LSTM ON IMPUTED DATA.

Import Libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.float_format', lambda x: '%.4f' % x)
import seaborn as sns
sns.set_context("paper", font_scale=1.5)
sns.set_style('darkgrid')
import warnings
warnings.filterwarnings('ignore')
from time import time
import matplotlib.ticker as tkr
from scipy import stats
from statsmodels.tsa.stattools import adfuller
from sklearn import preprocessing
from statsmodels.tsa.stattools import pacf
%matplotlib inline
import math
import pickle
import keras
from keras.models import Sequential
from keras.models import model_from_json
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.callbacks import EarlyStopping
import pydot_ng as pydot
from keras.utils import plot_model
import time
```

```
Using TensorFlow backend.
```

Load the data

In [2]:

```
df = pd.read_csv('sub1.csv', header=None)
```

In [3]:

```
df.columns
```

Out[3]:

```
Int64Index([0, 1, 2, 3, 4], dtype='int64')
```

Here, we picked one column to represent the data since they are pretty much alike

In [4]:

```
df['2nd']=df[2]
```

In [5]:

```
df=df['2nd']
```

Feature extraction using timesteps

We are using the previous data values to determine the next one.

so the value at (t-1)s will be used to determine value at (t)s

we are employing 12 timesteps, so the previous 12 values will be used to determine the 13th values.

In [6]:

```
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)
```

Normalisation

Normalisation-this helps the model work better as they dont understand the context of the data, ages of people and heights of people are different context which the model cant understand. normalisation gives the values a particular to help the model understand the limits and context of the data.

In [7]:

```
dataset     = df.values #numpy.ndarray
dataset     = dataset.astype('float32')
dataset     = np.reshape(dataset, (-1, 1))
scaler      = MinMaxScaler(feature_range=(0, 1))
dataset     = scaler.fit_transform(dataset)
train_size  = int(len(dataset) * 0.80)
test_size   = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

we also performed some splitting activities to divide the data so we can test the model with a data that it has not seen.

In [8]:

```
look_back = 12
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test = create_dataset(test, look_back)
```

2D to 3D

LSTM input layer requires a 3D data. In DNN, 2D data was used as [samples,features]. it assumes the timesteps are the features but LSTM takes the timesteps into consideration. so the 3D data will now be [samples, time steps, features]

In [9]:

```
# reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test  = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

LSTM Model

In [10]:

```
model = Sequential()
model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 100)               45200     
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101       
=================================================================
Total params: 45,301
Trainable params: 45,301
Non-trainable params: 0
_________________________________________________________________
```

In [11]:

```
start         = time.time()

history       = model.fit(X_train, Y_train, epochs=20, batch_size=100, validation_data=(X_t
                    callbacks=[EarlyStopping(monitor='val_loss', patience=10)], verbose=1,

end           = time.time()
training_time = end-start
print("Training time: ", training_time)
```

```
Train on 1514347 samples, validate on 378577 samples
Epoch 1/20
1514347/1514347 [==============================] - 72s 47us/step - loss:
6.7808e-04 - val_loss: 2.8431e-04
Epoch 2/20
1514347/1514347 [==============================] - 71s 47us/step - loss:
4.1485e-04 - val_loss: 2.6339e-04
Epoch 3/20
1514347/1514347 [==============================] - 70s 46us/step - loss:
3.8989e-04 - val_loss: 2.4707e-04
Epoch 4/20
1514347/1514347 [==============================] - 70s 46us/step - loss:
3.7172e-04 - val_loss: 2.2172e-04
Epoch 5/20
1514347/1514347 [==============================] - 69s 46us/step - loss:
3.5408e-04 - val_loss: 2.0710e-04
Epoch 6/20
1514347/1514347 [==============================] - 69s 45us/step - loss:
3.4124e-04 - val_loss: 1.9627e-04
Epoch 7/20
1514347/1514347 [==============================] - 70s 46us/step - loss:
3.3655e-04 - val_loss: 2.0373e-04
Epoch 8/20
1514347/1514347 [==============================] - 74s 49us/step - loss:
3.3447e-04 - val_loss: 1.9491e-04
Epoch 9/20
1514347/1514347 [==============================] - 73s 48us/step - loss:
3.3620e-04 - val_loss: 1.9445e-04
Epoch 10/20
1514347/1514347 [==============================] - 72s 48us/step - loss:
3.3444e-04 - val_loss: 1.9848e-04
Epoch 11/20
1514347/1514347 [==============================] - 73s 49us/step - loss:
3.3335e-04 - val_loss: 1.9844e-04
Epoch 12/20
1514347/1514347 [==============================] - 74s 49us/step - loss:
3.3175e-04 - val_loss: 1.9465e-04
Epoch 13/20
1514347/1514347 [==============================] - 74s 49us/step - loss:
3.3151e-04 - val_loss: 1.9564e-04
Epoch 14/20
1514347/1514347 [==============================] - 72s 48us/step - loss:
3.3272e-04 - val_loss: 1.9818e-04
Epoch 15/20
1514347/1514347 [==============================] - 73s 48us/step - loss:
3.3008e-04 - val_loss: 1.9682e-04
Epoch 16/20
1514347/1514347 [==============================] - 74s 49us/step - loss:
3.3066e-04 - val_loss: 2.0213e-04
Epoch 17/20
```

```
1514347/1514347 [==============================] - 75s 49us/step - loss:
3.3051e-04 - val_loss: 1.9667e-04
Epoch 18/20
1514347/1514347 [==============================] - 71s 47us/step - loss:
3.3053e-04 - val_loss: 2.0579e-04
Epoch 19/20
1514347/1514347 [==============================] - 73s 48us/step - loss:
3.3053e-04 - val_loss: 1.9251e-04
Epoch 20/20
1514347/1514347 [==============================] - 75s 49us/step - loss:
3.3109e-04 - val_loss: 1.9731e-04
Training time:  1445.359840631485
```

Prediction

In [12]:

```
start              = time.time()
train_predict      = model.predict(X_train)
test_predict       = model.predict(X_test)
# invert predictions
train_predict      = scaler.inverse_transform(train_predict)
Y_train            = scaler.inverse_transform([Y_train])
test_predict       = scaler.inverse_transform(test_predict)
Y_test             = scaler.inverse_transform([Y_test])
end                =time.time()
prediction_time =end-start
print("Prediction time: ", prediction_time)
```

```
Prediction time:  40.55455040931702
```

Error evaluation

In [13]:

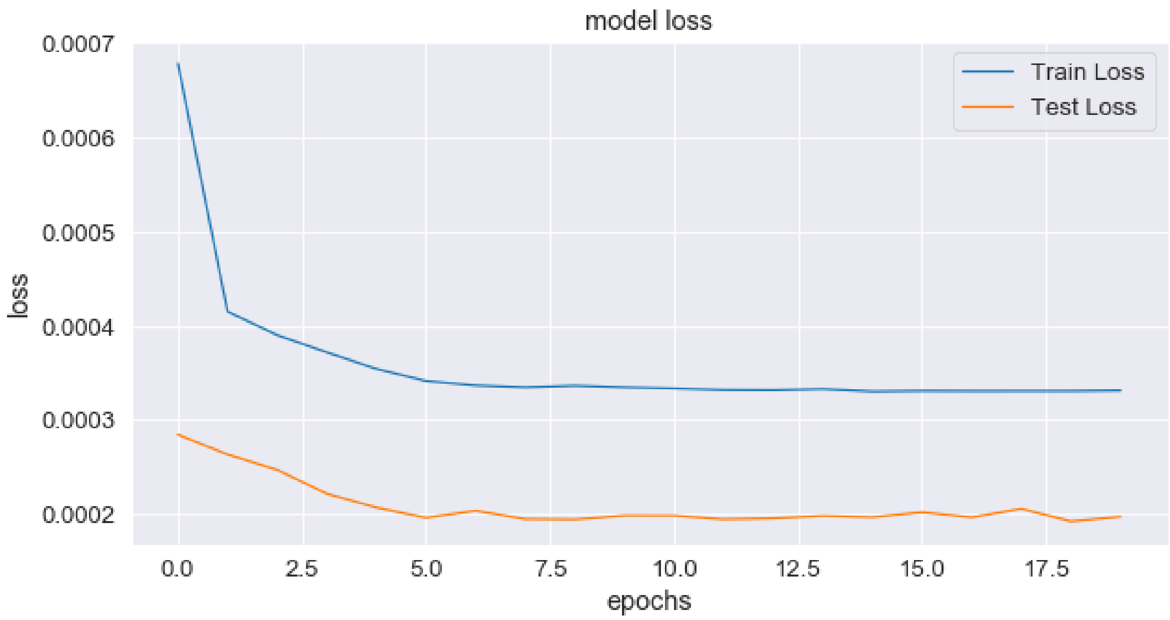
```
print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0], train_predict
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0], test_predict[:,
```

```
Train Mean Absolute Error: 0.9682103889849581
Train Root Mean Squared Error: 2.960199868748997
Test Mean Absolute Error: 0.7070643690272215
Test Root Mean Squared Error: 2.32091844195473
```

Model losses when training the model and testing

In [14]:

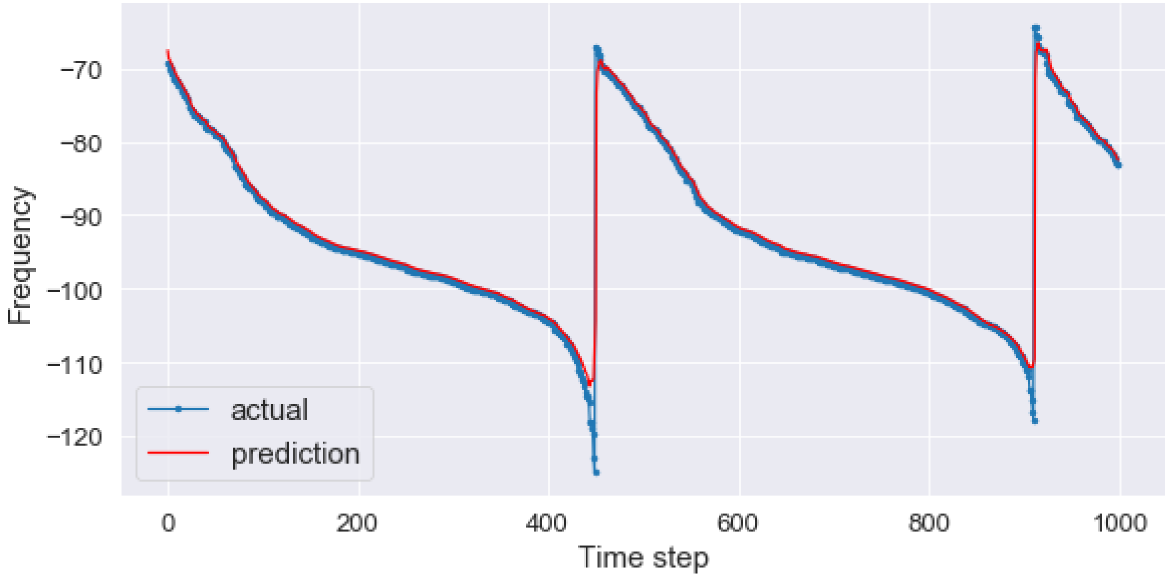
```
plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show()
```



Actual values vs Predicted values

In [15]:

```
aa = [x for x in range(1000)]
plt.figure(figsize=(8,4))
plt.plot(aa, Y_train[0][:1000], marker='.', label="actual")
plt.plot(aa, train_predict[:,0][:1000], 'r', label="prediction")
# plt.tick_params(left=False, labelleft=True) #remove ticks
plt.tight_layout()
sns.despine(top=True)
plt.subplots_adjust(left=0.07)
plt.ylabel('Frequency', size=15)
plt.xlabel('Time step', size=15)
plt.legend(fontsize=15)
plt.show()
```



Saving predicted data

In [16]:

```
sed = pd.DataFrame(test_predict)
sed.to_csv('imputedLSTMpredict.csv', index=False)
```

Saving the model

Saving the model

In [17]:

```
filename = 'imputed_model_LSTM.sav'
pickle.dump(model, open(filename, 'wb'))
```

In [18]:

```
model_json = model.to_json()
with open("imputedLSTM_model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("imputedLSTM_model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

Saving the normalisation model

the LSTM model was built on normalised data for better result. When attempting to predict new values, the data must be normalised before being sent to the model

In [19]:

```
filename = 'imputed_LSTMscaler.sav'
pickle.dump(scaler, open(filename, 'wb'))
```

Loading the saved model

In [20]:

```
json_file         = open('imputedLSTM_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model      = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("imputedLSTM_model.h5")
print("Loaded model from disk")
```

```
Loaded model from disk
```

In [21]:

```
look_back          = 12
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test   = create_dataset(test, look_back)
X_train          = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test           = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
train_predict    = loaded_model.predict(X_train)
test_predict     = loaded_model.predict(X_test)
# invert predictions
train_predict    = scaler.inverse_transform(train_predict)
Y_train          = scaler.inverse_transform([Y_train])
test_predict     = scaler.inverse_transform(test_predict)
Y_test           = scaler.inverse_transform([Y_test])
print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0], train_predict
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0], test_predict[:,
```

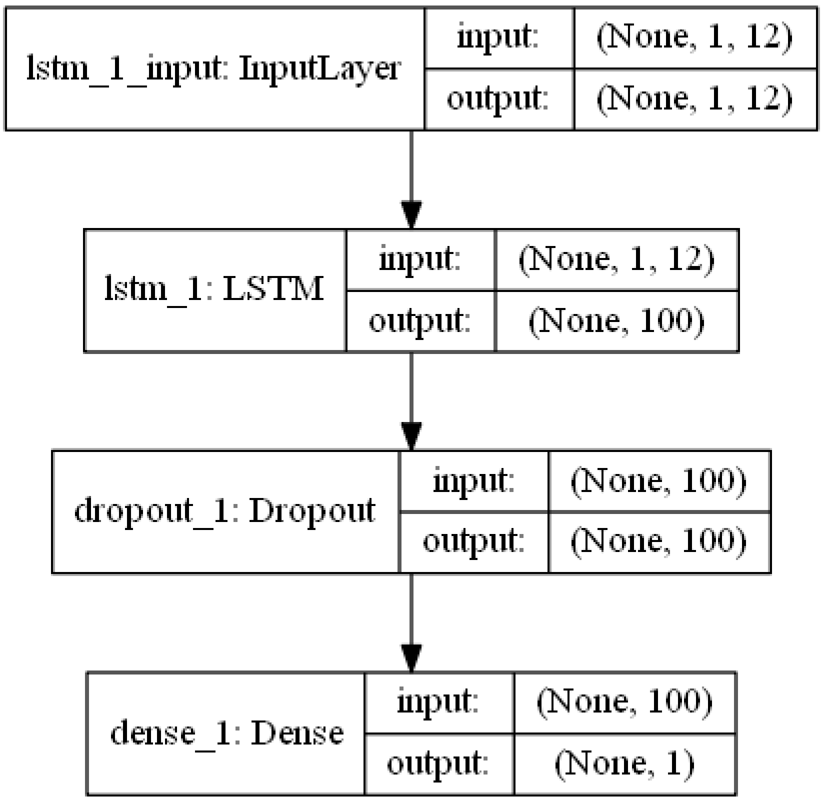
```
Train Mean Absolute Error: 0.9682103889849581
Train Root Mean Squared Error: 2.960199868748997
Test Mean Absolute Error: 0.7070643690272215
Test Root Mean Squared Error: 2.32091844195473
```

Visualisation of the Neural Network

In [22]:

```
plot_model(model, to_file='imputedLSTM.png', show_shapes=True)
```

Out[22]:



Model Parameters

- Epoch = 20
- Batch size = 100
- Hidden layers = 1
- Dropout layers = 1
- Dropout = 0.2
- Neurons = 100 in hidden layer, 1 in output layer

Observation	values
Train Mean Absolute Error	0.9682
Train Root Mean Squared Error	2.9601
Test Mean Absolute Error	0.7071
Test Root Mean Squared Error	2.3209
Training time	1445.3598
Prediction time	40.5545

Observation	LSTM	Improved LSTM
Train Mean Absolute Error	0.9682	0.6871
Train Root Mean Squared Error	2.9601	2.8156
Test Mean Absolute Error	0.7071	0.4220
Test Root Mean Squared Error	2.3209	2.1649
Training time	1445.3598	2062.3754
Prediction time	40.5545	56.5365

Models	Train Mean Absolute Error	Train Root Mean Squared Error	Test Mean Absolute Error	Test Root Mean Squared Error	Training time	Prediction time
DNN	1.0161	2.8540	0.7674	2.2315	385.8252	16.4987
Improved DNN	0.9607	2.8690	0.7121	2.2580	823.0135	27.5391
LSTM	0.9682	2.9601	0.7071	2.3209	1445.3598	40.5545
Improved LSTM	0.6871	2.8156	0.4220	2.1649	2062.3754	56.5365