```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import datetime
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import SimpleRNN
from sklearn.metrics import r2_score
```

Loading the Dataset

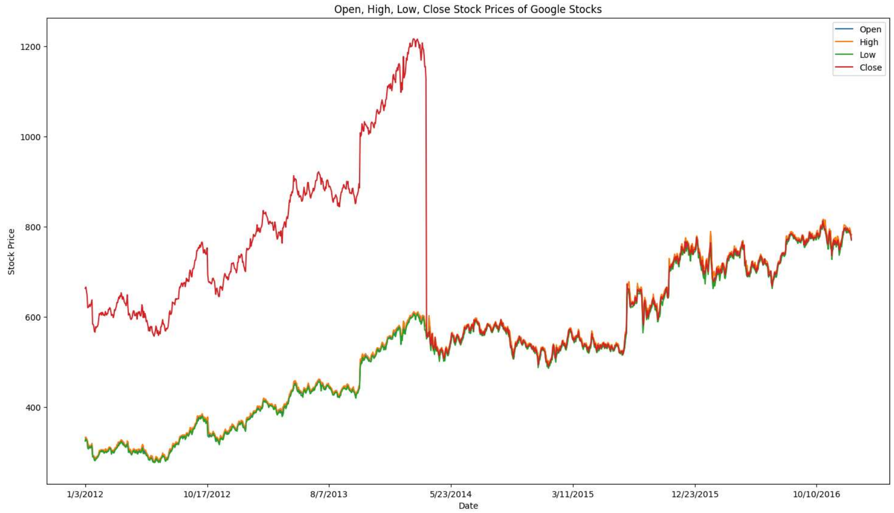
```
data = pd.read_csv('/content/Google_Stock_Price_Train.csv',thousands=',')
print(data.head(10))
data.shape
```

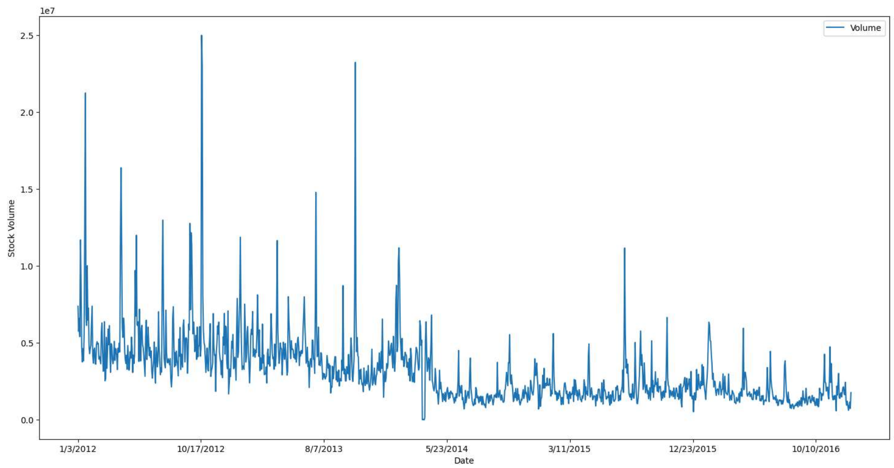
```
         Date    Open    High     Low   Close    Volume
0   1/3/2012  325.25  332.83  324.97  663.59   7380500
1   1/4/2012  331.27  333.87  329.08  666.45   5749400
2   1/5/2012  329.83  330.75  326.89  657.21   6590300
3   1/6/2012  328.34  328.77  323.68  648.24   5405900
4   1/9/2012  322.04  322.29  309.46  620.76  11688800
5  1/10/2012  313.70  315.72  307.30  621.43   8824000
6  1/11/2012  310.59  313.52  309.40  624.25   4817800
7  1/12/2012  314.43  315.26  312.08  627.92   3764400
8  1/13/2012  311.96  312.30  309.37  623.28   4631800
9  1/17/2012  314.81  314.81  311.67  626.86   3832800
(1258, 6)
```

Plotting the Dataset

```
ax1 = data.plot(x="Date", y=["Open", "High", "Low", "Close"],  figsize=(18,10),title='Open, High, Low, Close Stock Prices of Google Stoc
ax1.set_ylabel("Stock Price")

ax2 = data.plot(x="Date", y=["Volume"],  figsize=(18,9))
ax2.set_ylabel("Stock Volume")

```

```
Text(0, 0.5, 'Stock Volume')
```





Preprocessing Data

Checking for Missing values

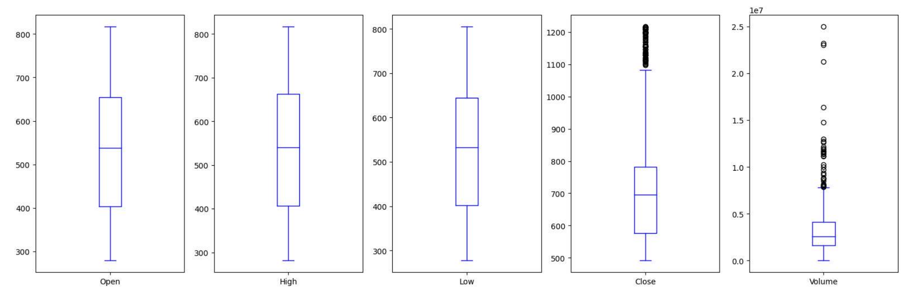
```
# Getting a summary of missing values for each field/attribute
print(data.isnull().sum())
```

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

We can see that there are no missing values in the dataset.

Handling Outliers

```
data[['Open','High','Low','Close','Volume']].plot(kind= 'box' ,layout=(1,5),subplots=True, sharex=False, sharey=False, figsize=(20,6),co
plt.show()
```



We can see that there are outliers in the Close and Volume attributes. However, we will not remove any outliers here since there are only a limited number of datapoints (less than 1300) and if we remove outliers, the dataset will become even smaller.

Feature Encoding

When we check the 1st 10 records of the dataset, we could see that all the data in the dataset are numerical data, and there are no categorical data that needs encoding. Therefore, no feature encoding process was carried out on this dataset.

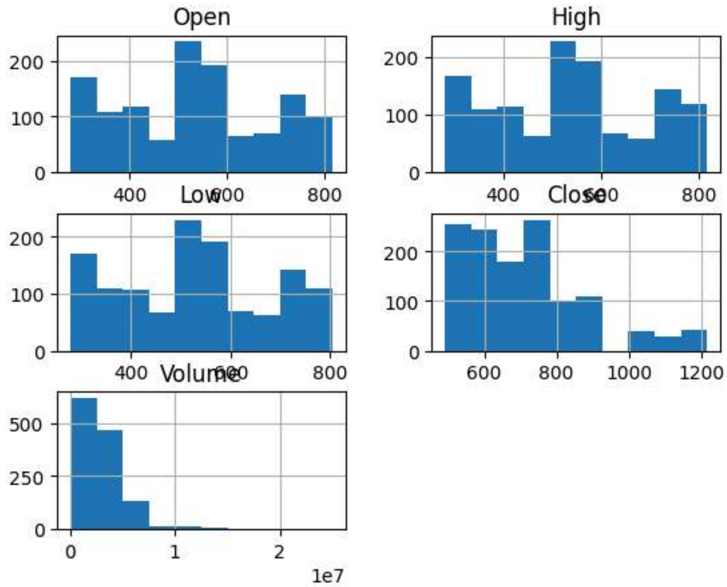
Feature Scaling

MinMaxScaler, which is said to be better to be used with time-series data, was used on this dataset for the feature scaling purposes

The following set of graphs show the attribute histogram before scaling

```
data.hist()
```

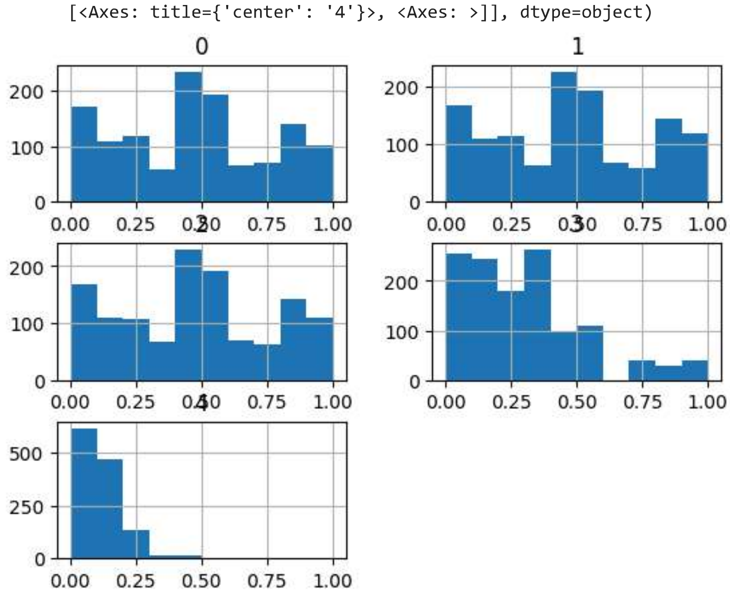
```
array([[<Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>],
       [<Axes: title={'center': 'Volume'}>, <Axes: >]], dtype=object)
```




```
scaler = MinMaxScaler()
data_without_date = data[['Open','High','Low','Close','Volume']]
data_scaled = pd.DataFrame(scaler.fit_transform(data_without_date))
```

```
data_scaled.hist()
```

```
array([[<Axes: title={'center': '0'}>, <Axes: title={'center': '1'}>],
       [<Axes: title={'center': '2'}>, <Axes: title={'center': '3'}>],
       [<Axes: title={'center': '4'}>, <Axes: >]], dtype=object)
```



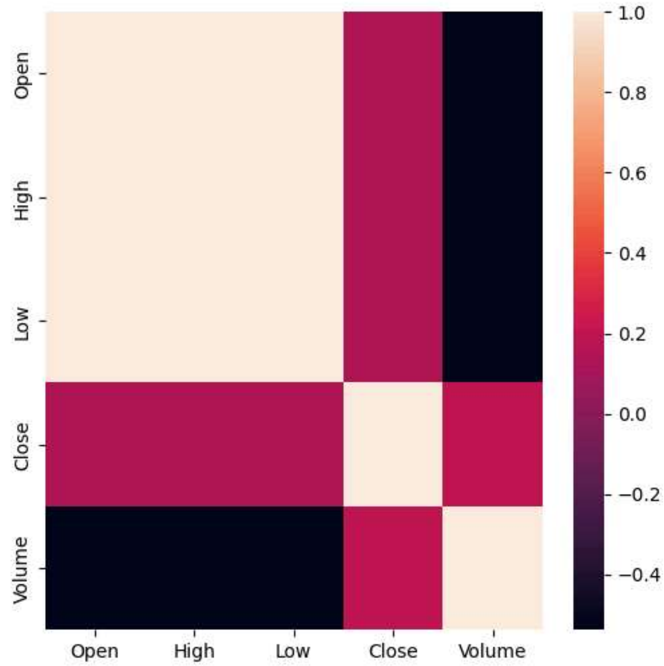
Feature Engineering

Drawing the Correlation Matrix

Correlation Coefficient checking mechanism checks the relationship between the different features with the predicting attribute.

```
plt.figure(figsize=(6,6))
sns.heatmap(data.corr())
```

```
<ipython-input-9-373595b0d5cd>:2: FutureWarning: The default value of numeric_only in D
  sns.heatmap(data.corr())
<Axes: >
```



We can see from the correlation matrix that the features Open, High, and Low, all are highly correlated with each other. Therefore, it is sufficient to consider only one feature from the above 3 features and remove the rest.

Furthermore, I am removing the Close field since it contains many abnormal data values, which is like a contexual outlier.

```
#to select final features, we remove open and low because of their perfect correlation (1.0) with High
#so we choose High from open low high and also we drop Close because it has a lot of abnormal data values
data_scaled=data_scaled.drop([0,2,3], axis=1)
data_scaled
```

	1	4
0	0.096401	0.295258
1	0.098344	0.229936
2	0.092517	0.263612
3	0.088819	0.216179
4	0.076718	0.467797
...	...	...
1253	0.955292	0.024650
1254	0.964853	0.031286
1255	0.958074	0.045891
1256	0.942574	0.029491
1257	0.936691	0.070569

1258 rows × 2 columns

Next steps: Generate code with `data_scaled` View recommended plots

Developing the RNN Model

When developing an RNN Model, we have to reshape the data that we are feeding into the model. To do that, first we have to find a pattern in the data available and define the number of timesteps according to the pattern.

When considering the plots we have created for this we could not see a clear cut pattern/trend in the data by just visual inspection. Therefore, I have taken the following code snippet to split the data into a sequence by trying to identify any pattern available.

Since there are two features in our dataset after cleaning, I have used a split sequence method for a multivariate dataset.

```
def split_seq_multivariate(sequence, n_past, n_future):

    '''
    n_past ==> no of past observations
    n_future ==> no of future observations
    '''
    x, y = [], []
    for window_start in range(len(sequence)):
        past_end = window_start + n_past
        future_end = past_end + n_future
        if future_end > len(sequence):
            break
        # slicing the past and future parts of the window
        past   = sequence[window_start:past_end, :]
        future = sequence[past_end:future_end, -1]
        x.append(past)
        y.append(future)

    return np.array(x), np.array(y)
```

In RNN, since we are dealing with time series data, we have to specify how many past data points we will be considering when generating the sequence.

In here, i have taken 60 past data points (time steps) when generating the data sequences.

```
# specify the window size
n_steps = 60

data_scaled = data_scaled.to_numpy()
data_scaled.shape

```

```
(1258, 2)
```

Next, I am using the split_seq_multivariate function to split the dataset into sequences.

```
# split into samples
X, y = split_seq_multivariate(data_scaled, n_steps,1)
```

```
# X is in the shape of [samples, timesteps, features]
print(X.shape)
print(y.shape)

# make y to the shape of [samples]
y=y[:,0]
y.shape

```

```
(1198, 60, 2)
(1198, 1)
(1198,)
```

Splitting the Data

In this step, I will be splitting the sequenced data into train and test sections with 0.2 test size.

Furthermore, i have split the training set again to train and validation data. I have not used the test data to do the validation because validation data are used to fine tune the model, and if i used the testing data for validation purposes, then those data will be already seen by the model when trying to predict them later.

Ref: https://machinelearningmastery.com/difference-test-validation-datasets/

```
# split into train/test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=50)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

```

```
(958, 60, 2) (240, 60, 2) (958,) (240,)
```

```
# further dividing the training set into train and validation data
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.2,random_state=30)

print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

```
(766, 60, 2) (192, 60, 2) (766,) (192,)
```

Define Model

```
# Define simple RNN
model_rnn = Sequential()
model_rnn.add(SimpleRNN(612, input_shape=(n_steps,2))) # Default activation='tanh',recurrent_activation='sigmoid' in keras
model_rnn.add(Dense(50, activation='relu'))
model_rnn.add(Dense(50, activation='relu'))
model_rnn.add(Dense(30, activation='relu'))
model_rnn.add(Dense(1))
```

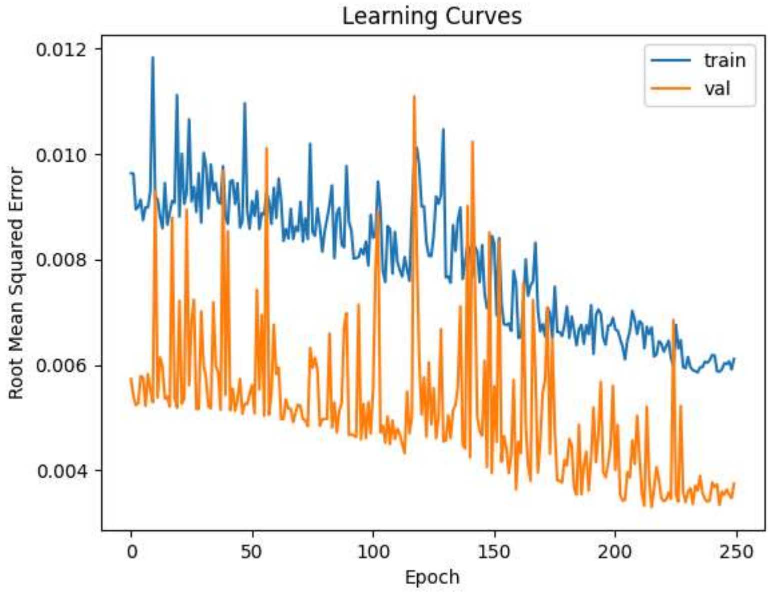
```
model_rnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 612)               376380

 dense (Dense)               (None, 50)                30650

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 30)                1530

 dense_3 (Dense)             (None, 1)                 31

=================================================================
Total params: 411141 (1.57 MB)
Trainable params: 411141 (1.57 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# compile the model
model_rnn.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
# fit the model
history = model_rnn.fit(X_train, y_train, epochs=250, batch_size=32, verbose=0, validation_data=(X_val, y_val))  # has used mini batch l
```

```
from matplotlib import pyplot
# plot learning curves
pyplot.title('Learning Curves')
pyplot.xlabel('Epoch')
pyplot.ylabel('Root Mean Squared Error')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
pyplot.show()
```




```
1 # evaluate the model
2 mse, mae = model_rnn.evaluate(X_test, y_test, verbose=0)
3 print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, np.sqrt(mse), mae))
```

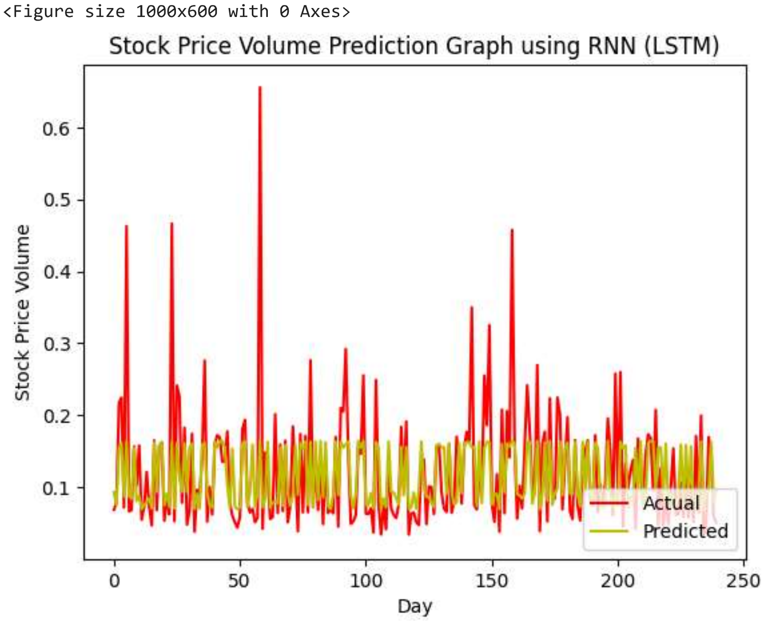
```
MSE: 0.004, RMSE: 0.067, MAE: 0.041
```

```
1 # predicting y_test values
2 print("Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence): ",X_test.shape)
3 predicted_values = model_rnn.predict(X_test)
4 print("Shape of Predicted Values: ",predicted_values.shape)
```

```
Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence):  (240, 60, 2)
8/8 [==============================] - 1s 49ms/step
Shape of Predicted Values:  (240, 1)
```

```
1 plt.plot(y_test,c = 'r')
2 plt.plot(predicted_values,c = 'y')
3 plt.xlabel('Day')
4 plt.ylabel('Stock Price Volume')
5 plt.title('Stock Price Volume Prediction Graph using RNN (LSTM)')
6 plt.legend(['Actual','Predicted'],loc = 'lower right')
7 plt.figure(figsize=(10,6))
```

```
<Figure size 1000x600 with 0 Axes>
```




```
<Figure size 1000x600 with 0 Axes>
```

```
# evaluating using R squared
R_square = r2_score(y_test, predicted_values)

print(R_square)
```

```
0.3328745729611804
```

Why LSTM over RNN?

- To solve vanishing and exploding gradients problem
- To tackle overfitting in RNN
- To get the Forget Gate Mechanism of LSTM for better predictions
- To Maintain cell state

```
# define LSTM model (LSTM is a special type of RNN)
model_lstm = Sequential()
model_lstm.add(LSTM(612, input_shape=(n_steps,2))) # Default LSTM activation='tanh',recurrent_activation='sigmoid' in keras
model_lstm.add(Dense(50, activation='relu'))
model_lstm.add(Dense(50, activation='relu'))
model_lstm.add(Dense(30, activation='relu'))
model_lstm.add(Dense(1))
```

The following code line gives a summary of the model we have created, mentioning each layers information.

```
model_lstm.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 612)               1505520

 dense (Dense)               (None, 50)                30650

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 30)                1530

 dense_3 (Dense)             (None, 1)                 31

=================================================================
Total params: 1540281 (5.88 MB)
Trainable params: 1540281 (5.88 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

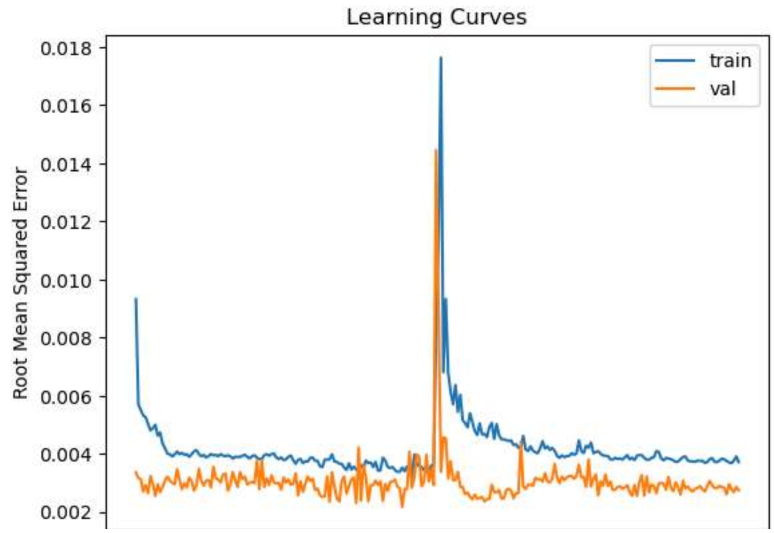
Compiling and Training the Model

```
# compile the model
model_lstm.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
# fit the model
history = model_lstm.fit(X_train, y_train, epochs=250, batch_size=32, verbose=0, validation_data=(X_val, y_val))  # has used mini batch
```

We can visualize the training and validation RMSE of the model are as follows.

```
from matplotlib import pyplot
# plot learning curves
pyplot.title('Learning Curves')
pyplot.xlabel('Epoch')
pyplot.ylabel('Root Mean Squared Error')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
pyplot.show()
```



Model Evaluation and Predictions

```
# evaluate the model
mse, mae = model_lstm.evaluate(X_test, y_test, verbose=0)
print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, np.sqrt(mse), mae))
```

```
MSE: 0.002, RMSE: 0.049, MAE: 0.031
```

```
# predicting y_test values
print("Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence): ",X_test.shape)
predicted_values = model_lstm.predict(X_test)
print("Shape of Predicted Values: ",predicted_values.shape)
```

```
Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence):  (240, 60, 2)
8/8 [==============================] - 0s 30ms/step
Shape of Predicted Values:  (240, 1)
```

```
plt.plot(y_test,c = 'r')
plt.plot(predicted_values,c = 'y')
```