Project

October 1, 2023

Qinyang Huang
Zhijiang Yang
Xinqian Li

1 Abstract

This study explores the predictive capabilities of linear and logistic regression models in forecasting current stock prices, utilizing a dataset of historical stock prices. Linear regression offers continuous price predictions, while logistic regression provides binary classification of price movements. Through rigorous evaluation and comparison of these models across diverse stocks and market conditions, our findings contribute valuable insights into their effectiveness, highlighting the potential for combining both techniques to enhance stock price prediction accuracy and offering a practical framework for market participants seeking informed decision-making tools in the dynamic world of finance.

Key word: Linear Regression, Logistic Regression, Stock Prediction

2 Introduction

The world of financial markets is characterized by its relentless dynamism, where fortunes are made and lost in the blink of an eye. Within this fast-paced environment, the ability to forecast stock prices accurately remains a coveted skill, one that can mean the difference between financial success and peril. In recent years, the application of machine learning techniques has opened new avenues for investors and analysts seeking to decipher the cryptic movements of stock markets. This project sets out to explore one such avenue, leveraging the power of linear regression and logistic regression to predict current stock prices based on the price data from the preceding ten days. Throughout this project, we will delve into the realms of data collection and preprocessing, ensuring the reliability and relevance of our dataset. We will use the data from the previous ten days as a feature to predict the stock price or rise or fall on that day. Finally, we will rigorously evaluate our linear regression model's performance, shining a light on its ability to navigate the complexities of stock price prediction.

3 Dataset Description

This dataset was provided by Yahoo Finance. It is publicly available for financial data scientists to use for finance analysis. We have exported the closing prices of Apple stock from 730 days ago up to today as the training set for the model, and the closing prices from 180 days ago up to today as the test set for the model.

4 Data Processing

Within this section we will cover some of the basic data-preprocessing techniques we had employed to get to a more understandable and descriptive data.

4.1 Reading the data

The data was first read from the Yahoo Finance using the following code. The table below shows how the data looks without any kind of pre-processing. Then, we get the data for Apple Company as follows.

```
# import libraries
import yfinance as yf
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import datetime
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,␣
↪accuracy_score
from math import sqrt

# Fetch data
start = datetime.datetime.today() - datetime.timedelta(days = 730)
end = datetime.datetime.today() - datetime.timedelta(days = 180)
AAPL = yf.download('AAPL', start, end)
AAPL.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-09-29	142.470001	144.449997	142.029999	142.830002	141.196472	74602000
2021-09-30	143.660004	144.380005	141.279999	141.500000	139.881699	89056700
2021-10-01	141.899994	142.919998	139.110001	142.649994	141.018509	94639600
2021-10-04	141.759995	142.210007	138.270004	139.139999	137.548676	98322000
2021-10-05	139.490005	142.240005	139.360001	141.110001	139.496140	80861100

4.2 Shaping and describing the data

We generate a 10-day lagged value for training machine learning models and then we get a 369*10 vector for X training set and a 369*1 vector for y training set. The descriptive statistics for X and y training sets are as follows.

Descriptive Statistics for y training set	
Count	368.000000
Mean	154.483207
Standard Deviation (std)	12.914451
Minimum (min)	125.019997
25%	145.485004
Median (50%)	152.665001
75%	164.877495
Maximum (max)	182.009995

	0	1	2	3	4	5	6	7	8	9
count	369.000000	369.000000	369.000000	369.000000	369.000000	369.000000	369.000000	369.000000	369.000000	369.000000
mean	153.964282	154.003767	154.051951	154.093089	154.146721	154.198591	154.242710	154.281626	154.330054	154.383035
std	13.036578	13.024823	13.011321	12.999165	12.978112	12.964107	12.950175	12.938743	12.929467	12.922186
min	125.019997	125.019997	125.019997	125.019997	125.019997	125.019997	125.019997	125.019997	125.019997	125.019997
25%	144.289993	144.490005	144.800003	144.839996	144.869995	145.029999	145.309998	145.380005	145.429993	145.429993
50%	151.600006	151.600006	151.729996	151.759995	151.919998	152.059998	152.339996	152.369995	152.550003	152.570007
75%	164.850006	164.850006	164.850006	164.850006	164.850006	164.850006	164.850006	164.850006	164.850006	164.850006
max	182.009995	182.009995	182.009995	182.009995	182.009995	182.009995	182.009995	182.009995	182.009995	182.009995

```
def data(df, gap = 10):
    X_train, y_train = [],[]
    for i in range(len(df)-gap):
        X_train.append(df['Close'].iloc[i:i+gap].values)
        y_train.append(df['Close'].iloc[i+gap])
    return pd.DataFrame(X_train), pd.Series(y_train)
X_train, y_train = data(df)
X_train = X_train.values, y_train = y_train.values
```

5 Predictive Modeling for Financial Analysis

5.1 Model Description

In this section, we employed a variety of machine learning algorithms with the dual aim of forecasting Apple Inc.'s closing stock price based on the preceding ten days of closing prices and determining whether the upcoming closing price will exceed the prior one by more than 2%.

For the initial objective, we explored two distinct algorithms: Linear Regression and the Normal Equation. The Linear Regression model was formulated utilizing the Gradient Descent algorithm, which enabled meticulous adjustments to optimize accuracy in our forecasts. In contrast, the Normal Equation was resolved through pure matrix calculations, allowing for a different approach to obtaining precise results. Each method provided its unique insights, contributing to a richer, more diversified understanding of the price trends of Apple Inc.'s stocks.

First, we need to understand the basic mathematical definition about models.

- Linear Regression

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$
$$\frac{\partial}{\partial w}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})x^{(i)}$$
$$\frac{\partial}{\partial b}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})$$
$$w = w - \alpha\frac{\partial}{\partial w}J(w,b)$$
$$b = b - \alpha\frac{\partial}{\partial b}J(w,b)$$

- Normal Equation

$$\boldsymbol{\theta} = (X^T X)^{-1}X^T \boldsymbol{y}$$

For the subsequent objective, we applied Logistic Regression, leveraging the Gradient Descent algorithm, to ascertain whether the forthcoming closing price would surpass the previous one by more than 2%. This classification serves as a crucial marker, aiding in the identification of potential significant increases in stock value.

- Logistic Regression

$$\frac{\partial}{\partial w}h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$
$$\frac{\partial}{\partial \theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

This comprehensive modeling approach was constructed to yield in-depth insights and precise predictions, facilitating improved decision-making and strategic planning in financial analysis and investment contexts. By combining diverse predictive models, we strive to offer a well-rounded perspective and a more in-depth comprehension of the market dynamics impacting Apple Inc.'s stock prices.

5.2 Model Construction

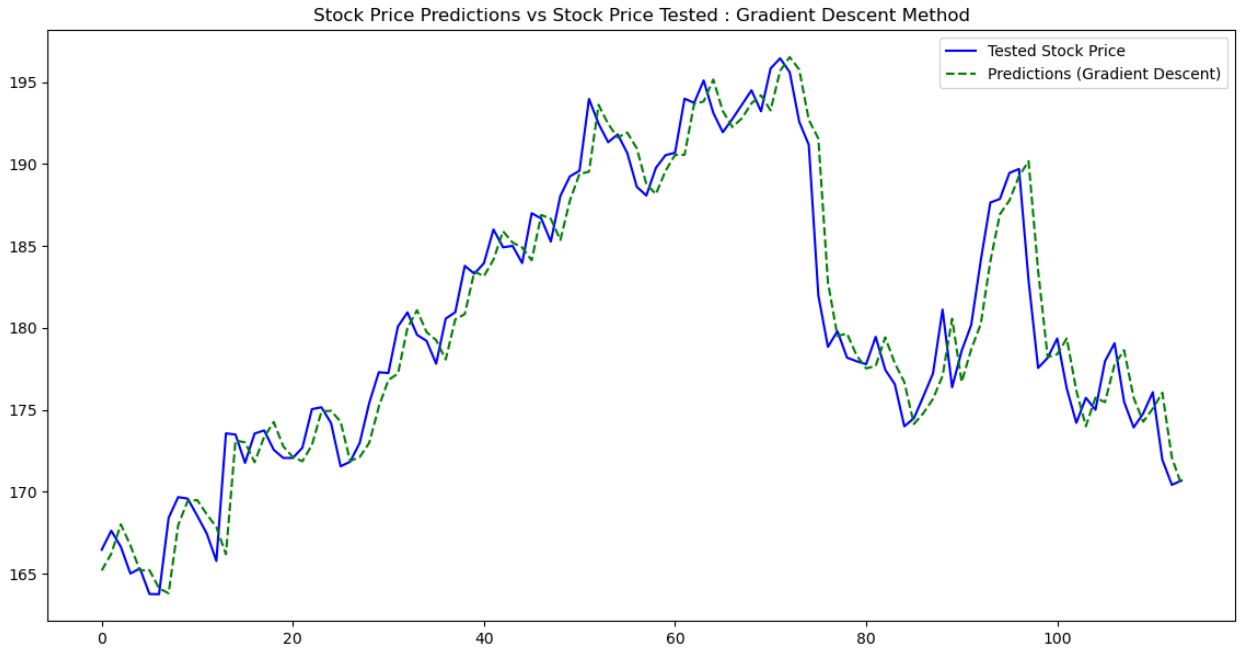
5.2.1 Gradient Descent Method

```
# Gradient Descent Method
def cost_function(X, y, w, b):
    m = X.shape[0]
```

```
    cost = 0

    for i in range (m):
        f_wb = np.dot(X[i], w) + b
        cost += (f_wb - y[i])**2
    total_cost = cost/(2*m)
    return total_cost

def gradient_descent(X, y, w, b):
    m,n = X.shape
    dj_dw = np.zeros(n,)
    dj_db = 0.
    for i in range(m):
        error = np.dot(X[i],w) + b - y[i]
        for j in range(n):
            dj_dw[j] = dj_dw[j] + error * X[i,j]
        dj_db = dj_db + error

    dj_dw = dj_dw / m
    dj_db = dj_db / m
    return dj_dw, dj_db

def gradient_function(X, y, w, b, alpha, iterations, gradient_descent, cost_
 ↪function):
    J_history = []
    for i in range(iterations):
        dj_dw, dj_db = gradient_descent(X, y, w, b)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db
        if i < 10000:
            J_history.append(cost_function(X, y, w, b))
        if i % (iterations//10) == 0:
            print(i, J_history[-1])
    return w,b,J_history

b_in = 0.
w_in = np.zeros(X_train.shape[1],)
alpha = 0.00000835
iterations = 50000
w_final, b_final, J_hist = gradient_function(X_train, y_train, w_in, b_in, alpha,␣
 ↪iterations, gradient_descent,cost_function)
print(w_final, b_final)

start_test = datetime.datetime.today() - datetime.timedelta(days = 180)
end_test = datetime.datetime.today() - datetime.timedelta(days = 0)
AAPL_test = yf.download('AAPL', start_test, end_test)
```

```
def data(df_test, gap = 10):
    X_test, y_test = [],[]
    for i in range(len(df_test)-gap):
        X_test.append(df_test['Close'].iloc[i:i+gap].values)
        y_test.append(df_test['Close'].iloc[i+gap])
    return pd.DataFrame(X_test), pd.Series(y_test)

X_test, y_test = data(df_test)
m = X_test.shape[0]
y_predict = np.zeros(m,)
for i in range(m):
    y_predict[i] = np.dot(w_final, X_test[i]) + b_final
print(y_predict[1:10])
r2 = r2_score(y_test, y_predict)

# Plot the predictions
plt.figure(figsize=(14, 7))
plt.plot(y_test, label="Tested Stock Price", color="blue")
plt.plot(y_predict, label="Predictions (Gradient Descent)", color="green",␣
 ↪linestyle="--")
plt.legend()
plt.title("Stock Price Predictions vs Stock Price Tested : Gradient Descent Method
 ↪")
plt.show()
```

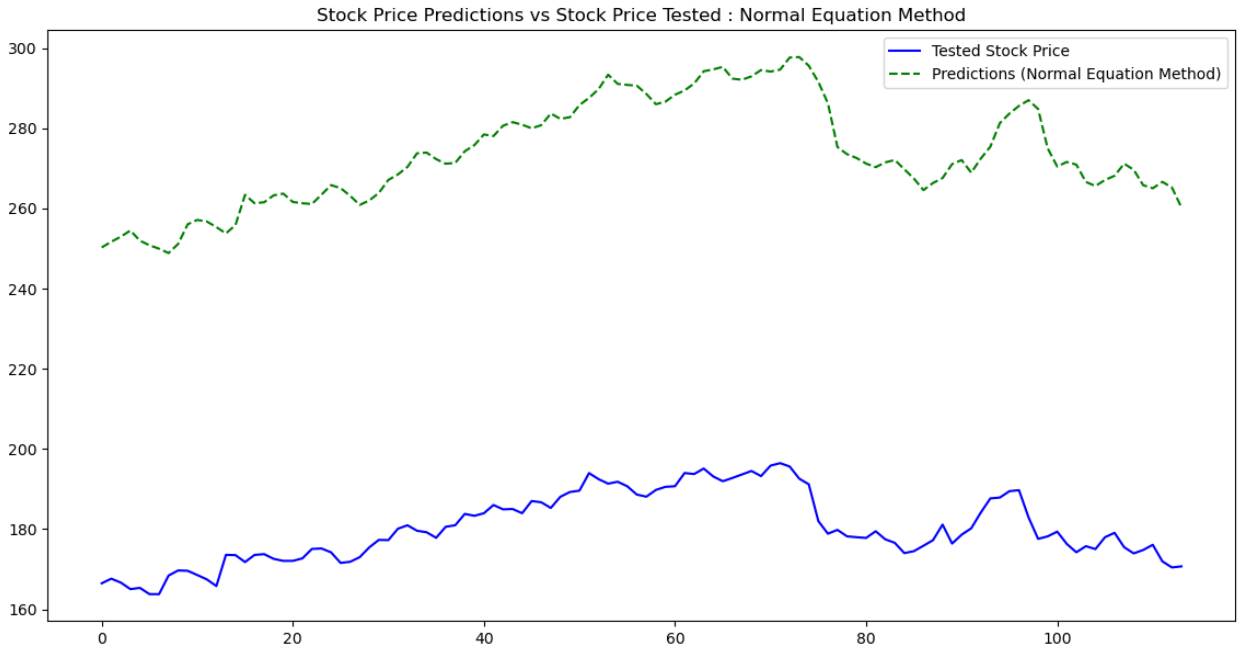
We find that when alpha equals 0.00000835, the result converges, so we use it as our learning rate and then plot the result of our prediction.



5.2.2 Normal Equation Method

```
X_train = np.hstack([X_train, np.ones_like(X_train)])
w = np.linalg.pinv(X_train.T.dot(X_train)).dot(X_train.T).dot(y_train)
y_pre = X_test.dot(w[1:11])

# Plot the predictions
plt.figure(figsize=(14, 7))
plt.plot(y_test, label="Tested Stock Price", color="blue")
plt.plot(y_pre, label="Predictions (Normal Equation Method)", color="green",␣
 ↪linestyle="--")
plt.legend()
plt.title("Stock Price Predictions vs Stock Price Tested : Normal Equation Method")
plt.show()
```

Normal Equation Method does not have a good way to accomplish the prediction task because the feature matrices are singular matrices. The result of our prediction is as follow.



5.2.3 Logistic Regression

```
y_train = (df_test['Close'].shift(-1) > df['Close']*1.02).astype(int)
y_train.dropna(inplace = True)
y_train = y_train.values
y_train.reshape(1,-1)
X_train = df['Close'].values
X_train.reshape(-1,1)

def sigmoid(Z):
```

```
    g = 1 / (1+np.exp(-Z))
    return g

def compute_cost_logistic(X,y,w,b):
    m = X.shape[0]
    cost = 0.
    for i in range(m):
        f_wb_i = sigmoid(np.dot(X[i],w) + b)
        cost += (-y[i]*np.log(f_wb_i))-(1-y[i])*np.log(1-f_wb_i)
    total_cost = cost/m
    return total_cost

def compute_gradient(X,y,w,b):
    m = 378
    dj_dw = 0.
    dj_db = 0.
    for i in range(m):
        f_wb_i = sigmoid(np.dot(X[i],w) + b)
        error = f_wb_i - y[i]
        dj_dw = dj_dw + error * X[i]
        dj_db = dj_db + error
    dj_db = dj_db / m
    dj_dw = dj_dw / m
    return dj_db, dj_dw

def gradient_descent(X,y,w,b,alpha,iterations):
    J_history = []
    for i in range(iterations):
        dj_db, dj_dw = compute_gradient(X,y,w,b)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db
        if i < 100000:
            J_history.append(compute_cost_logistic(X,y,w,b))
        if i % (iterations//10) == 0:
            print(i, J_history[-1])

    return w,b,J_history

w_in = np.zeros(1,)
b_in = 0
alpha = 0.00087
iterations = 150000

w_final, b_final, J_hist = gradient_descent(X_train, y_train, w_in, b_in, alpha,␣
 ↪iterations)
print(w_final, b_final)
```

```
def predict(X,w,b):
    m = X.shape[0]
    p = []
    for i in range(m):
        w = np.ones_like(X[i]) * w_in
        f_wb = sigmoid(np.dot(X[i],w)+b)
        if f_wb < 0.5:
            p.append(0)
        else:
            p.append(1)
    return np.array(p)

w_in = -0.01928325
y_pred = predict(X_test, w_in , b_final)
print(y_pred)
start_test = datetime.datetime.today() - datetime.timedelta(days = 180)
end_test = datetime.datetime.today() - datetime.timedelta(days = 0)
AAPL = yf.download('AAPL', start_test, end_test)
df_test = pd.DataFrame(AAPL.Close)
df_test.shape
y_test = (df_test['Close'].shift(-1) > df_test['Close']*1.02).astype(int)
y_test.dropna(inplace = True)
y_test = y_test.values
y_test.shape
X_test = df_test['Close'].values

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
 ↪score, roc_auc_score, confusion_matrix
from sklearn.model_selection import cross_val_score

# Retrieve the probability of the positive class
# Calculate various evaluation metrics.
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc_roc = roc_auc_score(y_test, y_pred)

# Output the evaluation metrics.
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# Output the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

5.3 Results

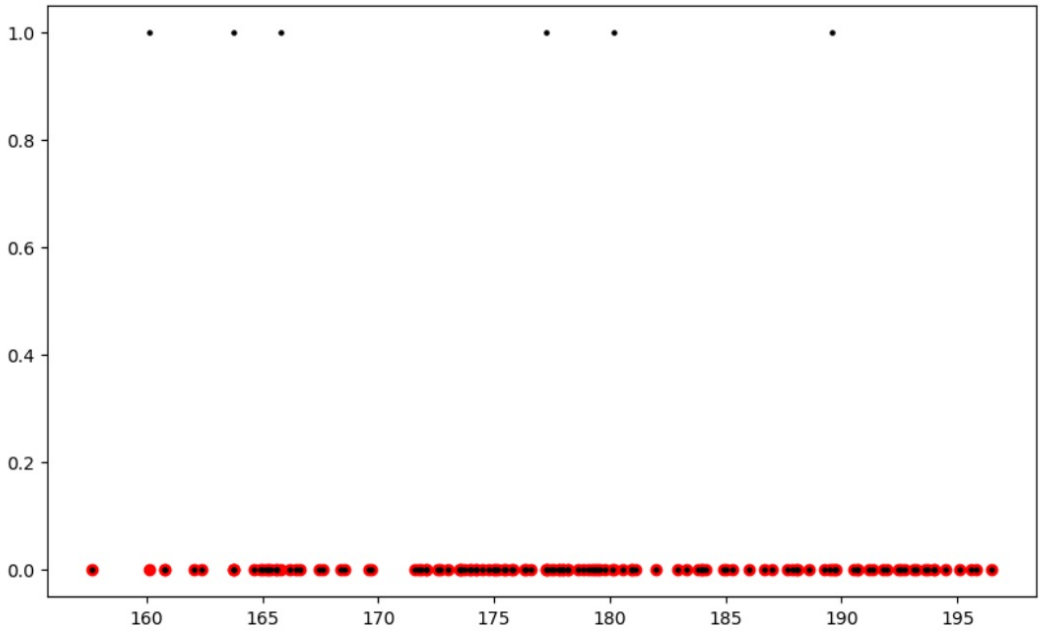
For the evaluation of linear regression models, we selected four assessment metrics: Root Mean Square Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2). We then conducted a comparative analysis across three different approaches: Normal Equation, Linear Regression, and Scikit-learn. The results are as follows:

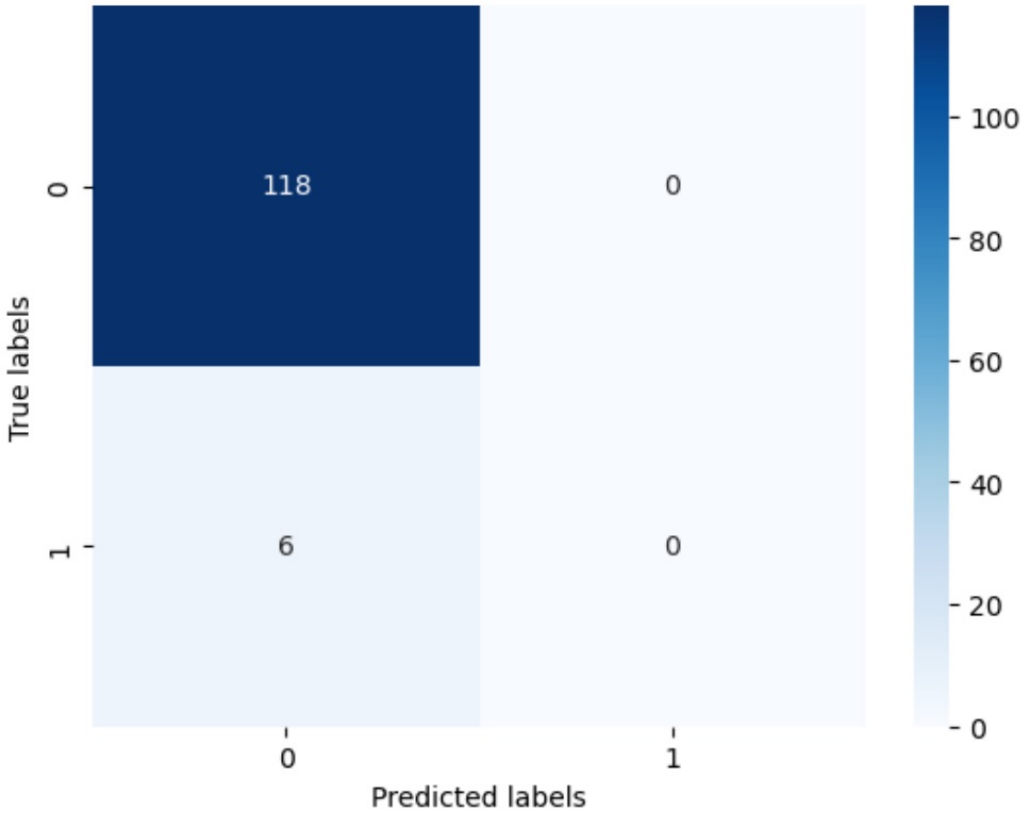
Method	RMSE	MSE	MAE	R^2
Normal Equation	91.469	8366.53	91.27	-102.56
Linear Regression	2.331	5.43	1.68	0.932
Scikit-learn	2.426	5.88	1.80	0.927

For the evaluation of logistic regression models, we chose four assessment metrics: Accuracy, Precision, Recall, and F1 Score. We then performed a comparative analysis across two different approaches: Logistic Regression and Scikit-learn. The results are presented below.

Method	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.9516	0.0	0.0	0.0
Scikit-learn	0.9516	0.0	0.0	0.0

The result of probability that next day's closing price being higher than 2 percentage of Today's and the confusion matrix are as follows.





Despite the inadequate performance of the logistic regression, attributed to the scarcity of positive samples in our experiment, we plan to expand the test set. This enlargement will facilitate the implementation of a meticulously fine-tuned deep learning model, anticipated to deliver superior results.

6 Conclusion

In the pursuit of predicting current stock prices based on historical data, this project has provided illuminating insights into the efficacy of linear and logistic regression models. The predictive stock price model, employing linear regression with the initial ten days of data, demonstrates strong performance. In contrast, the logistic regression model tasked with forecasting stock price increases did not exhibit the same level of effectiveness. Meanwhile, Normal Equation Method does not have a good way to accomplish the prediction task because the feature matrices are singular matrices.

7 Future works

Moving forward, we intend to continually refine and expand our predictive models to enhance the accuracy and reliability of Apple's stock price forecasts. The following delineate our prospective strategies and enhancements:

1) Advanced Model Integration: We plan to explore and integrate more sophisticated models, such as ensemble models and neural networks, which may offer richer, more nuanced insights compared to traditional linear and logistic regression models.
2) Feature Engineering & Selection: We aim to delve deeper into feature engineering and selection to identify the most significant predictors of stock price movements, enabling our models to generalize better to unseen data and potentially uncovering novel insights into market dynamics.
3) Parameter Optimization: Further work will involve rigorous optimization of model parameters, utilizing techniques such as grid search and randomized search, to ensure our models are operating at their maximal potential and providing the most accurate predictions possible.
4) Data Enrichment: We anticipate extending our dataset by incorporating additional relevant features, possibly including macroeconomic indicators, social media sentiments, or other market-specific data, which could provide our models with a more comprehensive view of the multifaceted factors influencing stock prices.
5) Model Evaluation & Validation: To ascertain the robustness and reliability of our models, we plan to employ a diverse range of evaluation metrics and validation techniques, allowing for more rigorous assessment of model performance and facilitating ongoing improvements in prediction accuracy.
6) Real-time Prediction & Automation: Ultimately, we aspire to develop a system capable of real-time predictions, automated data retrieval and preprocessing, and continuous learning from new data, enabling more timely and informed decision-making for investors and analysts.

By pursuing these strategies, we aim to significantly advance our capabilities in predicting Apple's stock price movements, providing valuable insights and supporting more informed investment decisions.

8 Reference

1.An Introduction to Machine Learning (Gopinath Rebala, Ajay Ravi, Sanjay Churiwala)

2.Yahoo Finance

Source Code

1.Predict the closing price of a stock or index based on last 10 days of closing prices.

– Use 1 year or two years' data as the training set.

– Use both Normal Equation Method and Gradient Descent Method.

– See if you get the same answer using both methods.

– See how your answers evolve as you play with different values of and criterion for determining convergence.

– Plot your model performance together with real data for 6 months for your choice of

[124]:
```
import yfinance as yf
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,␣
 ↪accuracy_score
from math import sqrt
```

- **Execute the preparation of the training dataset**

[147]:
```
#Fetch Data from 2021-09-24 to 2023-03-21
specified_date = datetime(2023, 9, 24)
start = specified_date - timedelta(days=730)
end = specified_date - timedelta(days=180)
AAPL = yf.download('AAPL', start, end)
AAPL.head()
```

```
[**********100%%**********]  1 of 1 completed
```

[147]:

Date	Open	High	Low	Close	Adj Close
2021-09-24	145.660004	147.470001	145.559998	146.919998	145.239716
2021-09-27	145.470001	145.960007	143.820007	145.369995	143.707413
2021-09-28	143.250000	144.750000	141.690002	141.910004	140.287003
2021-09-29	142.470001	144.449997	142.029999	142.830002	141.196457

```
2021-09-30  143.660004  144.380005  141.279999  141.500000  139.881699

               Volume
Date
2021-09-24   53477900
2021-09-27   74150700
2021-09-28  108972300
2021-09-29   74602000
2021-09-30   89056700
```

[148]:
```
df = pd.DataFrame(AAPL.Close)
```

[149]:
```
# Generate a 10-day lagged value for training machine learning models.
def data(df, gap = 10):
    X_train, y_train = [],[]
    for i in range(len(df)-gap):
        X_train.append(df['Close'].iloc[i:i+gap].values)
        y_train.append(df['Close'].iloc[i+gap])
    return pd.DataFrame(X_train), pd.Series(y_train)

X_train, y_train = data(df)
```

[150]:
```
X_train.describe()
X_train = X_train.values
```

[151]:
```
y_train.describe()
y_train = y_train.values
```

- **Gradient Descent Algorithm**

[152]:
```
def cost_function(X, y, w, b):
    m = X.shape[0]
    cost = 0

    for i in range (m):
        f_wb = np.dot(X[i], w) + b
        cost += (f_wb - y[i])**2

    total_cost = cost/(2*m)
    return total_cost

def gradient_descent(X, y, w, b):
    m,n = X.shape
    dj_dw = np.zeros(n,)
    dj_db = 0.

    for i in range(m):
```

```
        error = np.dot(X[i],w) + b - y[i]
        for j in range(n):
            dj_dw[j] = dj_dw[j] + error * X[i,j]
        dj_db = dj_db + error

    dj_dw = dj_dw / m
    dj_db = dj_db / m
    return dj_dw, dj_db

def gradient_function(X, y, w, b, alpha,␣
 ↪iterations,gradient_descent,cost_function):
    J_history = []

    for i in range(iterations):
        dj_dw, dj_db = gradient_descent(X, y, w, b)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        if i < 10000:
            J_history.append(cost_function(X, y, w, b))
        if i % (iterations//10) == 0:
            print(i, J_history[-1])

    return w,b,J_history
```

[153]:
```
b_in = 0.
w_in = np.zeros(X_train.shape[1],)
alpha = 0.00000835
iterations = 50000
w_final, b_final, J_hist = gradient_function(X_train, y_train, w_in, b_in,␣
 ↪alpha, iterations, gradient_descent,cost_function)
print(w_final, b_final)
```

```
0 11842.181972819059
5000 6.029090324504339
10000 5.40258195250923
15000 5.40258195250923
20000 5.40258195250923
25000 5.40258195250923
30000 5.40258195250923
35000 5.40258195250923
40000 5.40258195250923
45000 5.40258195250923
[-0.01267272  0.05797589 -0.04572824 -0.00915673 -0.05493214  0.07580223
 -0.00121348  0.02589478  0.03654299  0.92747243] 0.01572724641943894
```

- **Execute the preparation of the test dataset**

[154]:
```
#Fetch Data from 2023-03-28 to 2023-09-24
specified_date = datetime(2023, 9, 24)
start_test = specified_date  - timedelta(days = 180)
end_test = specified_date - timedelta(days = 0)

AAPL_test = yf.download('AAPL', start_test, end_test)
AAPL_test
```

```
[**********100%%**********]  1 of 1 completed
```

[154]:

Date	Open	High	Low	Close	Adj Close	Volume
2023-03-28	157.970001	158.490005	155.979996	157.649994	157.219925	45992200
2023-03-29	159.369995	161.050003	159.350006	160.770004	160.331421	51305700
2023-03-30	161.529999	162.470001	161.270004	162.360001	161.917084	49501700
2023-03-31	162.440002	165.000000	161.910004	164.899994	164.450150	68749800
2023-04-03	164.270004	166.289993	164.220001	166.169998	165.716690	56976200
…	…	…	…	…	…	…
2023-09-18	176.479996	179.380005	176.169998	177.970001	177.970001	67257600
2023-09-19	177.520004	179.630005	177.130005	179.070007	179.070007	51826900
2023-09-20	179.259995	179.699997	175.399994	175.490005	175.490005	58436200
2023-09-21	174.550003	176.300003	173.860001	173.929993	173.929993	63047900
2023-09-22	174.669998	177.080002	174.050003	174.789993	174.789993	56663000

[124 rows x 6 columns]

[155]:
```
df_test = pd.DataFrame(AAPL_test.Close)
df_test
```

[155]:

Date	Close
2023-03-28	157.649994

```
2023-03-29  160.770004
2023-03-30  162.360001
2023-03-31  164.899994
2023-04-03  166.169998
…                  …
2023-09-18  177.970001
2023-09-19  179.070007
2023-09-20  175.490005
2023-09-21  173.929993
2023-09-22  174.789993

[124 rows x 1 columns]
```

[156]:
```
# Generate a 10-day lagged value for training machine learning models.
def data(df_test, gap = 10):
    X_test, y_test = [],[]
    for i in range(len(df_test)-gap):
        X_test.append(df_test['Close'].iloc[i:i+gap].values)
        y_test.append(df_test['Close'].iloc[i+gap])
    return pd.DataFrame(X_test), pd.Series(y_test)

X_test, y_test = data(df_test)
```

[157]:
```
X_test = X_test.values
X_test[10]
```

[157]:
```
array([160.1000061 , 165.55999756, 165.21000671, 165.22999573,
       166.47000122, 167.63000488, 166.6499939 , 165.02000427,
       165.33000183, 163.77000427])
```

[158]:
```
y_test = y_test.values
len(y_test)
```

[158]:
```
114
```

- **Predicting the value of y utilizing the regression coefficients**

[159]:
```
m = X_test.shape[0]
y_predict = np.zeros(m,)
for i in range(m):
    y_predict[i] = np.dot(w_final, X_test[i]) + b_final
print(y_predict[1:10])
```

```
[160.00447704 165.25915429 164.95189558 165.18098536 166.21195724
 167.99382153 166.73167744 165.17740193 165.19999721]
```

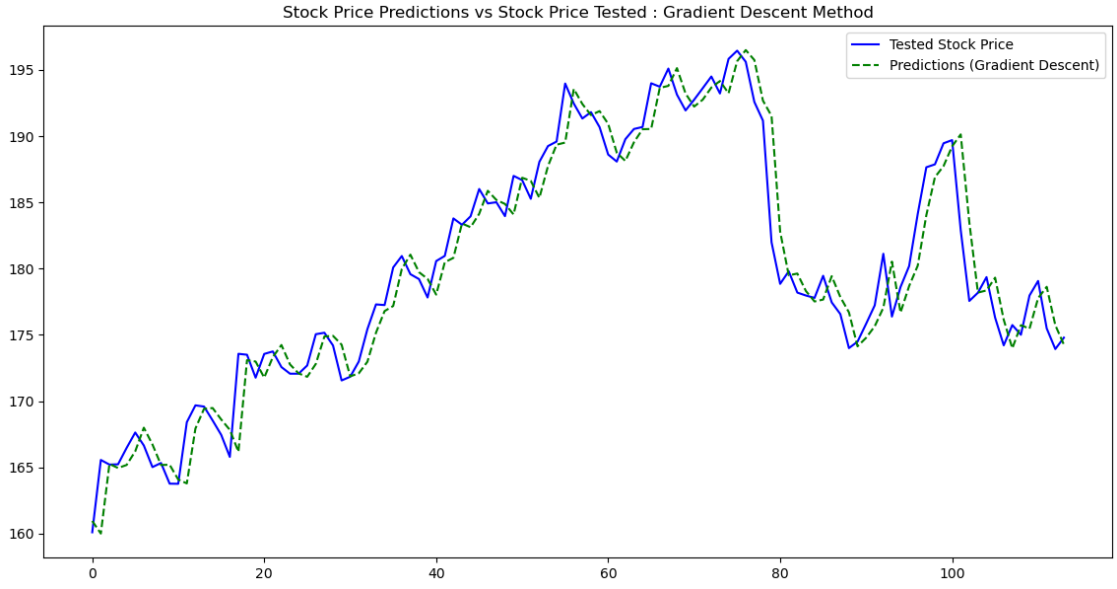
- **Evaluate the goodness of fit of the model**

[161]:
```
RMSE= float(format(np.sqrt(mean_squared_error(y_test, y_predict)), ".3f"))
MSE= mean_squared_error(y_test, y_predict)
MAE= mean_absolute_error(y_test, y_predict)
r2= r2_score(y_test, y_predict)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2)
```

```
RMSE = 2.331
MSE = 5.432708798648557
MAE = 1.687911205854015
R2 = 0.932751203564466
```

- **Plot the predictions**

[41]:
```
plt.figure(figsize=(14, 7))
plt.plot(y_test, label="Tested Stock Price", color="blue")
plt.plot(y_predict, label="Predictions (Gradient Descent)", color="green",␣
 ↪linestyle="--")
plt.legend()
plt.title("Stock Price Predictions vs Stock Price Tested : Gradient Descent␣
 ↪Method")
plt.show()
```



2 Normal Equation

[43]:
```
X_train = np.hstack([X_train, np.ones_like(X_train)])
X_train
```

[43]:
```
array([[146.91999817, 145.36999512, 141.91000366, …,   1.        ,
          1.        ,   1.        ],
       [145.36999512, 141.91000366, 142.83000183, …,   1.        ,
          1.        ,   1.        ],
       [141.91000366, 142.83000183, 141.5       , …,   1.        ,
          1.        ,   1.        ],
       …,
       [150.58999634, 148.5       , 150.47000122, …,   1.        ,
          1.        ,   1.        ],
       [148.5       , 150.47000122, 152.58999634, …,   1.        ,
          1.        ,   1.        ],
       [150.47000122, 152.58999634, 152.99000549, …,   1.        ,
          1.        ,   1.        ]])
```

- **This is a singular matrix, so Python can only provide us with the pseudo-inverse of the matrix**

[44]:
```
w = np.linalg.pinv(X_train.T.dot(X_train)).dot(X_train.T).dot(y_train)
print(w.reshape(1,-1))
w[1:11]
```

```
[[-0.0482739   0.09439403 -0.07995529  0.03184879 -0.0985945   0.1134461
  -0.03198492  0.08435436 -0.07484807  0.97792284  0.49252359  0.49252359
   0.49252359  0.49252359  0.49252359  0.49252359  0.49252359  0.49252359
   0.49252359  0.49252359]]
```

[44]:
```
array([ 0.09439403, -0.07995529,  0.03184879, -0.0985945 ,  0.1134461 ,
       -0.03198492,  0.08435436, -0.07484807,  0.97792284,  0.49252359])
```

[45]:
```
y_pre = X_test.dot(w[1:11])
print(y_pre)
```

```
[243.63323487 242.48001344 244.15461212 249.76193589 248.71753943
 250.10395375 251.39187395 252.90815305 250.41386967 249.25462171
 248.41659602 247.31010586 249.56491284 254.4483982  255.53599649
 255.15674981 253.7585777  252.20561407 254.24927251 261.75515141
 259.7046061  259.92342397 261.65499873 262.06898375 260.02104554
 259.66744108 259.49727105 261.79145681 264.15924461 263.42889695
 261.52632388 259.25485683 260.34337856 262.20914807 265.44411161
 266.82212043 268.67651442 272.06674359 272.24920763 270.64063771
 269.50059893 269.68048619 272.57527129 274.06999966 276.73703847
 276.35044897 278.88104719 279.80891053 279.14885184 278.28246267
 279.0268163  281.92898077 280.60173352 280.99778931 284.02659789
 285.77312291 287.97469003 291.56084561 289.29357953 289.0241093
```

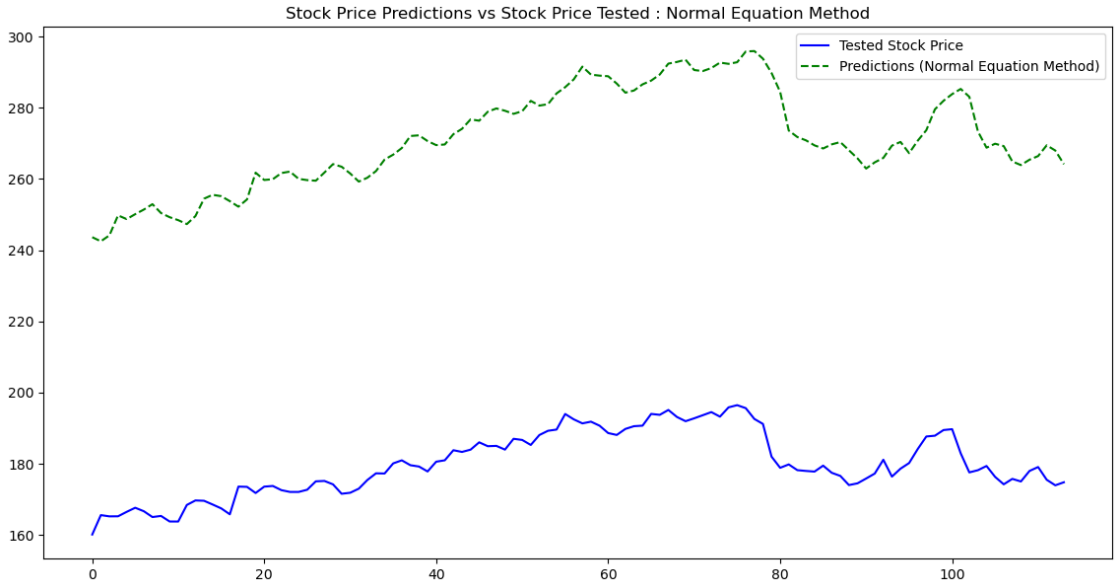
```
 288.84669471 286.78427027 284.2360452  284.83969981 286.57249267
 287.61704245 289.33063047 292.39231765 292.86754862 293.49276239
 290.59203013 290.29752826 291.14508782 292.67814807 292.33302867
 292.7835421  295.79588814 295.92477256 293.77471967 289.76748642
 284.46057091 273.59753153 271.77194222 270.84306799 269.4267246
 268.55076515 269.71446583 270.3557557  268.04813569 265.76741855
 262.91602483 264.66113349 265.90780785 269.35068605 270.39508469
 267.24028113 270.74130619 273.69348348 279.56226135 281.95180353
 283.82733449 285.28464599 283.06901358 273.34062806 268.77795698
 269.89147709 269.2239805  264.90956453 263.89181218 265.40151415
 266.42044638 269.45051933 267.97853607 264.14628616]
```

[46]:
```
print(y_test)
```

```
[160.1000061  165.55999756 165.21000671 165.22999573 166.47000122
 167.63000488 166.6499939  165.02000427 165.33000183 163.77000427
 163.75999451 168.41000366 169.67999268 169.58999634 168.53999329
 167.44999695 165.78999329 173.57000732 173.5        171.77000427
 173.55999756 173.75       172.57000732 172.07000732 172.07000732
 172.69000244 175.05000305 175.16000366 174.19999695 171.55999756
 171.83999634 172.99000549 175.42999268 177.30000305 177.25
 180.08999634 180.94999695 179.58000183 179.21000671 177.82000732
 180.57000732 180.96000671 183.78999329 183.30999756 183.94999695
 186.00999451 184.91999817 185.00999451 183.96000671 187.
 186.67999268 185.27000427 188.05999756 189.25       189.58999634
 193.97000122 192.46000671 191.33000183 191.80999756 190.67999268
 188.61000061 188.08000183 189.77000427 190.53999329 190.69000244
 193.99000549 193.72999573 195.1000061  193.13000488 191.94000244
 192.75       193.61999512 194.5        193.22000122 195.83000183
 196.44999695 195.61000061 192.58000183 191.16999817 181.99000549
 178.8500061  179.80000305 178.19000244 177.97000122 177.78999329
 179.46000671 177.44999695 176.57000732 174.         174.49000549
 175.83999634 177.22999573 181.11999512 176.38000488 178.61000061
 180.19000244 184.11999512 187.6499939  187.86999512 189.46000671
 189.69999695 182.91000366 177.55999756 178.17999268 179.36000061
 176.30000305 174.21000671 175.74000549 175.00999451 177.97000122
 179.07000732 175.49000549 173.92999268 174.78999329]
```

- **Plot the predictions**

[47]:
```
plt.figure(figsize=(14, 7))
plt.plot(y_test, label="Tested Stock Price", color="blue")
plt.plot(y_pre, label="Predictions (Normal Equation Method)",␣
 ↪color="green",linestyle="--")
plt.legend()
plt.title("Stock Price Predictions vs Stock Price Tested : Normal Equation␣
 ↪Method")
plt.show()
```



- **Evaluate the goodness of fit of the model**

[48]:
```
RMSE= float(format(np.sqrt(mean_squared_error(y_test, y_pre)), ".3f"))
MSE= mean_squared_error(y_test, y_pre)
MAE= mean_absolute_error(y_test, y_pre)
r2= r2_score(y_test, y_pre)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2)
```

```
RMSE = 91.469
MSE = 8366.532502043712
MAE = 91.27146170202613
R2 = -102.56513885691409
```

2. Predict whether the next closing price will be higher than 2% of the previous closing price.

– Python is preferred.

– Use 1 year or two years' data as the training set.

– See how your answers evolve as you play with different values of and criterion for determining convergence.

– Plot your model performance together with real data for 6 months for your choice of .

[20]:
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

```
import seaborn as sns
import datetime
import numpy as np
```

- **Execute the preparation of the training dataset**

[59]:
```
#Fetch Data from 2021-09-24 to 2023-03-21
specified_date = datetime(2023, 9, 24)
start = specified_date - timedelta(days=730)
end = specified_date - timedelta(days=180)
AAPL = yf.download('AAPL', start, end)
df_test = pd.DataFrame(AAPL.Close)
```

```
[**********************100%%**********************]  1 of 1 completed
```

[60]:
```
df_test.head()
```

[60]:

Date	Close
2021-09-24	146.919998
2021-09-27	145.369995
2021-09-28	141.910004
2021-09-29	142.830002
2021-09-30	141.500000

[61]:
```
# Generate y train dataset for training machine learning models.
y_train = (df_test['Close'].shift(-1) > df_test['Close']*1.02).astype(int)
y_train.dropna(inplace = True)
y_train = y_train.values
y_train.reshape(1,-1)
```

[61]:
```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
        1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
        0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0]])
```

[62]:
```
sum(y_train)/len(y_train)
```

[62]:
```
0.15873015873015872
```

[103]:
```
X_train = df['Close'].values
X_train = X_train.reshape(-1,1)
```

- **Logistic Regression Algorithm**

[64]:
```
def sigmoid(Z):
    g = 1 / (1+np.exp(-Z))
    return g

def compute_cost_logistic(X,y,w,b):
    m = X.shape[0]
    cost = 0.

    for i in range(m):
        f_wb_i = sigmoid(np.dot(X[i],w) + b)
        cost += (-y[i]*np.log(f_wb_i))-(1-y[i])*np.log(1-f_wb_i)
    total_cost = cost/m
    return total_cost

def compute_gradient(X,y,w,b):
    m = 378
    dj_dw = 0.
    dj_db = 0.

    for i in range(m):
        f_wb_i = sigmoid(np.dot(X[i],w) + b)
        error = f_wb_i - y[i]
        dj_dw = dj_dw + error * X[i]
        dj_db = dj_db + error
    dj_db = dj_db / m
    dj_dw = dj_dw / m

    return dj_db, dj_dw

def gradient_descent(X,y,w,b,alpha,iterations):
    J_history = []
    for i in range(iterations):
        dj_db, dj_dw = compute_gradient(X,y,w,b)

        w = w - alpha * dj_dw
        b = b - alpha * dj_db
```

```
        if i < 100000:
            J_history.append(compute_cost_logistic(X,y,w,b))
        if i % (iterations//10) == 0:
            print(i, J_history[-1])

    return w,b,J_history
```

[65]:
```
w_in = np.zeros(1,)
b_in = 0
alpha = 0.00087
iterations = 150000

w_final, b_final, J_hist = gradient_descent(X_train, y_train, w_in, b_in,␣
 ↪alpha, iterations)
print(w_final, b_final)
```

```
0 [1.10799577]
15000 [0.50045604]
30000 [0.50040777]
45000 [0.50036025]
60000 [0.50031347]
75000 [0.50026742]
90000 [0.50022209]
105000 [0.49393574]
120000 [0.49393574]
135000 [0.49393574]
[-0.01928325] [0.20007008]
```

- **Predicting the value of y utilizing the regression coefficients**

[66]:
```
def predict(X,w,b):
    m = X.shape[0]
    p = []

    for i in range(m):
        w = np.ones_like(X[i]) * w_in
        f_wb = sigmoid(np.dot(X[i],w)+b)
        if f_wb < 0.5:
            p.append(0)
        else:
            p.append(1)
    return np.array(p)
```

[76]:
```
w_in = -0.01928325
y_pred = predict(X_test, w_in , b_final)
print(y_pred)
```

```
len(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

[76]: 124

- **Execute the preparation of the test dataset**

[68]:
```
start_test = specified_date - timedelta(days = 180)
end_test = specified_date - timedelta(days = 0)
```

[69]:
```
AAPL = yf.download('AAPL', start_test, end_test)
df_test = pd.DataFrame(AAPL.Close)
df_test.shape
```

```
[**********************100%%**********************]  1 of 1 completed
```

[69]: (124, 1)

[77]:
```
y_test = (df_test['Close'].shift(-1) > df_test['Close']*1.02).astype(int)
len(y_test)
```

[77]: 124

[71]:
```
y_test.dropna(inplace = True)
```

[72]:
```
y_test = y_test.values
y_test.shape
```

[72]: (124,)

[78]:
```
X_test = df_test['Close'].values
```

- **Evaluate the goodness of fit of the model**

[79]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, confusion_matrix
from sklearn.model_selection import cross_val_score

# Retrieve the probability of the positive class

# Calculate various evaluation metrics.
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

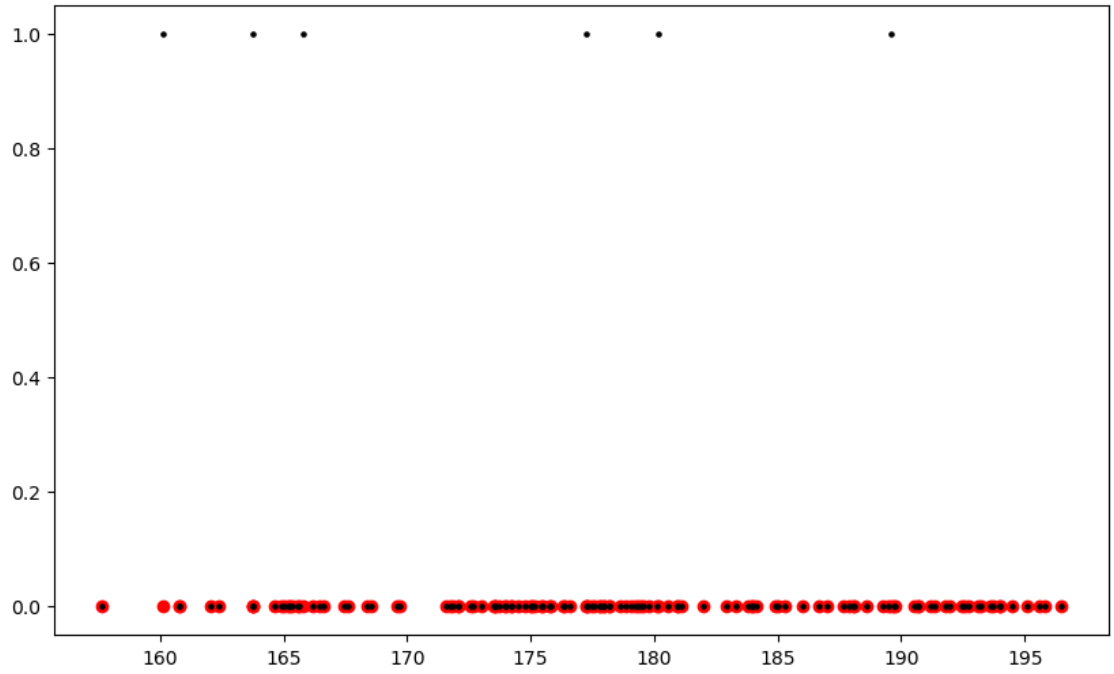
```
auc_roc = roc_auc_score(y_test, y_pred)

# Output the evaluation metrics.
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# Output the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9516129032258065
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
Confusion Matrix:
[[118   0]
 [  6   0]]
```

```
/Users/lilyhuang/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

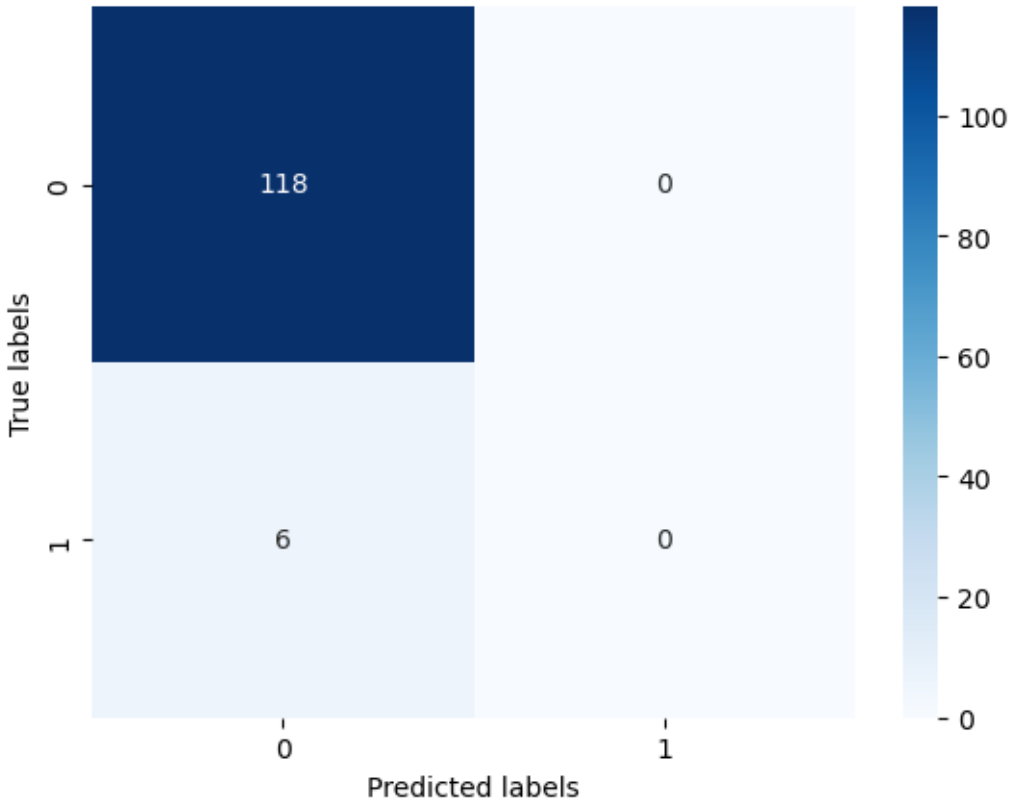
- **Plot the predictions**

[80]:
```
plt.figure(figsize = (10,6))
plt.scatter(X_test,y_test, color = 'black', zorder = 10, s = 5)
plt.scatter(X_test, y_pred, color = 'red')
plt.show()
```



[81]:
```
cm = confusion_matrix(y_test, y_pred)
```

[82]:
```
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```



- **Compare the results with those obtained using the scikit-learn library**

[83]:
```
from sklearn.linear_model import LogisticRegression
X_train = X_train.reshape(-1,1)
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
```

[83]:
```
LogisticRegression()
```

[84]:
```
print(lr_model.coef_,lr_model.intercept_)
```

```
[[-0.01092999]] [-6.84499266e-05]
```

[85]:
```
from sklearn.linear_model import LogisticRegression
X_test = X_test.reshape(-1,1)
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
y_prediction = lr_model.predict(X_test)
```

[86]:
```
len(y_pred)
```

[86]:
```
124
```

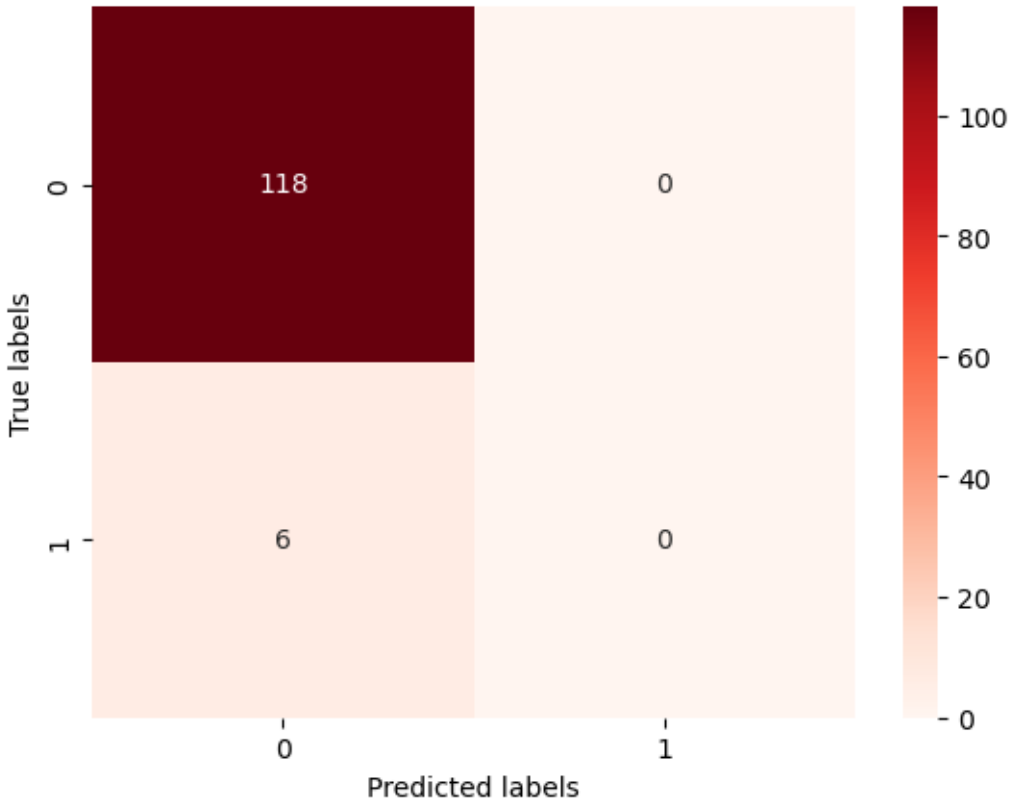
[88]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score, confusion_matrix
from sklearn.model_selection import cross_val_score


# Calculate various evaluation metrics.
accuracy = accuracy_score(y_test, y_prediction)
precision = precision_score(y_test, y_prediction)
recall = recall_score(y_test, y_prediction)
f1 = f1_score(y_test, y_prediction)

# Output the evaluation metrics.
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# Output the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9516129032258065
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
Confusion Matrix:
[[118   0]
 [  6   0]]
```

```
/Users/lilyhuang/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[89]:
```
cm = confusion_matrix(y_test, y_pred)
```

[90]:
```
sns.heatmap(cm, annot=True, cmap='Reds', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```


118
0
6
0
True labels
Predicted labels
0
1
0
1
100
80
60
40
20
0