Predicting Nvidia stock price using Machine Learning

1. Introduction

As a technology enthusiasts and investors, we are keenly interested in understanding and forecasting the behavior of stock prices. Stock price movements reflect a complex interplay of market sentiments, company performance, economic indicators and global events.

Machine Learning (ML) algorithms has changed financial forecasts by enabling the extraction of intricate relationships from historical stock data that traditional methods often overlook. Large institutional investors leverage these techniques to achieve highly accurate stock price predictions. It is important to note that ML models cannot predict global events such as geopolitical tensions, pandemics or regulatory shifts. Therefore, we can't just rely on calculations, we must also monitor what is happening in the real world.

In this project, we aim to try different machine learning models to predict Nvidia's stock price by using historical market data starting from 2018. We chose Nvidia due to its rapid growth, its central role in emerging technologies such as artificial intelligence and high-performance computing.

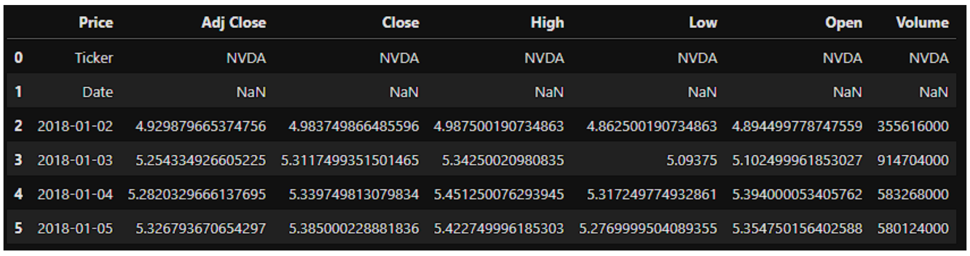
The report is organized as follows:

- Section 1 introduces the motivation and structure of the project.
- Section 2 details the dataset, features and prediction objectives.
- Section 3 discusses the machine learning methods and data processing techniques implemented.
- Section 4 presents experimental results and model evaluations.
- Section 5 concludes with reflections.

2. Problem Formulation

We model next day price forecasting for Nvidia stock (NVDA) as a supervised regression problem. For each trading day t, using only information available up to and including t, we predict the adjusted closing price at t + 1. Thus, each data point is one trading day.

The dataset used in this study is the "Nvidia Stock Data" from Kaggle [1]. It spans from 2.1.2018 to 30.9.2024 and contains 1697 daily observations. Each observation provides six numerical OHLCV variables: Open, High, Low, Close, Adjusted Close, and Volume. The label is the next day's adjusted closing price.



	Price	Adj Close	Close	High	Low	Open	Volume
0	Ticker	NVDA	NVDA	NVDA	NVDA	NVDA	NVDA
1	Date	NaN	NaN	NaN	NaN	NaN	NaN
2	2018-01-02	4.929879665374756	4.983749866485596	4.987500190734863	4.862500190734863	4.894499778747559	355616000
3	2018-01-03	5.254334926605225	5.3117499351501465	5.34250020980835	5.09375	5.102499961853027	914704000
4	2018-01-04	5.2820329666137695	5.339749813079834	5.451250076293945	5.317249774932861	5.394000053405762	583268000
5	2018-01-05	5.326793670654297	5.385000228881836	5.422749996185303	5.2769999504089355	5.354750156402588	580124000

Figure 1: Presentation of the raw data

We will be using Open, High, Low, Adjusted Close, Volume and two moving averages (7-days and 30-days) as features, since moving averages help capture short-term and medium-term trends in price movements. We chose to go with the adjusted closing price instead of the closing price because it accounts for splits and dividends. It's also more comparable over long periods. The other features correlate greatly with each other. All features and the label are continuous and numerical.

3. Methods

3.1. Preparing data

We selected the adjusted closing price as the target variable, while the other features are used as explanatory variables except the closing price. To prepare the dataset, we first fix the header by renaming "Price" to "Date". Then we remove two rows ("Ticker", "Date") because they contain zero valuable information as shown in Figure 1. After that we standardize column names to be more usable. Then convert Date to a timestamp which allows us to sort the table in ascending chronological order. Our target is the adjusted close, so we drop the unadjusted Close column because we don't need it. In addition, we compute 7-day and 30-day moving averages of the adjusted close, which serve as additional explanatory variables to represent short- and mid-term price momentum. In feature selection, we also used a heatmap.

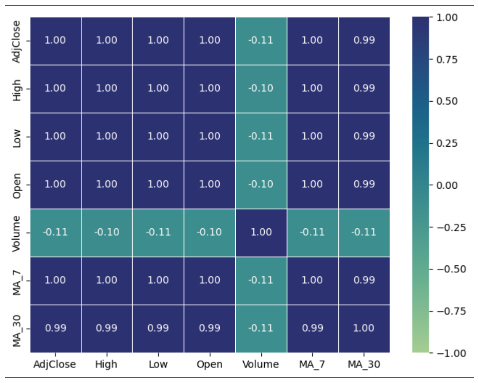
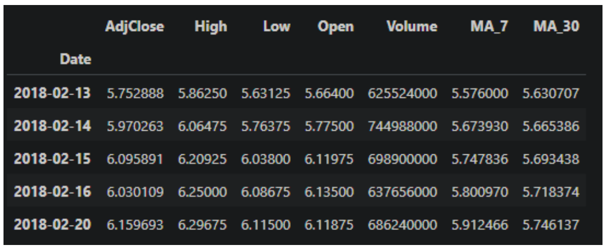


Figure 2: Heatmap

Next, we make the OHLCV columns available at time t. Then remove all rows with missing values in these required fields. The label is constructed by shifting the adjusted close with one trading day. After that the rows that don't have a subsequent label are removed. All inputs can then reflect information that is known at the end of day t. The raw file has 1697 rows and after cleaning the dataset it contains 1668 usable data points.



	AdjClose	High	Low	Open	Volume	MA_7	MA_30
Date							
2018-02-13	5.752888	5.86250	5.63125	5.66400	625524000	5.576000	5.630707
2018-02-14	5.970263	6.06475	5.76375	5.77500	744988000	5.673930	5.665386
2018-02-15	6.095891	6.20925	6.03800	6.11975	698900000	5.747836	5.693438
2018-02-16	6.030109	6.25000	6.08675	6.13500	637656000	5.800970	5.718374
2018-02-20	6.159693	6.29675	6.11500	6.11875	686240000	5.912466	5.746137

Figure 3: Presentation of engineered data

3.2. Methods

3.2.1 Linear regression

For our first supervised machine learning algorithm, we selected Linear Regression. This model assumes a linear hypothesis space and works by measuring the relationship between the target variable (stock price) and a set of explanatory variables such as previous day's adjusted closing price and trading volume plus an error term [2, 3, 4]. By modeling the stock price as a weighted sum of these features, linear regression tries to capture historical dependencies that might help predict future values.

The general form of the linear regression model is:

$\hat{y}_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \cdots + \beta_n x_{n,t} + \epsilon_t$

where: $\hat{y}_t$ is the predicted stock price, β_n are the model coefficients, $x_{n,t}$ are the input features, ϵ_t is the error term.

To train the model, coefficients should be estimated using the ordinary least squares method (OLS), which minimizes the total squared difference between actual and predicted values. The corresponding loss function is the mean squared error (MSE):

$\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2$, where y_t is the actual stock price and $\hat{y}_t$ is the predicted stock price.

The MSE penalizes large errors more heavily, making it particularly suitable for financial forecasting where large deviations from the true price can be costly.

We selected linear regression as first method because it is easy to read the values, easy to calculate and supported in most machine learning libraries. The learned coefficients make it possible to directly explain the effect of each feature on the predicted stock price. Because stock price movements are complex and often nonlinear, this method may have some limitations.

3.2.2 Random Forest

For the second supervised method we chose the Random Forest regressor. Random Forest is an ensemble learning technique that combines the predictions of multiple decision trees to improve generalization and robustness. Each decision tree is built and trained from different subsets of the data and makes an independent prediction. The final output is determined by averaging or taking a weighted average of all the trees' predictions [5, 6].

Formally, given B regression trees $\{T_b\}_{b=1}^{B}$, each trained on a different bootstrap sample, the Random Forest prediction for input vector x_t is: $\hat{y}_t = \frac{1}{B}\sum_{b=1}^{B} T_b(x_t)$ [7].

This approach reduces variance compared to a single decision tree and makes the model more robust to overfitting, which is particularly useful in noisy domains such as financial markets. Random forest can also capture nonlinear relationships between features and the target variable, something that linear regression cannot model effectively.

Random forest minimizes the mean squared error (MSE) when constructing the trees. This loss function is appropriate because our prediction target (next day's adjusted closing price) is a continuous numerical variable. MSE penalizes larger errors more heavily, which is useful in financial forecasting where large deviations can be particularly costly.

We selected Random Forest as our second method because stock prices are influenced by complex and possible nonlinear interactions between different variables (trading volume, daily high/low ranges and prior prices). In addition, Random Forest provides feature importance measures, which allow us to assess which input variables contribute most to prediction.

3.3. Model validation

To evaluate our models, we split the dataset into three sets: training, validation and test sets. Since this is time series, we used chronological splits: Training set: 60% of data, validation set: 20% of data, test set: 20% of data. This setup ensures that the models are trained on historical data, tuned with more recent data and evaluated with the most recent data to simulate a real forecasting scenario. With exact sizes, the split is: train = 1000, validation = 334 and test = 334, in sequence by date.

For evaluation, we use mean squared error (MSE) and root mean squared error (RMSE) to assess numerical prediction accuracy as well as visual comparisons of predicted and actual price trends. Since there is only data until 2024, we can compare our results to actual prices after.

4. Results

The Random Forest regression model and Linear regression model were trained to predict NVIDIA's adjusted closing price using historical market data from 2018 to 2024. The dataset was divided chronologically into training (60%), validation (20%), and test (20%) sets to ensure a realistic forward-looking evaluation. The features included Open, High, Low, Volume and two moving averages (7-day and 30-day), designed to capture both short-term fluctuations and longer-term price trends.

Model accuracy was evaluated using the MSE and RMSE. The Random Forest achieved a training RMSE of 0.085, a validation RMSE of 0.927, and a test RMSE of 52.37. The small training error compared to the very high-test error indicates that the model failed to generalize to unseen data. In particular, the predictions in the test period remained nearly constant around a single value, as shown in Figure 4. This suggests that the model suffered from underfitting and it was unable to capture the strong upward trend in NVIDIA's stock price during 2023–2024. The feature set, although including moving averages, did not provide enough temporal context for the Random Forest to learn evolving market patterns.

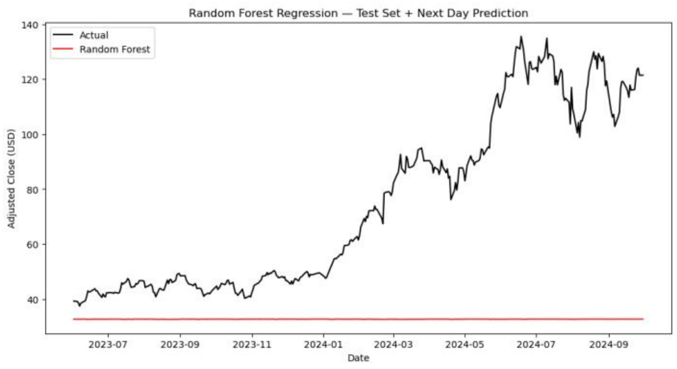


Figure 4: Random Forest prediction graph vs Actual price graph

Despite the poor dynamic fit, the model still produced a next-day forecast based on the latest available data. The predicted adjusted closing price (Figure 5) for the next trading day after 30 September 2024 was 119.60 USD.

```
Prediction for the next day:  119.60314701080323
```

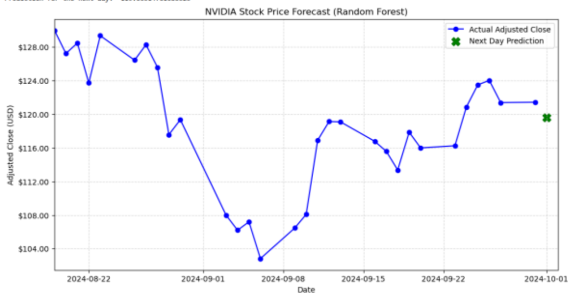


Figure 5: Random Forest price forecast

The feature importance analysis (Figure 6) revealed that the recent price levels were the most influential predictors. However, since all features were derived from price data alone, the model could not adapt to rapid structural changes or long-term growth trends. These results highlight the limitations of tree-based ensemble models for time-series data without lag features or external market indicators.

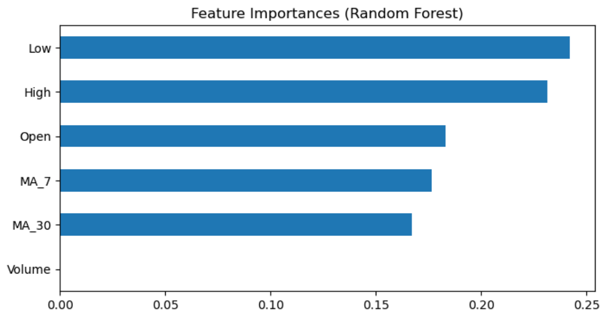


Figure 6: Feature importances in Random Forest

As for the Linear Regression it achieved Training MSE: 0.024 and RMSE: 0.154, Validation MSE: 0.073 and RMSE: 0.270 and Test MSE: 0.949, RMSE: 0.974. Training RMSE is clearly lower than the validation and test errors. The model tracks the price reasonably well on validation, but errors increase in the high-volatility test region. Compared to Random Forest the errors were much smaller, which indicates that the Linear Regression model has more stable generalization beyond the training window. However, the error difference between training and test set errors could be caused by the rapid stock growth during 2023-2024.

The training and validation/test gap (0.154 and 0.974) reflect either overfitting to the training window or distributional shift between periods which is common in markets, or it could be both. The training set overall was steady compared to test and validation sets.

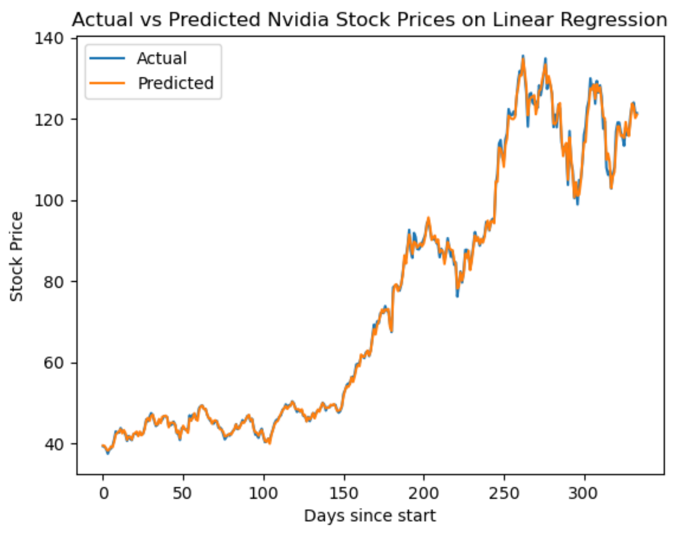


Figure 8: Actual vs Predicted Stock Prices on Linear Regression

By looking at the graph, the linear regression model tracks the price level closely across the test window. Errors cluster during the high-volatility regime at around day 200-330. There is mild under shooting at peaks and over shooting at lows.

We retrained the linear model on the full history up to the last available date 30.9.2024 on the dataset and produced a forecast for 1.10.2024. The predicted price for the next day is 121.11. This fit is kept separate from evaluation to avoid data leakage in the earlier reported metrics. We checked the true adjusted price for 1.10.2024 and it was [8]116.97. So, the predicted price was 3,5% larger than it actually was. It predicted the price to go up but instead it went down compared to 30.9.2024 price. What can be seen on this graph.

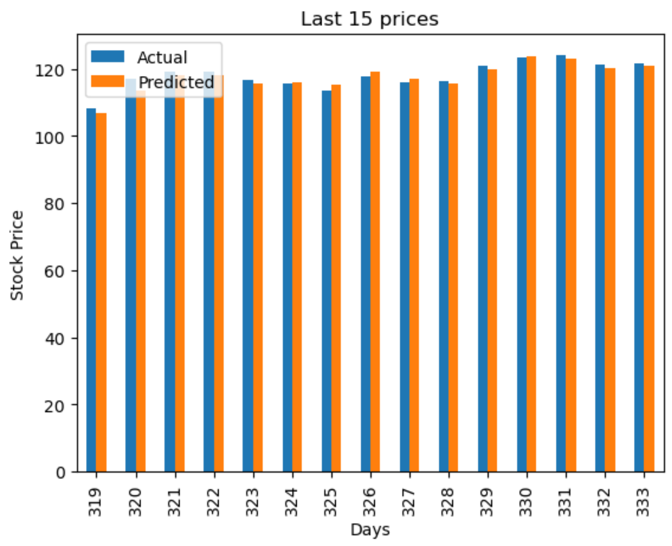


Figure 9: Last 15 actual and predicted prices on Linear Regression

5. Conclusions

This project applied Linear regression and Random Forest to forecast NVIDIA's next day adjusted closing price using stock price data from 2018 to 2024. Both models were evaluated with MSE and RMSE on chronologically split datasets. The Random Forest model underperformed due to weak generalization and inability to capture long-term trends, while Linear Regression achieved more consistent results and was selected as the final method. The results indicate that the problem is partially solved, but further improvement could come from adding lag features or external indicators to capture temporal dependencies and broader market influence.

References

[1] Dataset https://www.kaggle.com/datasets/muhammaddawood42/nvidia-stock-data

[2] IBM, What is Linear Regression? https://www.ibm.com/think/topics/linear-regression

[3] Statistics Solutions, What is Linear Regression? https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-linear-regression/

[4] Linear Regression in Machine Learning https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/

[5] IBM, What is random forest? https://www.ibm.com/think/topics/random-forest

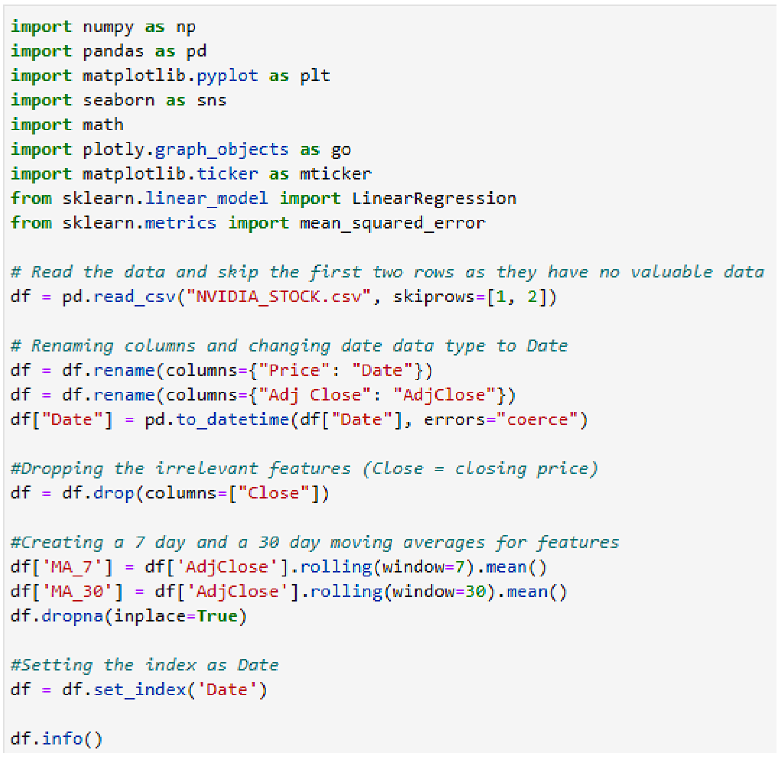
[6] GeeksforGeeks, Random Forest Regression in Python https://www.geeksforgeeks.org/machine-learning/random-forest-regression-in-python/

[7] Random forest https://en.wikipedia.org/wiki/Random_forest

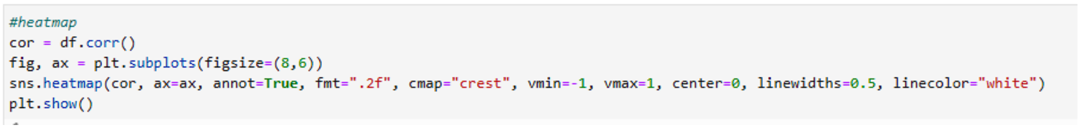
[8] Nvidia Stock Price
https://finance.yahoo.com/quote/NVDA/history/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAC-9H-I2MwJ5Fdc5h0JgNqGYczf9Jt8O7qjYq4f9RGISWFGsn551cYgSrcmb9G__0HXtlWVhjsP9qYuJyo_pMHakUMcP8HpKRpwnoGZ1TKBkRkaM1omNcAHReNNOm8xJ9xFvt4-pwT2DXVpZgImVEgQRHGW3tw8-RSCuBKv0RtyZ&period1=1727740800&period2=1728000000

[9] Linear Regression for Stock Market Prediction https://medium.com/@amit25173/linear-regression-for-stock-market-prediction-6039f1ea5c1b

Appendix



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
import plotly.graph_objects as go
import matplotlib.ticker as mticker
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Read the data and skip the first two rows as they have no valuable data
df = pd.read_csv("NVIDIA_STOCK.csv", skiprows=[1, 2])

# Renaming columns and changing date data type to Date
df = df.rename(columns={"Price": "Date"})
df = df.rename(columns={"Adj Close": "AdjClose"})
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")

#Dropping the irrelevant features (Close = closing price)
df = df.drop(columns=["Close"])

#Creating a 7 day and a 30 day moving averages for features
df['MA_7'] = df['AdjClose'].rolling(window=7).mean()
df['MA_30'] = df['AdjClose'].rolling(window=30).mean()
df.dropna(inplace=True)

#Setting the index as Date
df = df.set_index('Date')

df.info()
```



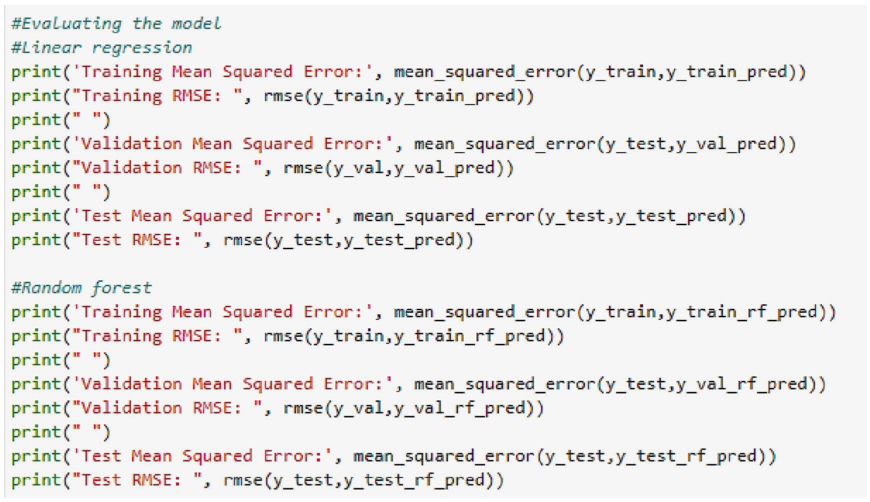
```
#heatmap
cor = df.corr()
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(cor, ax=ax, annot=True, fmt=".2f", cmap="crest", vmin=-1, vmax=1, center=0, linewidths=0.5, linecolor="white")
plt.show()
```

```python
#Visual of Closing price
df['AdjClose'].plot()
plt.title("Nvidia Stock Price", fontsize=13)
plt.ylabel('Price', fontsize=12)
plt.xlabel('Days', fontsize=12)
plt.grid(linewidth=0.5)
plt.show()
df.head()
```

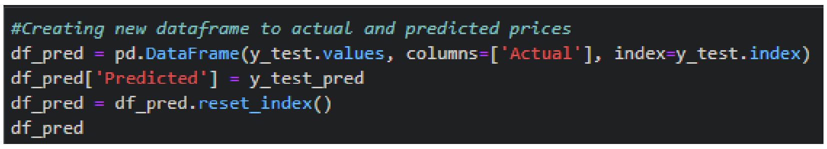
```python
#features and label
features = ["Open","High","Low","Volume","MA_7","MA_30"]
X = df[features]
y = df["AdjClose"]

#Splitting the data
n = len(df)
train_end = int(0.6 * n)
val_end = int(0.8 * n)

X_train, y_train = X.iloc[:train_end], y.iloc[:train_end]
X_val, y_val = X.iloc[train_end:val_end], y.iloc[train_end:val_end]
X_test, y_test = X.iloc[val_end:], y.iloc[val_end:]

# Shape for train data and validation data
print('Train:', X_train.shape)
print('Validation:', X_val.shape)
print('Test:', X_test.shape)
```

```
Train: (1000, 6)
Validation: (334, 6)
Test: (334, 6)
```

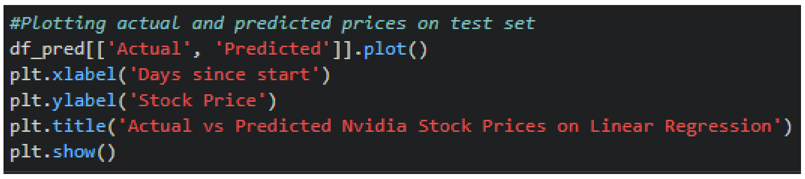
```python
#Training and fitting the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

#Train the Random forest model
rf = RandomForestRegressor(n_estimators = 200, random_state = 42, n_jobs = 1)
rf.fit(X_train, y_train)

#Creating RMSE
def rmse(y_true, y_pred):
    return float(np.sqrt(mean_squared_error(y_true, y_pred)))

# making predictions with linear regression
y_train_pred = model.predict(X_train)
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)

# making predictions with random forest
y_train_rf_pred = rf.predict(X_train)
y_val_rf_pred = rf.predict(X_val)
y_test_rf_pred = rf.predict(X_test)
```



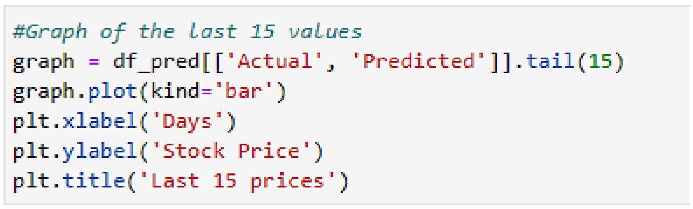
```
#Evaluating the model
#Linear regression
print('Training Mean Squared Error:', mean_squared_error(y_train,y_train_pred))
print("Training RMSE: ", rmse(y_train,y_train_pred))
print(" ")
print('Validation Mean Squared Error:', mean_squared_error(y_test,y_val_pred))
print("Validation RMSE: ", rmse(y_val,y_val_pred))
print(" ")
print('Test Mean Squared Error:', mean_squared_error(y_test,y_test_pred))
print("Test RMSE: ", rmse(y_test,y_test_pred))

#Random forest
print('Training Mean Squared Error:', mean_squared_error(y_train,y_train_rf_pred))
print("Training RMSE: ", rmse(y_train,y_train_rf_pred))
print(" ")
print('Validation Mean Squared Error:', mean_squared_error(y_test,y_val_rf_pred))
print("Validation RMSE: ", rmse(y_val,y_val_rf_pred))
print(" ")
print('Test Mean Squared Error:', mean_squared_error(y_test,y_test_rf_pred))
print("Test RMSE: ", rmse(y_test,y_test_rf_pred))
```



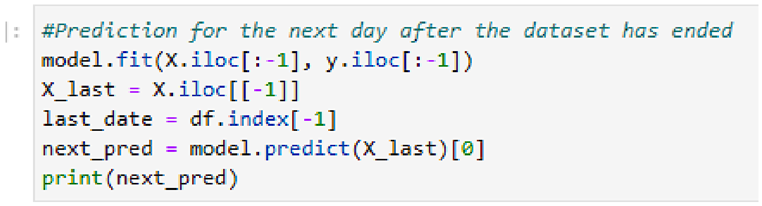
```
#Creating new dataframe to actual and predicted prices
df_pred = pd.DataFrame(y_test.values, columns=['Actual'], index=y_test.index)
df_pred['Predicted'] = y_test_pred
df_pred = df_pred.reset_index()
df_pred
```



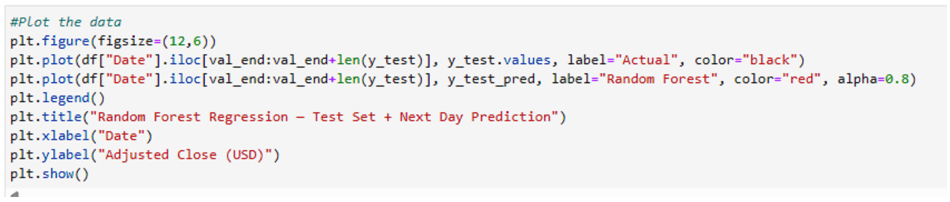
```
#Plotting actual and predicted prices on test set
df_pred[['Actual', 'Predicted']].plot()
plt.xlabel('Days since start')
plt.ylabel('Stock Price')
plt.title('Actual vs Predicted Nvidia Stock Prices on Linear Regression')
plt.show()
```



```
#Graph of the last 15 values
graph = df_pred[['Actual', 'Predicted']].tail(15)
graph.plot(kind='bar')
plt.xlabel('Days')
plt.ylabel('Stock Price')
plt.title('Last 15 prices')
```



```
#Prediction for the next day after the dataset has ended
model.fit(X.iloc[:-1], y.iloc[:-1])
X_last = X.iloc[[-1]]
last_date = df.index[-1]
next_pred = model.predict(X_last)[0]
print(next_pred)
```



```
#Plot the data
plt.figure(figsize=(12,6))
plt.plot(df["Date"].iloc[val_end:val_end+len(y_test)], y_test.values, label="Actual", color="black")
plt.plot(df["Date"].iloc[val_end:val_end+len(y_test)], y_test_pred, label="Random Forest", color="red", alpha=0.8)
plt.legend()
plt.title("Random Forest Regression – Test Set + Next Day Prediction")
plt.xlabel("Date")
plt.ylabel("Adjusted Close (USD)")
plt.show()
```

```python
#Calculate the prediction for the next day and plot it with actual prices
rf.fit(X.iloc[:-1], y.iloc[:-1])
X_last = X.iloc[[-1]]
last_date = df["Date"].iloc[-1]
next_pred = rf.predict(X_last)[0]
print("Prediction for the next day: ", next_pred)

next_day = last_date + pd.Timedelta(days=1)
plt.figure(figsize=(12,6))

lookback = 30
plt.plot(df["Date"].iloc[-lookback:], df["AdjClose"].iloc[-lookback:],
         "o-", color="blue", label="Actual Adjusted Close")

plt.scatter(next_day, next_pred, color="green", s=120, label="Next Day Prediction", marker="X")
plt.title("NVIDIA Stock Price Forecast (Random Forest)")
plt.xlabel("Date")
plt.ylabel("Adjusted Close (USD)")

ax = plt.gca()
ax.yaxis.set_major_locator(mticker.MaxNLocator(8))
ax.yaxis.set_major_formatter(mticker.StrMethodFormatter("${x:,.2f}"))

plt.grid(True, linestyle="--", alpha=0.5)
plt.legend()

plt.xlim(df["Date"].iloc[-lookback], next_day + pd.Timedelta(days=1))

plt.show()
```

```python
#Plot the data
plt.figure(figsize=(12,6))
plt.plot(df["Date"].iloc[val_end:val_end+len(y_test)], y_test.values, label="Actual", color="black")
plt.plot(df["Date"].iloc[val_end:val_end+len(y_test)], y_test_rf_pred, label="Random Forest", color="red", alpha=0.8)
plt.legend()
plt.title("Random Forest Regression – Test Set + Next Day Prediction")
plt.xlabel("Date")
plt.ylabel("Adjusted Close (USD)")
plt.show()
```