In [ ]:

```python
#**********************************************************************
#                   Session 32 -Assignment
#**********************************************************************
#Problem Statement: Students have to make ARIMA model over shampoo sales
#                   data and check the MSE between predicted and actual value


# importing the packages and load the data
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
% matplotlib inline
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import acf,pacf
from sklearn.metrics import mean_squared_error

print("Shampoo Sales Data\n",'-'*80)
shmpData = pd.read_csv('shampoo-sales.csv', header=0,names =['Date','Shampoo-sales'] )
shmpData['FDate'] = '190'+shmpData['Date']
shmpData.drop(['Date'],axis=1,inplace=True)
print(shmpData.head())

```

In [ ]:

```python
print(" Analysing the Data \n",'-'*80)
print(shmpData.info())
print('-'*80)
print(shmpData.describe())
print('-'*80)
print(shmpData.isnull().sum())
print('-'*80)
shmpData.dropna(inplace=True)
print(shmpData.isnull().sum())
print('-'*80)

print(shmpData.shape)
```

In [ ]:

```
print(" Making the Date as the Index for the DataFrame\n",'-'*80)
shmpData['FDate'] = pd.to_datetime(shmpData['FDate'])
shmpData.set_index('FDate',inplace=True)
print(shmpData.head())


```

In [ ]:

```
#Step1 : Visualising the Data

%matplotlib inline
shmpData.plot(figsize=(6,4))
plt.xlabel("Date")
plt.ylabel("Shampoo Sales")

print(" We Observe from the plot , that there is upward trend ")
```

In [ ]:

```
# We observe from the graph above that there is a upward trend in the shampoo sale
# we need to check for stationarity

# Defining the the rolling mean and std deviation and Dickey_fuller to see the plot is stationary
from statsmodels.tsa.stattools import adfuller

# get_mean_var function gives the mean and the variance for the values
def get_mean_var(series, no_of_samples):
    split_size = int(len(series) / no_of_samples)
    start = 0
    for i in range(no_of_samples):
        sample_series = series[i*split_size:(i+1)*split_size]
        #print(sample_series,"\n")
        print('Mean= %.2f, Variance= %.2f' % (sample_series.mean(), sample_series.var()))


def plot_rolling_statistics(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')

    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.figure(figsize=(20,40))
    plt.show()

# to perform Dickey fuller test

def dickey_fuller_test(timeseries):
    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries['Shampoo-sales'],autolag='AIC')
    #https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html

    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

```

```
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)




def test_stationarity(timeseries):
    get_mean_var(timeseries,10)
    plot_rolling_statistics(timeseries)
    dickey_fuller_test(timeseries)


```

In [ ]:

```
# To see if the indexed Data is stationary by calling the above functions
test_stationarity(shmpData)

print('-'*80,'''\n From the Stationarity Test we Observe
1. P Value is big, should be 0
2. T stats should be Less than Critical Val , bu it is opposite for all confidence
hence we reject null , saying it is not stationary

''')
```

In [ ]:

```
# Step 2 : Stationarizing the Time Series
import numpy as np
# Estimating the trend
shmpData_logScale = np.log(shmpData)

print(" Plotting Log Scale for Shampoo Data\n",'-'*80)
plt.plot(shmpData_logScale)
print(shmpData_logScale.head())

```

In [ ]:
```
test_stationarity(shmpData_logScale)
print('-'*80,'\nDurbin Watson Test :',sm.stats.durbin_watson(shmpData_logScale))

print('-'*80,'''\n From the Stationarity Test we Observe
1. P Value is big, should be 0
2. T stats should be Less than Critical Val , bu it is opposite for all confidence
hence we reject null , saying it is not stationary

''')
```

In [ ]:
```
# DataSet with moving average
rolmean1 = shmpData_logScale.rolling(window=12).mean()
#print(rolmean)
shmpData_logScaleMinusMovingAvg = shmpData_logScale - rolmean1

# we remove the null values
shmpData_logScaleMinusMovingAvg.dropna(inplace=True)
shmpData_logScaleMinusMovingAvg.head()
```

In [ ]:
```
test_stationarity(shmpData_logScaleMinusMovingAvg)
print('-'*80,'\nDurbin Watson Test :',sm.stats.durbin_watson(shmpData_logScaleMinusMovingAvg))

print('-'*80,'''\n From the Stationarity Test we Observe
1. P Value is big, should be 0
2. T stats should be Less than Critical Val , bu it is opposite for all confidence
hence we reject null , saying it is not stationary

''')
```

In [ ]:
```
# We Try Shifting the series , trying the Difference
shmpData_logScaleDiffShifting = shmpData_logScaleMinusMovingAvg - shmpData_logScaleMinusMovingAvg.shift()


shmpData_logScaleDiffShifting.dropna(inplace=True)
plt.plot(shmpData_logScaleDiffShifting)
```

In [ ]:

```
test_stationarity(shmpData_logScaleDiffShifting)

print('-'*80,'\nDurbin Watson Test :',sm.stats.durbin_watson(shmpData_logScaleDiffShifting))
print('-'*80,'''\n From the Stationarity Test we Observe
1. P Value is equal  0
2. T stats is Less than Critical Val for 90% and 95% confidence, we accept the null hypothesis
that time series is Stationary

''')
```

In [ ]:

```
# test stats is less than 10% and 5% confidence

from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(shmpData_logScaleDiffShifting)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(shmpData_logScaleDiffShifting,label="Original")
plt.legend(loc="best")

plt.subplot(412)
plt.plot(trend,label="Trend")
plt.legend(loc="best")

plt.subplot(413)
plt.plot(trend,label="seasonal")
plt.legend(loc="best")

plt.subplot(414)
plt.plot(residual,label="Residual")
plt.legend(loc="best")

plt.tight_layout()

```

In [ ]:
```
decomposeLogData = residual
decomposeLogData.dropna(inplace=True)
test_stationarity(decomposeLogData)
#print(decomposeLogData)
```

In [ ]:
```
# acf and pacf
from statsmodels.tsa.stattools import acf,pacf
lag_acf = acf(shmpData_logScaleDiffShifting,nlags=20)
lag_pacf = pacf(shmpData_logScaleDiffShifting,nlags=20,method="ols")

#acf- gives  q value where it touches the top confidnce level
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(shmpData_logScaleDiffShifting)),linestyle='--',color='gray')
plt.axhline(y= 1.96/np.sqrt(len(shmpData_logScaleDiffShifting)),linestyle='--',color='gray')
plt.title("AutoCorrelation")

#pacf -gives  p value where it touches the top confidnce level
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(shmpData_logScaleDiffShifting)),linestyle='--',color='gray')
plt.axhline(y= 1.96/np.sqrt(len(shmpData_logScaleDiffShifting)),linestyle='--',color='gray')
plt.title("Partial AutoCorrelation")
plt.tight_layout()


```

In [ ]:

```
%matplotlib inline
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))

ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(shmpData_logScaleDiffShifting.values.squeeze(), lags=20, ax=ax1)

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(shmpData_logScaleDiffShifting.values.squeeze(), lags=20, ax=ax2)
```

In [ ]:

```
# We observe that p and q here is 1
# Ar model

# check is it p=1 and q=1 as per the graph
# also when we using arima model , the sites mention to put the logged Dataframe , but we have achieved
# stationarity to DiffShiting
# For Moving Average model we get least least RSS

model = ARIMA(shmpData_logScaleDiffShifting,order=(0,1,1))
#model = ARIMA(shmpData_logScale,order=(1,1,1))
results_ARIMA = model.fit()
plt.plot(shmpData_logScaleDiffShifting)
plt.plot(results_ARIMA.fittedvalues,color="red")
plt.title('RSS: %.4f' % sum((results_ARIMA.fittedvalues-shmpData_logScaleDiffShifting['Shampoo-sales']).dropna())**2
print("Plotting ARIMA Model")

```

In [ ]:
```
# fitting the Arima Model and Predictions

predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues,copy=True)

# taking cumulativeSum
predictions_ARIMA_diff.cumsum = predictions_ARIMA_diff.cumsum()
# print(predictions_ARIMA_diff.cumsum.head())
# print('-'*80)


# predictions for the fitted values
#print(shmpData_logScale.head())
predictions_ARIMA_log= pd.Series(shmpData_logScale['Shampoo-sales'].values,index= shmpData_logScale.index)

rolmean1.fillna(0,inplace=True)
# print(rolmean1['Shampoo-sales'].values)
# print(predictions_ARIMA_diff.cumsum)

# #predictions_ARIMA_mean = predictions_ARIMA_log.add(rolmean1,fill_value=0)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff.cumsum,fill_value=0)
#predictions_ARIMA_Avg = predictions_ARIMA_log.add(rolmean1['Shampoo-sales'].values)
# predictions_ARIMA_log.head()
# print(predictions_ARIMA_Avg.head())


predictions_ARIMA = np.exp(predictions_ARIMA_log)
print(" Original Shampoo Data")
print(shmpData.head())
print(" Predicted Shampoo Data ")
print(predictions_ARIMA.head())

plt.plot(shmpData)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(((predictions_ARIMA.values- shmpData.values)**2).sum()/len(shmpData)))

```

In [ ]:
```
results_ARIMA.plot_predict(1,60)
```

In [ ]:

```python

def mean_forecast_err(y, yhat):
    return y.sub(yhat).mean()

def mean_absolute_err(y, yhat):
    return np.mean((np.abs(y.sub(yhat).mean()) / yhat)) # or percent error = * 100



print("Mean ForeCast Error -  MFE = ",mean_forecast_err(shmpData['Shampoo-sales'], predictions_ARIMA))
print("Mean Absolute Error - MAE = ",mean_absolute_err(shmpData['Shampoo-sales'], predictions_ARIMA))
print("Mean Square Error - MSE = ",mean_squared_error(shmpData['Shampoo-sales'], predictions_ARIMA))

```