1.Loading Data

```
# Imports
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")
```

```
data = pd.read_csv("./Tesla.csv - Tesla.csv.csv")
```

```
data.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001

```
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
```

2.Spliting Data as Train and Validation

```
length_data = len(data)      # rows that data has
split_ratio = 0.8            # %80 train + %20 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 : 1354
Validation data lenth : 338
```

```
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

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

	Date	Open
1354	2015-11-12	217.850006
1355	2015-11-13	212.949997
1356	2015-11-16	206.089996
1357	2015-11-17	215.199997
1358	2015-11-18	214.500000
...	...	...
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

338 rows × 2 columns

3.Creating Train Dataset from Train split

- We will get Open column as our dataset
- Dataset to be converted to array by adding `.values`

```
dataset_train = train_data.Open.values
dataset_train.shape
```

```
(1354,)
```

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

```
(1354, 1)
```

4.Normalization / Feature Scaling

- Dataset values will be in between 0 and 1 after scaling

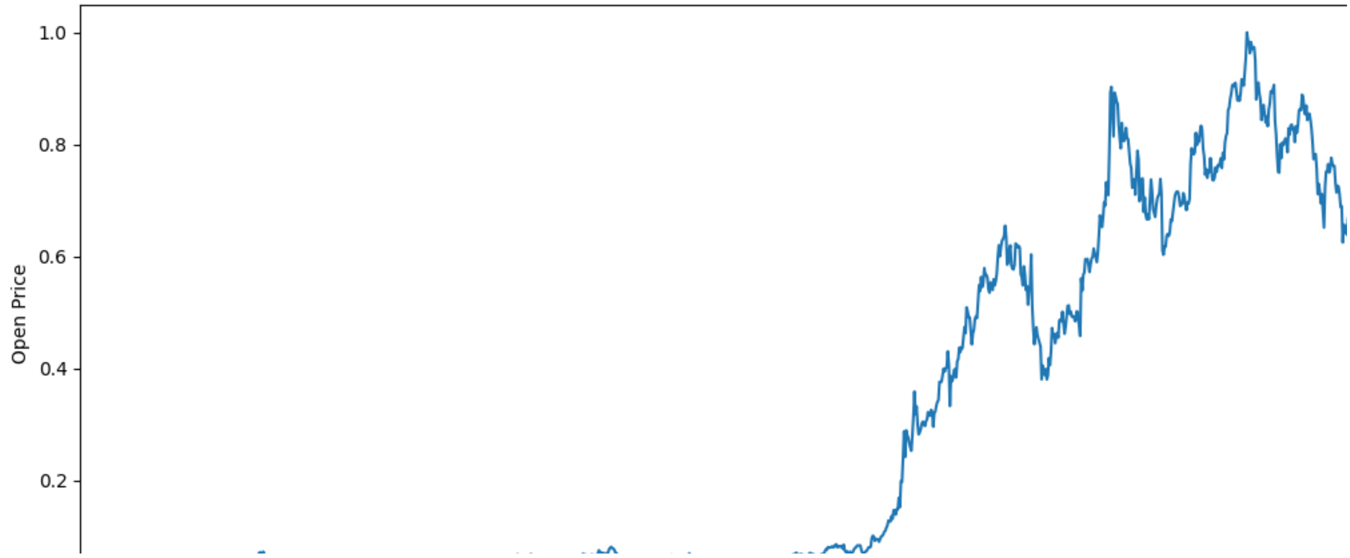
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0,1))
```

```
# scaling dataset
dataset_train_scaled = scaler.fit_transform(dataset_train)

dataset_train_scaled.shape
```

```
(1354, 1)
```

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



5.Creating X_train and y_train from Train data

image.png

- We have train data composed of stock open prices over days
- So, it has 1184 prices corresponding 1184 days
- My aim is to predict the open price of the next day.
- I can use a time step of 50 days.
- I will pick first 50 open prices (0 to 50), 1st 50 price will be in X_train data
- Then predict the price of 51th day; and 51th price will be in y_train data
- Again, i will pick prices from 1 to 51, those will be in X_train data
- Then predict the next days price, 52nd price will be in y_train data

```
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 : (1304, 50)
Shape of y_train before reshape : (1304,)
```

Reshape

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

```
Shape of X_train after reshape : (1304, 50, 1)
Shape of y_train after reshape : (1304, 1)
```

- Shape of X_train : 1134 x 50 x 1
- That means we have 1134 rows, each row has 50 rows and 1 column
- Lets check the first row: it has 50 rows (open prices of 49 days)

```
X_train[0]
```

```
array([[0.01053291],
       [0.03553936],
       [0.03262991],
       [0.02526425],
       [0.01421574],
       [0.00095754],
       [0.        ],
       [0.00530328],
       [0.00666594],
       [0.00460354],
       [0.00662911],
       [0.01399478],
       [0.01679373],
       [0.01926123],
       [0.02102899],
       [0.01664641],
       [0.01605716],
       [0.01859832],
       [0.01973999],
       [0.01756712],
       [0.0162413 ],
       [0.01705153],
       [0.01495231],
       [0.01605716],
       [0.01789858],
       [0.02139727],
       [0.01988731],
       [0.01458403],
       [0.01384746],
       [0.01292675],
       [0.00939123],
       [0.0061135 ],
       [0.00751299],
       [0.00850735],
       [0.01038559],
       [0.01270578],
       [0.00883881],
       [0.00924392],
       [0.01086436],
       [0.01145362],
       [0.01112216],
       [0.01381063],
       [0.01329503],
       [0.0131109 ],
       [0.01296358],
       [0.01281627],
       [0.0155784 ],
       [0.01741981],
       [0.01646228],
       [0.01664641]])
```

- Check the first item in y_train
- It is the price of 50th day

```
y_train[0]
```

```
array([0.01789858])
```

6.Creating RNN model

```
# 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 = "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))

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

# fitting the RNN
history = regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

```
Epoch 91/100
41/41 [==============================] - 8s 186ms/step - loss: 0.0024 - accuracy: 7.6687e-04
Epoch 92/100
41/41 [==============================] - 7s 159ms/step - loss: 0.0025 - accuracy: 7.6687e-04
Epoch 93/100
41/41 [==============================] - 8s 185ms/step - loss: 0.0023 - accuracy: 7.6687e-04
Epoch 94/100
41/41 [==============================] - 6s 153ms/step - loss: 0.0030 - accuracy: 7.6687e-04
Epoch 95/100
41/41 [==============================] - 8s 188ms/step - loss: 0.0026 - accuracy: 7.6687e-04
Epoch 96/100
41/41 [==============================] - 6s 152ms/step - loss: 0.0024 - accuracy: 7.6687e-04
Epoch 97/100
41/41 [==============================] - 8s 187ms/step - loss: 0.0030 - accuracy: 7.6687e-04
Epoch 98/100
41/41 [==============================] - 7s 163ms/step - loss: 0.0030 - accuracy: 7.6687e-04
Epoch 99/100
41/41 [==============================] - 7s 173ms/step - loss: 0.0025 - accuracy: 7.6687e-04
Epoch 100/100
41/41 [==============================] - 7s 182ms/step - loss: 0.0028 - accuracy: 7.6687e-04
```

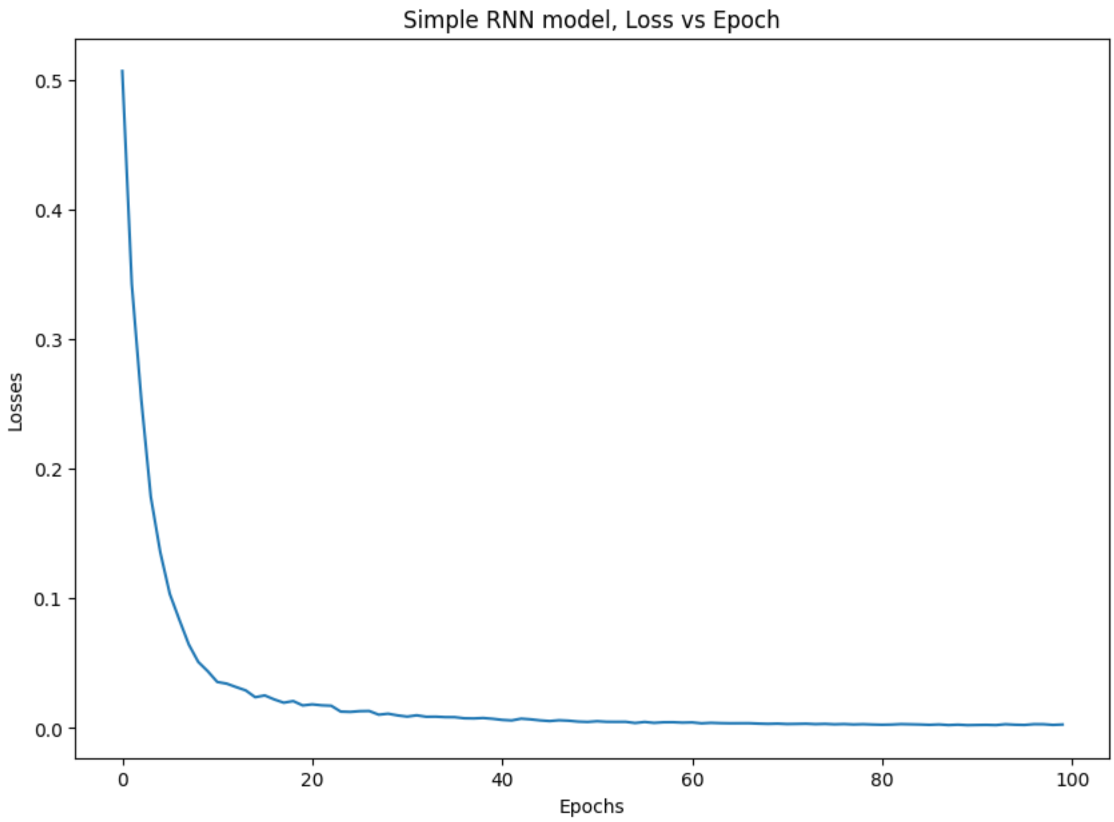
7.Evaluating Model

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

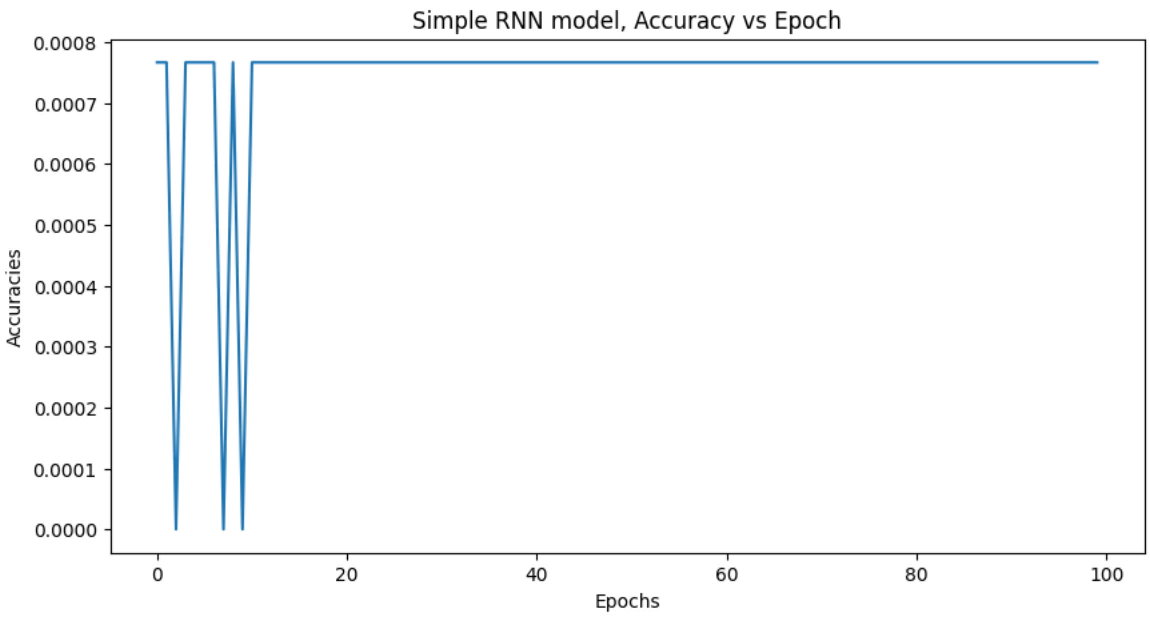
```
[0.5071741342544556,
 0.3427289128303528,
 0.2536590099334717,
 0.1786496788263321,
 0.13574403524398804,
 0.10361679643392563,
 0.0837988406419754,
 0.06458805501461029,
 0.05107660964131355,
 0.043899938464164734,
 0.03563394397497177,
 0.034298188984394073,
 0.0316346602563858,
 0.029088357463479042,
 0.02382340095937252,
 0.025215432047843933,
 0.022137243300676346,
 0.019610615447163582,
 0.020859327167272568,
 0.017494171857833862,
 0.018304765224456787,
 0.017577631399035454,
 0.017270535230636597,
 0.012743714265525341,
 0.01250794343650341,
 0.01307386253029108,
 0.013202602975070477,
 0.010362917557358742,
 0.011119531467556953,
 0.009787649847567081,
 0.008862764574587345,
 0.009876001626253128,
 0.008742311969399452,
 0.008808339945971966,
 0.008462285622954369,
 0.0084294518455863,
 0.007585796061903238,
 0.007473845966160297,
 0.007793285883963108,
 0.007194334641098976,
 0.006333345081657171,
 0.005909149069339037,
 0.007298567332327366,
 0.006748590152710676,
 0.005942140705883503,
 0.005422557238489389,
 0.006035550031810999,
 0.005699974950402975,
 0.005025919061154127,
 0.004770949948579073,
 0.005302132572978735,
 0.0048673199489712715,
 0.004832082893699408,
 0.004854266997426748,
 0.0039535327814519405,
 0.004750555846840143,
 0.004058447200804949,
 0.004516130778938532,
```

```python
# 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("Simple RNN model, Accuracy vs Epoch")
plt.show()
```



Model predictions for train data

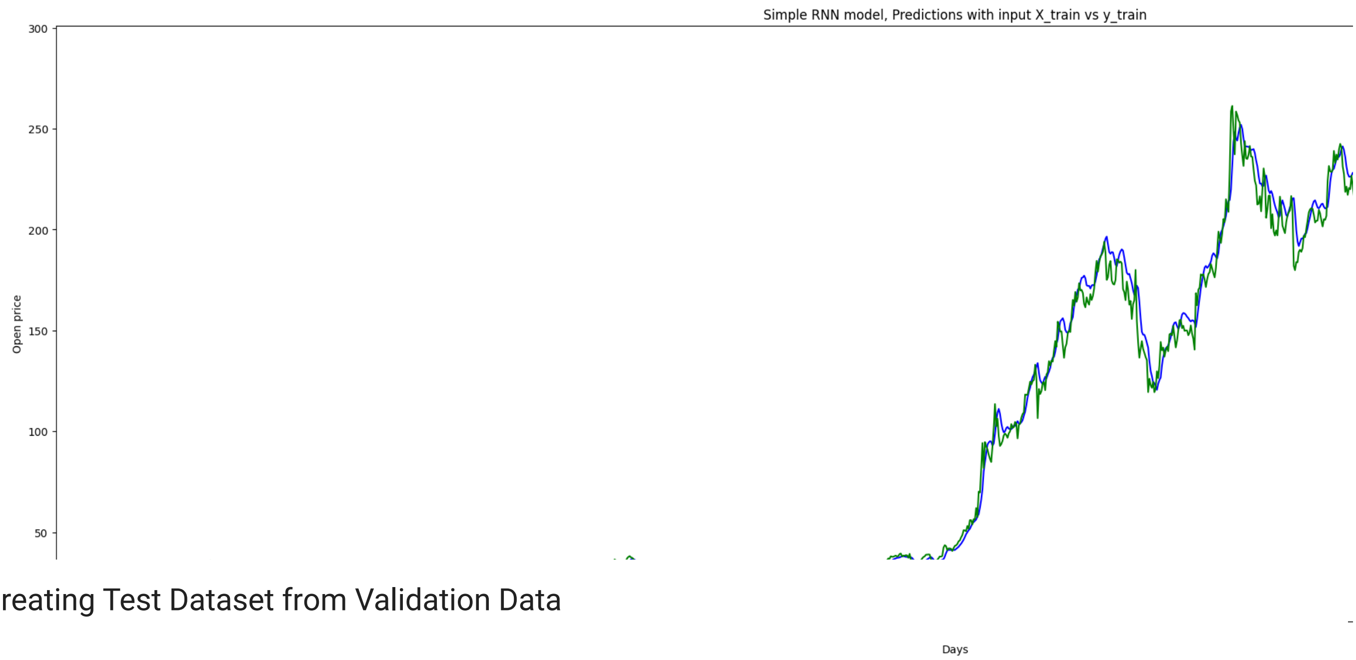
```
y_pred = regressor.predict(X_train)  # predictions
y_pred = scaler.inverse_transform(y_pred) # scaling back from 0-1 to original
y_pred.shape
```

```
41/41 [==============================] - 2s 26ms/step
(1304, 1)
```

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

```
(1304, 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("Simple RNN model, Predictions with input X_train vs y_train")
plt.legend()
plt.show()
```



8.Creating Test Dataset from Validation Data

Converting array and scaling

```
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 : (338, 1)
```

Creating X_test and y_test

```
# 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

```
# 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 : (288, 50)
Shape of y_test before reshape : (288,)
```

Reshape

```
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 : (288, 50, 1)
Shape of y_test after reshape : (288, 1)
```

+ Code + Text

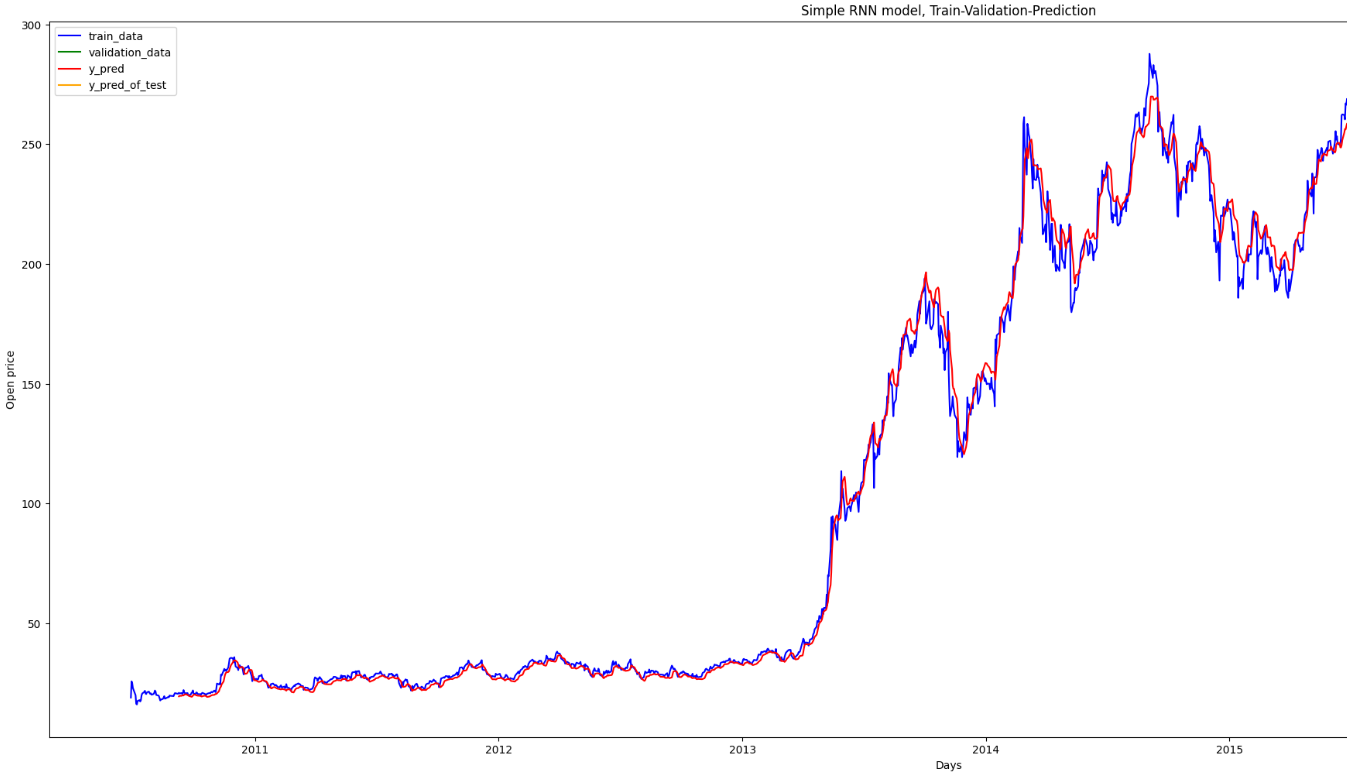
9.Evaluating with Validation Data

```
# 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)
```

```
9/9 [==============================] - 0s 17ms/step
Shape of y_pred_of_test : (288, 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()
```


```
# Visualisation
plt.subplots(figsize =(30,12))
plt.plot(train_data.Date, train_data.Open, label = "train_data", color = "b")
plt.plot(validation_data.Date, validation_data.Open, label = "validation_data", color = "g")
plt.plot(train_data.Date.iloc[time_step:], y_pred, label = "y_pred", color = "r")
plt.plot(validation_data.Date.iloc[time_step:], y_pred_of_test, label = "y_pred_of_test", color = "orange")
plt.xlabel("Days")
plt.ylabel("Open price")
plt.title("Simple RNN model, Train-Validation-Prediction")
plt.legend()
plt.show()
```



Simple RNN model, Train-Validation-Prediction
train_data
validation_data
y_pred
y_pred_of_test
Open price
Days
300
250
200
150
100
50
2011
2012
2013
2014
2015