Discription

- The dataset provides monthly total number of a US airline passengers from 1949 to 1960.
- It contains 2 columns:

```
1. Month
2. Passengers: number of passengers in this month
```

- We Will use this data as ***Time-Series*** and use previous months data to predict the number of passenger of the next one.

- Applying LSTM model to use 2 months data to predict the next one.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense,LSTM
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import MinMaxScaler
```

Impoting Data

In [2]:

```
#"C:\Users\Afnan\Downloads\AirPassengers.csv"
data = pd.read_csv("C:/Users/Afnan/Downloads/AirPassengers.csv")
data.head()
```

Out[2]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

Prepocessing

> fix the column name to be easier for use

In [3]:

```
data.rename(columns={'#Passengers':'passengers'},inplace=True)
```

> **Month** column can be dropped as is a ***Time-Series*** Model

In [4]:

```
data = data['passengers']
```

In [5]:

```
type(data)
```

Out[5]:

```
pandas.core.series.Series
```

> Convet data type into ***2D array*** to be able to apply methods

In [6]:

```
data = np.array(data).reshape(-1,1)
type(data)
```

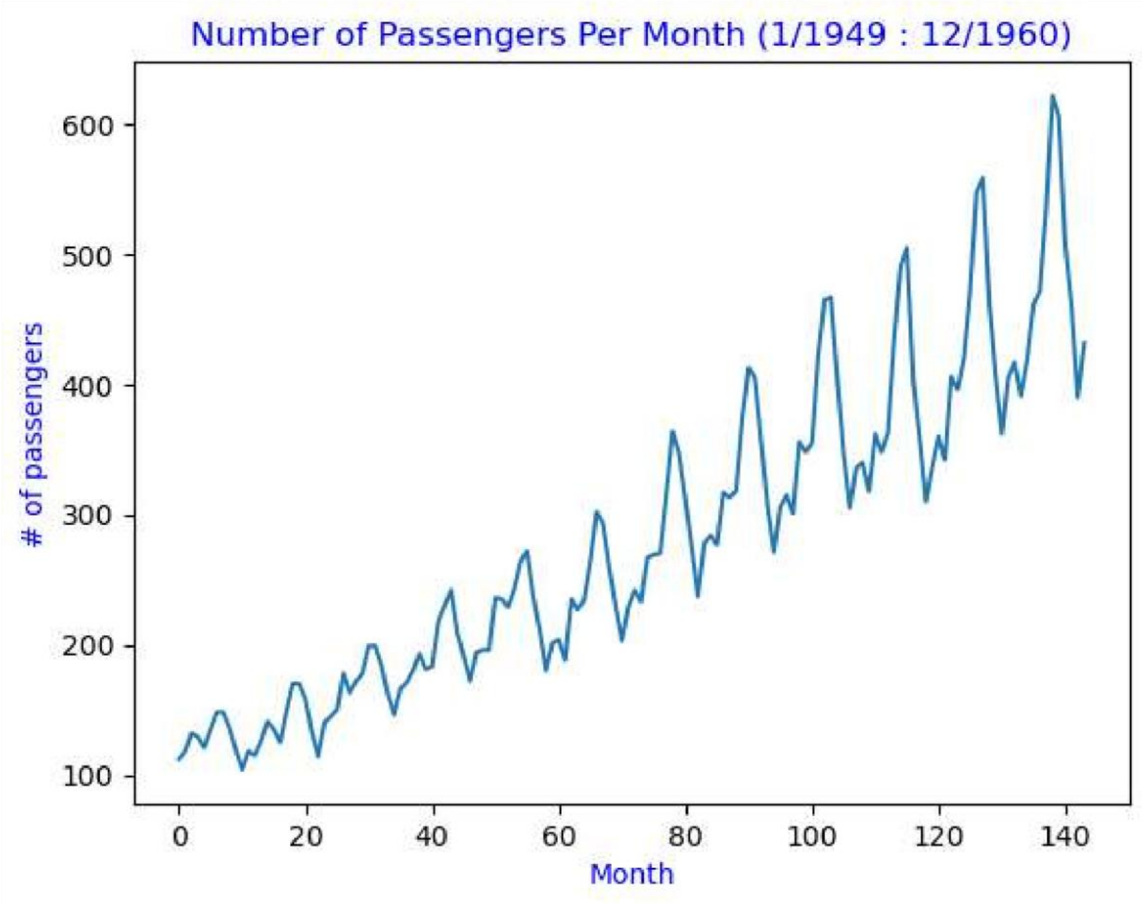
Out[6]:

```
numpy.ndarray
```

data is converted and reshaped as it had 1 feature

In [7]:

```
plt.plot(data)
plt.title('Number of Passengers Per Month (1/1949 : 12/1960)' , color='blue')
plt.xlabel('Month', color='blue')
plt.ylabel('# of passengers', color='blue')
plt.show()
```



We can see The ***Patterns*** and ***Seasons effects***

Scalling

> As LSTM is senstive to the scale of the data
> we will normalize (rescale to the range 0:1) it Using MinMax

In [8]:

```
scaler = MinMaxScaler()
data = scaler.fit_transform(data)
```

In [9]:

```
len(data)
```

Out[9]:

```
144
```

> Let Split into ***70%*** about **100 training and 44 testing**

In [10]:

```
train = data[0:100,:]
test = data[100:,:]
```

Set the Model Input

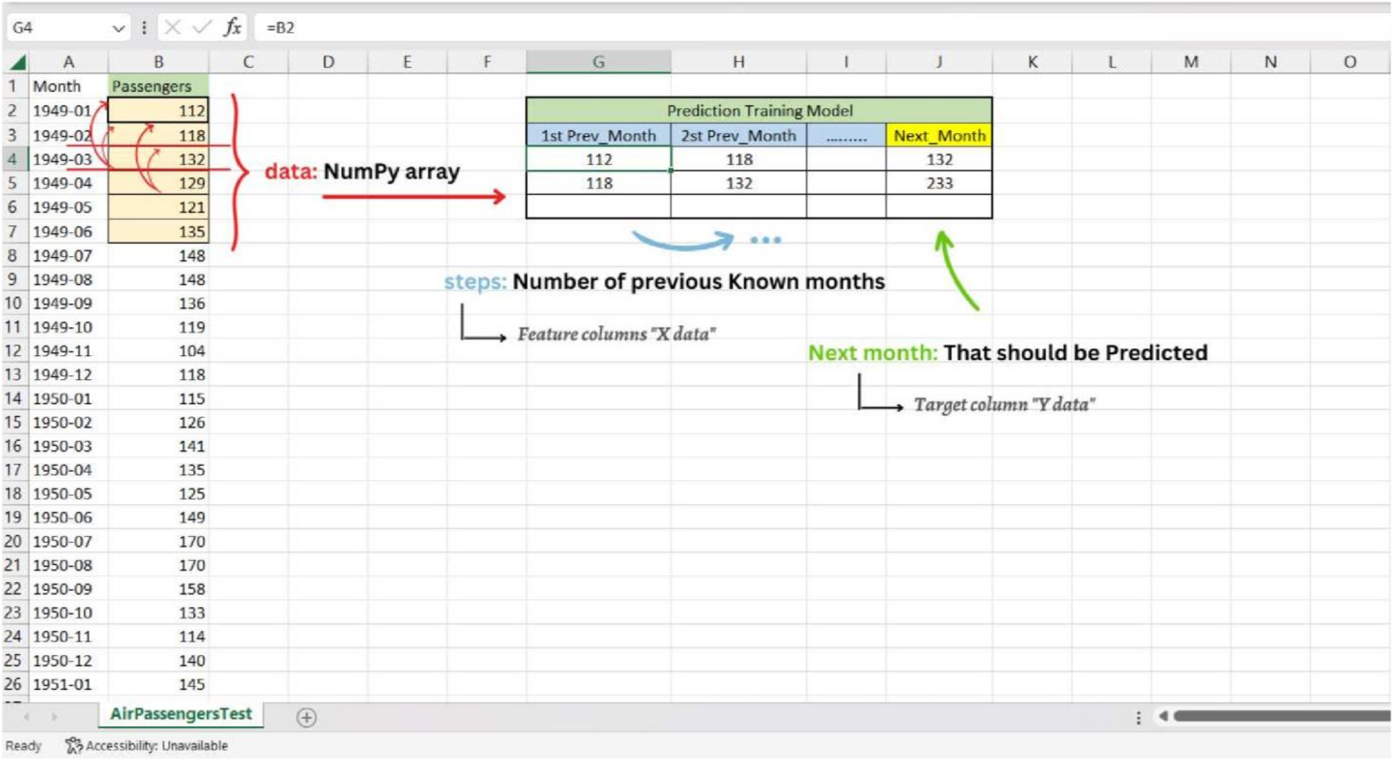
- For LSTM we need sequential input for training and testing

> Defining a function to prepare the data with 2 arguments:

1. **data:** NumPy array to be converted into the dataset.
2. **steps:** integer to represent the the previous time "months" steps.

In [11]:

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



In [12]:

```
steps = 2
```

In [13]:

```
X_train, y_train = get_data(train, steps)
X_test, y_test = get_data(test, steps)
```

reshape the data to **3D format** as required for LSTM model

In [14]:

```
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]))
```

Building the LSTM Model

- Sequential model with 2 hidden layers
  - The first has **120** memory blocks
  - The second has **65**
- With The default activation function of LSTM: **sigmoid**
- loss function: **mean square error**
- Using **adam** as the optimizer algorithm with a low memory requirment

In [15]:

```
model = Sequential()
model.add(LSTM(120, input_shape=(1, steps)))
model.add(Dense(65))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
```

In [16]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 120)               59040     

 dense (Dense)               (None, 65)                7865      

 dense_1 (Dense)             (None, 1)                 66        

=================================================================
Total params: 66,971
Trainable params: 66,971
Non-trainable params: 0
_________________________________________________________________
```

Training the Model

In [17]:

```
model.fit(X_train, y_train, epochs=25, batch_size=1)
```

```
Epoch 1/25
97/97 [==============================] - 4s 3ms/step - loss: 0.0092
Epoch 2/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0034
Epoch 3/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0036
Epoch 4/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0028
Epoch 5/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0032
Epoch 6/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0032
Epoch 7/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0031
Epoch 8/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 9/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0034
Epoch 10/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0027
Epoch 11/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 12/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 13/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0027
Epoch 14/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 15/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0027
Epoch 16/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0029
Epoch 17/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0027
Epoch 18/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 19/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0027
Epoch 20/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 21/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 22/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 23/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 24/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0028
Epoch 25/25
97/97 [==============================] - 0s 3ms/step - loss: 0.0025
```

Out[17]:

```
<keras.callbacks.History at 0x1a40cf52460>
```

Testing The model

In [18]:

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

```
2/2 [==============================] - 1s 6ms/step
```

> Rescale the prediction's result as the model return scaled ones

In [19]:

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

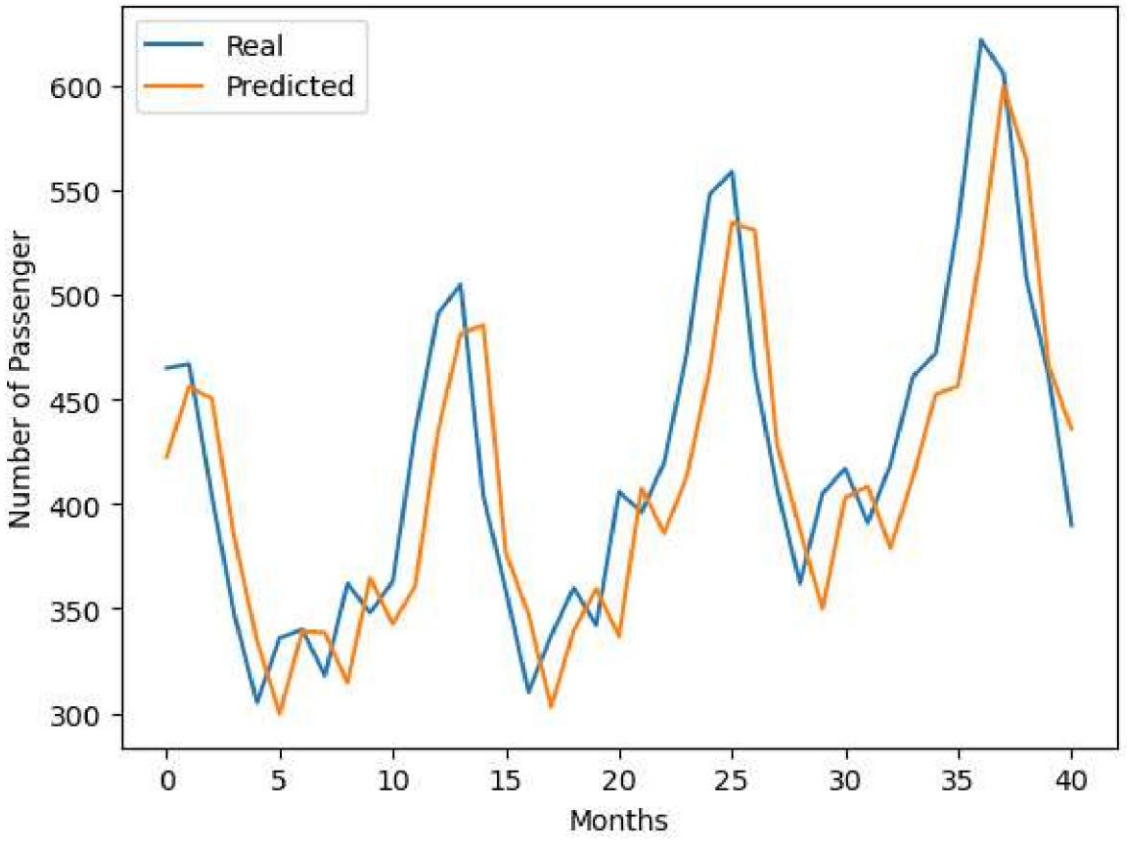
In [19]:

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

Plot Prediction Results

In [20]:

```
plt.plot(y_test, label = 'Real')
plt.plot(y_pred, label = 'Predicted')
plt.xlabel('Months')
plt.ylabel('Number of Passenger')
plt.legend()
plt.show()
```



As shown in the plot the model could ***catch the pattern*** of the Time-series and It was close to exact values at some points.