Future Stock Market Price Prediction with Supervised Machine Learning Techniques

Shanna Wallace
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, TN, USA
swalla16@vols.utk.edu

John Paul Saia
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, TN, USA
jsaia1@vols.utk.edu

Riley Taylor
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, TN, USA
jtayl219@vols.utk.edu

***Abstract*—This report details the progress made in the development of a machine learning model to predict future stock market closing price. This supervised learning model, trained on a set of stock data from Berkshire Hathaway, uses linear regression to predict stock price seven days into the future, based on trading volume and stock prices. Multiple techniques for regression were evaluated and compared to find the optimal method. This paper explores the development process of our baseline model, as well plans for further project improvements.**

***Index Terms*—machine learning, linear regression models, predictive algorithms**

I. Introduction

Stock market prediction can be challenging. It requires comprehensive analysis of data to predict patterns and trends, which can be difficult for someone who is uneducated on trading stocks. This project aims to simplify that process by providing future stock price predictions, making it easier for companies and individuals alike to predict future stock market trends and make educated trading decisions. To accomplish this goal, we have built a supervised machine learning model that aims to accurately predict the closing stock price seven days into the future.

II. Dataset

To train this model, we utilized a dataset of Berkshire Hathaway stock data from 2015 to 2024, sourced from Kaggle.com [1]. Berkshire Hathaway is a multinational conglomerate holding company. It is led by Warren Buffett, one of the most famous investors in the world [3]. The company's performance draws interest from investors worldwide, making its data a valuable resource for the development of predictive stock trading models. By focusing on Berkshire Hathaway's stock, we hope to provide insights that reflect the behavior of a stable, influential stock, potentially offering a basis for generalizable predictive insights in the market.

This dataset includes the following variables for each trading day from January 2, 2015, to July 29, 2024: Date, Open, High, Low, Close, Adj Close, and Volume [1]. Date is the date on which the values were recorded, in the format yyyy-mm-dd. Open is the initial trading price of the stock on the given day, while Close is its final price. High and Low refer to the highest and lowest prices of that day, respectively. Adj Close, the adjusted closing price, is calculated after accounting for stock splits and dividend distributions, reflecting the change in price, adjusted for corporate activities over time [9]. The Volume variable represents the sum of all shares that have exchanged hands that day and reflects the liquidity of a security and the interest investors have in it [8]. All of these price variables are measured in USD, while Volume is measured in shares traded per day.

These elements are of pivotal importance for the analysis of stocks. For example, Volume will underline spikes in trading activity that often precede price changes [8], while High and Low may show intra-day volatility. Analyzing these variables helps us build features on the trends and interactions of the data. This is also done to increase the predictive accuracy of the model by adding indicators like a moving average or any other technical indicator in order to make more confident and informed decisions while trading in stocks.

III. Data Analysis

A. Target Selection

Since our goal is to predict the closing price seven days in the future, our chosen target value is 'Close 7 Days'. We created this target by shifting the 'Close' column values up by 7 rows, so that the 'Close' values correspond to the feature values from seven days prior.

B. Feature Selection

The variables in our dataset are 'Date', 'High', 'Low', 'Open', 'Close', 'Adj Close', and 'Volume'. To create our feature set from these variables, we began by dropping the 'Date' variable and replacing it with 'Day', an integer in the range 1 to n, where 1 is the earliest date in the dataset and n is the latest. This allows us to more easily process and graph our variables as a function of time. For development of our baseline model, we have elected to ignore the 'Adj Close' variable, initially focusing only on the remaining price values and volume.

To visualize the relationship between the change in the volume, the various price features, and the target value over time, we plotted these values vs 'Day' on a line graph. Since volume is not measured in USD like the price features, we used Min-max scaling to shrink all of the values to a range of 0.0 to 1.0 for easier visualization. As seen in Fig. 1, we observed that the price features and our target value have a close linear relationship, tending to increase and decrease together over time. However, the volume feature does not appear to share this same type of linear relationship with the target.

C. Feature Correlation Analysis

To further evaluate the relationship between our features and the target, as well as the relationship between the features themselves, we created a correlation matrix, seen in Fig. 2. This provides a correlation coefficient that describes the linear relationship between each of the variables, with values closer to -1.0 indicating a closer inverse correlation, values closer to 1.0 indicating a closer positive correlation, and values closer to 0.0 indicating little to no correlation between the variables [10].

To narrow down our feature set to only the most relevant features, we dropped any feature with a correlation coefficient of less than 0.75 with the target, since these features with a weaker relationship will be less useful in training our model. This eliminated the 'Volume' feature, which only had a correlation coefficient of -0.056.

Next we examined the correlation coefficients between the remaining features, 'Open', 'Close', 'High', and 'Low.' We found that all four features were very highly correlated with one another, with correlation coefficients greater than 0.99. This means these features are all closely related, a condition called multicollinearity [11]. Features with multicollinearity provide similar information to our model in terms of predicting the target value, so including all of them in the feature set introduces redundant information that could negatively affect the performance of our model. As a result, we decided to keep only one of these features for the initial training of our baseline model, the 'Close' feature.

D. Data Cleaning and Normalization

Since we created our target by shifting the 'Close' column by 7 rows, this left the last 7 rows of the dataset with 'NaN' values for 'Close 7 Days'. To remedy this, we dropped these rows from our dataset.

We initially scaled the values for every variable down to a range of 0.0 to 1.0. This allowed us to more evenly compare the relationship between 'Volume' and the price features, since 'Volume' does not share the same units. Since we ultimately dropped the 'Volume' feature after performing feature correlation analysis, we did not scale the data in our final feature set and target.

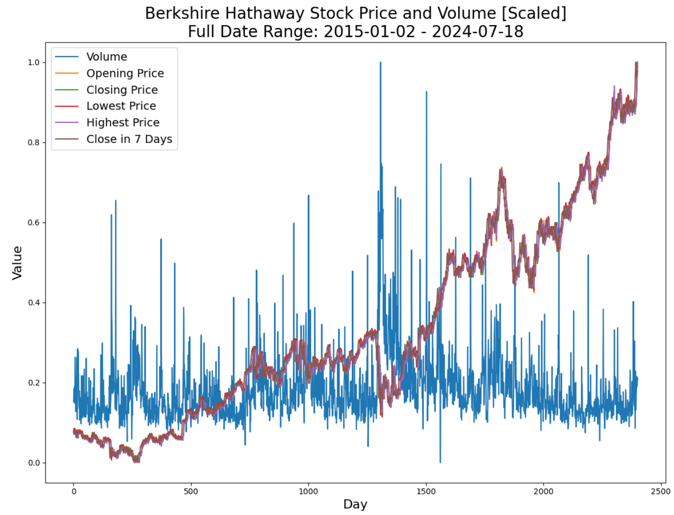


Fig. 1. Change in Price and Volume Over Time (Full Dataset)

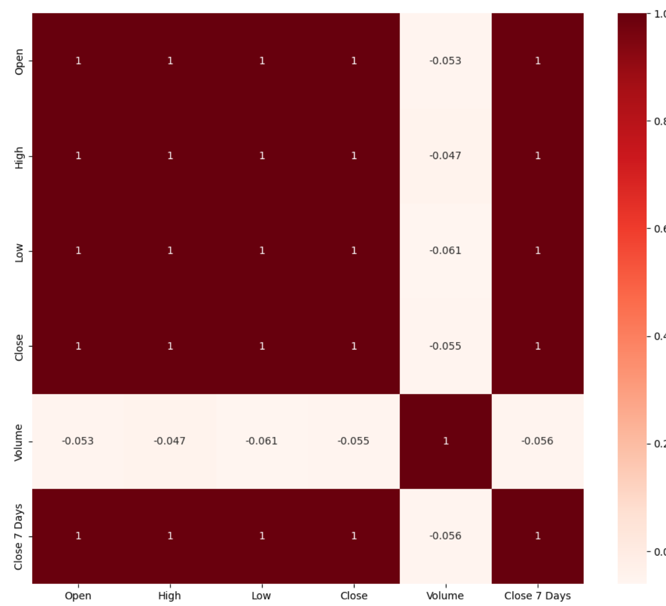


Fig. 2. Feature Correlation Matrix

E. Creating Training and Testing Sets

For our baseline model, we chose to use an 80/20 split for our data, with 80% of the samples being randomly assigned to the training set and the remaining 20% to the test set.

IV. Building the Baseline Model

A. Model Selection

We began by creating a baseline machine learning model which will be used to gauge performance improvement as future changes are made to the model. Since we observed a strongly correlated linear relationship between our price variables and target in our dataset, we decided that linear regression would be the most appropriate choice for our baseline model. Linear regression models are relatively simple to implement and easily interpretable, making them a good starting point for our project.

There are multiple methods that can be used to fit a linear regression model. To choose the best method for our predictive baseline model, we tested and compared the performance of three of these methods, beginning with Ordinary Least Squares and Gradient Descent. We also built a regression model with polynomial expansion to determine if higher complexity and a non-linear regression may perform better than linear regression for our project.

B. Performance Metrics

To evaluate the performance of these three models, we calculated and compared the values for R-Squared (R2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). R2 is the coefficient of determination and measures the proportion of variance in the dependent variable that can be explained by the dependent variable in the model. It ranges from 0.0 - 1.0, with values closer to 1.0 indicating better goodness-of-fit [5].

MAE and RMSE are metrics used to evaluate the accuracy of a model's predictions [12]. MAE calculates the average absolute differences between the model's predicted output and the actual values. An MAE score close to 0.0 means that the model's predictions are close to the actual values expected, indicating higher accuracy. RMSE measures the model's prediction error by taking the square root of the average of the squared differences between the predicted and actual values. As with MAE, an RMSE value closer to 0.0 indicates the model is predicting values closer to the actual values. Therefore the model with the highest R2 and lowest MAE and RMSE values is likely to be the most accurate and reliable.

C. Implementation with SKLearn and OLS

We first created a linear regression model using Scikit Learn's default linear regression class, which utilizes the Ordinary Least Squares (OLS) method to determine best fit [4]. OLS optimizes fit by calculating the parameters for the line that minimize the sum of squared residuals, or the difference between our model's expected output and its actual output [5].

D. Regression with Polynomial Expansion

Next, we created a more advanced model implementing polynomial expansion. This allows us to capture more complex, non-linear relationships between features by adding polynomial terms. To determine the best degree to use for our model, we tested degrees 1-10, calculating RMSE and R2 for each, as well as plotting the learning curves and fit for each to visualize the performance. Degree 6 had the lowest RMSE and R2 of the values tested, so we chose this value to initialize our gradient descent model.

E. Linear Regression with Gradient Descent

Finally, we implemented a linear regression model using gradient descent. Gradient descent is an optimization algorithm that iteratively adjusts the model's parameters to minimize the error in predicting output values. At each iteration, our model calculates predictions, computes the cost between the predicted and actual values using mean squared error (MSE) as the cost function, then calculates the gradient descent based on these values, and finally adjusts the parameters by subtracting the gradient, scaled by the learning rate.

To find the optimal learning rate, we applied gradient descent with learning rates 0.001, 0.005, 0.01, 0.05, and 0.1, and plotted the cost function convergence and fit to visualize the results for each trial. As the best learning rate is one that minimizes cost in the least number of iterations without diverging, we determined that a learning rate of 0.01 was the best choice of the values tested.

TABLE I
REGRESSION METHOD PERFORMANCE METRICS

	R2	**RMSE**	**MAE**
OLS	0.991199	7.380728	5.207674
Polynomial Expansion	0.991270	7.351159	5.154775
Gradient Descent	0.991198	7.381360	5.208097

V. RESULTS

A. Regression Method Comparison

The results shown in Table 1 were nearly the same for all three models tested, with the polynomial expansion model performing slightly better than the others with the lowest RSME, lowest MAE, and highest R2 values. Due to the strong linear relationship present amongst the variables in our dataset, increasing the model's complexity to capture non-linear relationships with polynomial expansion did not offer any significant benefit. Likewise, due to the relatively small size and simplistic nature of our dataset, the ability to process large amounts of data and model complex relationships with gradient descent will be unnecessary for our task. Since SKLearn's default OLS method is the simplest to implement and is computationally efficient for relatively small datasets such as ours, we believe it is the best choice for our baseline model.

B. Baseline Model Performance Analysis

The OLS method had an R2 value of 0.991199, meaning over 99% of changes in our dependent variable, Cost in 7 Days, can be explained by changes in our independent variable, Cost. This indicates a high goodness-of-fit and strong relationship between our input data and model output [6]. Our MAE of 5.207674 means that on average, our model's predictions differed from the actual values by about \$5.21. RMSE, which also measures the average difference between actual values and model output, was about \$7.38. Since the residual values are squared with RMSE, it gives more weight to larger errors, so outliers were given a higher penalty in this calculation than they are with MAE [12].

Overall, based on our chosen performance metrics, our baseline model performed with a high degree of accuracy in our testing and appears to model the relationship between our input feature and output very well. However, it is possible that our calculated R2 values could be somewhat inflated due to bias or variance in our model [6]. As shown in Fig. 3, when plotting our training error along with our testing error when performing gradient descent, our model had higher training error and lower testing error, indicating that our model may be underfitting and failing to capture enough complexity in our data [7].

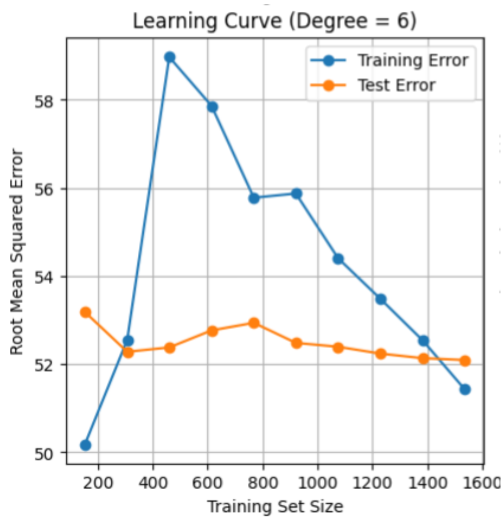


Fig. 3. Learning Curve

VI. Future Work and Improvements

We plan to perform more thorough exploratory data analysis, such as outlier detection methods. Outliers can skew model predictions and impact performance, especially with financial data because of how much it fluctuates, so we plan to implement methods to minimize the influence of extreme values [13]. We also plan to experiment with various data scaling methods to provide a clearer view of volume price interactions. We hope that this will improve our model's performance, as well as enhance the interpretability of trading volume as a predictor. Data smoothing techniques can ensure that our model focuses on more consistent patterns within the data, improving accuracy and making predictions more resilient to volatility in the market.

To improve our model's predictions, we also plan on expanding the feature set with more variables from the data and developing a model that may be able to capture complex interactions between these features more effectively. We will also experiment with the addition of new features created from our dataset, such as moving averages over 7, 14, and 21 days and standard deviation over 7 days, as these metrics have shown promise in stock price prediction models [14]. The relative strength index and Bollinger bands will be explored as well, as these features can help identify short term trends and volatility and provide more insight on when stocks are overbought or oversold [2]. We are also going to evaluate other splitting methods for our training and testing data, and are considering the addition of a validation set to allow us to more effectively fine-tune our model and increase its performance with unseen data. By splitting our data into training, validation, and test sets, we are going to ensure that our model generalizes well to new data rather than just memorizing the training set.

VII. Distribution of Work

Project code was written collaboratively.
Contributions to this report were made as follows:
Shanna Wallace: Abstract, Data Analysis, Building the Baseline Model, and Results.
JP Saia: Introduction, Dataset, and Future Work and Improvements.
Riley Taylor: Abstract, Introduction, Building the Baseline Model, and Results.

References

[1] U. Haddi. "Berkshire Hathaway Stock Price Data." Kaggle. Accessed: October 30, 2024. [Online]. Available: https://www.kaggle.com/datasets/umerhaddii/berkshire-hathaway-stock-price-data
[2] Tradingview. "Bollinger Bands (BB)." Accessed: October 20, 2024. Available: https://www.tradingview.com/scripts/bollingerbands/.
[3] W. Buffett. "A Message From Warren E. Buffett, CEO of Berkshire Hathaway Inc." Berkshire Hathaway. Accessed: October 20, 2024. Available: https://www.berkshirehathaway.com/message.html.
[4] Scikit Learn. "LinearRegression." Accessed: October 20, 2024. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LinearRegression.html.
[5] D. Signori. "Chapter 2: Ordinary Least Squares." Simon Fraser University. Accessed: October 20, 2024. Available: https://www.sfu.ca/ dsignori/buec333/lecture
[6] J. Frost. "Five Reasons Why Your R-squared can be Too High." Statistics By Jim. Accessed: October 20, 2024. Available: https://statisticsbyjim.com/regression/r-squared-too-high/.
[7] Amazon Web Services. "Model Fit: Underfitting vs. Overfitting." Accessed: October 20, 2024. Available: https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html.
[8] T. Dong. "Stock Volume Definition." U.S. News Money. Accessed: October 20, 2024. Available: https://money.usnews.com/investing/term/stock-volume.
[9] T. Corvin. "Adjusted Closing Price." The Trading Analyst. Accessed: October 20, 2024. Available: https://thetradinganalyst.com/adjusted-closing-price/.
[10] W3 Schools. "Data Science - Statistics Correlation Matrix." Accessed: October 20, 2024. Available: https://www.w3schools.com/datascience/ds_stat_correlation_matrix.asp.
[11] J. Frost. "Multicollinearity in Regression Analysis: Problems, Detection, and Solutions." Statistics By Jim. Accessed: October 24, 2024. Available: https://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/.
[12] The Data Scientist. "Performance measures: RMSE and MAE." Accessed: October 24, 2024. Available: https://thedatascientist.com/performance-measures-rmse-mae/.
[13] T. Firdose. "Understanding Outliers: Impact, Detection, and Remedies." Medium. Accessed: October 24, 2024. Available: https://tahera-firdose.medium.com/understanding-outliers-impact-detection-and-remedies-ea2192174477.
[14] I. Parmar, N. Agarwal, S. Saxena, R. Arora, S. Gupta, H. Dhiman. "Stock Market Prediction Using Machine Learning" IEEE. May 2, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8703332.

Stock Price Prediction - Simple Regression Model

COSC 325 Course Project
Fall 2024

Members:

- John Paul Saia

- Riley Taylor

- Shanna Wallace

Objective:

Create a machine learning model to predict the stock's closing price in 7 days

The Data

Data set: Berkshire Hathaway daily stock price and volume traded from 2015-01-02 to 2024-07-29
Format: .csv file

Target and features:

Target	Description
Close 7 Days	Closing price 7 days from the trading day

Features	Description
Date	The day the price data is from (yyyy-mm-dd)
Open	Opening price
High	Highest price
Low	Lowest price
Close	Closing price
Adj Close	Closing price after adjustments for applicable splits and dividend distributions
Volume	Total number of shares traded that day

The Baseline Model

- Create and compare 3 regression models using 1 feature:

    - Linear Regression with SKLearn's default class
    - Complex regression model with polynomial expansion.
    - Linear Regression with gradient descent.
- Optimize linear model performance.

Create Linear Regression Model:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error, r2_score,
mean_absolute_error
```

Load and Prepare Data:

- Load csv file contents to a DataFrame
- Add column for the target, Close 7 Days, by shifting Close by 7 days
- Add the additional features
- Clean the data by removing rows with missing values

Additional Features:

Features	Description
High - Low	Difference between highest and lowest price
Open - Close	Amount price changed from open to close
7 Day STD DEV	Standard deviation of closing price over previous 7 days
7 Day MA	Moving averages of closing price over previous 7 days
14 Day MA	Moving averages of closing price over previous 14 days
21 Day MA	Moving averages of closing price over previous 21 days

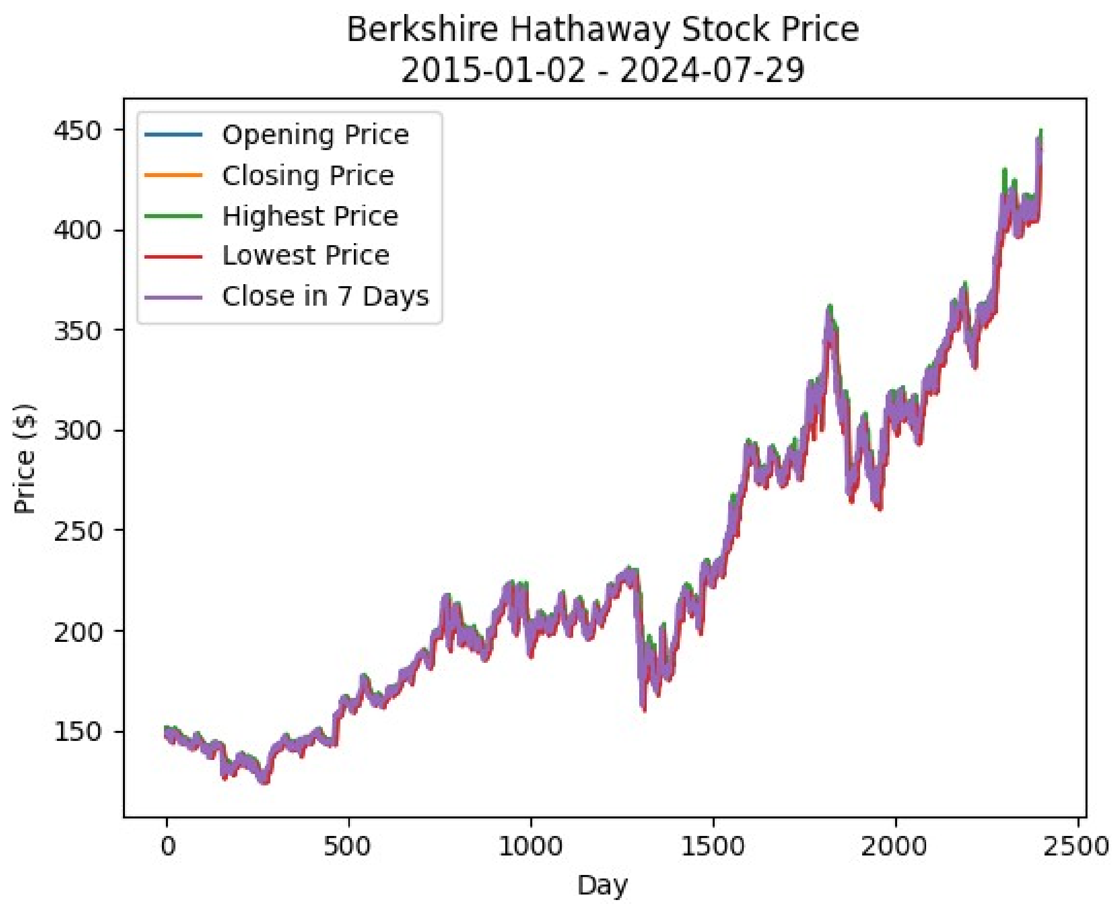
```
RANDOM_STATE = 42
file = "./berkshire_hathaway_data.csv"
stock_data = pd.read_csv(file)

# Target is next day's closing price
```

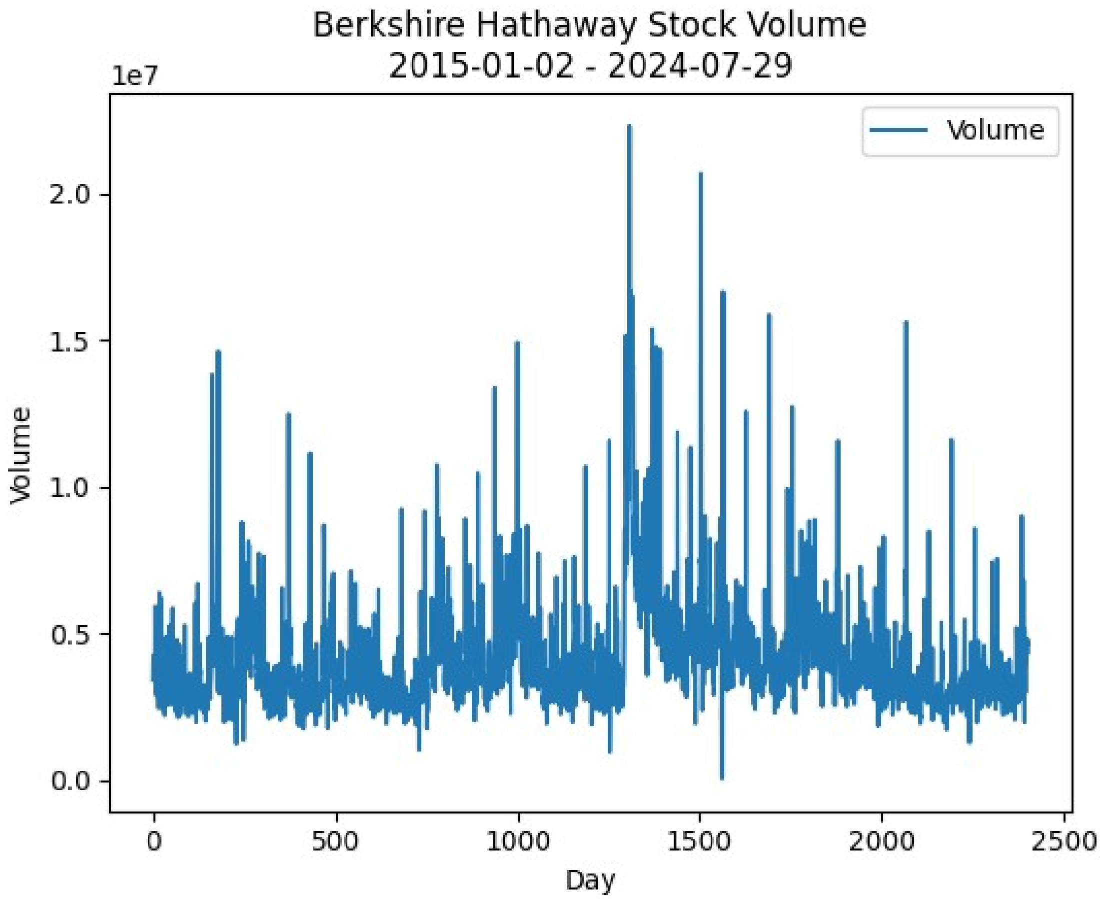
```
# So want to target row i to correspond to row i-1 in feature columns
# Add a column called next day's close
stock_data['Close 7 Days'] = stock_data['Close'].shift(-7)

stock_data['Day'] = np.arange(1, len(stock_data.index)+1)
stock_data = stock_data[['Day', 'Open', 'High', 'Low', 'Close',
'Volume', 'Close 7 Days']]


# Drop the rows with missing data
stock_data = stock_data.dropna().reset_index(drop=True)

stock_data.head()
```

	Day	Open	High	Low	Close	Volume	Close 7 Days
0	1	151.500000	151.600006	148.500000	149.169998	3436400	148.630005
1	2	148.809998	149.000000	146.779999	147.000000	4168800	147.820007
2	3	147.639999	148.529999	146.110001	146.839996	4116100	147.580002
3	4	147.940002	149.139999	147.649994	148.880005	4159100	149.210007
4	5	150.600006	151.369995	150.509995	151.369995	4282100	148.630005

Change in Stock Prices Over Time

```
plt.plot(stock_data['Day'], stock_data['Open'], label="Opening Price")
plt.plot(stock_data['Day'], stock_data['Close'], label="Closing
Price")
plt.plot(stock_data['Day'], stock_data['High'], label="Highest Price")
plt.plot(stock_data['Day'], stock_data['Low'], label="Lowest Price")
plt.plot(stock_data['Day'], stock_data['Close 7 Days'], label="Close
in 7 Days")
plt.title("Berkshire Hathaway Stock Price\n2015-01-02 - 2024-07-29")
plt.xlabel("Day")
plt.ylabel("Price ($)")
plt.legend()
plt.show()

plt.plot(stock_data['Day'], stock_data['Volume'], label="Volume")
plt.title("Berkshire Hathaway Stock Volume\n2015-01-02 - 2024-07-29")
plt.xlabel("Day")
plt.ylabel("Volume")
plt.legend()
plt.show()
```

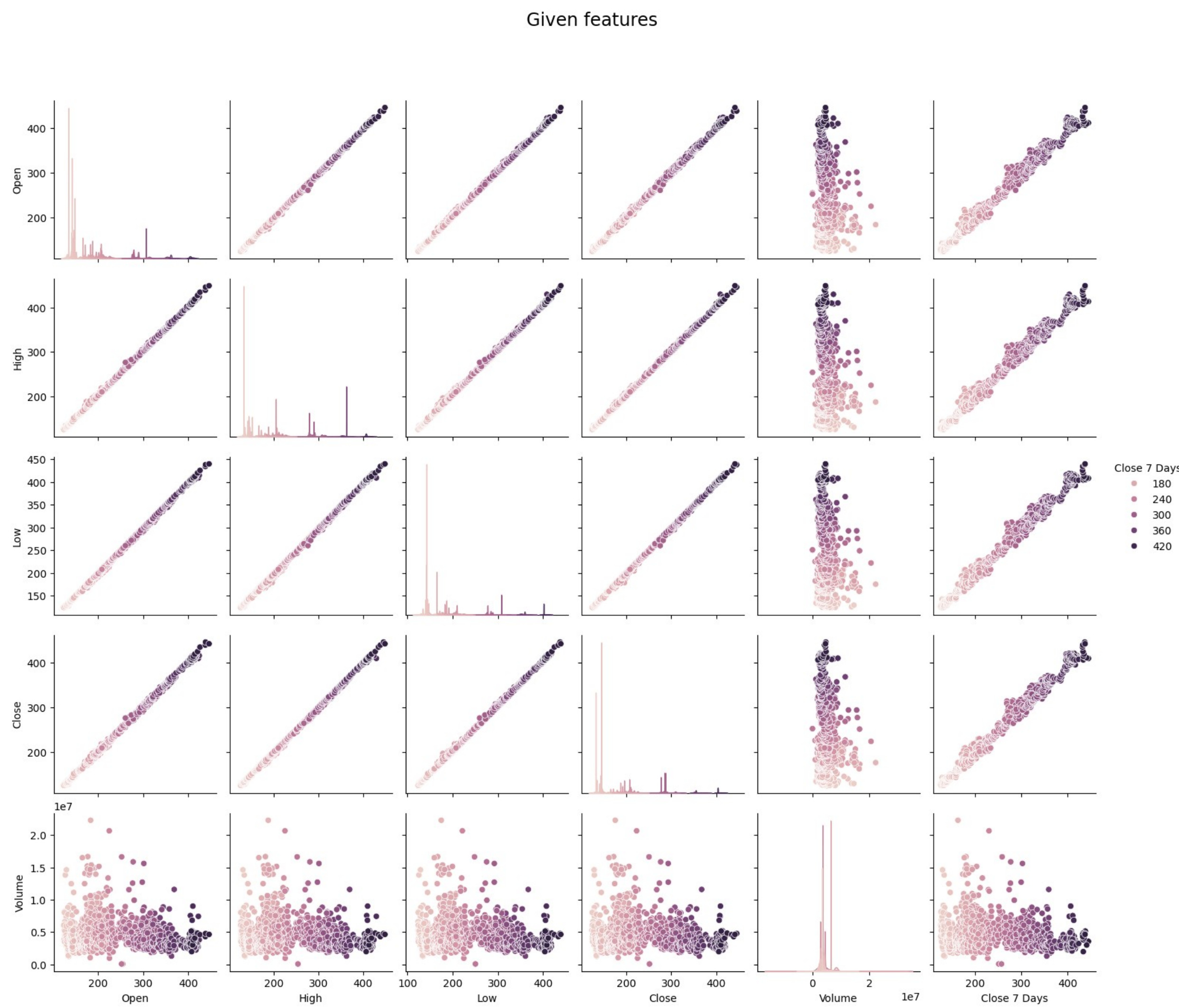

Berkshire Hathaway Stock Price
2015-01-02 - 2024-07-29
Opening Price
Closing Price
Highest Price
Lowest Price
Close in 7 Days
Price ($)
Day
450
400
350
300
250
200
150
0
500
1000
1500
2000
2500



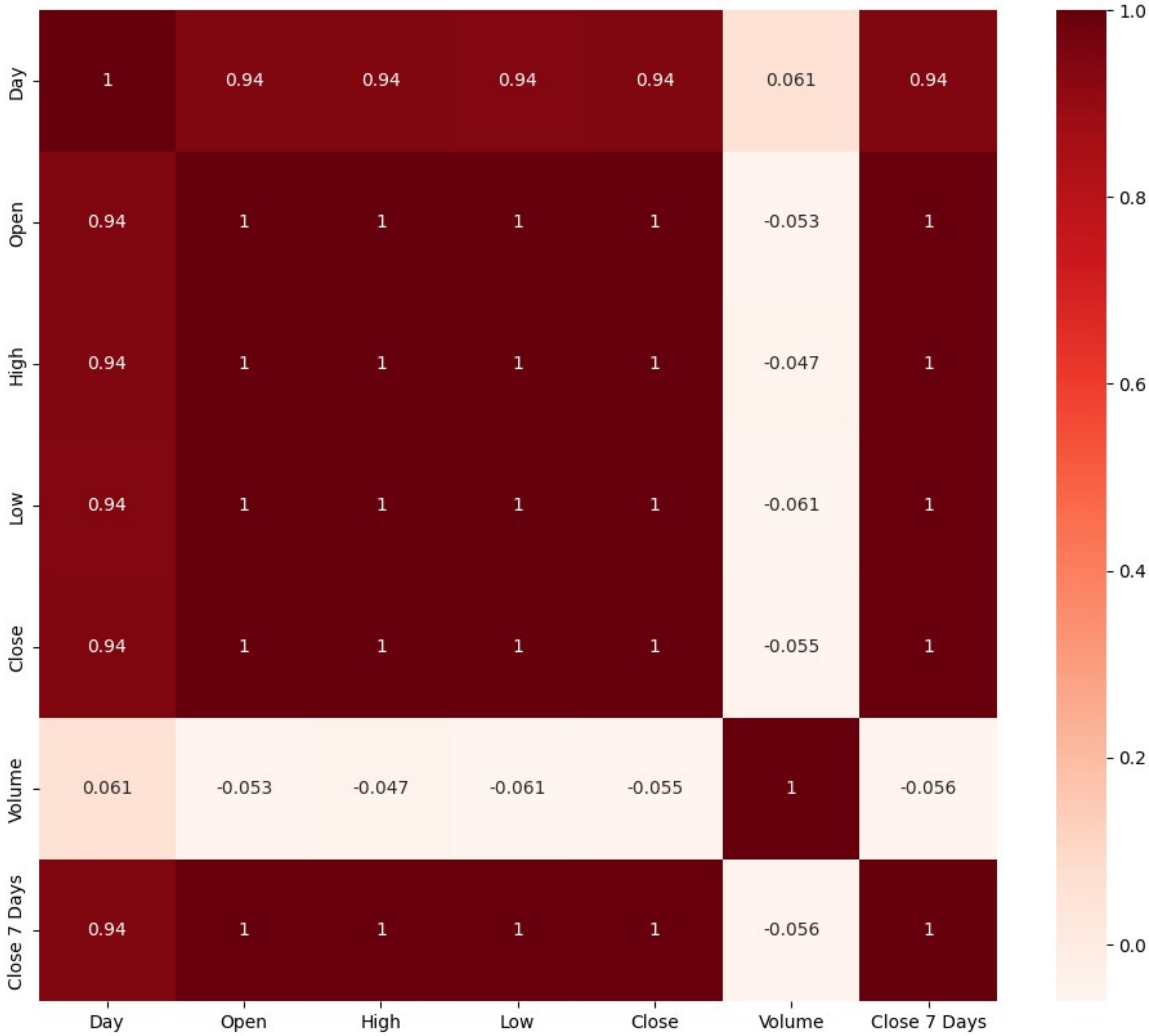
Feature Selection:

- Generate scatter plot to visualize relationships between variables
- Calculate correlation coefficient of each feature with the target and drop features with correlation value less than 0.5
- Look at the correlation coefficient of the features with the other features and drop ones that are highly correlated with each other

```
# Display features data trends for the given features with Seaborn
pairplot.
given_features = sns.pairplot(stock_data, x_vars=['Open', 'High',
'Low', 'Close', 'Volume', 'Close 7 Days'],
                 y_vars=['Open', 'High', 'Low', 'Close', 'Volume'],
hue='Close 7 Days')
given_features.figure.suptitle("Given features", y=1.08, fontsize='xx-
large')

Text(0.5, 1.08, 'Given features')
```




```
#stock_data = stock_data.drop(columns=["Day"])

#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = stock_data.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()

print(cor)

#Correlation with output variable
cor_target = abs(cor["Close 7 Days"])

#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.75]
print(relevant_features)
```




```
                  Day      Open      High       Low     Close
Volume  \
Day          1.000000  0.942059  0.943111  0.941314  0.942176
0.060807
Open         0.942059  1.000000  0.999815  0.999776  0.999570 -
0.053169
High         0.943111  0.999815  1.000000  0.999725  0.999792 -
0.047492
Low          0.941314  0.999776  0.999725  1.000000  0.999799 -
0.061293
Close        0.942176  0.999570  0.999792  0.999799  1.000000 -
0.055106
Volume       0.060807 -0.053169 -0.047492 -0.061293 -0.055106
1.000000
Close 7 Days 0.942154  0.995257  0.995485  0.995595  0.995708 -
```

```
0.056169

                Close 7 Days
Day                 0.942154
Open                0.995257
High                0.995485
Low                 0.995595
Close               0.995708
Volume             -0.056169
Close 7 Days        1.000000
Day               0.942154
Open              0.995257
High              0.995485
Low               0.995595
Close             0.995708
Close 7 Days      1.000000
Name: Close 7 Days, dtype: float64
```

Create Training and Test Sets

- Extract target and selected features
- Split data in to training and testing sets
  - Training 80% / Testing 20%

```
# Drop relevant features that are highly correlated with each other
relevant_feature_list = ["Close"]

X_relevant = stock_data[relevant_feature_list]
y = stock_data['Close 7 Days']

# Separate data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_relevant, y,
test_size=0.2, random_state=RANDOM_STATE)

# Print our selected features and set sizes
print(f"Training set size: {X_train.shape[0]}\nTesting set size:
{X_test.shape[0]}")

Training set size: 1920
Testing set size: 481
```

Simple Linear Regression Model with OLS

Create and Fit Linear Model using sklearn's default, Ordinary Least Squares

```
# Create the simple Model
ols_model = LinearRegression()
```

```python
# Create training sets and train the model
ols_X_train = X_train
ols_X_test = X_test

ols_model.fit(ols_X_train, y_train)
```

```
LinearRegression()
```

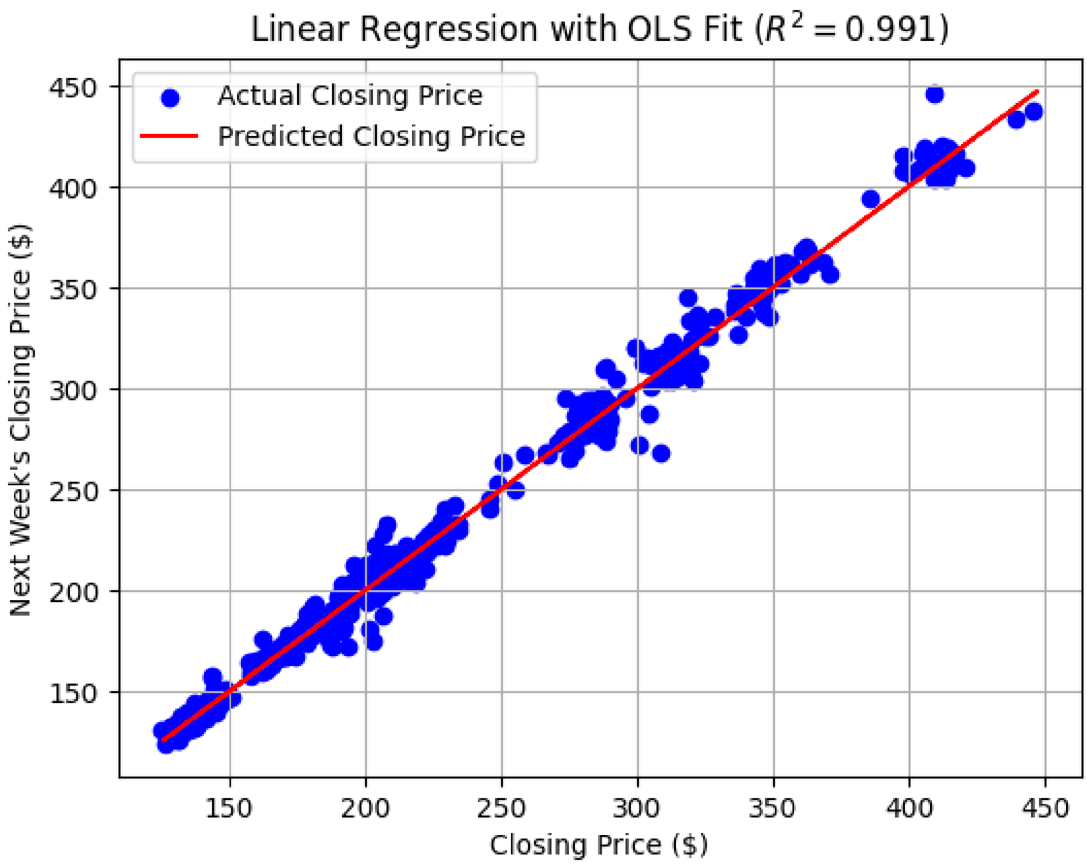
Get Predictions and Analyze Performance

- Plot the model
- Calculate performance metrics:
  - Bias
  - Variance
  - RMSE (Root Mean Squared Error)
  - MAE (Mean Absolue Error)

```python
# simple_pred = simple_model.predict(simple_X_test)
ols_pred = pd.Series(ols_model.predict(X_test), index=X_test.index)

# Calculate the mean of the predictions (expected prediction)
ols_mean_pred = np.mean(ols_pred)

# Calculate bias, variance, and Root Mean Squared Error (RMSE) on the
test set
ols_bias = np.mean((y_test - ols_mean_pred) ** 2)
ols_variance = np.mean((ols_pred - ols_mean_pred) ** 2)
ols_rmse = root_mean_squared_error(y_test, ols_pred)
ols_r2 = r2_score(y_test, ols_pred)
ols_mae = mean_absolute_error(y_test, ols_pred)

print(f"Simple Model:\n  Bias: {ols_bias}\n  Variance: {ols_variance}\
n  RMSE: ${ols_rmse}\n  MAE: ${ols_mae}")

plt.scatter(ols_X_test, y_test, label='Actual Closing Price',
color='blue')
plt.plot(ols_pred, ols_pred, label='Predicted Closing Price',
color='red')
plt.title(f"Closing Price vs. Next Week's Closing Price (Simple Linear
Model)")
plt.title(f"Linear Regression with OLS Fit ($R^2=${ols_r2:.3f})")
plt.xlabel(f"Closing Price ($)")
plt.ylabel("Next Week's Closing Price ($)")
plt.legend()
plt.grid(True)
```

```
Simple Model:
  Bias: 6189.905271899396
  Variance: 6061.442030563376
```

```
RMSE: $7.380728265514138
MAE: $5.207673919675837
```



Simple Linear Model with Polynomial Expansion

Determine optimal number of degrees to use:

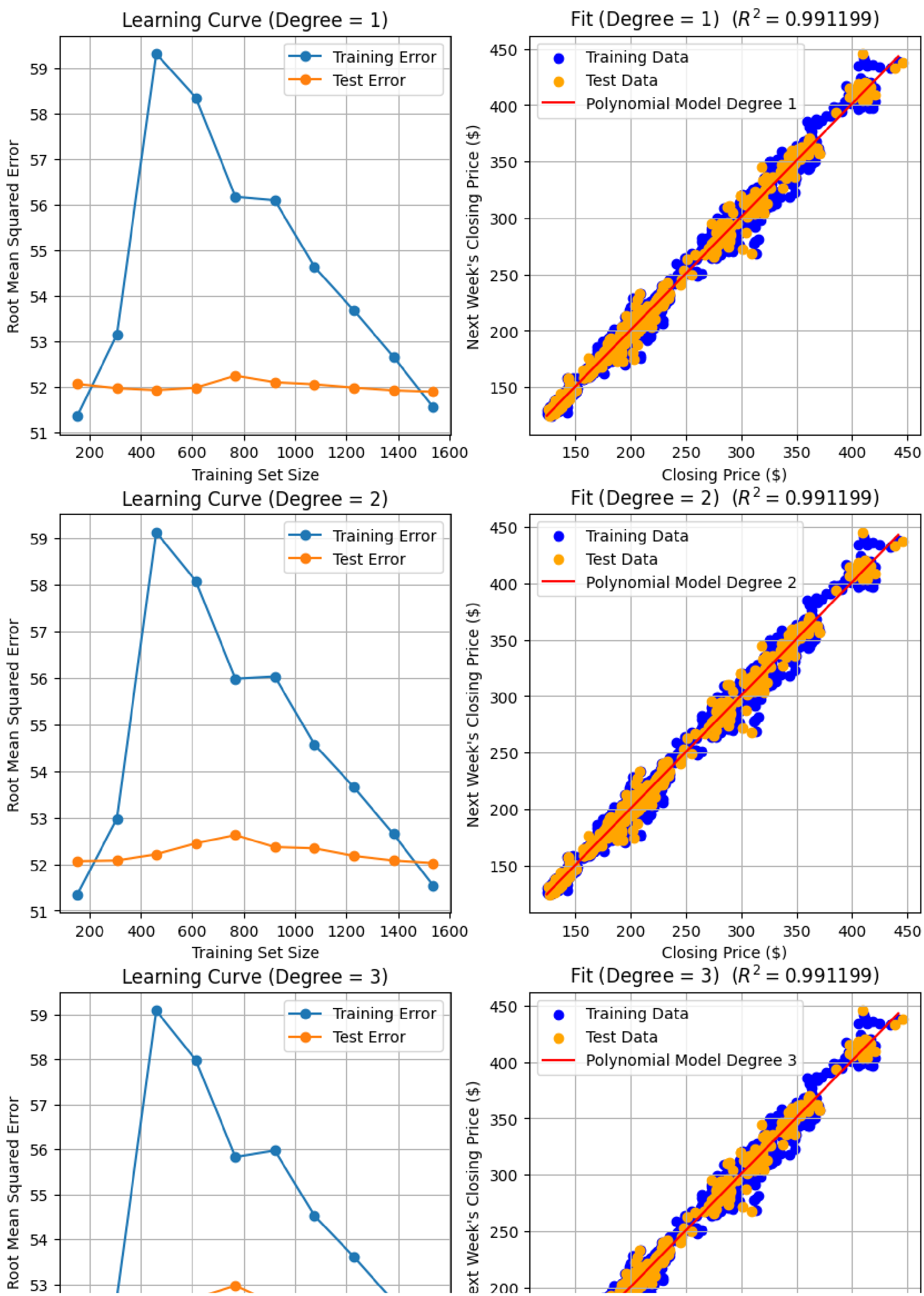
- Test degree values 1 through 10
- Compare R2 and RMSE for each
- Choose degree number that gives highest R2 and lowest RMSE

```
rmse_results = [0] * 10
r2_results = [0] * 10

fig, axes = plt.subplots(10, 2, figsize=(10,55))
# row for the subplot
r = 0

# Define a function for plotting the learning curve
for degree in range(1, 11):
    # Complex Model: Using the features with Polynomial Expansion
    poly = PolynomialFeatures(degree=degree)
```

```
    # Transform the training and test data
    X_train_poly = poly.fit_transform(X_train.values)
    X_test_poly = poly.transform(X_test.values)

    # Initialize the linear regression model
    poly_model = LinearRegression()
    poly_model.fit(X_train_poly, y_train)

    poly_test_predictions = poly_model.predict(X_test_poly)

    # Calculate the mean of the predictions (expected prediction)
    poly_mean_prediction = np.mean(poly_test_predictions)

    # Calculate learning curves for the polynomial model
    train_sizes, train_scores, test_scores = learning_curve(
        poly_model, X_train_poly, y_train,
scoring='neg_mean_squared_error',
        train_sizes=np.linspace(0.1, 1.0, 10),
random_state=RANDOM_STATE
    )

    # Calculate mean error
    train_errors = -np.mean(train_scores, axis=1)
    test_errors = -np.mean(test_scores, axis=1)


    # set row and column for the subplot

    c = 0

    # Plot learning curves
    axes[r,c].plot(train_sizes, train_errors, label='Training Error',
marker='o')
    axes[r,c].plot(train_sizes, test_errors, label='Test Error',
marker='o')
    axes[r,c].set_title(f'Learning Curve (Degree = {degree})')
    axes[r,c].set_xlabel('Training Set Size')
    axes[r,c].set_ylabel('Root Mean Squared Error')
    axes[r,c].legend()
    axes[r,c].grid(True)

    c += 1

    # Calculate Root Mean Squared Error and R2
    rmse_results[degree - 1] = root_mean_squared_error(y_test,
poly_test_predictions)
    r2_results[degree - 1] = r2_score(y_test, poly_test_predictions)

    # Visualize the learning process
    x_curve = np.linspace(X_train.min(), X_train.max(), 100).reshape(-
```

```
1, 1)
    y_curve_poly = poly_model.predict(poly.transform(x_curve))


    axes[r,c].scatter(X_train, y_train, color='blue', label='Training
Data')
    axes[r,c].scatter(X_test, y_test, color='orange', label='Test
Data')

    axes[r,c].plot(x_curve, y_curve_poly, color='red',
label=f'Polynomial Model Degree {degree}')
    axes[r,c].set_title(f'Fit (Degree = {degree})  ($R^2=$
{r2_results[degree - 1]:.6f})')
    axes[r,c].set_xlabel("Closing Price ($)")
    axes[r,c].set_ylabel("Next Week's Closing Price ($)")
    axes[r,c].legend()
    axes[r,c].grid(True)

    r += 1

# Want to minimize errors and maximize R2
max_r2 = max(r2_results)
min_rmse = min(rmse_results)
print(f"\nDegree with maximum R2: {r2_results.index(max_r2) + 1} with
{max_r2}")
print(f"Degree with minimum RMSE: {rmse_results.index(min_rmse) + 1}
with ${min_rmse}")
```

```
Degree with maximum R2: 6 with 0.9912695991682327
Degree with minimum RMSE: 6 with $7.351159603326189
```



Learning Curve (Degree = 1)
Training Error
Test Error
Root Mean Squared Error
Training Set Size
Fit (Degree = 1) ($R^2 = 0.991199$)
Training Data
Test Data
Polynomial Model Degree 1
Next Week's Closing Price ($)
Closing Price ($)
Learning Curve (Degree = 2)
Training Error
Test Error
Root Mean Squared Error
Training Set Size
Fit (Degree = 2) ($R^2 = 0.991199$)
Training Data
Test Data
Polynomial Model Degree 2
Next Week's Closing Price ($)
Closing Price ($)
Learning Curve (Degree = 3)
Training Error
Test Error
Root Mean Squared Error
Fit (Degree = 3) ($R^2 = 0.991199$)
Training Data
Test Data
Polynomial Model Degree 3

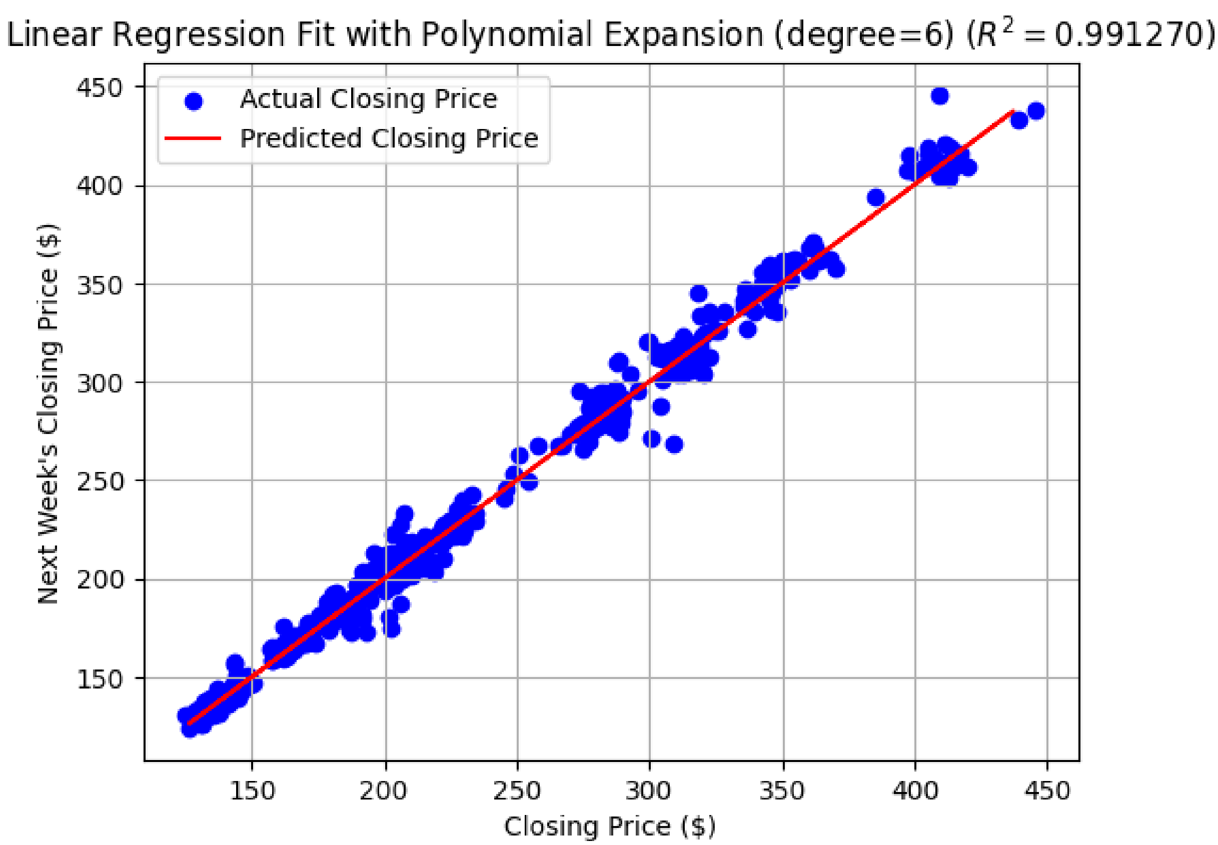
Create Linear Model with Polynomial Expansion using Best Degree

Get Predictions and Analyze Performance

- Plot the model
- Calculate performance metrics:
  - Bias
  - Variance
  - RMSE (Root Mean Squared Error)
  - MAE (Mean Absolue Error)

```
degree = rmse_results.index(min_rmse) + 1

# Complex Model: Using the features with Polynomial Expansion
poly = PolynomialFeatures(degree=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)
poly_pred = poly_model.predict(X_test_poly)

# Calculate the mean of the predictions (expected prediction)
poly_mean_pred = np.mean(poly_pred)

# Calculate bias, variance, and Root Mean Squared Error (RMSE) on the
test set
poly_bias = np.mean((y_test - poly_mean_pred) ** 2)
poly_variance = np.mean((poly_pred - poly_mean_pred) ** 2)
poly_rmse = root_mean_squared_error(y_test, poly_pred)
poly_r2 = r2_score(y_test, poly_pred)
poly_mae = mean_absolute_error(y_test, poly_pred)

print(f"Regression Model with Polynomial Expansion:\n  Bias:
{poly_bias}\n  Variance: {poly_variance}\n  RMSE: ${poly_rmse}\n  MAE:
${poly_mae}")

plt.scatter(X_test, y_test, label='Actual Closing Price',
color='blue')
plt.plot(poly_pred, poly_pred, label='Predicted Closing Price',
color='red')
plt.title(f"Linear Regression Fit with Polynomial Expansion
(degree={degree}) ($R^2=${poly_r2:.6f})")
plt.xlabel(f"Closing Price ($)")
plt.ylabel("Next Week's Closing Price ($)")
plt.legend()
plt.grid(True)
```

```
Linear Model with Polynomial Expansion:
  Bias: 6189.933611006034
  Variance: 6046.653570281796
  RMSE: $7.351159603326189
  MAE: $5.154774973438695
```



Simple Linear Regression Model with Gradient Descent

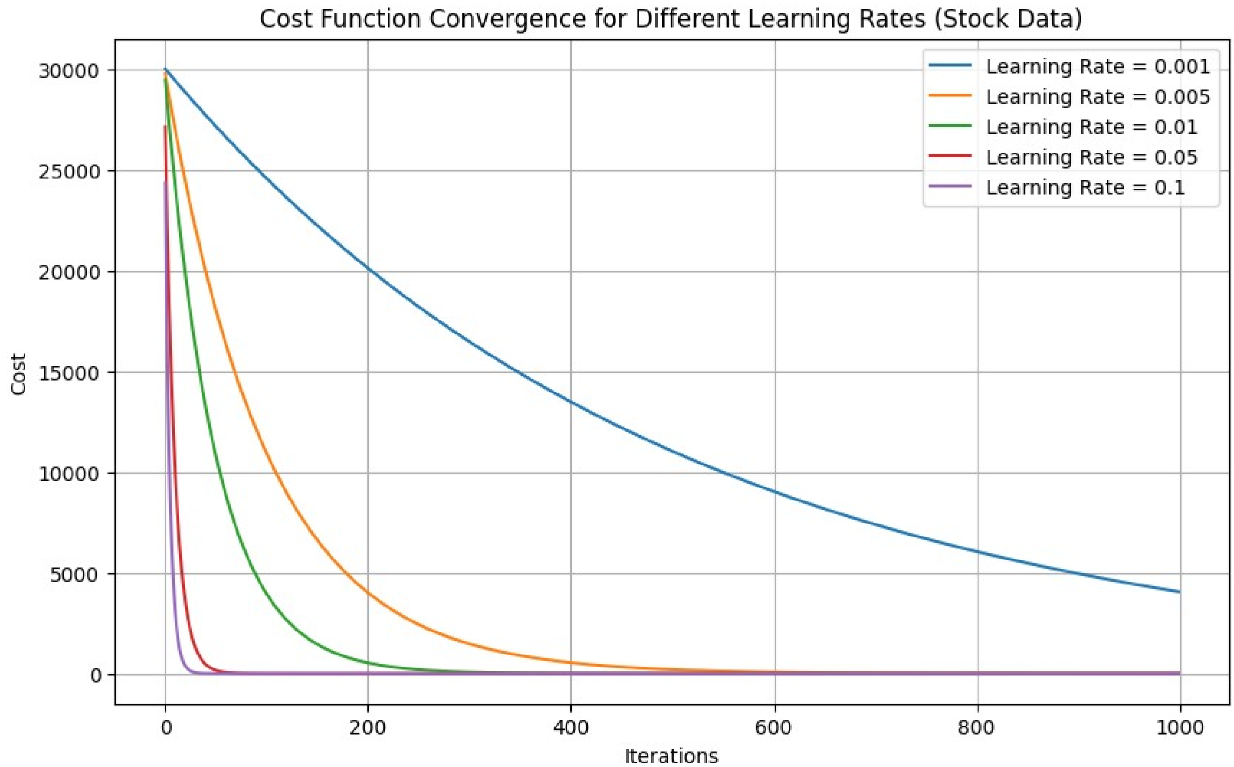
Choose best learning rate

- Test different **learning rates** to find the **optimal learning rate** that minimizes the cost function most efficiently

```
# Define the cost function
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1/(2*m)) * np.sum(np.square(predictions - y))
    return cost

# Define the gradient descent function (with cost history tracking)
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)  # number of samples
    cost_history = np.zeros(iterations)  # To store the cost at each
```

```
iteration

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        gradient = (1/m) * X.T.dot(errors)
        theta = theta - alpha * gradient
        cost_history[i] = compute_cost(X, y, theta)  # Save the cost
at each iteration

    return theta, cost_history

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Add a column of ones to the scaled feature matrices for the
intercept term
X_train_scaled = np.c_[np.ones(X_train_scaled.shape[0]),
X_train_scaled]
X_test_scaled = np.c_[np.ones(X_test_scaled.shape[0]), X_test_scaled]


# Initialize parameters for gradient descent
iterations = 1000
theta = np.zeros(X_train_scaled.shape[1])  # Initialize theta with
zeros


# Test with different learning rates
learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1]
plt.figure(figsize=(10, 6))

for alpha in learning_rates:
    theta = np.zeros(X_train_scaled.shape[1])  # Reset theta for each
learning rate
    theta, cost_history = gradient_descent(X_train_scaled, y_train,
theta, alpha, iterations)
    plt.plot(range(iterations), cost_history, label=f'Learning Rate =
{alpha}')


# Plot the cost function convergence
plt.title('Cost Function Convergence for Different Learning Rates
(Stock Data)')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

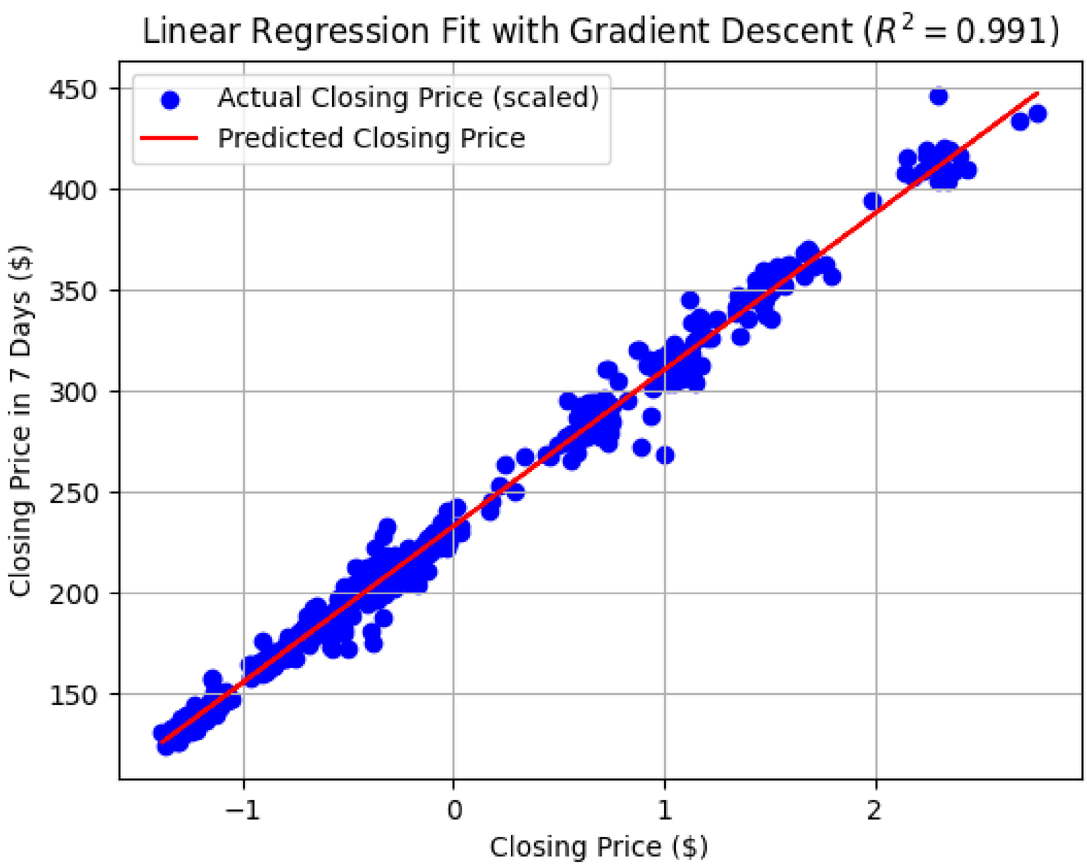


Apply Gradient Descent

- Using the optimal learning rate (alpha) of 0.01 based on the above graph

```
# Gradient Descent with optimal learning rate
alpha = 0.01
theta = np.zeros(X_train_scaled.shape[1])
theta, cost_history = gradient_descent(X_train_scaled, y_train, theta,
alpha, iterations)

# get the predictions for the final theta
gd_pred = np.dot(X_test_scaled, theta)
gd_mean_pred = np.mean(gd_pred)

# RMSE, R2, MAE, bias, and variance
gd_bias = np.mean((y_test - gd_mean_pred) ** 2)
gd_variance = np.mean((poly_pred - gd_mean_pred) ** 2)
gd_rmse = root_mean_squared_error(y_test, gd_pred)
gd_r2 = r2_score(y_test, gd_pred)
gd_mae = mean_absolute_error(y_test, gd_pred)

plt.scatter(X_test_scaled[:,1], y_test, label='Actual Closing Price
(scaled)', color='blue')
```

```
plt.plot(X_test_scaled[:,1], gd_pred, label='Predicted Closing Price',
color='red')
plt.title(f"Linear Regression Fit with Gradient Descent ($R^2=$
{gd_r2:.3f})")
plt.xlabel(f"Closing Price ($)")
plt.ylabel("Closing Price in 7 Days ($)")
plt.legend()
plt.grid(True)
plt.show()
```



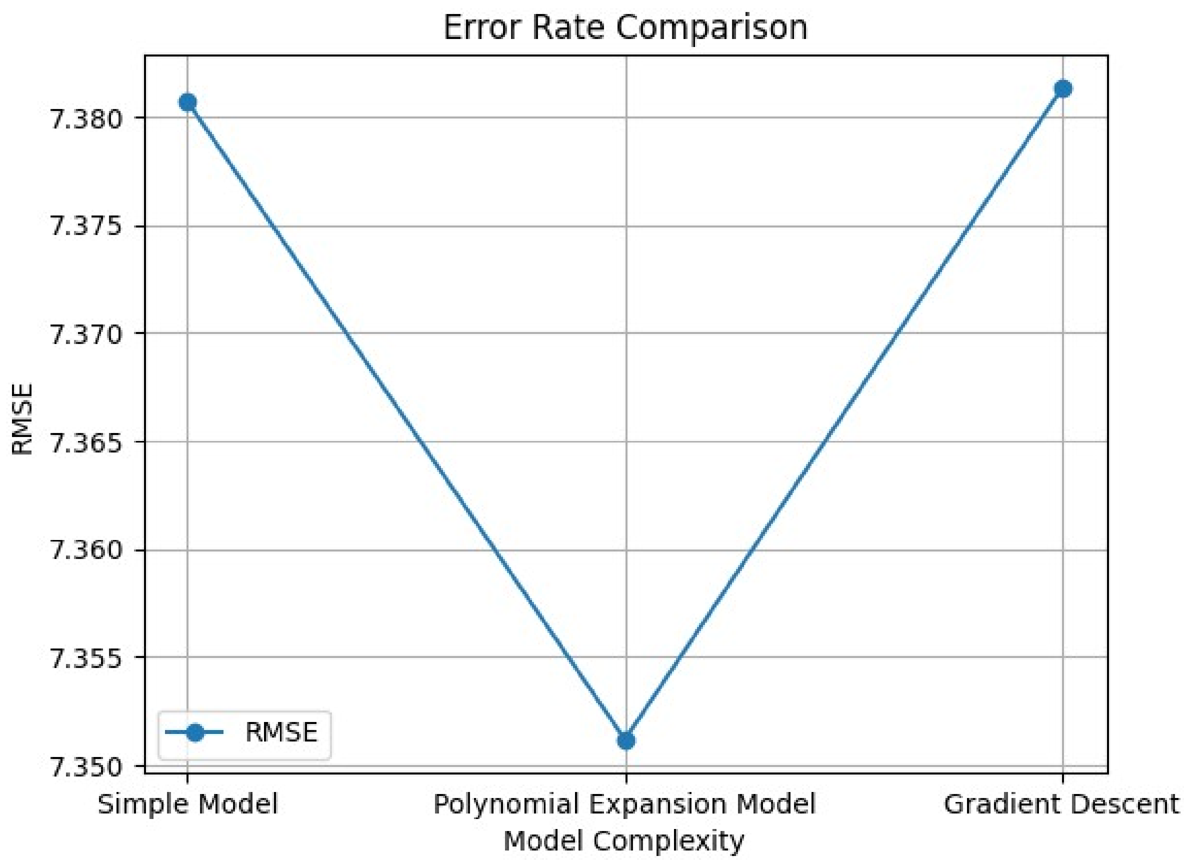
Compare Performance of OLS, Polynomial Expansion, and Gradient Descent

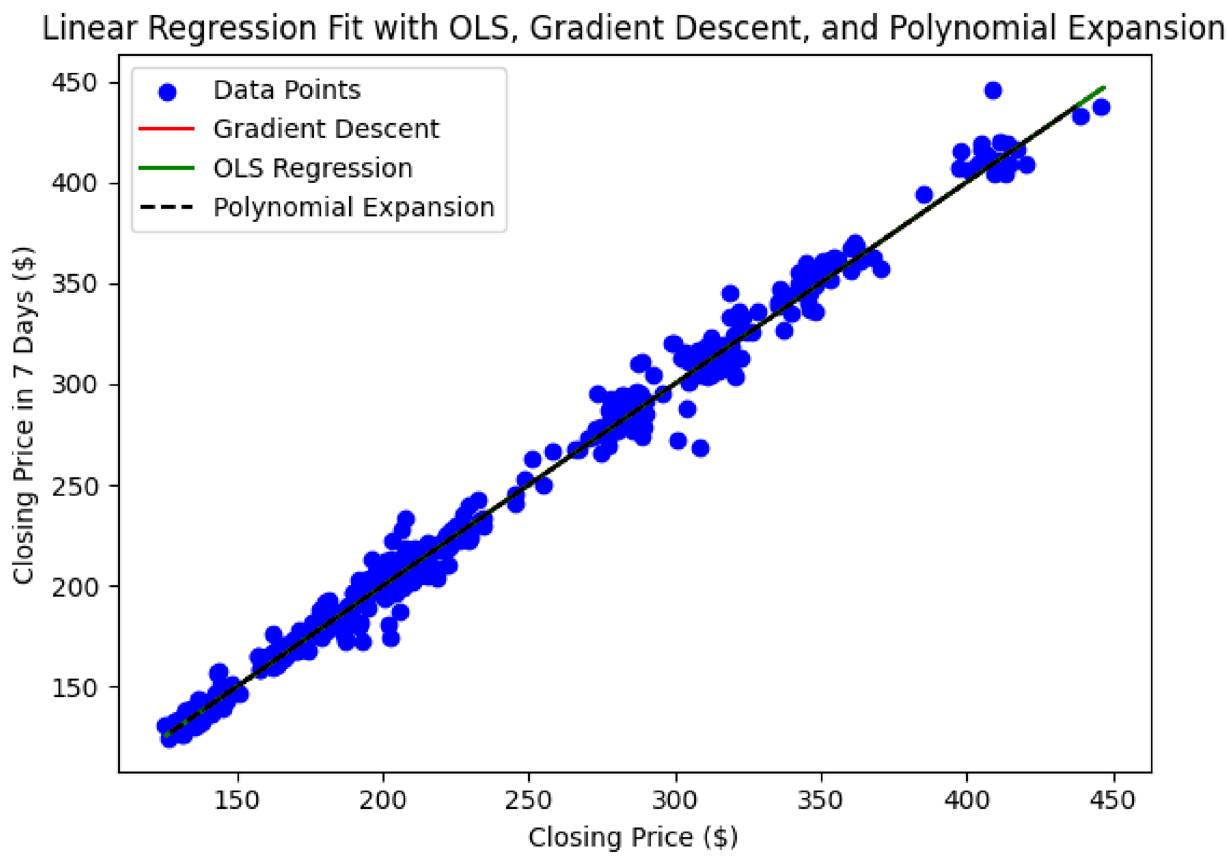
```
# OLS results
print(f"Linear Regression with OLS:\n  R2: {ols_r2}\n  RMSE: $
{ols_rmse}")
print(f"  MAE: ${ols_mae}\n  Bias: {ols_bias}\n  Variance:
{ols_variance}\n")

# Polynomial expansion
print(f"Regression with Polynomial Expansion:\n  R2: {poly_r2}\n
```

```
RMSE: ${poly_rmse}")
print(f"  MAE: ${poly_mae}\n  Bias: {poly_bias}\n  Variance:
{poly_variance}\n")

# Gradient Descent
print(f"Linear Regression with Gradient Descent:\n  R2: {gd_r2}\n
RMSE: ${gd_rmse}")
print(f"  MAE: ${gd_mae}\n  Bias: {gd_bias}\n  Variance:
{gd_variance}\n")

# Error Comparison Visualization
model_complexity = ['Simple Model', 'Polynomial Expansion Model',
'Gradient Descent']
mse_scores = [ols_rmse, poly_rmse, gd_rmse]

plt.plot(model_complexity, mse_scores, label='RMSE', marker='o')
plt.title('Error Rate Comparison')
plt.xlabel('Model Complexity')
plt.ylabel('RMSE')
plt.legend()
plt.grid(True)
plt.show()


# Plot the linear fit
x = np.array([0,10]).reshape(-1,1)
x_b = np.c_[np.ones((len(x), 1)), x]
plt.scatter(X_test, y_test, color='blue', label="Data Points")
plt.plot(gd_pred, gd_pred, color='red', label="Gradient Descent")
plt.plot(ols_pred, ols_pred, color='green', label="OLS Regression")
plt.plot(poly_pred,  poly_pred, "--", color='black', label="Polynomial
Expansion")
plt.title(f"Linear Regression Fit with OLS, Gradient Descent, and
Polynomial Expansion")
plt.xlabel("Closing Price ($)")
plt.ylabel("Closing Price in 7 Days ($)")
plt.legend()
plt.tight_layout()
plt.show()
```

```
Linear Regression with OLS:
  R2: 0.9911992251158583
  RMSE: $7.380728265514138
  MAE: $5.207673919675837
  Bias: 6189.905271899396
  Variance: 6061.442030563376

Regression with Polynomial Expansion:
  R2: 0.9912695991682327
  RMSE: $7.351159603326189
```

```
  MAE: $5.154774973438695
  Bias: 6189.933611006034
  Variance: 6046.653570281796

Linear Regression with Gradient Descent:
  R2: 0.991197717396704
  RMSE: $7.381360459210692
  MAE: $5.208097193763133
  Bias: 6189.911398943576
  Variance: 6046.654698325222
```




Linear Regression Fit with OLS, Gradient Descent, and Polynomial Expansion
Data Points
Gradient Descent
OLS Regression
Polynomial Expansion
Closing Price in 7 Days ($)
Closing Price ($)
150
200
250
300
350
400
450