Bhuvi Ghosh
60009210191

Experiment 5

Aim: Predicting Stock Prices with Linear Regression using Quandl dataset.

Predicting Stock Prices with Linear Regression

Theory: Machine learning (ML) is a technology that gives the systems the ability to learn on its own through real-world interactions and generalizing from examples without being explicitly programmed as in the case of rule-based programming. Machine Learning can play a key role in a wide range of critical applications. In machine learning, Linear Regression (LR) is a basic technique by which a linear trend can be obtained.

Simple Linear Regression :

Logistic Regression (LR)applications are used in banking, corporate finance, investments and other areas. I will briefly touch on simple linear regression in this post, but I do have an article specifically about simple linear regression using Python that can be found here and it may be a bit more detailed and helpful.

Linear regression can be used to find a relationship between two or more variables of interest and allows us to make predictions once these relationships are found. In simple linear regression, there are only two variables: one dependent variable and one independent variable.

Simple linear regression will provide a line of best fit, or the regression line. This regression line can be written as the following formula:

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

y — The independent variable α — The
constant/y-intercept β — The beta coefficient
(slope)
ε — The error term/residuals

Challenge:

Write a Python script that uses linear regression to predict the price of a stock for a given company of your choice.

Predicting Stock Prices with Linear Regression

Use Stock Price history data from the Quadl API and apply a regression analysis method to accurately predict stock prices over time.

Import Libraries

```
import numpy as np
import pandas as pd import quandl
import datetime

%matplotlib inline import matplotlib.pyplot
as plt import seaborn as sns
```

plt.style.use('seaborn-darkgrid')
plt.rc('figure', figsize=(16,10))
plt.rc('lines', markersize=4)

Configure Quandl

 # Import API key from file import API_config

 # Quandl API Auth
quandl.ApiConfig.api_key = API_config.API_KEY

Get the Data
 # Set start and end date for stock prices start_date =
datetime.date(2009, 3,8) end_date =
datetime.date.today() # Load data from Quandl
data = quandl.get('FSE/SAP_X', start_date=start_date, end_date=end_date)
# Save data to CSV file data.to_csv('data/sap_stock.csv')

data.head()

 # Check data types in columns
data.info()

 # Get descriptive statistics summary of data set data.describe()

 # Display features in data set data.columns

Select Subset with relevant features
Use the daily closing price **Close** as the value to predict, hence discard the other features.

* 'Close' column has numerical data type
* The 'Date' is the index column and contains datetime values

 # Create a new DataFrame with only closing price and date df =
pd.DataFrame(data, columns=['Close'])

 # Reset index column so that we have integers to represent time for later analysis df =
df.reset_index()

df.head()

 # Check data types in columns df.info()

 # Check for missing values in the columns
df.isna().values.any()

Explore the Data

After looking at the price movement over time by simply plotting the *Closing price* vs *Time*, it shows that the price continously increases over time and estimate trend could be linear.

 # Import matplotlib package for date plots import matplotlib.dates as mdates

years = mdates.YearLocator() # Get every year
yearsFmt = mdates.DateFormatter('%Y') # Set year format

 # Create subplots to plot graph and control axes fig, ax = plt.subplots()
ax.plot(df['Date'], df['Close'])

 # Format the ticks ax.xaxis.set_major_locator(years)
ax.xaxis.set_major_formatter(yearsFmt)

 # Set figure title
plt.title('Close Stock Price History [2009 - 2019]', fontsize=16)
 # Set x label
plt.xlabel('Date', fontsize=14)
 # Set y label
plt.ylabel('Closing Stock Price in $', fontsize=14)

 # Rotate and align the x labels fig.autofmt_xdate()

 # Show plot
plt.show()

Linear Regression

This data contains only one **independent variable (X) ** which represents the *date* and the **dependent variable (Y) ** we are trying to predict is the *Stock Price*. To fit a line to the data points, which then represents an estimated relationship between X and Y, use a **Simple Linear Regression**.

The best fit line can be described with
Y = \beta_0 + \beta_1 X

where
* Y is the predicted value of the dependent variable
* \beta_0 is the y-intercept
* \beta_1 is the slope
* X is the value of the independent variable

The goal is to find such coefficients \beta_0 and \beta_1 that the **Sum of Squared Errors**, which represents the difference between each point in the dataset with its corresponding predicted value outputted by the model, is minimal.

Training a Linear Regression Model
Train Test Split

 # Import package for splitting data set
from sklearn.model_selection import train_test_split

 # Split data into train and test set: 80% / 20% train, test =
train_test_split(df, test_size=0.20)

Create and Train the Model

 # Import package for linear model
from sklearn.linear_model import LinearRegression

 # Reshape index column to 2D array for .fit() method
X_train = np.array(train.index).reshape(-1, 1) y_train =
train['Close']

 # Create LinearRegression Object model =
LinearRegression()
 # Fit linear model using the train data set model.fit(X_train, y_train)

Model Evaluation

 # The coefficient
print('Slope: ', np.asscalar(np.squeeze(model.coef_)))
 # The Intercept
print('Intercept: ', model.intercept_)

Interpreting the coefficients:

* The **slope** coefficient tells us that with a 1 unit increase in **date** the **closing price** increases by 0.0276
* The **intercept** coefficient is the price at wich the **closing price** measurement started, the stock price value at date zero

 # Train set graph plt.figure(1,
figsize=(16,10))
plt.title('Linear Regression | Price vs Time')
plt.scatter(X_train, y_train, edgecolor='w', label='Actual Price') plt.plot(X_train,
model.predict(X_train), color='r', label='Predicted Price') plt.xlabel('Integer Date')
plt.ylabel('Stock Price') plt.legend()
plt.show()

Prediction from our Model

 # Create test arrays
X_test = np.array(test.index).reshape(-1, 1)
y_test = test['Close']

 # Generate array with predicted values y_pred =
model.predict(X_test)

Regression Evaluation

Compare the predicted values with the actual value on random sample from data set.

 # Get number of rows in data set for random sample df.shape

 # Generate 25 random numbers
randints = np.random.randint(2550, size=25)

 # Select row numbers == random numbers df_sample =
df[df.index.isin(randints)]

df_sample.head()

 # Create subplots to plot graph and control axes fig, ax =
plt.subplots()
df_sample.plot(x='Date', y=['Close', 'Prediction'], kind='bar', ax=ax)

 # Set figure title
plt.title('Comparison Predicted vs Actual Price in Sample data selection', fontsize=16)

 # Set x label
plt.xlabel('Date', fontsize=14)

 # Set y label
plt.ylabel('Stock Price in $', fontsize=14)

 # Show plot
plt.show()

Observe larger variations between predicted and actual values in the random sample. Also observe the model performance over the whole test data set.

 # Plot fitted line, y test plt.figure(1,
figsize=(16,10))
plt.title('Linear Regression | Price vs Time')
plt.plot(X_test, model.predict(X_test), color='r', label='Predicted Price') plt.scatter(X_test, y_test,
edgecolor='w', label='Actual Price')

plt.xlabel('Integer Date')
plt.ylabel('Stock Price in $')

plt.show()

 # Plot predicted vs actual prices plt.scatter(y_test, y_pred)

plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')

plt.title('Predicted vs Actual Price')

plt.show()

The data points are mostly close to a diagonal, which indicates, that the predicted values are close to the actual value and the model's performance is largerly quite good.

Yet there are some areas, around 55 to 65, the model seems to be quite random and shows no relationship between the predicted and actual value.

Also in the area around 85 - 110 the data point is spread out quite heavily and the predictions don't cover the values above 100.

Residual Histogram

The residuals are nearly normally distributed around zero, with a slight skewedness to the right.

 # Import norm package to plot normal distribution from scipy.stats
import norm

 # Fit a normal distribution to the data:
mu, std = norm.fit(y_test - y_pred)

ax = sns.distplot((y_test - y_pred), label='Residual Histogram & Distribution')

 # Calculate the pdf over a range of values
x = np.linspace(min(y_test - y_pred), max(y_test - y_pred), 100) p =
norm.pdf(x, mu, std)

 # And plot on the same axes that seaborn put the histogram ax.plot(x, p,
'r', lw=2, label='Normal Distribution')

plt.legend()
plt.show()

 # Add new column for predictions to df
df['Prediction'] = model.predict(np.array(df.index).reshape(-1, 1))

df.head()

Error Evaluation Metrics

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:
Mean Squared Error (MSE) is the mean of the squared errors:
Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors: Minimize all of these are **cost functions**.

# Import metrics package from sklearn for statistical analysis from sklearn
import metrics

Statistical summary of test data df['Close'].describe()

 # Calculate and print values of MAE, MSE, RMSE
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred)) print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

* The MAE is 3% (of minimum) and 6% (of maximum) of the Closing Price.
* The other two errors are larger, because the errors are squared and have therefore a greater influence on the result.

Accuracy Evaluation Metrics

To see how accurate our model is, calculate the **Coefficient of determination**, which describes the ratio between the total error and the error, that is explained by designed model. It's value is between 0 and 1, with 1 meaning 100% of the error is acoounted for by the model.

Coefficient of determination

Residual Sum of Squares (RSS)

Total Sum of Squares (TSS)

print('R2: ', metrics.r2_score(y_test, y_pred))

from sklearn.metrics import explained_variance_score explained_variance_score(y_test, y_pred)

The value of R^2 shows that are model accounts for nearly 94% of the differences between the actual stock prices and the predicted prices.

Lab Assignment to be done by students:
Use Stock Price history data from the Quadl API and apply a regression analysis method to accurately predict stock prices over time.

1. **Import Libraries, Configure Quandl**
2. **Import API key from file, Quandl API Auth**
3. **Get the Data # Set start and end date for stock prices**
4. **Get descriptive statistics summary of data set, # Display features in data set**
5. **# Select Subset with relevant features, # Create a new DataFrame with only closing price and date**
6. **# Training a Linear Regression Model**
7. **# Split data into train and test set: 80% / 20%**
8. **# Create LinearRegression Object**
9. **# Fit linear model using the train data set**
10. **# Model Evaluation**
11. **# The coefficient # The Intercept**
12. **# Prediction from Model**

13. **# Get number of rows in data set for random sample**
14. **# Generate 25 random numbers**
15. **# Plot predicted vs actual prices**
16. **# Plot Residual Histogram**
17. **# Error Evaluation Metrics, # Calculate and print values of MAE, MSE, RMSE**
18. **# Accuracy Evaluation Metrics,** **Coefficient of determination**

    Residual Sum of Squares (RSS)

    Total Sum of Squares (TSS)

```
%pip install quandl
```

```
Collecting quandl
  Downloading Quandl-3.7.0-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: pandas>=0.14 in /usr/local/lib/python3.10/dist-packages (from quandl) (1.5.3)
Requirement already satisfied: numpy>=1.8 in /usr/local/lib/python3.10/dist-packages (from quandl) (1.25.2)
Requirement already satisfied: requests>=2.7.0 in /usr/local/lib/python3.10/dist-packages (from quandl) (2.31.0)
Collecting inflection>=0.3.1 (from quandl)
  Downloading inflection-0.5.1-py2.py3-none-any.whl (9.5 kB)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from quandl) (2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from quandl) (1.16.0)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.10/dist-packages (from quandl) (10.1.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.14->quandl) (2023.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->quandl) (3.3.2
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->quandl) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->quandl) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->quandl) (2024.2.2)
Installing collected packages: inflection, quandl
Successfully installed inflection-0.5.1 quandl-3.7.0
```

```
!curl "https://data.nasdaq.com/api/v3/datatables/SHARADAR/SEP.csv?ticker=AAPL&api_key=CP_rsBMcgrLNeXDkWcW7"
```

```
AAPL,2018-11-23,43.735,44.149,43.025,43.072,94495888.0,41.288,172.29,2024-02-09
AAPL,2018-11-21,44.932,45.068,44.138,44.195,124496840.0,42.364,176.78,2024-02-09
AAPL,2018-11-20,44.593,45.367,43.877,44.245,271300988.0,42.412,176.98,2024-02-09
AAPL,2018-11-19,47.5,47.675,46.248,46.465,167683488.0,44.54,185.86,2024-02-09
AAPL,2018-11-16,47.625,48.742,47.365,48.383,147713012.0,46.378,193.53,2024-02-09
AAPL,2018-11-15,47.097,47.992,46.725,47.852,185915204.0,45.87,191.41,2024-02-09
AAPL,2018-11-14,48.475,48.62,46.483,46.7,243203828.0,44.765,186.8,2024-02-09
AAPL,2018-11-13,47.907,49.295,47.862,48.057,187531744.0,46.066,192.23,2024-02-09
AAPL,2018-11-12,49.75,49.962,48.447,48.542,204542072.0,46.531,194.17,2024-02-09
AAPL,2018-11-09,51.388,51.502,50.562,51.117,137463000.0,48.999,204.47,2024-02-09
AAPL,2018-11-08,52.495,52.53,51.688,52.123,101450544.0,49.963,208.49,2024-02-09
AAPL,2018-11-07,51.492,52.515,51.032,52.487,133697736.0,50.137,209.95,2024-02-09
AAPL,2018-11-06,50.48,51.18,50.422,50.943,127531524.0,48.661,203.77,2024-02-09
AAPL,2018-11-05,51.075,51.097,49.542,50.398,264654676.0,48.141,201.59,2024-02-09
AAPL,2018-11-02,52.388,53.413,51.358,51.87,365314616.0,49.547,207.48,2024-02-09
AAPL,2018-11-01,54.763,55.59,54.203,55.555,233292720.0,53.067,222.22,2024-02-09
AAPL,2018-10-31,54.22,55.112,54.155,54.715,153435732.0,52.265,218.86,2024-02-09
AAPL,2018-10-30,52.788,53.795,52.318,53.325,146639960.0,50.937,213.3,2024-02-09
AAPL,2018-10-29,54.797,54.922,51.523,53.06,183742080.0,50.684,212.24,2024-02-09
AAPL,2018-10-26,53.975,55.047,53.167,54.075,189033500.0,51.653,216.3,2024-02-09
AAPL,2018-10-25,54.428,55.345,54.188,54.95,119423072.0,52.489,219.8,2024-02-09
AAPL,2018-10-24,55.65,56.057,53.635,53.773,163700652.0,51.364,215.09,2024-02-09
AAPL,2018-10-23,53.958,55.812,53.675,55.682,155071384.0,53.189,222.73,2024-02-09
AAPL,2018-10-22,54.947,55.84,54.735,55.163,115168328.0,52.692,220.65,2024-02-09
AAPL,2018-10-19,54.515,55.315,54.358,54.828,132314904.0,52.372,219.31,2024-02-09
AAPL,2018-10-18,54.465,54.935,53.25,54.005,130325260.0,51.587,216.02,2024-02-09
AAPL,2018-10-17,55.575,55.66,54.835,55.297,91541588.0,52.821,221.19,2024-02-09
AAPL,2018-10-16,54.733,55.748,54.191,55.538,116735852.0,53.05,222.15,2024-02-09
AAPL,2018-10-15,55.29,55.458,54.318,54.34,123164028.0,51.907,217.36,2024-02-09
AAPL,2018-10-12,55.105,55.72,54.21,55.528,161351404.0,53.041,222.11,2024-02-09
AAPL,2018-10-11,53.63,54.875,53.08,53.612,212497568.0,51.212,214.45,2024-02-09
AAPL,2018-10-10,56.365,56.587,54.013,54.09,167962216.0,51.668,216.36,2024-02-09
AAPL,2018-10-09,55.91,56.818,55.562,56.718,107564116.0,54.178,226.87,2024-02-09
AAPL,2018-10-08,55.553,56.2,55.05,55.943,118655692.0,53.437,223.77,2024-02-09
AAPL,2018-10-05,56.99,57.102,55.145,56.072,134321852.0,53.561,224.29,2024-02-09
AAPL,2018-10-04,57.695,58.087,56.682,56.998,128168000.0,54.445,227.99,2024-02-09
AAPL,2018-10-03,57.513,58.367,57.445,58.017,114619196.0,55.419,232.07,2024-02-09
AAPL,2018-10-02,56.812,57.5,56.657,57.32,99152680.0,54.753,229.28,2024-02-09
AAPL,2018-10-01,56.987,57.355,56.587,56.815,94403208.0,54.271,227.26,2024-02-09
AAPL,2018-09-28,56.197,56.46,56.005,56.435,91717456.0,53.908,225.74,2024-02-09
AAPL,2018-09-27,55.955,56.61,55.885,56.237,120724908.0,53.719,224.95,2024-02-09
AAPL,2018-09-26,55.25,55.938,54.94,55.105,95938824.0,52.637,220.42,2024-02-09
AAPL,2018-09-25,54.938,55.705,54.925,55.547,98217516.0,53.06,222.19,2024-02-09
AAPL,2018-09-24,54.205,55.315,54.157,55.197,110773432.0,52.726,220.79,2024-02-09
AAPL,2018-09-21,55.195,55.34,54.322,54.415,384986992.0,51.978,217.66,2024-02-09
AAPL,2018-09-20,55.06,55.57,54.788,55.008,106435176.0,52.544,220.03,2024-02-09
AAPL,2018-09-19,54.625,54.905,53.825,54.593,108495332.0,52.148,218.37,2024-02-09
AAPL,2018-09-18,54.447,55.462,54.28,54.56,126286848.0,52.117,218.24,2024-02-09
AAPL,2018-09-17,55.538,55.737,54.318,54.47,148780532.0,52.031,217.88,2024-02-09
AAPL,2018-09-14,56.438,56.71,55.63,55.96,127997156.0,53.454,223.84,2024-02-09
AAPL,2018-09-13,55.88,57.087,55.642,56.602,166825508.0,54.068,226.41,2024-02-09
AAPL,2018-09-12,56.235,56.25,54.96,55.267,197114960.0,52.793,221.07,2024-02-09
```

```
AAPL,2018-09-11,54.502,56.075,54.14,55.962,142996196.0,53.456,223.85,2024-02-09
AAPL,2018-09-10,55.237,55.462,54.117,54.583,158065812.0,52.138,218.33,2024-02-09
AAPL,2018-09-07,55.462,56.343,55.178,55.325,150479240.0,52.847,221.3,2024-02-09
AAPL,2018-09-06,56.557,56.837,55.325,55.775,137159904.0,53.277,223.1,2024-02-09
AAPL,2018-09-05,57.248,57.417,56.275,56.718,133331840.0,54.178,226.87,2024-02-09
AAPL,2018-09-04,57.102,57.295,56.657,57.09,109560528.0,54.533,228.36,2024-02-09
```

```
%pip install nasdaq-data-link
import nasdaqdatalink
```

```
Collecting nasdaq-data-link
  Downloading Nasdaq_Data_Link-1.0.4-py2.py3-none-any.whl (28 kB)
Requirement already satisfied: pandas>=0.14 in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (1.5.3)
Requirement already satisfied: numpy>=1.8 in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (1.25.2)
Requirement already satisfied: requests>=2.7.0 in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (2.31.0)
Requirement already satisfied: inflection>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (0.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (1.16.0)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.10/dist-packages (from nasdaq-data-link) (10.1.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.14->nasdaq-data-link) (2023.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->nasdaq-data-li
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->nasdaq-data-link) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->nasdaq-data-link) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.7.0->nasdaq-data-link) (2
Installing collected packages: nasdaq-data-link
Successfully installed nasdaq-data-link-1.0.4
```

```
import nasdaqdatalink nasdaqdatalink.ApiConfig.api_key =
"CP_rsBMcgrLNeXDkWcW7"
```

```
data = nasdaqdatalink.get_table('SHARADAR/SEP', ticker='AAPL')
```

```
/usr/local/lib/python3.10/dist-packages/urllib3/connectionpool.py:1100: InsecureRequestWarning: Unverified HTTPS request is being made t
warnings.warn(
```

```
data
```

	ticker	date	open	high	low	close	volume	closeadj	closeunadj	las
None										
0	AAPL	2018-12-31	39.633	39.840	39.120	39.435	140013864.0	37.801	157.74	20
1	AAPL	2018-12-28	39.375	39.630	38.638	39.057	169165696.0	37.439	156.23	20
2	AAPL	2018-12-27	38.960	39.193	37.517	39.038	212468260.0	37.420	156.15	20
3	AAPL	2018-12-26	37.075	39.307	36.680	39.292	234330176.0	37.664	157.17	20
4	AAPL	2018-12-24	37.038	37.888	36.648	36.708	148676928.0	35.186	146.83	20
...	...	...	...	...	...	...	...	...	...	
77	AAPL	2018-09-10	55.237	55.462	54.117	54.583	158065812.0	52.138	218.33	20
78	AAPL	2018-09-07	55.462	56.343	55.178	55.325	150479240.0	52.847	221.30	20

Next steps: Generate code with data View recommended plots

```
print(data.describe())
print(data.columns)
```

```
                date       open       high        low      close  \
count      82.000000  82.000000  82.000000  82.000000  82.000000
mean   736998.085366  50.243037  50.819232  49.485488  50.109305   std
34.382095   6.242109   6.262278   6.299712   6.314060   min
```

```
736941.000000  37.038000  37.888000  36.648000  36.708000   25%
736969.250000  44.284750  45.071750  43.769000  44.177750
50%    736997.500000  53.794000  54.890000  52.699000  53.468500   75%
737026.750000  55.280000  55.833000  54.640750  55.318000   max
737059.000000  57.695000  58.367000  57.445000  58.017000

              volume   closeadj  closeunadj
 count  8.200000e+01  82.000000   82.000000
 mean   1.623882e+08  47.927476  200.437317
 std    5.969095e+07   5.968683   25.256091
 min    9.154159e+07  35.186000  146.830000
 25%    1.249443e+08  42.347500  176.712500
50%    1.496299e+08  51.074500  213.875000  75%
1.828051e+08  52.840500  221.272500  max
3.849870e+08  55.419000  232.070000
Index(['ticker', 'date', 'open', 'high', 'low', 'close', 'volume', 'closeadj',
       'closeunadj', 'lastupdated'],
dtype='object')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82 entries, 0 to 81
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
0   ticker       82 non-null     object
1   date         82 non-null     datetime64[ns]
2   open         82 non-null     float64
3   high         82 non-null     float64
4   low          82 non-null     float64
5   close        82 non-null     float64
6   volume       82 non-null     float64
7   closeadj     82 non-null     float64
8   closeunadj   82 non-null     float64        9   lastupdated  82 non-null     datetime64[ns] dtypes: datetime64[ns](2), float64(7),
    object(1) memory usage: 6.5+ KB
```

```
selected_columns = data[['date', 'close']]
selected_columns
```

None	date	close
0	737059	39.435
1	737056	39.057
2	737055	39.038
3	737054	39.292
4	737052	36.708
...	...	...
77	736947	54.583
78	736944	55.325
79	736943	55.775
80	736942	56.718
81	736941	57.090

82 rows × 2 columns

Next steps: Generate code with selected_columns View recommended plots

```
import pandas as pd import numpy as np from sklearn.model_selection import
train_test_split from sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

```
import datetime as dt data['date'] =
pd.to_datetime(data['date'])
data['date']=data['date'].map(dt.datetime.toordinal
)
```

```
# Step 7: Split data into train and test set: 80% / 20%
X = data[['date']]  # Feature (date)
y = data['close']   # Target variable (close price)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Step 8: Create LinearRegression Object model =
LinearRegression()


# Step 9: Fit linear model using the train data set
model.fit(X_train, y_train)
```

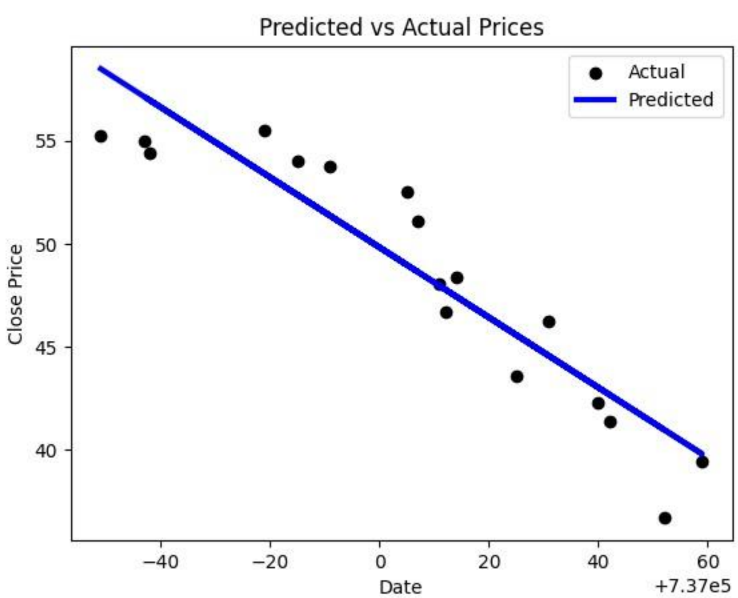
```
▾LinearRegression
LinearRegression()
```

```
# Step 10: Model Evaluation # The
coefficient print("Coefficient:",
model.coef_)
# The Intercept print("Intercept:",
model.intercept_)
```

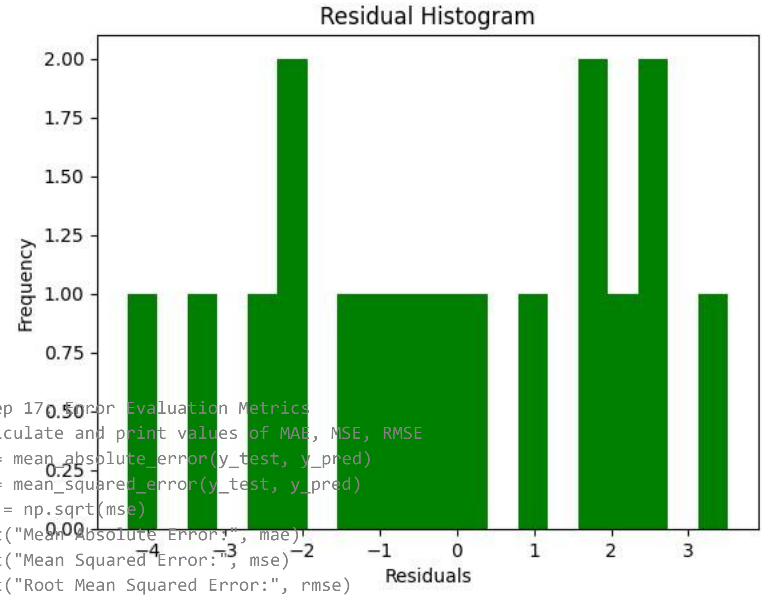
```
Coefficient: [-0.1699761]
Intercept: 125322.20268305475
```

```
# Step 12: Prediction from Model
y_pred = model.predict(X_test)


# Step 15: Plot predicted vs actual prices plt.scatter(X_test, y_test,
color='black', label='Actual') plt.plot(X_test, y_pred, color='blue',
linewidth=3, label='Predicted') plt.xlabel('Date') plt.ylabel('Close
Price') plt.legend() plt.title('Predicted vs Actual Prices')
plt.show()
```




```
# Step 16: Plot Residual Histogram
residuals = y_test - y_pred
plt.hist(residuals, bins=20,
color='green') plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Residual Histogram') plt.show()
```




```
# Step 17: Error Evaluation Metrics
# Calculate and print values of MAE, MSE, RMSE
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
Mean Absolute Error: 1.915621580405647
Mean Squared Error: 4.8537922165995155
Root Mean Squared Error: 2.203132364747864
```

```
# Step 18: Accuracy Evaluation Metrics, Coefficient of determination (R^2)
r2 = r2_score(y_test, y_pred) print("Coefficient of determination
(R^2):", r2)

# Additional: Residual Sum of Squares (RSS) and Total Sum of Squares (TSS)
rss = np.sum((y_test - y_pred) ** 2) tss = np.sum((y_test -
np.mean(y_test)) ** 2) print("Residual Sum of Squares (RSS):", rss)
print("Total Sum of Squares (TSS):", tss)
```

```
Coefficient of determination (R^2): 0.8618569673780222
Residual Sum of Squares (RSS): 82.51446768219176
Total Sum of Squares (TSS): 597.3118304705885
```