ECONOMETRICSANDFINANCELAB

(COURSECODE:22UPCSC4E11)

A Laboratory record Submitted to Periyar University, Salem.
In partial fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

IN

DATA SCIENCE

BY

ANANDARATHI.U.B

REGNO:U22PG507DTS004

DEPARTMENTOFCOMPUTERSCIENCE PERIYAR

UNIVERSITY

PERIYARPALKALAINAGAR,

SALEM – 636011

CERTIFICATE

This is to certify that the Programming Laboratory entitled **"ECONOMETRICS AND FINANCELAB"**(22UPCSC4E11)is a bonafide record work done by **MS. ANANDARATHI.U.B** Register No. U22PG507DTS004 as partial fulfillment of the requirements degree of MASTER OF SCIENCEIN DATA SCIENCE in the Department of Computer Science, Periyar University, Salem, during the year 2022 – 2024.

Faculty In-Charge   Head of the Department

Submitted for the practical examination held on……./……./………………

MarksObtained

Internal Examiner   External Examiner

INDEX

EXNO:1	FUNDAMENTALOFECONOMETRICS
DATE:06.07.23	

AIM:

To understand the fundamentals of econometrics.

THEORY:

Econometrics is a branch of economics that uses statistical and mathematical methods to analyze and quantify economic relationships and make predictions based on data. It Combines economic theory with statistical tools to provide a rigorous framework for Understanding and modeling economic phenomena. The fundamentals of econometrics can be broadly categorized into several key components:

1. **Data Collection and Sources:**

Econometric analysis starts with the collection of relevant data. Researchers gather data from various sources, including government agencies, surveys, financial reports, and other sources. Data should be accurate, reliable, and representative of the economic phenomena under investigation.

2. Economic Theory:

Economic theory provides the foundation for econometric modeling. It involves formulating hypotheses and constructing theoretical models that describe the relationships between economic variables. Economic theories help in developing hypotheses and Specifying the functional forms of econometric models.

3. Specification of Econometric Models:

Econometric models are mathematical representations of economic relationships. These models specify how dependent and independent variables are related. The choice of model depends on the economic theory and the specific question being addressed. Common model types include linear regression models, time-series models, and panel data models.

4. Estimation:

Estimation involves determining the parameters of the econometric model. Researchers use statistical techniques to estimate these parameters based on the available data. Ordinary Least Squares (OLS) regression is a widely used method for estimating parameters in linear models. Maximum Likelihood Estimation(MLE) is another common technique for estimating model parameters.

5.Hypothesis Testing:

After estimating the model parameters, researchers test hypotheses about the relationships between economic variables. Hypothesis testing helps determine the significance of the estimatedcoefficientsandwhethertheproposedrelationshipsarestatisticallyvalid.Common tests include t-tests, F-tests, and likelihood ratio tests.

1. Model Evaluation:

Econometric models should be evaluated for their goodness of fit and predictive power. Researchers use various statistical criteria, such as R-squared ,Akaike Information Criterion (AIC),and Bayesian Information Criterion(BIC),to assess the model's performance. A good model should provide a close fit to the data and be able to make accurate predictions.

2. Causality and Endogeneity:

Causality is a fundamental concept in econometrics, and researchers strive to establish causal relationships between variables. Endogeneity issues, where variables are correlated with the error term, can lead to biased parameter estimates. Various techniques, such as instrumental variables and control functions, are used to address endogeneity.

3. Multi collinearity and Heteroskedasticity:

Multi collinearity occurs when independent variables are highly correlated, making it difficult to is late the individual effects of each variable. Heteroskedasticity is the presence of non-constant variance in the error term. These issues can affect the efficiency of parameter estimates, and econometricians use diagnostic tests and remedies to address them.

4. Time Series Analysis:

Econometric analysis often deals with time series data, where observations are collected over time. Time series models account for temporal dependencies and trends in the data. Techniques like autoregressive integrated moving average (ARIMA) models and co integration are commonly used in time series econometrics.

5. Forecasting:

Econometric models can be used for forecasting future economic trends and outcomes. Forecasting involves extrapolating the model to make predictions about future values of economic variables, providing valuable information for decision-making.

RESULT:

Thus, the above program has executed successfully.

EXNO:2	**AUTO REGRESSION(AR)**
DATE:27.07.23	

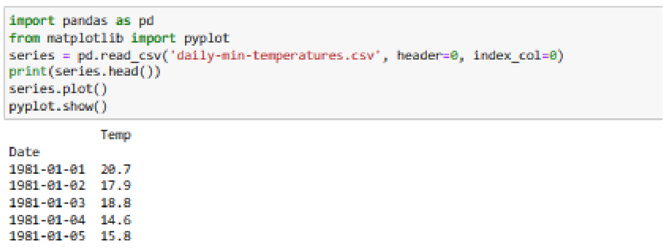
AIM:

To perform Auto regression in a time series dataset.

THEORY:

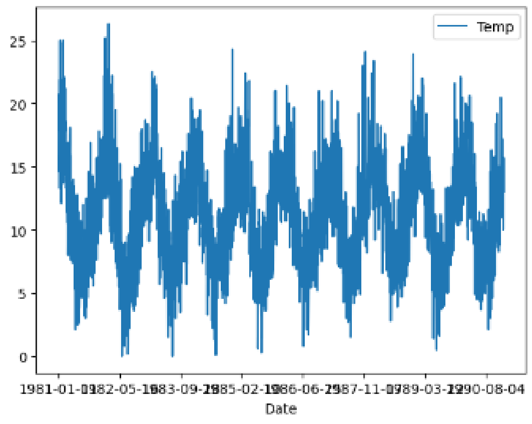
Auto regression, often abbreviated as AR, is a statistical and time series modeling technique used in econometrics and data analysis. It models the relationship between a Variable and its own past values. In an auto regressive model, the value of a variable at a given time is regressed on its past values, which allows for the analysis of trends, patterns, and autocorrelations in time series data. Auto regressive models are denoted as AR(p), where "p" represents the number of past time periods considered in the regression. These models are widely used for time series forecasting and understanding the temporal dynamics of various phenomena.

PROGRAM:



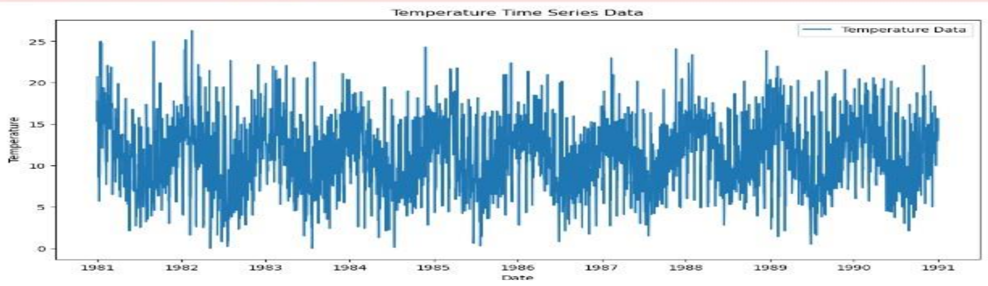
```
import pandas as pd
from matplotlib import pyplot
series = pd.read_csv('daily-min-temperatures.csv', header=0, index_col=0)
print(series.head())
series.plot()
pyplot.show()
```

```
            Temp
Date
1981-01-01  20.7
1981-01-02  17.9
1981-01-03  18.8
1981-01-04  14.6
1981-01-05  15.8
```



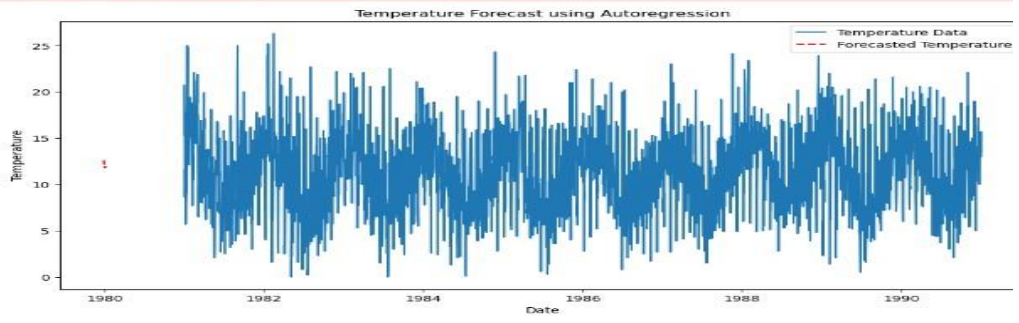
```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('daily-min-temperatures.csv')  # Replace 'temperature_data.csv' with your dataset file

# Convert the date column to datetime type (if it's not already)
data['Date'] = pd.to_datetime(data['Date'])

# Set the date column as the index
data.set_index('Date', inplace=True)

# Sort the data by date (just to be safe)
data = data.sort_index()

# Plot the original time series data
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Temperature'], label='Temperature Data')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.title('Temperature Time Series Data')
plt.legend()
plt.show()
```

```
C:\Users\afrah\AppData\Local\Temp\ipykernel_18284\432076665.py:9: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst
=False (the default) was specified. This may lead to inconsistently parsed dates! Specify a format to ensure consistent parsin
g.
  data['Date'] = pd.to_datetime(data['Date'])
```



```
# Create an autoregressive model (AR)
lags = 7  # Number of lags for the autoregressive model (you can change this)
model = sm.tsa.AutoReg(data['Temperature'], lags=lags)
results = model.fit()

# Perform forecasting
forecast_values = results.predict(start=len(data), end=len(data) + lags - 1, dynamic=False)

# Plot the original data and the forecasted values
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Temperature'], label='Temperature Data')
plt.plot(forecast_values.index, forecast_values, label='Forecasted Temperature', color='red', linestyle='dashed')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.title('Temperature Forecast using Autoregression')
plt.legend()
plt.show()
```

```
C:\Users\afrah\.conda\envs\tensorflow_env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index
as been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\afrah\.conda\envs\tensorflow_env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: ValueWarning: No supported
ndex is available. Prediction results will be given with an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\afrah\.conda\envs\tensorflow_env\lib\site-packages\statsmodels\tsa\base\tsa_model.py:836: FutureWarning: No supporte
index is available. In the next version, calling this method in a model without a supported index will result in an exception
  return get_prediction_index(
C:\Users\afrah\.conda\envs\tensorflow_env\lib\site-packages\statsmodels\tsa\deterministic.py:302: UserWarning: Only PeriodInd
es, DatetimeIndexes with a frequency set, RangesIndexes, and Index with a unit increment support extending. The index is set
ll contain the position relative to the data length.
  fcast_index = self._extend_index(index, steps, forecast_index)
```



```
# Print the forecasted values
print("Forecasted Temperature Values:")
print(forecast_values)
```

```
Forecasted Temperature Values:
3650    12.607152
3651    12.331274
3652    12.072388
3653    11.820248
3654    11.796653
3655    11.919470
3656    11.769712
dtype: float64
```

RESULT:

Thus, auto regression has executed successfully.

EXNO:3	MOVING AVERAGE(MA)
DATE:17.08.23	

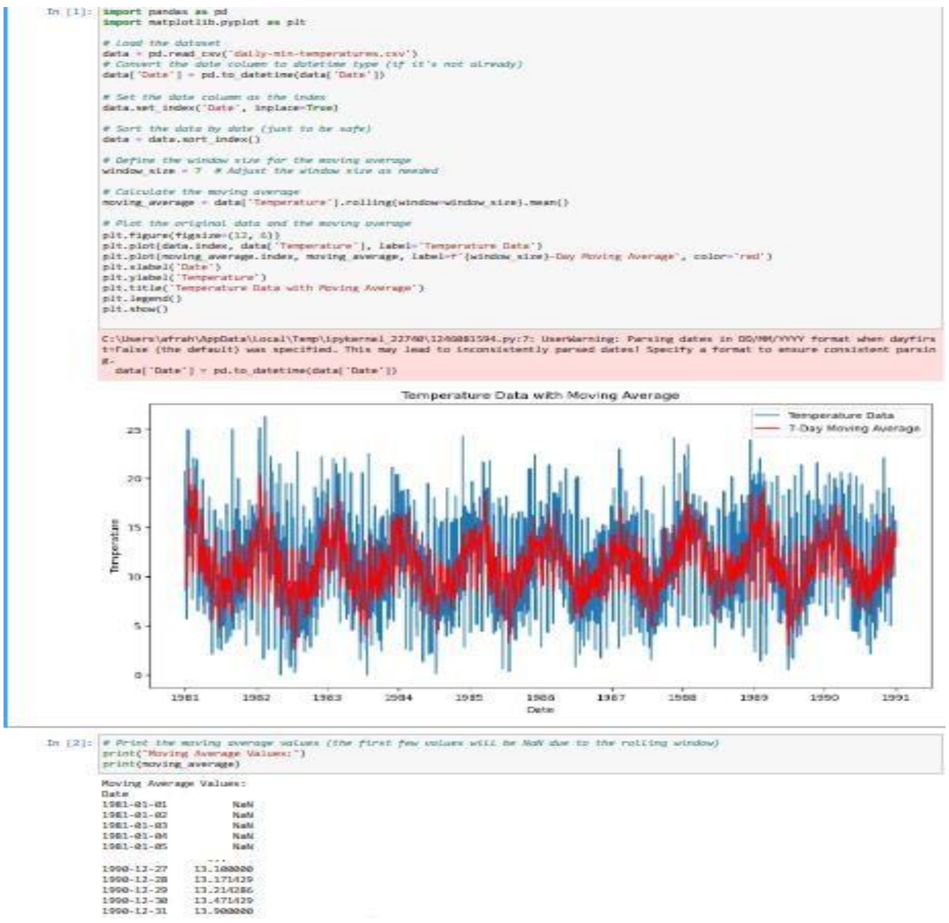
AIM:

To perform moving average in a time series dataset.

THEORY:

A moving average is a statistical calculation used to smooth out fluctuations in data over time. It involves taking the average of a set of data points within a moving window or time period. This technique is commonly used in time series analysis to reveal trends and patterns in data while reducing noise. There are different types of moving averages, including simple moving averages(SMA), exponential moving averages(EMA),and weighted moving averages.

PROGRAM:



```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt

        # Load the dataset
        data = pd.read_csv('daily-min-temperatures.csv')
        # Convert the date column to datetime type (if it's not already)
        data['Date'] = pd.to_datetime(data['Date'])

        # Set the date column as the index
        data.set_index('Date', inplace=True)

        # Sort the data by date (just to be safe)
        data = data.sort_index()

        # Define the window size for the moving average
        window_size = 7  # Adjust the window size as needed

        # Calculate the moving average
        moving_average = data['Temperature'].rolling(window=window_size).mean()

        # Plot the original data and the moving average
        plt.figure(figsize=(12, 6))
        plt.plot(data.index, data['Temperature'], label='Temperature Data')
        plt.plot(moving_average.index, moving_average, label=f'{window_size}-Day Moving Average', color='red')
        plt.xlabel('Date')
        plt.ylabel('Temperature')
        plt.title('Temperature Data with Moving Average')
        plt.legend()
        plt.show()

C:\Users\afrah\AppData\Local\Temp\ipykernel_22740\1240881594.py:7: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirs
t=False (the default) was specified. This may lead to inconsistently parsed dates! Specify a format to ensure consistent parsin
g.
  data['Date'] = pd.to_datetime(data['Date'])
```

```
In [2]: # Print the moving average values (the first few values will be NaN due to the rolling window)
        print("Moving Average Values:")
        print(moving_average)

Moving Average Values:
Date
1981-01-01          NaN
1981-01-02          NaN
1981-01-03          NaN
1981-01-04          NaN
1981-01-05          NaN
                    ...
1990-12-27    13.100000
1990-12-28    13.171429
1990-12-29    13.214286
1990-12-30    13.471429
1990-12-31    13.900000
```

RESULT:

Thus, moving average has executed successfully.

EXNO:4	AUTO REGRESSIVE MOVING AVERAGE(ARMA)
DATE:24.08.23	

AIM:

To perform auto regression moving average in a time series dataset.

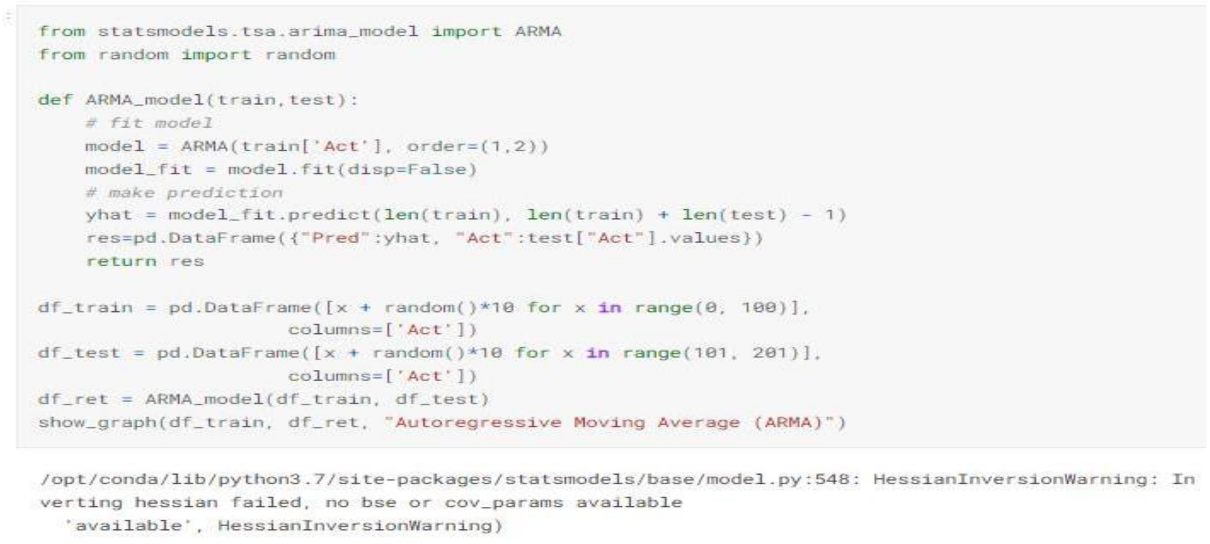
THEORY:

An ARMA(Auto Regressive Moving Average) model is a time series forecasting model used to predict future values based on past observations. It combines two components:

Auto Regressive(AR)component:Thispartmodelstherelationshipbetweenthecurrent Value and past values in a time series. It assumes that the current value is a linear combination of previous values.

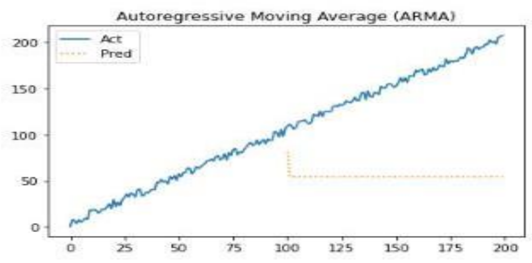
Moving Average (MA) component: This part models the relationship between the current value and past forecast errors .It helps account for random fluctuations or noise in the data.

PROGRAM:



```
from statsmodels.tsa.arima_model import ARMA
from random import random

def ARMA_model(train,test):
    # fit model
    model = ARMA(train['Act'], order=(1,2))
    model_fit = model.fit(disp=False)
    # make prediction
    yhat = model_fit.predict(len(train), len(train) + len(test) - 1)
    res=pd.DataFrame({"Pred":yhat, "Act":test["Act"].values})
    return res

df_train = pd.DataFrame([x + random()*10 for x in range(0, 100)],
                        columns=['Act'])
df_test = pd.DataFrame([x + random()*10 for x in range(101, 201)],
                        columns=['Act'])
df_ret = ARMA_model(df_train, df_test)
show_graph(df_train, df_ret, "Autoregressive Moving Average (ARMA)")
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:548: HessianInversionWarning: In
verting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
```



RESULT:

Thus ,auto regression moving average has executed successfully.

EXNO:5	**AUTOREGRESSIVEINTEGRATEDMOVINGAVERAGE(ARIMA)**
DATE:31.08.23	

AIM:

To perform auto regression integrated moving average in a time series dataset.

THEORY:

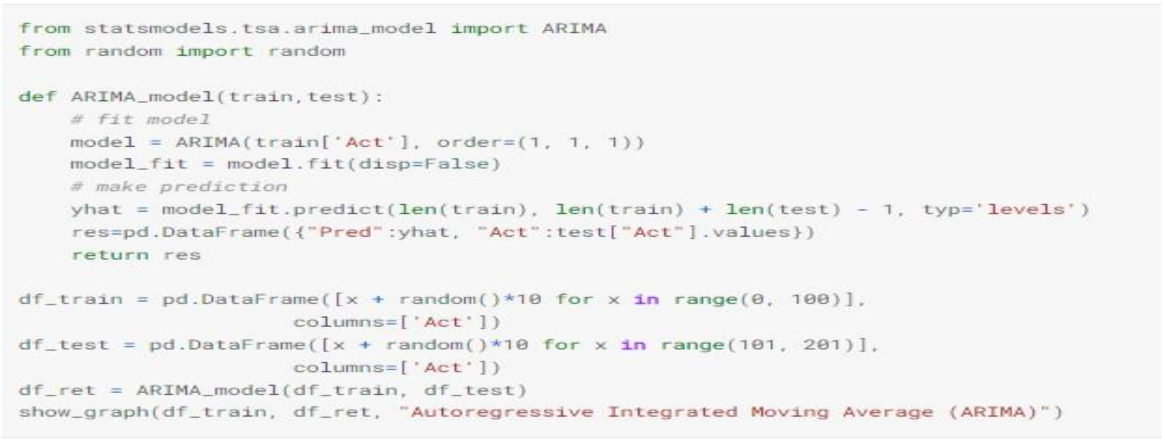
AnARIMA(AutoRegressiveIntegratedMovingAverage)modelisatimeseriesforecasting method used to analyze and predict data with temporal dependencies. It combines three components:

Auto Regressive(AR):Captures linear relationships between the current data point and previous data points.

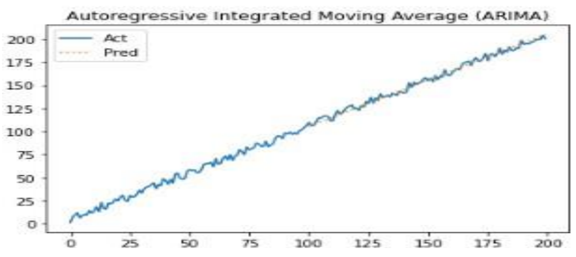
Integrated(I):Refers to differencing the data to make it stationary(i.e., constant mean and variance).

MovingAverage(MA):Modelstherelationshipbetweenthecurrentdatapoint and past forecast errors.

PROGRAM:



```
from statsmodels.tsa.arima_model import ARIMA
from random import random

def ARIMA_model(train,test):
    # fit model
    model = ARIMA(train['Act'], order=(1, 1, 1))
    model_fit = model.fit(disp=False)
    # make prediction
    yhat = model_fit.predict(len(train), len(train) + len(test) - 1, typ='levels')
    res=pd.DataFrame({"Pred":yhat, "Act":test["Act"].values})
    return res

df_train = pd.DataFrame([x + random()*10 for x in range(0, 100)],
                     columns=['Act'])
df_test = pd.DataFrame([x + random()*10 for x in range(101, 201)],
                     columns=['Act'])
df_ret = ARIMA_model(df_train, df_test)
show_graph(df_train, df_ret, "Autoregressive Integrated Moving Average (ARIMA)")
```



RESULT:

Thus, auto regression integrated moving average has executed successfully.

EXNO:6	**SEASONAL AUTO REGRESSIVE INTEGRATED MOVING-AVERAGE (SARIMA)**
DATE:07.09.23	

AIM:

Toperformseasonalautoregressionintegratedmovingaverageinatimeseries dataset.

THEORY:

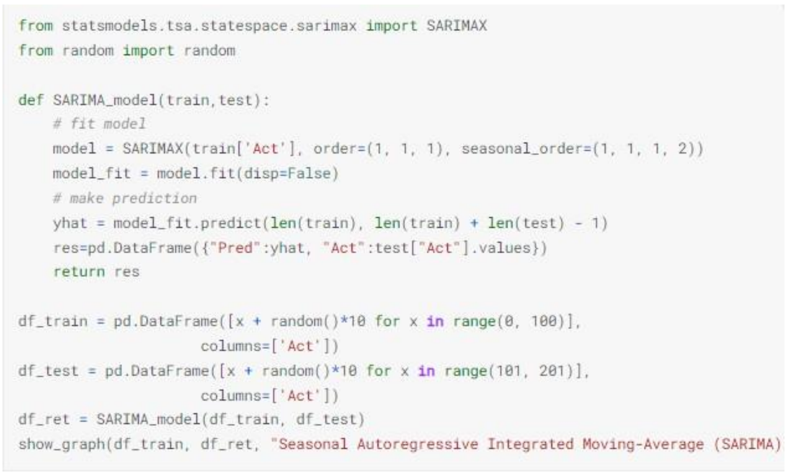
SARIMA(Seasonal Auto Regressive Integrated Moving Average) model is an extension of theARIMAmodelfortimeseriesdata.Itincorporatesseasonalityandcanbeusedto analyze and forecast data with repetitive patterns over time. SARIMA models include three main components:

Seasonal Auto Regressive(SAR)component

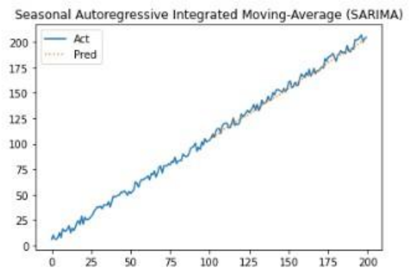
Seasonal Integrated (I) component

Seasonal Moving Average(SMA)component

PROGRAM:



```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from random import random

def SARIMA_model(train,test):
    # fit model
    model = SARIMAX(train['Act'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 2))
    model_fit = model.fit(disp=False)
    # make prediction
    yhat = model_fit.predict(len(train), len(train) + len(test) - 1)
    res=pd.DataFrame({"Pred":yhat, "Act":test["Act"].values})
    return res

df_train = pd.DataFrame([x + random()*10 for x in range(0, 100)],
                        columns=['Act'])
df_test = pd.DataFrame([x + random()*10 for x in range(101, 201)],
                        columns=['Act'])
df_ret = SARIMA_model(df_train, df_test)
show_graph(df_train, df_ret, "Seasonal Autoregressive Integrated Moving-Average (SARIMA)
```



RESULT:

Thus, seasonal auto regression integrated moving average has executed successfully.

EXNO:7	**SEASONAL AUTO REGRESSIVE INTEGRATED MOVING-AVERAGE WITH EXOGENOUS REGRESSORS (SARIMAX)**
DATE:14.09.23	

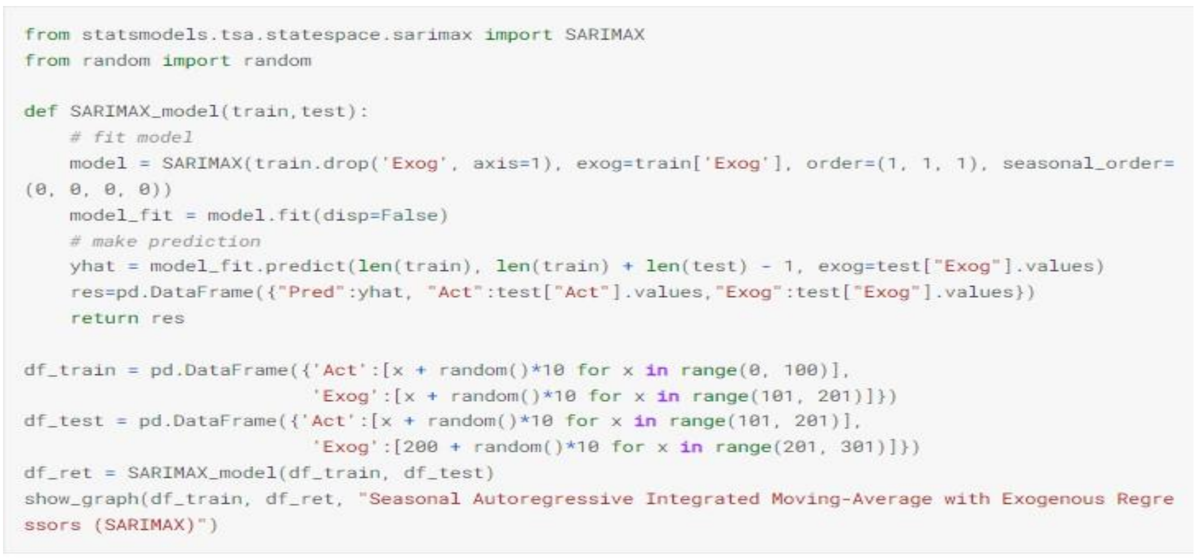
AIM:

To perform seasonal auto regression integrated moving average with exogenous Regressors in a time series dataset.

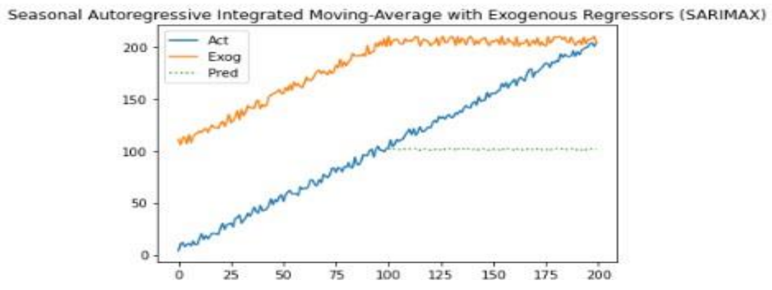
THEORY:

SARIMAX(SeasonalAutoRegressiveIntegratedMovingAveragewitheXogenousvariables) is a time series forecasting model that extends the traditional ARIMA model to account for seasonality and exogenous (external) variables. It combines autoregressive (AR) and moving average (MA) components with differencing and seasonal differencing to model and forecast time series data while allowing for the inclusion of external predictor variables that can Improve forecast accuracy. SARIMAX is particularly useful for data with clear seasonal patterns and when additional factors impact the time series.

PROGRAM:



```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from random import random

def SARIMAX_model(train,test):
    # fit model
    model = SARIMAX(train.drop('Exog', axis=1), exog=train['Exog'], order=(1, 1, 1), seasonal_order=
(0, 0, 0, 0))
    model_fit = model.fit(disp=False)
    # make prediction
    yhat = model_fit.predict(len(train), len(train) + len(test) - 1, exog=test["Exog"].values)
    res=pd.DataFrame({"Pred":yhat, "Act":test["Act"].values,"Exog":test["Exog"].values})
    return res

df_train = pd.DataFrame({'Act':[x + random()*10 for x in range(0, 100)],
                         'Exog':[x + random()*10 for x in range(101, 201)]})
df_test = pd.DataFrame({'Act':[x + random()*10 for x in range(101, 201)],
                         'Exog':[200 + random()*10 for x in range(201, 301)]})
df_ret = SARIMAX_model(df_train, df_test)
show_graph(df_train, df_ret, "Seasonal Autoregressive Integrated Moving-Average with Exogenous Regre
ssors (SARIMAX)")
```



RESULT:

Thus, SARIMAX has been executed successfully.

EXNO:8	VECTOR AUTO REGRESSION(VAR)
DATE:28.09.23	

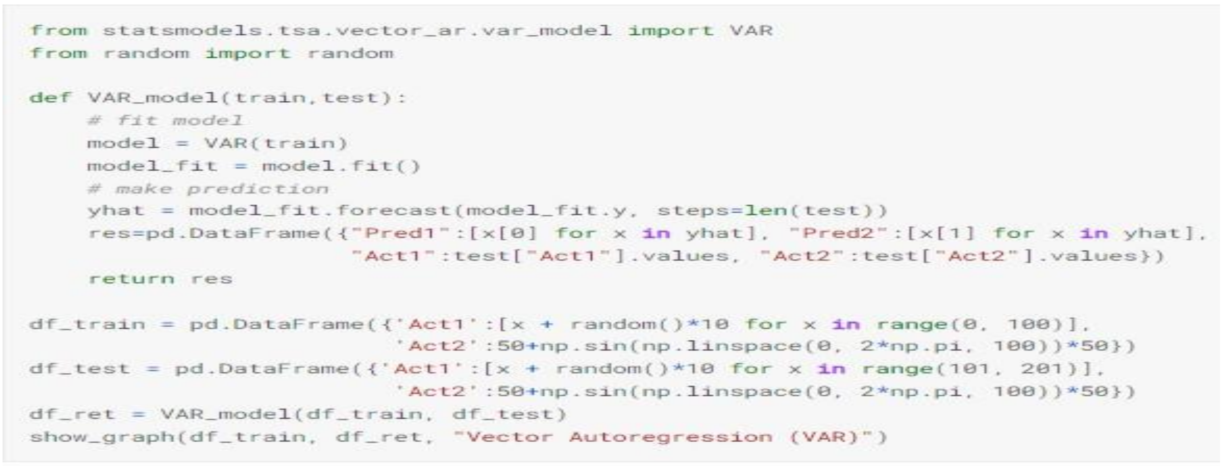
AIM:

To perform vector auto regression in a time series dataset.

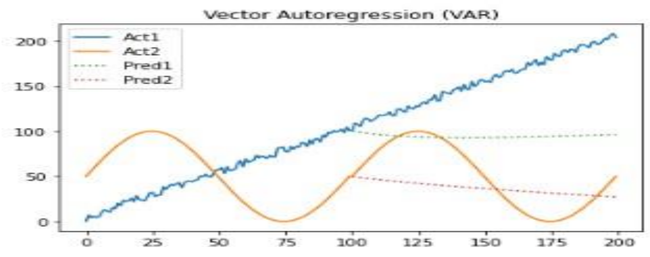
THEORY:

A Vector Auto regression (VAR) model is a statistical time series model used to analyze and forecast multiple related variables simultaneously. Unlike univariate time series models, VAR models consider the interdependencies and dynamic relationships between multiple time series. They are commonly used in economics, finance, and various fields to capture the joint behavior of variables, making them valuable for forecasting and policy analysis. VAR models are specified by lag orders and rely on the assumption that each variable is a linear combination of its past values and the past values of other variables in the system.

PROGRAM:



```python
from statsmodels.tsa.vector_ar.var_model import VAR
from random import random

def VAR_model(train,test):
    # fit model
    model = VAR(train)
    model_fit = model.fit()
    # make prediction
    yhat = model_fit.forecast(model_fit.y, steps=len(test))
    res=pd.DataFrame({"Pred1":[x[0] for x in yhat], "Pred2":[x[1] for x in yhat],
                      "Act1":test["Act1"].values, "Act2":test["Act2"].values})
    return res

df_train = pd.DataFrame({'Act1':[x + random()*10 for x in range(0, 100)],
                         'Act2':50+np.sin(np.linspace(0, 2*np.pi, 100))*50})
df_test = pd.DataFrame({'Act1':[x + random()*10 for x in range(101, 201)],
                        'Act2':50+np.sin(np.linspace(0, 2*np.pi, 100))*50})
df_ret = VAR_model(df_train, df_test)
show_graph(df_train, df_ret, "Vector Autoregression (VAR)")
```



RESULT:

Thus, vector auto regression has been executed successfully.

EXNO:9	VECTOR AUTO REGRESSION MOVING-AVERAGE(VARMA)
DATE:12.10.23	

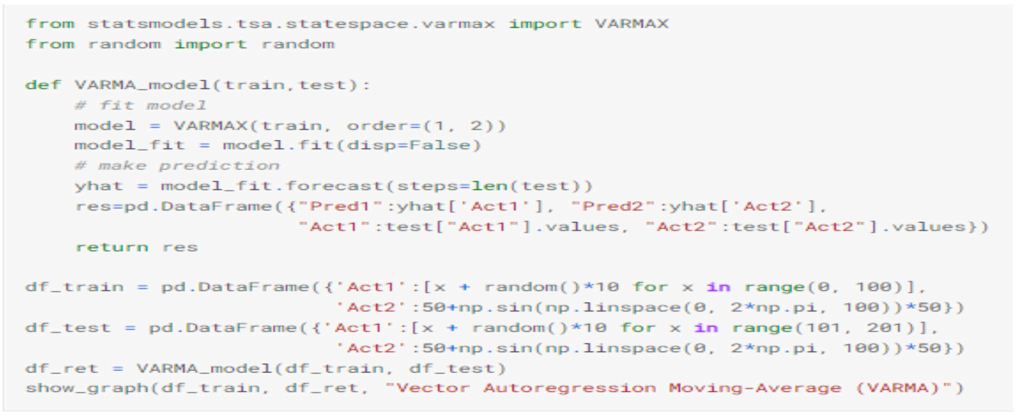
AIM:

To perform vector auto regression moving average in a time series dataset.

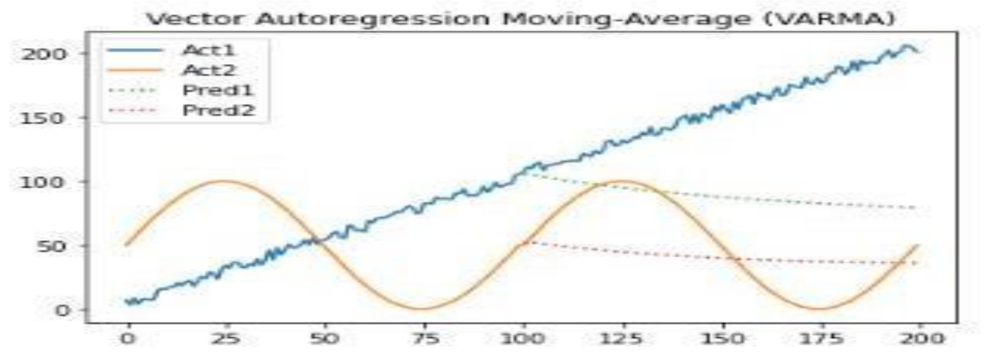
THEORY:

A Vector Auto regression (VAR) model is a statistical time series model used to analyze and forecast multiple related variables simultaneously. Unlike univariate time series models, VAR models consider the interdependencies and dynamic relationships between multiple time series. They are commonly used in economics, finance, and various fields to capture the joint behavior of variables, making them valuable for forecasting and policy analysis. VAR models are specified by lag orders and rely on the assumption that each variable is a linear combination of its past values and the past values of other variables in the system.

PROGRAM:



```python
from statsmodels.tsa.statespace.varmax import VARMAX
from random import random

def VARMA_model(train,test):
    # fit model
    model = VARMAX(train, order=(1, 2))
    model_fit = model.fit(disp=False)
    # make prediction
    yhat = model_fit.forecast(steps=len(test))
    res=pd.DataFrame({"Pred1":yhat['Act1'], "Pred2":yhat['Act2'],
                      "Act1":test["Act1"].values, "Act2":test["Act2"].values})
    return res

df_train = pd.DataFrame({'Act1':[x + random()*10 for x in range(0, 100)],
                         'Act2':50+np.sin(np.linspace(0, 2*np.pi, 100))*50})
df_test = pd.DataFrame({'Act1':[x + random()*10 for x in range(101, 201)],
                        'Act2':50+np.sin(np.linspace(0, 2*np.pi, 100))*50})
df_ret = VARMA_model(df_train, df_test)
show_graph(df_train, df_ret, "Vector Autoregression Moving-Average (VARMA)")
```



RESULT:

Thus, vector auto regression moving average has been executed successfully.

EXNO:10	VECTOR AUTO REGRESSION MOVING-AVERAGE WITH EXOGENOUS REGRESSORS (VARMAX)
DATE:19.10.23	

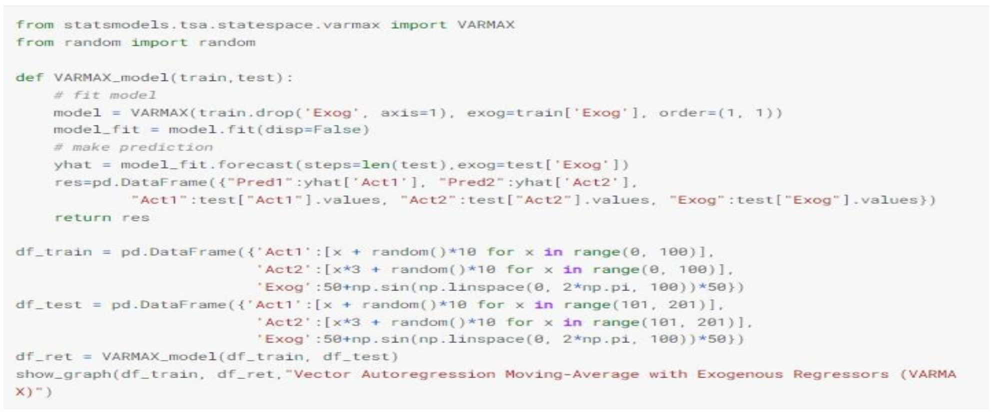
AIM:

To perform VARMAX in a time series dataset.

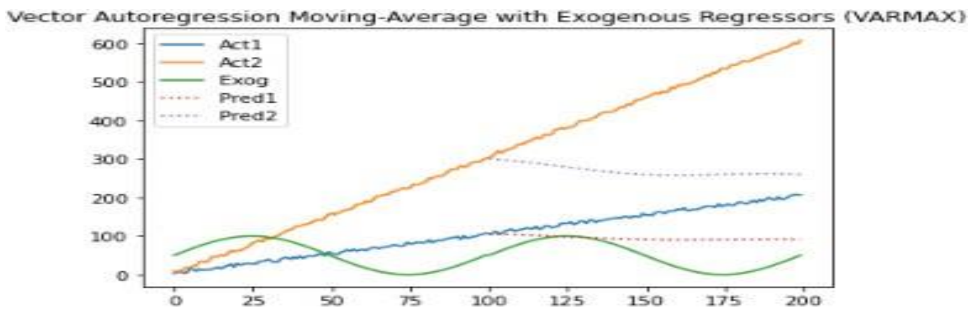
THEORY:

AVARMAX(VectorAutoregressiveMovingAveragewithExogenousVariables)modelisa multivariate time series model used for forecasting and analyzing the relationships between multiple time series variables. It extends the VAR (Vector Autoregressive) model by In coperating both auto regressive and moving average components, and it can also handle exogenous variables. VARMAX models are useful for capturing complex interactions and dependencies among multiple time series, making them valuable in various fields such as economics, finance, and macro econometrics.

PROGRAM:



```
from statsmodels.tsa.statespace.varmax import VARMAX
from random import random

def VARMAX_model(train,test):
    # fit model
    model = VARMAX(train.drop('Exog', axis=1), exog=train['Exog'], order=(1, 1))
    model_fit = model.fit(disp=False)
    # make prediction
    yhat = model_fit.forecast(steps=len(test),exog=test['Exog'])
    res=pd.DataFrame({"Pred1":yhat['Act1'], "Pred2":yhat['Act2'],
            "Act1":test["Act1"].values, "Act2":test["Act2"].values, "Exog":test["Exog"].values})
    return res

df_train = pd.DataFrame({'Act1':[x + random()*10 for x in range(0, 100)],
                         'Act2':[x*3 + random()*10 for x in range(0, 100)],
                         'Exog':50+np.sin(np.linspace(0, 2*np.pi, 100))*50})
df_test = pd.DataFrame({'Act1':[x + random()*10 for x in range(101, 201)],
                         'Act2':[x*3 + random()*10 for x in range(101, 201)],
                         'Exog':50+np.sin(np.linspace(0, 2*np.pi, 100))*50})
df_ret = VARMAX_model(df_train, df_test)
show_graph(df_train, df_ret,"Vector Autoregression Moving-Average with Exogenous Regressors (VARMA
X)")
```



RESULT:

Thus, VARMAX has been executed successfully.

EXNO:11	
DATE:26.10.23	**SIMPLE EXPONENTIAL SMOOTHING(SES)**

AIM:

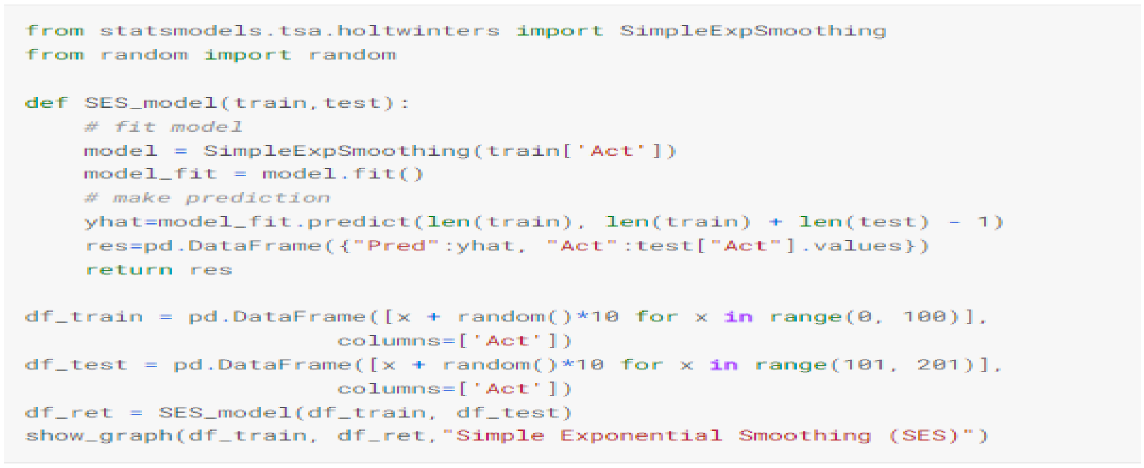
To perform SIMPLE EXPONENTIAL SMOOTHING in a time series dataset.

THEORY:

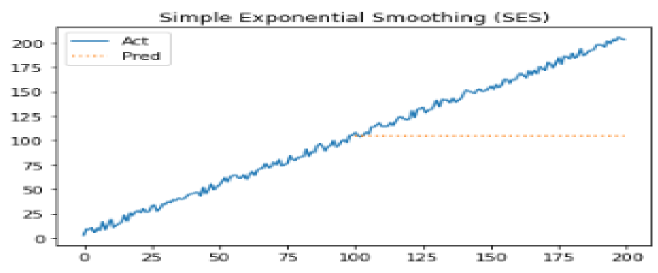
The SES (Simple Exponential Smoothing) model is a time series forecasting method used to make short-term predictions. It's a simple, univariate model that assigns exponentially decreasing weights to past observations, giving more weight to recent data. SES is suitable for time series data with no significant trend or seasonality and is easy to implement.

The forecast for the next period is a weighted average of the previous observation and the Previous forecast. The model is useful for short-term forecasting and can be a basis for more sophisticated methods when data exhibits no complex patterns.

PROGRAM:



```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from random import random

def SES_model(train,test):
    # fit model
    model = SimpleExpSmoothing(train['Act'])
    model_fit = model.fit()
    # make prediction
    yhat=model_fit.predict(len(train), len(train) + len(test) - 1)
    res=pd.DataFrame({"Pred":yhat, "Act":test["Act"].values})
    return res

df_train = pd.DataFrame([x + random()*10 for x in range(0, 100)],
                        columns=['Act'])
df_test = pd.DataFrame([x + random()*10 for x in range(101, 201)],
                        columns=['Act'])
df_ret = SES_model(df_train, df_test)
show_graph(df_train, df_ret,"Simple Exponential Smoothing (SES)")
```



RESULT:

Thus, has SIMPLE EXPONENTIAL SMOOTHING been executed successfully.

EXNO:12	HOLT WINTER’S EXPONENTIAL SMOOTHING(HWES)
DATE:02.11.23	

AIM:

To perform HOLT WINTER’S EXPONENTIAL SMOOTHING in a time series dataset.

THEORY:

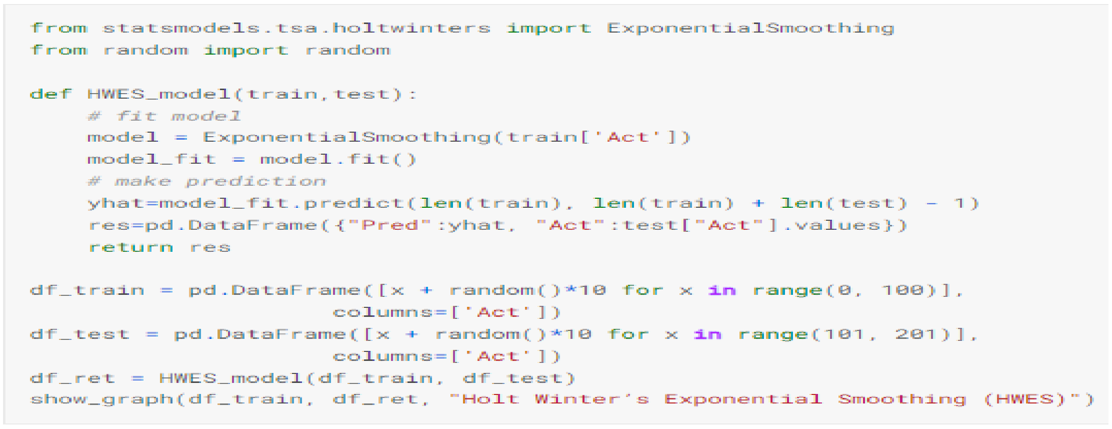
The Holt-Winters Exponential Smoothing(HWES)model is a time series forecasting method that extends the simple exponential smoothing method to handle seasonality and trend in Data .It consists of three components:

Level(α): This represents the underlying value or average of the time series data. It's updated based on the latest observed value.

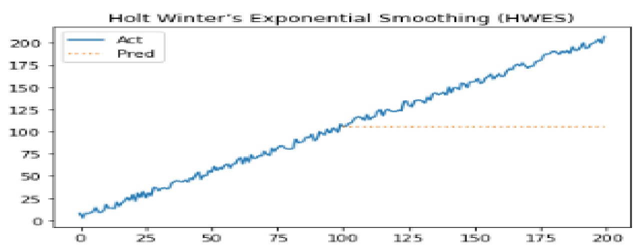
Trend(β):This component captures the direction and rate of change in the time series. It's updated based on the difference between the current level and the previous level.

Seasonality(γ): Seasonal patterns or cycles in the data are captured by this component. It's updated based on past seasonality patterns.

PROGRAM:



```python
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from random import random

def HWES_model(train,test):
    # fit model
    model = ExponentialSmoothing(train['Act'])
    model_fit = model.fit()
    # make prediction
    yhat=model_fit.predict(len(train), len(train) + len(test) - 1)
    res=pd.DataFrame({"Pred":yhat, "Act":test["Act"].values})
    return res

df_train = pd.DataFrame([x + random()*10 for x in range(0, 100)],
                        columns=['Act'])
df_test = pd.DataFrame([x + random()*10 for x in range(101, 201)],
                        columns=['Act'])
df_ret = HWES_model(df_train, df_test)
show_graph(df_train, df_ret, "Holt Winter's Exponential Smoothing (HWES)")
```



RESULT:

Thus, has HOLT WINTER’S EXPONENTIAL SMOOTHING been executed successfully.