Students

stock price pediction

TEAM MEMBER

# 

**AU732521104019:JAYASURYA.A**

**PHASE 5: SUBMISSION DOCUMENT**

Team member

Project title: stock price prediction

A graph with arrows pointing up

Description automatically generated with medium confidence

**INTRODUCTION:**

In this project, you want to build a stock price prediction model. You can approach this task using various techniques, including machine learning and time series analysis. Here, I'll give you a simplified example using a basic machine learning approach with Python and some popular libraries like NumPy, Pandas, and Scikit-Learn

1. **Data Collection:**

Stock price prediction relies heavily on historical data. This data typically includes historical stock prices, trading volumes, financial reports, and other relevant information. Data can be collected from various sources such as financial news websites, stock exchanges, and financial databases.

1. **Features:**

In machine learning and data analysis, features are variables that help in making predictions. In stock price prediction, features can include historical stock prices, trading volumes, company financial indicators, economic indicators, sentiment analysis of news articles, and more

.

1. **Time Series Analysis:**

Stock price data is typically analyzed as a time series, which means that the data points are ordered chronologically. Time series analysis techniques are often used to identify patterns, trends, and seasonality in the data.

1. **Model Selection:**

To predict stock prices, various machine learning models can be used, such as linear regression, decision trees, random forests, support vector machines, and deep learning models like recurrent neural networks (RNNs) and long short-term memory networks (LSTMs). The choice of model depends on the complexity of the problem and the quality of the data.

1. **Evaluation Metrics:**

To assess the performance of a stock price prediction model, several evaluation metrics can be used. Common metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and accuracy scores.

1. **Feature Engineering:**

This is the process of selecting, transforming, or creating new features from the raw data to improve the model's predictive performance. It may involve techniques like smoothing, differencing, or lagging.

1. **Overfitting and Generalization:**

Preventing overfitting is crucial in stock price prediction. Overfitting occurs when a model learns to perform well on the training data but doesn't generalize well to new, unseen data. Regularization techniques and careful model selection help mitigate overfitting.

1. **Market Sentiment Analysis:**

Public sentiment, as expressed in news articles, social media, and analyst reports, can influence stock prices. Sentiment analysis techniques are often integrated into models to capture this aspect.

1. **Risk Management:**

Stock price prediction is inherently risky. It's important to understand that no model can predict stock prices with absolute certainty. Risk management strategies, such as stop-loss orders and portfolio diversification, are essential for minimizing potential losses.

1. **Backtesting:**

To assess the effectiveness of a stock price prediction model, it's common to conduct backtesting. This involves applying the model to historical data to see how it would have performed in the past.

**EXPLORATORY ANALYSIS**

To begin this exploratory analysis, first import libraries and define functions for plotting the data using matplotlib. Depending on the data, not all plots will be made. (Hey, I'm just a simple kerneling bot, not a Kaggle Competitions Grandmaster!)

**DATA SOURCE:**

**DATASET LINK**:<https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset/>

**Data Collection and Preprocessing:**

* Importing the dataset: Obtain a comprehensive dataset containing relevant features

such as square footage, number of bedrooms, location, amenities, etc.

* Data preprocessing: Clean the data by handling missing values, outliers, and

categorical variables. Standardize or normalize numerical features.

**PROGRAM:**

**STOCK PRICEPREDICTION**

**Project Overview:**

In this project, we aim to predict the future price of a specific stock based on historical data and relevant features. We'll follow these steps:

**Data Collection:** Obtain historical stock price data and other relevant financial data.

**Feature Engineering**: Create meaningful features from the collected data.

**Model Training:** Train a machine learning model on historical data.

**Model Evaluation:** Evaluate the model's performance using appropriate metrics.

**Predictions:** Use the trained model to make future stock price predictions.

**Data preprocessing**

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt *# plotting*

import numpy as np *# linear algebra*

import os *# accessing directory structure*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

There is 1 csv file in the current version of the dataset:

|  |
| --- |
| In [2]:  for dirname, \_, filenames **in** os.walk('/kaggle/input'):  for filename **in** filenames:  print(os.path.join(dirname, filename))  /kaggle/input/MSFT.csv |

The next hiden code cells define functions for plotting data. Click on the "Code" button in the published kernel to reveal the hidden code.

|  |
| --- |
| # Distribution graphs (histogram/bar graph) of column data  def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):  nunique = df.nunique()  df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have between 1 and 50 unique values  nRow, nCol = df.shape  columnNames = list(df) |
| nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow  plt.figure(num = None, figsize = (6 \* nGraphPerRow, 8 \* nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')  for i in range(min(nCol, nGraphShown)):  plt.subplot(nGraphRow, nGraphPerRow, i + 1)  columnDf = df.iloc[:, i]  if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):  valueCounts = columnDf.value\_counts()  valueCounts.plot.bar()  else:  columnDf.hist()  plt.ylabel('counts')  plt.xticks(rotation = 90)  plt.title(f'{columnNames[i]} (column {i})')  plt.tight\_layout(pad = 1.0, w\_pad = 1.0, h\_pad = 1.0)  plt.show( | |

Now you're ready to read in the data and use the plotting functions to visualize the data.

Let's check 1st file: /kaggle/input/MSFT.csv

|  |  |
| --- | --- |
| |  | | --- | | In [6]:  nRowsRead = 1000 *# specify 'None' if want to read whole file* | |

*# MSFT.csv may have more rows in reality, but we are only loading/previewing the first 1000 ro*

|  |
| --- |
| df1 = pd.read\_csv('/kaggle/input/MSFT.csv', delimiter=',', nrows = nRowsRead)  df1.dataframeName = 'MSFT.csv'  nRow, nCol = df1.shape  print(f'There are **{nRow}** rows and **{nCol}** columns') |

There are 1000 rows and 7 columns

Let's take a quick look at what the data looks like:

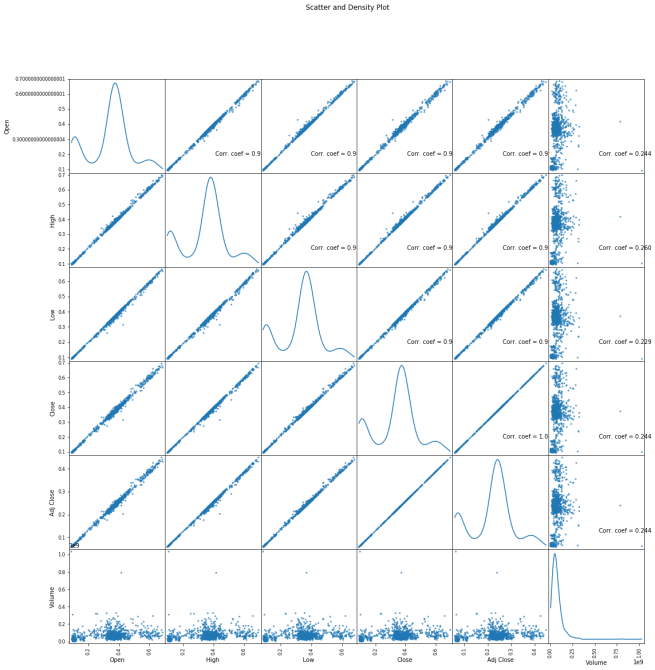
|  |
| --- |
| In [7]:  df1.head(5) |

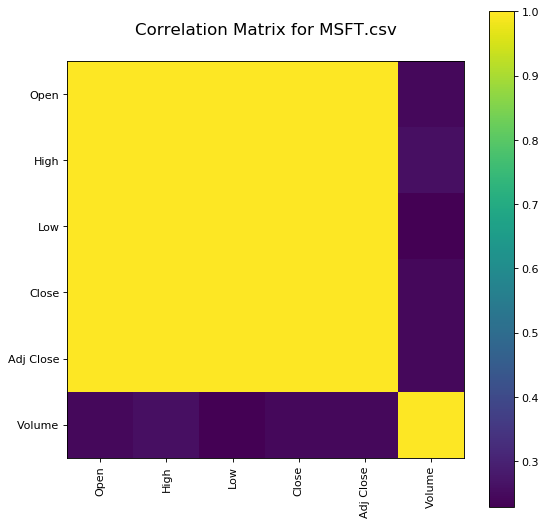
|  |
| --- |
| In [8]:  plotPerColumnDistribution(df1, 10, 5)  <Figure size 2400x512 with 0 Axes>  Correlation matrix: |

Out[7]:

Distribution graphs (histogram/bar graph) of sampled columns:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | Open | High | Low | Close | Adj Close | Volume |
| 1986-03-13 | 0.088542 | 0.101563 | 0.088542 | 0.097222 | 0.062549 | 1031788800 |
| 1986-03-14 | 0.097222 | 0.102431 | 0.097222 | 0.100694 | 0.064783 | 308160000 |
| 1986-03-17 | 0.100694 | 0.103299 | 0.100694 | 0.102431 | 0.065899 | 133171200 |
| 1986-03-18 | 0.102431 | 0.103299 | 0.098958 | 0.099826 | 0.064224 | 67766400 |
| 1986-03-19 | 0.099826 | 0.100694 | 0.097222 | 0.098090 | 0.063107 | 47894400 |





Scatter and density plots

**Feature Engineering**

Feature Engineeringhelps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

* Python3

|  |
| --- |
| splitted = df['Date'].str.split('/', expand=True)    df['day'] = splitted[1].astype('int')  df['month'] = splitted[0].astype('int')  df['year'] = splitted[2].astype('int')    df.head() |

**Output:**

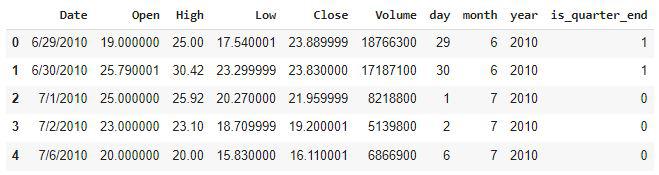


Now we have three more columns namely ‘day’, ‘month’ and ‘year’ all these three have been derived from the ‘Date’ column which was initially provided in the data.

* Python3

|  |
| --- |
| df['is\_quarter\_end'] = np.where(df['month']%3==0,1,0)  df.head() |

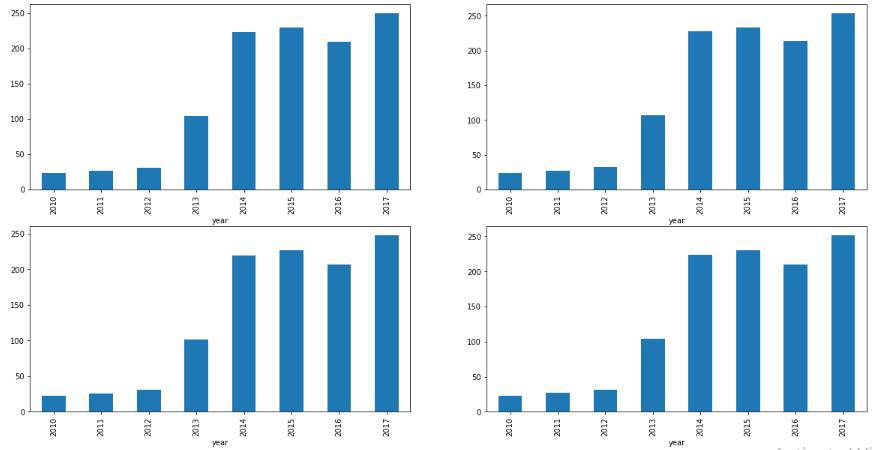
**Output:**



A quarter is defined as a group of three months. Every company prepares its quarterly results and publishes them publicly so, that people can analyze the company’s performance. These quarterly results affect the stock prices heavily which is why we have added this feature because this can be a helpful feature for the learning model.

* Python3

|  |
| --- |
| data\_grouped = df.groupby('year').mean()  plt.subplots(figsize=(20,10))    for i, col in enumerate(['Open', 'High', 'Low', 'Close']):    plt.subplot(2,2,i+1)    data\_grouped[col].plot.bar()  plt.show() |

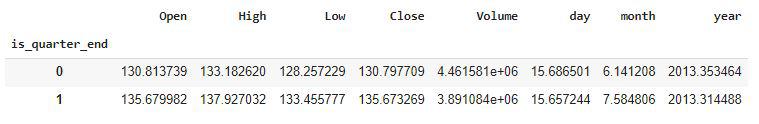
**Output:**

From the above [bar graph](https://www.geeksforgeeks.org/bar-plot-in-matplotlib/), we can conclude that the stock prices have doubled from the year 2013 to that in 2014.

* Python3

|  |
| --- |
| df.groupby('is\_quarter\_end').mean() |

**Output:**



Here are some of the important observations of the above-grouped data:

* Prices are higher in the months which are quarter end as compared to that of the non-quarter end months.
* The volume of trades is lower in the months which are quarter end.
* Python3

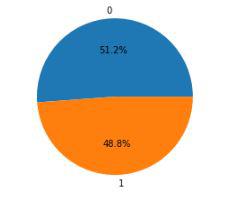
|  |
| --- |
| df['open-close']  = df['Open'] - df['Close']  df['low-high']  = df['Low'] - df['High']  df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0) |

Above we have added some more columns which will help in the training of our model. We have added the target feature which is a signal whether to buy or not we will train our model to predict this only. But before proceeding let’s check whether the target is balanced or not using a [pie chart](https://www.geeksforgeeks.org/plot-a-pie-chart-in-python-using-matplotlib/).

* Python3

|  |
| --- |
| plt.pie(df['target'].value\_counts().values,          labels=[0, 1], autopct='%1.1f%%')  plt.show() |

**Output:**

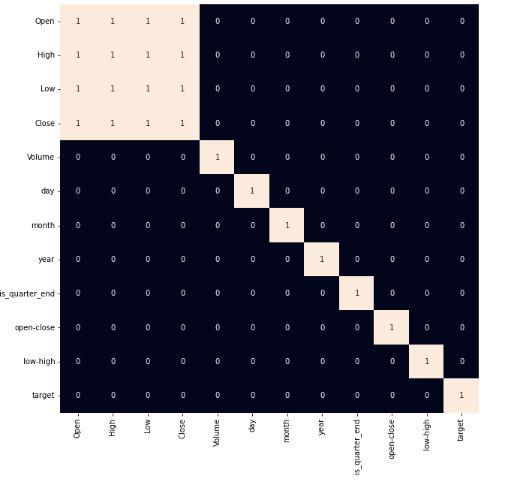


When we add features to our dataset we have to ensure that there are no highly correlated features as they do not help in the learning process of the algorithm.

* Python3

|  |
| --- |
| plt.figure(figsize=(10, 10))    # As our concern is with the highly  # correlated features only so, we will visualize  # our heatmap as per that criteria only.  sb.heatmap(df.corr() > 0.9, annot=True, cbar=False)  plt.show() |

**Output:**



*Heatmap of the correlation between the features*

From the above [heatmap](https://www.geeksforgeeks.org/seaborn-heatmap-a-comprehensive-guide/), we can say that there is a high correlation between OHLC that is pretty obvious, and the added features are not highly correlated with each other or previously provided features which means that we are good to go and build our model.

**Data Splitting and Normalization**

* Python3

|  |
| --- |
| features = df[['open-close', 'low-high', 'is\_quarter\_end']]  target = df['target']    scaler = StandardScaler()  features = scaler.fit\_transform(features)    X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(      features, target, test\_size=0.1, random\_state=2022)  print(X\_train.shape, X\_valid.shape) |

**Output:**

(1522, 3) (170, 3)

After selecting the features to train the model on we should [**normalize**](https://www.geeksforgeeks.org/what-is-data-normalization-and-why-is-it-important/)the data because normalized data leads to stable and fast training of the model. After that whole data has been [**split**](https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/)into two parts with a 90/10 ratio so, that we can evaluate the performance of our model on unseen data

**Model Development and Evaluation**

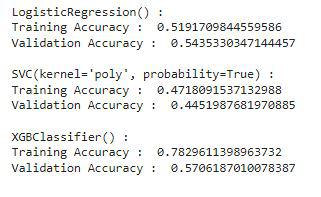
Now is the time to train some state-of-the-art machine learning models([Logistic Regression](https://www.geeksforgeeks.org/understanding-logistic-regression/), [Support Vector Machine](https://www.geeksforgeeks.org/support-vector-machine-algorithm/), [XGBClassifier](https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/)), and then based on their performance on the training and validation data we will choose which ML model is serving the purpose at hand better.

For the evaluation metric, we will use the [ROC-AUC curve](https://www.geeksforgeeks.org/auc-roc-curve/) but why this is because instead of predicting the hard probability that is 0 or 1 we would like it to predict soft probabilities that are continuous values between 0 to 1. And with soft probabilities, the ROC-AUC curve is generally used to measure the accuracy of the predictions.

* Python3

|  |
| --- |
| models = [LogisticRegression(), SVC(    kernel='poly', probability=True), XGBClassifier()]    for i in range(3):    models[i].fit(X\_train, Y\_train)      print(f'{models[i]} : ')    print('Training Accuracy : ', metrics.roc\_auc\_score(      Y\_train, models[i].predict\_proba(X\_train)[:,1]))    print('Validation Accuracy : ', metrics.roc\_auc\_score(      Y\_valid, models[i].predict\_proba(X\_valid)[:,1]))    print() |

**Output:**



*Evaluation of the model on training and the testing data.*

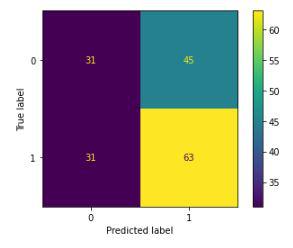
Among the three models, we have trained XGBClassifier has the highest performance but it is pruned to [overfitting](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/) as the difference between the training and the validation accuracy is too high. But in the case of the Logistic Regression, this is not the case.

Now let’s plot a [**confusion matrix**](https://www.geeksforgeeks.org/confusion-matrix-machine-learning/) for the validation data.

* Python3

|  |
| --- |
| metrics.plot\_confusion\_matrix(models[0], X\_valid, Y\_valid)  plt.show() |

**Output:**



*Confusion matrix for the validation data*

**Conclusion:**

We can observe that the accuracy achieved by the state-of-the-art ML model is no better than simply guessing with a probability of 50%. Possible reasons for this may be the lack of data or using a very simple model to perform such a complex task as Stock Market prediction.