Aim

To analyze and forecast monthly sales data using ARIMA and SARIMAX models.

Objective

- Clean and preprocess the sales data.
- Test for stationarity and make the data stationary.
- Fit ARIMA and SARIMAX models to the data.
- Forecast future sales and visualize the results.

Summary

This project involves analyzing a monthly sales dataset to forecast future sales using time series analysis techniques. The data is preprocessed to handle missing values and ensure stationarity. ARIMA and SARIMAX models are then fitted to the data, and forecasts are generated for future time periods. The results are visualized to assess the model's performance.

Tools and Libraries Used

- **Python**
- **NumPy**
- **Pandas**
- **Matplotlib**
- **Statsmodels**

Procedure (Explain the Code)

1. **Import Libraries and Load Data**:
    - Imported necessary libraries for data manipulation and visualization.
    - Loaded the sales dataset into a Pandas DataFrame.
2. **Data Cleaning**:
    - Renamed columns for clarity.
    - Dropped the last two rows due to data inconsistency.
    - Converted the 'Month' column to datetime format and set it as the index.
3. **Data Visualization**:
    - Plotted the sales data to visualize trends and patterns.
4. **Stationarity Test**:
    - Conducted the Augmented Dickey-Fuller (ADF) test to check if the data is stationary.

- Differenced the data to achieve stationarity.

5. **Autocorrelation and Partial Autocorrelation**:
    - Plotted the autocorrelation and partial autocorrelation to determine the order of the ARIMA model.
6. **ARIMA Model**:
    - Fitted an ARIMA model to the data.
    - Generated forecasts using the ARIMA model and visualized the results.
7. **SARIMAX Model**:
    - Fitted a SARIMAX model to account for seasonality in the data.
    - Generated forecasts using the SARIMAX model and visualized the results.
8. **Future Forecasting**:
    - Extended the date range and generated future forecasts using the SARIMAX model.

CODE:

```
import numpy as np

import pandas as pd


import matplotlib.pyplot as plt

%matplotlib inline

df=pd.read_csv('/content/drive/MyDrive/sales_dataset/perrin-freres-monthly-champagne-.csv')

df.head()

df.tail()

## Cleaning up the data

df.columns=["Month","Sales"]

df.head()

## Drop last 2 rows

df.drop(106,axis=0,inplace=True)

df.tail()

df.drop(105,axis=0,inplace=True)

df.tail()
```

```
# Convert Month into Datetime
df['Month']=pd.to_datetime(df['Month'])
df.head()
df.set_index('Month',inplace=True)
df.head()
df.describe()
#Visualize the Data
df.plot()
### Testing For Stationarity


from statsmodels.tsa.stattools import adfuller
test_result=adfuller(df['Sales'])
#Ho: It is non stationary
#H1: It is stationary


def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")
adfuller_test(df['Sales'])
#Differencing
df['Sales First Difference'] = df['Sales'] - df['Sales'].shift(1)
```

```
df['Sales'].shift(1)
df['Seasonal First Difference']=df['Sales']-df['Sales'].shift(12)
df.head(14)
## Again test dickey fuller test
adfuller_test(df['Seasonal First Difference'].dropna())
df['Seasonal First Difference'].plot()
#Auto Regressive Model
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df['Sales'])
plt.show()
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['Seasonal First Difference'].iloc[13:], lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['Seasonal First Difference'].iloc[13:], lags=40, ax=ax2)
plt.show()
# For non-seasonal data
#p=1, d=1, q=0 or 1
from statsmodels.tsa.arima.model import ARIMA
model=ARIMA(df['Sales'],order=(1,1,1))
model_fit=model.fit()
model_fit.summary()
df['forecast']=model_fit.predict(start=90,end=103,dynamic=True)
df[['Sales','forecast']].plot(figsize=(12,8))
model=sm.tsa.statespace.SARIMAX(df['Sales'],order=(1, 1, 1),seasonal_order=(1,1,1,12))
results=model.fit()
```

```
results.summary()

df['forecast']=results.predict(start=90,end=103,dynamic=True)

df[['Sales','forecast']].plot(figsize=(12,8))

from pandas.tseries.offsets import DateOffset

future_dates=[df.index[-1]+ DateOffset(months=x)for x in range(0,24)]

future_datest_df=pd.DataFrame(index=future_dates[1:],columns=df.columns)

future_datest_df.tail()

future_df=pd.concat([df,future_datest_df])

future_df['forecast'] = results.predict(start = 104, end = 120, dynamic= True)

future_df[['Sales', 'forecast']].plot(figsize=(12, 8))
```

Highlights

- **Data Cleaning**: Addressed inconsistencies in the dataset by dropping erroneous rows and converting data types.
- **Stationarity**: Applied differencing techniques to make the data stationary, which is crucial for time series modeling.
- **Modeling**: Utilized both ARIMA and SARIMAX models to capture non-seasonal and seasonal patterns in the data.
- **Visualization**: Plotted actual and forecasted sales to visually assess model performance.

Conclusion

The project successfully demonstrated the process of forecasting monthly sales using ARIMA and SARIMAX models. By making the data stationary and carefully selecting model parameters, accurate forecasts were generated. The visualizations provided clear insights into the sales trends and the effectiveness of the models.