Time Series Forecasting with Yahoo Stock Price LSTM

In [66]:
```
# Import Modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, LSTM
```

In [67]:
```
df2 = pd.read_csv("yahoo_stock.csv")
df2
```

Out[67]:

	Unnamed: 0	High	Low	Open	Close	Volume	Adj Close
0	2015-11-23	2095.610107	2081.389893	2089.409912	2086.590088	3.587980e+09	2086.590088
1	2015-11-24	2094.120117	2070.290039	2084.419922	2089.139893	3.884930e+09	2089.139893
2	2015-11-25	2093.000000	2086.300049	2089.300049	2088.870117	2.852940e+09	2088.870117
3	2015-11-26	2093.000000	2086.300049	2089.300049	2088.870117	2.852940e+09	2088.870117
4	2015-11-27	2093.290039	2084.129883	2088.820068	2090.110107	1.466840e+09	2090.110107
...	...	...	...	...	...	...	...
1820	2020-11-16	3628.510010	3600.159912	3600.159912	3626.909912	5.281980e+09	3626.909912
1821	2020-11-17	3623.110107	3588.679932	3610.310059	3609.530029	4.799570e+09	3609.530029
1822	2020-11-18	3619.090088	3567.330078	3612.090088	3567.790039	5.274450e+09	3567.790039
1823	2020-11-19	3585.219971	3543.840088	3559.409912	3581.870117	4.347200e+09	3581.870117
1824	2020-11-20	3581.229980	3556.850098	3579.310059	3557.540039	2.224339e+09	3557.540039

1825 rows × 7 columns

In [68]:
```
#rename
df2.rename(columns={"Unnamed: 0": "Date"}, inplace=True)
df2
```

Out[68]:

	Date	High	Low	Open	Close	Volume	Adj Close
0	2015-11-23	2095.610107	2081.389893	2089.409912	2086.590088	3.587980e+09	2086.590088
1	2015-11-24	2094.120117	2070.290039	2084.419922	2089.139893	3.884930e+09	2089.139893
2	2015-11-25	2093.000000	2086.300049	2089.300049	2088.870117	2.852940e+09	2088.870117
3	2015-11-26	2093.000000	2086.300049	2089.300049	2088.870117	2.852940e+09	2088.870117
4	2015-11-27	2093.290039	2084.129883	2088.820068	2090.110107	1.466840e+09	2090.110107
...	...	...	...	...	...	...	...
1820	2020-11-16	3628.510010	3600.159912	3600.159912	3626.909912	5.281980e+09	3626.909912
1821	2020-11-17	3623.110107	3588.679932	3610.310059	3609.530029	4.799570e+09	3609.530029
1822	2020-11-18	3619.090088	3567.330078	3612.090088	3567.790039	5.274450e+09	3567.790039
1823	2020-11-19	3585.219971	3543.840088	3559.409912	3581.870117	4.347200e+09	3581.870117
1824	2020-11-20	3581.229980	3556.850098	3579.310059	3557.540039	2.224339e+09	3557.540039

1825 rows × 7 columns

In [69]:
```
#dataframe info
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1825 entries, 0 to 1824
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1825 non-null   object
 1   High       1825 non-null   float64
 2   Low        1825 non-null   float64
 3   Open       1825 non-null   float64
 4   Close      1825 non-null   float64
 5   Volume     1825 non-null   float64
 6   Adj Close  1825 non-null   float64
dtypes: float64(6), object(1)
memory usage: 99.9+ KB
```

In [70]:
```
#re indexing a column
df2.set_index("Date", inplace = True)
```

In [71]:
```
# spliting the dataframe for training and test set
dataset2 = df2.values #Numpy Arrays
training_data_len = int(np.ceil( len(dataset2) * .75 ))
training_data_len
```

Out[71]: 1369

In [74]:
```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data2 = scaler.fit_transform(dataset2)

scaled_data2.shape
```

Out[74]: (1825, 6)

In [75]:
```
# Create the training data set
# Create the scaled training data set
train_data = scaled_data2[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i<= 61:
        print(x_train)
        print(y_train)
        print()
# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

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

```
[array([0.13819427, 0.13736603, 0.1367434 , 0.1367434 , 0.13690462,
       0.13690462, 0.13690462, 0.13719368, 0.14250781, 0.14300803,
       0.13229646, 0.13721037, 0.13721037, 0.13721037, 0.13530921,
       0.12609859, 0.1297006 , 0.1226521 , 0.11132359, 0.11132359,
       0.11132359, 0.09778823, 0.11499237, 0.12769386, 0.12749938,
       0.10773271, 0.10773271, 0.10773271, 0.0977771 , 0.10880549,
       0.12102901, 0.12249101, 0.12249101, 0.12249101, 0.12249101,
       0.1171602 , 0.1303843 , 0.12803856, 0.11981169, 0.11981169,
       0.11981169, 0.11981169, 0.10628183, 0.09724342, 0.09155691,
       0.07688756, 0.06303538, 0.06303538, 0.06303538, 0.04927766,
       0.05579798, 0.05743776, 0.04862171, 0.03873287, 0.03873287,
       0.03873287, 0.03873287, 0.03026139, 0.01622024, 0.02381891])]
[0.034380388952514405]

[array([0.13819427, 0.13736603, 0.1367434 , 0.1367434 , 0.13690462,
       0.13690462, 0.13690462, 0.13719368, 0.14250781, 0.14300803,
       0.13229646, 0.13721037, 0.13721037, 0.13721037, 0.13530921,
       0.12609859, 0.1297006 , 0.1226521 , 0.11132359, 0.11132359,
       0.11132359, 0.09778823, 0.11499237, 0.12769386, 0.12749938,
       0.10773271, 0.10773271, 0.10773271, 0.0977771 , 0.10880549,
       0.12102901, 0.12249101, 0.12249101, 0.12249101, 0.12249101,
       0.1171602 , 0.1303843 , 0.12803856, 0.11981169, 0.11981169,
       0.11981169, 0.11981169, 0.10628183, 0.09724342, 0.09155691,
       0.07688756, 0.06303538, 0.06303538, 0.06303538, 0.04927766,
       0.05579798, 0.05743776, 0.04862171, 0.03873287, 0.03873287,
       0.03873287, 0.03873287, 0.03026139, 0.01622024, 0.02381891]), array([0.13736603, 0.1367434 , 0.1367434 ,
0.13690462, 0.13690462,
       0.13690462, 0.13719368, 0.14250781, 0.14300803, 0.13229646,
       0.13721037, 0.13721037, 0.13721037, 0.13530921, 0.12609859,
       0.1297006 , 0.1226521 , 0.11132359, 0.11132359, 0.11132359,
       0.09778823, 0.11499237, 0.12769386, 0.12749938, 0.10773271,
       0.10773271, 0.10773271, 0.0977771 , 0.10880549, 0.12102901,
       0.12249101, 0.12249101, 0.12249101, 0.12249101, 0.1171602 ,
       0.1303843 , 0.12803856, 0.11981169, 0.11981169, 0.11981169,
       0.11981169, 0.10628183, 0.09724342, 0.09155691, 0.07688756,
       0.06303538, 0.06303538, 0.06303538, 0.04927766, 0.05579798,
       0.05743776, 0.04862171, 0.03873287, 0.03873287, 0.03873287,
       0.03873287, 0.03026139, 0.01622024, 0.02381891, 0.03438039])]
[0.034380388952514405, 0.034380388952514405]
```

Out[75]: ((1309, 60, 1), (1309,))

In [76]:
```
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data2[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset2[training_data_len:,0]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i,0])

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

x_test.shape,y_test.shape
```

Out[76]: ((456, 60, 1), (456,))

In [11]:
```
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
1309/1309 [==============================] - 56s 38ms/step - loss: 0.0010
```

Out[11]: <keras.src.callbacks.History at 0x29b90bc6cd0>

In [12]:
```
train_predictions = model.predict(x_train).flatten()
train_results = pd.DataFrame(data={'Train Predictions':train_predictions, 'Actuals':y_train})
train_results
```

```
41/41 [==============================] - 3s 47ms/step
```

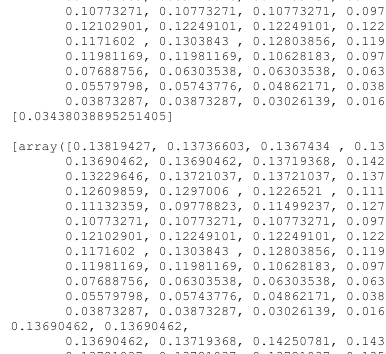
Out[12]:

	Train Predictions	Actuals
0	0.040098	0.034380
1	0.037793	0.034380
2	0.036758	0.034380
3	0.036506	0.032952
4	0.036526	0.033202
...	...	...
1304	0.603571	0.581787
1305	0.601676	0.602560
1306	0.602787	0.598464
1307	0.604767	0.601298
1308	0.607297	0.607052

1309 rows × 2 columns

In [13]:
```
import matplotlib.pyplot as plt
plt.plot(train_results['Train Predictions'][50:100])
plt.plot(train_results['Actuals'][50:100])
```

Out[13]: [<matplotlib.lines.Line2D at 0x29b962d6280>]



In [15]:
```
test_predictions = model.predict(x_test).flatten()
test_results = pd.DataFrame(data={'Test Predictions':test_predictions, 'Actuals':y_test})
test_results
```

```
15/15 [==============================] - 1s 46ms/step
```

Out[15]:

	Test Predictions	Actuals
0	0.610432	2927.010010
1	0.612704	2927.010010
2	0.614271	2927.010010
3	0.615269	2879.270020
4	0.612832	2898.790039
...	...	...
451	0.940698	3628.510010
452	0.944634	3623.110107
453	0.948307	3619.090088
454	0.951174	3585.219971
455	0.951173	3581.229980

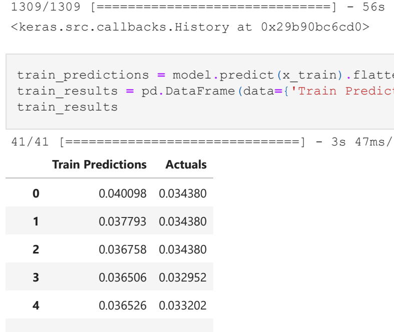
456 rows × 2 columns

- If a model has a high train accuracy but a low validation accuracy then the model is suffering from overfitting.

In [17]:
```
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5)
```

```
Epoch 1/5
41/41 [==============================] - 11s 170ms/step - loss: 3.3044e-04 - val_loss: 9971867.0000
Epoch 2/5
41/41 [==============================] - 5s 128ms/step - loss: 2.1499e-04 - val_loss: 9971873.0000
Epoch 3/5
41/41 [==============================] - 5s 127ms/step - loss: 1.9484e-04 - val_loss: 9971842.0000
Epoch 4/5
41/41 [==============================] - 6s 138ms/step - loss: 1.9461e-04 - val_loss: 9971875.0000
Epoch 5/5
41/41 [==============================] - 6s 145ms/step - loss: 1.7995e-04 - val_loss: 9971864.0000
```

In [18]:
```
#visualaziing learning rate
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'g', label='Validation loss')
plt.title('Training VS Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Regression Evaluation Metrics

- Mean Squared Error: mean_squared_error, MSE or mse
- Mean Absolute Error: mean_absolute_error, MAE, mae
- Mean Absolute Percentage Error: mean_absolute_percentage_error, MAPE, mape
- Cosine Proximity: cosine_proximity, cosine

In [52]:
```
# Build the LSTM model
model2 = Sequential()
model2.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model2.add(LSTM(64, return_sequences=False))
model2.add(Dense(25))
model2.add(Dense(1))

# Compile the model
model2.compile(loss='mean_squared_error',optimizer='adam', metrics=['mse'])

# Train the model
history = model2.fit(x_train, y_train,validation_data=(x_test, y_test), batch_size=1, epochs=5)
```

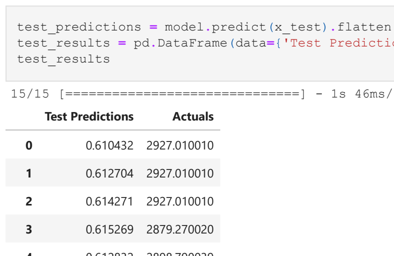
```
Epoch 1/5
1309/1309 [==============================] - 70s 49ms/step - loss: 9.1763e-04 - mse: 9.1763e-04 - val_loss: 997
1669.0000 - val_mse: 9971669.0000
Epoch 2/5
1309/1309 [==============================] - 67s 51ms/step - loss: 4.3587e-04 - mse: 4.3587e-04 - val_loss: 997
1674.0000 - val_mse: 9971674.0000
Epoch 3/5
1309/1309 [==============================] - 63s 48ms/step - loss: 2.6567e-04 - mse: 2.6567e-04 - val_loss: 997
1849.0000 - val_mse: 9971849.0000
Epoch 4/5
1309/1309 [==============================] - 52s 40ms/step - loss: 2.0569e-04 - mse: 2.0569e-04 - val_loss: 997
1791.0000 - val_mse: 9971791.0000
Epoch 5/5
1309/1309 [==============================] - 55s 42ms/step - loss: 2.4507e-04 - mse: 2.4507e-04 - val_loss: 997
1718.0000 - val_mse: 9971718.0000
```

In [ ]:
```
plt.plot(history.history['mse'])
plt.show()
```

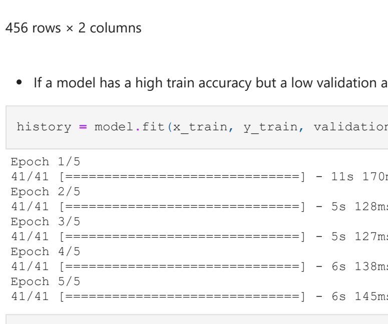
In [39]:
```
# Build the LSTM model
model3 = Sequential()
model3.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model3.add(LSTM(64, return_sequences=False))
model3.add(Dense(25))
model3.add(Dense(1))

# Compile the model
model3.compile(loss='mean_squared_error',optimizer='adam', metrics=['mape'])

# Train the model
history = model3.fit(x_train, y_train,validation_data=(x_test, y_test), batch_size=1, epochs=5)
```

```
Epoch 1/5
1309/1309 [==============================] - 68s 47ms/step - loss: 9.6990e-04 - mape: 27411.1973 - val_loss: 99
71836.0000 - val_mape: 99.9776
Epoch 2/5
1309/1309 [==============================] - 60s 46ms/step - loss: 4.2943e-04 - mape: 28334.3184 - val_loss: 99
72081.0000 - val_mape: 99.9788
Epoch 3/5
1309/1309 [==============================] - 61s 47ms/step - loss: 2.8759e-04 - mape: 34970.4805 - val_loss: 99
71910.0000 - val_mape: 99.9780
Epoch 4/5
1309/1309 [==============================] - 60s 46ms/step - loss: 2.4214e-04 - mape: 14395.8838 - val_loss: 99
71412.0000 - val_mape: 99.9755
Epoch 5/5
1309/1309 [==============================] - 60s 46ms/step - loss: 2.0790e-04 - mape: 18142.3320 - val_loss: 99
71832.0000 - val_mape: 99.9776
```

In [40]:
```
plt.plot(history.history['mape'])
plt.show()
```



In [41]:
```
#visualaziing learning rate
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'g', label='Validation loss')
plt.title('Training VS Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Mitigating Overfitting

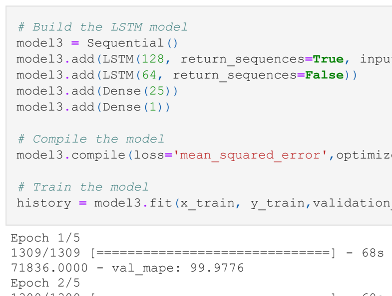
- Regularization: L1 and L2 regularization add a penalty to the loss function, discouraging the model from learning overly complex patterns in the data. In Keras, you can add regularization to your LSTM layers using the kernel_regularizer, bias_regularizer, and activity_regularizer parameters.
- Dropout: Dropout randomly sets a fraction of the input units to 0 during training, which helps prevent overfitting. In Keras, you can add dropout to your LSTM layers using the dropout and recurrent_dropout parameters.
- Early Stopping: Early stopping terminates training when the validation loss stops improving, preventing the model from overfitting. In Keras, you can implement early stopping using the EarlyStopping callback.
- Data Augmentation: Data augmentation generates new training samples by applying random transformations to the existing data, increasing the diversity of the training data and helping the model generalize better.

Using Regularization

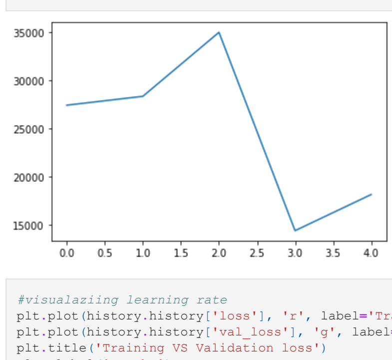
In [54]:
```
from tensorflow.keras.regularizers import L1L2

# Build the LSTM model
model4 = Sequential()
model4.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model4.add(LSTM(64, return_sequences=False, kernel_regularizer = L1L2(l1=0.01, l2=0.0)))
model4.add(Dense(25))
model4.add(Dense(1))

# Compile the model
model4.compile(loss='mean_squared_error',optimizer='adam', metrics=['mape'])

# Train the model
history = model4.fit(x_train, y_train,validation_data=(x_test, y_test), batch_size=1, epochs=5)
```

```
Epoch 1/5
1309/1309 [==============================] - 30s 21ms/step - loss: 0.7162 - mape: 33378.3242 - val_loss: 997275
2.0000 - val_mape: 99.9816
Epoch 2/5
1309/1309 [==============================] - 27s 21ms/step - loss: 0.0409 - mape: 54075.4297 - val_loss: 997284
2.0000 - val_mape: 99.9823
Epoch 3/5
1309/1309 [==============================] - 28s 21ms/step - loss: 0.0397 - mape: 53835.0547 - val_loss: 997270
5.0000 - val_mape: 99.9813
Epoch 4/5
1309/1309 [==============================] - 29s 22ms/step - loss: 0.0391 - mape: 43861.2305 - val_loss: 997228
7.0000 - val_mape: 99.9797
Epoch 5/5
1309/1309 [==============================] - 29s 22ms/step - loss: 0.0387 - mape: 47364.0195 - val_loss: 997232
6.0000 - val_mape: 99.9800
```

In [55]:
```
plt.plot(history.history['mape'])
plt.show()
```



In [64]:
```
#visualaziing learning rate
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'g', label='Validation loss')
plt.title('Training VS Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

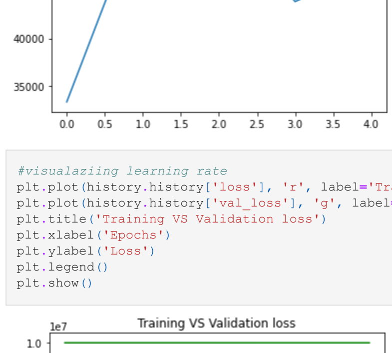


Using Dropout

In [77]:
```
# Build the LSTM model
model4 = Sequential()
model4.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model4.add(Dropout(0.2))
model4.add(LSTM(64, return_sequences=False))
model4.add(Dense(25))
model4.add(Dense(1))

# Compile the model
model4.compile(loss='mean_squared_error',optimizer='adam', metrics=['mape'])

# Train the model
history = model4.fit(x_train, y_train,validation_data=(x_test, y_test), batch_size=1, epochs=5)
```

```
Epoch 1/5
1309/1309 [==============================] - 75s 50ms/step - loss: 0.0010 - mape: 16406.7266 - val_loss: 997213
5.0000 - val_mape: 99.9788
Epoch 2/5
1309/1309 [==============================] - 61s 47ms/step - loss: 5.4207e-04 - mape: 19354.6953 - val_loss: 99
71964.0000 - val_mape: 99.9783
Epoch 3/5
1309/1309 [==============================] - 64s 49ms/step - loss: 3.7970e-04 - mape: 28994.2891 - val_loss: 99
71824.0000 - val_mape: 99.9775
Epoch 4/5
1309/1309 [==============================] - 61s 47ms/step - loss: 3.7998e-04 - mape: 21149.2520 - val_loss: 99
71472.0000 - val_mape: 99.9757
Epoch 5/5
1309/1309 [==============================] - 61s 47ms/step - loss: 3.4304e-04 - mape: 2390.1558 - val_loss: 997
1602.0000 - val_mape: 99.9763
```

In [78]:
```
#visualaziing learning rate
plt.plot(history.history['loss'], 'r', label='Training loss')
plt.plot(history.history['val_loss'], 'g', label='Validation loss')
plt.title('Training VS Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Using Early Stopping

In [79]:
```
# Build the LSTM model
model5 = Sequential()
model5.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model5.add(Dropout(0.2))
model5.add(LSTM(64, return_sequences=False))
model5.add(Dense(25))
model5.add(Dense(1))

# Compile the model
model5.compile(loss='mean_squared_error',optimizer='adam', metrics=['mape'])

# Train the model
from keras.callbacks import EarlyStopping
earlyStop=EarlyStopping(monitor="val_loss",verbose=2,mode='min',patience=3)
history = model5.fit(x_train, y_train,validation_data=(x_test, y_test), batch_size=1, epochs=10, verbose=2,call
```

```
Epoch 1/10
1309/1309 - 58s - loss: 0.0012 - mape: 26092.5625 - val_loss: 9972045.0000 - val_mape: 99.9786 - 58s/epoch - 44
ms/step
Epoch 2/10
1309/1309 - 58s - loss: 6.2666e-04 - mape: 41283.6445 - val_loss: 9972021.0000 - val_mape: 99.9785 - 58s/epoch
- 44ms/step
Epoch 3/10
1309/1309 - 57s - loss: 4.9135e-04 - mape: 25288.6074 - val_loss: 9971562.0000 - val_mape: 99.9762 - 57s/epoch
- 44ms/step
Epoch 4/10
1309/1309 - 57s - loss: 4.8616e-04 - mape: 20578.8340 - val_loss: 9971849.0000 - val_mape: 99.9776 - 57s/epoch
- 44ms/step
Epoch 5/10
1309/1309 - 52s - loss: 4.0259e-04 - mape: 4386.6582 - val_loss: 9971929.0000 - val_mape: 99.9781 - 52s/epoch -
40ms/step
Epoch 6/10
1309/1309 - 58s - loss: 3.5819e-04 - mape: 16999.6270 - val_loss: 9971967.0000 - val_mape: 99.9784 - 58s/epoch
- 45ms/step
Epoch 6: early stopping
```

In [81]:
```
test_predictions = model5.predict(x_test).flatten()
test_results = pd.DataFrame(data={'Test Predictions':test_predictions, 'Actuals':y_test})
test_results
```

```
15/15 [==============================] - 2s 34ms/step
```

Out[81]:

	Test Predictions	Actuals
0	0.585269	2927.010010
1	0.584005	2927.010010
2	0.582180	2927.010010
3	0.581022	2879.270020
4	0.569699	2898.790039
...	...	...
451	0.912085	3628.510010
452	0.921038	3623.110107
453	0.925913	3619.090088
454	0.926790	3585.219971
455	0.918600	3581.229980

456 rows × 2 columns