
from statsmodels.datasets import macrodata
from statsmodels.tsa.arima.model import ARIMA

#load gdp data
macro=macrodata.load_pandas().data
gdp=macro['realgdp']

#fit arma(2,1)
model=ARIMA(gdp,order=(2,0,1)).fit()
innovations=model.resid

print("\=== Program 2 Output ===")
print(model.summary())
print("\nFirst 10 innvoations:\n",innovations.head(10))
```

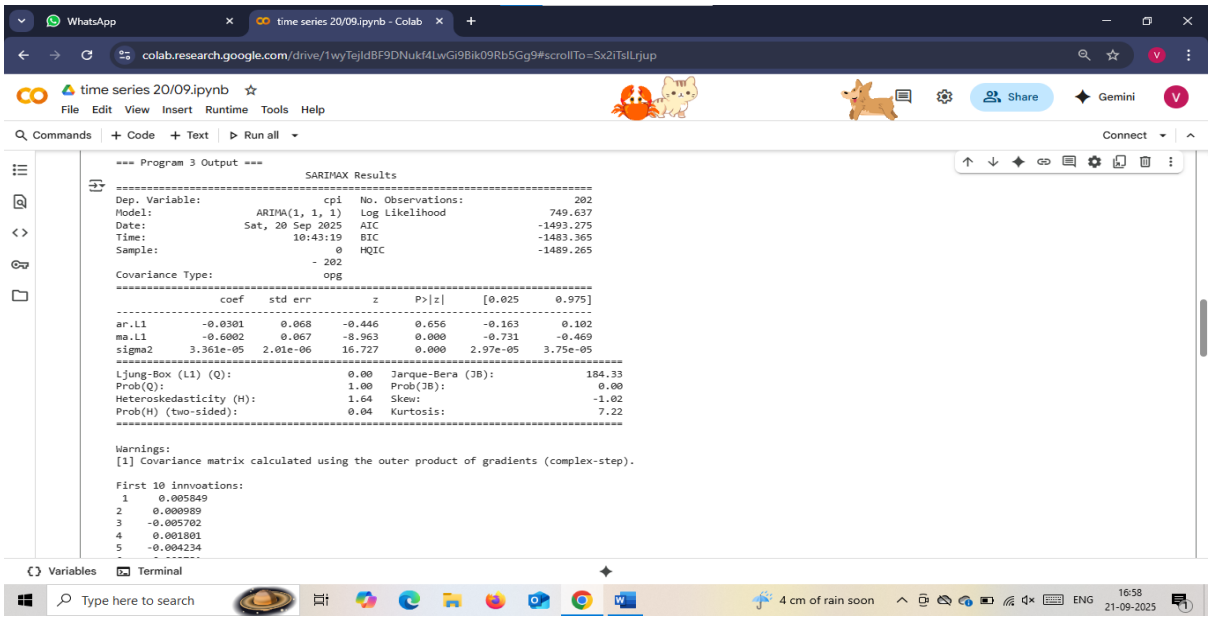
Output:



Code:

```
#Program 3: innovations from ARIMA(1,1,1)(CPI inflation)

import numpy as np
#compute quarterly inflation
cpi=macro['cpi']
inflation=np.log(cpi).diff().dropna()

model=ARIMA(inflation,order=(1,1,1)).fit()
innovations=model.resid

print("\n\n=== Program 3 Output ===")
print(model.summary())

print("\nFirst 10 innvoations:\n",innovations.head(10))
```

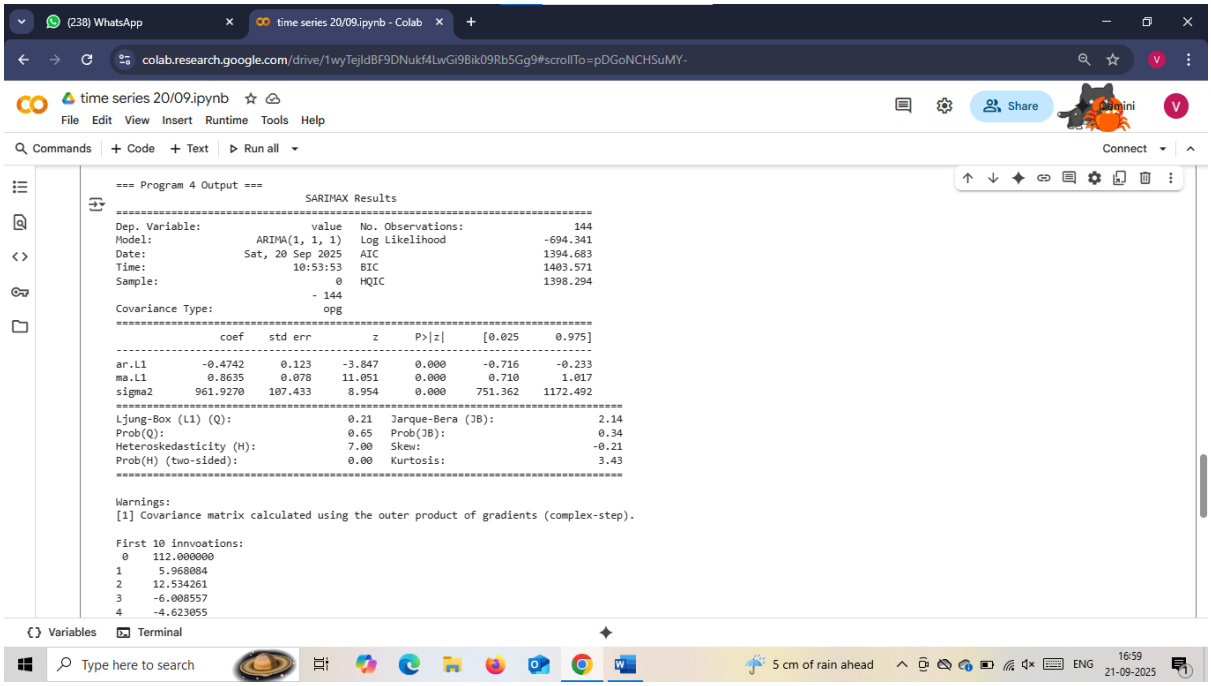
Output:



Code:

```
from statsmodels.datasets import get_rdataset
from statsmodels.tsa.arima.model import ARIMA
#load airline data
airline_data = get_rdataset("AirPassengers").data['value']
#fit arima(1,1,1)
model = ARIMA(airline_data, order=(1, 1, 1)).fit()
innovations=model.resid

print("\n\n=== Program 4 Output ===")
print(model.summary())

print("\nFirst 10 innvoations:\n",innovations.head(10))
```

Output:



Code:

```
#program 5 :Ljung box test on innovations
from statsmodels.stats.diagnostic import acorr_ljungbox
lb_test = acorr_ljungbox(innovations,lags=[10],return_df=True)
print("\n\n=== Program 5 Output ===")
print(lb_test)
```

Output:

```
=== Program 5 Output ===
     lb_stat  lb_pvalue
10  28.82991   0.001328
```