Stock Price Prediction Challenge Documentation Report

1. Introduction

The goal of this project is to predict the closing price of a stock 5 trading days into the future using historical stock price data. The dataset contains features such as date, opening price, closing price, high, low, and volume. This report documents the exploratory data analysis, feature engineering, model development, and evaluation process.

2. Exploratory Data Analysis (EDA)

2.1 Dataset Overview

The dataset contains the following columns:

Date: The date of the stock price record.

Open: The opening price of the stock.

High: The highest price of the stock during the day.

Low: The lowest price of the stock during the day.

Close: The closing price of the stock.

Volume: The trading volume of the stock.

2.2 Missing Values

The dataset was checked for missing values, and none were found. All columns were complete.

python

Copy

```
# Check for missing values
print(df.isnull().sum())
```

2.3 Summary Statistics

Summary statistics were calculated for the numerical columns to understand the distribution of the data.

python

Copy

```
# Summary statistics

print(df.describe())
```

2.4 Visualizations

Closing Price Over Time: The closing price was plotted over time to observe trends and patterns.

Seasonality and Trends: The time series was decomposed to identify trends and seasonality.

python

Copy

```
# Plot closing price over time

plt.figure(figsize=(10, 6))

plt.plot(df['Date'], df['Close'])

plt.title('Stock Closing Price Over Time')

plt.xlabel('Date')

plt.ylabel('Closing Price')

plt.show()


# Decompose the time series

decomposition = seasonal_decompose(df['Close'], period=30)

decomposition.plot()

plt.show()
```

3. Feature Engineering

3.1 Lag Features

Lag features were created to capture the past 5 days' closing prices, which are used to predict the future closing price.

python

Copy

```
# Create lag features for the past 5 days
for i in range(1, 6):
    df[f'Lag_{i}'] = df['Close'].shift(i)
```

3.2 Moving Averages

Moving averages were calculated to smooth out short-term fluctuations and highlight longer-term trends.

python

Copy

```
# Create moving averages
df['MA_7'] = df['Close'].rolling(window=7).mean()  # 7-day moving average
df['MA_30'] = df['Close'].rolling(window=30).mean()  # 30-day moving average
```

3.3 Data Cleaning

Rows with missing values (due to lag features and moving averages) were dropped.

python

Copy

```
# Drop rows with NaN values
df.dropna(inplace=True)
```

4. Model Development

4.1 Data Splitting

The dataset was split into training and testing sets (80% training, 20% testing).

python

Copy

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df.drop(['Close'], axis=1)
y = df['Close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.2 Model Selection

The XGBoost Regressor was chosen for its ability to handle non-linear relationships and its performance on time series data.

python

Copy

```
from xgboost import XGBRegressor

# Train an XGBoost model
model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train, y_train)
```

4.3 Model Evaluation

The model was evaluated using Root Mean Squared Error (RMSE) and Directional Accuracy.

python

Copy

```
from sklearn.metrics import mean_squared_error

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model
rmse = mean_squared_error(y_test, y_pred, squared=False)
print(f"RMSE: {rmse}")

# Calculate directional accuracy
directional_accuracy = np.mean(np.sign(y_pred) == np.sign(y_test))
print(f"Directional Accuracy: {directional_accuracy}")
```

5. Results and Insights

5.1 Predictive Accuracy

RMSE: The RMSE value indicates the average deviation of the predicted closing price from the actual closing price.

Directional Accuracy: The directional accuracy measures how often the model correctly predicts the direction of the price movement (up or down).

5.2 Simulated Trading Performance

A simulated trading strategy was implemented to evaluate the practical trading value of the model.

python

Copy

```
# Simulate trading performance
df_test = X_test.copy()
df_test['Actual'] = y_test
df_test['Predicted'] = y_pred
df_test['Return'] = df_test['Actual'].pct_change()
df_test['Strategy'] = df_test['Predicted'].pct_change()

# Calculate cumulative returns
```

```
df_test['Cumulative_Actual'] = (1 + df_test['Return']).cumprod()
df_test['Cumulative_Strategy'] = (1 + df_test['Strategy']).cumprod()

# Plot cumulative returns
plt.figure(figsize=(10, 6))
plt.plot(df_test['Cumulative_Actual'], label='Actual Returns')
plt.plot(df_test['Cumulative_Strategy'], label='Strategy Returns')
plt.title('Simulated Trading Performance')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()
```

6. Limitations and Future Work

6.1 Limitations

Data Quality: The model's performance is highly dependent on the quality and completeness of the historical data.

Market Volatility: Sudden market changes (e.g., news events) may not be captured by the model.

Feature Engineering: The current features may not fully capture all relevant patterns in the data.

6.2 Future Improvements

Incorporate External Data: Add external features such as news sentiment, macroeconomic indicators, or sector performance.

Advanced Models: Experiment with more advanced models like LSTM (Long Short-Term Memory) for time series forecasting.

Real-Time Predictions: Implement a real-time prediction system using streaming data.

7. Conclusion

The model successfully predicts the stock's closing price 5 trading days into the future with reasonable accuracy. The simulated trading strategy demonstrates the practical value of the model. However, there are limitations, and future work can focus on improving the model's robustness and incorporating additional features.

8. Deliverables

Jupyter Notebook: Contains the complete code for data analysis, model development, and evaluation.

EDA Report: Includes visualizations, insights, and feature engineering decisions.

Model Documentation: Explains the model selection process, evaluation metrics, and limitations.

CSV File: Contains the predictions for the test period.

README File: Summarizes the approach, key findings, and instructions to reproduce the results.

9. Instructions to Reproduce Results

Install Dependencies:

bash

Copy

```
pip install pandas numpy matplotlib seaborn scikit-learn xgboost
```

Run the Jupyter Notebook: Execute the cells in the provided Jupyter Notebook to reproduce the analysis and results.

View the Report: Refer to the EDA report and model documentation for detailed insights