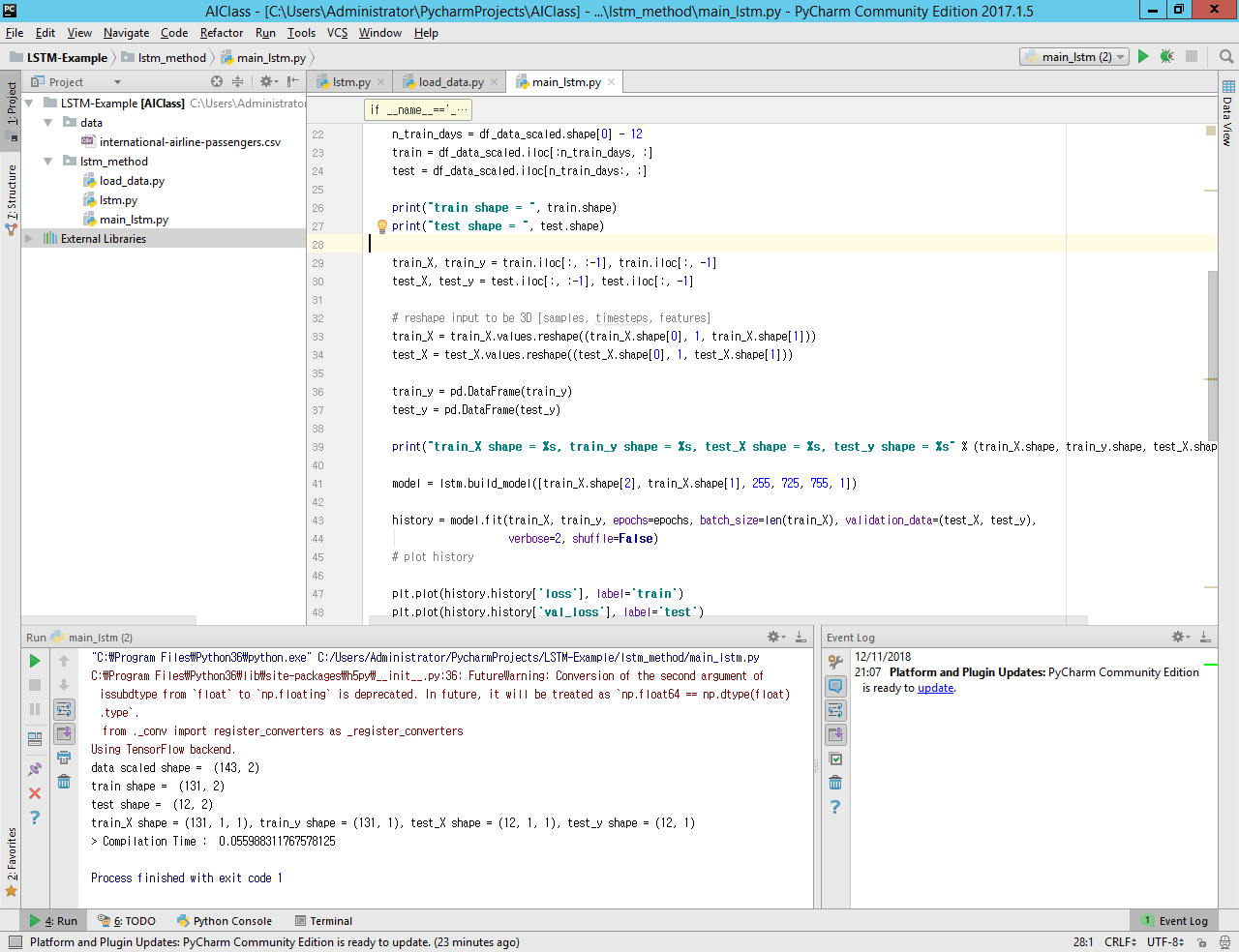
**Predicting Airline Passenger Using LSTM**

Create the directory hierarchy same as follow:



1. **Save the following code to load\_data.py**
2. Importing some packages.

Pandas as data manipulator, MinMaxScaler for normalizing the data to particular range value, and Numpy for metric operation.

|  |
| --- |
| **import** pandas **as** pd **from** sklearn.preprocessing **import** MinMaxScaler **from** pandas **import** concat **import** numpy **as** np |

1. Loading data from csv file.

The csv data consists of two columns, namely Month and Num of Passenger. set the column Month as date index which corresponding the time series data

|  |
| --- |
| **def** load\_data(file):  df = pd.read\_csv(file,  index\_col=[**"Month"**],  usecols=[**"Month"**,**"Num of Passenger"**])   df.index = pd.to\_datetime(df.index)  **return** df |

1. Converting series to supervised

Converting data to X and Y format, where the input is one-dimensional data it will create the Y data from X data. Such as follow:

|  |
| --- |
| X Y  112 118  118 132  132 129  129 121  121 135 |

*n\_in* corresponds to time steps before, and *n\_out* corresponds to number of time steps after.

|  |
| --- |
| **def** series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=**True**):  n\_vars = 1 **if** type(data) **is** list **else** data.shape[1]  df = pd.DataFrame(data)  cols, names = list(), list()  *# input sequence (t-n, ... t-1)* **for** i **in** range(n\_in, 0, -1):  cols.append(df.shift(i))  names += [(**'var%d(t-%d)'** % (j + 1, i)) **for** j **in** range(n\_vars)]  *# forecast sequence (t, t+1, ... t+n)* **for** i **in** range(0, n\_out):  cols.append(df.shift(-i))  **if** i == 0:  names += [(**'var%d(t)'** % (j + 1)) **for** j **in** range(n\_vars)]  **else**:  names += [(**'var%d(t+%d)'** % (j + 1, i)) **for** j **in** range(n\_vars)]  *# put it all together* agg = concat(cols, axis=1)  agg.columns = names  *# drop rows with NaN values* **if** dropnan:  agg.dropna(inplace=**True**)  **return** agg |

1. Normalizing the data

The number of passenger value is so high and volatile, therefore to make it more tractable to predict we convert it into 0 to 1 range.

|  |
| --- |
| **def** normalize\_data(data):  scaler = MinMaxScaler(feature\_range=(0, 1))  data\_scaled = scaler.fit\_transform(data)  **return** scaler, data\_scaled |

1. Inverting normalized data

This function for inverting normalize data; e.g. range 0 to 1, to the real value.

|  |
| --- |
| **def** inverse\_normdata(scaler, dataX, yhat):  dataX = dataX.reshape((dataX.shape[0], dataX.shape[2]))  inv\_y = np.concatenate((dataX, yhat), axis=1)  inv\_y = scaler.inverse\_transform(inv\_y)  inv\_y = inv\_y[:, -1]  **return** inv\_y |

1. **Save the following code as lstm.py**
2. Importing some packages.

The important package is Keras for creating LSTM model

|  |
| --- |
| **import** os **import** time **import** warnings **from** keras.layers.core **import** Dense, Activation, Dropout **from** keras.layers.recurrent **import** LSTM  **from** keras.models **import** Sequential  os.environ[**'TF\_CPP\_MIN\_LOG\_LEVEL'**] = **'3'** *# Hide messy TensorFlow warnings* warnings.filterwarnings(**"ignore"**) *# Hide messy Numpy warnings* |

1. Create LSTM model

Function for build LSTM model, where the input is array of layer which containing five index. layers [0,1,2,3,4,5] layer 0 correspond to dimension of data value, layer 1 correspond to the time-step, layer 2,3, and 4 corresponds to number of neurons in hidden layer and layer 5 correspond to the output.

|  |
| --- |
| *#layers [0,1,2,3,4,5]* **def** build\_model(layers):  model = Sequential()   model.add(LSTM(100,  input\_shape=(layers[1], layers[0]),  return\_sequences=**True**))  model.add(Dropout(0.5))   model.add(LSTM(  layers[2],  return\_sequences=**True**))  model.add(Dropout(0.2))   model.add(LSTM(  layers[3],  return\_sequences=**True**))  model.add(Dropout(0.1))   model.add(LSTM(  layers[4],  return\_sequences=**False**))  model.add(Dropout(0.5))   model.add(Dense(  output\_dim=layers[5]))  model.add(Activation(**"tanh"**))   start = time.time()  model.compile(loss=**"mse"**, optimizer=**"adam"**)  print(**"> Compilation Time : "**, time.time() - start)  model.reset\_states()  **return** model |

1. **Save the following code as main\_lstm.py**
2. Importing some packages

|  |
| --- |
| **import** lstm\_method.load\_data **as** data **import** lstm\_method.lstm **as** lstm **import** time **import** matplotlib.pyplot **as** plt **import** pandas **as** pd **from** sklearn.metrics **import** mean\_squared\_error **from** math **import** sqrt |

1. Main class

|  |
| --- |
| **if** \_\_name\_\_==**'\_\_main\_\_'**:  global\_start\_time = time.time()  epochs = 100  df = data.load\_data(**"../data/international-airline-passengers.csv"**)   data\_reframed = data.series\_to\_supervised(df, 1, 1)   df\_data\_reframed = pd.DataFrame(data\_reframed)   scaler, data\_scaled = data.normalize\_data(data\_reframed.values)  df\_data\_scaled = pd.DataFrame(data\_scaled, index=df\_data\_reframed.index, columns=df\_data\_reframed.columns)   print(**"data scaled shape = "**, df\_data\_scaled.shape)  n\_train\_days = df\_data\_scaled.shape[0] - 12  train = df\_data\_scaled.iloc[:n\_train\_days, :]  test = df\_data\_scaled.iloc[n\_train\_days:, :]   print(**"train shape = "**, train.shape)  print(**"test shape = "**, test.shape)   train\_X, train\_y = train.iloc[:, :-1], train.iloc[:, -1]  test\_X, test\_y = test.iloc[:, :-1], test.iloc[:, -1]   *# reshape input to be 3D [samples, timesteps, features]* train\_X = train\_X.values.reshape((train\_X.shape[0], 1, train\_X.shape[1]))  test\_X = test\_X.values.reshape((test\_X.shape[0], 1, test\_X.shape[1]))   train\_y = pd.DataFrame(train\_y)  test\_y = pd.DataFrame(test\_y)   print(**"train\_X shape = %s, train\_y shape = %s, test\_X shape = %s, test\_y shape = %s"** % (train\_X.shape, train\_y.shape, test\_X.shape, test\_y.shape))   model = lstm.build\_model([train\_X.shape[2], train\_X.shape[1], 255, 725, 755, 1])   history = model.fit(train\_X, train\_y, epochs=epochs, batch\_size=len(train\_X), validation\_data=(test\_X, test\_y),  verbose=2, shuffle=**False**)  *# plot history* plt.plot(history.history[**'loss'**], label=**'train'**)  plt.plot(history.history[**'val\_loss'**], label=**'test'**)  plt.legend()  plt.show()   *# plot model and train data* y\_train = model.predict(train\_X)  inv\_ymodel = data.inverse\_normdata(scaler, train\_X, y\_train)  df\_inv\_ymodel = pd.DataFrame(inv\_ymodel, index=train\_y.index)   *# plot test data* y\_test = model.predict(test\_X)  inv\_y\_test = data.inverse\_normdata(scaler, test\_X, y\_test)   df\_inv\_ytest = pd.DataFrame(inv\_y\_test, index=test\_y.index)  plt.plot(df, color=**'orange'**, label=**'Actual'**)  plt.plot(df\_inv\_ymodel, color=**'green'**, label=**'Trained'**)  plt.plot(df\_inv\_ytest, color=**'red'**, label=**'Predicted'**)  plt.legend()  plt.show()   inv\_test\_y = data.inverse\_normdata(scaler, test\_X, test\_y)  rmse\_normal = sqrt(mean\_squared\_error(inv\_test\_y, inv\_y\_test))  rmse\_scaled = sqrt(mean\_squared\_error(test\_y, y\_test))  print(**'Test RMSE normal : %.3f'** % rmse\_normal)  print(**'Test RMSE scaled : %.3f'** % rmse\_scaled) |

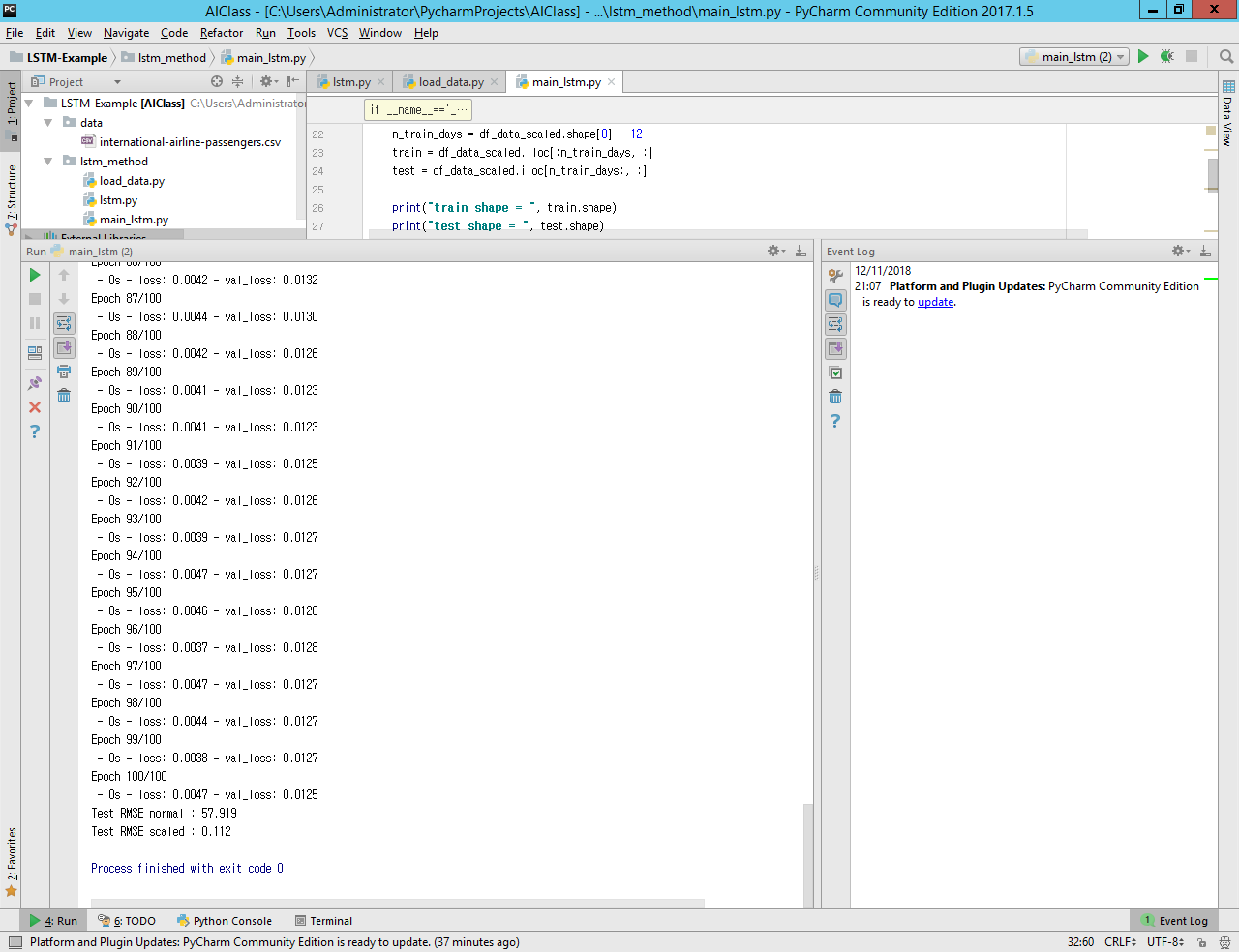


Figure . Epoch and RMSE

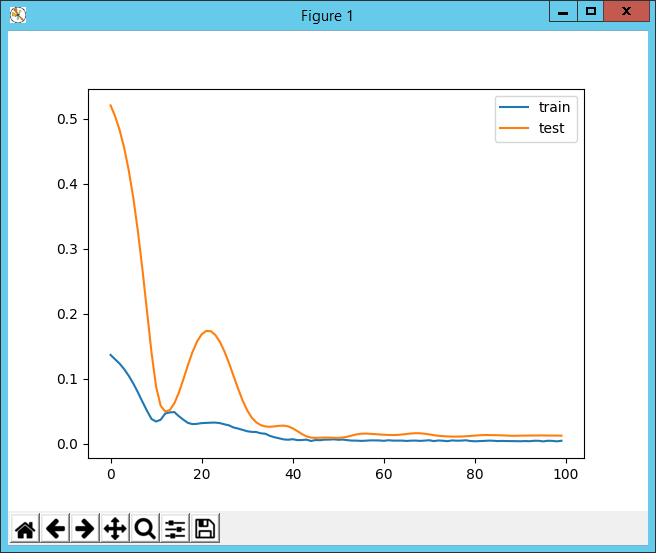


Figure 2. Loss function

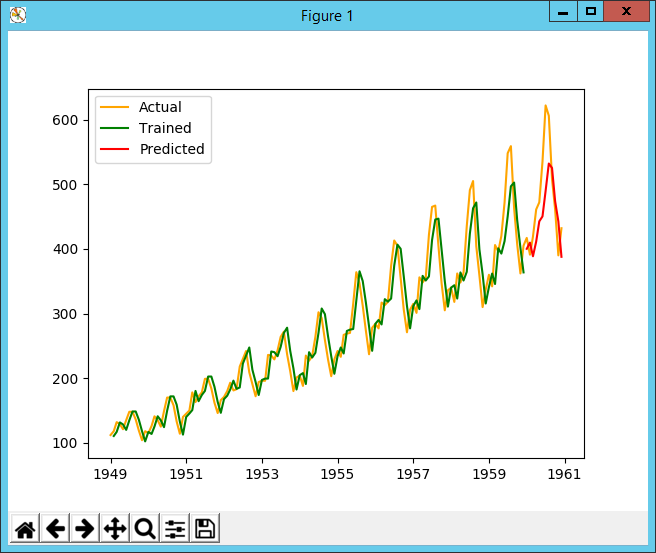


Figure 3. trained and predicted data