2347138_C2

July 15, 2024

1 Q1

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
data=pd.read_csv("E:/4TH_sem/ADA/NIFTY 50-15-01-2024-to-15-07-2024.csv")
data.head()
```

[ ]:
```
          Date      Open      High       Low     Close  Shares Traded  \
0  15-JAN-2024  22053.15  22115.55  21963.55  22097.45      345543523
1  16-JAN-2024  22080.50  22124.15  21969.80  22032.30      292433764
2  17-JAN-2024  21647.25  21851.50  21550.45  21571.95      455999867
3  18-JAN-2024  21414.20  21539.40  21285.55  21462.25      387341268
4  19-JAN-2024  21615.20  21670.60  21575.00  21622.40      343055124

   Turnover (  Cr)
0         29523.15
1         24435.94
2         47533.44
3         39718.84
4         34429.24
```

[ ]:
```
data.describe()
```

[ ]:
```
               Open          High           Low         Close  Shares Traded  \
count    123.000000    123.000000    123.000000    123.000000   1.230000e+02
mean   22547.605691  22649.634146  22415.028862  22539.645122   3.388198e+08
std      772.378182    757.623411    784.705204    776.982588   1.162314e+08
min    21185.250000  21459.000000  21137.200000  21238.800000   1.906457e+07
25%    22024.125000  22129.050000  21910.750000  22017.650000   2.771433e+08
50%    22385.700000  22476.450000  22259.550000  22368.000000   3.258235e+08
75%    22782.475000  22951.875000  22658.150000  22787.600000   3.742559e+08
max    24459.850000  24592.200000  24331.900000  24502.150000   1.006105e+09

       Turnover (  Cr)
count       123.000000
mean      33701.596504
```

```
std       11800.683576
min        1572.770000
25%       27745.010000
50%       31258.400000
75%       38786.215000
max       93786.440000
```

[ ]:
```python
import pandas as pd

data=pd.read_csv('E:/4TH_sem/ADA/NIFTY 50-15-01-2024-to-15-07-2024.csv')

data.columns = data.columns.str.strip()

print("Column names after stripping spaces:", data.columns)

data['Date'] = pd.to_datetime(data['Date'], format='%d-%b-%Y')

data.set_index('Date', inplace=True)
missing_values = data.isnull().sum()
print(data.info())
print("Missing Values are :",missing_values)
```

```
Column names after stripping spaces: Index(['Date', 'Open', 'High', 'Low',
'Close', 'Shares Traded',
       'Turnover (  Cr)'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 123 entries, 2024-01-15 to 2024-07-12
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Open             123 non-null    float64
 1   High             123 non-null    float64
 2   Low              123 non-null    float64
 3   Close            123 non-null    float64
 4   Shares Traded    123 non-null    int64
 5   Turnover (  Cr)  123 non-null    float64
dtypes: float64(5), int64(1)
memory usage: 6.7 KB
None
Missing Values are : Open              0
High               0
Low                0
Close              0
Shares Traded      0
Turnover (  Cr)    0
dtype: int64
```

[ ]:
```python
data.dtypes
```

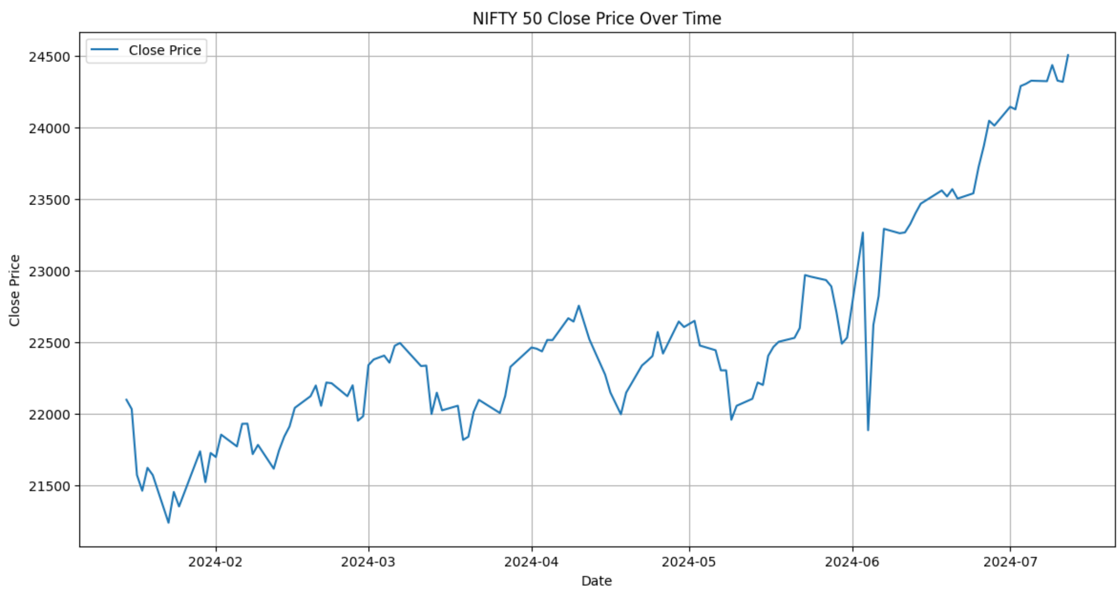
[ ]:
```
Open                 float64
High                 float64
Low                  float64
Close                float64
Shares Traded          int64
Turnover (  Cr)     float64
dtype: object
```

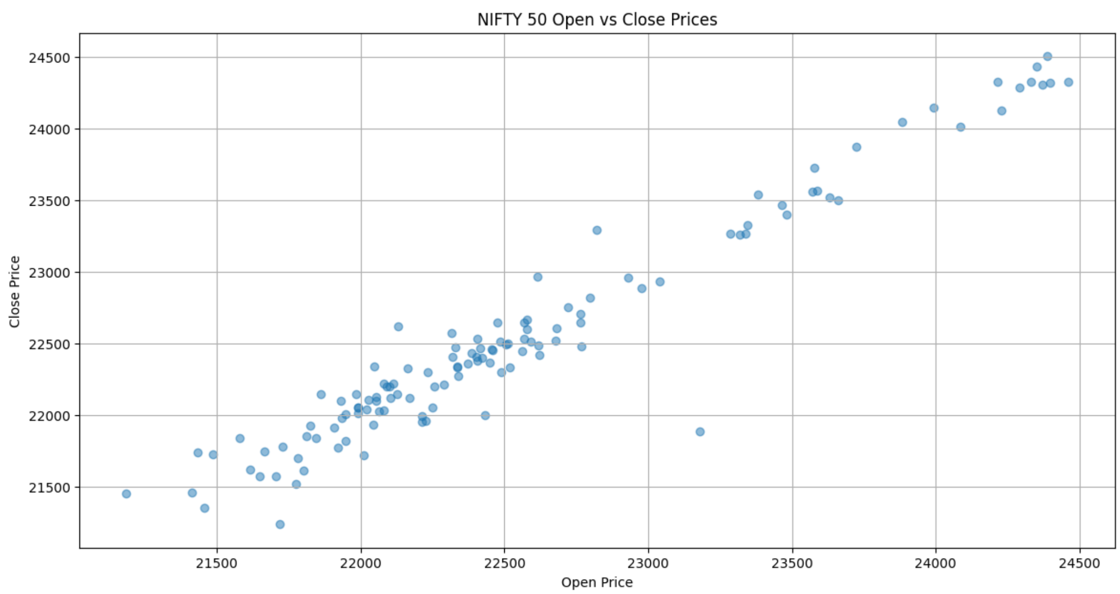
[ ]:
```python
data_ffill = data.ffill()

values_after_ffill = data_ffill.isnull().sum()
print("Missing values after fill imputation:")
print(values_after_ffill)
```

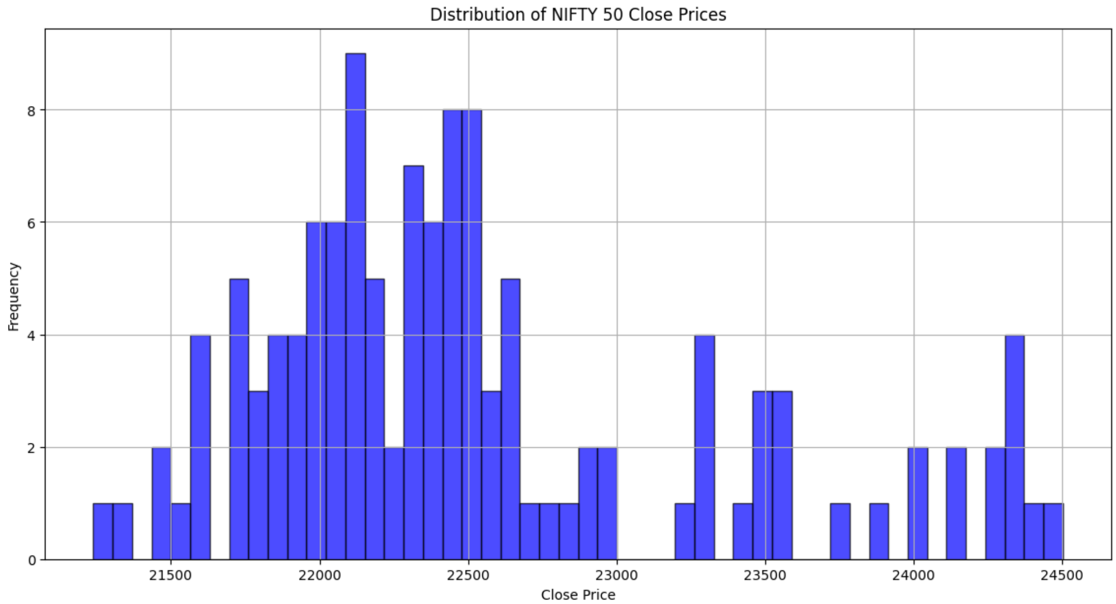
```
Missing values after forward fill imputation:
Open                0
High                0
Low                 0
Close               0
Shares Traded       0
Turnover (  Cr)     0
dtype: int64
```

[ ]:
```python
# Line chart of the 'Close' price
plt.figure(figsize=(14, 7))
plt.plot(data_ffill.index, data_ffill['Close'], label='Close Price')
plt.title('NIFTY 50 Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.grid(True)
plt.show()

# Scatter plot of 'Open' vs 'Close' prices
plt.figure(figsize=(14, 7))
plt.scatter(data_ffill['Open'], data_ffill['Close'], alpha=0.5)
plt.title('NIFTY 50 Open vs Close Prices')
plt.xlabel('Open Price')
plt.ylabel('Close Price')
plt.grid(True)
plt.show()

# Histogram of the 'Close' price
plt.figure(figsize=(14, 7))
plt.hist(data_ffill['Close'], bins=50, alpha=0.7, color='b', edgecolor='black')
plt.title('Distribution of NIFTY 50 Close Prices')
```

```
plt.xlabel('Close Price')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```







1.1 Interpretation:

2 Loading and Cleaning the Data:

Column Names: The initial step of stripping leading and trailing spaces from column names ensures that each column can be referenced correctly. This prevents errors related to column name mismatches due to hidden spaces. Date Conversion: Converting the 'Date' column to a datetime format is essential for time series analysis. This transformation allows the data to be indexed by date, enabling chronological data manipulation and analysis. Setting the 'Date' Column as Index:

Purpose: Setting the 'Date' column as the index transforms the DataFrame into a time series format. This allows for time-based operations such as resampling, rolling averages, and time series forecasting. Outcome: The DataFrame now treats dates as the primary key, facilitating easier access to time-specific data points and enabling sophisticated time series analysis techniques.

3 Missing Values:

Identification: Checking for missing values helps identify any gaps in the data that could affect analysis accuracy. Handling Missing Values: Imputing missing values using forward fill (or other methods) ensures continuity in the dataset. Forward fill is particularly useful for financial data, as it assumes the most recent available value is the best estimate for a missing data point.

4 Visualizations:

5 1. Line Chart of ‘Close’ Price:

Purpose: The line chart displays the trend of the NIFTY 50 closing prices over time. Interpretation: Trend Analysis: A rising line indicates an upward trend, suggesting an overall increase in the index value, while a falling line indicates a downward trend.

6 2. Scatter Plot of ‘Open’ vs ‘Close’ Prices:

Purpose: The scatter plot shows the relationship between the opening and closing prices of the NIFTY 50 index. Interpretation: Correlation: Points clustering around the line y=x indicate a strong correlation between open and close prices, suggesting that the index often closes near its opening price.

Histogram of ‘Close’ Prices: Purpose: The histogram displays the distribution of closing prices.

Interpretation: Frequency Distribution: Peaks in the histogram indicate the most common closing price ranges.

Data Preparation: Proper data cleaning and preparation, such as removing spaces in column names and converting date formats, are critical for accurate analysis. Handling Missing Values: Forward fill imputation is appropriate for financial time series data as it maintains the continuity and trends in the dataset. Visualizations: The line chart helps in understanding the overall trend and volatility of the NIFTY 50 index. The scatter plot shows the relationship between open and close prices, providing insights into daily market behavior. The histogram illustrates the distribution of closing prices, offering a view of the most frequent price ranges and the overall spread of the data.

7 Q2

[ ]:
```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

def adf_test(timeseries):
    result = adfuller(timeseries)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    for key, value in result[4].items():
        print(f'Critical Value {key}: {value}')
    return result[1]

# Conduct ADF test
print("ADF Test for Close Price series data :")
adf_pvalue = adf_test(data['Close'])
```

```
ADF Test for Close Price series data :
ADF Statistic: 0.29029842448736826
p-value: 0.9768949795047296
Critical Value 1%: -3.485585145896754
```

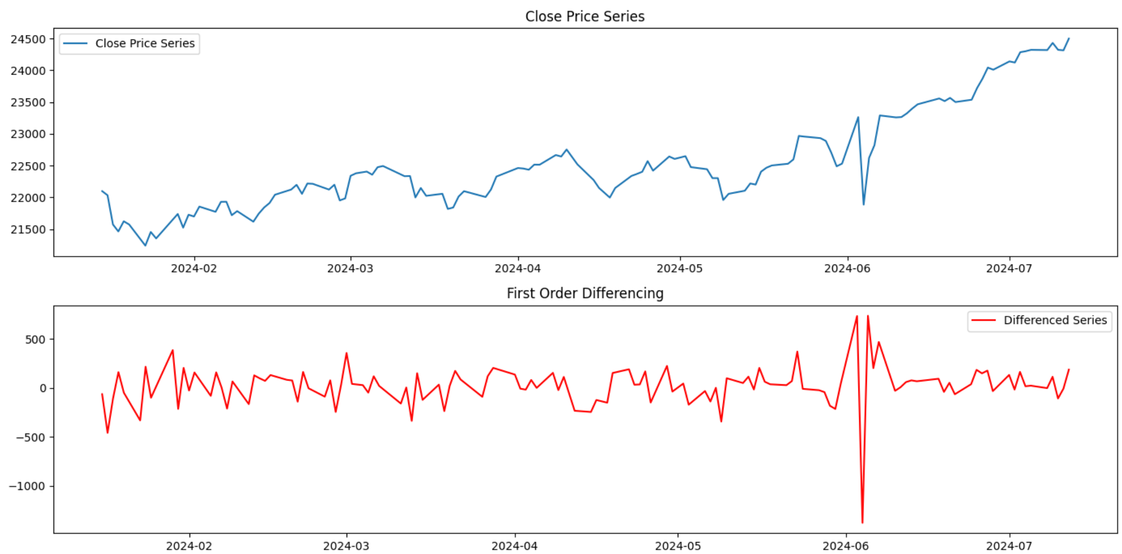
```
Critical Value 5%: -2.885738566292665
Critical Value 10%: -2.5796759080663887
```

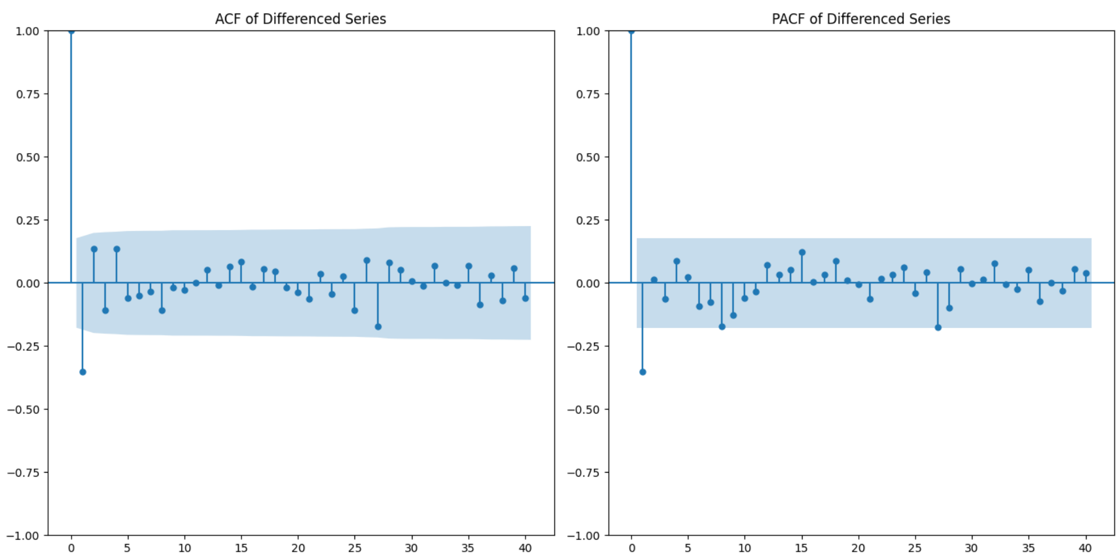
7.1 Interpretation for ADF Test

By seeing the p-value we can identify that it is non stationary, Because to be in stationary p-value should be <=0.05. Since we have got p value more that 0.05 its not in stationary. We fail to reject the null hypothesis. This suggests that the 'Close' price series is non-stationary.

[ ]:
```
#Differencing the Series
if adf_pvalue > 0.05:
    # Apply first differencing
    data['Differenced'] = data['Close'].diff().dropna()

    print("\nADF Test for Differenced Series:")
    adf_test(data['Differenced'].dropna())

    # Plot the original and differenced series
    plt.figure(figsize=(14, 7))

    plt.subplot(211)
    plt.plot(data['Close'], label='Close Price Series')
    plt.title('Close Price Series')
    plt.legend(loc='best')

    plt.subplot(212)
    plt.plot(data['Differenced'], label='Differenced Series', color='red')
    plt.title('First Order Differencing')
    plt.legend(loc='best')

    plt.tight_layout()
    plt.show()
else:
    print("The series is stationary and does not require differencing.")

# Step 3: Plot ACF and PACF
plt.figure(figsize=(14, 7))

plt.subplot(121)
plot_acf(data['Differenced'].dropna(), ax=plt.gca(), lags=40)
plt.title('ACF of Differenced Series')

plt.subplot(122)
plot_pacf(data['Differenced'].dropna(), ax=plt.gca(), lags=40)
plt.title('PACF of Differenced Series')

plt.tight_layout()
plt.show()
```

```
ADF Test for Differenced Series:
ADF Statistic: -15.735668156576093
p-value: 1.2736816611620572e-28
Critical Value 1%: -3.485585145896754
Critical Value 5%: -2.885738566292665
Critical Value 10%: -2.5796759080663887
```





7.2 INTERPRETATION

The p-value: 1.2736816611620572e-28 which is extremely small, essentially close to zero in the context of the Augmented Dickey-Fuller (ADF) test indicates strong evidence against the null hypothesis. The null hypothesis for the ADF test is that the series has a unit root, meaning it is non-stationary.

Since the p-value is much smaller than the typical significance level (0.05), you can reject the null hypothesis. This means that the series is stationary. Hence we have made it to stationary by taking necessary actions as shown in above code

8 Q3

9 Model Selection

[ ]:
```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the data
file_path = "E:/4TH_sem/ADA/NIFTY 50-15-01-2024-to-15-07-2024.csv"
data = pd.read_csv(file_path)

# Clean column names by stripping whitespace
data.columns = data.columns.str.strip()

# Parse the Date column and set it as the index
data['Date'] = pd.to_datetime(data['Date'], format='%d-%b-%Y')
data.set_index('Date', inplace=True)

# Ensure the data is sorted by date
data.sort_index(inplace=True)

# Split data into training and test sets
train = data['Close'][:-30]  # all but the last month
test = data['Close'][-30:]   # last month

# Define ARIMA and SARIMA parameters
p, d, q = 1, 1, 1  # Example ARIMA order
P, D, Q, s = 1, 1, 1, 12  # Example SARIMA seasonal order

# Fit ARIMA model
arima_model = ARIMA(train, order=(p, d, q)).fit()

# Fit SARIMA model (if seasonality is present)
```

```
sarima_model = SARIMAX(train, order=(p, d, q), seasonal_order=(P, D, Q, s)).
 ↪fit()

# Make predictions
arima_predictions = arima_model.forecast(steps=30)
sarima_predictions = sarima_model.forecast(steps=30)

# Forward fill missing values in test and predictions
test = test.ffill()
arima_predictions = arima_predictions.ffill()
sarima_predictions = sarima_predictions.ffill()

print("ARIMA Predictions:", arima_predictions)
print("SARIMA Predictions:", sarima_predictions)



# Evaluate model performance using mean absolute error and root mean squared␣
 ↪error
arima_mae = mean_absolute_error(test, arima_predictions)
arima_rmse = np.sqrt(mean_squared_error(test, arima_predictions))

sarima_mae = mean_absolute_error(test, sarima_predictions)
sarima_rmse = np.sqrt(mean_squared_error(test, sarima_predictions))

print(f'ARIMA MAE: {arima_mae}, ARIMA RMSE: {arima_rmse}')
print(f'SARIMA MAE: {sarima_mae}, SARIMA RMSE: {sarima_rmse}')
```

```
c:\Users\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
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\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
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\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
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\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
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\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: A date index has
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\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Non-stationary
starting autoregressive parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
c:\Users\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
```

```
ARIMA Predictions: 93     22502.784576
94     22492.301769
95     22500.076268
96     22494.310366
97     22498.586606
98     22495.415163
99     22497.767241
100    22496.022840
101    22497.316563
102    22496.357082
103    22497.068674
104    22496.540927
105    22496.932327
106    22496.642048
107    22496.857331
108    22496.697668
109    22496.816081
110    22496.728261
111    22496.793392
112    22496.745088
113    22496.780912
114    22496.754343
115    22496.774048
116    22496.759434
117    22496.770273
118    22496.762234
119    22496.768196
120    22496.763775
121    22496.767054
122    22496.764622
Name: predicted_mean, dtype: float64
SARIMA Predictions: 93     22527.161050
94     22616.989655
```

```
95     22700.797169
96     22695.411669
97     22642.812242
98     22618.305384
99     22707.282237
100    22740.420749
101    22646.711253
102    22661.868321
103    22610.674563
104    22598.116270
105    22615.215916
106    22723.707608
107    22794.804229
108    22800.024560
109    22742.252445
110    22726.105611
111    22818.384052
112    22853.826081
113    22759.288823
114    22774.938716
115    22718.393936
116    22701.488419
117    22718.161456
118    22826.932317
119    22897.759132
120    22903.171276
121    22845.162943
122    22829.080500
Name: predicted_mean, dtype: float64
ARIMA MAE: 1194.895912437509, ARIMA RMSE: 1312.839178811847
SARIMA MAE: 982.9873031353203, SARIMA RMSE: 1085.9566569399324
```

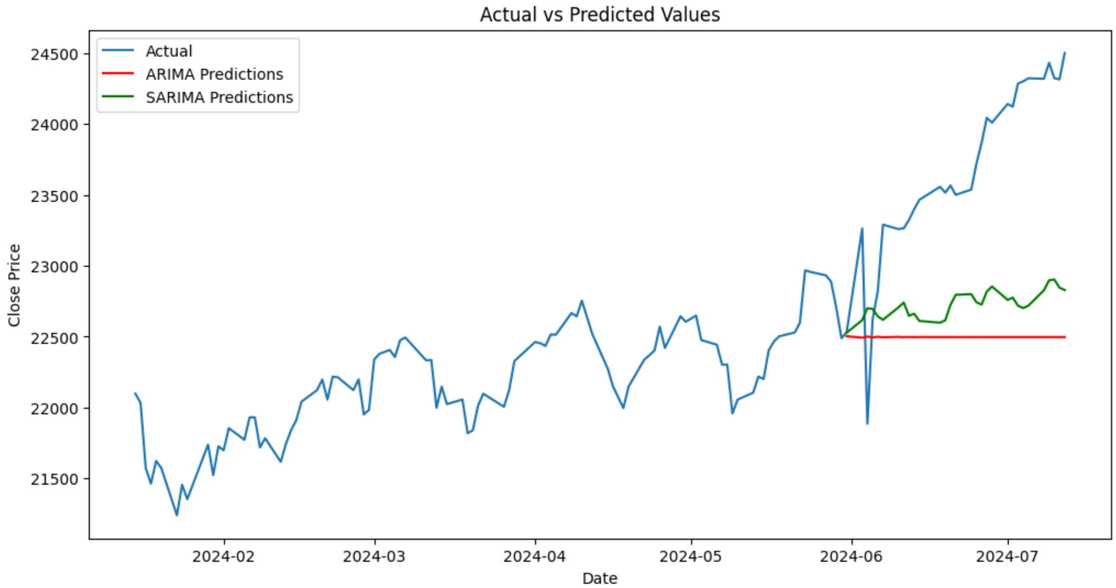
```
c:\Users\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:836: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
c:\Users\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:836: FutureWarning: No supported
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\Pratham.m\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:836: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
```

10 EVALUATE ACCURACY

[ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Ensure the index alignment between predictions and test set
arima_predictions.index = test.index
sarima_predictions.index = test.index

# Calculate metrics for ARIMA
arima_mae = mean_absolute_error(test, arima_predictions)
arima_rmse = np.sqrt(mean_squared_error(test, arima_predictions))

# Calculate metrics for SARIMA
sarima_mae = mean_absolute_error(test, sarima_predictions)
sarima_rmse = np.sqrt(mean_squared_error(test, sarima_predictions))

print(f'ARIMA MAE: {arima_mae}, ARIMA RMSE: {arima_rmse}')
print(f'SARIMA MAE: {sarima_mae}, SARIMA RMSE: {sarima_rmse}')
```

```
ARIMA MAE: 1194.895912437509, ARIMA RMSE: 1312.839178811847
SARIMA MAE: 982.9873031353203, SARIMA RMSE: 1085.9566569399324
```

[ ]:
```
import matplotlib.pyplot as plt

# Ensure the index alignment between predictions and test set
arima_predictions.index = test.index
sarima_predictions.index = test.index

# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Close'], label='Actual')
plt.plot(test.index, arima_predictions, label='ARIMA Predictions', color='red')
plt.plot(test.index, sarima_predictions, label='SARIMA Predictions',␣
 ↪color='green')
plt.title('Actual vs Predicted Values')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



10.1 ARIMA (AutoRegressive Integrated Moving Average)

Components:

AutoRegressive (AR) part (p): This component uses the dependency between an observation and a number of lagged observations (i.e., previous time points). Integrated (I) part (d): This component involves differencing the observations (subtracting the previous observation from the current observation) in order to make the time series stationary (i.e., to remove trends and seasonality). Moving Average (MA) part (q): This component uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

10.2 When to Use:

ARIMA models are typically used for non-seasonal time series data where there is no strong seasonal pattern. If the data shows a trend but no seasonality, an ARIMA model might be sufficient. ARIMA is best for data that can be made stationary through differencing and where the patterns can be captured by autoregressive and moving average terms. SARIMA (Seasonal AutoRegressive Integrated Moving Average)

10.3 Components:

AutoRegressive (AR) part (p): Similar to ARIMA. Integrated (I) part (d): Similar to ARIMA. Moving Average (MA) part (q): Similar to ARIMA. Seasonal components (P, D, Q, s): These are the seasonal counterparts of the non-seasonal components. P: Seasonal autoregressive order. D: Seasonal differencing order. Q: Seasonal moving average order. s: Length of the seasonal cycle (e.g., 12 for monthly data with yearly seasonality).

11 When to Use SARIMA:

SARIMA models are used when the time series data exhibits clear seasonal patterns. If there are seasonal effects (e.g., monthly data with a yearly cycle), SARIMA is preferred. The seasonal component helps in capturing the periodic fluctuations that occur at regular intervals. Key Differences

#Seasonality:

ARIMA is suitable for non-seasonal data. SARIMA extends ARIMA to capture seasonal effects by adding seasonal components. Complexity:

ARIMA has three parameters (p, d, q). SARIMA has seven parameters (p, d, q, P, D, Q, s), making it more complex but also more capable of handling seasonality.

12 When to Prefer One Over the Other

12.1 Prefer ARIMA when:

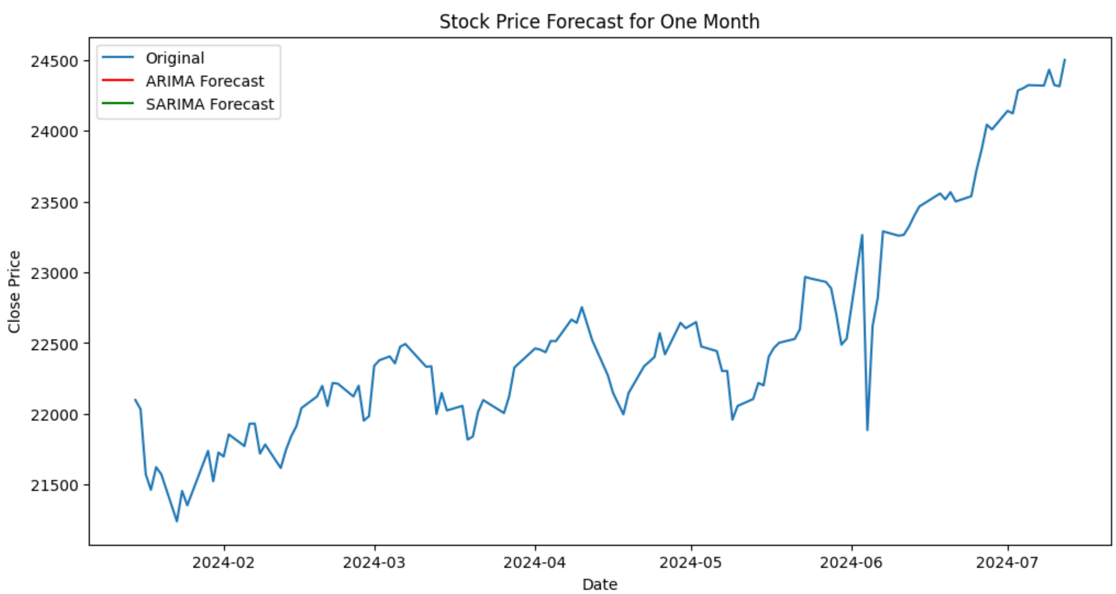
The time series data does not have a strong or evident seasonal pattern. The data can be made stationary by differencing and any patterns are well captured by the AR and MA terms. The focus is on modeling trend and noise without periodic seasonal fluctuations. ## Prefer SARIMA when:

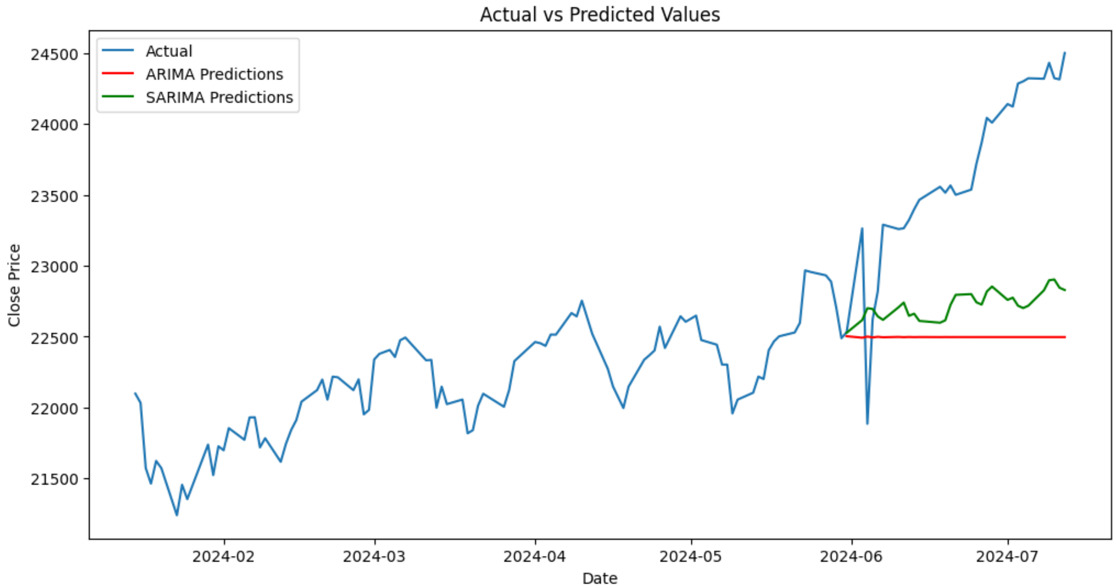
The time series data exhibits clear seasonal patterns. There are periodic fluctuations that occur at regular intervals (e.g., monthly sales data with a yearly cycle). Capturing seasonality is crucial for making accurate forecasts.

13 Q4

[ ]:
```python
# Create a date range for the forecast
forecast_dates = pd.date_range(start=data.index[-1] + pd.Timedelta(days=1),␣
 ↪periods=30, freq='D')

# Convert forecast to DataFrame
arima_forecast_df = pd.DataFrame(arima_predictions, index=forecast_dates,␣
 ↪columns=['ARIMA_Forecast'])
sarima_forecast_df = pd.DataFrame(sarima_predictions, index=forecast_dates,␣
 ↪columns=['SARIMA_Forecast'])

# Plot the original time series and forecasts
plt.figure(figsize=(12, 6))

# Plot original time series
plt.plot(data['Close'], label='Original')

# Plot ARIMA forecast
plt.plot(arima_forecast_df, label='ARIMA Forecast', color='red')

# Plot SARIMA forecast
plt.plot(sarima_forecast_df, label='SARIMA Forecast', color='green')
```

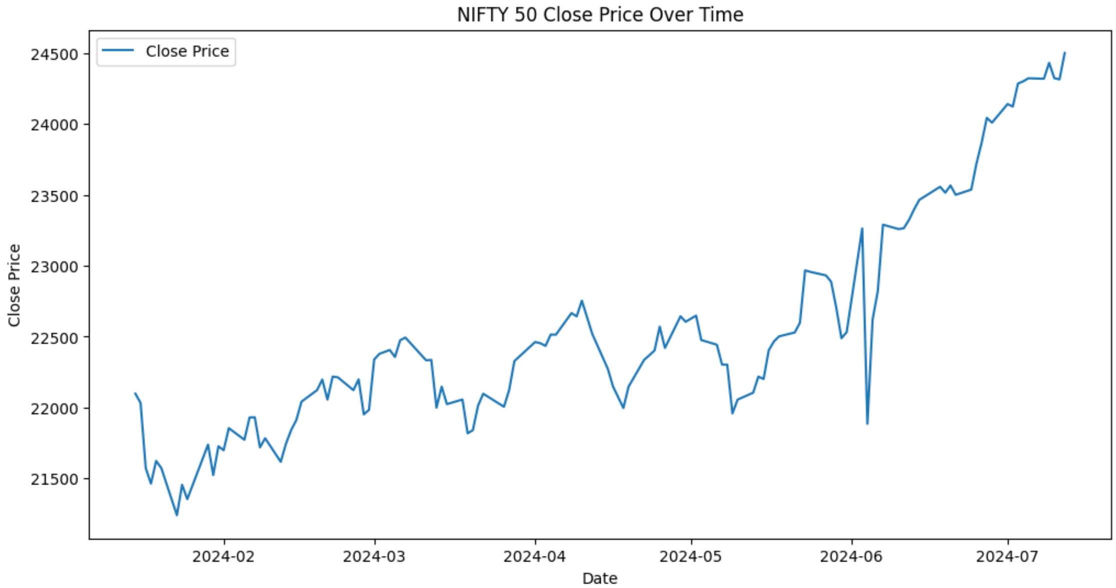
```python
# Customize plot
plt.title('Stock Price Forecast for One Month')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Close'], label='Actual')
plt.plot(test.index, arima_predictions, label='ARIMA Predictions', color='red')
plt.plot(test.index, sarima_predictions, label='SARIMA Predictions',␣
 ↪color='green')
plt.title('Actual vs Predicted Values')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

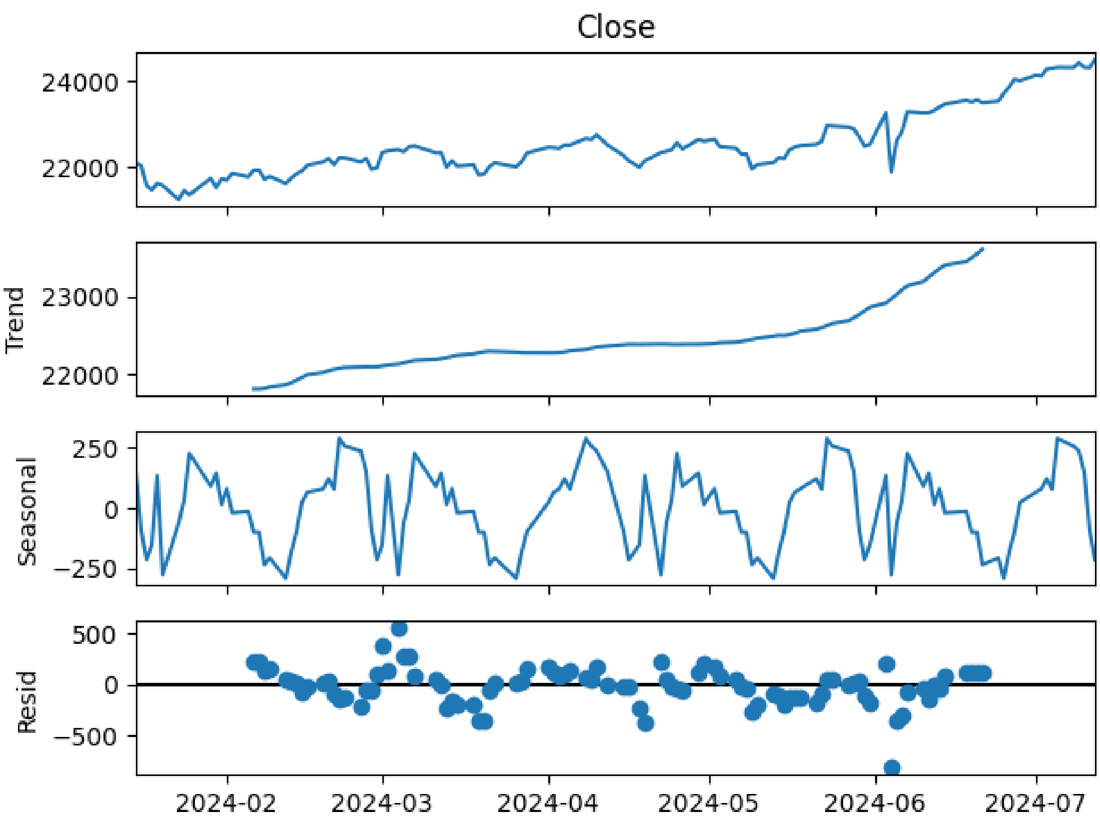




14 Q5

[ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
# Plot the entire time series to inspect for trends and seasonality
plt.figure(figsize=(12, 6))
plt.plot(data['Close'], label='Close Price')
plt.title('NIFTY 50 Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Decompose the time series to check for trends and seasonality
decomposition = seasonal_decompose(data['Close'], model='additive', period=30)
decomposition.plot()
plt.show()
```


NIFTY 50 Close Price Over Time
Close Price
24500
24000
23500
23000
22500
22000
21500
Close Price
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
Date


Close
24000
22000
Trend
23000
22000
Seasonal
250
0
−250
Resid
500
0
−500
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07

15 6.Interpretation

Exploring the Time Series Data:

Trends: The visual inspection and decomposition of the time series data show a clear upward trend in the stock prices over the given period. Seasonality: The decomposition plot indicates a noticeable seasonal pattern, suggesting that there are regular fluctuations in the stock prices that repeat over a consistent period.

15.1 Model Performance:

ARIMA Model:

MAE: The Mean Absolute Error (MAE) indicates the average magnitude of errors in the predictions.

RMSE: The Root Mean Squared Error (RMSE) further quantifies the model's prediction accuracy. SARIMA Model:

MAE: The SARIMA model performs better indicating smaller average errors compared to the ARIMA model. RMSE: The SARIMA model also shows a lower RMSE suggesting better predictive accuracy than the ARIMA model.