Project-1

According to the project, we should predict the stock price with three models. I have decided to predict the stock price of Nivida company and downloaded the data for yfinance form 2018 to 10th October 2024.Following are the models that I have selected for the prediction of the stock price.

1) LSTM model
2) GRU model
3) MPL model

LSTM model and GRU model are both Recurrent neural network. MLP model is FeedForward neural network. This model uses the supervised learning.

Step by step approach for building the models

While building the model there are common steps that needs to be followed. The common steps that needs to be followed are as below.

1) Install all the packages need for the model.

- LSTM model

```
[1]: !pip install numpy
     !pip install pandas
     !pip install matplotlib
     !pip install seaborn
     !pip install yfinance
     !pip install datetime
     !pip install tensorflow
     !pip install scikit-learn
```

```
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages (1.26.4)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.12/site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.12/site-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.21 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

-GRU model

```
[2]: !pip install numpy
     !pip install pandas
     !pip install matplotlib
     !pip install seaborn
     !pip install yfinance
     !pip install datetime
     !pip install tensorflow
     !pip install scikit-learn
```

```
4.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorboard<2.18,>=
2.17->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/anaconda3/lib/python3.12/site-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.
0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/anaconda3/lib/python3.12/site-packages (from werkzeug>=1.0.1->tensorboard<2.18,>=2.1
7->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in /opt/anaconda3/lib/python3.12/site-packages (from rich->keras>=3.2.0->tensorfl
ow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/anaconda3/lib/python3.12/site-packages (from rich->keras>=3.2.0->tensorflow) (
2.15.1)
Requirement already satisfied: mdurl~=0.1 in /opt/anaconda3/lib/python3.12/site-packages (from markdown-it-py<3.0.0,>=2.2.0->rich->keras>=3.
2.0->tensorflow) (0.1.0)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.12/site-packages (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-learn) (2.2.0)
```

- MPL model

[2]:
```
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install seaborn
!pip install yfinance
!pip install datetime
!pip install scikit-learn
```

```
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages (1.26.4)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.12/site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/python3.12/site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.12/site-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.21 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

2) After installation, import all the packages.

- LSTM mode

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as date
import yfinance as yf # helps to access the data from yahoo finance

# For data pre-processing (Normalization)
from sklearn.preprocessing import StandardScaler
# For spliting the dataset
from sklearn.model_selection import train_test_split
# Evaluating model
from sklearn.metrics import mean_squared_error, r2_score


# Api's for designing neural network model: LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
```

- GRU model

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as date
import yfinance as yf # helps to access the data from yahoo finance

# For data pre-processing (Normalization)
from sklearn.preprocessing import StandardScaler
# For spliting the dataset
from sklearn.model_selection import train_test_split
# Evaluating model
from sklearn.metrics import mean_squared_error, r2_score


# Api's for designing neural network model: LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
```

- MPL model

[90]:
```
# Import all the packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as date
import yfinance as yf # helps to access the data from yahoo finance

# For data pre-processing (Normalization)
from sklearn.preprocessing import StandardScaler
# For spliting the dataset
from sklearn.model_selection import train_test_split
# MLP model(FNN)
from sklearn.neural_network import MLPRegressor
# Evaluating model
from sklearn.metrics import mean_squared_error, r2_score
```

3) Provide the dataset and read it.

[3]:
```
# Downloading the data
company_name = 'NVDA'
start_date = date.datetime(2018,1,1)
end_date = date.datetime(2024,10,7)
nivida_stock_data = yf.download(company_name, start=start_date, end=end_date)
# Converting index into column
nivida_stock_data = nivida_stock_data.reset_index()
```

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

[4]:
```
# Looking into dataset
nivida_stock_data.head()
```

[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-01-02	4.89450	4.98750	4.86250	4.98375	4.930221	355616000
1	2018-01-03	5.10250	5.34250	5.09375	5.31175	5.254699	914704000
2	2018-01-04	5.39400	5.45125	5.31725	5.33975	5.282397	583268000
3	2018-01-05	5.35475	5.42275	5.27700	5.38500	5.327161	580124000
4	2018-01-08	5.51000	5.62500	5.46450	5.55000	5.490389	881216000

4) Analyze the data structure of the dataset.

[5]:
```
# Analyze the data structure
nivida_stock_data.info()
nivida_stock_data.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1701 entries, 0 to 1700
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1701 non-null   datetime64[ns]
 1   Open       1701 non-null   float64
 2   High       1701 non-null   float64
 3   Low        1701 non-null   float64
 4   Close      1701 non-null   float64
 5   Adj Close  1701 non-null   float64
 6   Volume     1701 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 93.2 KB
```

[5]:
```
(1701, 7)
```

5) Verify the dataset for any missing or duplicate rows. Ensure that all data is properly organized and free from errors to create a clean and reliable dataset.

[6]:
```
# Checking the missing values
nivida_stock_data.isnull().sum()
```

[6]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

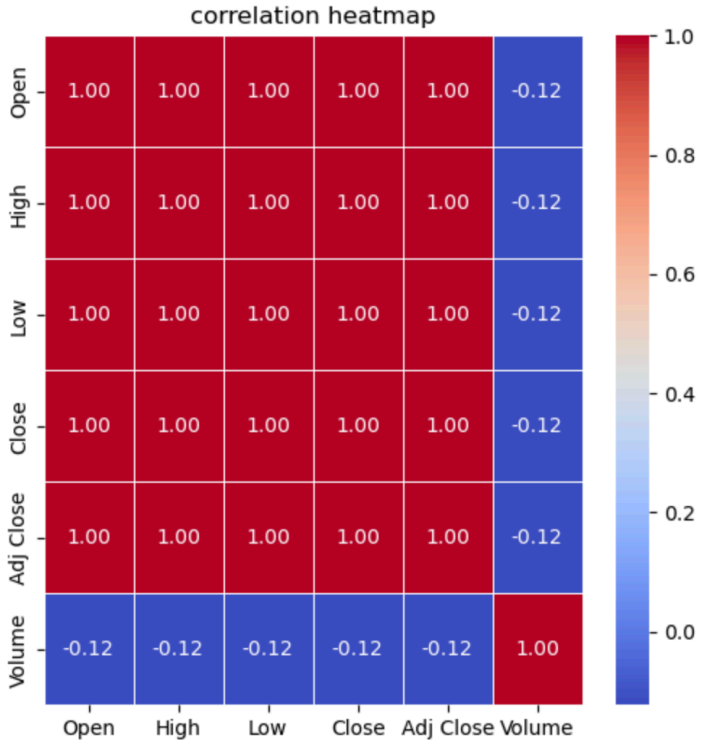
[7]:
```
# Checking for redundent values
nivida_stock_data.duplicated().sum()
```

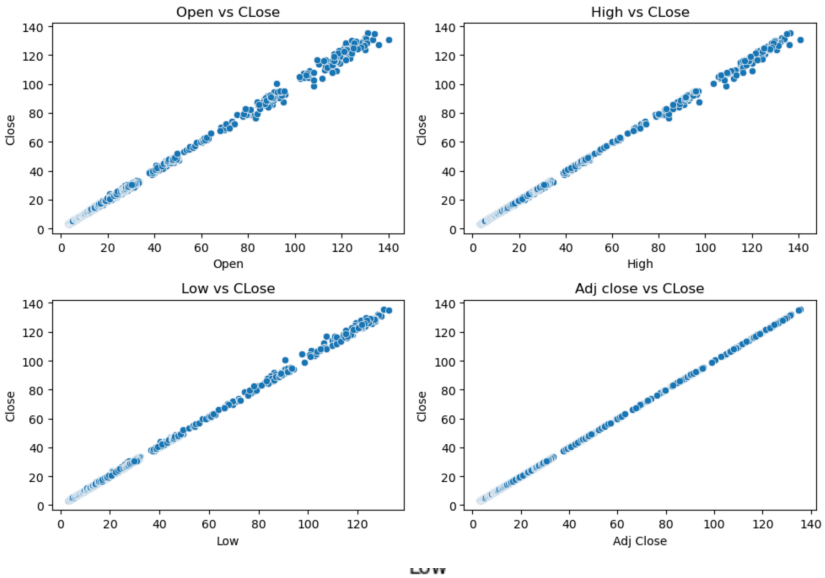
[7]:
```
0
```

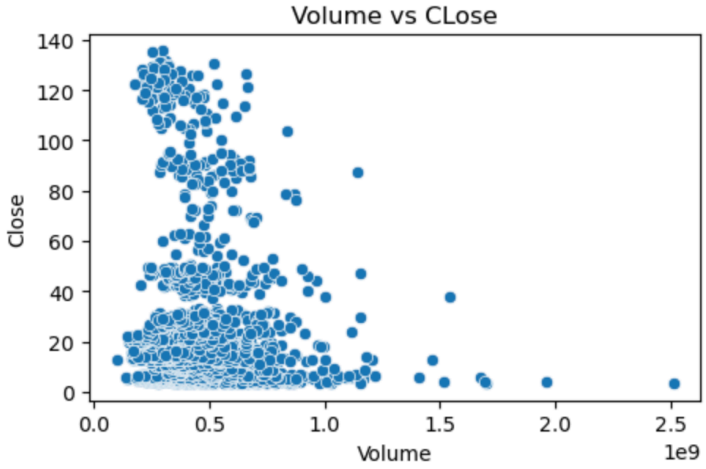
6) Once the data is cleaned, perform Exploratory Data Analysis (EDA) on the dataset. This will help identify which features have a strong or weak correlation with the target variable. We utilized scatter plots and heatmaps to conduct the EDA.

[44]:
```
# Performing the EDA(Exploratory data analysis)
# Analyzing the relationship between each feature variables and target variable visually
# Plotting the scatter plot
plt.figure(figsize = (10,10)) # declaring the grid size
# Relationship between Open vs CLose
plt.subplot(3,2,1)
sns.scatterplot(x = 'Open', y = 'Close', data = nivida_stock_data)
plt.title('Open vs CLose')
# Relationship between High vs CLose
plt.subplot(3,2,2)
sns.scatterplot(x = 'High', y = 'Close', data = nivida_stock_data )
plt.title('High vs CLose')
# Relationship between Low vs CLose
plt.subplot(3,2,3)
sns.scatterplot(x = 'Low', y = 'Close', data = nivida_stock_data  )
plt.title('Low vs CLose')
# Relationship between Adj Close vs CLose
plt.subplot(3,2,4)
sns.scatterplot(x = 'Adj Close', y = 'Close', data = nivida_stock_data  )
plt.title('Adj close vs CLose')
# Relationship between Volume vs CLose
plt.subplot(3,2,5)
sns.scatterplot(x = 'Volume', y = 'Close', data = nivida_stock_data  )
plt.title('Volume vs CLose')
# Managing the grid
plt.tight_layout()
plt.show()
```

[45]:
```
# Analyzing the heatmap
# Calculating the correlation matrix
correlation_matrix = nivida_stock_data.drop(columns = ['Date']).corr()
# Allocating the grid size
plt.figure(figsize = (6,6))
# Displaying the heatmap
sns.heatmap(correlation_matrix, annot = True, cmap = 'coolwarm', fmt ='.2f', linewidth = 0.5)
# Declaring the tittle
plt.title('correlation heatmap')
# Displaying the graph
plt.show()
```




Open vs CLose
Close
Open
High vs CLose
Close
High
Low vs CLose
Close
Low
Adj close vs CLose
Close
Adj Close


Volume vs CLose
Close
Volume
1e9

7) Following EDA, it was observed that the Volume feature has a weak correlation with the target variable. All other features showed a strong relationship.Therefore, the Volume feature will be excluded from the model. Both the feature variables and target variable need to be properly labeled.

[62]:
```
# Declaring the feature variables and target variables
feature_variables_data = nivida_stock_data.drop(columns = [ 'Close','Date','Volume'])
target_variables_data = nivida_stock_data['Close']
feature_variables_data.head()
```

[62]:

	Open	High	Low	Adj Close
0	4.89450	4.98750	4.86250	4.930221
1	5.10250	5.34250	5.09375	5.254699
2	5.39400	5.45125	5.31725	5.282397
3	5.35475	5.42275	5.27700	5.327161
4	5.51000	5.62500	5.46450	5.490389

8) Its time to split date set into training and testing. We will allocate 80 percent of data for training and 20 percent of data fro testing.

[50]:
```
# Spliting data into training and testing
x_train, x_test, y_train, y_test = train_test_split(feature_variables_data, target_variables_data, test_size = 0.20)
```

9) After splitting the data, it time to normalize the data. We are using standardization method for normalization.

[52]:
```
# Normalizing the dataset (standarization)
# Initialing the scaler
scaler = StandardScaler()
# Scalering the inputs data (features variables) of training
x_train_normalize = scaler.fit_transform(x_train)
# Scalering the inputs data (features variables) of testing
x_test_normalize = scaler.fit_transform(x_test)

# Scalering the inputs data (target variables) of training
y_train_normalize = scaler.fit_transform(y_train.values.reshape(-1, 1))
print(y_train_normalize.shape)
# Scalering the inputs data (target variables) of testing
y_test_normalize = scaler.fit_transform(y_test.values.reshape(-1, 1))
print(y_test_normalize.shape)
```

```
(1360, 1)
(341, 1)
```

This are the common steps that needs to be done while making models. Now, its time to design the neural network.

LSTM model

- LSTM model takes 3D inputs. (Samples, Tlmesteps, Feature)
- Needs to convert inputs to LSTM model data structure. We are setting timesteps to 7. This means that our model will observe for seven days and update it.

- It's time to design the neural network architecture. The network will have five hidden layers with 120, 120, 90, 90, and 90 neuron's, followed by a single output neurone. Regularization techniques will be applied to prevent overfitting. The Adam optimizer will be utilized for optimization, and the loss function will be Mean Squared Error (MSE).

```
]: # Designing the model with LSTM layer
model = Sequential()
# Initializing Input Shape
model.add(Input(shape=(time_steps,x_test_normalize.shape[1])))
# Add first hiden layer
model.add(LSTM(units = 120, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add second hiden layer
model.add(LSTM(units = 120, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add third hiden layer
model.add(LSTM(units = 90, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add forth hiden layer
model.add(LSTM(units = 90, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add fifth hiden layer
model.add(LSTM(units = 90, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add output layer
model.add(Dense(units = 1))
# Add optimizer and loss calculation
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

- Training the model.

```
# Training the model
model.fit(conv_x_train_normalize, conv_y_train_normalize, epochs = 100, batch_size = 30)
```

```
Epoch 1/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 1.1139
Epoch 2/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1.0408
Epoch 3/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 1.0536
Epoch 4/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.9589
Epoch 5/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.9324
Epoch 6/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 1.0439
Epoch 7/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.9445
Epoch 8/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 1.0323
Epoch 9/100
46/46 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 1.1472
```

- Predict the model with test dataset.

[19]:
```
# Predicting with test dataset
prediction = model.predict(conv_x_test_normalize)
# Take only the last timestep's prediction from each sample
prediction_last_timestep = prediction[:, -1, :]

# Removing the Normalizing form target values (predication values)
prediction_stock_price = scaler.inverse_transform(prediction_last_timestep)

# Removing the Normalizing form target values (actual values)
actual_stock_price = scaler.inverse_transform(conv_y_test_normalize)
# Displaying the prediction value
print(prediction_stock_price)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
[[23.0609  ]
 [23.183306]
 [23.17552 ]
 [23.015347]
 [23.007225]
 [23.606663]
 [25.506586]
 [22.7242  ]
 [22.613426]
 [22.532825]
 [22.696684]
 [22.804276]
 [22.92827 ]
 [22.906809]
 [22.954416]
 [22.840889]
 [23.333527]
```

- Evaluate the model with Mean square error and R- square score

[20]:
```
# Evaluating the model
# Mean Squared Error
MS_Error = mean_squared_error(actual_stock_price, prediction_stock_price)
print(f'Mean Squared Error: {MS_Error}')
# R square score
r2 = r2_score(actual_stock_price, prediction_stock_price)
print(f'R-squared score: {r2}')
```

```
Mean Squared Error: 812.904548258556
R-squared score: -0.0027010771398203737
```

GRU model

- GRU model takes 3D inputs. (Samples, Tlmesteps, Feature)
- Needs to convert inputs to LSTM model data structure. We are setting timesteps to 7. This means that our model will observe for seven days and update it.

[15]:
```
# Converting inputs into GRU data structure( GRU takes 3D as input(samples , timesteps, features)
def convert_to_LSTM_data_structure(features, target, time_steps):
    F , T = [], []
    for i in range(len(features) - time_steps ):
        F.append(features[i:(i + time_steps)])
        T.append(target[i + time_steps])
    return np.array(F), np.array(T)

# Set the number of timesteps
time_steps = 7  # one week observation

# Creating the training dataset for GRU
conv_x_train_normalize, conv_y_train_normalize = convert_to_LSTM_data_structure(x_train_normalize, y_train_normalize, time_steps)
print(conv_x_train_normalize.shape)
print(conv_y_train_normalize.shape)
# Creating the testing dataset for GRU
conv_x_test_normalize, conv_y_test_normalize = convert_to_LSTM_data_structure(x_test_normalize, y_test_normalize, time_steps)
print(conv_y_test_normalize.shape)
```

```
(1353, 7, 4)
(1353, 1)
(334, 1)
```

- It's time to design the neural network architecture. The network will have five hidden layers with 120, 120, 90, 90, and 90 neuron's, followed by a single output neurone. Regularization techniques will be applied to prevent overfitting. The Adam optimizer will be utilized for optimization, and the loss function will be Mean Squared Error (MSE).

[16]:
```
# Designing the model with LSTM layer
model = Sequential()
# Initializing Input Shape
model.add(Input(shape=(time_steps,x_test_normalize.shape[1])))
# Add first hiden layer
model.add(GRU(units = 120, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add second hiden layer
model.add(GRU(units = 120, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add third hiden layer
model.add(GRU(units = 90, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add forth hiden layer
model.add(GRU(units = 90, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add fifth hiden layer
model.add(GRU(units = 90, return_sequences = True))
# Add Regularization to avoid overfitting
model.add(Dropout(0.35))
# Add output layer
model.add(Dense(units = 1))
# Add optimizer and loss calculation
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

- Training the model.

```
# Training the model
model.fit(conv_x_train_normalize, conv_y_train_normalize, epochs = 30, batch_size = 5)
```

```
Epoch 1/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 1.0004
Epoch 2/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.9972
Epoch 3/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1.0082
Epoch 4/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1.0624
Epoch 5/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1.0074
Epoch 6/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1.0076
Epoch 7/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 1.0459
Epoch 8/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.9837
Epoch 9/30
271/271 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.9390
```

- Predict the model with test dataset.

[17]:
```
# Predicting with test dataset
prediction = model.predict(conv_x_test_normalize)
# Take only the last timestep's prediction from each sample
prediction_last_timestep = prediction[:, -1, :]

# Removing the Normalizing form target values (predication values)
prediction_stock_price = scaler.inverse_transform(prediction_last_timestep)

# Removing the Normalizing form target values (actual values)
actual_stock_price = scaler.inverse_transform(conv_y_test_normalize)
# Displaying the prediction value
print(prediction_stock_price)
print(actual_stock_price)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
[[26.628576]
 [26.623716]
 [26.620804]
 [26.620918]
 [26.62461 ]
 [26.629114]
 [26.63165 ]
 [26.631794]
 [26.629036]
 [26.627428]
 [26.627617]
 [26.629192]
 [26.631903]
 [26.633968]
 [26.636427]
 [26.637545]
 [26.638548]
```

- Evaluate the model with Mean square error and R- square score

[18]:
```
# Evaluating the model
# Mean Squared Error
MS_Error = mean_squared_error(actual_stock_price, prediction_stock_price)
print(f'Mean Squared Error: {MS_Error}')
# R square score
r2 = r2_score(actual_stock_price, prediction_stock_price)
print(f'R-squared score: {r2}')
```

```
Mean Squared Error: 924.0809869515657
R-squared score: -0.0006035364735652582
```

MLP model

- It's time to design the neural network architecture. The network will have five hidden layers with 14, 12, 10, 7, and 3 neuron's. The Adam optimizer will be utilized for optimization, 'tanh' activation function is used and 10,000 loops has been used.

[80]:
```
# Using MLP: Multilayer Perceptron by sklearn
mlp = MLPRegressor(solver='adam', alpha = 1e-2, activation = 'tanh', hidden_layer_sizes = (14,12,10,7,3), max_iter= 10000)
```

- Training the model

[82]:
```
# Training the model
mlp.fit(x_train_normalize , y_train_numpy)
```

[82]:
```
MLPRegressor(activation='tanh', alpha=0.01,
             hidden_layer_sizes=(14, 12, 10, 7, 3), max_iter=10000)
```

- Predict the model by test dataset.

[84]:
```
# Prediction with test data
predictions = mlp.predict(x_test_normalize)
# Display the prediction
print(predictions)
```

```
[ 16.75260948 106.99332912   3.93558284  11.95036099  15.05372424
  41.6404905    5.78377655  25.40790731   3.36855752  35.38689461
  12.21500293  54.11117127  26.29777655  27.54307275 111.38481648
  77.21083693  12.1713119    3.55445105   4.37488097  85.7765518
  39.12687951  10.89448186  75.2571769   82.81401863   8.24831988
  65.79306701  18.40500709   6.52076603 113.17106924   3.34063273
   4.74863744   3.97848398   4.37022016   6.68959656 106.60979704
   3.43891241   5.7691779   23.31228945  40.6521857    4.81372306
  12.28886174  14.25019934  13.00799035  20.61280564  98.6999585
   6.39050747  14.11640521  35.21166924  20.71391227  80.6245697
   4.87303677  44.94349866   6.80744288  86.45241896  15.17862029
   3.50374416   8.01970806  17.58589999   5.14321118  14.4951174
   6.13953465  11.11378062  12.01222017  83.83336622   5.62443636
  21.21300018   4.89362275  12.17552985  11.8924238  117.34305633
  38.32240934  12.32936899   5.62234065   5.67808841  97.81276933
  20.11844102   4.17909998  17.49057419  15.60495252  20.17469889
  12.91719999   3.47222499  19.82608398  56.92043411  19.57330827
  18.60344134   6.53380244  18.53067688  13.9459099    3.67527703
```

- Evaluate the model with Mean square error and R- square score

[132]:
```
# Evaluating the model
# Mean Squared Error
MS_Error = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {MS_Error}')
# R square score
r2 = r2_score(y_test, predictions)
print(f'R-squared score: {r2}')
```

```
Mean Squared Error: 15.226752620564275
R-squared score: 0.9849963722156294
```

Drawing the conclusion:

After assessing all the models, we can conclude that the MLPRegression model has outperformed the others. This model can be trained quickly. I attempted to train other models with a higher number of iterations, which took a considerable amount of time, but the results showed gradual improvement.

Outperformed model from scratch with numpy

As other models this model has also common step that needs to be preformed which I have discussed above. Form splitting the dataset we are going to use numpy and form that point we are going to discuss below.

- Splitting the dataset into training(80%) and testing(20%)

[18]:
```
# Allocate the 80% of data for training
train_size = int(0.8*len(feature_variables_data))
feature_trained_data = feature_variables_data[:train_size]
feature_test_data  = feature_variables_data[train_size:]
target_trained_data = target_variables_data[:train_size]
target_test_data = target_variables_data[train_size:]

# Displaying the training and testing data
print(f'Training:{feature_trained_data .shape}, {target_trained_data.shape}')
print(f'Testing:{feature_test_data.shape}, {target_test_data.shape}')
```

```
Training:(1360, 4), (1360,)
Testing:(341, 4), (341,)
```

- Normalizing the data

[25]:
```python
# Method for Normalizing the data with standardization by numpy
def scaler(data):
    # Calculating the mean
    mean = np.mean(data)
    # Calculating the standard_deviation
    standard_deviation = np.std(data)
    # Furmula: convertion = (input - mean) diviaded by standard deviation
    normalized_data = (data - mean) / standard_deviation
    return normalized_data
```

[29]:
```python
# converstion
# Arguments are passed in numpy array
feature_trained_data_normalize = scaler(feature_trained_data.values)
target_trained_data_normalize = scaler(target_trained_data.values)
feature_test_data_normalize = scaler(feature_test_data.values)
target_test_data_normalize = scaler(target_test_data.values)
```

- Defining the activation function

96]:
```python
# Define the Activation function: we are using Sigmoid

# Define the Sigmoid activation function
def sigmoid(inputs):
    inputs = np.clip(inputs, -500, 500)  # Clipping the input range
    return 1 / (1 + np.exp(-inputs))

# Define the derivative of Sigmoid for backpropagation
def sigmoid_derivative(inputs):
    inputs = np.clip(inputs, -500, 500)  # Clipping the input range
    return inputs * (1 - inputs)
```

- Designing the MLP model

1252…

```python
# Designing the FNN(FeedFoward neural network) model
class MLP():
    # Constructor
    def __init__(self, input, output, hidden_layer_sizes):
        self.input_size = input.shape[1] # counting the number of features
        self.total_hidden_layer = len(hidden_layer_sizes)
        print(hidden_layer_sizes[0])
        # Converting the dimension
        self.output_size = output
        self.w = []
        self.b = []
        self.computed_h = []
        self.h = []
        self.error_hidden_h = []
        self.dw = []
        self.db = []

        #Initialize weights and biases
        for i in range(self.total_hidden_layer+1):
            if i == 0:
                self.w.append(np.random.rand(self.input_size, hidden_layer_sizes[i]))
                self.b.append(np.zeros((1, hidden_layer_sizes[i])))
            elif i == (self.total_hidden_layer):
                self.w.append(np.random.rand(hidden_layer_sizes[i-1], self.output_size))
                self.b.append(np.zeros((1, self.output_size )))
            else:
                self.w.append(np.random.rand(hidden_layer_sizes[i-1], hidden_layer_sizes[i]))
                self.b.append(np.zeros((1, hidden_layer_sizes[i])))
```

```python
    # Passing forward
    def fit(self, input):
        for i in range(self.total_hidden_layer+1):
            if i == (self.total_hidden_layer):
                #self.computed_output= np.dot(self.h[i], self.w[i]) + self.b[i]
                self.computed_h.append(np.dot(self.h[i-1], self.w[i]) + self.b[i])
                output = sigmoid(self.computed_h[i])
            elif i == 0:
                #self.computed_h[i] = np.dot(input, self.w[i]) + self.b[i]
                self.computed_h.append(np.dot(input, self.w[i]) + self.b[i])
                #self.h[i] = sigmoid(self.computed_h[i])
                self.h.append(sigmoid(self.computed_h[i]))
            else:
                #self.computed_h[i] = np.dot(self.h[i-1],self.w[i]) + self.b[i]
                self.computed_h.append(np.dot(self.h[i-1],self.w[i]) + self.b[i])
                #self.h[i] = sigmoid(self.computed_h[i])
                self.h.append(sigmoid(self.computed_h[i]))

        return output
```

```
    # Backward passing
    def backward(self, input, output, fw_output, learning_rate):

        # Converting the dimension of target variable
        reshape_output = output.reshape(output.size,1)
        # Calculate the error in the output
        output_error = fw_output - reshape_output
        j = 0
        # Calculate the error, delta weights and delta bais
        for i in reversed(range(self.total_hidden_layer+1)):

            if i == 0:
                self.dw.append(np.dot(input.T,self.error_hidden_h[i]))
                self.db.append(np.sum(self.error_hidden_h[i], axis=0, keepdims=True))

            elif i == (self.total_hidden_layer):
                self.dw.append(self.h[i-1].T.dot(output_error * sigmoid_derivative(self.computed_h[i])))
                self.db.append(np.sum(output_error * sigmoid_derivative(self.computed_h[i]), axis=0, keepdims=True))
                self.error_hidden_h.append(output_error.dot(self.w[i].T) * sigmoid_derivative(self.computed_h[i-1]) )

            else:
                self.dw.append(self.h[i-1].T.dot(self.error_hidden_h[j] * sigmoid_derivative(self.computed_h[i])))
                self.db.append(np.sum(self.error_hidden_h[j] * sigmoid_derivative(self.computed_h[i]), axis=0, keepdims=True))
                self.error_hidden_h.append(self.error_hidden_h[j].dot(self.w[i].T) * sigmoid_derivative(self.computed_h[i-1]))
                j +=1

    #    z = 0
        # Update weights and biases
     #   for i in reversed(range(self.total_hidden_layer+1)):
      #      if i == 0:
       #         print('-----')
        #        print(z)
         #       print(i)
          #      self.w[i] -= learning_rate * self.dw[z]
           #     self.b[i] -= learning_rate * self.db[z]

            # elif i == (self.total_hidden_layer):
             #    print(len(self.b))
              #   self.w[i] -= learning_rate * self.dw[z]
               #  print('one')
                # self.b[i] -= learning_rate * self.db[z]
                # print('two')
             #else:
              #   print(z)
               #  self.w[i] -= learning_rate * self.dw[z]
               #  print('three')
               #  self.b[i] -= learning_rate * self.db[z]
             #z += 1
```

- Defining the training method

[17]:
```
# Defining the training method
def training_MLP(model, input, output, iteration, learning_rate):
    for iteration in range(iteration):
        # Forward passing
        fw_output = model.fit(input)
        # Backward passing and updating weights and bais
        model.backward(input, output, fw_output, learning_rate)

        if (iteration+1) % 100 == 0:
            loss = np.mean((output.reshape(output.size,1)- fw_output) ** 2)
            print(f'iteration {iteration+1}, Loss: {loss:.4f}')
```

- Initialing the model parameter

[44]:
```
# Initialing
hidden_layer_sizes = (14,12,10,7,3)
output= 1
model = MLP(feature_trained_data_normalize, output, hidden_layer_sizes)
```

- Training the model

[19]:
```
# Training the model
# Call the method
training_MLP(model, feature_trained_data_normalize, target_trained_data_normalize, 100, 0.01)
```

```
iteration 100, Loss: 1.6719
```

- Predict the model with test data

[20]:
```
# Evaluate the trained MLP with test data
predicted_outputs = model.fit(feature_test_data_normalize)
print(predicted_outputs)
```

```
[[0.81972147]
 [0.81972343]
 [0.81972456]
 ...
 [0.81975034]
 [0.81975034]
 [0.81975034]]
```

- Evaluate the model with mean square error

[21]:
```
#calculating mean square error
MSE = np.mean((target_test_data_normalize  - predicted_outputs) ** 2)
print(f'Mean square error: {MSE}')
```

```
Mean square error: 1.6719706440160127
```