5303 Final Project

December 3, 2023

1 Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures, StandardScaler,␣
 ↪MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
```

2 Data Loading

[2]:
```
# Load the dataset
df = pd.read_csv('AAPL.csv')
```

[3]:
```
# Convert 'Date' to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```

3 Feature Engineering

[4]:
```
# Creating additional features
df['DayOfWeek'] = df.index.dayofweek
df['Month'] = df.index.month
```

4 Data Splitting (Train-Test Split)

[5]:
```
# Splitting the data into training and testing sets
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]

# Define the features and target variable
```

```
features = ['DayOfWeek', 'Month', 'Open', 'High', 'Low', 'Adj Close']
target = 'Close'
X_train, y_train = train[features], train[target]
X_test, y_test = test[features], test[target]
```

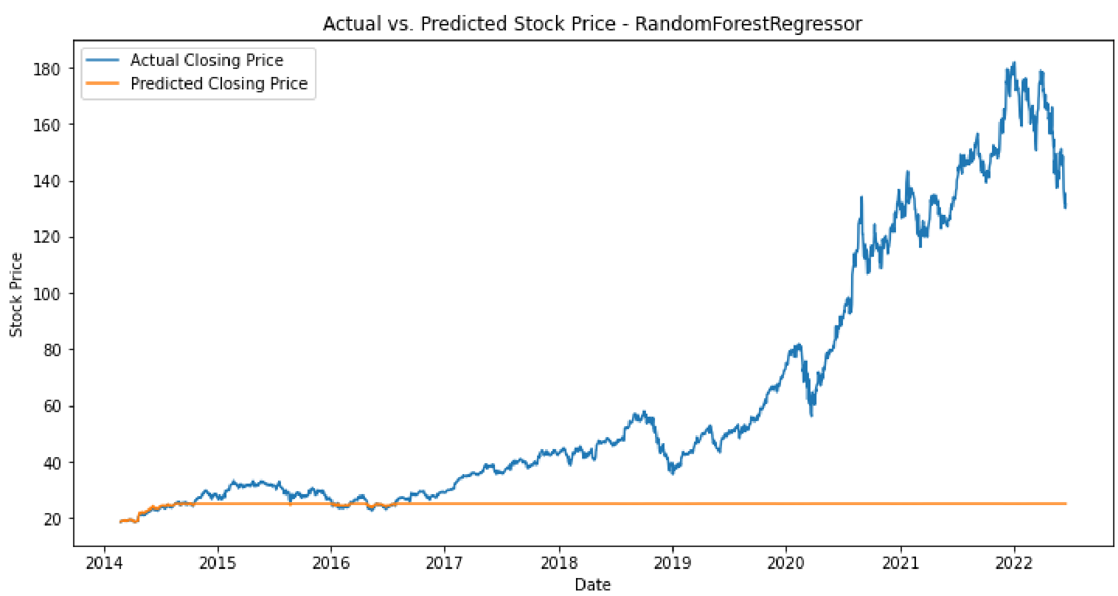
5 RandomForestRegressor Model

[6]:
```
# RandomForestRegressor Model
rf_model = RandomForestRegressor(n_estimators=500, random_state=42, max_depth=20)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# MSE for RandomForestRegressor
mse_rf = mean_squared_error(y_test, y_pred_rf)
mse_rf
```

[6]: 3518.5581836062847

[7]:
```
# Plotting actual vs. predicted values for RandomForestRegressor
plt.figure(figsize=(12, 6))
plt.plot(test.index, y_test, label='Actual Closing Price')
plt.plot(test.index, y_pred_rf, label='Predicted Closing Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Actual vs. Predicted Stock Price - RandomForestRegressor')
plt.legend()
plt.savefig('actual_vs_predicted_rf.png')
```



6 Polynomial Regression Model

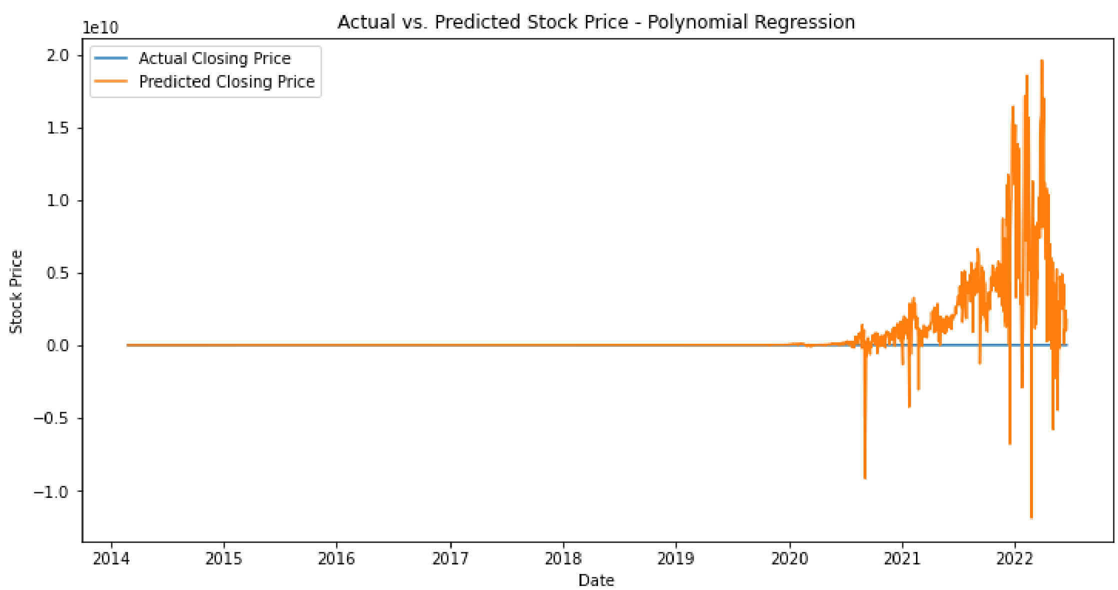
[8]:
```
# Polynomial Regression Model
degree = 6
poly = PolynomialFeatures(degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)
y_pred_poly = poly_model.predict(X_test_poly)

# MSE for Polynomial Regression
mse_poly = mean_squared_error(y_test, y_pred_poly)
mse_poly
```

[8]: 5.870888318691471e+18

[9]:
```
# Plotting actual vs. predicted values for Polynomial Regression
plt.figure(figsize=(12, 6))
plt.plot(test.index, y_test, label='Actual Closing Price')
plt.plot(test.index, y_pred_poly, label='Predicted Closing Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Actual vs. Predicted Stock Price - Polynomial Regression')
plt.legend()
plt.savefig('actual_vs_predicted_poly.png')
```



7 Linear Regression with Look-Back Feature

[10]:
```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data_normalized = scaler.fit_transform(df[['Close']].values)

# Function to prepare the data for linear regression
def create_dataset(data, look_back=20):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back), 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)

X, y = create_dataset(data_normalized)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,␣
 ↪shuffle=False)

# Reshape the input data for linear regression
X_train = X_train.reshape(-1, 20)
X_test = X_test.reshape(-1, 20)

# Build the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Denormalize the predictions and actual values
y_pred_denormalized = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_denormalized = scaler.inverse_transform(y_test.reshape(-1, 1))


# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test_denormalized, y_pred_denormalized)
print(f'Mean Squared Error (MSE): {mse}')
# Cross-Validation


cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
```

```
cv_mse_scores = -cv_scores
```

```
Mean Squared Error (MSE): 4.548183229476901
```

[11]:
```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data_normalized = scaler.fit_transform(df[['Close']].values)

# Function to prepare the data for linear regression
def create_dataset(data, look_back=20):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back), 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)

X, y = create_dataset(data_normalized)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,␣
 ↪shuffle=False)

# Reshape the input data for linear regression
X_train = X_train.reshape(-1, 20)
X_test = X_test.reshape(-1, 20)

# Build the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Denormalize the predictions and actual values
y_pred_denormalized = scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test_denormalized = scaler.inverse_transform(y_test.reshape(-1, 1))

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test_denormalized, y_pred_denormalized)
print(f'Mean Squared Error (MSE): {mse}')

# Cross-Validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_scores
print(f'Cross-Validation MSE Scores: {cv_mse_scores}')
print(f'Average Cross-Validation MSE: {np.mean(cv_mse_scores)}')
```

```
# Plot actual vs. predicted values
plt.figure(figsize=(12, 6))
plt.plot(df.index[-len(y_test_denormalized):], y_test_denormalized,␣
 ↪label='Actual Closing Price')
plt.plot(df.index[-len(y_test_denormalized):], y_pred_denormalized,␣
 ↪label='Predicted Closing Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Actual vs. Predicted Stock Price (Linear Regression)')
plt.legend()
plt.show()
```

```
Mean Squared Error (MSE): 4.548183229476901
Cross-Validation MSE Scores: [3.75548566e-09 5.51271240e-09 2.05451248e-08
1.41266552e-06
 7.36069294e-05]
Average Cross-Validation MSE: 1.5009881657762964e-05
```