```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

Bounding box detection - Racoon data

Data files

- images_racoon.rar: contain images of racoons
- train_labels.cv: contains coordinates for bounding box for every image

Import the necessary libraries

```
# IMPORT LIBRARIES AND PACKAGES
import tensorflow as tf
import csv
import numpy as np
from PIL import Image
import pandas as pd
from keras import Model
from keras.applications.mobilenet import MobileNet, preprocess_input
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, Callback
from keras.layers import Conv2D, Reshape
from keras.utils import Sequence
from keras.backend import epsilon
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tens

```
Using TensorFlow backend.
```

Change directory

```
## NA
```

Load the training data from train.csv file

```
df = pd.read_csv('/content/drive/My Drive/Lab Internal-I  Residency-IX/train_labels.csv')
```

```
df.head(4)
```

```
         filename  width  height    class  xmin  ymin  xmax  ymax
   0  raccoon-17.jpg   259     194  raccoon    95    60   167   118
   1  raccoon-11.jpg   660     432  raccoon     3     1   461   431
   2  raccoon-63.jpg   600     400  raccoon    74   107   280   290
   3  raccoon-63.jpg   600     400  raccoon   227    93   403   298
```

Print the shape of the train dataset

```
df.shape
```

```
(173, 8)
```

Declare a variable IMAGE_SIZE = 128 as we will be using MobileNet which will be

```
# SETTINGS

ALPHA = 1.0 # Width hyper parameter for MobileNet (0.25, 0.5, 0.75, 1.0). Higher width means

IMAGE_SIZE = 128 # MobileNet takes images of size 128*128*3

EPOCHS = 10 # Number of epochs. I got decent performance with just 5.
BATCH_SIZE = 32 # Depends on your GPU or CPU RAM.

DATASET_FOLDER = "/content/drive/My Drive/Lab Internal-I  Residency-IX/"
TRAIN_CSV = DATASET_FOLDER+"train_labels.csv"
# VALIDATION_CSV = DATASET_FOLDER+"validation.csv"

images_zip_path = DATASET_FOLDER + "images_racoon.rar"

from zipfile import ZipFile

#with ZipFile(images_zip_path, 'r') as z:
#  z.extractall()
```

With the help of csv.reader write a for loop which can load the train.csv file and s x0,y0,x1,y1 in induvidual variables.

1. Create a list variable known as 'path' which has all the path for all the training images
2. Create an array 'coords' which has the resized coordinates of the bounding box for the training images

Note: All the training images should be downsampled to 128 * 128 as it is the input shape of MobileN
detection). Hence the corresponding coordinates of the bounding boxes should be changed to match

```
import csv
with open(TRAIN_CSV, 'r') as csvfile:
    paths = []
    coords = np.zeros((sum(1 for line in csvfile)-1, 4))
    reader = csv.reader(csvfile, delimiter=',')
    csvfile.seek(0)## Reading at zero line
    next(reader) ## TO SKIP THE HEADER
    print(coords.shape)
```

```
(173, 4)
```

```
df.head(4)
```

	filename	width	height	class	xmin	ymin	xmax	ymax
0	raccoon-17.jpg	259	194	raccoon	95	60	167	118
1	raccoon-11.jpg	660	432	raccoon	3	1	461	431
2	raccoon-63.jpg	600	400	raccoon	74	107	280	290
3	raccoon-63.jpg	600	400	raccoon	227	93	403	298

```
import csv
with open(TRAIN_CSV, 'r') as csvfile:
    paths = []
    coords = np.zeros((sum(1 for line in csvfile)-1, 4))
    reader = csv.reader(csvfile, delimiter=',')
    csvfile.seek(0)## Reading at zero line
    next(reader) ## TO SKIP THE HEADER

    for col, row in enumerate(reader):

     # print(row)
      #for i, r in enumerate(row[1:8]): # Parse row with seven entities
            #print([ line[0][0] for  line in r])
            #row[i+1] = (r)
            #print(int(r))
        path, image_width, image_height,_, x0, y0, x1, y1 = row
        path = '/content/drive/My Drive/Lab Internal-I  Residency-IX/images/'+str(path)

        #path = "./" + path.split('/')[-2] + "/" + path.split('/')[-1]

        coords[col, 0] = int(x0) * IMAGE_SIZE / int(image_width) # Normalize bounding box by
        coords[col, 1] = int(y0) * IMAGE_SIZE / int(image_height) # Normalize bounding box by
        coords[col, 2] = (int(x1) - int(x0))* IMAGE_SIZE / int(image_width) # Normalize bounc
        coords[col, 3] = (int(y1) - int(y0)) * IMAGE_SIZE / int(image_height)
        paths.append(path)
```

```
print(paths)
```

```
['/content/drive/My Drive/Lab Internal-I  Residency-IX/images/raccoon-17.jpg', '/content
```

Write a for loop which can load all the training images into a variable 'batch_imag 'paths' variable

Note: Convert the image to RGB scale as the MobileNet accepts 3 channels as inputs

```
batch_images = np.zeros((len(paths), IMAGE_SIZE, IMAGE_SIZE, 3), dtype=np.float32)
for i, f in enumerate(paths):
    img = Image.open(f) # Read image
    img = img.resize((IMAGE_SIZE, IMAGE_SIZE)) # Resize image
    img = img.convert('RGB')
    batch_images[i] = preprocess_input(np.array(img, dtype=np.float32))
```

Import MobileNet and load MobileNet into a variable named 'model' which takes Freeze all the layers. Add convolution and reshape layers at the end to ensure the

```
model = MobileNet(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), include_top=False, alpha=ALPHA) #
# Do not include classification (top) layer

# to freeze layers, except the new top layer, of course, which will be added below
for layer in model.layers:
    layer.trainable = False

# Add new top layer which is a conv layer of the same size as the previous layer so that only
x = model.layers[-1].output
x = Conv2D(4, kernel_size=4, name="coords")(x)
# In the line above kernel size should be 3 for img size 96, 4 for img size 128, 5 for img si
x = Reshape((4,))(x) # These are the 4 predicted coordinates of one BBox

model = Model(inputs=model.input, outputs=x)
```

```
model.summary()
```

Define a custom loss function IoU which calculates Intersection Over Union

```
gt = coords
def loss(gt,pred):
    intersections = 0
    unions = 0
    diff_width = np.minimum(gt[:,0] + gt[:,2], pred[:,0] + pred[:,2]) - np.maximum(gt[:,0], p
    diff_height = np.minimum(gt[:,1] + gt[:,3], pred[:,1] + pred[:,3]) - np.maximum(gt[:,1], 
    intersection = diff_width * diff_height

    # Compute union
    area_gt = gt[:,2] * gt[:,3]
    area_pred = pred[:,2] * pred[:,3]
    union = area_gt + area_pred - intersection

#     Compute intersection and union over multiple boxes
    for j, _ in enumerate(union):
        if union[j] > 0 and intersection[j] > 0 and union[j] >= intersection[j]:
            intersections += intersection[j]
            unions += union[j]

    # Compute IOU. Use epsilon to prevent division by zero
    iou = np.round(intersections / (unions + epsilon()), 4)
    iou = iou.astype(np.float32)
    return iou

def IoU(y_true, y_pred):
    iou = tf.py_func(loss, [y_true, y_pred], tf.float32)
    return iou
```

Write model.compile function & model.fit function with:

1. Optimizer = Adam, Loss = 'mse' and metrics = IoU
2. Epochs = 30, batch_size = 32, verbose = 1

```
gt.shape
```

```
(173, 4)
```

```
model.compile(optimizer='Adam', loss='mse', metrics=[IoU]) # Regression loss is MSE

#checkpoint = ModelCheckpoint("model-{val_iou:.2f}.h5", verbose=1, save_best_only=True,
#                              save_weights_only=True, mode="max", period=1) # Checkpoint bes
#stop = EarlyStopping(monitor="val_iou", patience=PATIENCE, mode="max") # Stop early, if the
#reduce_lr = ReduceLROnPlateau(monitor="val_iou", factor=0.2, patience=10, min_lr=1e-7, verbo
# Reduce learning rate if Validation IOU does not improve
```

```
model.fit(batch_images,gt,
            epochs=30,batch_size = 32,
            verbose=1)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_

Epoch 1/30
173/173 [==============================] - 8s 47ms/step - loss: 2805.7204 - IoU: 0.0890
Epoch 2/30
173/173 [==============================] - 5s 30ms/step - loss: 662.1935 - IoU: 0.4246
Epoch 3/30
173/173 [==============================] - 5s 29ms/step - loss: 662.7160 - IoU: 0.5335
Epoch 4/30
173/173 [==============================] - 5s 29ms/step - loss: 574.0978 - IoU: 0.5340
Epoch 5/30
173/173 [==============================] - 5s 30ms/step - loss: 345.5321 - IoU: 0.5827
Epoch 6/30
173/173 [==============================] - 5s 29ms/step - loss: 276.4565 - IoU: 0.5777
Epoch 7/30
173/173 [==============================] - 5s 29ms/step - loss: 236.0282 - IoU: 0.6175
Epoch 8/30
173/173 [==============================] - 5s 29ms/step - loss: 192.1124 - IoU: 0.6605
Epoch 9/30
173/173 [==============================] - 5s 29ms/step - loss: 168.6057 - IoU: 0.7033
Epoch 10/30
173/173 [==============================] - 5s 29ms/step - loss: 142.7914 - IoU: 0.7222
Epoch 11/30
173/173 [==============================] - 5s 30ms/step - loss: 125.0813 - IoU: 0.7303
Epoch 12/30
173/173 [==============================] - 5s 29ms/step - loss: 117.4120 - IoU: 0.7320
Epoch 13/30
173/173 [==============================] - 5s 30ms/step - loss: 105.2716 - IoU: 0.7476
Epoch 14/30
173/173 [==============================] - 5s 30ms/step - loss: 97.6738 - IoU: 0.7677
Epoch 15/30
173/173 [==============================] - 5s 29ms/step - loss: 94.4373 - IoU: 0.7686
Epoch 16/30
173/173 [==============================] - 5s 29ms/step - loss: 89.4607 - IoU: 0.7759
Epoch 17/30
173/173 [==============================] - 5s 30ms/step - loss: 81.3200 - IoU: 0.7798
Epoch 18/30
173/173 [==============================] - 5s 29ms/step - loss: 74.8755 - IoU: 0.7986
Epoch 19/30
173/173 [==============================] - 5s 29ms/step - loss: 76.0237 - IoU: 0.7922
Epoch 20/30
173/173 [==============================] - 5s 30ms/step - loss: 69.0520 - IoU: 0.8097
Epoch 21/30
173/173 [==============================] - 5s 30ms/step - loss: 68.3373 - IoU: 0.8103
Epoch 22/30
173/173 [==============================] - 5s 29ms/step - loss: 68.1968 - IoU: 0.8218
Epoch 23/30
173/173 [==============================] - 5s 29ms/step - loss: 66.7474 - IoU: 0.8164
Epoch 24/30
173/173 [==============================] - 5s 30ms/step - loss: 63.5243 - IoU: 0.8180
Epoch 25/30
173/173 [==============================] - 5s 29ms/step - loss: 64.5664 - IoU: 0.8237
Epoch 26/30
173/173 [==============================] - 5s 30ms/step - loss: 71.9686 - IoU: 0.8146
Epoch 27/30
```

```
173/173 [==============================] - 5s 29ms/step - loss: 62.3348 - IoU: 0.8160
Epoch 28/30
173/173 [==============================] - 5s 29ms/step - loss: 57.3480 - IoU: 0.8237
Epoch 29/30
173/173 [==============================] - 5s 31ms/step - loss: 61.3351 - IoU: 0.8187
Epoch 30/30
```

Pick a test image from the given data

```
# Pick a test image, run model, show image, and show predicted bounding box overlaid on the i
import cv2
filename = '/content/drive/My Drive/Lab Internal-I  Residency-IX/images/raccoon-111.jpg'
unscaled = cv2.imread(filename) # Original image for display
```

Resize the image to 128 * 128 and preprocess the image for the MobileNet mode

```
image_height, image_width, _ = unscaled.shape
image = cv2.resize(unscaled, (IMAGE_SIZE, IMAGE_SIZE)) # Rescaled image to run the network
feat_scaled = preprocess_input(np.array(image, dtype=np.float32))
```

```
region = model.predict(x=np.array([feat_scaled]))[0] # Predict the BBox
```

Predict the coordinates of the bounding box for the given test image

Plot the test image using .imshow and draw a boundary box around the image wi the model

```
x0 = int(region[0] * image_width / IMAGE_SIZE) # Scale the BBox
y0 = int(region[1] * image_height / IMAGE_SIZE)

x1 = int((region[2]) * image_width / IMAGE_SIZE)
y1 = int((region[3]) * image_height / IMAGE_SIZE)
```

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import numpy as np
```

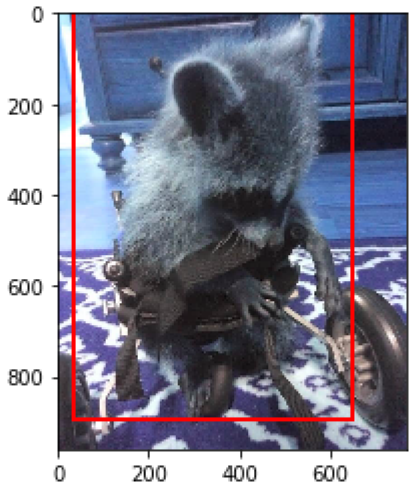
```
# Create figure and axes
fig,ax = plt.subplots(1)
```

```
# Display the image
ax.imshow(unscaled)

# Create a Rectangle patch
rect = patches.Rectangle((x0, y0), (x1 - x0) , (y1 - y0) , linewidth=2, edgecolor='r', facecc

# Add the patch to the Axes
ax.add_patch(rect)

plt.show()
```



Time Series Prediction using LSTM

Download Data

Link: https://datamarket.com/data/set/2324/daily-minimum-temperatures-in-melbourne-australia-19

Description

Daily minimum temperatures in Melbourne, Australia, 1981-1990

Units: Degrees Celcius

Steps before loading

- Rename the column name with temprature values to "Temprature"
- In the last, there is one extra row in the data, remove it by opening the file and save it again.
- There are some values in Temprature column which have a "?" before them, they will give error,
- If you don't want to do these steps, just load the data file given by Great Learning.

Mount google drive

Change your present working directory

Load your data file

```
df_L  = pd.read_csv('/content/drive/My Drive/Lab Internal-I  Residency-IX/daily-minimum-tempe
```

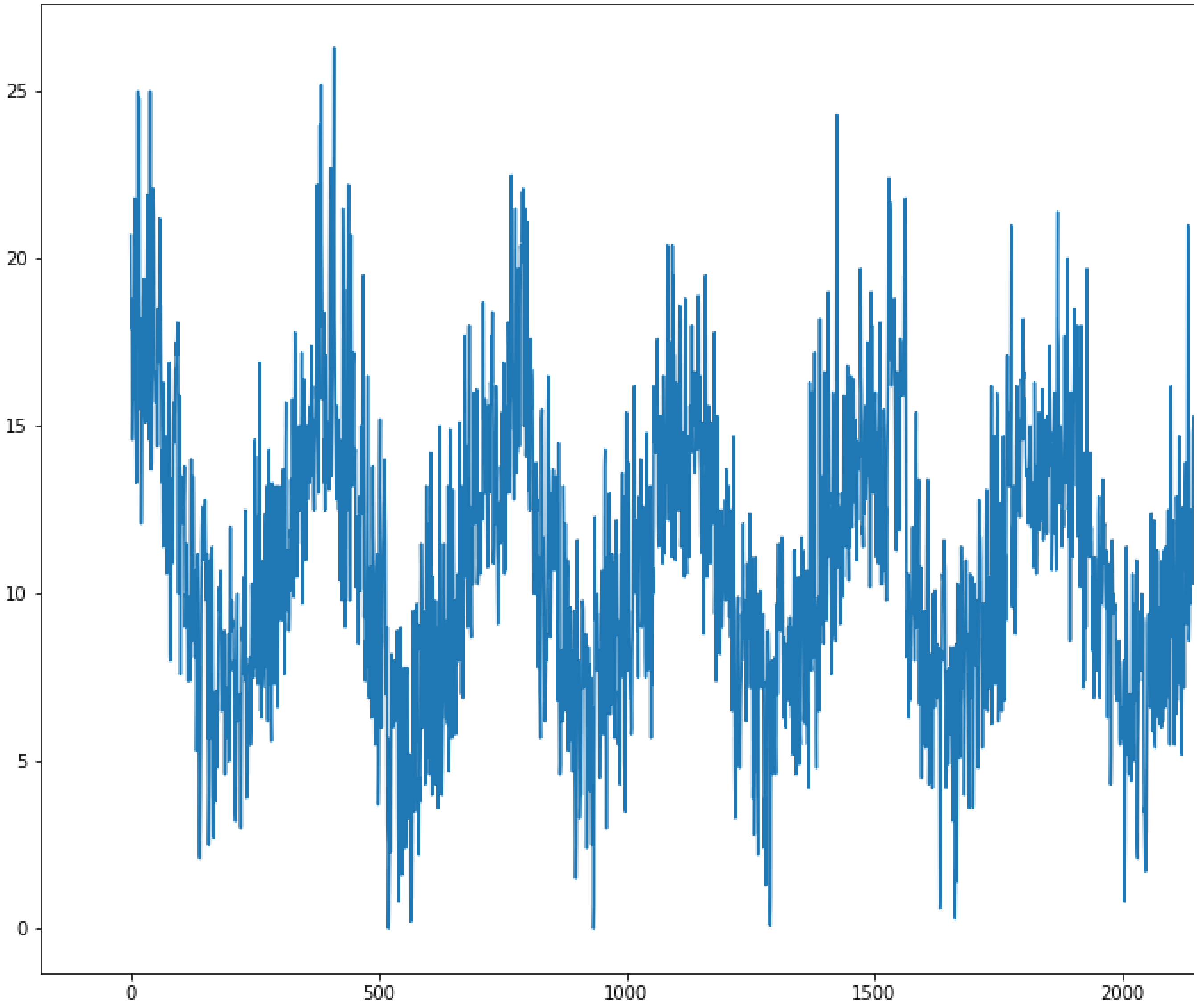
Plot data

```
df_L.head(4)
```

	Date	Temperature
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6

```
import matplotlib.pyplot as plt
```

```
# plot baseline and predictions
plt.figure(figsize=(20,10))
plt.plot(df_L['Temperature'])
plt.show()
```



Descibe your dataframe

Check for null values

```
df_L.isnull().sum()
```

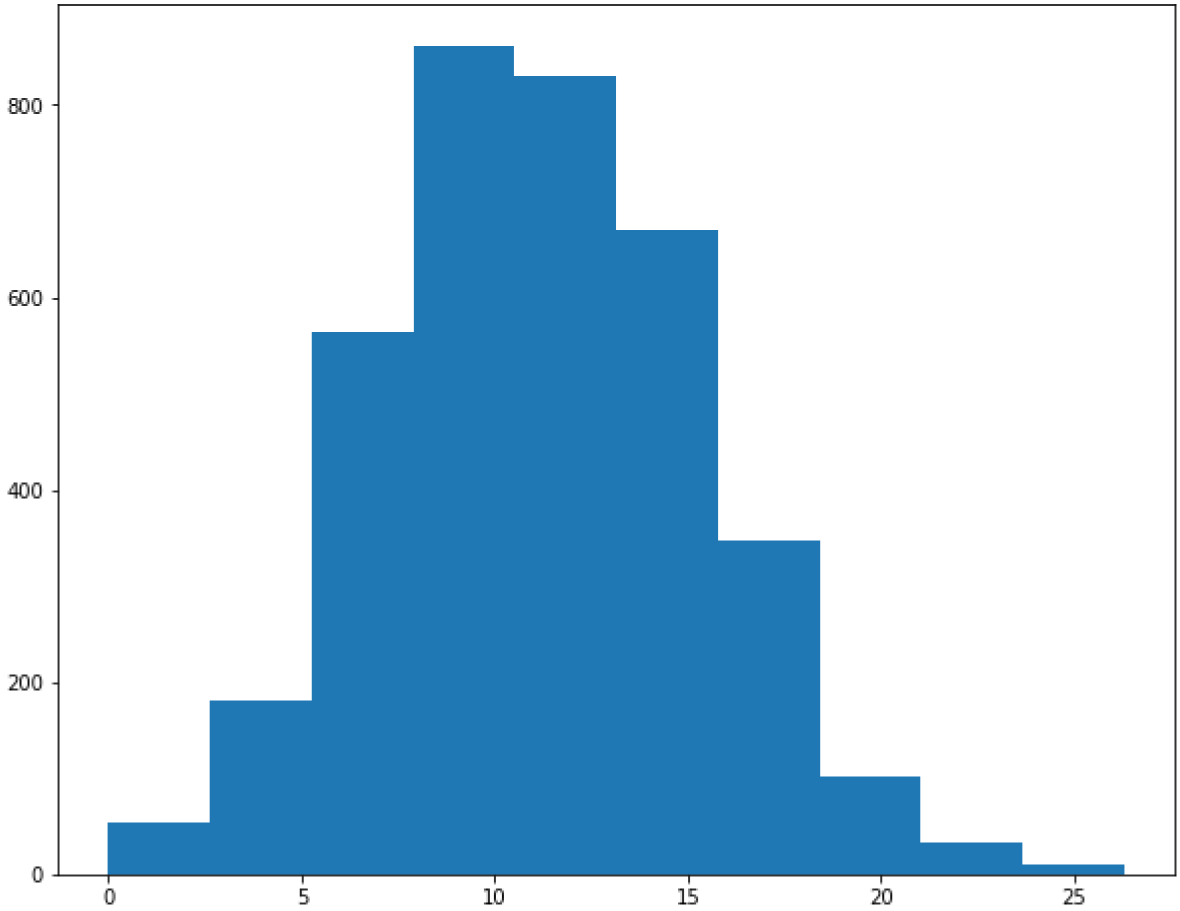
```
Date           0
Temperature    0
dtype: int64
```

Drop null values

```
# No null values
```

Get the representation of the distribution of data in the form of histogram

```
# plot baseline and predictions
plt.figure(figsize=(10,8))
plt.hist(df_L['Temperature'])
plt.show()
```



Check the maximum and minimum values

```
#Check Data Range
print('Min', np.min(df_L))
print('Max', np.max(df_L))
```

```
Min Date          1981-01-01
Temperature                0
dtype: object
Max Date          1990-12-31
Temperature             26.3
dtype: object
```

Normalize the data

```
from sklearn.preprocessing import MinMaxScaler
```

```
df_L.head(4)
```

	Date	Temperature
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6

```
df_L1 = df_L.drop(columns='Date',axis=1)
```

```
#Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(np.array(df_L1))
```

Check the maximum and minimum values of scaled data

```
#Check Data Range
print('Min', np.min(scaled))
print('Max', np.max(scaled))
```

```
Min 0.0
Max 1.0
```

Look into some of the scaled values

```
scaled[0:8]
```

```
array([[0.78707224],
       [0.68060837],
       [0.7148289 ],
       [0.55513308],
       [0.60076046],
       [0.60076046],
       [0.60076046],
       [0.66159696]])
```

Split data into Training and Testing

```
#70% examples will be used for training (in the beginning)
```

```
#70% examples will used for training (in the begining)
train_size = int(len(scaled) * 0.70)

#30% will be used for Test
test_size = len(scaled - train_size)


#Split the data
train, test = scaled[0:train_size, :], scaled[train_size: len(scaled), :]
print('train: {}\ntest: {}'.format(len(train), len(test)))
```

```
train: 2555
test: 1095
```

Print train and test size

Create the sequential data

Map the temprature at a particular time t to the temprature at time t+n, where n is any number you de

For example: to map tempratures of consecutive days, use t+1, i.e. loop_back = 1

Define your function to create dataset

```
#window - how long the sequence will be
def create_dataset(dataset, window=1):

    dataX, dataY = [], []

    for i in range(len(dataset)-window):

        a = dataset[i:(i+window), 0]
        dataX.append(a)
        dataY.append(dataset[i + window, 0])

    return np.array(dataX), np.array(dataY)
```

Use function to get training and test set

```
#Create Input and Output
window_size = 1
X_train, y_train = create_dataset(train, window_size)
X_test, y_test = create_dataset(test, window_size)
```

Transform the prepared train and test input data into the expected structure using numpy.

```
#Make it 3 Dimensional Data - needed for LSTM
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
print(X_train.shape)
print(X_test.shape)
```

```
(2554, 1, 1)
(1094, 1, 1)
```

Define Model

Define sequntial model, add LSTM layer and compile the model

```
import tensorflow as tf
```

```
tf.keras.backend.clear_session()
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(32, input_shape=(window_size, 1)))
model.add(tf.keras.layers.Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam' , metrics=['mse'])
```

```
(X_train.shape,y_train.shape, X_test.shape, y_test.shape)
```

```
((2554, 1, 1), (2554,), (1094, 1, 1), (1094,))
```

Summarize your model

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

Train the model

```
dense (Dense)                (None, 1)                 33
```

```
model.fit(X_train, y_train, epochs=200, validation_data=(X_test, y_test), batch_size=32)
```

```
Train on 2554 samples, validate on 1094 samples
Epoch 1/200
2554/2554 [==============================] - 1s 347us/sample - loss: 0.1061 - mean_squar
Epoch 2/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0178 - mean_squar
Epoch 3/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0141 - mean_squar
Epoch 4/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0133 - mean_squar
Epoch 5/200
2554/2554 [==============================] - 0s 126us/sample - loss: 0.0126 - mean_squar
Epoch 6/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0119 - mean_squar
Epoch 7/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0114 - mean_squar
Epoch 8/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0109 - mean_squar
Epoch 9/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0106 - mean_squar
Epoch 10/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0103 - mean_squar
Epoch 11/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0102 - mean_squar
Epoch 12/200
2554/2554 [==============================] - 0s 128us/sample - loss: 0.0102 - mean_squar
Epoch 13/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0101 - mean_squar
Epoch 14/200
2554/2554 [==============================] - 0s 129us/sample - loss: 0.0101 - mean_squar
Epoch 15/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0101 - mean_squar
Epoch 16/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0101 - mean_squar
Epoch 17/200
2554/2554 [==============================] - 0s 133us/sample - loss: 0.0101 - mean_squar
Epoch 18/200
2554/2554 [==============================] - 0s 130us/sample - loss: 0.0101 - mean_squar
Epoch 19/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 20/200
2554/2554 [==============================] - 0s 124us/sample - loss: 0.0100 - mean_squar
Epoch 21/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 22/200
2554/2554 [==============================] - 0s 129us/sample - loss: 0.0100 - mean_squar
Epoch 23/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 24/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0101 - mean_squar
Epoch 25/200
2554/2554 [==============================] - 0s 120us/sample - loss: 0.0101 - mean_squar
Epoch 26/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0101 - mean_squar
Epoch 27/200
2554/2554 [==============================] - 0s 124us/sample - loss: 0.0101 - mean_squar
Epoch 28/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0100 - mean_squar
```

```
Epoch 29/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0100 - mean_squar
Epoch 30/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 31/200
2554/2554 [==============================] - 0s 128us/sample - loss: 0.0100 - mean_squar
Epoch 32/200
2554/2554 [==============================] - 0s 133us/sample - loss: 0.0100 - mean_squar
Epoch 33/200
2554/2554 [==============================] - 0s 128us/sample - loss: 0.0100 - mean_squar
Epoch 34/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0101 - mean_squar
Epoch 35/200
2554/2554 [==============================] - 0s 121us/sample - loss: 0.0101 - mean_squar
Epoch 36/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0101 - mean_squar
Epoch 37/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 38/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0101 - mean_squar
Epoch 39/200
2554/2554 [==============================] - 0s 115us/sample - loss: 0.0101 - mean_squar
Epoch 40/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 41/200
2554/2554 [==============================] - 0s 131us/sample - loss: 0.0100 - mean_squar
Epoch 42/200
2554/2554 [==============================] - 0s 124us/sample - loss: 0.0100 - mean_squar
Epoch 43/200
2554/2554 [==============================] - 0s 126us/sample - loss: 0.0100 - mean_squar
Epoch 44/200
2554/2554 [==============================] - 0s 114us/sample - loss: 0.0100 - mean_squar
Epoch 45/200
2554/2554 [==============================] - 0s 115us/sample - loss: 0.0100 - mean_squar
Epoch 46/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 47/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0100 - mean_squar
Epoch 48/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0100 - mean_squar
Epoch 49/200
2554/2554 [==============================] - 0s 130us/sample - loss: 0.0100 - mean_squar
Epoch 50/200
2554/2554 [==============================] - 0s 124us/sample - loss: 0.0100 - mean_squar
Epoch 51/200
2554/2554 [==============================] - 0s 114us/sample - loss: 0.0100 - mean_squar
Epoch 52/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 53/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0102 - mean_squar
Epoch 54/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 55/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0100 - mean_squar
Epoch 56/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0101 - mean_squar
Epoch 57/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0101 - mean_squar
```

```
Epoch 58/200
2554/2554 [==============================] - 0s 126us/sample - loss: 0.0100 - mean_squar
Epoch 59/200
2554/2554 [==============================] - 0s 130us/sample - loss: 0.0100 - mean_squar
Epoch 60/200
2554/2554 [==============================] - 0s 124us/sample - loss: 0.0101 - mean_squar
Epoch 61/200
2554/2554 [==============================] - 0s 136us/sample - loss: 0.0100 - mean_squar
Epoch 62/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0101 - mean_squar
Epoch 63/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 64/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0100 - mean_squar
Epoch 65/200
2554/2554 [==============================] - 0s 113us/sample - loss: 0.0100 - mean_squar
Epoch 66/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0100 - mean_squar
Epoch 67/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0100 - mean_squar
Epoch 68/200
2554/2554 [==============================] - 0s 120us/sample - loss: 0.0100 - mean_squar
Epoch 69/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 70/200
2554/2554 [==============================] - 0s 120us/sample - loss: 0.0101 - mean_squar
Epoch 71/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 72/200
2554/2554 [==============================] - 0s 112us/sample - loss: 0.0100 - mean_squar
Epoch 73/200
2554/2554 [==============================] - 0s 121us/sample - loss: 0.0100 - mean_squar
Epoch 74/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 75/200
2554/2554 [==============================] - 0s 115us/sample - loss: 0.0100 - mean_squar
Epoch 76/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0100 - mean_squar
Epoch 77/200
2554/2554 [==============================] - 0s 120us/sample - loss: 0.0100 - mean_squar
Epoch 78/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0101 - mean_squar
Epoch 79/200
2554/2554 [==============================] - 0s 115us/sample - loss: 0.0100 - mean_squar
Epoch 80/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0100 - mean_squar
Epoch 81/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 82/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 83/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0101 - mean_squar
Epoch 84/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0100 - mean_squar
Epoch 85/200
2554/2554 [==============================] - 0s 114us/sample - loss: 0.0100 - mean_squar
Epoch 86/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
```

```
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 87/200
2554/2554 [==============================] - 0s 113us/sample - loss: 0.0100 - mean_squar
Epoch 88/200
2554/2554 [==============================] - 0s 110us/sample - loss: 0.0100 - mean_squar
Epoch 89/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0101 - mean_squar
Epoch 90/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 91/200
2554/2554 [==============================] - 0s 123us/sample - loss: 0.0100 - mean_squar
Epoch 92/200
2554/2554 [==============================] - 0s 116us/sample - loss: 0.0100 - mean_squar
Epoch 93/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0100 - mean_squar
Epoch 94/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0100 - mean_squar
Epoch 95/200
2554/2554 [==============================] - 0s 128us/sample - loss: 0.0100 - mean_squar
Epoch 96/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 97/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0101 - mean_squar
Epoch 98/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 99/200
2554/2554 [==============================] - 0s 117us/sample - loss: 0.0100 - mean_squar
Epoch 100/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0100 - mean_squar
Epoch 101/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0100 - mean_squar
Epoch 102/200
2554/2554 [==============================] - 0s 126us/sample - loss: 0.0100 - mean_squar
Epoch 103/200
2554/2554 [==============================] - 0s 118us/sample - loss: 0.0100 - mean_squar
Epoch 104/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 105/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0100 - mean_squar
Epoch 106/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0100 - mean_squar
Epoch 107/200
2554/2554 [==============================] - 0s 136us/sample - loss: 0.0100 - mean_squar
Epoch 108/200
2554/2554 [==============================] - 0s 144us/sample - loss: 0.0101 - mean_squar
Epoch 109/200
2554/2554 [==============================] - 0s 127us/sample - loss: 0.0100 - mean_squar
Epoch 110/200
2554/2554 [==============================] - 0s 130us/sample - loss: 0.0100 - mean_squar
Epoch 111/200
2554/2554 [==============================] - 0s 125us/sample - loss: 0.0100 - mean_squar
Epoch 112/200
2554/2554 [==============================] - 0s 119us/sample - loss: 0.0100 - mean_squar
Epoch 113/200
2554/2554 [==============================] - 0s 122us/sample - loss: 0.0100 - mean_squar
Epoch 114/200
2554/2554 [==============================] - 0s 119us/sample - loss: 0.0100 - mean_squar
Epoch 115/200
```