STOCK PRICE PREDICTION

PHASE II

Explanation on Problem Statement

The problem is to build a predictive model that forecasts stock prices based on historical market data. The goal is to create a tool that assists investors in making well-informed decisions and optimizing their investment strategies. This project involves data collection, data preprocessing, feature engineering, model selection, training, and evaluation.

As the problem definition of the project tells us that , the main motive of our model is to predict prices of a given stock and then predict its price for the next few days. As this will help the investors in understanding the market using appropriate data points and thus giving them a clear vision in making the right and well informed decisions.

Dataset

A dataset is a structured collection of data that is organized and formatted for a specific purpose. It can contain various types of information, such as text, numbers, images, or any other data. Datasets are commonly used in various fields, including research, machine learning, statistics, and data analysis. They serve as the foundation for tasks like training machine learning models, conducting experiments, and drawing conclusions based on data. Datasets can vary in size and complexity, from small and simple ones to large and intricate collections of information.

This dataset has been taken from kaggle.com
Link: https://www.kaggle.com/datasets/camnugent/sandp500

. A large and well structured dataset on a wide array of companies can be hard to come by. Here we have provided a dataset with historical stock prices (last 5 years) for all companies currently found on the S&P 500 index.

The files are made with .csv extension

note:the dataset includes data up to Feb 2018.

The folder individual_stocks_5yr contains files of data for individual stocks, labelled by their stock ticker name. The all_stocks_5yr.csv contains the same data, presented in a merged .csv file.

DETAILS OF COLUMN USED:

Date - in format: yy-mm-dd

Open - price of the stock at market open (this is NYSE data so all in USD)

High - Highest price reached in the day

Low Close - Lowest price reached in the day

Volume - Number of shares traded

Name - the stock's ticker name

In the model, we generally make use of the date column to mark it correctly in the corresponding row of the graph and we use the close value of the stocks to predict its future price.

We can even make use of open high volume and close to predict even more accurate prices as the no. of parameter increases hence making the model more precise.

LIBRARIES USED:

These packages are commonly used in a stock price prediction model. Here's a brief explanation of their uses:

1. Numpy:
   - NumPy is used for numerical and mathematical operations.
   - It's helpful for handling arrays and matrices, which are common data structures in machine learning.

2. Pandas:
   - Pandas is used for data manipulation and analysis.
   - It can load, clean, and organize the historical stock price data for training and testing.

3. Matplotlib.pyplot:
   - Matplotlib is used for data visualization.
   - Matplotlib.pyplot, a submodule, allows you to create plots and charts to visualize historical stock price trends and model performance.

4. TensorFlow:
   - TensorFlow is an open-source machine learning framework.
   - It's used to build, train, and evaluate machine learning and deep learning models for stock price prediction.

5. TensorFlow.keras.models:
   - This module from TensorFlow contains tools for building machine learning models, including neural networks.

6. TensorFlow.keras.layers:
   - Used in combination with TensorFlow.keras.models to define layers in neural networks, such as input, hidden, and output layers.

7. Sklearn.preprocessing:
   - Scikit-learn's preprocessing module is used for data preprocessing tasks.
   - It helps in scaling, encoding categorical variables, and preparing the data for machine learning models.

8. Sklearn.model_selection - train_test_split:

- The `train_test_split` function from scikit-learn is used to split the dataset into training and testing subsets.
- It's crucial for evaluating the model's performance on unseen data.

9. Sklearn.linear_model - LinearRegression:
- The `LinearRegression` class in scikit-learn is used for building linear regression models.
- Linear regression can be used for simple stock price prediction models based on historical data.

10. Sklearn.metrics:
- Scikit-learn's metrics module contains various metrics to evaluate model performance.
- In the context of stock price prediction, metrics like Mean Absolute Error (MAE) and Mean Squared Error (MSE) can be used to assess the model's accuracy.

These packages collectively provide the tools and functionalities required to preprocess data, build predictive models, and assess their performance in a stock price prediction project. The specific choice of packages and models may vary depending on the complexity and goals of the prediction task.So accordingly we are supposed to import the modules as per our model requirments.

DOWNLOADING THE MODULES:

All the above modules can easily be downloaded using the pip package manager
Just by giving the command in the command prompt as :

pip install {package name}

But if we are using a cloud-based terminal like jupyter notebook or google colab there is no need to download the modules.

TRAINING AND TESTING THE MODEL:

Training and testing a stock market price prediction model involves several steps. Here's a high-level overview:

1. Data Collection: Gather historical stock price data, here the data is taken from the above link .
2. Data Preprocessing:
- Handle missing data: we can fill missing data or remove rows with missing values.
- Feature engineering: Create relevant features or indicators that can improve prediction, like moving averages or technical indicators.

3. Data Splitting:
- Split the data into training, validation, and test sets. A typical split might be 80% for training 20% for testing,which can be done using a built-in function called train_test_split()

4. Model Selection:

- Choose an appropriate machine learning or deep learning model for your prediction task. We here have choosen include Linear Regression and a neural network like CNN or even Long Short-Term Memory networks (LSTMs).

5. Model Training:
- Train the selected model using the training data. The model learns to make predictions based on historical data.

6. Hyperparameter Tuning:
- Fine-tune the model's hyperparameters to optimize its performance. This can be done using techniques like grid search or random search.

7. Model Evaluation:
- Use the validation set to evaluate the model's performance. Common evaluation metrics for regression tasks include Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

8. Model Testing:
Use the test set to evaluate its generalization to unseen data. This helps you assess how well the model might perform in real-world scenarios.

SAMPLE CODE:

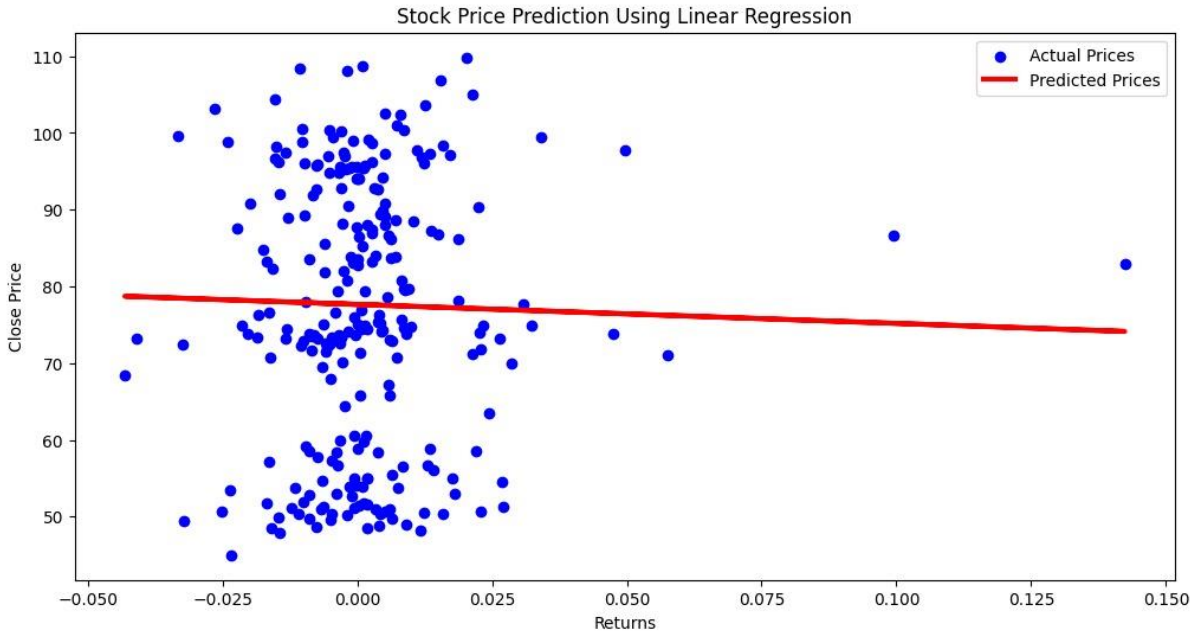
The below is a simple code to predict the stock price using the Linear Regression model

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load your historical stock price data
data = pd.read_csv('data.csv')
# Assume the data contains columns like 'Date', 'Open', 'High', 'Low', 'Close', 'Volume'.

# Preprocess the data
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
data['Returns'] = data['Close'].pct_change()
data.dropna(inplace=True)

# Define the feature(s) and target variable
# For simplicity, we'll use only one feature ('Returns') to predict 'Close' prices.
X = data[['Returns']].values
y = data['Close'].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Plot actual vs. predicted prices
plt.figure(figsize=(12, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Prices')
plt.plot(X_test, y_pred, color='red', linewidth=3, label='Predicted Prices')
plt.legend()
plt.title('Stock Price Prediction Using Linear Regression')
plt.xlabel('Returns')
plt.ylabel('Close Price')
plt.show()

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Absolute Error: {mae:.2f}')
print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared: {r2:.2f}')
```

Here we have used the Linear Regression model to predict the value of a stock .
Here at first we have imported all the required modules and also taken the dataset of a stock called TAP from S&P500 after which the training and testing of the model is done and at the end a plot is been made for the model.

OUTPUT:



Mean Absolute Error: 14.67
Mean Squared Error: 300.32
R-squared: -0.01

ACCURACY CHECKING:

When evaluating a stock market price prediction model, several metrics are commonly used to check its accuracy and performance.Here are some commonly used metrics for evaluating stock price prediction models:

1. Regression Metrics:
   - Mean Absolute Error (MAE):This metric measures the average absolute difference between the predicted and actual stock prices. It is less sensitive to outliers.
   - Mean Squared Error (MSE): MSE measures the average of the squared differences between predicted and actual prices. It penalizes larger errors more heavily than MAE.
   - Root Mean Squared Error (RMSE): RMSE is the square root of MSE and provides a more interpretable error metric in the same units as the target variable.

2.Profit and Loss Metrics (for trading strategies):
   - Profit and Loss (P&L): Calculate the actual profit or loss generated by trading based on the model's predictions. This metric is the most relevant if you are using the model for actual trading.
   - Sharpe Ratio: Measures the risk-adjusted return of a trading strategy. It considers both the returns and the volatility of the strategy.

3. Other Metrics (for risk assessment):
   - Volatility Metrics: Evaluate the model's ability to predict stock price volatility.
   - Risk Metrics: Assess the model's risk by considering downside risk measures like Value at Risk (VaR) or Conditional Value at Risk (CVaR).

The above strategy can be used for creating a more complex and accurate model.

This is how the project can be approached to produce a fruitful useful and meaningful output.