IntelliHack 5.0 Task 04

The dataset provided for task 04 contains historical stock price which contains price details with respect to dates from `1980-03-17 00:00:00` to `2024-12-27 00:00:00`.
First to understand the data set we got the statistical details about all the columns.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11291 entries, 0 to 11290
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  11291 non-null  int64
 1   Date        11291 non-null  datetime64[ns]
 2   Adj Close   11291 non-null  float64
 3   Close       11291 non-null  float64
 4   High        11291 non-null  float64
 5   Low         11291 non-null  float64
 6   Open        11291 non-null  float64
 7   Volume      11291 non-null  float64
dtypes: datetime64[ns](1), float64(6), int64(1)
memory usage: 705.8 KB
None
```

```
         Unnamed: 0                           Date     Adj Close  \
count  11291.000000                          11291  11291.000000
mean    5645.000000  2002-07-28 16:29:40.391462144     63.585462
min        0.000000            1980-03-17 00:00:00      2.259452
25%     2822.500000            1991-05-15 12:00:00     19.224636
50%     5645.000000            2002-07-25 00:00:00     50.583900
75%     8467.500000            2013-10-09 12:00:00    104.660000
max    11290.000000            2024-12-27 00:00:00    254.770004
std     3259.575279                            NaN     52.271655

              Close          High           Low          Open        Volume
count  11291.000000  11291.000000  11291.000000  11291.000000  1.129100e+04
mean      72.058797     72.480062     71.645541     67.956679  2.147261e+05
min        3.237711      3.237711      3.237711      0.000000  0.000000e+00
25%       27.548208     27.735613     27.548208      0.000000  1.351500e+04
50%       66.040001     66.650002     65.440002     66.000000  9.070000e+04
75%      114.295002    114.895000    113.589996    114.294998  2.922000e+05
max      254.770004    255.229996    253.589996    255.000000  1.858270e+07
std       51.296089     51.554292     50.978072     55.851397  3.874063e+05
```

To get an understanding about the dataset we retrieved the first elements of the dataset as well.

```
   Unnamed: 0        Date  Adj Close     Close      High       Low      Open  \
0           0  1980-03-17   2.296798  3.291227  3.344743  3.291227  0.000000
1           1  1980-03-18   2.306134  3.304606  3.358122  3.304606  0.000000
2           2  1980-03-19   2.306134  3.304606  3.304606  3.304606  3.304606
3           3  1980-03-20   2.306134  3.304606  3.358122  3.304606  0.000000
4           4  1980-03-21   2.362154  3.384880  3.438396  3.384880  0.000000

    Volume
0  41109.0
1   9343.0
2      0.0
3  10277.0
4   8409.0
```

And also we check the null value counts in each data column.

```
Unnamed: 0      0
Date          110
Adj Close      93
Close         117
High           95
Low           127
Open          103
Volume        145
dtype: int64
```

To get an understanding of the distribution of the dataset and the relationship between different features we used several data visualization techniques. But before doing that, we followed some data pre-processing techniques to clean the data, encode the data and to do feature engineering steps as well.

Since all these data vary with time we used the forward fill method to fill the missing values of the above columns.

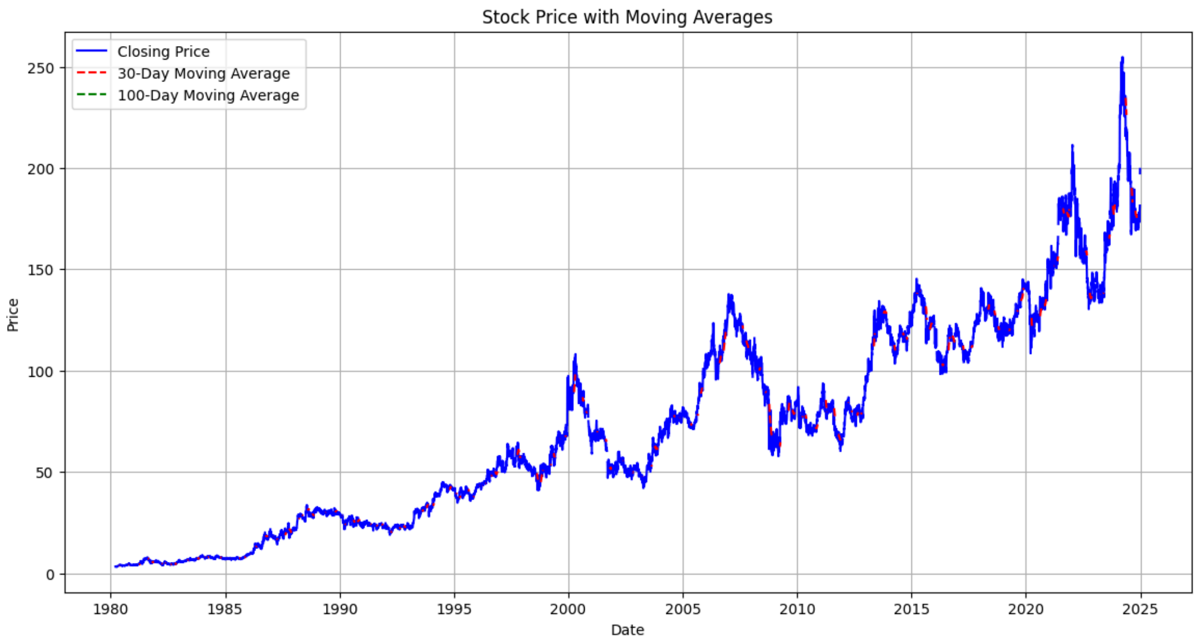
When performing feature engineering first we changed the Date column into datetime objects in pandas. And then create new features to retrieve year, month and day from the date. Furthermore, since we are doing time-series analysis we created features such as, `'rolling_mean_5', 'rolling_std_5'` `, 'lag_1', 'lag_5', 'lag_10', 'ema_10', 'rolling_volume_5', 'volatility', 'fourier_1', 'close_diff_1', 'close_diff_5'` which are important to analyze the time-series analysis for the stock price of the closing days.

And then we do some visualizations for the created features and data in ‘Close’ column.

Visualization of closing price with respect to date.

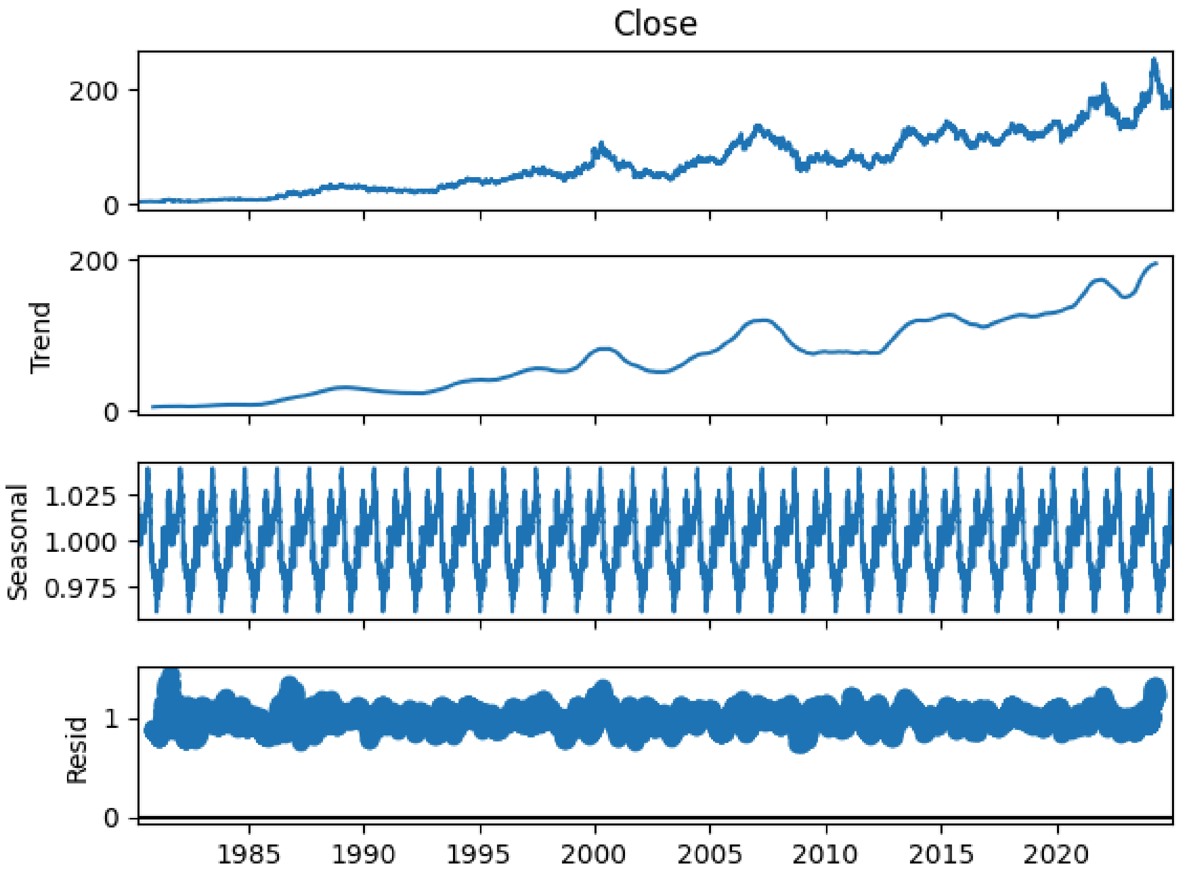
Visualization of closing price with moving averages of the close price.

Since the closing prices are highly correlated with time it is necessary to check its relationship with moving averages as well.



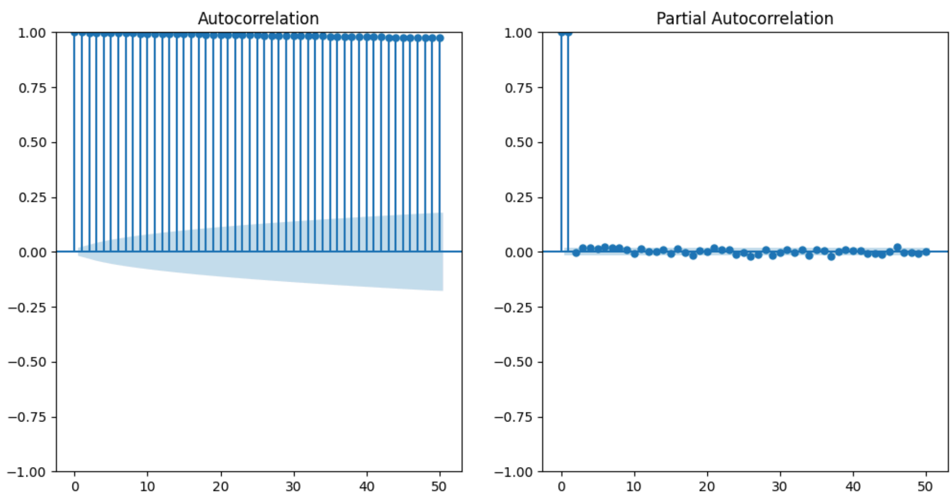
Visualization of closing price with moving averages of the close price.

This helps to identify trends and to smooth out short term fluctuations. Also this detects abnormal trends in stock prices.

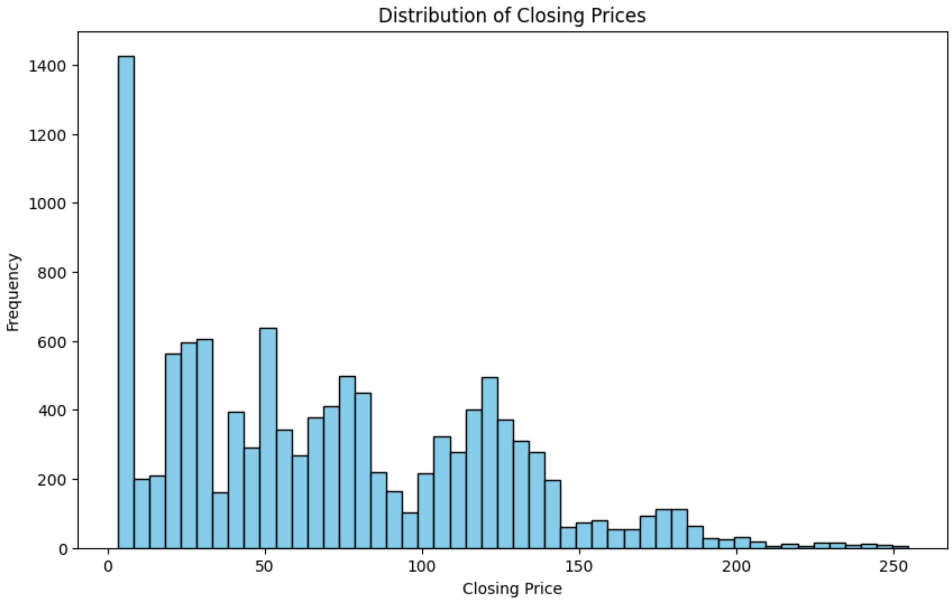


ACF and PACF plots to check the correlation at different lags

This determines the correlation of the time series (closing price) with its past values.
ACF shows the strength of correlation at different lags (useful for identifying seasonality) and PACF helps in selecting lag values for AR terms in models like ARIMA.

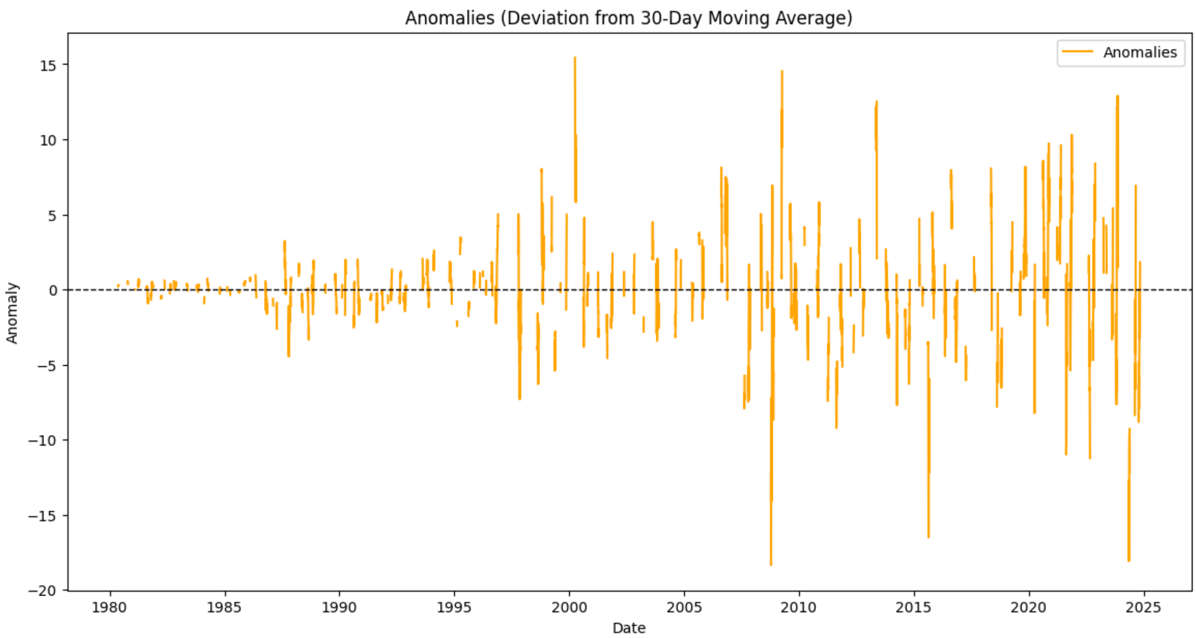


Distribution of closing price



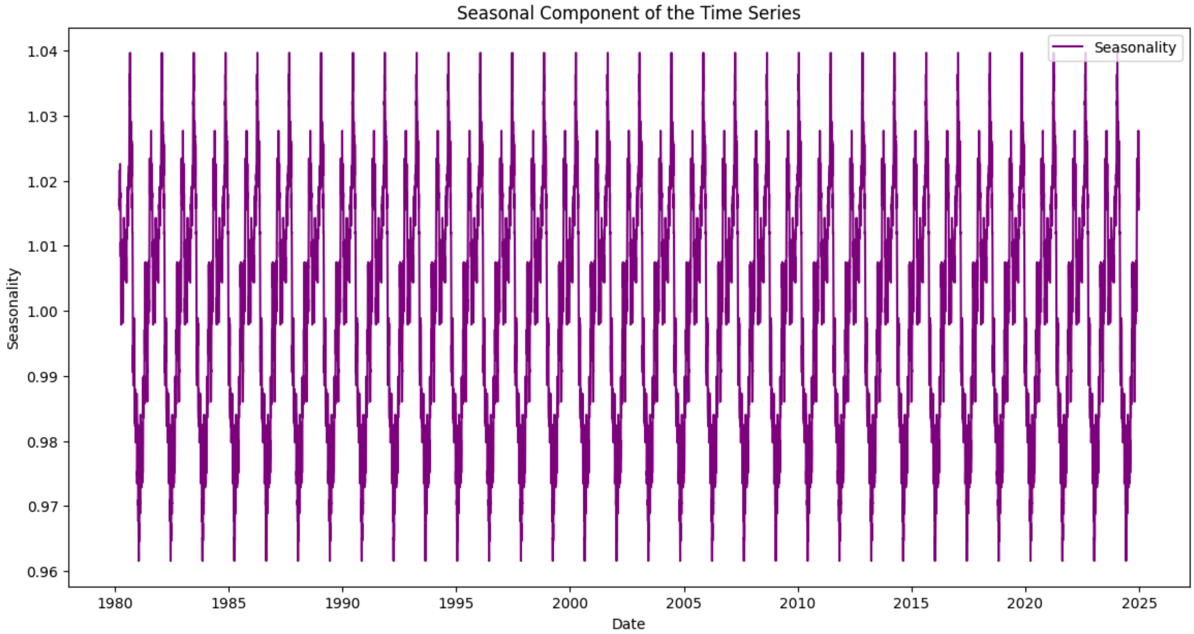
Visualization of Anomalies (Deviation from 30-day moving average)

This identifies unusual spikes or drops in data that may indicate market events or errors. It compares actual prices with a smoothed 30-day moving average and highlights large deviations. It is useful to detect unexpected volatility, fraud detection, or data errors.



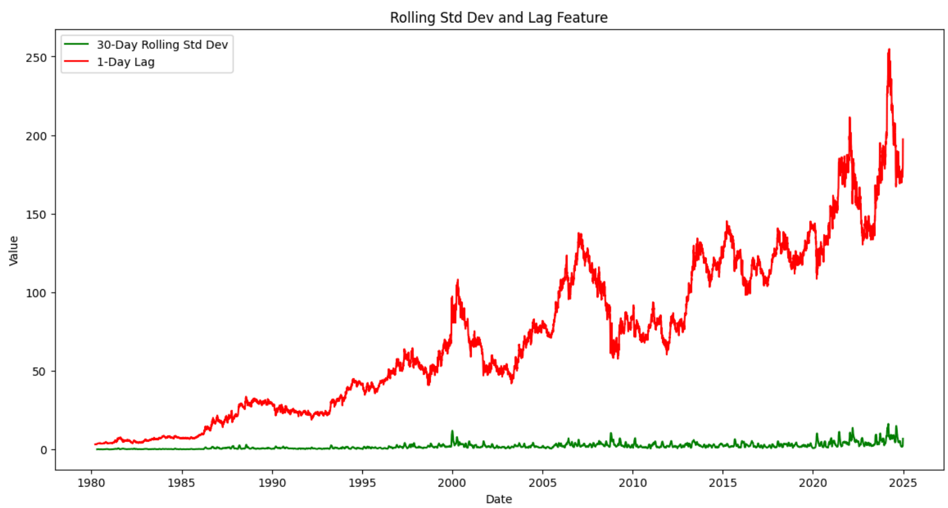
Visualization of Seasonal Component Decompositon.

It breaks the time series into trend, seasonality, and residual components. It helps separate predictable seasonal patterns from random noise. It is important in understanding cyclic behaviors (e.g., weekly or yearly patterns in stock prices).

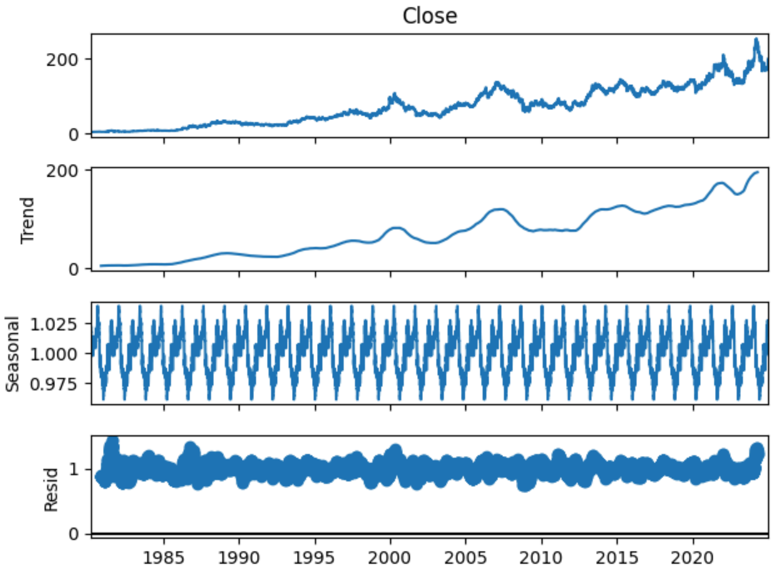


Visualization of Rolling Std Dev and Lag Feature.

It measures how volatility changes over time and checks the effect of previous values on the current price. The rolling standard deviation shows variability in price movements, while lag features analyze autocorrelation. It helps detect market stability, risk assessment, and feature engineering for ML models.



Insights gained from the above visualizations.



Observed (Top Plot - "Close")

- The original time series, showing the closing price trend over time.
- There is a general **upward trend**, with notable fluctuations, especially post-2000.

Trend Component (Second Plot - "Trend")

- Captures the long-term movement of the stock price.
- The trend shows steady growth with periodic dips, possibly reflecting economic recessions or market crashes (e.g., 2008 financial crisis).

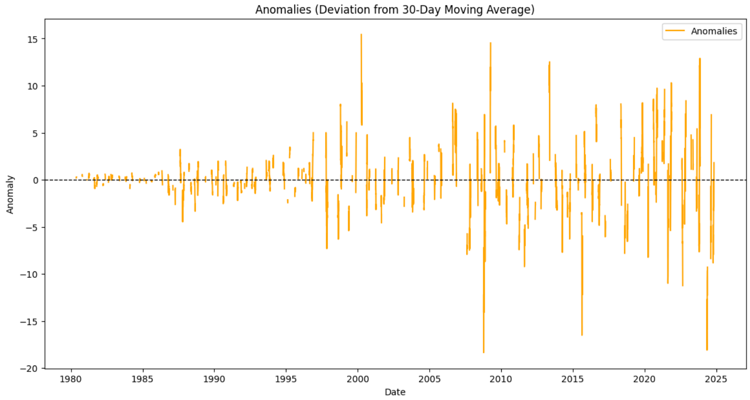
Seasonal Component (Third Plot - "Seasonal")

- Displays repeating patterns, suggesting seasonal influences.
- There is a clear periodic pattern (likely annual), indicating a seasonal cycle in stock prices.
- This could be due to **annual financial cycles, earnings reports, or macroeconomic factors**.

Residual Component (Bottom Plot - "Resid")

- Represents noise and irregular variations after removing trend and seasonality.
- Higher variance in recent years indicates increased market volatility.
- If the residuals show a pattern instead of randomness, the model may need adjustments.

This visualization suggests that, stock price has been increasing over the years and the regular patterns indicate predictable cycles and volatility has increased over time.
Hence this suggests that a SARIMA or Prophet model with seasonal components may perform well for forecasting.



Small Deviations Initially (Before 1990s)

- Anomalies are minor, indicating a relatively stable market with low volatility.

Increasing Deviations (1990s - 2000s)

- More frequent and larger fluctuations appear, possibly due to **market crashes, economic shifts, or major financial events**.
- Noticeable spikes could indicate **financial crises (e.g., 2000 Dot-com Bubble, 2008 Financial Crisis)**.

Highly Volatile Periods (Post-2010s)

- There are **extreme anomalies**, both positive and negative, showing a highly volatile market.
- This could be linked to global events, policy changes, or digital transformations in stock trading.
- Large downward spikes may suggest **market crashes**, while large upward spikes could indicate **speculative bubbles**.

And this visualization helps in identifying periods of extreme fluctuations and the spikes correlate with market crashes or boom also the large deviations indicate increased investment risk.

To fill the missing values created in newly created features we perform different techniques.

1) <u>Linear Interpolation</u>

```
[53] print(data.isnull().sum())
     data['MA_30'] = data['MA_30'].interpolate(method='linear')
     data['MA_100'] = data['MA_100'].interpolate(method='linear')
     data['Anomalies'] = data['Anomalies'].interpolate(method='linear')
     data['Rolling_Std_30'] = data['Rolling_Std_30'].interpolate(method='linear')
     data['Lag_1'] = data['Lag_1'].ffill()
```

```
Close                 0
MA_30              9755
MA_100            11685
Anomalies          9755
Rolling_Std_30       29
Lag_1                 1
dtype: int64
```

```
print(data.isnull().sum())
```

```
Close                 0
MA_30                44
MA_100            11685
Anomalies            44
Rolling_Std_30       29
Lag_1                 1
dtype: int64
```

But this technique failed to fill all the missing values.

2) Spline Interpolation

```
# Perform spline interpolation for missing values
data['MA_30'] = data['MA_30'].interpolate(method='spline', order=3)
data['MA_100'] = data['MA_100'].interpolate(method='spline', order=3)
data['Anomalies'] = data['Anomalies'].interpolate(method='spline', order=3)
data['Rolling_Std_30'] = data['Rolling_Std_30'].interpolate(method='spline', order=3)
data['Lag_1'] = data['Lag_1'].interpolate(method='spline', order=3)
print(data.isnull().sum())
```

```
Close                 0
MA_30                44
MA_100            11685
Anomalies            44
Rolling_Std_30       29
Lag_1                 1
dtype: int64
```

This technique also failed to fill all the missing values. So, we performed the following technique to fill the missing values.

3) Using ARIMA Model to predict the missing values.

```
from statsmodels.tsa.arima.model import ARIMA

# Fit an ARIMA model for time series forecasting
model = ARIMA(data['Close'], order=(5, 1, 0))  # Example parameters, modify as needed
model_fit = model.fit()

# Predict missing values (forecast)
forecast = model_fit.predict(start=0, end=len(data)-1, dynamic=False)

# Fill missing values in the 'Anomalies' column with forecasted values
data['Anomalies'] = data['Anomalies'].fillna(pd.Series(forecast, index=data.index))
data['MA_30'] = data['MA_30'].fillna(pd.Series(forecast, index=data.index))
data['MA_100'] = data['MA_100'].fillna(pd.Series(forecast, index=data.index))
data['Rolling_Std_30'] = data['Rolling_Std_30'].fillna(pd.Series(forecast, index=data.index))
data['Lag_1'] = data['Lag_1'].fillna(pd.Series(forecast, index=data.index))

print(data.isnull().sum())
```

```
Close             0
MA_30             0
MA_100            0
Anomalies         0
Rolling_Std_30    0
Lag_1             0
dtype: int64
```

And this method was able to predict and fill all the missing values.

Model Selection

To build a model to predict the stock price in closing dates, we use 3 different models.

1) Linear Regression

   This model got the following accuracy metrics for this analysis.

```
Linear Regression MAE: 1.4470
Linear Regression RMSE: 2.0838
Linear Regression R2: 0.9956
```

2) Random Forest Regression

   This model got the following accuracy metrics for this analysis.

```
Random Forest MAE: 14.2053
Random Forest RMSE: 26.1327
Random Forest R2: 0.3049
```

3) SARIMAX Model

   This model got the following accuracy metrics.

```
📌 Forecasted Stock Prices:
        Date  Predicted_Close
0 2016-01-15        90.001240
1 2016-01-18        89.163468
2 2016-01-19        88.900410
3 2016-01-20        86.373078
4 2016-01-21        83.749442

📈 SARIMAX Performance Metrics:
Mean Absolute Error (MAE): 25.4825
Mean Squared Error (MSE): 652.1081
Root Mean Squared Error (RMSE): 25.5364
Mean Absolute Percentage Error (MAPE): 0.2253
R-squared (R² Score): -241.3797
Mean Error (ME): 25.4825
Median Absolute Error: 26.2269
Mean Percentage Error (MPE): 22.5310%
```

When analysing the performance of the above three different models,

```
# Compare the performance of all models
models = ['Linear Regression', 'Random Forest', 'SARIMAX']
mae_values = [lr_mae, rf_mae, sarima_mae]
rmse_values = [lr_rmse, rf_rmse, sarima_rmse]
r2_values = [lr_r2, rf_r2, sarima_r2]

results = pd.DataFrame({
    'Model': models,
    'MAE': mae_values,
    'RMSE': rmse_values,
    'R2': r2_values
})

print(results)
```

```
               Model        MAE       RMSE          R2
0  Linear Regression   1.446952   2.083786    0.995580
1      Random Forest  14.205261  26.132723    0.304862
2            SARIMAX  25.482472  25.536407 -241.379658
```

By analysing the above metrics, even though the visualizations suggests to use SARIMA model, it is better to use linear regression as the model which is having minimum error and also when considering the R^2 score, it is clear that Linear Regression is generalized well than the other complex models.