CSE 676
Deep Learning
Final Project Team 18

Prediction Of Future Stock Price Values In Time Series Analysis Using Deep Learning Architectures

About the dataset:

The dataset consists the stock data for Tata Motors. It contains the following columns:

1. **Date**: It consists of Stock data.
2. **Open**: It consists of opening price of the stock.
3. **High**: It consists of highest price of the stock.
4. **Low**: It consists of lowest price of the stock.
5. **Close**: It consists of closing price of the stock.
6. **Adj Close**: It consists of adjusted closing price, which accounts for stock splits.
7. **Volume**: It consists of trading volume of the stock.

The dataset begins on January 2, 1991 in TATAMOTORS.NS.CSV Dataset consists of 8459 rows and 7 columns.

Data Cleaning Steps:

1. Data Frame

```
[9] #Printing frist 10 rows of data
    data.head(10)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1991-01-02	20.959597	21.857864	20.959597	21.857864	15.690222	0.0
1	1991-01-03	20.959597	21.857864	20.959597	21.857864	15.690222	0.0
2	1991-01-04	21.857864	21.857864	21.857864	21.857864	15.690222	0.0
3	1991-01-07	20.360750	21.259020	20.061329	21.109308	15.152890	0.0
4	1991-01-08	21.109308	21.109308	21.109308	21.109308	15.152890	0.0
5	1991-01-09	21.259020	21.259020	20.061329	20.510462	14.723013	0.0
6	1991-01-10	20.510462	20.510462	20.510462	20.510462	14.723013	0.0
7	1991-01-11	20.360750	20.959597	20.061329	20.959597	15.045416	0.0
8	1991-01-14	20.660173	20.660173	20.061329	20.360750	14.615552	0.0
9	1991-01-15	20.360750	20.360750	20.360750	20.360750	14.615552	0.0

2. To check the duplicates in data

```
[14] #To Find the duplicate row
     data.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
8454    False
8455    False
8456    False
8457    False
8458    False
Length: 8459, dtype: bool
```

3. Finding Null Values and dropping from data.

```
[15] #To find null values in the rows
     data.isnull().sum()
```

```
Date         0
Open         7
High         7
Low          7
Close        7
Adj Close    7
Volume       7
dtype: int64
```

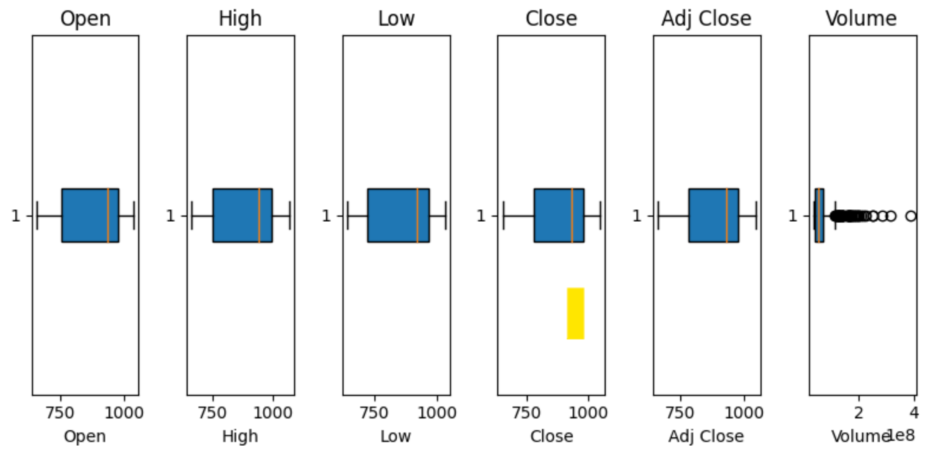
```
[16] #Dropping the null values
     data=data.dropna()
     data
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1991-01-02	20.959597	21.857864	20.959597	21.857864	15.690222	0.0
1	1991-01-03	20.959597	21.857864	20.959597	21.857864	15.690222	0.0
2	1991-01-04	21.857864	21.857864	21.857864	21.857864	15.690222	0.0
3	1991-01-07	20.360750	21.259020	20.061329	21.109308	15.152890	0.0
4	1991-01-08	21.109308	21.109308	21.109308	21.109308	15.152890	0.0
...	...	...	...	...	...	...	...
8454	2024-06-10	977.000000	984.900024	969.099976	975.150024	972.150024	9258931.0
8455	2024-06-11	973.799988	992.549988	966.650024	987.099976	987.099976	14828702.0
8456	2024-06-12	994.500000	1010.250000	987.000000	988.700012	988.700012	17527993.0
8457	2024-06-13	1002.000000	1002.000000	980.750000	985.849976	985.849976	12157226.0
8458	2024-06-14	990.000000	997.250000	981.400024	993.400024	993.400024	11591421.0

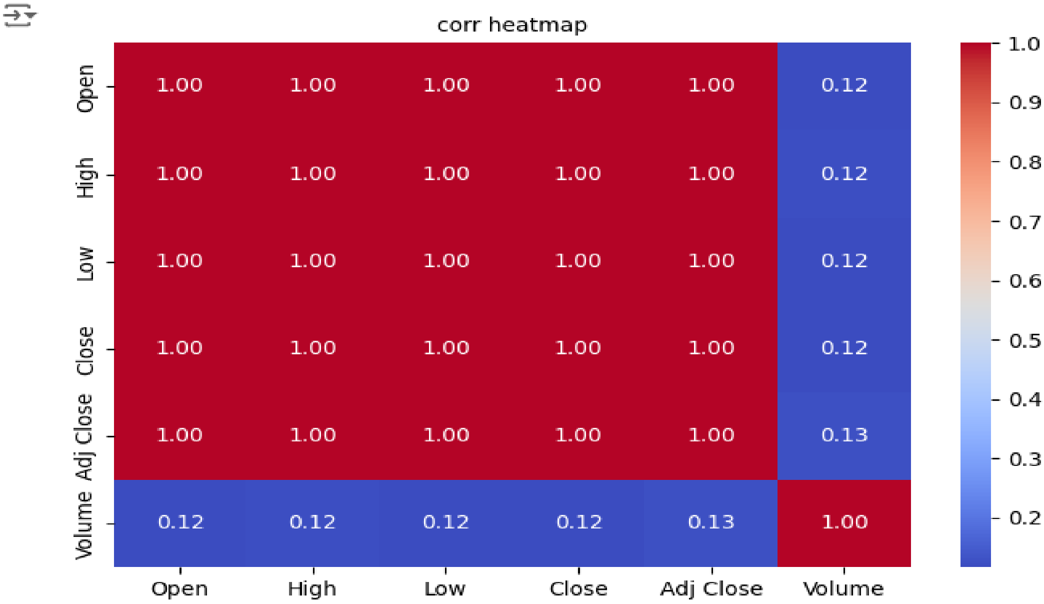
8452 rows × 7 columns

Checking and Removing Outliers from the data:

```
#Plotting to check the outliers
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 4))
for i, feature in enumerate(fts, start=1):
    plt.subplot(1, len(fts), i)
    plt.boxplot(data_valid[feature][outlierss[feature]], vert=False, patch_artist=True)
    plt.xlabel(feature)
    plt.title(f'{feature}')

plt.tight_layout()
plt.show()
```



Correlation Matrix:



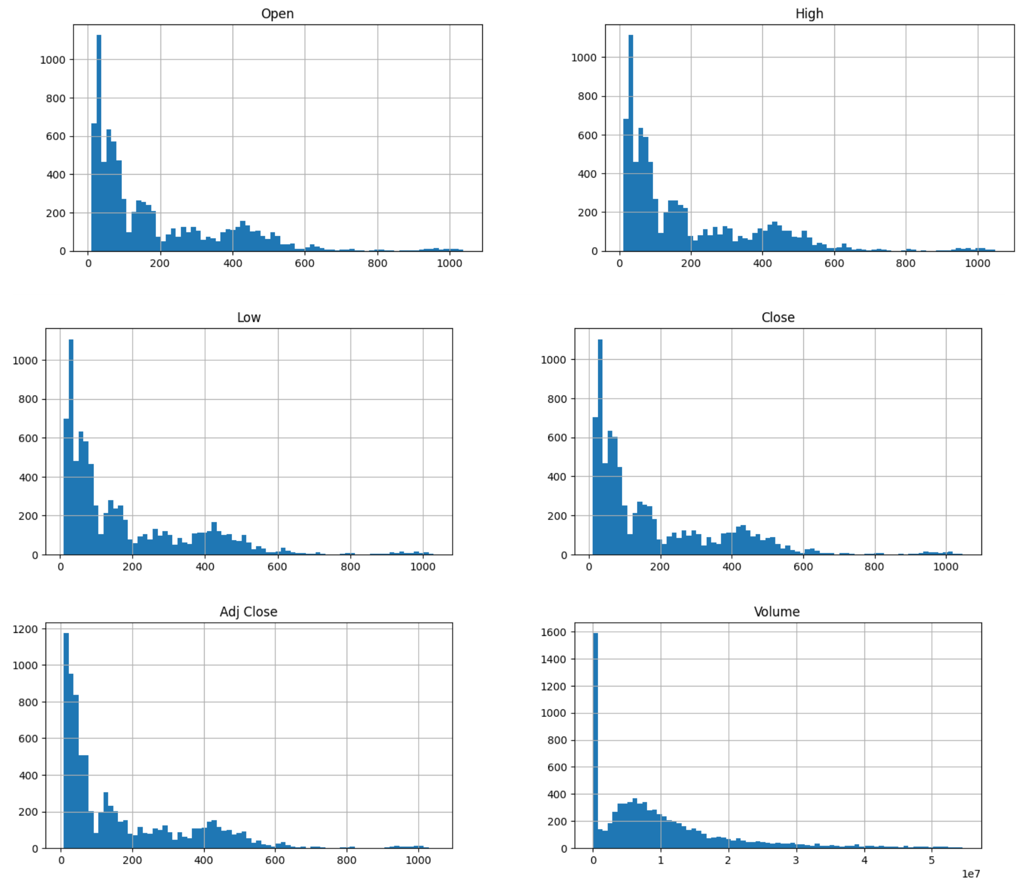
From the correlation matrix plot we can say Open, High, Low, Close, Adj Close columns are perfectly correlated with each other that indicates that value increases in proportional manner.

Visualization Graphs:

1. Generate histograms for each column

```
[26] #Generate histograms for each column
     data_final.hist(bins=75,figsize=(16,14))
```

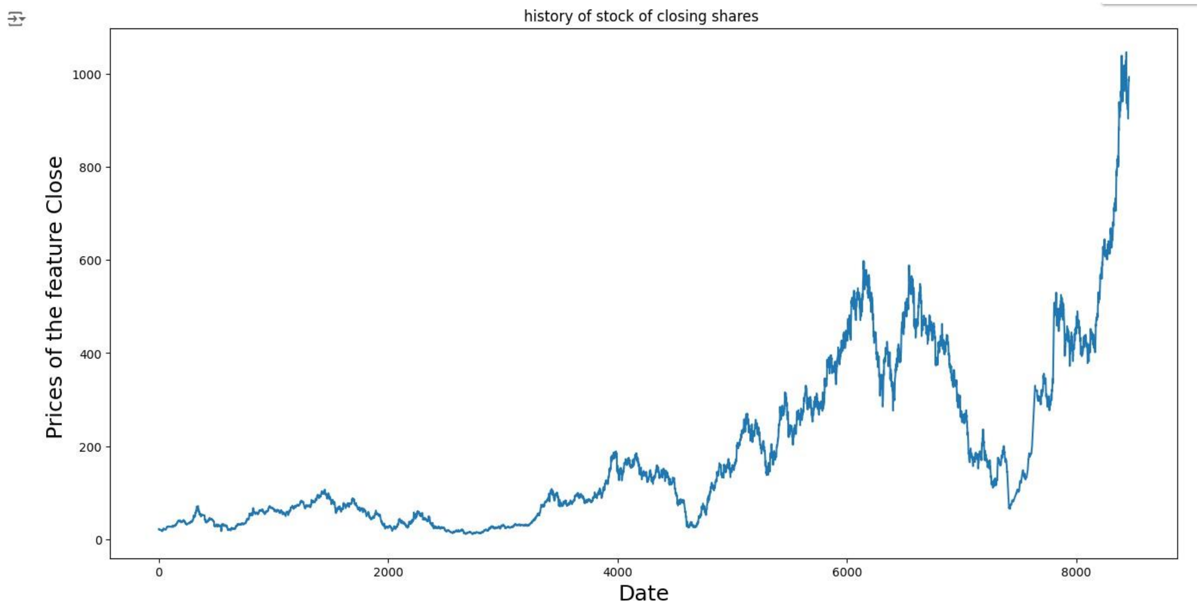
```
array([[<Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>],
       [<Axes: title={'center': 'Adj Close'}>,
        <Axes: title={'center': 'Volume'}>]], dtype=object)
```



The histograms plot provides distribution of values for each column in the dataset.

2.Graph closing prices of the stock over time

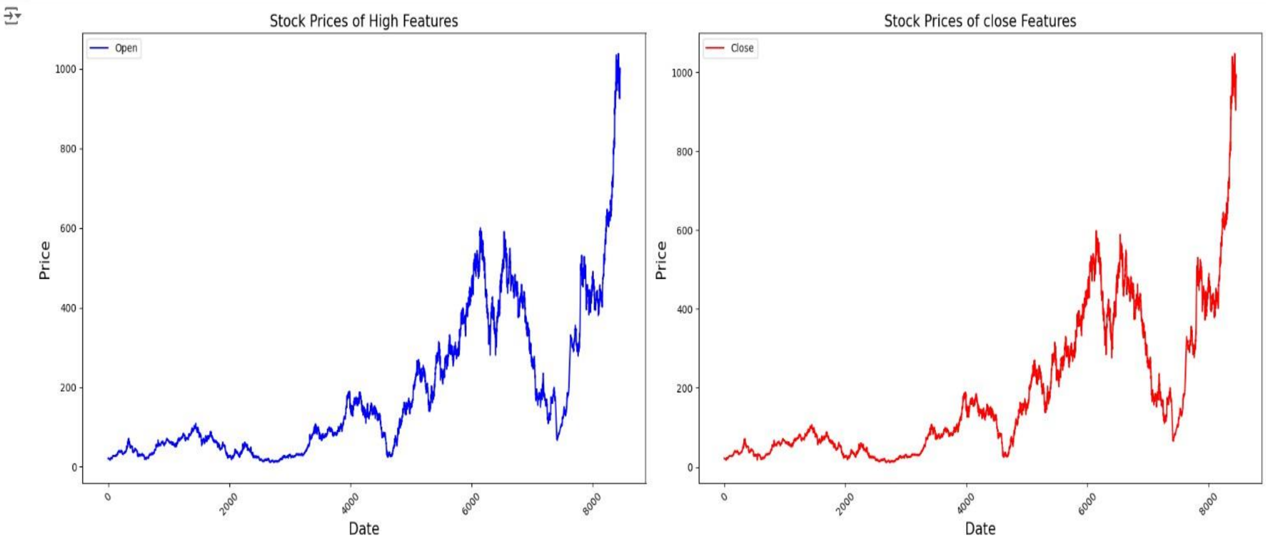
```
#Historical graph closing prices of the stock over time.
plt.figure(figsize=(16,8))
plt.title('history of stock of closing shares ')
plt.plot(data_final['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Prices of the feature Close ', fontsize=18)
plt.show()
```



This plot helps to visualize the price movements such as increase or decrease in trends in closing prices stock.

2. Plotting the 'High' Features and 'Close' Features

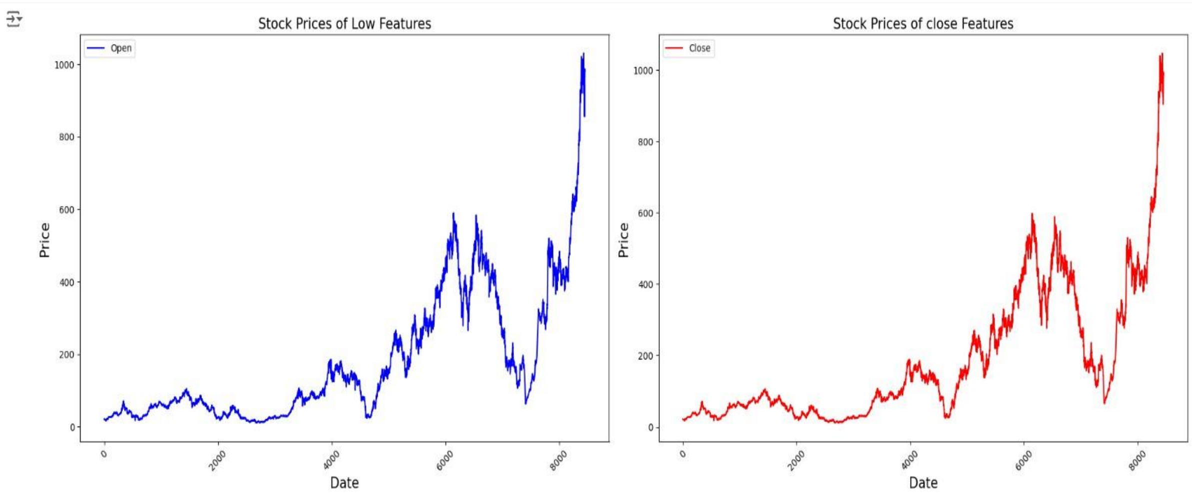
```python
import pandas as pd
import matplotlib.pyplot as plt

def plot(data_final):
    fig, pt = plt.subplots(1, 2, figsize=(20, 7))
    pt[0].plot(data_final['Open'], label='Open', color='blue')
    pt[0].set_xlabel('Date', size=15)
    pt[0].set_ylabel('Price', size=15)
    pt[0].legend()
    pt[0].set_title('Stock Prices of High Features', size=15)
    pt[0].tick_params(axis='x', rotation=45)

    pt[1].plot(data_final['Close'], label='Close', color='red')
    pt[1].set_xlabel('Date', size=15)
    pt[1].set_ylabel('Price', size=15)
    pt[1].legend()
    pt[1].set_title('Stock Prices of close Features', size=15)
    pt[1].tick_params(axis='x', rotation=45)

    plt.tight_layout()
    plt.show()


plot(data_final)
```



The plot of the 'High' and 'Close' prices in whole dataset, to see how the stock prices fluctuated from high to close each day.

3. Plotting the 'Low' prices and 'Close' prices

```
[ ] #Plotting the 'Low' prices and 'Close' prices
    import pandas as pd
    def plot(data_final):
        fig, pt = plt.subplots(1, 2, figsize=(20, 7))
        pt[0].plot(data_final['Low'], label='Open', color='blue')
        pt[0].set_xlabel('Date', size=15)
        pt[0].set_ylabel('Price', size=15)
        pt[0].legend()
        pt[0].set_title('Stock Prices of Low Features', size=15)
        pt[0].tick_params(axis='x', rotation=45)

        pt[1].plot(data_final['Close'], label='Close', color='red')
        pt[1].set_xlabel('Date', size=15)
        pt[1].set_ylabel('Price', size=15)
        pt[1].legend()
        pt[1].set_title('Stock Prices of close Features', size=15)
        pt[1].tick_params(axis='x', rotation=45)

        plt.tight_layout()
        plt.show()


    plot(data_final)
```



The plot of the 'Low' and 'Close' prices in whole dataset, to see how the stock prices fluctuated from low to close each day.

Model 1

LSTM Model:

We implemented LSTM model for time series prediction, specifically for predicting stock prices. Consider the features and label from my dataset:

```
#Defining fetaures(X) and label(y)
X=data_final.drop('Close' , axis=1)
y=data_final['Close']
```

LSTM model architecture:

```
from tensorflow.keras.callbacks import EarlyStopping

# Defining the LSTM model architecture
lstm_model=Sequential()
lstm_model.add(LSTM(units=50, activation='relu', return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2]))) # Updated input shape
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(units=60, activation='relu', return_sequences=True))
lstm_model.add(Dropout(0.3))
lstm_model.add(LSTM(units=80, activation='relu', return_sequences=True))
lstm_model.add(Dropout(0.4))
lstm_model.add(LSTM(units=120, activation='relu', return_sequences=False))
lstm_model.add(Dropout(0.5))

lstm_model.add(Dense(units=1))
lstm_model.summary()
# Compiling the model
lstm_model.compile(optimizer='adam', loss='mean_squared_error')
# Defining the early stopping with patience-10 to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

The model consists of LSTM layers with 50, 60,80,120 units, dropout layers to prevent overfitting, the ReLU activation function and the dense layer with 1 unit for regression tasks.

Output:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 1, 50)             11200

 dropout_4 (Dropout)         (None, 1, 50)             0

 lstm_5 (LSTM)               (None, 1, 60)             26640

 dropout_5 (Dropout)         (None, 1, 60)             0

 lstm_6 (LSTM)               (None, 1, 80)             45120

 dropout_6 (Dropout)         (None, 1, 80)             0

 lstm_7 (LSTM)               (None, 120)               96480

 dropout_7 (Dropout)         (None, 120)               0

 dense_1 (Dense)             (None, 1)                 121

=================================================================
Total params: 179561 (701.41 KB)
Trainable params: 179561 (701.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

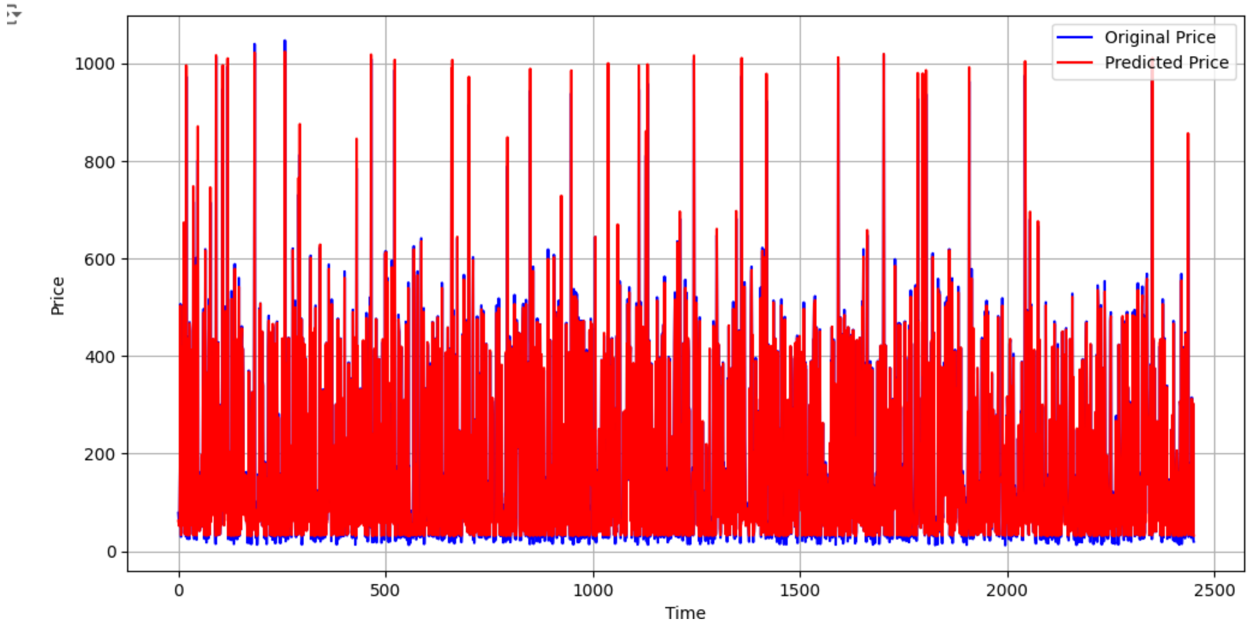
Training the model for epochs =100 and saving the model with .h5 extension:

```
[ ]  # Training the LSTM model
     lstm_model.fit(X_train, y_train, epochs=100)
```

```
# Saving the trained model
lstm_model.save('lstm_model.h5')
```

Plotting the predictions against the true values:

```
# Plotting the predictions against the true values
plt.figure(figsize = (12,6))
plt.plot(y_test, 'b', label = "Original Price")
plt.plot(pred, 'r', label = "Predicted Price")
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



It shows the plot of the original and predicted prices, with the predicted values exhibiting much higher variance between predicted and original price.

Mean absolute error, mean squared error, r2 score:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, pred)
rmse = mean_squared_error(y_test, pred, squared=False)

print("MAE:", mae)
print("RMSE:", rmse)
```

```
MAE: 6.58317012855747
RMSE: 9.32319357829117
```

```
from sklearn.metrics import r2_score
# Calculate evaluation metrics
r2 = r2_score(y_test, pred) *100
print("R-squared percentage:", r2)
```

```
R-squared percentage: 99.74468230824802
```

From the above MAE, RMSE, R2-score we can say that this model is a good fit

Model 2

So for model 2 we are going GRU model in LSTM.Here is berify description how the GRU model code implemented to the data set

The code generates and assembles a neural network model for time series prediction using bidirectional GRU layers. The model architecture includes four bidirectional GRU layers, each of which has 128 units and a tanh activation function. To prevent overfitting, each GRU layer includes a dropout layer that randomly removes 10% of the input units during training. The end result is a dense layer consisting of a single unit. The model is built using the Stochastic Gradient Descent (SGD) optimizer, which has a small decay rate that gradually lowers the learning rate over time as well as a learning rate of 0.001. Furthermore, early stopping is set up to restore the model weights from the best epoch by monitoring the validation loss and stopping training if it doesn't improve for five consecutive epochs. By balancing overfitting and model complexity, this configuration seeks to offer a reliable framework for time series prediction.

The below image the describes about the layer and shapes on bidirectional GRU model:

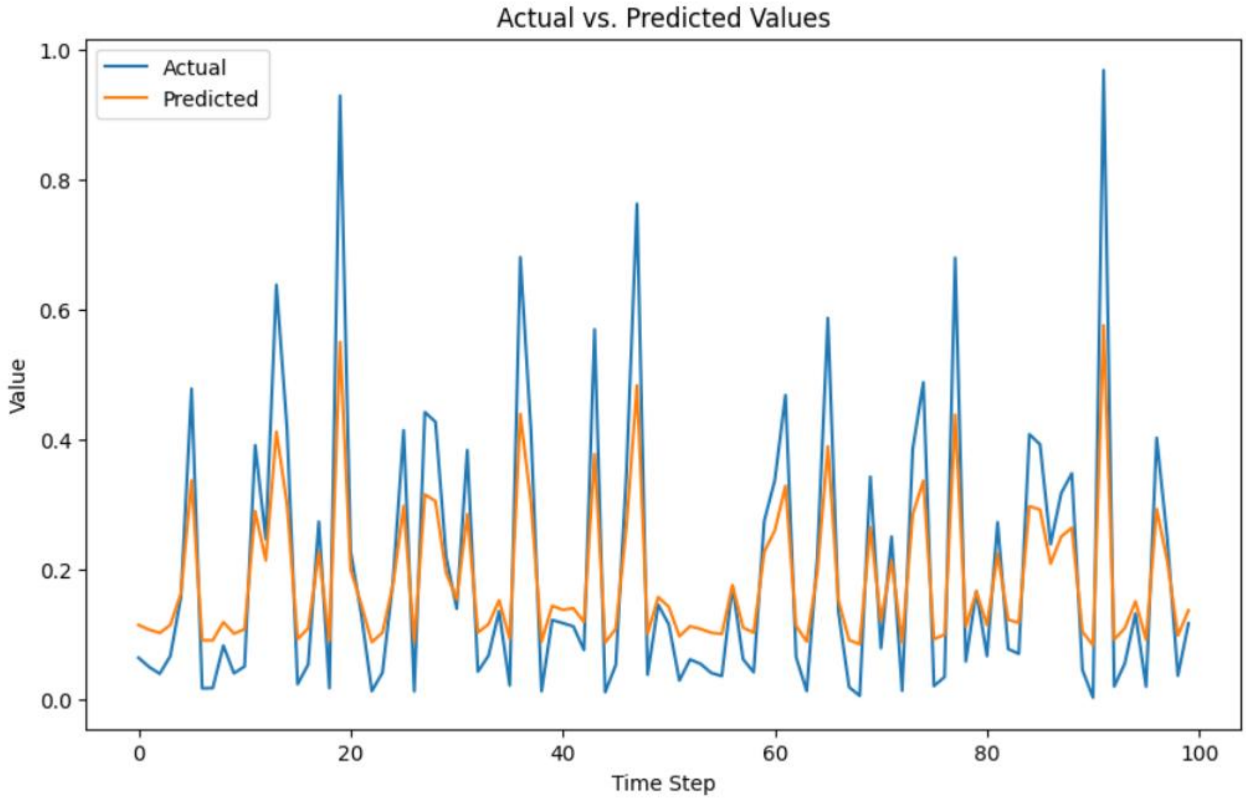
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional_27 (Bidirect  (None, 1, 256)            103680
 ional)

 dropout_27 (Dropout)        (None, 1, 256)            0

 bidirectional_28 (Bidirect  (None, 1, 256)            296448
 ional)

 dropout_28 (Dropout)        (None, 1, 256)            0

 bidirectional_29 (Bidirect  (None, 1, 256)            296448
 ional)

 dropout_29 (Dropout)        (None, 1, 256)            0

 bidirectional_30 (Bidirect  (None, 256)               296448
 ional)

 dropout_30 (Dropout)        (None, 256)               0

 dense_6 (Dense)             (None, 1)                 257

=================================================================
Total params: 993281 (3.79 MB)
Trainable params: 993281 (3.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

We are training the model by using model.fit where we are training neural network models and by passing the parameters as like train, test, no of epochs, and batch size. Once the model training is done we are saving weights in a model.h5 file. Were .h5 extension is usually used to indicate that a file is HDF5 (Hierarchical Data Format version 5), which is a format and toolkit for handling complex data. The.h5 extension is frequently used to save and load model architecture, model weights, or the complete model when using Keras and TensorFlow.

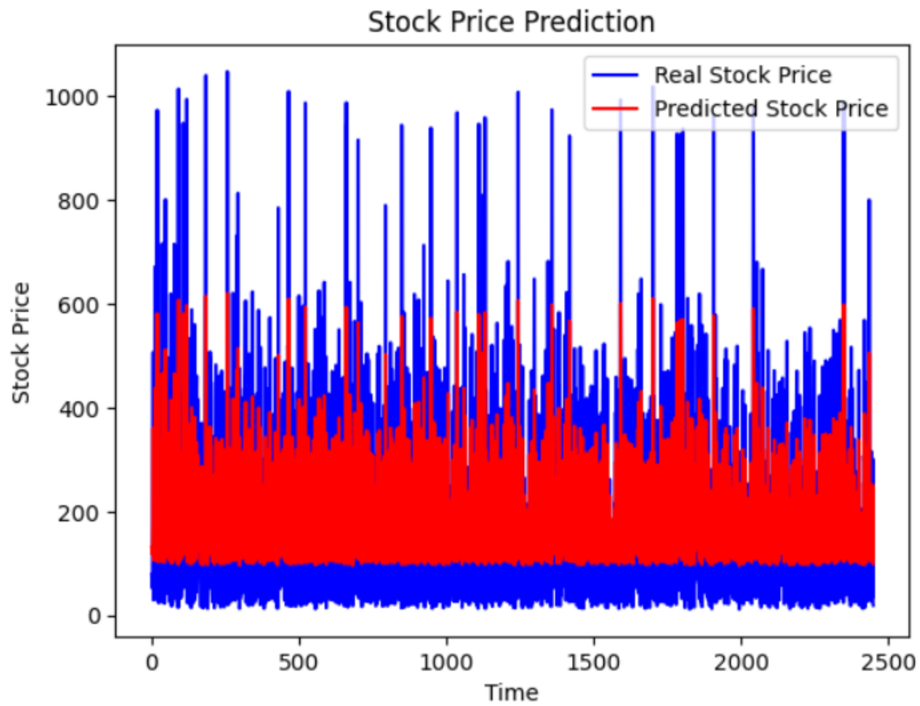
Once training the model is done we start prdicting the stock prices with x_test and y_test. And then we will calculate the accuracy, F1 score, precision, and recall.

```
Accuracy: 0.9678
F1 Score: 0.4060
Precision: 1.0000
Recall: 0.2547
```

After calculating all these accuracy, F1 score , recall and precision we are evaluting the model on testing set.post that we are ploting the predicted values and actual values.



And post that prediction we plot the graph for original stock price and predicted stock price.



And then finally we calculated the values of mean square error, root mean square error and absoulte mean error.

```
77/77 [==============================] - 1s 8ms/step
Mean Squared Error: 0.0071807131934769185
Root Mean Squared Error: 0.08473908893466414
Mean Absolute Error: 0.06771632602827368
```

Model 3

So for the model 3 we implemented conventional neural network model (**CNN**)

After all the preprocessing of data is done we are implementing CNN on the dataset to predict the increment or decrement of the stock price for the feature.

Here the CNN model we implemented are We used the Keras library to define a sequential Convolutional Neural Network (CNN) for a binary classification task, which was later modified to a multi-class classification task. With an input shape determined by seq_length and the number of features in x_train, the network begins with a Conv1D layer that has 64 filters, a kernel size of 3, and the 'tanh' activation function. A MaxPooling1D layer and a Dropout layer with a 20% dropout rate come next in order to avoid overfitting. In a similar manner, MaxPooling1D, Dropout, and another Conv1D layer are added. Subsequently, the output is compressed and introduced into a Dense layer featuring 50 units and 'tanh' activation, succeeded by an additional Dropout layer. For binary classification, the final Dense layer originally has a single unit with a'sigmoid' activation. However, for multi-class classification, model.pop() removes this unit and replaces it with a Dense layer with two units and a'softmax' activation. The model's architecture is shown by the model.summary() function.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_2 (Conv1D)           (None, 98, 64)            448       
                                                                 
 max_pooling1d_2 (MaxPoolin  (None, 49, 64)            0         
 g1D)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 49, 64)            0         
                                                                 
 conv1d_3 (Conv1D)           (None, 47, 64)            12352     
                                                                 
 max_pooling1d_3 (MaxPoolin  (None, 23, 64)            0         
 g1D)                                                            
                                                                 
 dropout_4 (Dropout)         (None, 23, 64)            0         
                                                                 
 flatten_1 (Flatten)         (None, 1472)              0         
                                                                 
 dense_3 (Dense)             (None, 50)                73650     
                                                                 
 dropout_5 (Dropout)         (None, 50)                0         
                                                                 
 dense_5 (Dense)             (None, 2)                 102       
                                                                 
=================================================================
Total params: 86552 (338.09 KB)
Trainable params: 86552 (338.09 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

A previously established model with 70 epochs of training data (x_train and y_train). During training, the model's performance is also evaluated using the validation data (x_val and y_val). After processing each of the 32 samples in the training batch, the model updates its weights. For further analysis or visualisation, the model.fit() function returns a history object containing information about the training procedure, such as the loss and accuracy for the training and validation sets at each epoch.
The below screen shot is the results of epochs on training and testing and validation data:

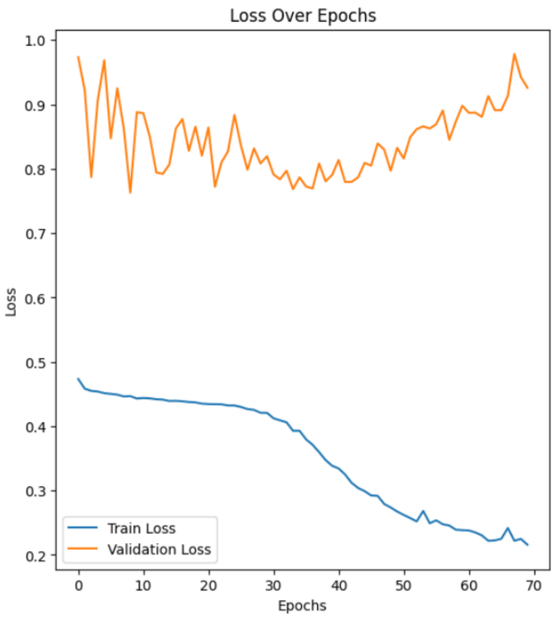
```
Epoch 1/70
168/168 [==============================] - 2s 6ms/step - loss: 0.4734 - accuracy: 0.4616 - val_loss: 0.9734 - val_accuracy: 0.5067
Epoch 2/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4583 - accuracy: 0.4619 - val_loss: 0.9222 - val_accuracy: 0.5097
Epoch 3/70
168/168 [==============================] - 1s 4ms/step - loss: 0.4549 - accuracy: 0.4621 - val_loss: 0.7872 - val_accuracy: 0.5180
Epoch 4/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4539 - accuracy: 0.4629 - val_loss: 0.9065 - val_accuracy: 0.5112
Epoch 5/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4513 - accuracy: 0.4627 - val_loss: 0.9687 - val_accuracy: 0.5045
Epoch 6/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4502 - accuracy: 0.4627 - val_loss: 0.8473 - val_accuracy: 0.5187
Epoch 7/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4490 - accuracy: 0.4627 - val_loss: 0.9251 - val_accuracy: 0.5022
Epoch 8/70
168/168 [==============================] - 1s 4ms/step - loss: 0.4463 - accuracy: 0.4621 - val_loss: 0.8627 - val_accuracy: 0.5060
Epoch 9/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4467 - accuracy: 0.4623 - val_loss: 0.7630 - val_accuracy: 0.5247
Epoch 10/70
168/168 [==============================] - 1s 6ms/step - loss: 0.4430 - accuracy: 0.4621 - val_loss: 0.8880 - val_accuracy: 0.5075
Epoch 11/70
168/168 [==============================] - 1s 7ms/step - loss: 0.4438 - accuracy: 0.4625 - val_loss: 0.8864 - val_accuracy: 0.5037
Epoch 12/70
168/168 [==============================] - 1s 6ms/step - loss: 0.4432 - accuracy: 0.4634 - val_loss: 0.8496 - val_accuracy: 0.5090
Epoch 13/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4419 - accuracy: 0.4625 - val_loss: 0.7942 - val_accuracy: 0.5052
Epoch 14/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4413 - accuracy: 0.4627 - val_loss: 0.7922 - val_accuracy: 0.5097
Epoch 15/70
168/168 [==============================] - 1s 5ms/step - loss: 0.4392 - accuracy: 0.4634 - val_loss: 0.8068 - val_accuracy: 0.5142
```

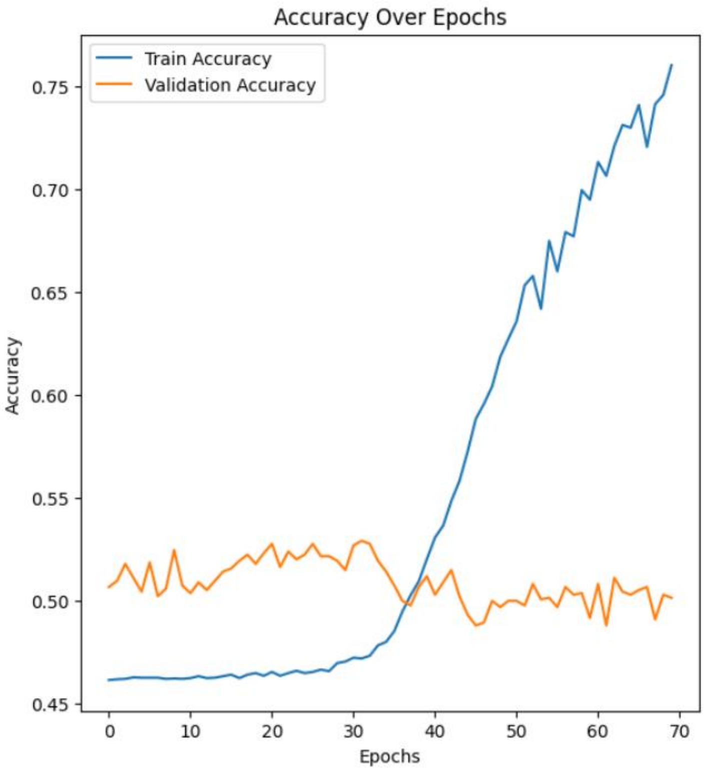
The below mentioned are the accuracies and loss on training, validation and testing:

```
168/168 [==============================] - 1s 4ms/step - loss: 0.1314 - accuracy: 0.8775
42/42 [==============================] - 0s 2ms/step - loss: 0.9260 - accuracy: 0.5015
53/53 [==============================] - 0s 2ms/step - loss: 0.8748 - accuracy: 0.5009
```

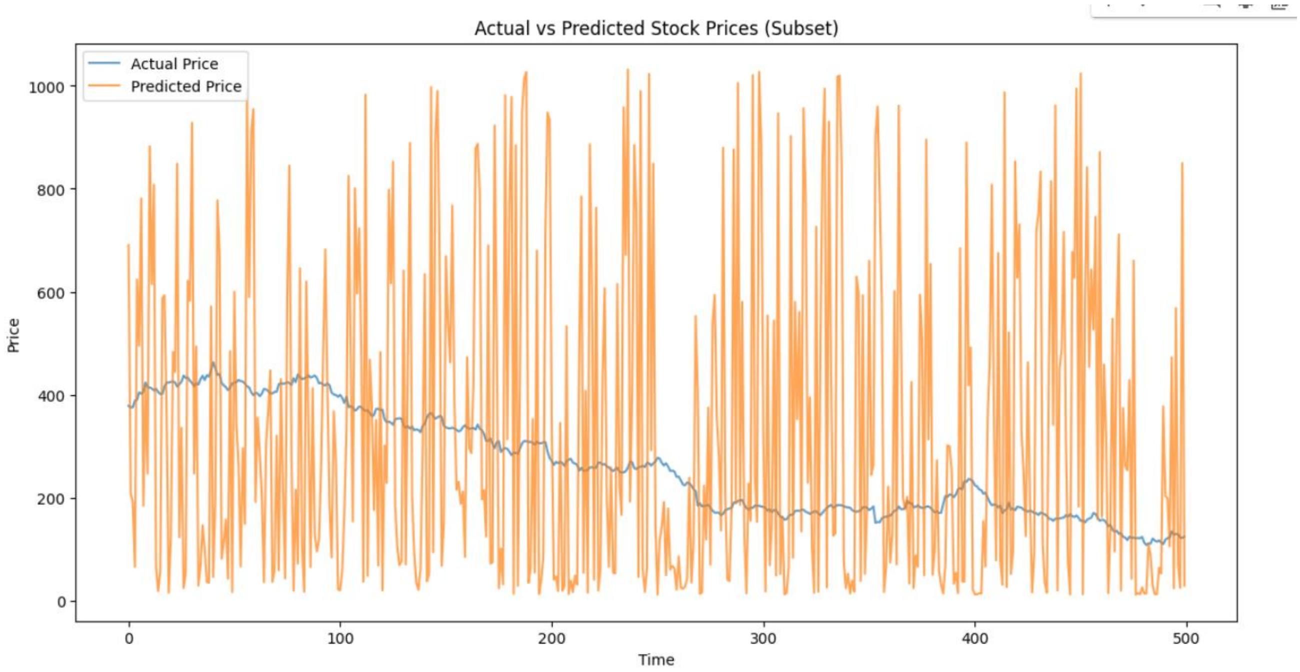
And all training the model we are all the results in a file named as model with .h5 extension.

We aslo ploted the graph for loss and accuracies for the trained model:



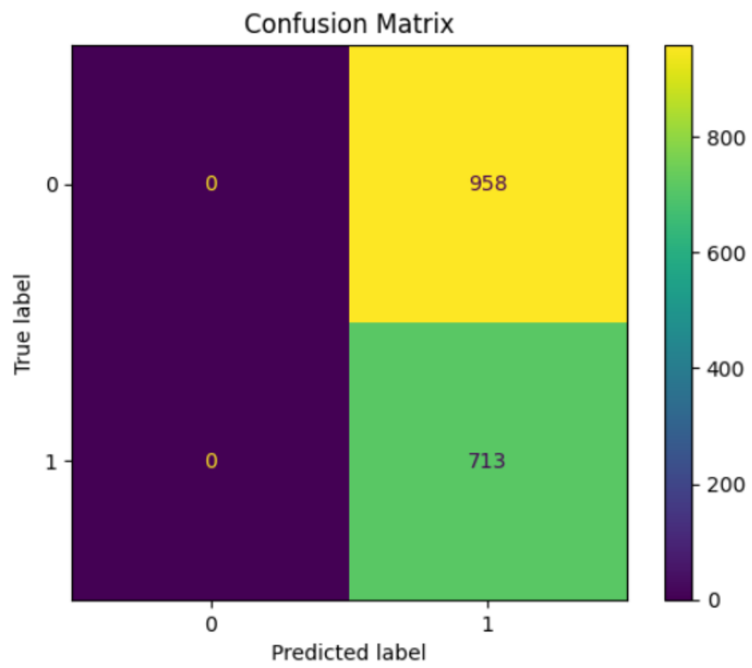


After this plots in the code the are predicting furture prices values of the stock and plot the graph for original and predicted price.



The above is the graph which we ploted for the predicted and actual stock price.

And we also ploted the confusion matrix and precision and recall scores also.




```
              precision    recall  f1-score   support

        Down       0.00      0.00      0.00       958
          Up       0.43      1.00      0.60       713

    accuracy                           0.43      1671
   macro avg       0.21      0.50      0.30      1671
weighted avg       0.18      0.43      0.26      1671
```

```
Precision: 0.5080789946140036
Recall: 0.7676311030741411
```

- Of all the models – Model 2 (using GRU)worked best with an accuracy of 96% while Model 3 (Using CNN) is under fitting for our time series dataset and Model 1 which is using LSTM method is over fitting as its accuracy is about 99%. Dependieng on the metrics Model 2 Using GRU worked best for our dataset.

Team Member	Project Part	Contribution (%)
Tejesh Reddy	Model 3 & documentation for model 3	30%
Kota Ruchitha	Model 2 & documentation for model 1	30%
Sannapureddy,Uma Maheswara Reddy	Model 1 & documentation for model 2	40%

References:

- Yahoo Finance - Stock Market Live, Quotes, Business C Finance News
- https://www.kaggle.com/datasets/arashnic/time-series-forecasting-with-yahoo-stock- price/code
- https://www.kaggle.com/code/dylanyves/lstm-predicting-stock-market-beginner https://www.tensorflow.org/
- https://medium.com/@sayahfares19/time-series-analysis-with-pandas-and-matplotlib-yahoo-
- finance-data-fc4ad67c268c
- https://urbizedge.com/time-series-forecast-of-yahoo-finance-data/ https://medium.com/@deepml1818/predicting-stock-prices-using-lstm-and-yahoo-finance-
- data-0e2534b269a1
- Yahoo Finance - Stock Market Live, Quotes, Business C Finance News
- https://www.kaggle.com/datasets/arashnic/time-series-forecasting-with-yahoo-stock- price/code
- https://www.kaggle.com/code/dylanyves/lstm-predicting-stock-market-beginner https://www.tensorflow.org/
- https://www.kaggle.com/code/dpamgautam/stock-price-prediction-lstm-gru-rnn
- https://github.com/AshrafAlroomi/stock-market-cnn/blob/master/jupyter/model_performance.ipynb
- https://www.scirp.org/journal/paperinformation?paperid=132499
- https://matplotlib.org/
- https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html?gclid=Cj0KCQjw1qO0BhDwARIsANfnkv858taV2hs5vjqlUwwIMvqbzOqygX2RrSpB8PbjdHbm8D94OuUXquAaAiBQEALw_wcB&ef_id=Cj0KCQjw1qO0BhDwARIsANfnkv858taV2hs5vjqlUwwIMvqbzOqygX2RrSpB8PbjdHbm8D94OuUXquAaAiBQEALw_wcB:G:s&s_kwcid=AL!8664!3!591866074057!b!!g!!%2Bconvolutional%20%2Bneural%20%2Bnetwork&s_eid=psn_57384017272&q=+convolutional++neural++network&gad_source=1
- https://www.kaggle.com/code/yassineghouzam/introduction-to-cnn-keras-0-997-top-6
- https://matplotlib.org/
- https://www.datacamp.com/tutorial/seaborn-heatmaps
- https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/
- https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial
- https://www.datacamp.com/tutorial/what-is-a-confusion-matrix-in-machine-learning?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720830&utm_adgroupid=157156377071&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=684592141199&utm_targetid=dsa-2218886984380&utm_loc_interest_ms=&utm_loc_physical_ms=9005555&utm_content=&utm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-us_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_10-na_11-na&gad_source=1&gclid=Cj0KCQjw1qO0BhDwARIsANfnkv_5wiKv9tTYVlM8QHB0YYxje8xttjAtOW8K3hpip6QjBdY7fMX8GPAaAiKfEALw_wcB