Certainly! Below is an outline for preprocessing the dataset, implementing ARIMA and SARIMA models, and generating insights. We'll assume you have the dataset in a CSV format named `tcs.csv`.

### Step 1: Preprocessing the Data

1. \*\*Import Necessary Libraries\*\*:

```python

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from sklearn.metrics import mean\_squared\_error

```

2. \*\*Load the Data\*\*:

```python

# Load dataset

df = pd.read\_csv('tcs.csv', parse\_dates=['Date'], index\_col='Date')

```

3. \*\*Data Inspection\*\*:

```python

print(df.head())

print(df.info())

```

4. \*\*Check for Missing Values\*\*:

```python

print(df.isnull().sum())

```

5. \*\*Visualize the Time Series\*\*:

```python

plt.figure(figsize=(12, 6))

plt.plot(df['Close Price'])

plt.title('TCS Close Price Over Time')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.show()

```

6. \*\*Stationarity Check\*\*:

Use Augmented Dickey-Fuller test to check for stationarity.

```python

from statsmodels.tsa.stattools import adfuller

result = adfuller(df['Close Price'])

print('ADF Statistic:', result[0])

print('p-value:', result[1])

```

7. \*\*Differencing\*\*:

If the series is non-stationary, apply differencing.

```python

df['Close Price Diff'] = df['Close Price'].diff().dropna()

```

### Step 2: Build ARIMA Model

1. \*\*Identify p, d, q Parameters\*\*:

Use ACF and PACF plots.

```python

plot\_acf(df['Close Price'].dropna(), lags=20)

plt.show()

plot\_pacf(df['Close Price'].dropna(), lags=20)

plt.show()

```

2. \*\*Build ARIMA Model\*\*:

```python

model = ARIMA(df['Close Price'], order=(p, d, q))

model\_fit = model.fit()

print(model\_fit.summary())

```

3. \*\*Make Predictions\*\*:

```python

forecast = model\_fit.forecast(steps=30)

plt.plot(df['Close Price'], label='Historical')

plt.plot(forecast, label='Forecast', color='red')

plt.legend()

plt.show()

```

### Step 3: Build SARIMA Model

1. \*\*Build SARIMA Model\*\*:

```python

model\_sarima = SARIMAX(df['Close Price'], order=(p, d, q), seasonal\_order=(P, D, Q, s))

model\_sarima\_fit = model\_sarima.fit()

print(model\_sarima\_fit.summary())

```

2. \*\*Make Predictions\*\*:

```python

forecast\_sarima = model\_sarima\_fit.forecast(steps=30)

plt.plot(df['Close Price'], label='Historical')

plt.plot(forecast\_sarima, label='Forecast', color='green')

plt.legend()

plt.show()

```

### Step 4: Insights

1. \*\*Trend Analysis\*\*:

- Assess long-term trends based on the model.

2. \*\*Seasonality\*\*:

- Identify seasonal patterns in the SARIMA model.

3. \*\*Error Metrics\*\*:

- Compare predictions against actual values using RMSE or MSE.

```python

rmse = np.sqrt(mean\_squared\_error(actual\_values, predicted\_values))

print(f'RMSE: {rmse}')

```

4. \*\*Investment Insights\*\*:

- Discuss potential buy/sell signals based on forecast trends.

### Final Thoughts

1. \*\*Model Validation\*\*:

- Split your data into training and testing sets to validate your models.

2. \*\*Refinement\*\*:

- Tune model parameters based on performance metrics.

Feel free to modify the parameters (p, d, q, P, D, Q, s) based on your exploratory analysis results! If you need further assistance with a specific part, let me know!

Certainly! Below is a complete A to Z code example for preprocessing the dataset, applying both ARIMA and SARIMA models, and providing insights at each step.

### Complete Code Example

```python

# Import Necessary Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from statsmodels.tsa.stattools import adfuller

from sklearn.metrics import mean\_squared\_error

# Load the dataset

df = pd.read\_csv('tcs.csv', parse\_dates=['Date'], index\_col='Date')

# Display the first few rows and info

print(df.head())

print(df.info())

# Check for missing values

print("Missing values:\n", df.isnull().sum())

# Visualize the Close Price

plt.figure(figsize=(12, 6))

plt.plot(df['Close Price'], label='Close Price')

plt.title('TCS Close Price Over Time')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.legend()

plt.show()

# Check stationarity

result = adfuller(df['Close Price'])

print('ADF Statistic:', result[0])

print('p-value:', result[1])

# If p-value > 0.05, the series is non-stationary. We can difference it.

df['Close Price Diff'] = df['Close Price'].diff().dropna()

# Plot ACF and PACF

plt.figure(figsize=(12, 6))

plt.subplot(121)

plot\_acf(df['Close Price'].dropna(), lags=20, ax=plt.gca())

plt.title('ACF of Close Price')

plt.subplot(122)

plot\_pacf(df['Close Price'].dropna(), lags=20, ax=plt.gca())

plt.title('PACF of Close Price')

plt.tight\_layout()

plt.show()

# Define p, d, q based on ACF and PACF plots

p = 1 # Example value based on PACF

d = 1 # Differencing

q = 1 # Example value based on ACF

# Build and fit ARIMA model

model = ARIMA(df['Close Price'], order=(p, d, q))

model\_fit = model.fit()

print(model\_fit.summary())

# Forecasting using ARIMA

forecast = model\_fit.forecast(steps=30)

plt.figure(figsize=(12, 6))

plt.plot(df['Close Price'], label='Historical')

plt.plot(pd.date\_range(start=df.index[-1], periods=30, freq='B'), forecast, label='Forecast', color='red')

plt.title('ARIMA Forecast')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.legend()

plt.show()

# Calculate RMSE for ARIMA

actual\_values = df['Close Price'].iloc[-30:] # Last 30 actual values

predicted\_values = forecast

rmse\_arima = np.sqrt(mean\_squared\_error(actual\_values, predicted\_values))

print(f'ARIMA RMSE: {rmse\_arima}')

# Build and fit SARIMA model

P = 1 # Seasonal AR

D = 1 # Seasonal Differencing

Q = 1 # Seasonal MA

s = 12 # Seasonal period

model\_sarima = SARIMAX(df['Close Price'], order=(p, d, q), seasonal\_order=(P, D, Q, s))

model\_sarima\_fit = model\_sarima.fit()

print(model\_sarima\_fit.summary())

# Forecasting using SARIMA

forecast\_sarima = model\_sarima\_fit.forecast(steps=30)

plt.figure(figsize=(12, 6))

plt.plot(df['Close Price'], label='Historical')

plt.plot(pd.date\_range(start=df.index[-1], periods=30, freq='B'), forecast\_sarima, label='Forecast', color='green')

plt.title('SARIMA Forecast')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.legend()

plt.show()

# Calculate RMSE for SARIMA

rmse\_sarima = np.sqrt(mean\_squared\_error(actual\_values, forecast\_sarima))

print(f'SARIMA RMSE: {rmse\_sarima}')

# Insights

print("\nInsights:")

if result[1] > 0.05:

print("The series is non-stationary; differencing was applied.")

print(f"ARIMA Model RMSE: {rmse\_arima}")

print(f"SARIMA Model RMSE: {rmse\_sarima}")

if rmse\_arima < rmse\_sarima:

print("The ARIMA model performed better based on RMSE.")

else:

print("The SARIMA model performed better based on RMSE.")

# Overall, if forecasts trend upwards, it may indicate a buying opportunity.

```

### Explanation of Each Section

1. \*\*Imports\*\*: The necessary libraries for data manipulation, time series analysis, and visualization are imported.

2. \*\*Load Data\*\*: The dataset is loaded, and the date column is set as the index.

3. \*\*Initial Inspection\*\*: Displays the first few rows and checks for missing values.

4. \*\*Visualization\*\*: The historical close prices are plotted to understand trends and patterns.

5. \*\*Stationarity Check\*\*: The Augmented Dickey-Fuller test checks if the series is stationary.

6. \*\*Differencing\*\*: If the series is non-stationary, it is differenced.

7. \*\*ACF and PACF Plots\*\*: These help identify the appropriate parameters (p, q) for the ARIMA model.

8. \*\*ARIMA Model\*\*: The ARIMA model is built and fit to the data, and a forecast is generated.

9. \*\*RMSE Calculation\*\*: The root mean squared error is calculated for the ARIMA model.

10. \*\*SARIMA Model\*\*: Similar steps are followed for the SARIMA model, which incorporates seasonal parameters.

11. \*\*Final Insights\*\*: A summary of the results, including model performance metrics and interpretation.

### Insights to Consider

- The ADF test result helps in understanding the stationarity of the series.

- RMSE provides a measure of model accuracy.

- Comparing both models gives insights into which model better captures the time series dynamics.

- Forecast trends indicate potential market behavior.

You can adjust the parameters based on your exploratory analysis and the results you get. Let me know if you need any more specific insights or modifications!

Bilkul! Aapko Python ka istemal karte hue ARIMA model me stationarity check karne aur stationary banane ka code deta hun. Hum `pandas`, `numpy`, `statsmodels`, aur `matplotlib` libraries ka istemal karenge. Yahan par step by step code diya gaya hai.

### Step 1: Libraries Install Karna

```bash

pip install pandas numpy statsmodels matplotlib

```

### Step 2: Data Load Karna

```python

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Sample data load karna (aap apne data se replace kar sakte hain)

# Yahan par ek random time series create karte hain

np.random.seed(42)

data = np.random.randn(100).cumsum() # Cumulative sum for trend

time\_series = pd.Series(data)

# Data plot karna

plt.figure(figsize=(10, 5))

plt.plot(time\_series)

plt.title('Time Series Data')

plt.xlabel('Time')

plt.ylabel('Value')

plt.show()

```

### Step 3: Stationarity Check Karna

#### Visual Inspection

```python

# ACF Plot karna

from statsmodels.graphics.tsaplots import plot\_acf

plt.figure(figsize=(10, 5))

plot\_acf(time\_series)

plt.title('ACF Plot')

plt.show()

```

#### ADF Test

```python

from statsmodels.tsa.stattools import adfuller

adf\_result = adfuller(time\_series)

print(f'ADF Statistic: {adf\_result[0]}')

print(f'p-value: {adf\_result[1]}')

if adf\_result[1] < 0.05:

print("Series is stationary (reject H0)")

else:

print("Series is non-stationary (fail to reject H0)")

```

### Step 4: Stationarity Banane ke Liye Differencing

```python

# First differencing

differenced\_series = time\_series.diff().dropna()

# Differenced data plot karna

plt.figure(figsize=(10, 5))

plt.plot(differenced\_series)

plt.title('Differenced Time Series Data')

plt.xlabel('Time')

plt.ylabel('Value')

plt.show()

# ADF Test on differenced data

adf\_result\_diff = adfuller(differenced\_series)

print(f'ADF Statistic (Differenced): {adf\_result\_diff[0]}')

print(f'p-value (Differenced): {adf\_result\_diff[1]}')

if adf\_result\_diff[1] < 0.05:

print("Differenced series is stationary (reject H0)")

else:

print("Differenced series is non-stationary (fail to reject H0)")

```

### Step 5: Log Transformation (Agar zarurat ho)

```python

# Log transformation

log\_series = np.log(time\_series)

# Log data plot karna

plt.figure(figsize=(10, 5))

plt.plot(log\_series)

plt.title('Log Transformed Time Series Data')

plt.xlabel('Time')

plt.ylabel('Value')

plt.show()

# ADF Test on log-transformed data

adf\_result\_log = adfuller(log\_series)

print(f'ADF Statistic (Log Transformed): {adf\_result\_log[0]}')

print(f'p-value (Log Transformed): {adf\_result\_log[1]}')

```

### Summary

Is code ke zariye aap:

1. Data load karna.

2. Stationarity check karna using visual methods and ADF test.

3. Differencing ya log transformation ka istemal kar ke series ko stationary banana.

Agar aapko kisi specific section ya function me detail chahiye ya kuch aur karna hai, to bata sakte hain!