Submitted By : Mrinal Bhan

DSAI - 211020428

```python
import numpy as np  # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")

np.random.seed(42)
data = pd.read_csv("/kaggle/input/tesla-stock-price/Tesla.csv - Tesla.csv.csv")

```

```python
data.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1692 entries, 0 to 1691
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1692 non-null   object 
 1   Open       1692 non-null   float64
 2   High       1692 non-null   float64
 3   Low        1692 non-null   float64
 4   Close      1692 non-null   float64
 5   Volume     1692 non-null   int64  
 6   Adj Close  1692 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 92.7+ KB
```

```python
length_data = len(data)  # rows that data has
split_ratio = 0.7  # %70 train + %30 validation
length_train = round(length_data * split_ratio)
length_validation = length_data - length_train
print("Data length :", length_data)
print("Train data length :", length_train)
print("Validation data lenth :", length_validation)

```

```
Data length : 1692
Train data length : 1184
Validation data lenth : 508
```

```python
train_data = data[:length_train].iloc[:, :2]
train_data["Date"] = pd.to_datetime(
    train_data["Date"]
)  # converting to date time object
train_data

```

	Date	Open
0	2010-06-29	19.000000
1	2010-06-30	25.790001
2	2010-07-01	25.000000
3	2010-07-02	23.000000
4	2010-07-06	20.000000
...	...	...
1179	2015-03-06	199.210007
1180	2015-03-09	194.389999
1181	2015-03-10	188.460007
1182	2015-03-11	191.149994
1183	2015-03-12	193.750000

1184 rows × 2 columns

```python
# Creatting train dataset
dataset_train = train_data.Open.values
dataset_train.shape

```

```
(1184,)
```

```
# Change 1d array to 2d array
# Changing shape from (1692,) to (1692,1)
dataset_train = np.reshape(dataset_train, (-1, 1))
dataset_train.shape

```

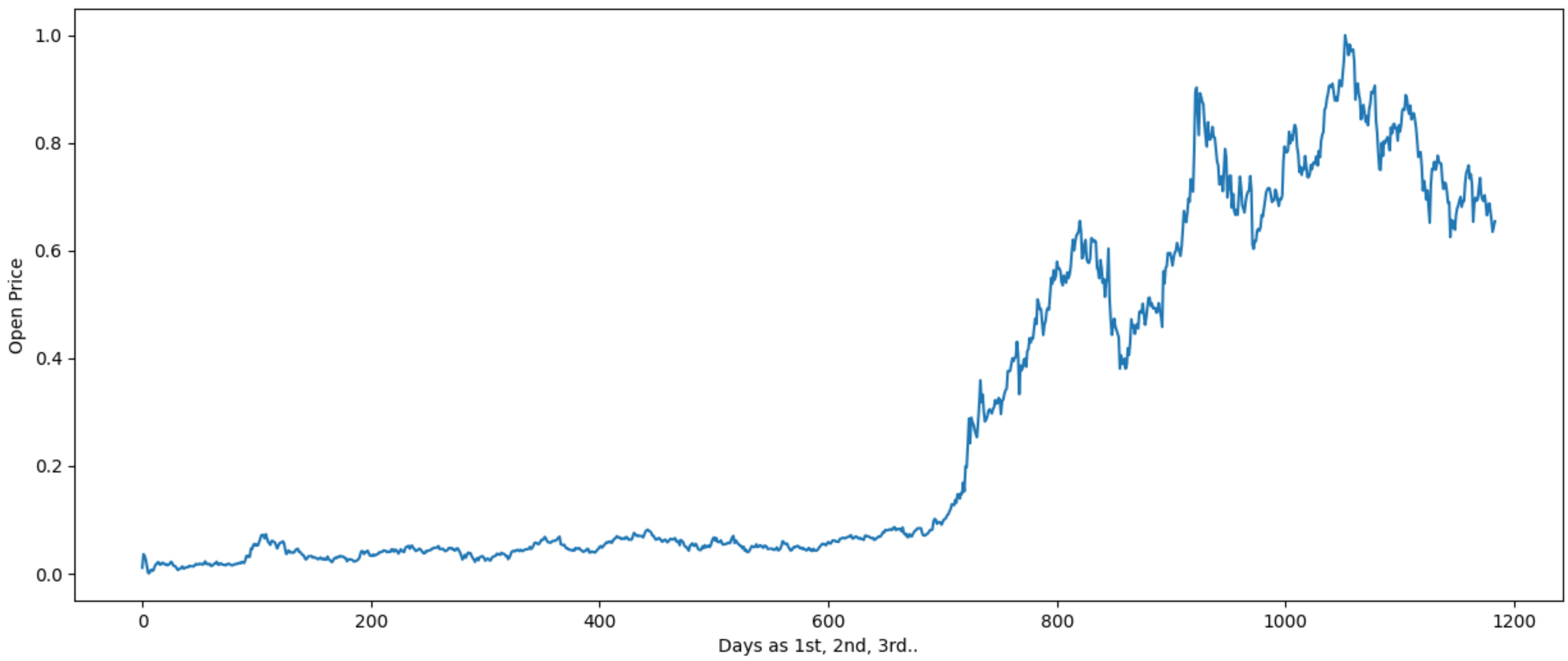
```
(1184, 1)
```

Feature scaling

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))


# scaling dataset
dataset_train_scaled = scaler.fit_transform(dataset_train)

dataset_train_scaled.shape

```

```
(1184, 1)
```

```
plt.subplots(figsize=(15, 6))
plt.plot(dataset_train_scaled)
plt.xlabel("Days as 1st, 2nd, 3rd..")
plt.ylabel("Open Price")
plt.show()

```




```
X_train = []
y_train = []

time_step = 50

for i in range(time_step, length_train):
    X_train.append(dataset_train_scaled[i - time_step : i, 0])
    y_train.append(dataset_train_scaled[i, 0])

# convert list to array
X_train, y_train = np.array(X_train), np.array(y_train)

```

```
print("Shape of X_train before reshape :", X_train.shape)
print("Shape of y_train before reshape :", y_train.shape)

```

```
Shape of X_train before reshape : (1134, 50)
Shape of y_train before reshape : (1134,)
```

```
# Reshaping the data
```

```
# Reshaping the data
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
y_train = np.reshape(y_train, (y_train.shape[0], 1))

print("Shape of X_train after reshape :", X_train.shape)

```

```
Shape of X_train after reshape : (1134, 50, 1)
```

Training the RNN architecture

```
# importing libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import Dropout

```

```
# initializing the RNN
regressor = Sequential()

# adding first RNN layer and dropout regulatization
regressor.add(
    SimpleRNN(
        units=50,
        activation="ReLU",
        return_sequences=True,
        input_shape=(X_train.shape[1], 1),
    )
)

regressor.add(Dropout(0.2))


# adding second RNN layer and dropout regulatization
regressor.add(SimpleRNN(units=50, activation="ReLU", return_sequences=True))

regressor.add(Dropout(0.2))

# adding third RNN layer and dropout regulatization
regressor.add(SimpleRNN(units=50, activation="ReLU", return_sequences=True))

regressor.add(Dropout(0.2))

# adding fourth RNN layer and dropout regulatization
regressor.add(SimpleRNN(units=50))

regressor.add(Dropout(0.2))

# adding the output layer
regressor.add(Dense(units=1))

```

```
# compiling RNN
regressor.compile(optimizer="adam", loss="mean_squared_error", metrics=["accuracy"])

# fitting the RNN
history = regressor.fit(X_train, y_train, epochs=50, batch_size=32)

```

```
Epoch 1/50
36/36 [==============================] - 5s 45ms/step - loss: 0.0782 - accuracy: 8.8183e-04
Epoch 2/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0172 - accuracy: 0.0000e+00
Epoch 3/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0123 - accuracy: 8.8183e-04
Epoch 4/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0094 - accuracy: 8.8183e-04
Epoch 5/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0088 - accuracy: 8.8183e-04
Epoch 6/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0084 - accuracy: 8.8183e-04
Epoch 7/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0078 - accuracy: 8.8183e-04
Epoch 8/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0058 - accuracy: 8.8183e-04
Epoch 9/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0062 - accuracy: 8.8183e-04
Epoch 10/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0059 - accuracy: 8.8183e-04
Epoch 11/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0053 - accuracy: 8.8183e-04
```

```
Epoch 12/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0045 - accuracy: 8.8183e-04
Epoch 13/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0050 - accuracy: 8.8183e-04
Epoch 14/50
36/36 [==============================] - 2s 45ms/step - loss: 0.0046 - accuracy: 8.8183e-04
Epoch 15/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0041 - accuracy: 8.8183e-04
Epoch 16/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0043 - accuracy: 8.8183e-04
Epoch 17/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0040 - accuracy: 8.8183e-04
Epoch 18/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0042 - accuracy: 8.8183e-04
Epoch 19/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0040 - accuracy: 8.8183e-04
Epoch 20/50
36/36 [==============================] - 2s 45ms/step - loss: 0.0039 - accuracy: 8.8183e-04
Epoch 21/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0038 - accuracy: 8.8183e-04
Epoch 22/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0046 - accuracy: 8.8183e-04
Epoch 23/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0042 - accuracy: 8.8183e-04
Epoch 24/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0039 - accuracy: 8.8183e-04
Epoch 25/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0035 - accuracy: 8.8183e-04
Epoch 26/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0037 - accuracy: 8.8183e-04
Epoch 27/50
36/36 [==============================] - 2s 48ms/step - loss: 0.0039 - accuracy: 8.8183e-04
Epoch 28/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0033 - accuracy: 8.8183e-04
Epoch 29/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0031 - accuracy: 8.8183e-04
```

Utilizing tanh as activation function

```python
# initializing the RNN
regressor = Sequential()

# adding first RNN layer and dropout regulatization
regressor.add(
    SimpleRNN(
        units=50,
        activation="tanh",
        return_sequences=True,
        input_shape=(X_train.shape[1], 1),
    )
)

regressor.add(Dropout(0.2))


# adding second RNN layer and dropout regulatization
regressor.add(SimpleRNN(units=50, activation="tanh", return_sequences=True))

regressor.add(Dropout(0.2))

# adding third RNN layer and dropout regulatization
regressor.add(SimpleRNN(units=50, activation="tanh", return_sequences=True))

regressor.add(Dropout(0.2))

# adding fourth RNN layer and dropout regulatization
regressor.add(SimpleRNN(units=50))

regressor.add(Dropout(0.2))

# adding the output layer
regressor.add(Dense(units=1))

```

```python
# compiling RNN
regressor.compile(optimizer="adam", loss="mean_squared_error", metrics=["accuracy"])

# fitting the RNN
history = regressor.fit(X_train, y_train, epochs=50, batch_size=32)

```

```
Epoch 1/50
36/36 [==============================] - 5s 45ms/step - loss: 0.3447 - accuracy: 8.8183e-04
Epoch 2/50
36/36 [==============================] - 2s 47ms/step - loss: 0.2111 - accuracy: 8.8183e-04
```

```
Epoch 3/50
36/36 [==============================] - 2s 46ms/step - loss: 0.1401 - accuracy: 8.8183e-04
Epoch 4/50
36/36 [==============================] - 2s 47ms/step - loss: 0.1071 - accuracy: 0.0000e+00
Epoch 5/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0693 - accuracy: 8.8183e-04
Epoch 6/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0522 - accuracy: 8.8183e-04
Epoch 7/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0496 - accuracy: 8.8183e-04
Epoch 8/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0410 - accuracy: 8.8183e-04
Epoch 9/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0305 - accuracy: 8.8183e-04
Epoch 10/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0264 - accuracy: 8.8183e-04
Epoch 11/50
36/36 [==============================] - 2s 48ms/step - loss: 0.0275 - accuracy: 8.8183e-04
Epoch 12/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0250 - accuracy: 8.8183e-04
Epoch 13/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0229 - accuracy: 8.8183e-04
Epoch 14/50
36/36 [==============================] - 2s 48ms/step - loss: 0.0219 - accuracy: 8.8183e-04
Epoch 15/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0182 - accuracy: 8.8183e-04
Epoch 16/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0155 - accuracy: 8.8183e-04
Epoch 17/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0150 - accuracy: 8.8183e-04
Epoch 18/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0152 - accuracy: 8.8183e-04
Epoch 19/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0158 - accuracy: 8.8183e-04
Epoch 20/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0129 - accuracy: 8.8183e-04
Epoch 21/50
36/36 [==============================] - 2s 52ms/step - loss: 0.0126 - accuracy: 8.8183e-04
Epoch 22/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0113 - accuracy: 8.8183e-04
Epoch 23/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0108 - accuracy: 8.8183e-04
Epoch 24/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0103 - accuracy: 8.8183e-04
Epoch 25/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0090 - accuracy: 8.8183e-04
Epoch 26/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0086 - accuracy: 8.8183e-04
Epoch 27/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0100 - accuracy: 8.8183e-04
Epoch 28/50
36/36 [==============================] - 2s 46ms/step - loss: 0.0096 - accuracy: 8.8183e-04
Epoch 29/50
36/36 [==============================] - 2s 47ms/step - loss: 0.0077 - accuracy: 8.8183e-04
```

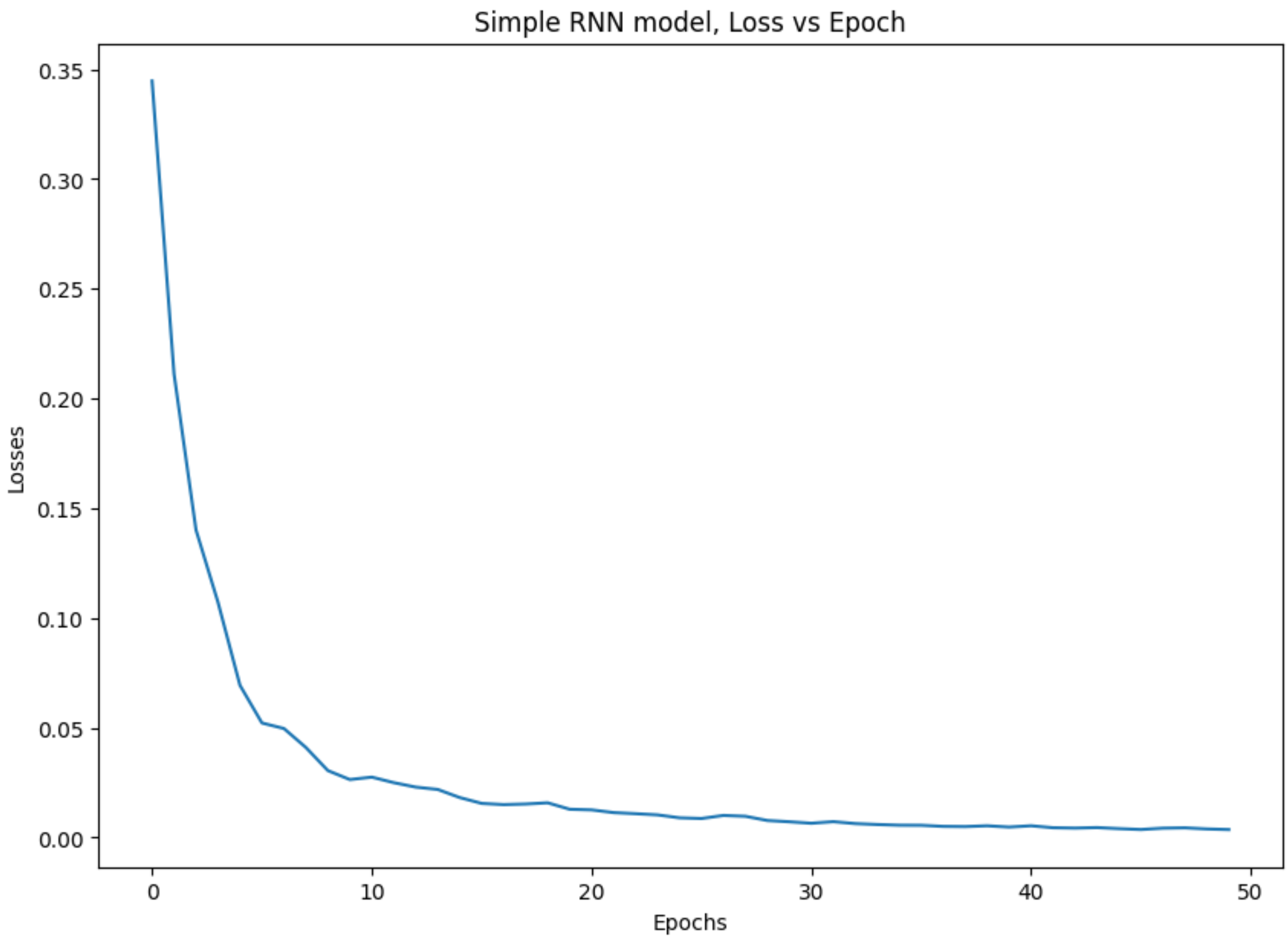
Evaluating the model

```
# Losses
history.history["loss"]

```

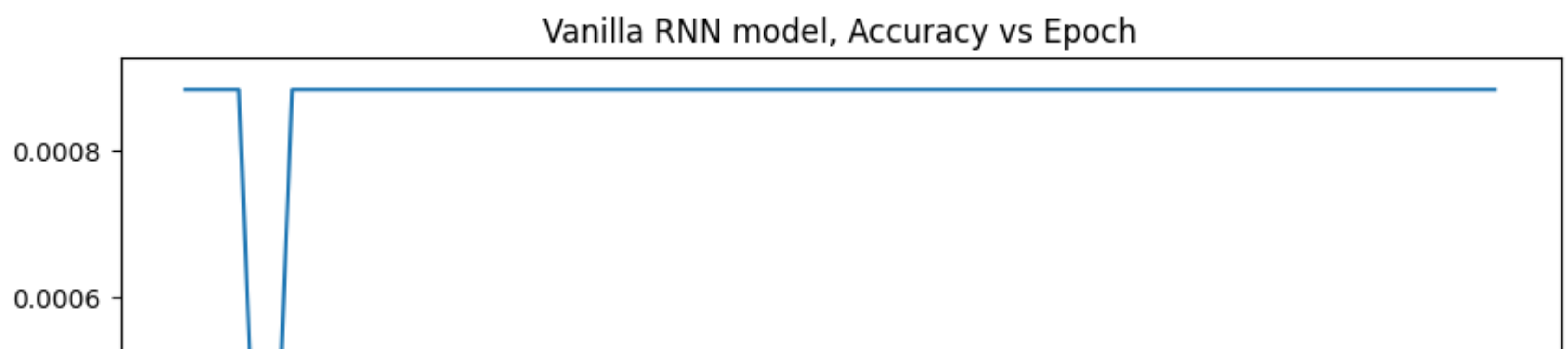
```
[0.3446778953075409,
 0.21114075183868408,
 0.14008377492427826,
 0.10714800655841827,
 0.0693284198641777,
 0.052157990634441376,
 0.04964325204491615,
 0.041003912687301636,
 0.030470574274659157,
 0.026379797607660294,
 0.02748677134513855,
 0.024967987090349197,
 0.022941455245018005,
 0.021872660145163536,
 0.018185725435614586,
 0.015520120970904827,
 0.0149683291092515,
 0.0152418939396739,
 0.015794532373547554,
 0.01286360714584589,
 0.012567155994474888,
 0.011335828341543674,
 0.010833166539669037,
 0.010291064158082008,
 0.00895395502448082,
 0.008637135848402977,
 0.010038164444426775,
 0.009649925276303291
```

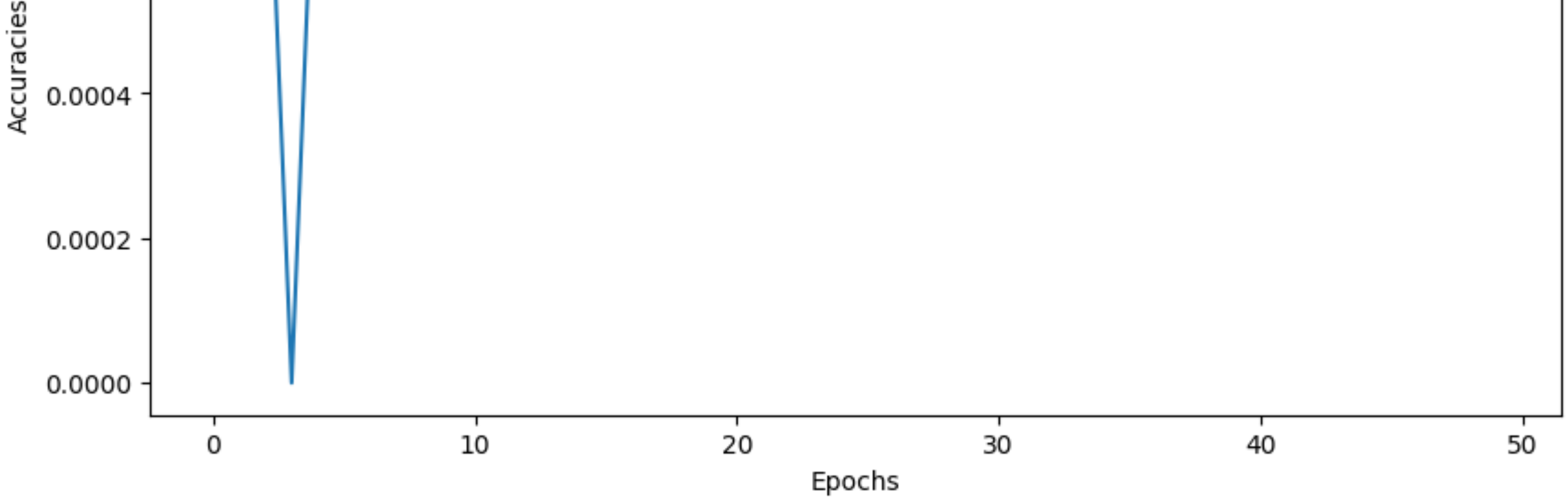
```
 0.009649025276305291,
 0.007730260491371155,
 0.007144193165004253,
 0.006479351315647364,
 0.007197853643447161,
 0.006324528716504574,
 0.005895284470170736,
 0.005587760824710131,
 0.005545112770050764,
 0.005054568871855736,
 0.004961862228810787,
 0.005322640761733055,
 0.004741175100207329,
 0.0053269946947693825,
 0.004431688692420721,
 0.004269062075763941,
 0.004498624708503485,
 0.004016960971057415,
 0.0036271694116294384,
 0.00423276424407959,
 0.004385589621961117,
 0.0038776553701609373,
 0.0036004732828587294]
```

```
# Plotting Loss vs Epochs
plt.figure(figsize=(10, 7))
plt.plot(history.history["loss"])
plt.xlabel("Epochs")
plt.ylabel("Losses")
plt.title("Simple RNN model, Loss vs Epoch")
plt.show()

```




```
# Plotting Accuracy vs Epochs
plt.figure(figsize=(10, 5))
plt.plot(history.history["accuracy"])
plt.xlabel("Epochs")
plt.ylabel("Accuracies")
plt.title("Vanilla RNN model, Accuracy vs Epoch")
plt.show()

```





Model prediction for train data

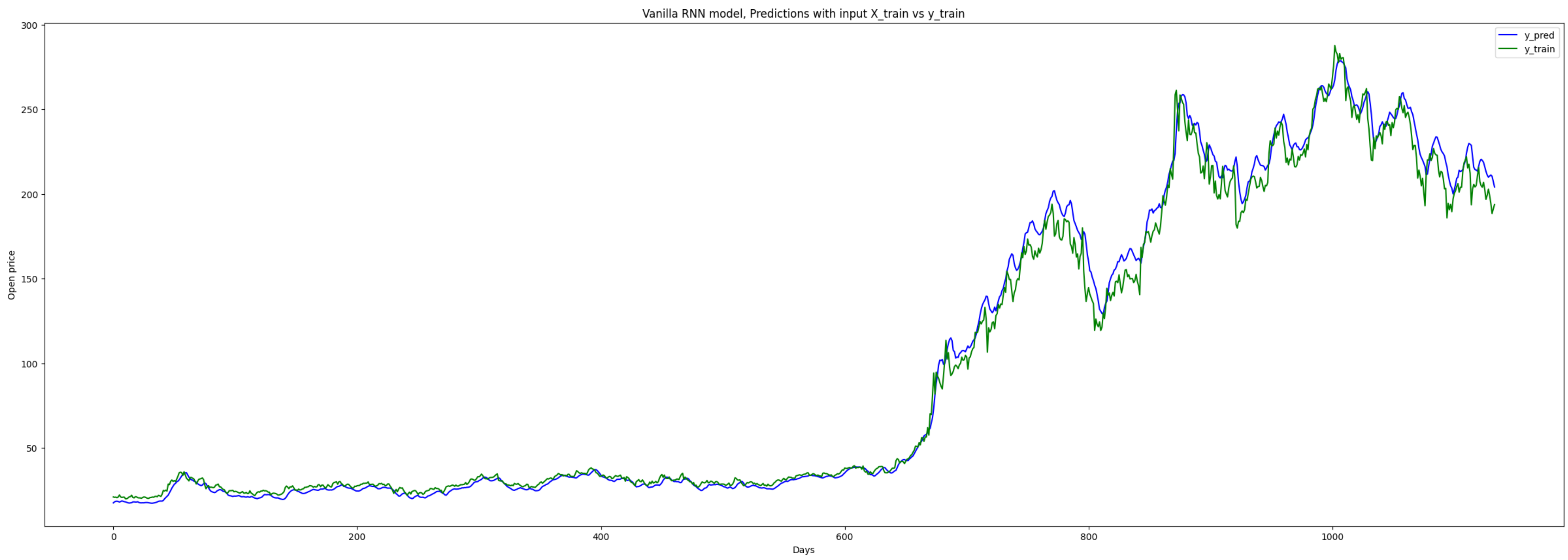
```
y_pred = regressor.predict(X_train)  # predictions
y_pred = scaler.inverse_transform(y_pred)  # scaling back from 0-1 to original
y_pred.shape

```

```
36/36 [==============================] - 1s 13ms/step
(1134, 1)
```

```
y_train = scaler.inverse_transform(y_train)  # scaling back from 0-1 to original
y_train.shape

```

```
(1134, 1)
```

```
# visualisation
plt.figure(figsize=(30, 10))
plt.plot(y_pred, color="b", label="y_pred")
plt.plot(y_train, color="g", label="y_train")
plt.xlabel("Days")
plt.ylabel("Open price")
plt.title("Vanilla RNN model, Predictions with input X_train vs y_train")
plt.legend()
plt.show()

```



Creating Test Dataset from Validation Data

```
validation_data = data[length_train:].iloc[:, :2]
validation_data["Date"] = pd.to_datetime(
    validation_data["Date"]
```

```
    validation_data["Date"]
)  # converting to date time object
validation_data
```

	Date	Open
1184	2015-03-13	188.949997
1185	2015-03-16	192.000000
1186	2015-03-17	195.429993
1187	2015-03-18	194.960007
1188	2015-03-19	202.000000
...	...	...
1687	2017-03-13	244.820007
1688	2017-03-14	246.110001
1689	2017-03-15	257.000000
1690	2017-03-16	262.399994
1691	2017-03-17	264.000000

508 rows × 2 columns

```
dataset_validation = (
    validation_data.Open.values
)  # getting "open" column and converting to array
dataset_validation = np.reshape(
    dataset_validation, (-1, 1)
)  # converting 1D to 2D array
scaled_dataset_validation = scaler.fit_transform(
    dataset_validation
)  # scaling open values to between 0 and 1
print("Shape of scaled validation dataset :", scaled_dataset_validation.shape)
```

```
Shape of scaled validation dataset : (508, 1)
```

```
# Creating X_test and y_test
X_test = []
y_test = []

for i in range(time_step, length_validation):
    X_test.append(scaled_dataset_validation[i - time_step : i, 0])
    y_test.append(scaled_dataset_validation[i, 0])
```

```
# Converting to array
X_test, y_test = np.array(X_test), np.array(y_test)
```

```
print("Shape of X_test before reshape :", X_test.shape)
print("Shape of y_test before reshape :", y_test.shape)
```

```
Shape of X_test before reshape : (458, 50)
Shape of y_test before reshape : (458,)
```

```
X_test = np.reshape(
    X_test, (X_test.shape[0], X_test.shape[1], 1)
)  # reshape to 3D array
y_test = np.reshape(y_test, (-1, 1))  # reshape to 2D array
```

```
print("Shape of X_test after reshape :", X_test.shape)
print("Shape of y_test after reshape :", y_test.shape)
```

```
Shape of X_test after reshape : (458, 50, 1)
Shape of y_test after reshape : (458, 1)
```

Evaluating with validation data

```
# predictions with X_test data
y_pred_of_test = regressor.predict(X_test)
# scaling back from 0-1 to original
y_pred_of_test = scaler.inverse_transform(y_pred_of_test)
print("Shape of y_pred_of_test :", y_pred_of_test.shape)
```

```
15/15 [==============================] - 0s 14ms/step
Shape of y_pred_of_test : (458, 1)
```

```
# visualisation
plt.figure(figsize=(30, 10))
plt.plot(y_pred_of_test, label="y_pred_of_test", c="orange")
plt.plot(scaler.inverse_transform(y_test), label="y_test", c="g")
plt.xlabel("Days")
plt.ylabel("Open price")
plt.title("Simple RNN model, Prediction with input X_test vs y_test")
plt.legend()
plt.show()

```

Future Price Prediction

```
X_input = data.iloc[
    -time_step:
].Open.values  # getting last 50 rows and converting to array
X_input = scaler.fit_transform(
    X_input.reshape(-1, 1)
)  # converting to 2D array and scaling
X_input = np.reshape(X_input, (1, 50, 1))  # reshaping : converting to 3D array
print("Shape of X_input :", X_input.shape)
X_input

```

```
Shape of X_input : (1, 50, 1)
array([[[0.        ],
        [0.00946363],
        [0.04731867],
        [0.10354429],
        [0.04917441],
        [0.04898868],
        [0.06643166],
        [0.19075893],
        [0.18983106],
        [0.38652815],
        [0.35331247],
        [0.36054942],
        [0.43755803],
        [0.57320468],
        [0.5171645 ],
        [0.46316584],
        [0.48450549],
        [0.42345532],
        [0.49415485],
        [0.40675446],
        [0.47300067],
        [0.45611434],
        [0.58953431],
```

```
        [0.57394708],
        [0.7390982 ],
        [0.80478773],
        [0.82241588],
        [0.97624793],
        [0.99424758],
        [0.94971253],
        [0.73074763],
        [0.90981655],
        [1.        ],
        [0.69734648],
        [0.48691791],
        [0.40359993],
        [0.32974585],
        [0.51512331],
        [0.43217682],
        [0.45128979],
        [0.39877539],
        [0.47318612],
        [0.38188907],
        [0.39357964],
        [0.36722971],
        [0.34143643],
        [0.36537397],
        [0.56745225],
        [0.66765626],
        [0.69734648]]])
```

```
vanilla_RNN_prediction = scaler.inverse_transform(regressor.predict(X_input))
print(
    "Vanilla RNN, Open price prediction for 3/18/2017      :",
    vanilla_RNN_prediction[0, 0],
)

```

```
1/1 [==============================] - 0s 34ms/step
Vanilla RNN, Open price prediction for 3/18/2017      : 257.36172
```