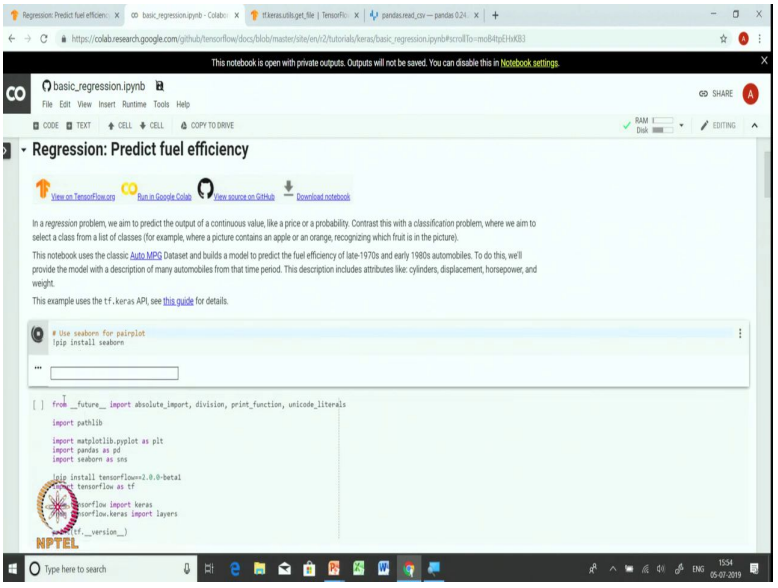
Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Lecture – 15
Regression

[FL]. In this session, we will build a deep learning model for Regression. In the last session, we built a deep learning model for classifying fashion accessories into one of the ten categories. So, in this particular session we will build a regression model for predicting fuel efficiency of vehicles.

(Refer Slide Time: 00:45)

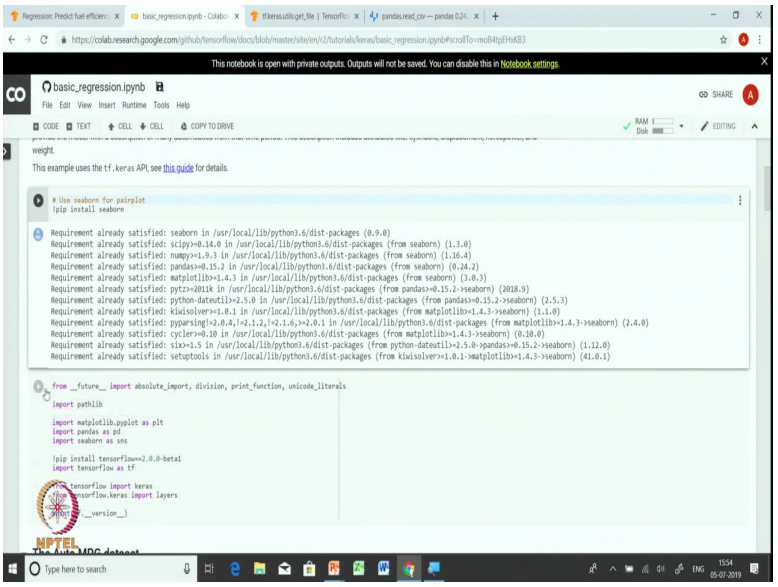


In regression problem, the output is a real number which is like a price or fuel efficiency measured by miles per gallon. Contrasting this with classification problem where we aim to select a class from list of classes, regression problem strives to predict a real number.

In this particular exercise we will use a classic Auto MPG Dataset and build a model to predict the fuel efficiency of 1970s and 80s automobiles. To do this, we will provide a model with description of many automobiles from that time period. These features include cylinders, displacement, horsepower and weight of the automobile. As usual before starting the

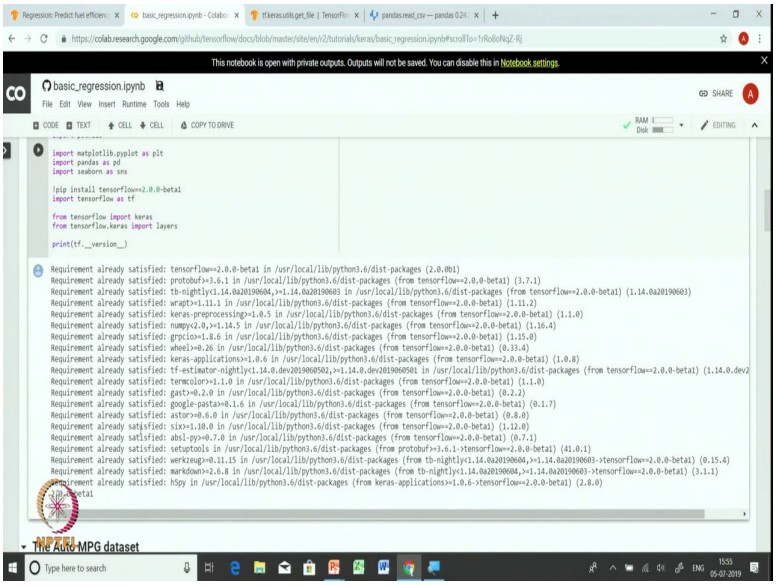
notebook let us connect to a collab runtime and install the useful softwares which are not present. First we will install a software called seaborn for plotting a pair plot.

(Refer Slide Time: 01:59)



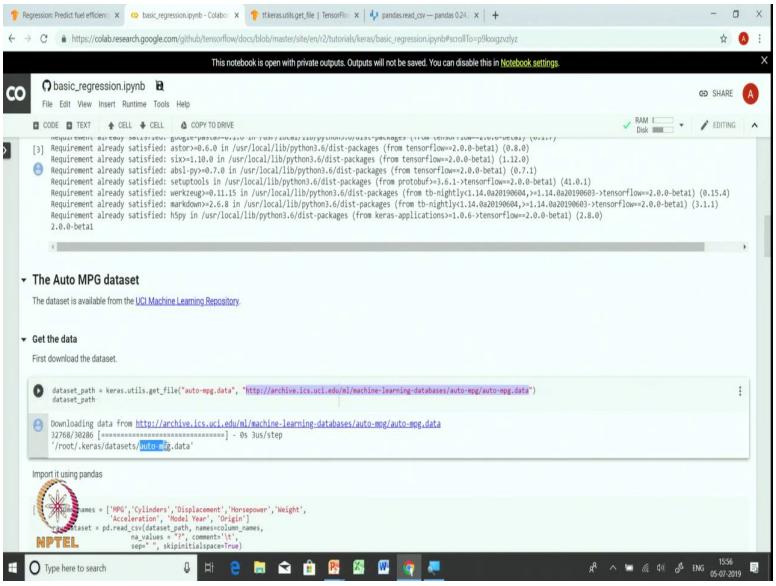
So, what is the seaborn is installed we will import some of the plotting libraries like matplotlib.pyplot. We will import pandas for manipulating data. Then we will we will import seaborn for pair plot and we will also install tensorflow 2.0 and import keras and layers library from tensorflow.

(Refer Slide Time: 02:29)



Let us run this particular cell. At the end of the cell we make sure that we have the right version of tensor flow present in our colab runtime. We ensure that by printing the tf version which is 2.0.0 beta 1 which is the desired version for this particular exercise.

(Refer Slide Time: 02:49)

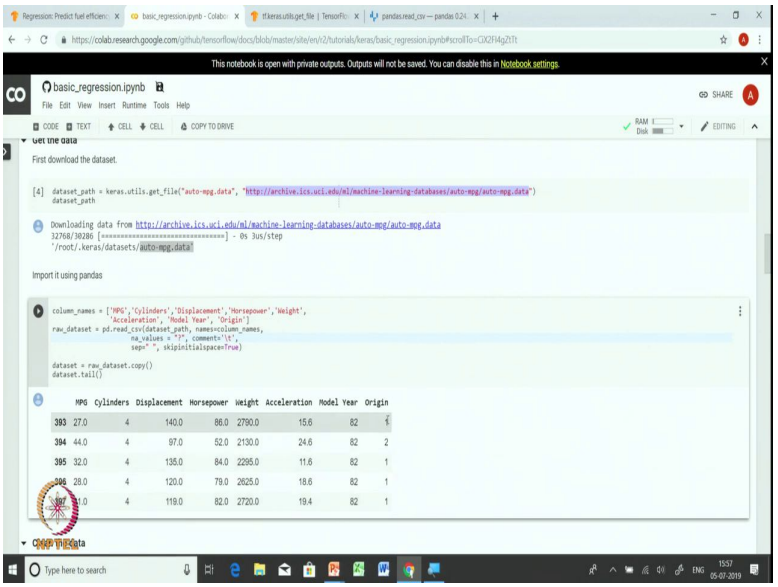


The auto MPG dataset is available from UCI machine learning repository. Our first job is to get this particular data. In the last exercise, we the dataset was present in the tensor flow data sets and it was easier for us to import and load the data in tensorflow. In the case of auto

MPG, this data set is not available in tensorflow so, we will have to first download the file and then load the data using pandas and we will import the data from pandas data frame into tensorflow. Let us first understand that particular process.

So, at the first step we will download the data. We will use keras utils.get_file() function for that the first argument is where we want to store the data and in the second argument here is the url of the file containing the auto mpg data. So, let us run this which will download this particular cell will download the data and the data is now present in auto MPG dot data as printed by the dataset path.

(Refer Slide Time: 03:55)



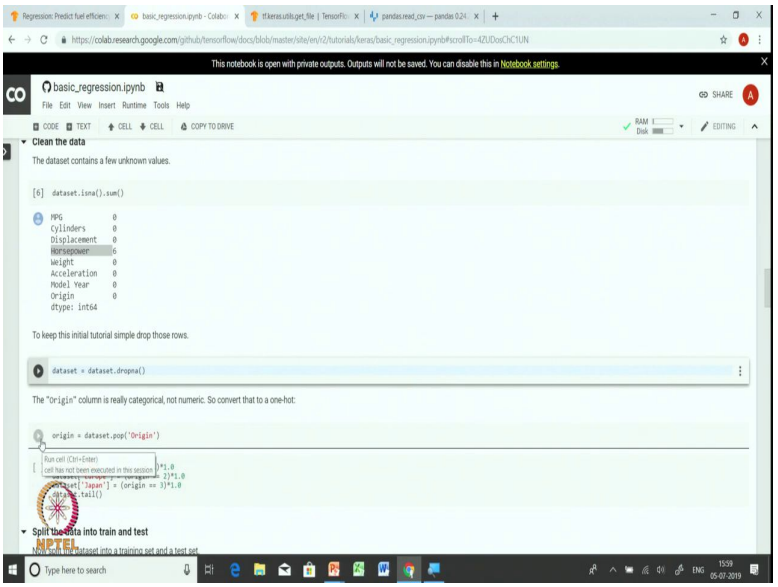
Let us import the data using pandas dataframe. We will first list out the column names. We have column names like Miles Per Gallon, Cylinders, Displacement, Horsepower, Weight, Acceleration, Model Year and the Origin. We will use pandas.read_csv() function because this data is present in a CSV format.

This particular function takes the data set, path column names and we tell how to handle the NA values. So, we want to replace NA values by question mark. We want to ignore anything after the tap and that is specified using the comment argument. The separator is a space and we want to skip initial spaces in the file. After specifying all these arguments we are able to

read the content of auto MPG dataset into the data frame raw_dataset. We will make a copy of raw_dataset as dataset. Let us run this code cell and examine what is there in the dataset.

So, we printed last five rows of dataset using dataset.tail() command and you can see here there are features like MPGs, Cylinders, Displacement, Horsepower and so on. Most of the features are numeric features in nature whereas, origin is a discrete feature it just has value 1 or 2. So, this particular part takes care of downloading the data and loading that into pandas dataframe. Once the data is loaded our next job is to clean the data and remove null values and perform normalization on the dataset. Let us take steps to preprocess the data.

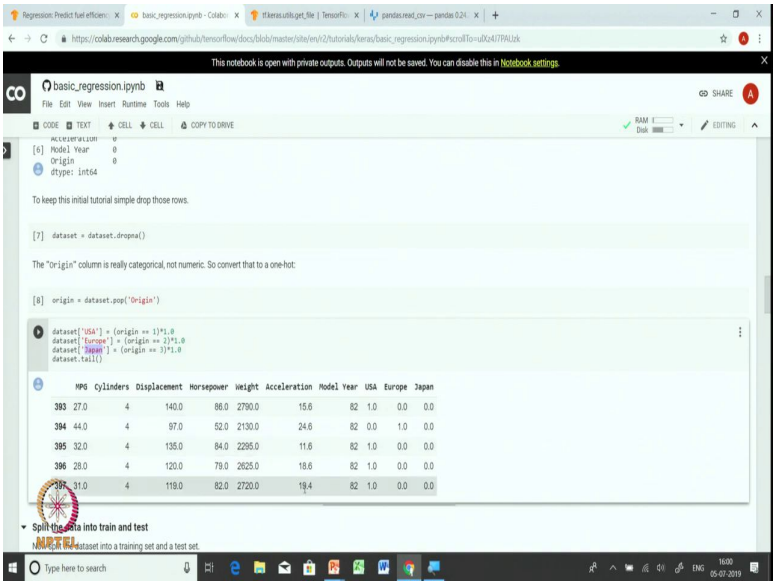
(Refer Slide Time: 06:01)



So, let us first find out if dataset has got any null values. So, we essentially use isna() function and then followed by a some function to identify columns where null values are present. As you can see on the screen, horsepower column has few null values. In order to keep this tutorial or this exercise simple we will simply drop these rows. So, we will drop all the rows containing null values using function dropna().

Next we will convert some of the categorical attributes into one-hot encoding. So, we just saw that origin is a categorical attribute and we will use one-hot encoding to convert that into something that is usable by the machine learning algorithm.

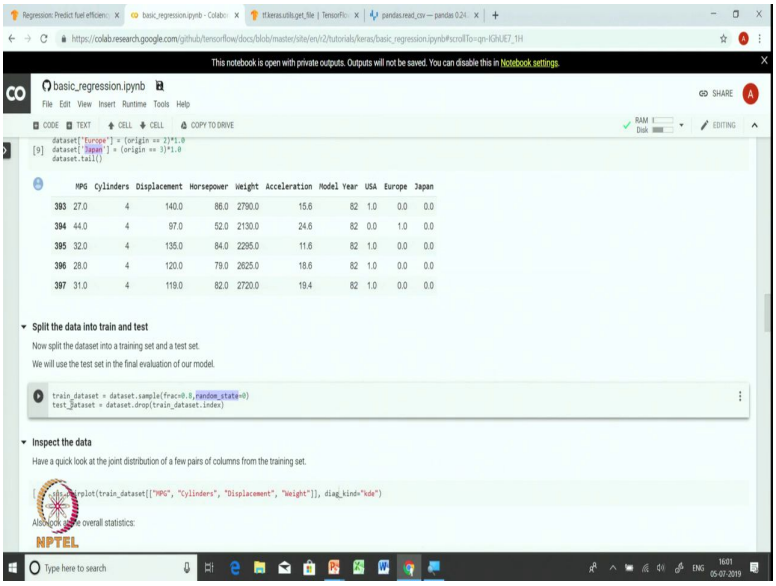
(Refer Slide Time: 06:57)



So, let us pop the; let us pop the origin column from the data set and replace the origin column with three values because origin originally has three values USA, Europe and Japan. So, for USA we use the value of 1, for Europe we will use the value of 2 and for Japan we will use the value of 3.

So, let us do this particular step of converting the categorical attribute into numeric attribute and you can see that whenever USA is present you will see the value 1 appearing; whenever USA is present Europe and Japan are obviously, not present so, the values are 0; whenever Europe is present you will see a value of 1 corresponding to a column Europe; whenever value is 1 that column corresponds to whenever there is a value of one in the column of Japan that corresponds to the third value in the original data set. So, this is how we converted our numerical attribute into a categorical attribute and we have a transform data set.

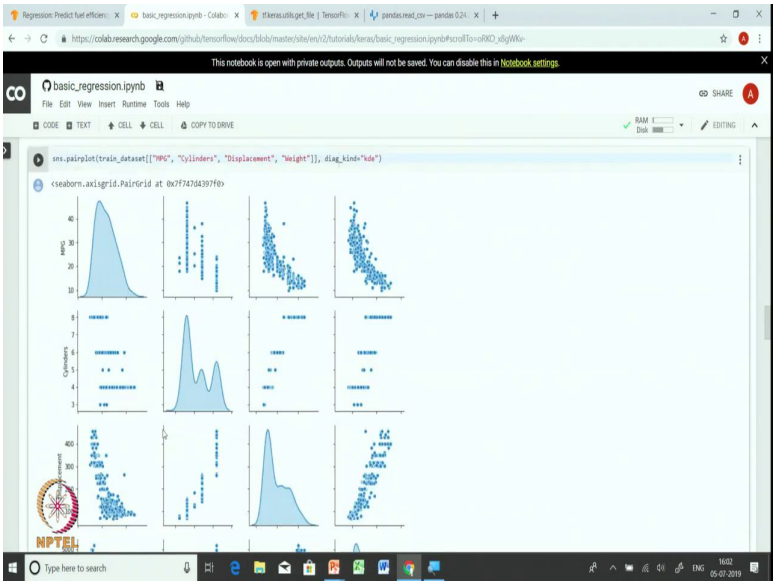
(Refer Slide Time: 08:07)



Now that we have pre processed and cleaned dataset, the next job is to split the data into test and training set. So, we will simply use dataset.sample() and we set random seed so that every time we run this particular collab we get essentially the same training set. So, let us run this particular command this will select 80 percent of the examples in the training and anything that is in the training we drop and the remaining examples are copied into test_dataset.

So, to running this colab we have two data sets - train_dataset and test_dataset containing training and test examples respectively. Let us have a quick look at the joint distribution of a few pairs of columns from the training set. We will use sns.pairplot() for plotting the joint distribution of a few pairs of columns and we do that for columns like MPG miles per gallon which is our target column or the column that we want to predict Cylinders, Displacement and Weights. Let us run this particular code cell and look at the pairplot.

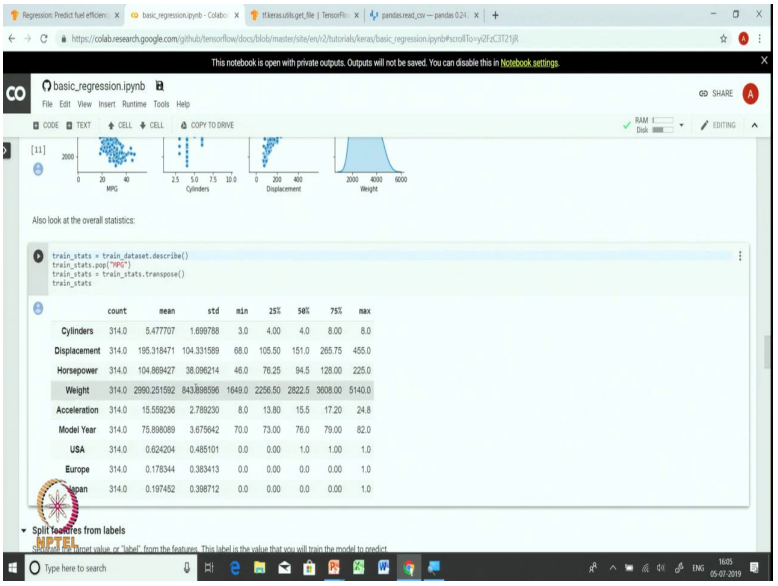
(Refer Slide Time: 09:23)



So, in this particular pairplot on diagonal you have you can see the distribution of the values in each of the features. The off-diagonal elements essentially show the relationship between the feature on the row and the feature on the column.

As you can see that there is hardly any relationship between cylinder and MPG. Cylinder and MPG seems to be uncorrelated whereas, displacement and weight have some correlation with MPG because as weight goes up MPG seems to be coming down the same thing is happening with displacement as displacement increases the MPG seems to be coming down. You can see that MPG and weight are correlated features because as the weight increases displacement also tends to increase. So, we can get some useful insights into the features by looking at this plot and how these features affect the outcome can also be seen in this pair plot.

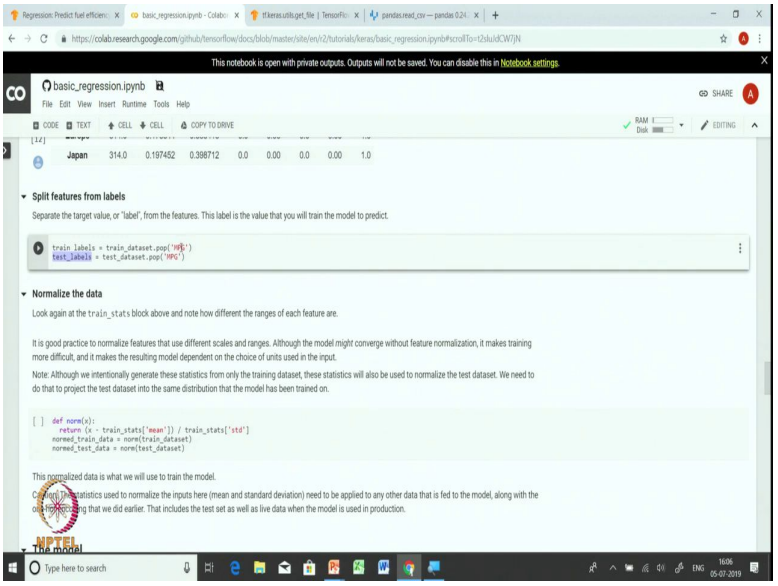
(Refer Slide Time: 11:01)



We first use describe() command on the dataset and obtain training stats, we remove the field for miles per gallon because there is a field that we want to predict and we do the transpose of the stats so that we can display it in a nice tabular fashion. So, you can see all the features of the model on the row side and on the column we have various statistics for each of the features. So, you can see that there are 314 rows in the data set that is why we have 314 as the count for each of the feature.

Then we can look at the mean of every feature, then the standard deviation and pretty much the min, max and few quartiles in between. As we can see that different features are on different scales. For example, minimum number of cylinders is 3 and the maximum number of cylinders is 8; minimum weight is 1649 maximum weight is 5114. So, you can see that the features are on a different scale and in order to bring them on the same scale later we will perform normalization operation.

(Refer Slide Time: 12:21)

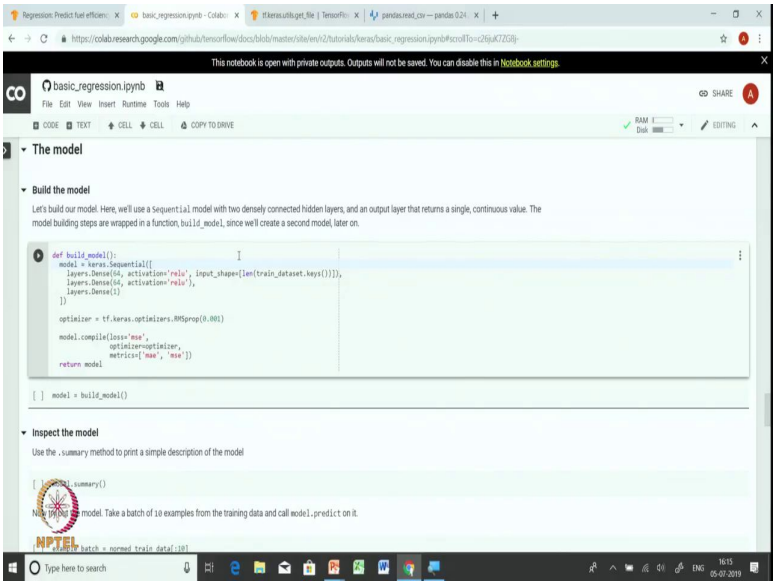


Before doing normalization we will split features from the label. So, right now in our train_dataset we have features as well as labels. So, we will use pop command and we will remove the column corresponding to the label the label column. So, we essentially remove miles per gallon column and that gives us the training label and using the pop command on the test data we get the test label. Let us normalize the data leaving aside the labeling column.

Now, this normalization is a very important process. So, we want to make sure that we do we perform exactly the same normalization on test data as a training data or any other future data set that is coming to the model for prediction we apply the same normalization. So, in order to do that we will we have already calculated a train statistics. We define a normalization function that does the z score normalization which is defined as the value minus mean divided by the standard deviation. So, we calculate we perform normalization on the training data as well as test data.

And, we will also store this normalization parameters which are present in train underscore test and we will apply them for the prediction as well as any test on any other test instances. Now, that we have explored the data and pre-processed it, the next step is to build a model. Here we will try to build a neural network model to perform the regression task. Let us look at the architecture of neural network model that we will be building.

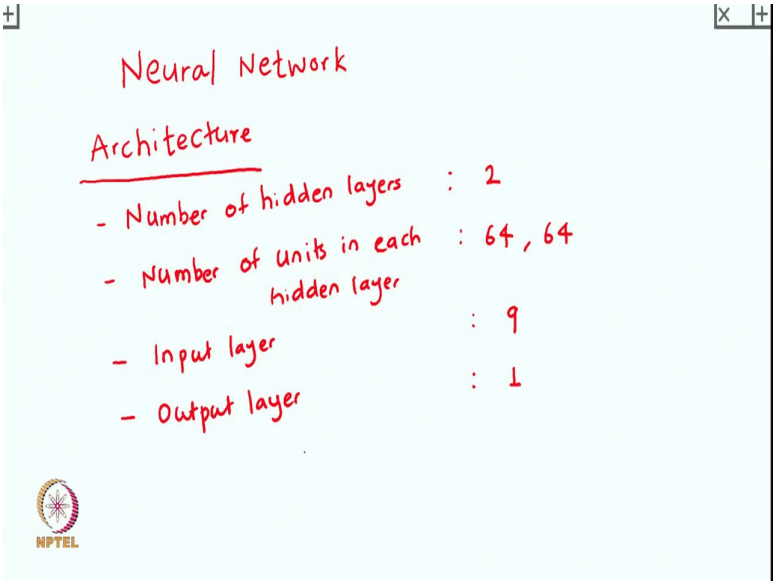
(Refer Slide Time: 14:19)



Before proceeding further I would like you to pay attention that we have 9 features in our model. We will use a keras sequential model or a feed forward neural network with a couple of hidden layers.

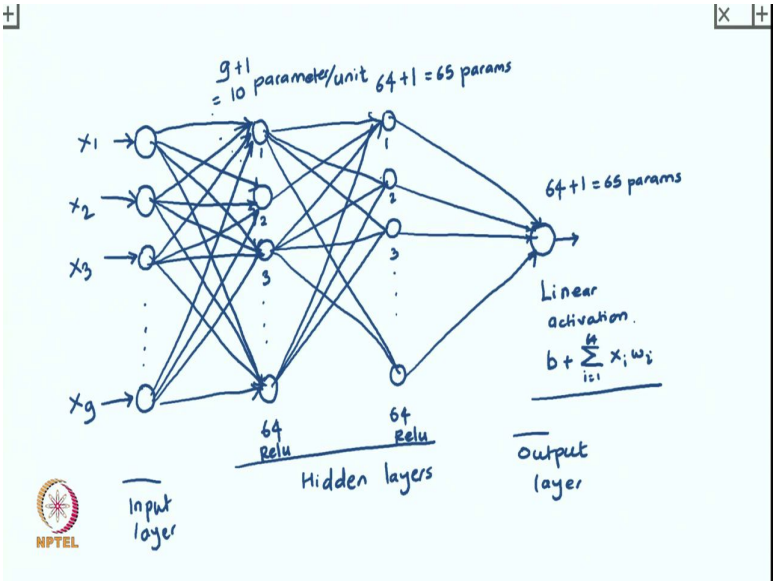
The first hidden layer is a dense layer with 64 unit, the second layer is also another dense layer is 64 units. In both the layers we use activation as relu and we have finally, a dense layer with exactly 1 unit. Let us look at this particular architecture on a board.

(Refer Slide Time: 14:53)



So, again coming back to neural network you have to fix the architecture of the neural network. We need to specify the number of hidden layers and number of units in each hidden layer. You also specify the input layer or what is the input looking like and the output layer. So, here we are using two hidden layers it is using 64 units say you have 9 features and the output layer we just have 1 unit.

(Refer Slide Time: 16:25)



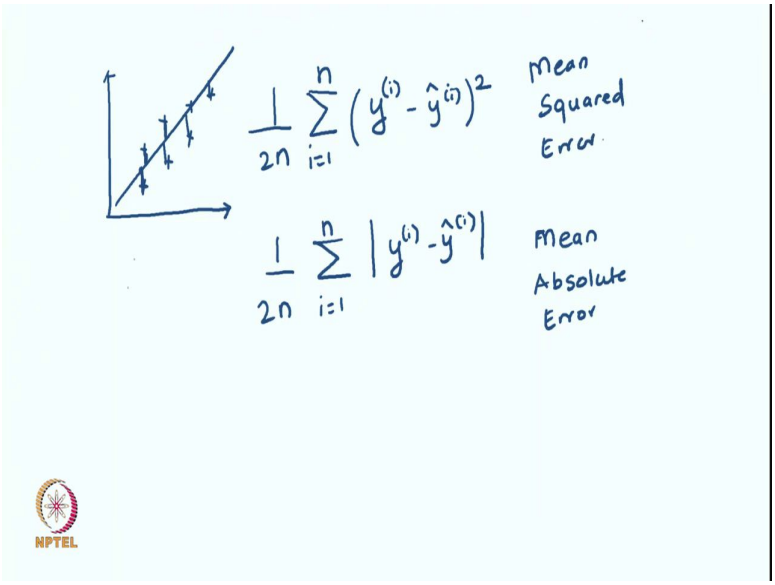
We are using dense layers and in dense layers, we connect each node from the previous layer to all the 64 units in the current layer. There is an additional bias unit on each of the hidden units which I am not showing explicitly. We have a total of 10 parameters per unit in the first layer.

In the second layer, here we have 64 inputs from the previous layer plus one bias parameter. So, we have 65 parameters per hidden unit. The second thing you have to care about is what kind of activation we will be using. We will use relu as an activation function in the hidden layers and linear activation in the output layer.

So, you can see here we wrote a special function called build_model() and this particular part of the code is building the model layers.dense defines the first layer where we use dense units we are going to use 64 such kind of units using activation of relu and the input shape is exactly equal to the number of features that we are using in the training data set.

Next we also stack another dense layer with 64 units which uses relu activation function and finally, we have an output layer containing one dense unit. After specifying the model let us also specify the optimizer and the loss function. So, in the case of regression we use mean squared loss as loss function which is a standard loss function for regression and we use RMSprop as an optimizer in using RMSprop with learning rate of 0.001 and we track metrics like mean absolute error and mean squared error. Let us look at mean squared error and mean absolute error and let us look at mathematically what do they mean.

(Refer Slide Time: 21:57)

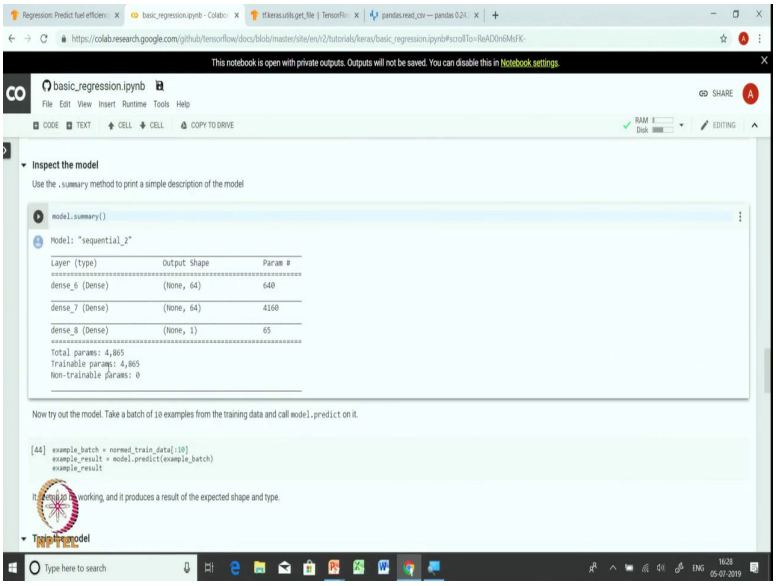


So in the regression framework what happens is that we learn; so, this is our training data for example and let us say this is the model that we learnt. Now, we make end up making some error on each of the points, but you can see that the point on the line is the predicted point and the point denoted by cross is the actual point. So, there is a difference over here there is some difference, here there is some difference here and so on.

So, if we essentially look at every value of y which is y i or let us use slightly different notation. So, we take i-th value of y and we look at the predicted value of i-th example and squared it. So, this defined as squared error for example, if you sum it across all the example we get sum squared error and if we divide it by 2n we get mean squared error.

Instead of doing the square instead if we just use look at the difference or mod of difference between the true value and the predicted value and average it out this is called as mean absolute error and this is called as mean squared error. Let us go back to collab and now train the model. So, let us run this particular code cell and let us build a model. So, now we have a model. Let us see how model looks like.

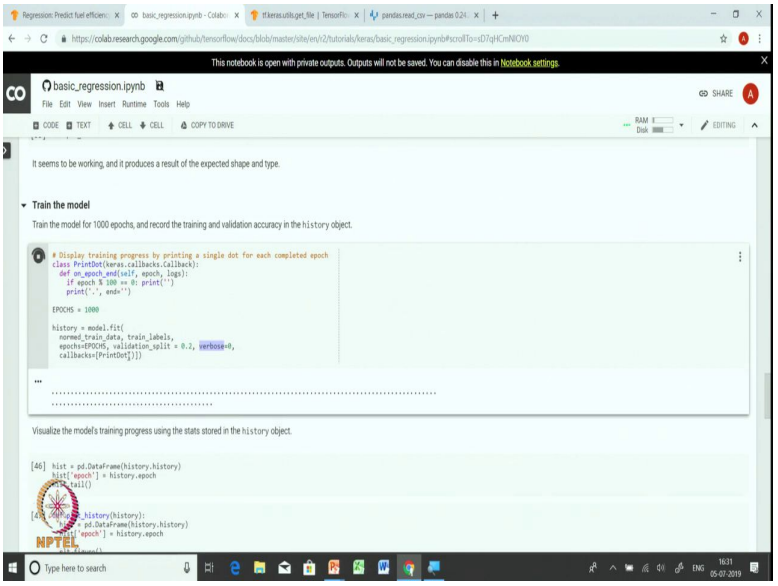
(Refer Slide Time: 24:15)



We can see from the model summary that we have a sequential model or a feed forward neural network. We are using essentially three dense layer; two of them are hidden layer and the last layer is an output laye with exactly one output.

Now, that we have built a model let us try the model on a few examples. Let us take a batch of 10 examples from their training data and call model.predict() on it. Mind you, I mean we have not yet trained the model, but we are using randomly initialize rates and seeing whether model.predict() works.

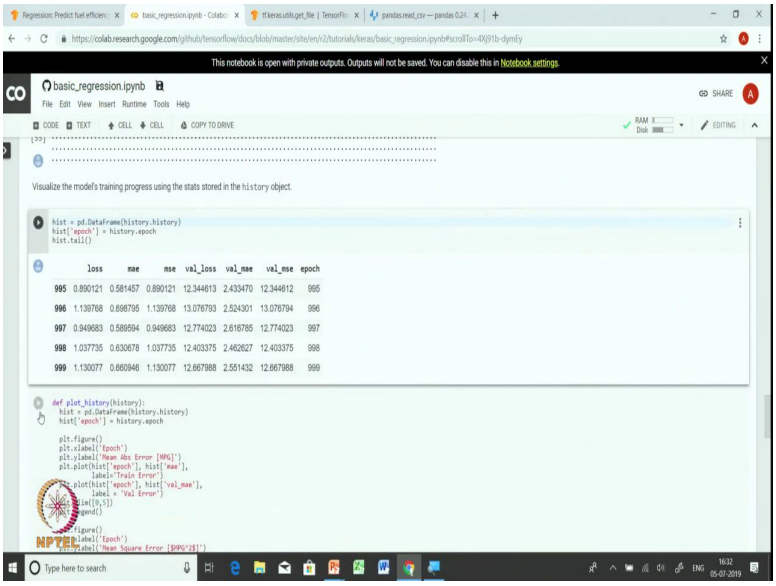
(Refer Slide Time: 25:11)



Yes, it seems to be working and it produces a result of expected shape and type. So, we use this in order to make sure that the model has been set up properly and all shapes of the tensors are set as expected. Let us train the model for 1000 epochs and record training and validation accuracy in the history object.

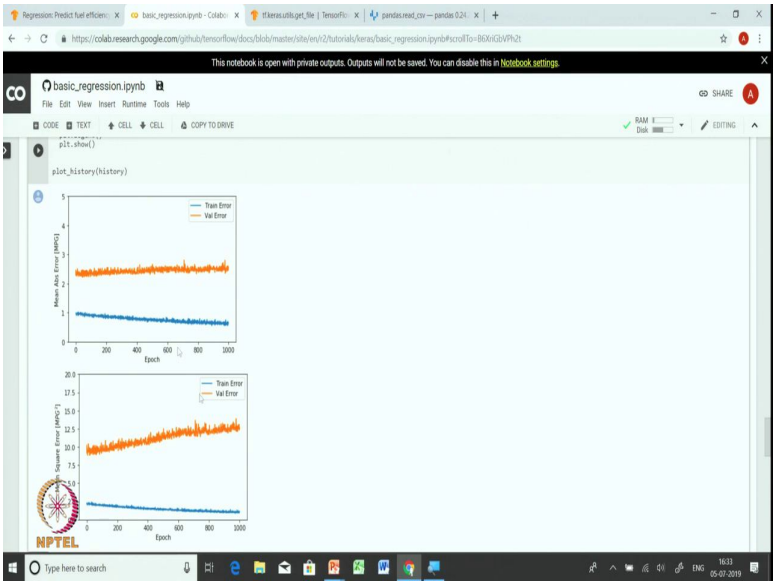
We give the normalized training data and corresponding labels as input to mode.fit function and we record all the history in the history object that is returned by model.fit function. Let us train the model. So, model is training as you can see from the progress and once model is trained we will visualize a history data whatever is told in the history data it seems model is training is complete.

(Refer Slide Time: 26:23)



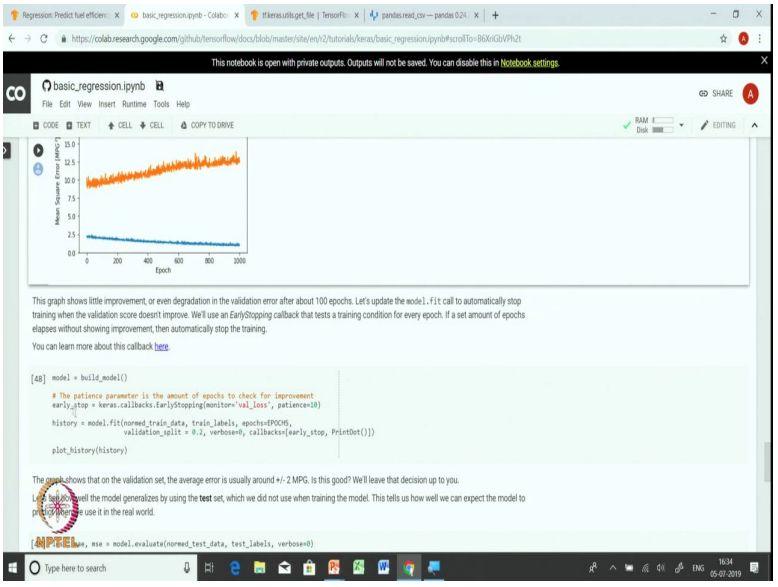
So, let us look at the last five rows of history data. History will have 1000 rows; one for every epoch and you can see that how the loss, so the loss is actually it is very interesting to note here loss is quite fluctuating loss is going up than coming down and again going up and again up. So, let us look at what is happening to the validation loss. Validation loss seem to be going up then coming down and here it has gone up. So, let us plot the history and see how it looks like.

(Refer Slide Time: 27:07)



Interesting, so, you can see that training data mean in terms of mean absolute error the training error is going down, but validation error is going up. The same thing we observe in the mean squared error that while training error is going down validation error does not seem to be improving.

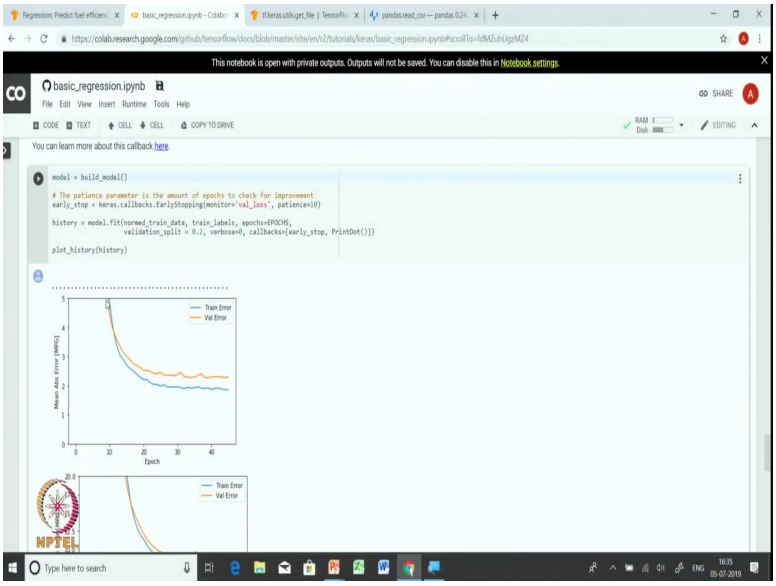
(Refer Slide Time: 27:33)



So, this points to some kind of a; some kind of an over fitting problem in this particular data set. So, we will in order to overcome this particular over fitting problem, we will try to use early stopping as a means of correcting this issue. So we will stop the model before it starts over fitting. So, we will look at we will keep an eye on validation error and use early stopping callback for stopping the model before it starts over fitting.

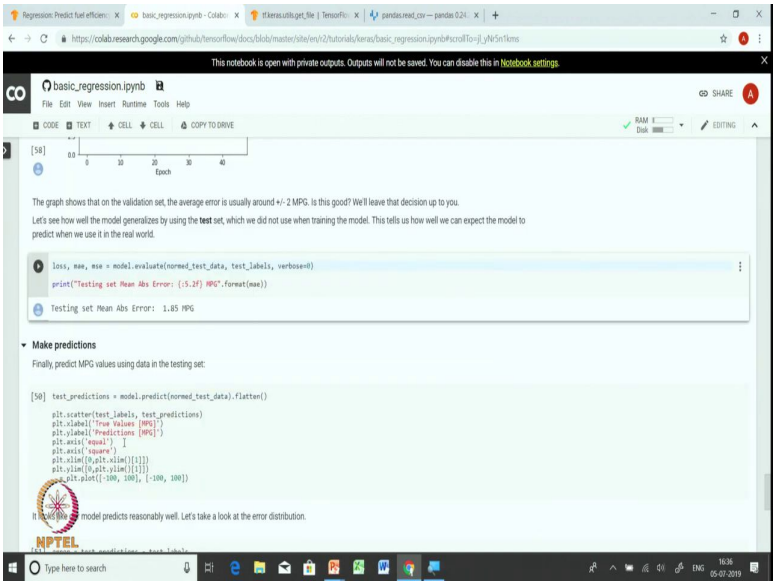
So, early stopping callback is set up in this particular way; we use keras.callbacks.EarlyStopping where we use validation loss as a monitoring mechanism and we wait for 10 epochs. If in 10 epochs validation loss does not improve then we will then we decide to stop the model. And, we use and let us train the model again with the early stopping callback and we will plot the history to see how model does this time.

(Refer Slide Time: 28:41)



The model looks much better compared to the earlier model. The earlier model's training error seems to improve while validation error became worse or at least stayed the same. So that was an overfitting problem, but after doing early stopping regularization; regularization is one way of addressing the over fitting problem. So, we applied early stopping regularization here and after applying early stopping regularization we see that training loss and validation loss or validation error both seems to be improving as we keep training for more epochs.

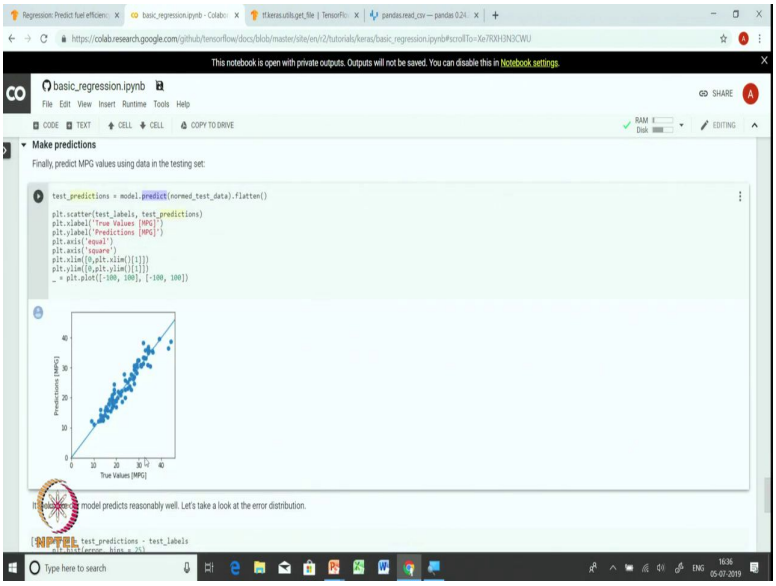
(Refer Slide Time: 29:45)



So, let us evaluate the model on the test data and obtain numbers like mean absolute error and mean squared error. So, we have mean absolute error was of 1.85 miles per gallon. So, now, we have a model and we will use the model to predict the miles per gallon for the test data.

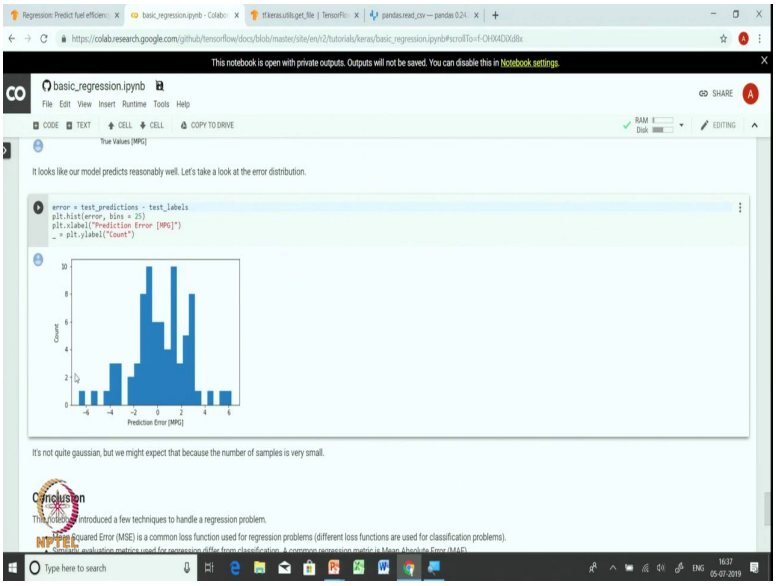
So, here we will give normalized test data has an input and ask the model to predict miles per gallon. Let us plot the predicted value of miles per gallon. So, we have true values of miles per gallon.

(Refer Slide Time: 30:23)



And, the predicted values of miles per gallon and you can see that in most of the cases predicted values the true value and the predicted values are actually close by and are model seem to be working reasonably well.

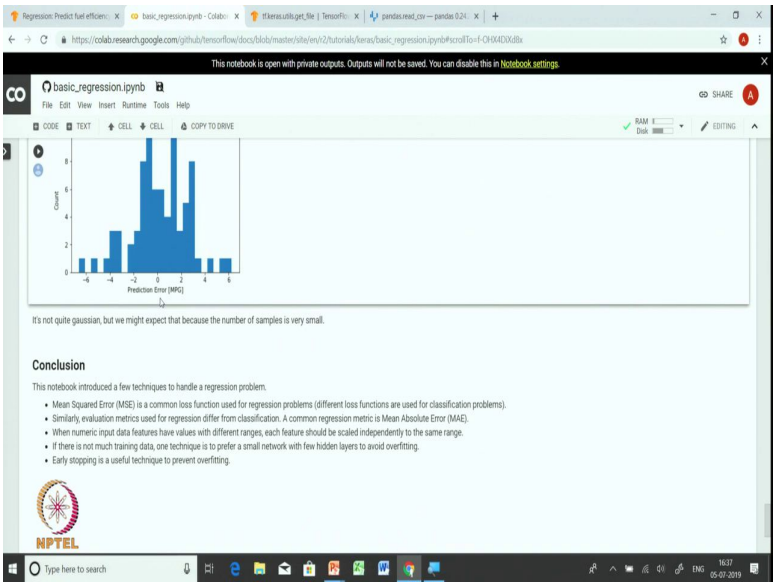
(Refer Slide Time: 30:43)



Let us also plot the error distribution the mean absolute error distribution; mean absolute error distribution is not really Gaussian. But, we expect we might expect that because here we

have a very small number of samples. We just had 314 rows out of which 80 percent where use for training and only 20 percent for use for test.

(Refer Slide Time: 31:05)



So, this brings us to the end of the regression exercise. In this exercise we learned how to use deep neural network model for the task of regression; we also studied how to how to remedy or how to prevent the overfitting using early stopping as regularization mechanism. In the coming session we will look at how to use this deep neural network model to make predictions on the structured data. We will also study how to store and retrieve the model for the deployment purpose and also study underfitting and overfitting through some practical examples.