

Technological Institute of the Philippines		Quezon City - Computer Engineering	
Course Code:	CPE 019		
Code Title:	Emerging Technologies in CpE 2 - Fundamentals of Computer Vision		
2nd Semester	AY 2023-2024		
<u>ACTIVITY NO.</u>		<b>PRELIM EXAMINATION</b>	
Name	Dela Cruz, Irish & Gullas, Rainer		
Section	CPE32S3		
Date Performed:	02/28/2024		
Date Submitted:	03/06/2024		
Instructor:	Engr. Roman M. Richard		

## OBJECTIVES

In this exam, you will show the application of the following algorithms:

### Part 1: Linear Regression

- Singular Linear Regression
- Multiple Linear Regression
- Polynomial Linear Regression

### Part 2: Logistic Regression

### Part 3: Decision Tress

### Part 4: Random Forest

## SCENARIO / BACKGROUND

In this exam, we will show each application of algorithms that listed above, including the datasets which contains the 2,516 entries.

The tasked is to analyzed historical stock market data for the past 10 years to identify patterns and build predictive models for stock price movements.

The goal is to develop models that can accurately predict wheter the stock price increase or decrease in the future based on various factors.

## REQUIRED RESOURCES

- PC / Laptop with Internet Access
- **Python libraries:** pandas, sklearn, IPython.display, numpy, and seaborn.
- **Additional application:** Graphviz
- **Datafiles:** Meta Stock Price Technical Indicator.csv

✓ The datasets contain the following variables:

Variable	Description
1. Date	Date for which data is recorded
2. Open	Opening price stock on trading day
3. High	Highest price at which stock traded on during trading day
4. Low	Lowest price at which stock traded on during trading day.
5. Close	Final price at which stock is valued for the day
6. Volume	No. of shares or contrast traded in the security/market during a given period.
7. RSI_7	<b>7-day Relative Strength Index</b> <ul style="list-style-type: none"> <li>• Measure the magnitude of recent prices changes to evaluate overbought/oversold conditions in price of stock</li> </ul>
8. RSI_14	<b>14-day Relative Strength Index</b> <ul style="list-style-type: none"> <li>• Calculated over 14 days</li> </ul>
9. CCI_7	<b>7-day Commodity Channel Index</b> <ul style="list-style-type: none"> <li>• Measure the difference between current price and historical average price</li> </ul>
10. CCI_14	<b>14-day Commodity Channel Index</b> <ul style="list-style-type: none"> <li>• 14 days for medium-term trends</li> </ul>

11. SMA_50	<b>50-day Simple Moving Average</b> <ul style="list-style-type: none"> <li>• Average the closing prices of stock over the past 50 days.</li> </ul>
12. EMA_50	<b>50-day Exponential Moving Average</b> <ul style="list-style-type: none"> <li>• Weight to recent prices</li> </ul>
13. SMA_100	<b>100-day Simple Moving Average</b> <ul style="list-style-type: none"> <li>• Average of closing prices of stock for over the past 100 days</li> </ul>
14. EMA_100	<b>100-day Exponential Moving Average</b> <ul style="list-style-type: none"> <li>• Responsive to recent price changes</li> </ul>
15. MACD	<b>Moving Average Convergence Divergence</b> <ul style="list-style-type: none"> <li>• Show the relationship between 2 moving averages of stock price</li> </ul>
16. Bollinger	Price envelope
17. TrueRange	Measure the volatility that consider the range between high, low, and close of stock
18. ATR_7	<b>7-day Average True Range</b> <ul style="list-style-type: none"> <li>• measures the market volatility by decomposing the entire range of stock for that period</li> </ul>
19. ATR_14	<b>14-day Average True Range</b>

## ✓ Import the libraries and data

```

1 #importing the important libraries that needed in the study
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sn
5 from sklearn.model_selection import train_test_split
6 from sklearn.linear_model import LinearRegression
7 from sklearn.metrics import mean_squared_error, r2_score
8 import numpy as np

```

```

1 from google.colab import drive
2 drive.mount('/content/Mydrive')

```

Mounted at /content/Mydrive

```

1 #load the Dataset and named as "stock"
2 stock = pd.read_csv("/content/Mydrive/MyDrive/Meta Stock Price Technical Indicators (10 Y

```

```
1 #select the first 20 rows of the data
2 stock.head(20)
```

	date	open	high	low	close	volume	rsi_7	rsi_14	
0	2014-01-02	54.830002	55.220001	54.189999	54.709999	43195500	51.917475	58.077822	-6
1	2014-01-03	55.020000	55.650002	54.529999	54.560001	38246200	50.604988	57.387622	-4
2	2014-01-06	54.419998	57.259998	54.049999	57.200001	68852600	67.483917	65.221525	4
3	2014-01-07	57.700001	58.549999	57.220001	57.919998	77207400	70.672584	67.003189	15
4	2014-01-08	57.599998	58.410000	57.230000	58.230000	56682400	72.049420	67.768804	10
5	2014-01-09	58.650002	58.959999	56.650002	57.220001	92253300	61.139242	62.667059	6
6	2014-01-10	57.130001	58.299999	57.060001	57.939999	42449500	65.485896	64.706800	5
7	2014-01-13	57.910000	58.250000	55.380001	55.910000	63010900	47.872555	55.499705	-3
8	2014-01-14	56.459999	57.779999	56.099998	57.740002	37503600	59.366740	60.900791	-1
9	2014-01-15	57.980000	58.570000	57.270000	57.599998	33663400	58.220894	60.297824	4
10	2014-01-16	57.259998	58.020000	56.830002	57.189999	34541800	54.619116	58.472055	-2
11	2014-01-17	57.299999	57.820000	56.070000	56.299999	40849200	47.220911	54.606761	-9
12	2014-01-21	56.599998	58.580002	56.500000	58.509998	48669200	62.094919	61.425729	8

```
1 #Calculating the summary statistics of data
2 stock.describe()
```

	open	high	low	close	volume	rsi_7	
<b>count</b>	2516.000000	2516.000000	2516.000000	2516.000000	2.516000e+03	2516.000000	2516
<b>mean</b>	178.035517	180.328621	175.826359	178.129122	2.617010e+07	54.339039	54
<b>std</b>	81.123720	82.200952	80.118216	81.166653	1.782460e+07	15.808831	11
<b>min</b>	54.020000	54.939999	51.849998	53.529999	5.467500e+06	14.083263	21
<b>25%</b>	115.787498	117.452497	114.007502	115.565003	1.563175e+07	43.513651	46
<b>50%</b>	170.114998	172.110001	168.224998	170.245002	2.106275e+07	55.357801	54
<b>75%</b>	220.297504	221.827499	216.492501	219.864994	3.022008e+07	66.148583	62
<b>max</b>	381.679993	384.329987	378.809998	382.179993	2.323166e+08	93.579562	86

```

1 #Provides the summary of Dataframe.
2 #Including the index, columnss, data type, and non-null values.
3 stock.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2516 entries, 0 to 2515
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   date                  2516 non-null  object
1   open                  2516 non-null  float64
2   high                  2516 non-null  float64
3   low                   2516 non-null  float64
4   close                 2516 non-null  float64
5   volume                2516 non-null  int64
6   rsi_7                 2516 non-null  float64
7   rsi_14                2516 non-null  float64
8   cci_7                 2516 non-null  float64
9   cci_14                2516 non-null  float64
10  sma_50                2516 non-null  float64
11  ema_50                2516 non-null  float64
12  sma_100               2516 non-null  float64
13  ema_100               2516 non-null  float64
14  macd                  2516 non-null  float64
15  bollinger             2516 non-null  float64
16  TrueRange             2516 non-null  float64
17  atr_7                 2516 non-null  float64
18  atr_14                2516 non-null  float64
19  next_day_close        2516 non-null  float64
dtypes: float64(18), int64(1), object(1)
memory usage: 393.2+ KB

```

```

1 #Checking if there's a missing or null values in Dataframe.
2 #isnull() and sum() calculates the total number of missing values for each column.
3 stock.isnull().sum()

```

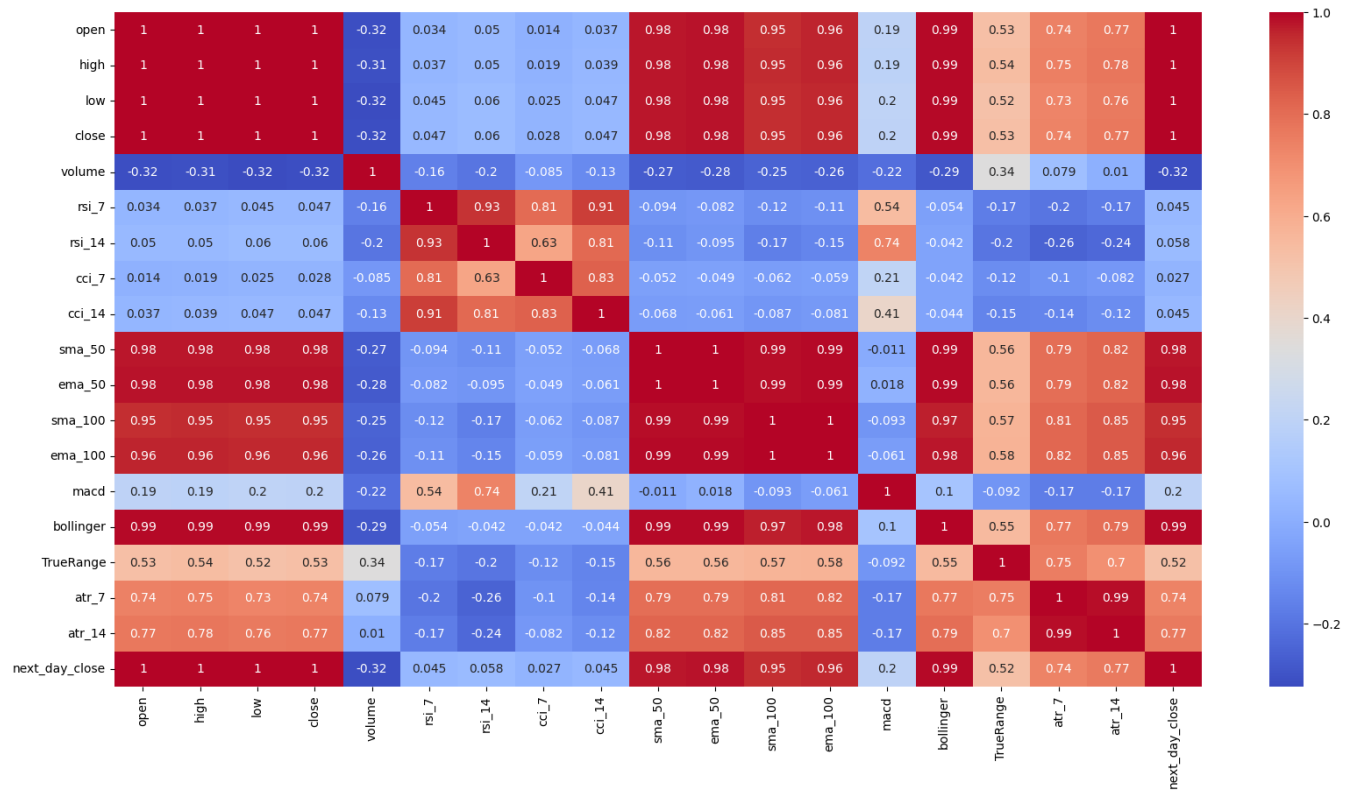
```
date          0
open          0
high          0
low           0
close         0
volume        0
rsi_7         0
rsi_14        0
cci_7         0
cci_14        0
sma_50        0
ema_50        0
sma_100       0
ema_100       0
macd          0
bollinger     0
TrueRange     0
atr_7         0
atr_14        0
next_day_close 0
dtype: int64
```

## ✓ Analysis

Once there's missing values the statistic analysis can lead to inaccurate results and predictions. That's why we check if there's a missing value above to fill the missing values to ensure that the analysis will be based on reliable information. Also, there's algorithms cannot handle missing values and produces errors.

```
1 #importing seaborn for better visualization with color palettes that looks professional
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 #annotation - used to display actual values by each cell
6 #color map - 'coolwarm' for better visualization
7 plt.figure(figsize=(20,10))
8 sns.heatmap(stock.corr(), annot= True, cmap='coolwarm')
9 plt.show()
```

```
<ipython-input-11-5c89026a36ff>:8: FutureWarning: The default value of numeric_only in [
sns.heatmap(stock.corr(), annot=True, cmap='coolwarm')
```



# Analysis

We used heatmap with coolwarm tone to easily visualized the correlation coefficient of the data that indicates the relationship between 2 or more variables if they are closely related to each other.

Here in the visual aids, when the diagonal indicate as 1 it signifies a perfect positive correlation. The independent variables (open, high, low, close) have a perfect positibe correlation with dependent variable (next\_day\_close) that can used to predict or explain the values with high certainty.

## Part 1: Linear Regression

In this part of exam, we will build and evaluate 3 types of Linear Regression: **Singular Regression, Multiple Regression, Polynomial Regression**. These models will be trained on labeled dataset containing technical indicator for stock prices over the past 10 years.

Our goal is to predict the classification of Meta Stock Price Movements from 2014 - 2023 based on indicators.

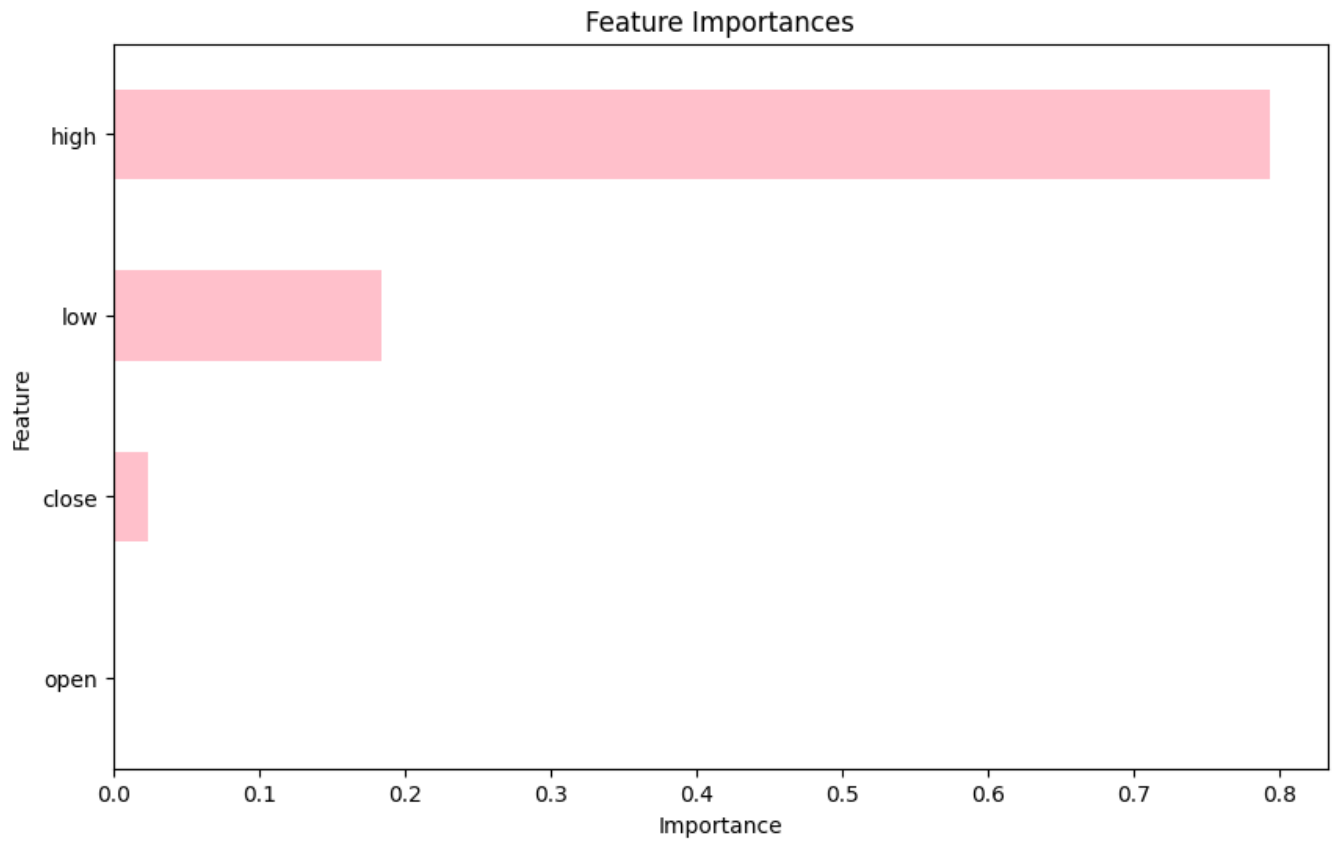
### ✓ Singular Linear Regression

Statistical method used to model relationship between dependent variable (target or response) and independent variables (predictors or features). It assumes if there's a linear relationship between the independent and dependent variables.

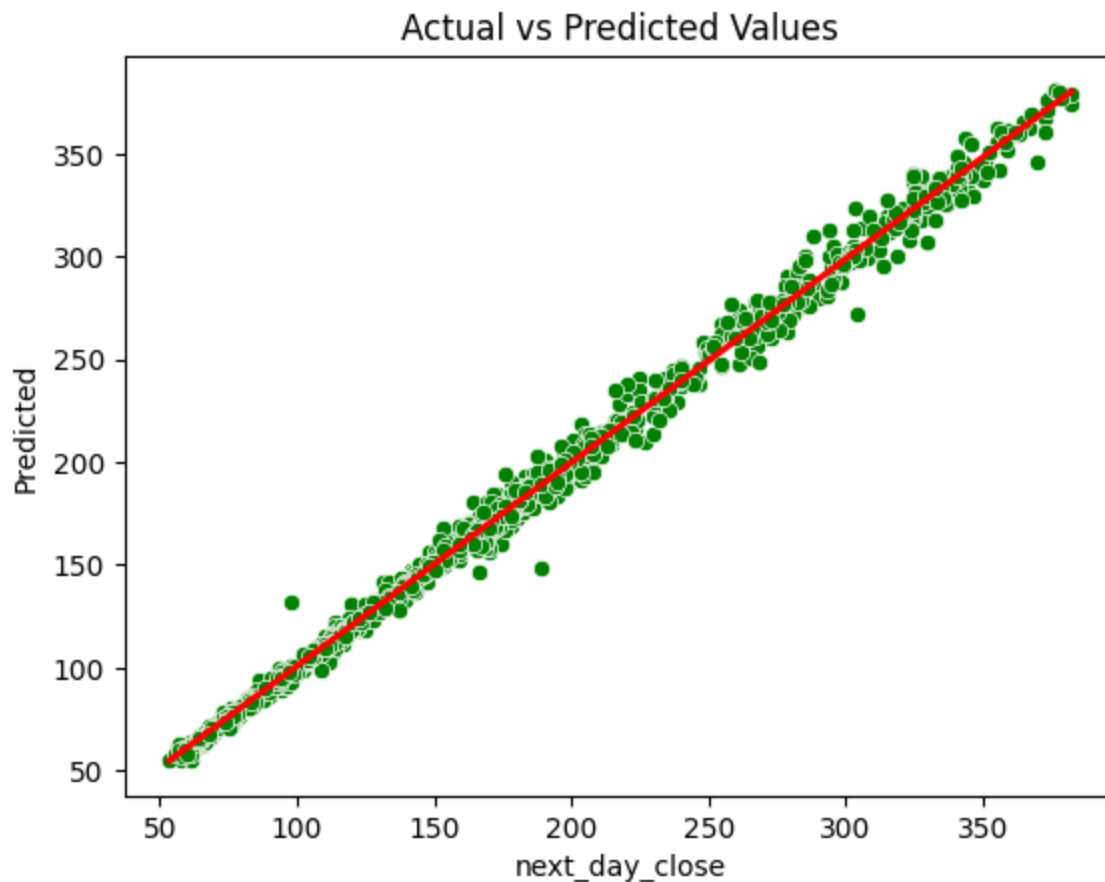


```
1 from sklearn.linear_model import LinearRegression
2 #Column name
3 feature_names = ['open','high', 'low', 'close', 'volume', 'rsi_7', 'rsi_14', 'cci_7', 'cc
4
5 # Create a Linear Regression model
6 model = LinearRegression()
7
8 # Fit the model to the data
9 model.fit(X_train, y_train)
10
11 # Get weights of the features
12 coefficients = model.coef_
13
14 print("Number of features:", len(feature_names))
15 print("Number of coefficients:", len(coefficients))
16
17 # Create horizontal bar plot
18 plt.figure(figsize=(10, 6))
19 ax = model_ranks.plot(kind='barh', color='pink')
20
21 # Set plot title and labels
22 plt.title('Feature Importances')
23 plt.xlabel('Importance')
24 plt.ylabel('Feature')
```

```
Number of features: 19  
Number of coefficients: 4  
Text(0, 0.5, 'Feature')
```



```
1 import seaborn as sns
2
3 # Define predictors and target variable
4 X = stock[['open']] # Independent variables
5 y = stock['next_day_close'] # Dependent variable
6
7 # Split the data into training and testing sets
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=23)
9
10 # Create a linear regression model
11 model = LinearRegression()
12
13 # Fit the model to the training data
14 model.fit(X_train, y_train)
15
16 # Make predictions on the test set
17 y_pred = model.predict(X_test)
18
19 # Plot actual vs predicted values
20 sns.scatterplot(x=y_test, y=y_pred, color='green')
21 plt.xlabel('Actual')
22 plt.ylabel('Predicted')
23 plt.title('Actual vs Predicted Values')
24
25 # Plot regression line
26 sns.regplot(x=y_test, y=y_pred, scatter=False, color='red')
27 plt.show()
```



## Analysis

We can visually assess the performance of linear regression model in predicting the 'next\_day\_closing' prices based on the opening prices. The points are closely align along diagonal line which indicates that the model's prediction closely match to actual values.

### ✓ Evaluate the model (Singular Linear Regression)

```
1 #Computation for MSE and R-squared score
2 mse = mean_squared_error(y_test, y_pred) #ave sqr diff. between actual and predicted valu
3 r_squared = r2_score(y_test, y_pred) #measure the proportion of variance in x and y
4
5 print("Mean Squared Error:", mse)
6 print("R-squared Score:", r_squared)
```

Mean Squared Error: 27.762230502330063

R-squared Score: 0.995662789441985

# Analysis

MSE is approximately 27.76 which is low value indicating that the model's prediction are close to actual values on average, which suggest a good accuracy.

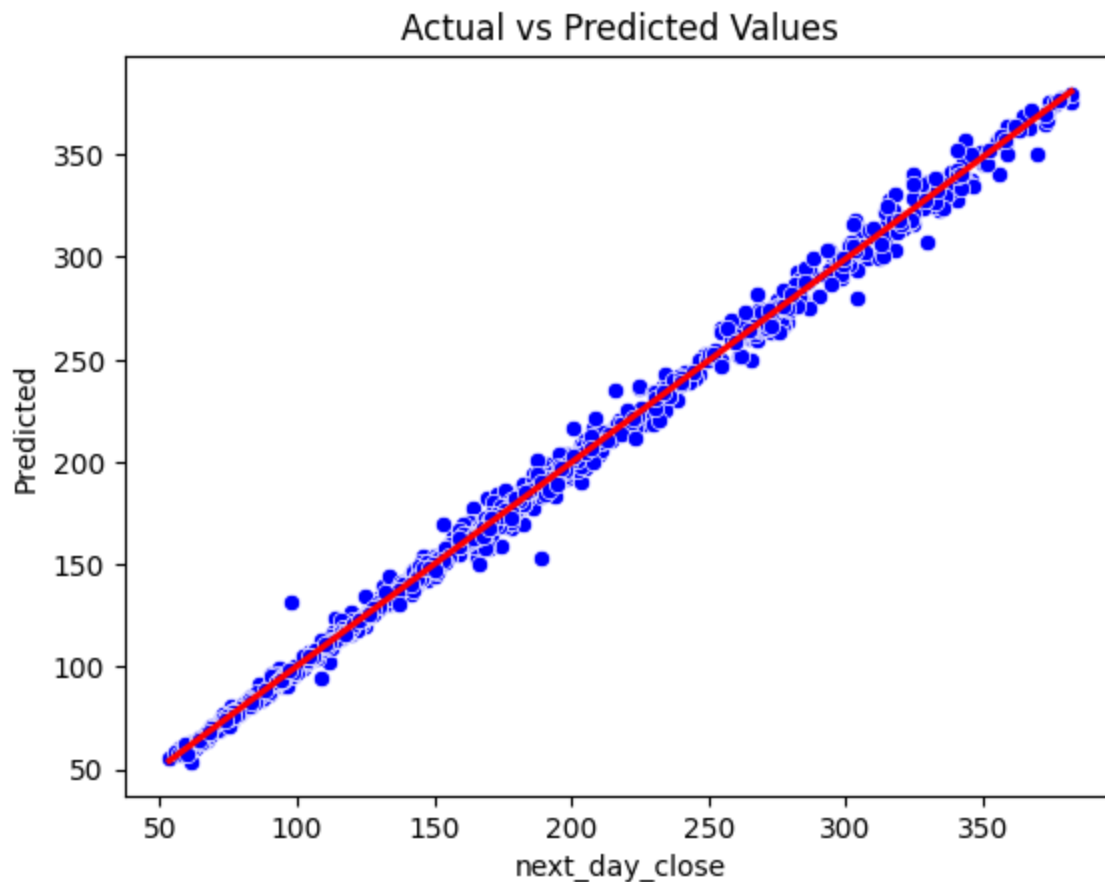
The ranges of r2-score from 0 to 1, the r-squared score of the data was 0.996 closes to 1. Which suggest that 99.6% of independent variables in model are effective in explaining the variation in dependent variable.

**Note:** Closes to 1 means a perfect fit for the variability of response data around it mean

## ✓ Multi-linear Regression

Statistical technique used to analyzed relationship between two or more independent variables and dependent variables by fitting a linear equation to observed data.

```
1 import seaborn as sns
2
3 # Define predictors and target variable
4 X = stock[['open', 'high', 'low', 'close']] # Independent variables
5 y = stock['next_day_close'] # Dependent variable
6
7 # Split the data into training and testing sets
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=23)
9
10 # Create a linear regression model
11 model = LinearRegression()
12
13 # Fit the model to the training data
14 model.fit(X_train, y_train)
15
16 # Make predictions on the test set
17 y_pred = model.predict(X_test)
18
19 # Plot actual vs predicted values
20 sns.scatterplot(x=y_test, y=y_pred, color='blue')
21 plt.xlabel('Actual')
22 plt.ylabel('Predicted')
23 plt.title('Actual vs Predicted Values')
24
25 # Plot regression line
26 sns.regplot(x=y_test, y=y_pred, scatter=False, color='red')
27 plt.show()
```



## Analysis

The points closely follow the red line, which suggests that the model's prediction are accurate and have a strong linear relationship between the independent variable (open, high, low, close) and dependent variable (next\_day\_close)

Note: If the points from tigh cluster around the regression line, it indicates that the model's prediction closely match the actual values and model capturing the patterns in data effectively.

## ✓ Evaluate the model (Multi-linear Regression)

```
1 # Calculation of MSE and R2-score
2 mse = mean_squared_error(y_test, y_pred)
3 r_squared = r2_score(y_test, y_pred)
4 print("Mean Squared Error:", mse)
5 print("R-squared Score:", r_squared)
```

Mean Squared Error: 19.765984763393266  
R-squared Score: 0.9969120190901752

# Analysis

The MSE is approximately 19.77 - low value which means the predicted values are close to the actual values. While the  $r^2$ -score is equivalent to 0.997 or 99.7% which explains the model effectively captures the relationship of independent and dependent variable as evidenced by the high R-squared and low MSE.

## ✓ Polynomial Regression

Regression analysis used to model the relationship between a dependent variable and one or more independent variable by fitting the polynomial function to observed data.

Note: Polynomial Regression fits a curve.

```

1 # Plotpolly - visualization of polynomial regression models.
2 # model - callable variable that can pass to independent and get vack the predicted value
3 # independent - used in regression / dependent - observed values
4 # name - label for x-axis (independent variable)
5 def PlotPolly(model, independent_variable, dependent_variabbble, Name):
6     # Genearate 100 space between 15 & 55, used to create smooth curve
7     x_new = np.linspace(15, 55, 100)
8     # predict the dependent variable based on model and x variable
9     y_new = model(x_new)
10
11     # Plots the original data points as dots and polynomial curve
12     plt.plot(independent_variable, dependent_variabbble, '.', x_new, y_new, '-')
13     #Title Plot
14     plt.title('Polynomial Fit with Matplotlib for Stock Meta ~ Length')
15     # Current axes instance
16     ax = plt.gca()
17     # Set the background color
18     ax.set_facecolor((0.898, 0.898, 0.898))
19     # Current figure instance
20     fig = plt.gcf()
21     # label for x-axis as name
22     plt.xlabel(Name)
23     # plot y-axis as next_day_close
24     plt.ylabel('next_day_close')
25     # display plot
26     plt.show()

```

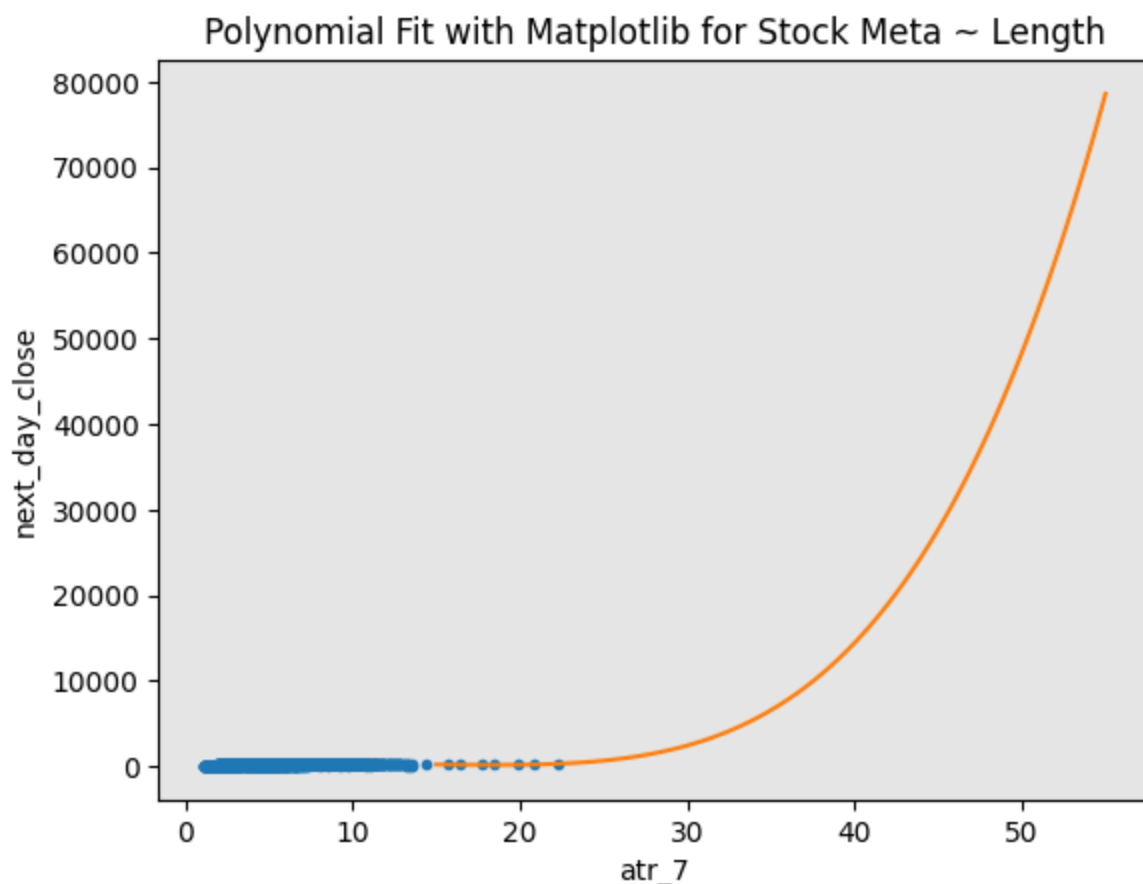
## ✓ Analysis

Designed that provide a convenient way to visualized the polynomial fit for regression model along side the original data points. It helps to capture the relationship between independent and dependent variable

```

1 # Two important variables that set as example for this model
2 x = stock['atr_7']
3 y = stock['next_day_close']
4
5 # Inside the polynomial model 'p' used to predict values of dependent based on independent
6 f = np.polyfit(x, y, 4)
7 p = np.poly1d(f)
8
9 # Plot the polynomial fit
10 PlotPolly(p, x, y, 'atr_7')

```



## ✓ Analysis



The graph show the relationship between 2 variables: 'atr\_7' and 'next\_day\_close'. The dots represent as actual stock prices. The blue lines represents the polynomial fit that capture the general trend of stock as 'atr\_7' changes.

This suggest that the volatility of stock increases ('atr\_7' value), the prices tend to decrease or show movement.

```
1 y_pred
```

```
array([ 99.76628779, 184.8569084 , 330.64921011, ..., 136.13264497,
       167.3435064 , 281.64088622])
```

## ✓ Evaluation of Polynomial Linear Regression

```
1 # Calculate predicted values using the polynomial regression model
2 y_pred = p(x)
3
4 # Calculate R-squared score
5 r_squared = r2_score(y, y_pred)
6 print('R-squared score:', r_squared)
7
8 # Calculate Mean Squared Error (MSE)
9 mse = mean_squared_error(y, y_pred)
10 print('Mean Squared Error:', mse)
```

```
R-squared score: 0.6170762448390501
Mean Squared Error: 2523.689159316617
```

## Analysis

The MSE was equivalent to 2523.689, indicate the squared difference between the actual and predicted stock prices. The higher MSE suggest a larger predictions error, so there's another factor aside from 'atr\_7' that affecting the stock prices.

r2-score of 0.617 or 61.7% of the variation in dependent variable (next\_day\_close) is explained by indepdent variable ('atr\_7'), which suggest that the model doesn't not explain much of variability in the data.

## Part 2: Logistic Regression

Statistical method used for binary classification task, where target variable has 2 possible outcomes. It used when dependent variable is categorical.

Note: Linear Regression - predicts continuous values, while Logistic Regression - is a binary classification algorithm, it requires the target variable to be categorical with two classes.

## ✓ Ridge Regression instead of Logistic Regression

(continuous values are not suitable for logistic regression)

variant of linear regression that used when the data suffers from multicollinearity (high correlation among predictor variables) or when overfitting in the model. It adds penalty term to ordinary least objective function, which helps to reduce the complexity of model and prevent overfitting.

```
1 from sklearn.linear_model import Ridge
2 from sklearn.metrics import mean_squared_error, r2_score
3 from sklearn.model_selection import train_test_split
4
5 # Splitting the data into training and testing sets
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
7
8 # Creating and fitting the Ridge regression model
9 ridge_model = Ridge(alpha=1.0) # You can adjust the regularization strength by changing
10 ridge_model.fit(X_train, y_train)
11
12 # Making predictions on the test set
13 predictions = ridge_model.predict(X_test)
14
15 # Calculating Mean Squared Error
16 mse = mean_squared_error(y_test, predictions)
17
18 # Calculating R-squared score
19 r_squared = r2_score(y_test, predictions)
20
21 print("Mean Squared Error:", mse)
22 print('R-squared score:', r_squared)
23
```

```
Mean Squared Error: 1.7115637211977852
R-squared score: 0.9997318045506529
```

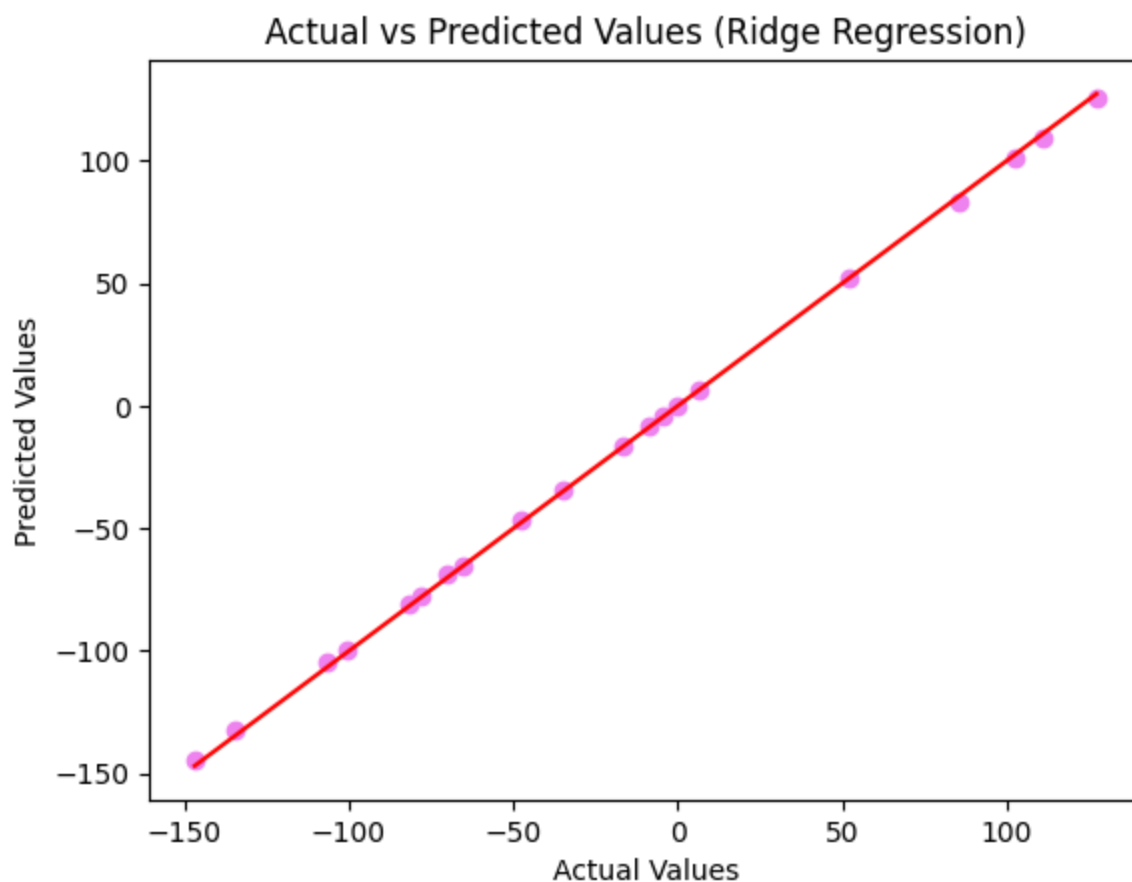
## ✓ Analysis

Our target variables are continuous that's why we used ridge regression as a substitute for the logistic.

The MSE value of approximately 1.71 indicates that, on average, the squared difference between the actual and predicted target values is 1.71 closer to actual value

R-squared score of approximately 0.9997 suggests that the Ridge regression model explains about 99.97% perfectly explain the variance in the target variable.

```
1 # Plotting the actual vs predicted values
2 plt.scatter(y_test, predictions, color='violet')
3 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='
4 plt.xlabel('Actual Values')
5 plt.ylabel('Predicted Values')
6 plt.title('Actual vs Predicted Values (Ridge Regression)')
7 plt.show()
```



## Analysis

The visual aid seems that the Actual Value was 50 and 100. Then the Predicted Values are: 100, 50, -50, -100, -150, -150. The model's predictions seem to be lower than actual in some instances and higher to others, However the Actual value of 100 was accurate. Since the use of Logistic

Regression is for the catehorical values and our dataset uses continous values we opted to use Ridge Regression

## Part 3: Decision Trees

A flowchart structure used for classification or regression, where each internal node represents a test on an attribute (feature), branches represent the outcome of the test, and leaf nodes represent the class label (classification) or numerical value (regression).

Decision tree or if-else statements used to determine the best feature and decision rule for splitting data, minimizing squared error at each node.

### ✓ Installing graphviz for displaying the image

```
1 !pip install graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.1)
```



### ✓ Importing Libraries

```
1 from sklearn.tree import DecisionTreeRegressor, export_graphviz
2 from IPython.display import display, Image
3 import graphviz
```

```
1 #data["condition"].fillna(data["condition"].mean(), inplace = True)
```

```
1 # Dependent variable
2 y_target = stock["next_day_close"].values
```

```
1 # 4 important variables
2 columns = ["open", "close", "low", "high"]
3 X_input = stock[list(columns)].values
```

```

1 # Creating Decision Tree Regression
2 regressor = DecisionTreeRegressor(max_depth=3) # depth 3 meand that tree will make split
3
4 #used to make prediction on new data
5 regressor.fit(X_input, y_target)

```

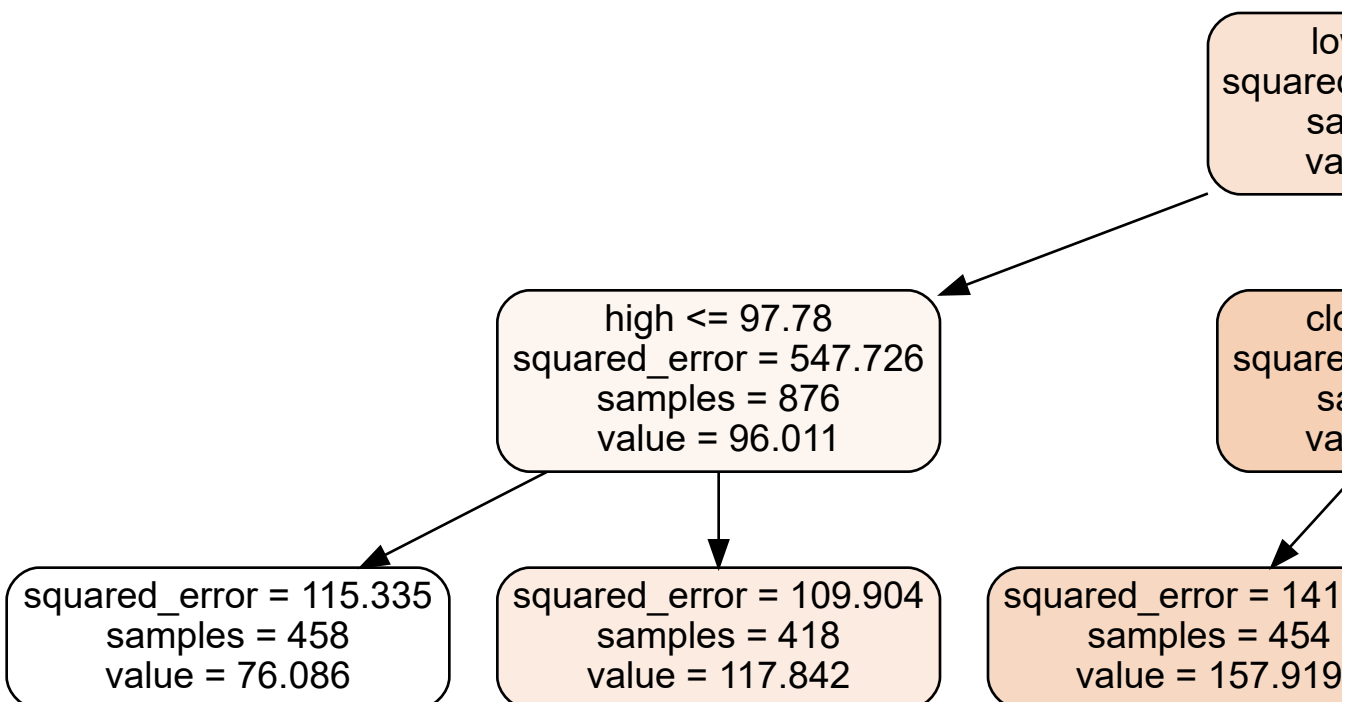
▼ DecisionTreeRegressor

```
DecisionTreeRegressor(max_depth=3)
```

```

1 # exporting the train decision tree model into textual representation in DOT format which
2 dot_data = export_graphviz(regressor, out_file=None, feature_names=columns, filled=True,
3 # represent the decision tree graph from DOT data
4 graph = graphviz.Source(dot_data)
5
6 # set the length and width of the graph
7 graph.format = 'png' # or 'pdf', 'svg', etc.
8 graph.render("decision_tree_graph", format='png', cleanup=True, view=True, directory=None)
9
10 # Displaying the decision tree
11 display(graph)

```



## Analysis

The squared errors for specific condition and no. of samples indicates as data points that meet that condition.

For the first line the condition is met ( $\text{high} \leq 97.78$ ), the squared error is 115.335 calculated based on 458 samples.

For the second line the condition is met ( $\text{high} \leq 97.78$ ) is true, used in calculation 76.086

The squared errors and no of sample are calculated in different consditions for ex. " $\text{high} \leq 97.78$ ", " $\text{low} \leq 133.705$ ", and " $\text{close} \leq 173.83$ " represent different rules or threshold for splitting data based on values of features.

The true condition lead to one branch of decision tree as true, false lead to another branch as false.

## ✓ Evaluation for Decision Tree

```
1 # Predict the target variable using the trained model
2 y_pred = regressor.predict(X_input)
3
4 # Compute Mean Squared Error (MSE)
5 mse = mean_squared_error(y_target, y_pred)
6 print("Mean Squared Error:", mse)
7
8 # Compute R-squared (R2) score
9 r_squared = r2_score(y_target, y_pred)
10 print("R-squared:", r_squared)
```

```
Mean Squared Error: 139.11747590258972
R-squared: 0.9788914628869912
```

## Analysis

Since we used Decision Tree Regressor, the evaluation metrics will be MSE and R2 score.

Results suggest that the regression model has performed well, with a relatively low MSE as 139.12 and a high R2 score as 0.979 Or 97.9%. It indicates that the model's predictions are close to the actual values, and the model explains a large portion of the variability in the target variable.

Note:

- $y_{\text{test}}$  or  $y_{\text{pred}}$  contained continuous values instead of discrete class we can't use confusion matrix for that.

- Decision Tree Classifier used for classification problems where target variable is categorical "class labels".

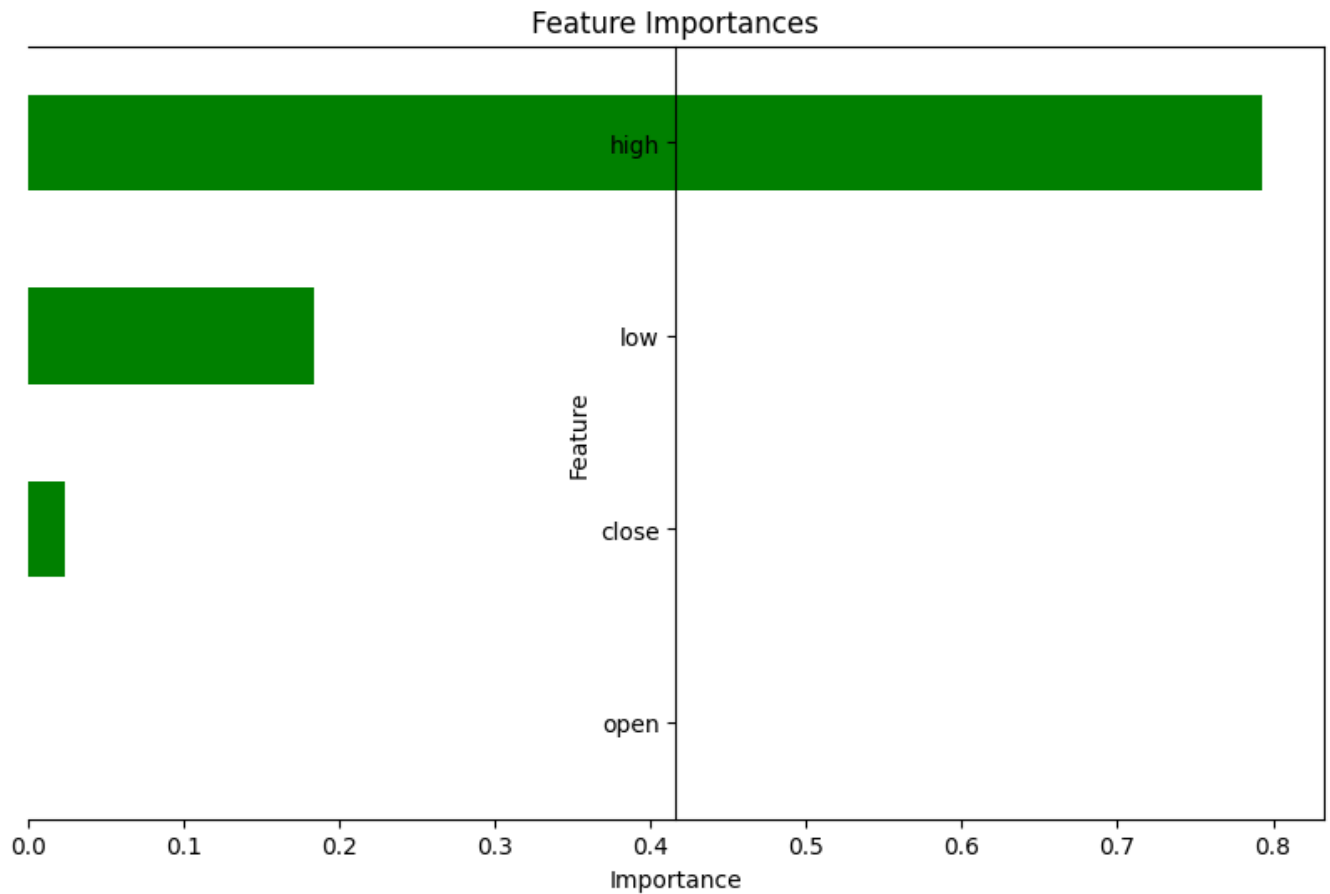
## ✓ Part 4: Random Forest

An ensemble learning method that combines multiple decision trees to improve prediction accuracy and prevent overfitting.

```
1 # Contain the values of all columns in data except 'date'
2 X = stock.drop(["date"],axis=1).values

1 # Removing rows with missing values
2 x = stock.dropna(inplace = True)

1 # Get feature importances from the trained random forest regressor model
2 feature_importances = regressor.feature_importances_
3
4 # Get the names of the features
5 feature_names = ['open','high', 'low', 'close']
6
7 # Create a Series with feature importances and their corresponding column names
8 model_ranks = pd.Series(feature_importances, index=feature_names, name='Importance')
9
10 # Sort feature importances
11 model_ranks = model_ranks.sort_values(ascending=True)
12
13 # Create horizontal bar plot
14 plt.figure(figsize=(10, 6))
15 ax = model_ranks.plot(kind='barh', color='green')
16
17 # Set plot title and labels
18 plt.title('Feature Importances')
19 plt.xlabel('Importance')
20 plt.ylabel('Feature')
21
22 # Move x-axis to the center
23 ax.spines['left'].set_position('center')
24
25 # Show plot
26 plt.show()
```



## ✓ Analysis:

It's very essential to know what's the important variable in the model. So it's easy to identify what features contribute the most model's predictions. Which variables have little to no impact on prediction.

We used to display the importance variable in order for us to provide insight in the relationship between features and target variables that can useful for feature selection.




```

1 from sklearn.ensemble import RandomForestRegressor
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import mean_squared_error
4
5 X = stock[['open', 'high', 'low', 'close']] # Independent variables
6 y = stock['next_day_close'] # Dependent variable
7
8 # Split the data into training and testing sets
9 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
10
11 # Calling the module
12 # 100 = parameters/ reproducibility
13 rf = RandomForestRegressor(n_estimators = 100, random_state=42)
14 rf.fit(X_train, y_train)
15
16 # Generates the predictions in given data
17 y_train_pred = rf.predict(X_train)
18 y_test_pred = rf.predict(X_test)

1 import pandas as pd
2
3 # Contains predicted values of training set in y_train_pred
4 y_train_pred = pd.DataFrame(y_train_pred, columns = ['yPredict'])
5 y_train_pred

```

	yPredict	
0	337.288201	
1	180.581400	
2	112.069499	
3	231.207198	
4	167.067301	
...	...	
1756	174.646600	
1757	276.510301	
1758	127.845697	
1759	149.247500	
1760	231.628995	

1761 rows × 1 columns

## ✓ Analysis

it's important to know the `y_train_pred` for evaluating the performance model on training data. By comparing the predicted values with actual values.

This information can be used to fine-tune the model to prevent overfitting.

```
1 # Generates prediction in x_test that stored to y_test_pred  
2 y_test_pred
```

```

95.28969966, 230.72679812, 75.0825004 , 351.86390178,
183.40819942, 54.88260014, 203.7600036 , 362.19029669,
128.97620116, 177.09130029, 179.41020004, 92.44370036,
131.59489987, 90.18330318, 77.89450079, 170.5622977 ,
175.10009974, 162.18780018, 82.10340055, 179.02009968,
305.39100172, 194.7873997 , 73.43420008, 61.36190078,
297.84559922, 280.86050768, 342.84269614, 237.374901 ,
54.78489997, 90.87400161, 377.62710121, 62.06750023,
112.18270099, 115.36970002, 97.17060102, 175.97570163,
209.35120194, 271.44260035, 129.27229922, 350.43130111,
80.04610022, 213.31830109, 185.54589911, 324.65970291,
173.30619929, 202.92680005, 96.91510144, 284.94369237,
183.90159665, 288.4127998 , 171.91200138, 102.79589963,
210.26810053, 262.34469611, 207.00280254, 304.10880012,
347.50129563, 297.75169572, 169.6184036 , 227.03810025,
68.78269847, 158.04669969, 183.1957991 , 142.24650117,

```

```
1 # Size of data and numbers of features.
```

```
2 stock.shape
```

```
(2516, 20)
```

```
1 plt.figure(figsize=(10, 5))
```

```
2 plt.scatter(y_test, y_test_pred, color='red', label='Comparison of Prediction Between Act
```

```
3 plt.legend()
```

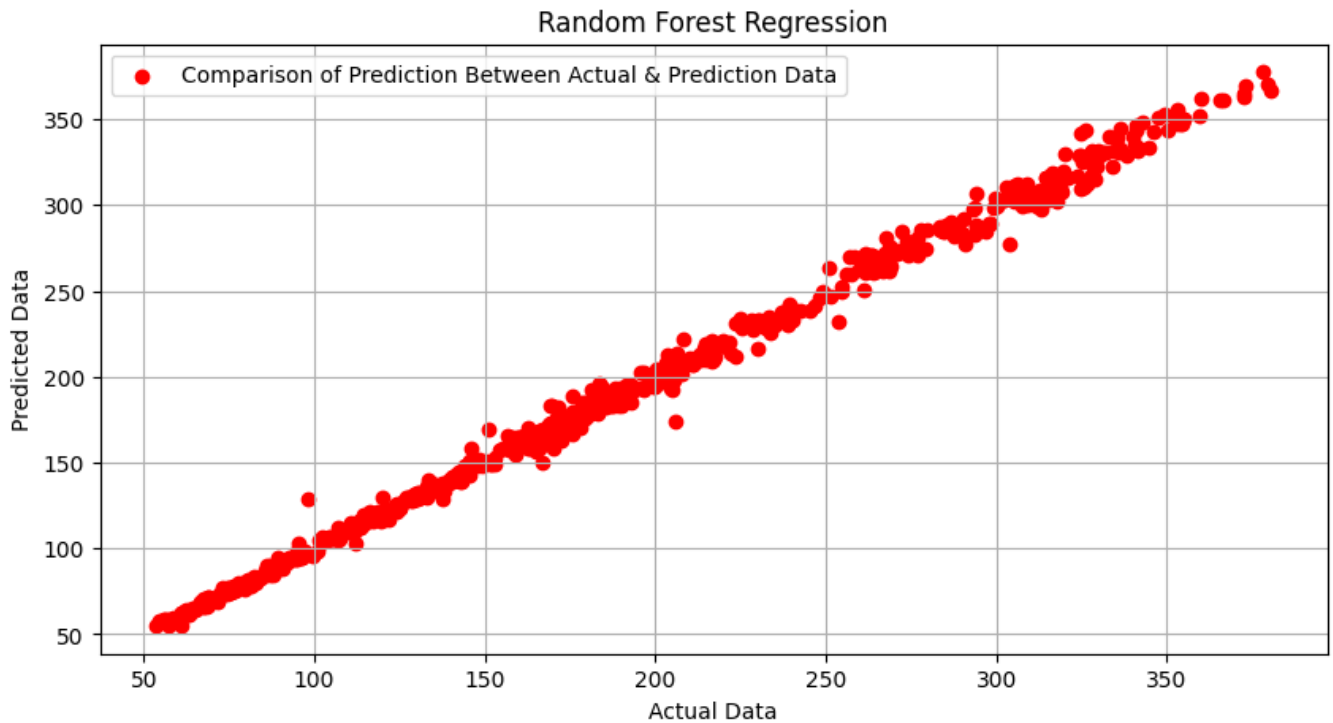
```
4 plt.grid()
```

```
5 plt.title('Random Forest Regression')
```

```
6 plt.xlabel('Actual Data')
```

```
7 plt.ylabel('Predicted Data')
```

```
8 plt.show()
```



## Analysis

The visual aid compares the actual and predicted data for random forest regression. It suggests that the model was able to predict values for the test set with accuracy.

### ✓ Evaluation for Random Forest (Actual vs. Predicted)

```

1 from sklearn.metrics import mean_squared_error, r2_score
2
3 # Make predictions on the training and testing data
4 y_train_pred = model.predict(X_train)
5 y_test_pred = model.predict(X_test)
6
7 # Calculate MSE for training and testing data
8 mse_train = mean_squared_error(y_train, y_train_pred)
9 mse_test = mean_squared_error(y_test, y_test_pred)
10
11 # Calculate R-squared for training and testing data
12 r2_train = r2_score(y_train, y_train_pred)

```

## ✓ Analysis

MSE values for both training and testing datasets are relatively low, with test MSE (19.77) being slightly low than train MSE (22.96). Indicates that the model generalizes well to unseen data.

Both train and test of R2 score are equal to 1, indicates that the model explains nearly variability in data.

It suggest that the model captures the patterns in data effectively without memorizing it, allowing to make accurate predictions on new data samples (unseen data that hasn't been trained on).

```

1 from sklearn.datasets import make_regression
2 from sklearn.ensemble import RandomForestRegressor
3 from sklearn.tree import export_graphviz
4 import graphviz

1 # Generating a sample dataset
2 X, y = make_regression(n_samples=100, n_features=4, noise=0.1, random_state=42)
3
4 # Creating a random forest regressor model
5 regressor = RandomForestRegressor(n_estimators=100, random_state=42)
6
7 # Fitting the model to the data
8 regressor.fit(X, y)
9
10 # Visualizing the decision trees in the random forest
11 dot_data = export_graphviz(regressor.estimators_[0],
12                             out_file=None,
13                             feature_names=['open', 'high', 'low', 'close'])

```