Neural Network based Passengers Prediction of Airlines

RECURRENT NEURAL NETWORKS (RNN)

LONG SHORT-TERM MEMORY (LSTM)

TIME SERIES

In [1]:

```
# LSTM for International Airline Passengers problem with Regression framing
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler
```

In [2]:

```
dataset = pd.read_csv('E:/RNN-LSTM/AirPassengers.csv')
dataset
```

Out[2]:

	Month	Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121
...	...	...
139	1960-08	606
140	1960-09	508
141	1960-10	461
142	1960-11	390
143	1960-12	432

144 rows × 2 columns

In [3]:

```
dataset = dataset['Passengers']
dataset
```

Out[3]:

```
0       112
1       118
2       132
3       129
4       121
       ...
139     606
140     508
141     461
142     390
143     432
```

```
Name: Passengers, Length: 144, dtype: int64
```

In [4]:

```
#Converting Time Series data into Numpy Array
dataset = np.array(dataset).reshape(-1,1)
dataset
```

Out[4]:

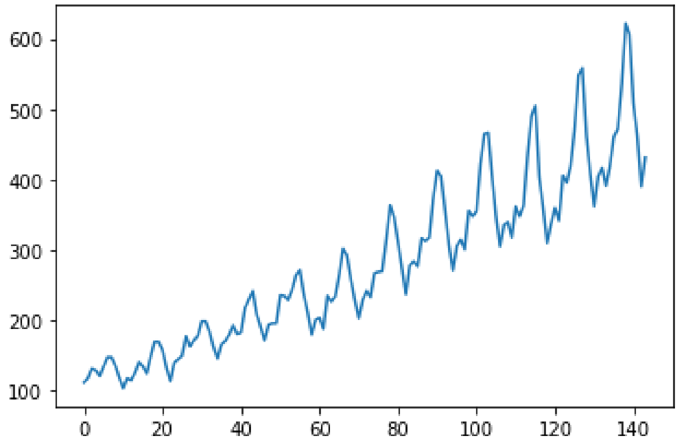
```
array([[112],
       [118],
       [132],
       [129],
       [121],
       [135],
       [148],
       [148],
       [136],
       [119],
       [104],
       [118],
       [115],
       [126],
       [141],
       [135],
       [125],
       [149],
       [170],
       [170],
       [158],
       [133],
       [114],
       [140],
       [145],
       [150],
       [178],
       [163],
       [172],
       [178],
       [199],
       [199],
       [184],
       [162],
       [146],
       [166],
       [171],
       [180],
       [193],
       [181],
       [183],
       [218],
       [230],
       [242],
       [209],
       [191],
       [172],
       [194],
       [196],
       [196],
       [236],
       [235],
       [229],
       [243],
       [264],
       [272],
       [237],
       [211],
       [180],
       [201],
       [204],
       [188],
        [235],
        [227],
        [234],
        [264],
        [302],
        [293],
        [259],
        [229],
        [203],
        [229],
        [242],
        [233],
        [267],
        [269],
        [270],
        [315],
        [364],
        [347],
        [312],
        [274],
        [237],
        [278],
        [284],
        [277],
        [317],
        [313],
        [318],
        [374],
        [413],
        [405],
        [355],
        [306],
        [271],
        [306],
        [315],
        [301],
        [356],
        [348],
        [355],
        [422],
        [465],
        [467],
        [404],
        [347],
        [305],
        [336],
        [340],
        [318],
        [362],
        [348],
        [363],
        [435],
        [491],
        [505],
        [404],
        [359],
        [310],
        [337],
        [360],
        [342],
        [406],
        [396],
        [420],
        [472],
        [548],
        [559],
        [463],
        [407],
        [362],
        [405],
        [417],
        [391],
        [419],
        [461],
        [472],
        [535],
        [622],
        [606],
        [508],
        [461],
        [390],
        [432]], dtype=int64)
```

In [5]:

```
plt.plot(dataset)
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x25208bfe520>]
```



Total Data is of 144 Months

Training Data = 100 Months (0 to 100)

Testing Data = 44 Months (101 to 144)

In [6]:

```
#Neural Networks works better if Inputs are in between '0' to '1'
#MinMaxScaler to get values from '0' to '1'
scaler = MinMaxScaler()
dataset = scaler.fit_transform(dataset)
dataset
```

Out[6]:

```
array([[0.01544402],
       [0.02702703],
       [0.05405405],
       [0.04826255],
       [0.03281853],
       [0.05984556],
       [0.08494208],
       [0.08494208],
       [0.06177606],
       [0.02895753],
       [0.        ],
       [0.02702703],
       [0.02123552],
       [0.04247104],
       [0.07142857],
       [0.05984556],
       [0.04054054],
       [0.08687259],
       [0.12741313],
       [0.12741313],
       [0.1042471 ],
       [0.05598456],
       [0.05598456],
       [0.01930502],
       [0.06949807],
       [0.07915058],
       [0.08880309],
       [0.14285714],
       [0.11389961],
       [0.13127413],
       [0.14285714],
       [0.18339768],
       [0.18339768],
       [0.15444015],
       [0.11196911],
       [0.08108108],
       [0.11969112],
       [0.12934363],
       [0.14671815],
       [0.17181467],
       [0.14864865],
       [0.15250965],
       [0.22007722],
       [0.24324324],
       [0.26640927],
       [0.2027027 ],
       [0.16795367],
       [0.13127413],
       [0.17374517],
       [0.17760618],
       [0.17760618],
       [0.25482625],
       [0.25289575],
       [0.24131274],
       [0.26833977],
       [0.30888031],
       [0.32432432],
       [0.25675676],
       [0.20656371],
       [0.14671815],
       [0.18725869],
       [0.19305019],
       [0.16216216],
       [0.25289575],
       [0.23745174],
       [0.25096525],
       [0.30888031],
       [0.38223938],
       [0.36486486],
       [0.2992278 ],
       [0.24131274],
       [0.19111969],
       [0.24131274],
       [0.26640927],
       [0.24903475],
       [0.31467181],
       [0.31853282],
       [0.32046332],
       [0.40733591],
       [0.5019305 ],
       [0.46911197],
       [0.4015444 ],
       [0.32818533],
       [0.25675676],
       [0.33590734],
       [0.34749035],
       [0.33397683],
       [0.41119691],
       [0.4034749 ],
       [0.41312741],
       [0.52123552],
       [0.5965251 ],
       [0.58108108],
       [0.48455598],
       [0.38996139],
       [0.32239382],
       [0.38996139],
       [0.40733591],
       [0.38030888],
       [0.48648649],
       [0.47104247],
       [0.48455598],
       [0.61389961],
       [0.6969112 ],
       [0.7007722 ],
       [0.57915058],
       [0.46911197],
       [0.38803089],
       [0.44787645],
       [0.45559846],
       [0.41312741],
       [0.4980695 ],
       [0.47104247],
       [0.5       ],
       [0.63899614],
       [0.74710425],
       [0.77413127],
       [0.57915058],
       [0.49227799],
       [0.3976834 ],
       [0.44980695],
       [0.49420849],
       [0.45945946],
       [0.58301158],
       [0.56370656],
       [0.61003861],
       [0.71042471],
       [0.85714286],
       [0.87837838],
       [0.69305019],
       [0.58494208],
       [0.4980695 ],
       [0.58108108],
       [0.6042471 ],
       [0.55405405],
       [0.60810811],
       [0.68918919],
       [0.71042471],
       [0.83204633],
       [1.        ],
       [0.96911197],
       [0.77992278],
       [0.68918919],
       [0.55212355],
       [0.63320463]])
```

In [7]:

```
dataset.min()  #Minimum Value
```

Out[7]:

```
0.0
```

In [8]:

```
dataset.max()  #Maximum Value
```

Out[8]:

```
1.0
```

In [9]:

```
train_size = 100
test_size = 44
```

In [10]:

```
train = dataset[0:train_size, :] #from '0' to 'train_size = 100' , all the Columns
train.shape
```

Out[10]:

```
(100, 1)
```

In [11]:

```
test = dataset[train_size:144, :] #from 'train_size = 144' , all the Columns
test.shape
```

Out[11]:

```
(44, 1)
```

Build TRAINING & TESTING Dataset

In [12]:

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

In [13]:

```
look_back = 1
X_train, y_train = get_data(train, look_back)
```

In [14]:

```
X_train.shape
```

Out[14]:

```
(98, 1)
```

In [15]:

```
y_train
```

Out[15]:

```
array([0.02702703, 0.05405405, 0.04826255, 0.03281853, 0.05984556,
       0.08494208, 0.08494208, 0.06177606, 0.02895753, 0.        ,
       0.02702703, 0.02123552, 0.04247104, 0.07142857, 0.05984556,
       0.04054054, 0.08687259, 0.12741313, 0.12741313, 0.1042471 ,
       0.05598456, 0.01930502, 0.06949807, 0.07915058, 0.08880309,
       0.14285714, 0.11389961, 0.13127413, 0.14285714, 0.18339768,
       0.18339768, 0.15444015, 0.11196911, 0.08108108, 0.11969112,
       0.12934363, 0.14671815, 0.17181467, 0.14864865, 0.15250965,
       0.22007722, 0.24324324, 0.26640927, 0.2027027 , 0.16795367,
       0.13127413, 0.17374517, 0.17760618, 0.17760618, 0.25482625,
       0.25289575, 0.24131274, 0.26833977, 0.30888031, 0.32432432,
       0.25675676, 0.20656371, 0.14671815, 0.18725869, 0.19305019,
       0.16216216, 0.25289575, 0.23745174, 0.25096525, 0.30888031,
       0.38223938, 0.36486486, 0.2992278 , 0.24131274, 0.19111969,
       0.24131274, 0.26640927, 0.24903475, 0.31467181, 0.31853282,
       0.32046332, 0.40733591, 0.5019305 , 0.46911197, 0.4015444 ,
       0.32818533, 0.25675676, 0.33590734, 0.34749035, 0.33397683,
       0.41119691, 0.4034749 , 0.41312741, 0.52123552, 0.5965251 ,
       0.58108108, 0.48455598, 0.38996139, 0.32239382, 0.38996139,
       0.40733591, 0.38030888, 0.48648649])
```

In [16]:

```
X_test, y_test = get_data(test, look_back)
```

In [17]:

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

Build the Model

In [18]:

```
model = Sequential()
model.add(LSTM(5, input_shape = (1, look_back)))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
```

In [19]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm (LSTM)                  (None, 5)                 140       
_________________________________________________________________
dense (Dense)                (None, 1)                 6         
=================================================================
Total params: 146
Trainable params: 146
Non-trainable params: 0
_________________________________________________________________
```

In [20]:

```
model.fit(X_train, y_train, epochs = 50, batch_size = 1)
```

```
Epoch 1/50
98/98 [==============================] - 0s 1ms/step - loss: 0.0402
Epoch 2/50
98/98 [==============================] - 0s 712us/step - loss: 0.0187
Epoch 3/50
98/98 [==============================] - 0s 762us/step - loss: 0.0144
Epoch 4/50
98/98 [==============================] - 0s 889us/step - loss: 0.0131
Epoch 5/50
98/98 [==============================] - 0s 854us/step - loss: 0.0120
Epoch 6/50
98/98 [==============================] - 0s 1ms/step - loss: 0.0109
Epoch 7/50
98/98 [==============================] - 0s 1ms/step - loss: 0.0098
Epoch 8/50
98/98 [==============================] - 0s 1ms/step - loss: 0.0086
Epoch 9/50
98/98 [==============================] - 0s 1ms/step - loss: 0.0075
Epoch 10/50
98/98 [==============================] - 0s 818us/step - loss: 0.0064
Epoch 11/50
98/98 [==============================] - 0s 770us/step - loss: 0.0055
Epoch 12/50
98/98 [==============================] - 0s 820us/step - loss: 0.0046
Epoch 13/50
98/98 [==============================] - 0s 875us/step - loss: 0.0039
Epoch 14/50
98/98 [==============================] - 0s 996us/step - loss: 0.0034
Epoch 15/50
98/98 [==============================] - 0s 719us/step - loss: 0.0030
Epoch 16/50
98/98 [==============================] - 0s 738us/step - loss: 0.0026
Epoch 17/50
```

```
98/98 [==============================] - 0s 723us/step - loss: 0.0025
Epoch 18/50
98/98 [==============================] - 0s 719us/step - loss: 0.0023
Epoch 19/50
98/98 [==============================] - 0s 910us/step - loss: 0.0022
Epoch 20/50
98/98 [==============================] - 0s 885us/step - loss: 0.0021
Epoch 21/50
98/98 [==============================] - 0s 826us/step - loss: 0.0023
Epoch 22/50
98/98 [==============================] - 0s 747us/step - loss: 0.0021
Epoch 23/50
98/98 [==============================] - 0s 742us/step - loss: 0.0022
Epoch 24/50
98/98 [==============================] - 0s 754us/step - loss: 0.0022
Epoch 25/50
98/98 [==============================] - 0s 905us/step - loss: 0.0021
Epoch 26/50
98/98 [==============================] - 0s 860us/step - loss: 0.0021
Epoch 27/50
98/98 [==============================] - 0s 696us/step - loss: 0.0022
Epoch 28/50
98/98 [==============================] - 0s 712us/step - loss: 0.0021
Epoch 29/50
98/98 [==============================] - 0s 711us/step - loss: 0.0021
Epoch 30/50
98/98 [==============================] - 0s 734us/step - loss: 0.0021
Epoch 31/50
98/98 [==============================] - 0s 714us/step - loss: 0.0021
Epoch 32/50
98/98 [==============================] - 0s 721us/step - loss: 0.0022
Epoch 33/50
98/98 [==============================] - 0s 733us/step - loss: 0.0021
Epoch 34/50
98/98 [==============================] - 0s 714us/step - loss: 0.0022
Epoch 35/50
98/98 [==============================] - 0s 737us/step - loss: 0.0021
Epoch 36/50
98/98 [==============================] - 0s 851us/step - loss: 0.0021
Epoch 37/50
98/98 [==============================] - 0s 867us/step - loss: 0.0022
Epoch 38/50
98/98 [==============================] - 0s 856us/step - loss: 0.0021
Epoch 39/50
98/98 [==============================] - 0s 878us/step - loss: 0.0021
Epoch 40/50
98/98 [==============================] - 0s 895us/step - loss: 0.0021
Epoch 41/50
98/98 [==============================] - 0s 773us/step - loss: 0.0022
Epoch 42/50
98/98 [==============================] - 0s 715us/step - loss: 0.0021
Epoch 43/50
98/98 [==============================] - 0s 747us/step - loss: 0.0021
Epoch 44/50
98/98 [==============================] - 0s 889us/step - loss: 0.0021
Epoch 45/50
98/98 [==============================] - 0s 847us/step - loss: 0.0021
Epoch 46/50
98/98 [==============================] - 0s 773us/step - loss: 0.0022
Epoch 47/50
98/98 [==============================] - 0s 717us/step - loss: 0.0022
Epoch 48/50
98/98 [==============================] - 0s 739us/step - loss: 0.0021
Epoch 49/50
98/98 [==============================] - 0s 731us/step - loss: 0.0021
Epoch 50/50
98/98 [==============================] - 0s 854us/step - loss: 0.0022
```

Out[20]:

```
<tensorflow.python.keras.callbacks.History at 0x25209392b80>
```

In [21]:

```
y_pred = model.predict(X_test)
```

In [22]:

```
scaler.scale_
```

Out[22]:

```
array([0.0019305])
```

In [23]:

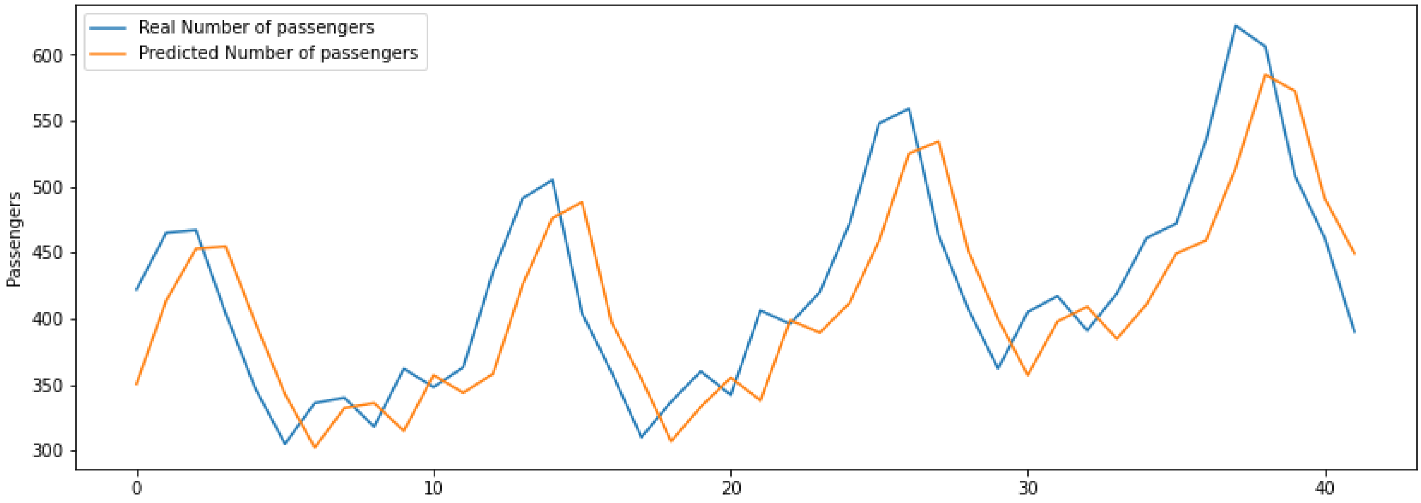
```
y_pred = scaler.inverse_transform(y_pred)
```

In [24]:

```
y_test = np.array(y_test)
y_test = y_test.reshape(-1, 1)
y_test = scaler.inverse_transform(y_test)
```

In [25]:

```
#Plot Baseline & Predictions
plt.figure(figsize = (14,5))
plt.plot(y_test, label = 'Real Number of passengers')
plt.plot(y_pred, label = 'Predicted Number of passengers')
plt.ylabel('Passengers')
plt.legend()
plt.show()
```



In [26]:

```
scaler.scale_
```

Out[26]:

```
array([0.0019305])
```