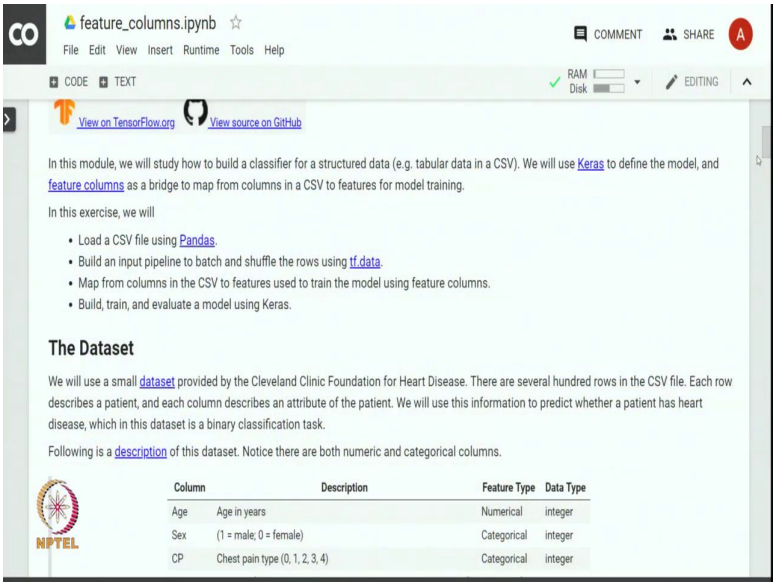
Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 16
Classify Structured Data

Since the last few modules, we have used TensorFlow API for building models, for image classification and regression problems. One of the important forms of data that we frequently encounter in practice is structured data, the data that is stored in tabular format in CSV file. In this session we will study how to build models for structured data using TensorFlow API.

(Refer Slide Time: 00:49)

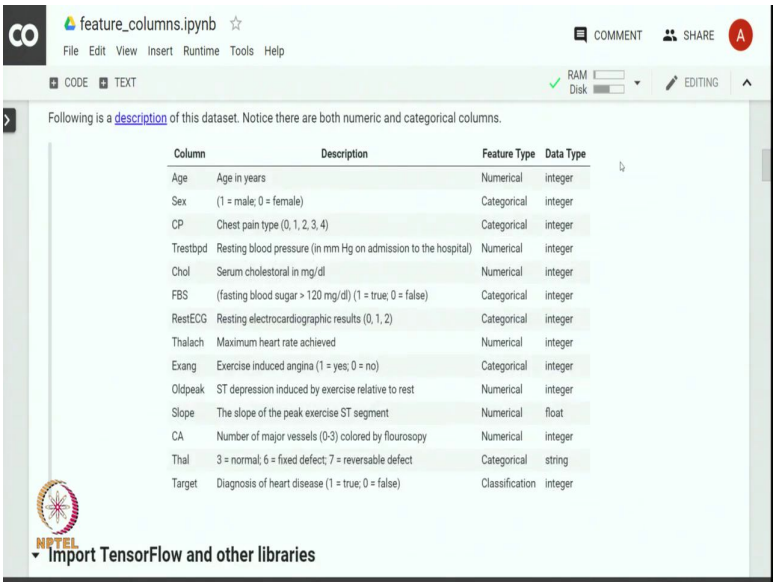


We will use Keras to define the model and features columns as a bridge to map from columns in CSV to features for model training. In this exercise we will load CSV using pandas which is a library for handling structured data in python. We will build an input pipeline using tf.data library, then we will map from columns in CSV to features used to train model using feature columns and finally, we will build train and evaluate a model with tf.keras library.

Let us look at the dataset we are going to use for this exercise. It is a data set provided by the Cleveland Clinic Foundation for Heart Disease. The idea is to build a model to predict

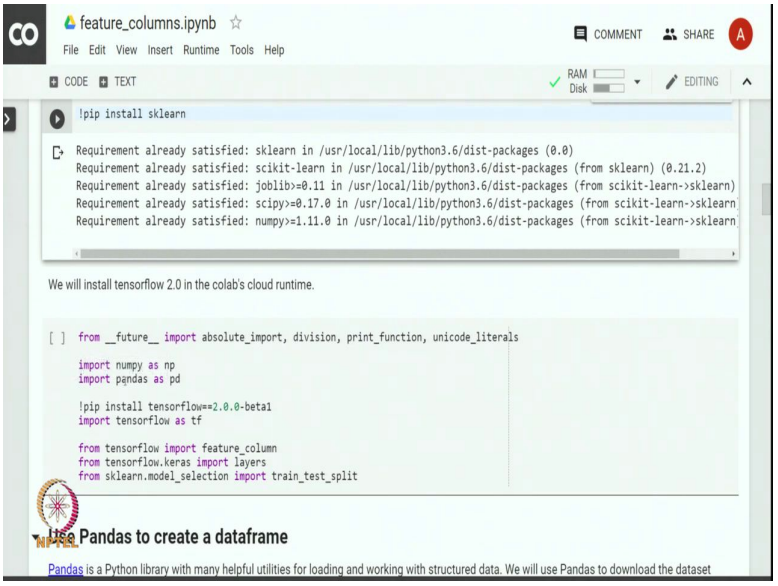
whether a patient will suffer from heart disease. There are several hundred rows in the CSV file. Each row describes a patient and each column describes an attribute of the patient.

(Refer Slide Time: 01:59)



You can see that the data set has a bunch of columns. There are a mix of columns in terms of the data types, lots of the columns are numerical columns, and a few are categorical columns. The final variable over here which is the target; the target variable is the classification label. We use label 1 if patient has a heart disease otherwise we use label 0.

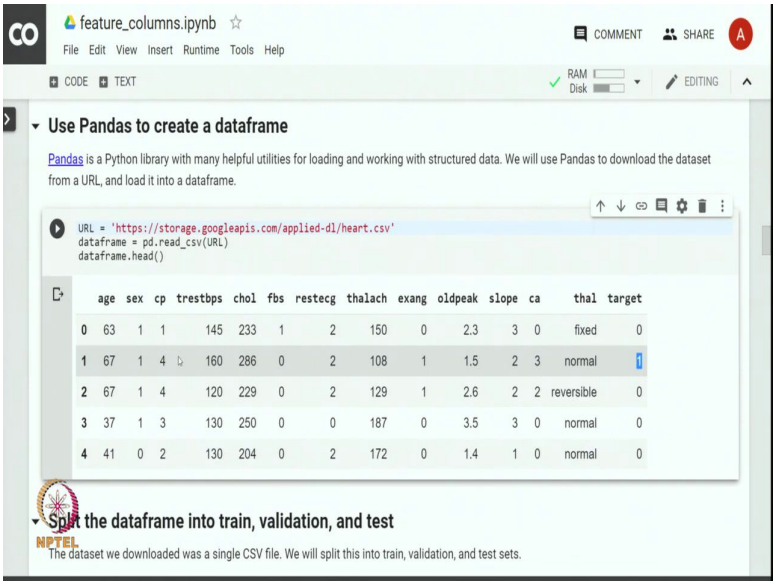
(Refer Slide Time: 02:29)



So, we will first install sklearn package which we will use for splitting the data into training and test set. Next, we will install the TensorFlow package and import TensorFlow and other associated libraries like feature columns, layers. From sklearn you will use train_test_split function.

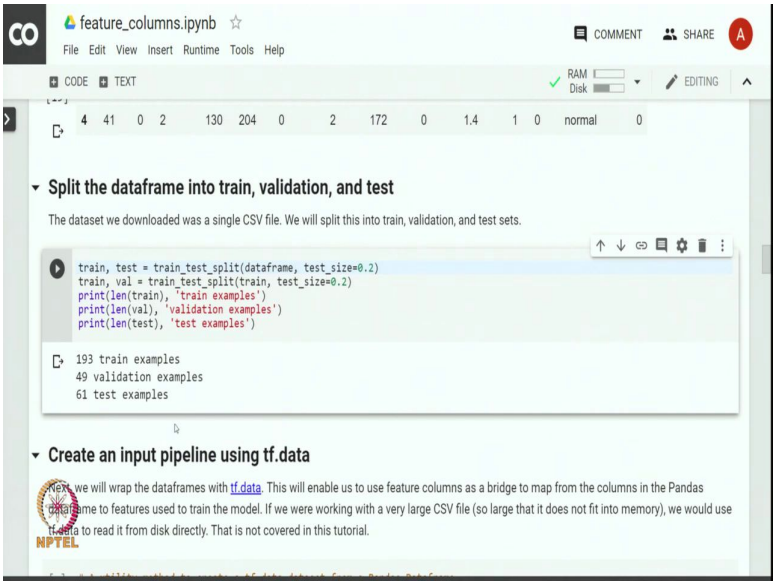
We will use numpy and pandas as libraries for manipulating the data. Now, we will download the data set and read it with pandas.

(Refer Slide Time: 03:17)



Now, what we will do is we will read the CSV file directly from the url using pandas.read_csv function and we will print top five rows in the data frame. You can see that that the top five rows are on the screen and the second patient has a heart disease. And, you can see attributes like age, sex, cholesterol levels and other associated attributes in the table.

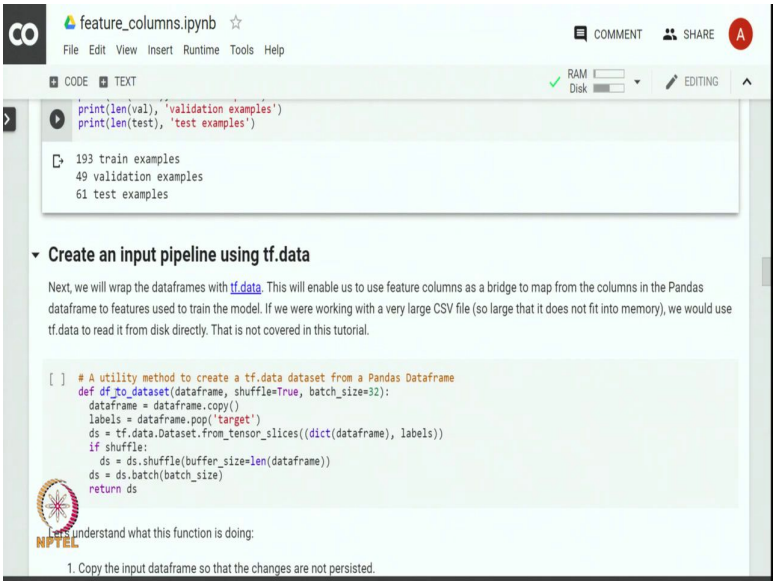
(Refer Slide Time: 04:09)



Next, now that we have loaded the data set in memory we will split that into train validation and test. So, we will use train_test_split function from sklearn. Let us split the data and print

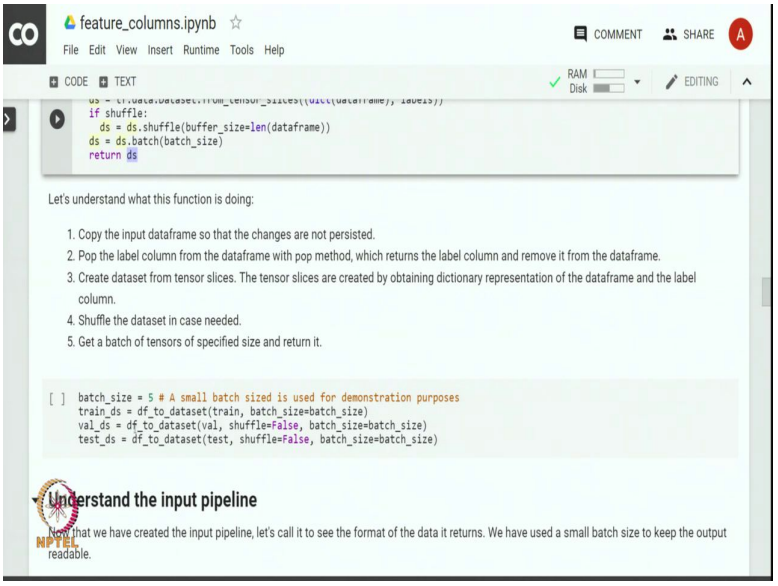
the statistics about training validation and test examples. So, we have 193 training examples, 49 validation examples and 61 test examples. Test data will never be exposed to the model during the training. So, model building will happen on 193 training samples. We will validate the model on 49 samples and we will finally, test its performance on the 61 examples.

(Refer Slide Time: 04:57)



We have loaded our data set in the pandas dataframe and we will wrap the dataframe with tf.data. This will enable us to use feature columns as a bridge to map from the columns in the dataframe to features in the model. If you are working with a very large CSV file we could also use tf.data to read it from the disk directly. Let us look at df_to_dataset method that we are using for wrapping the dataframes with tf.data.

(Refer Slide Time: 05:41)

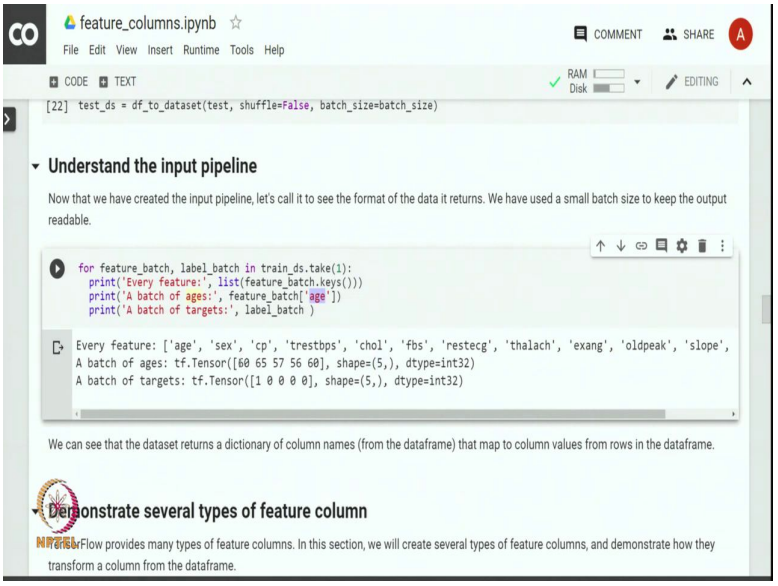


Here, we are first copying the dataframe so that any of the changes are not persisted. Next, we remove the target column or the column which contains the classification label and we remove it and store that in the labels array.

We create dataset from tensor slices. The tensor slices are created by obtaining the dictionary representation of the data frame and a labels column. Then, we shuffle data set if required we are passing flag for shuffling the data set if the flag is true then we shuffle the data set. And finally, we batch the tensors of specified size specified by batch_size variable and we return the batch of tensors which we will later be consumed during model training.

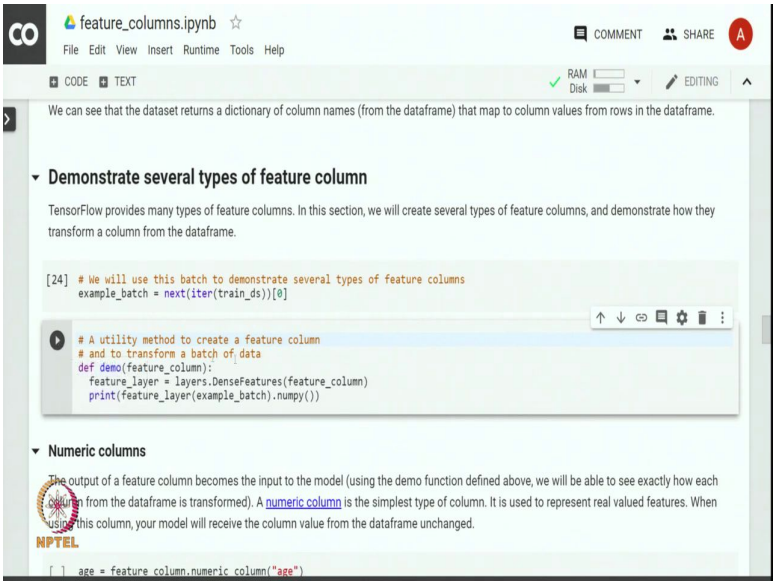
Now, what we will do is we will try to demonstrate we will we will try to convert we will try to convert the data frame to dataset. So, this is a small code where we will use a batch size of pipe and we convert training, validation and test set into the data set.

(Refer Slide Time: 07:09)



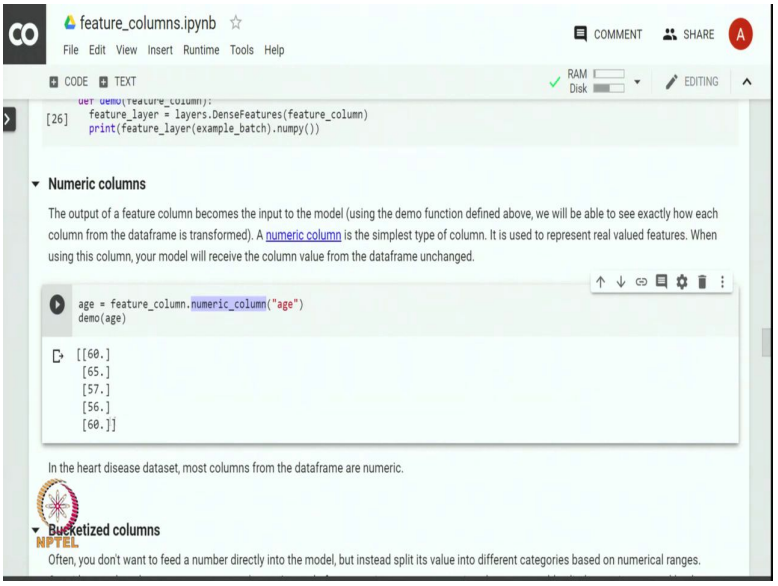
Now, that we have created the input pipeline let us call it to see the format of data it returns. We used a small batch size of 5 to keep the output readable. You can see that we have a list of features printed from this particular statement. Then, we can see the five values from the column age or from the feature age and we also have five values from the target and you can look at the shape and the data type of these tensors. So, both of these tensors are vectors. They contain exactly five elements. The data that is contained in them is essentially a 32 bit integer.

(Refer Slide Time: 08:09)



So, TensorFlow provides many types of feature columns. In this section what we will do is we will create several types of feature columns and demonstrate how to transform a column from the data frame. We create a utility method to create a feature column and to transform a batch of data.

(Refer Slide Time: 08:31)



Let us look at the first type of feature column for numeric attributes. A numeric column is the simplest type of column. It is used to represent real valued features. When we use this column

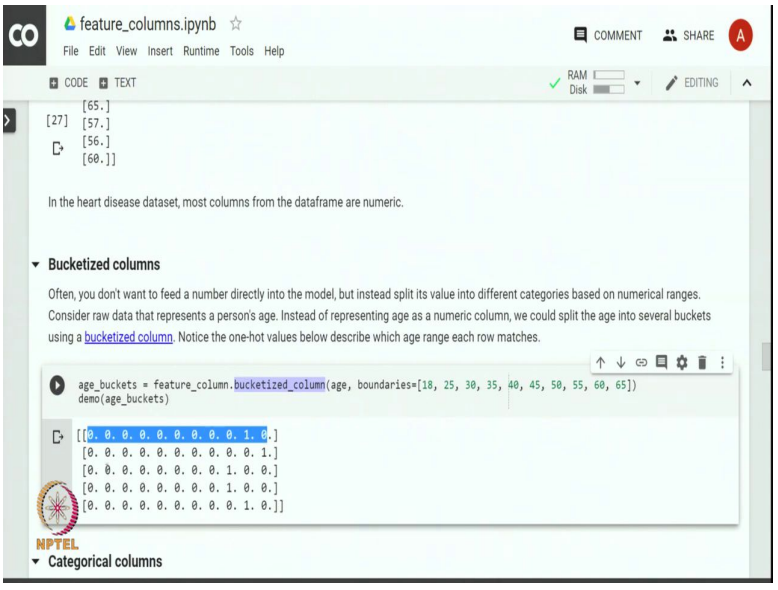
our model we will receive the column value from the dataframe unchanged. So, numeric values are passed as it is to the model.

So, we use feature_column.numeric_column to convert the numeric columns into the features into the feature_column. So, age is a numeric attribute. So, we use feature_column.numeric_column for transforming age. Let us look at what it returns.

We should see that this particular utility method returns a vector containing five elements. So, we have the first five values printed here because we used small batch size of 5.

We also have a large number of columns that are categorical and we cannot really feed a nonnumeric data to the TensorFlow. Hence we need to convert the non-numeric data into numbers. Let us look at some of the ways in which we can use feature_column to convert non-numeric data into feature columns.

(Refer Slide Time: 10:23)

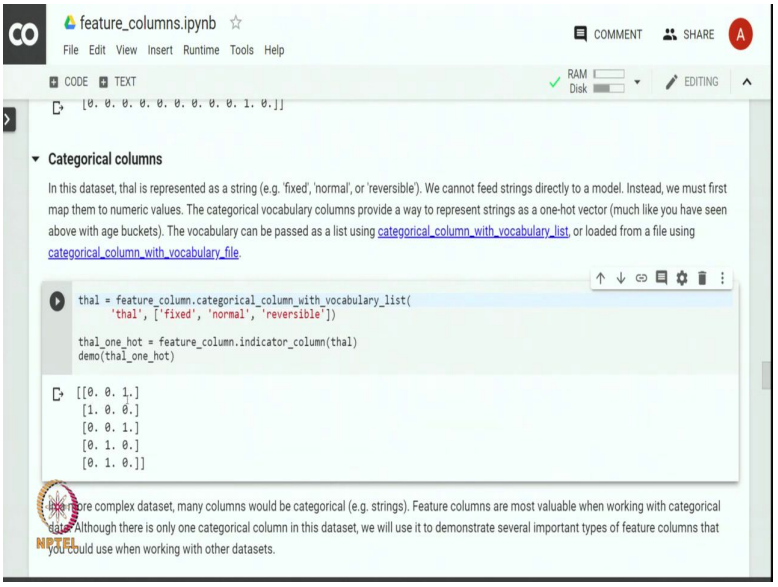


The first option that we have is called bucketized_column. Here, what happens is instead of using a number directly in the model, we split its values into different categories based on numerical ranges. Instead of representing age as a numeric column, we could split the age into several buckets using bucketized_column.

Here what happens is bucketized_column represents age as one-hot values based on the range matching based on the range match. So, here in this case there will be 11 ranges created the first range is for all ages below 18, then second for between 18 to 25, then 25 to 30 and so on until 60 to 65 and more than 65. Let us look at how the first five numeric ages are converted into bucketized_column.

So, you can see that the age is 60. So, the tenth value has got one then look at 65; 65 the eleventh value is one you can see that there are eleven values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. So, we use eleven length vector for representing the ranges because there are eleven ranges based on the boundaries that we have selected and you can see how the other values are also represented here.

(Refer Slide Time: 12:17)



In this dataset, there are also several categorical features that are represented by strings. Again, we cannot feed strings directly into the model. Instead we must first map it to a numeric value we have already seen one-hot encoding as one possible way to encode string values. In this case we will use categorical vocabulary column as a way to represent strings as a one hot vector.

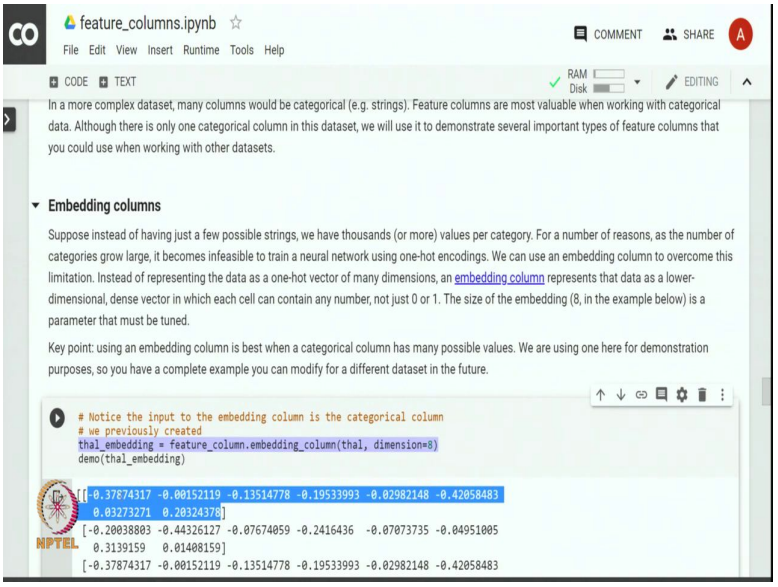
The vocabulary here can be passed either using a vocabulary list or vocabulary can also be loaded from a file. Depending on whether you are going to pass on the vocabulary list or file

there are two methods; one is categorical_column_with_vocabulary_list and then a similar one for vocabulary_file. So, let us look at converting a categorical column with vocabulary list.

Here thal is an attribute which has got three possible values; fixed, normal and reversible. We use feature_column.categorical_column_with_vocabulary_list to convert thal attribute into one hot encoding feature. So, we first use the categorical column with vocabulary list and then pass on that feature column to indicator_column to get a one hot encoded representations.

Let us look at one hot encoding of thal column. In each of the rows exactly one particular position is 1, all of the other positions are 0. This is exactly how the one hot encoding works as you may recall from the previous classes.

(Refer Slide Time: 14:25)



So, here we have most of the categorical attributes that do not have a lot of different values, but you will often encounter data sets in real life that would have a large number of strings in inner column and that could happen for multiple columns.

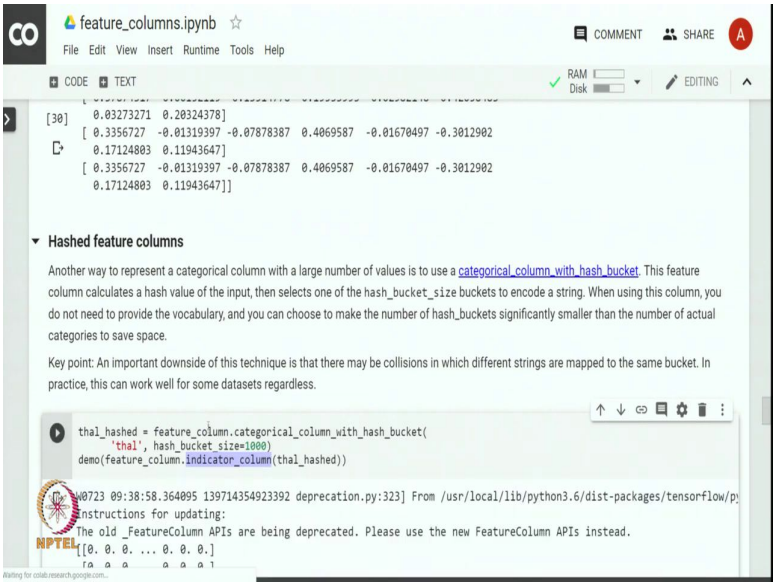
TensorFlow also provides a mechanism to encode columns with large number of possible string values. So, there are few mechanisms like embedding columns or hash columns that helps us to convert a column with a large number of strings into numbers. So, what happens

if you have thousands of values per category. So, we do not want to really use one-hot encoding here because that representation will be extremely sparse representation.

So, instead of using one hot encoding here we use what is called as an embedding column. Embedding column represents the data as a lower dimensional dense vector in which each cell can contain any number between any number and not just 0 and 1. The size of the embedding is a parameter that must be tuned. In this particular example here we are creating an embedding for thal column and we are using number of dimensions as 8. So, let us try to see how it works.

In this particular case we can see that it is a dense representation we are we have converted thal into 8 values and each value can take any number; number can be either positive or negative.

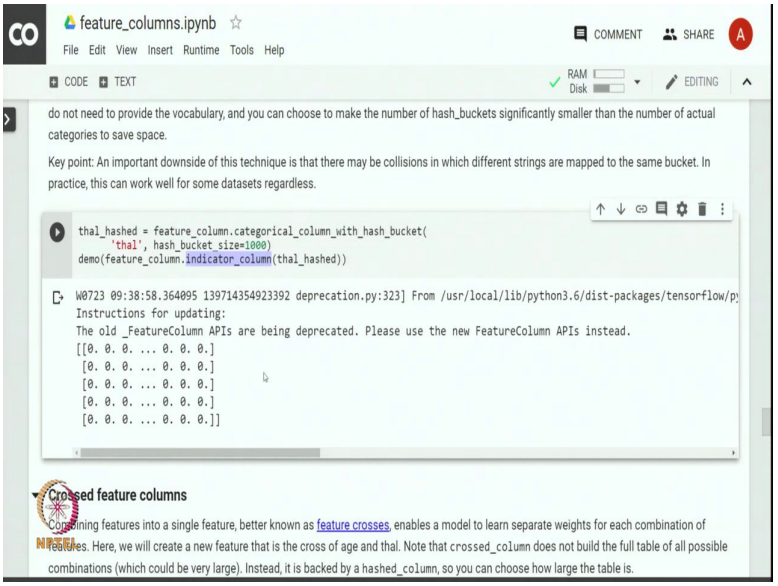
(Refer Slide Time: 16:29)



Apart from embedding column we also have another way to represent a categorical column, it is called hash column or hashed feature column. Here the central idea is to use hashing. This feature column calculates a hash value of the input and then select one of the buckets to encode a string. When using this column we do not need to specify a vocabulary and we can choose to make the number of buckets significantly smaller than the number of actual categories to save the space.

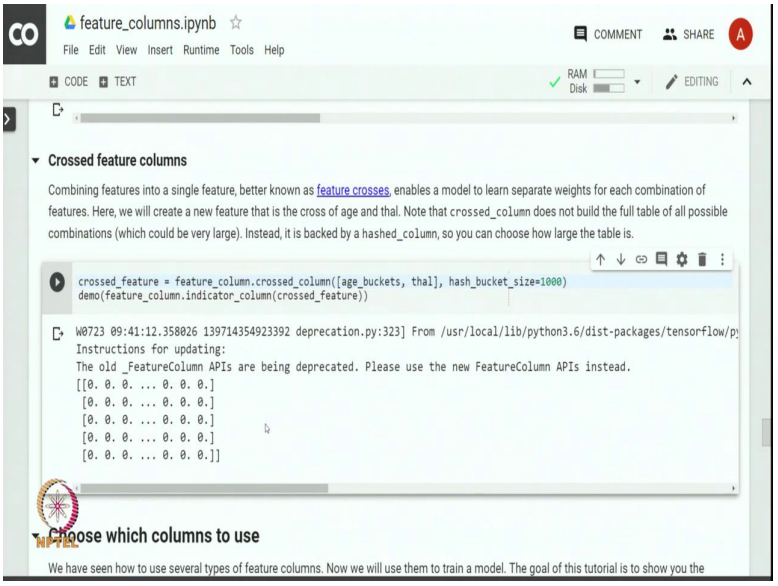
One of the important things to note while using this particular technique is that it has a downside that there may be collisions in which different strings would get map to the same bucket. However, in practice this scheme works quite well for some datasets. So, let us convert thal into numbers using the method of hash buckets. So, we use categorical_column_with_hash_bucket to convert thal into hash buckets of size 1000. Then convert the hashed representation into an indicator column representation.

(Refer Slide Time: 18:03)



So, we will again get a one hot encoding kind of representation after we convert the feature column into an indicator column and you can see that now we have one hot encoding in with one hot encoding done with the vector which has got 1000 entries. Only one of them we will be be 1 based on based on the bucket ID in which the value was hashed. In the past, we also saw that in order to construct complex dataset we often need to cross columns.

(Refer Slide Time: 18:37)



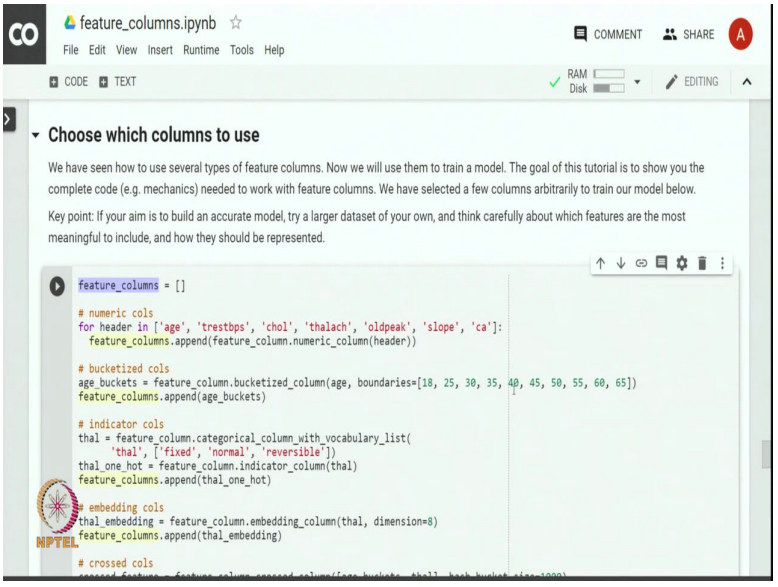
So, combining features commonly known as feature crossing, is a popular way to build complex decision boundaries. So, after crossing the features we create a new feature that is cross of two original features. In this case we cross two columns one is the age bucket that we created based on bucketization and thal value.

The cross column does not build the full table of all possible combinations, just because it could be very large and take a lot of memory. Instead of that we use hash column for hashing the values coming out of the crossed column. So, we can see here we have to simply use crossed_column to cross two columns and we have to specify the hash bucket size. After crossing the values in the columns hashing is automatically carried out based on the bucket size that we specify here.

Then we can convert the hash representation of the cross feature to an indicator representation using indicator_column command. You may recall that indicator_column command is used to create one hot encoding representation for a feature. We have studied a few methods to convert the non-numeric features into numbers, specifically we looked at methods like one-hot encoding using list or one-hot encoding using values mentioned in files or used in hashing technique or embedding techniques to convert strings into numbers.

For numerical attributes, we looked at the numerical columns or we also use bucketized column to bucketize the number into various buckets and then get representation of that particular column in a bucketized format. And, we also looked at how to construct feature crosses and represent the crossed features using crossed_column.

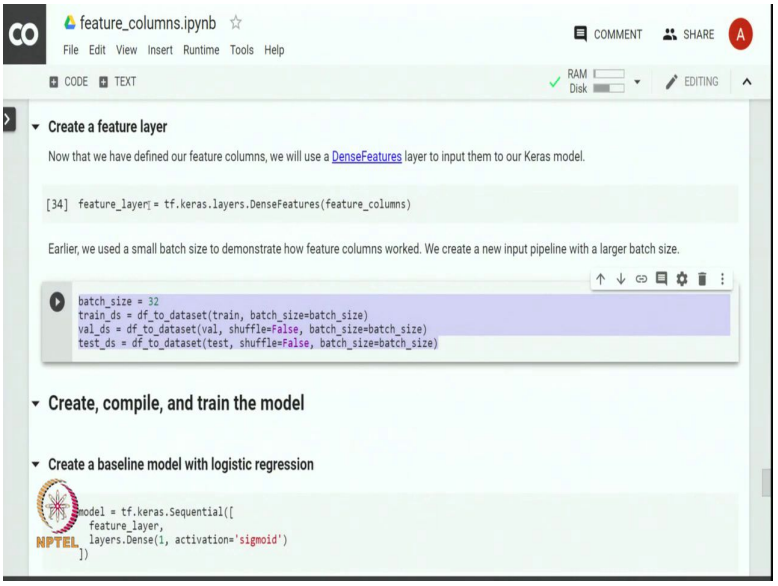
(Refer Slide Time: 21:01)



Next, we will choose the columns to use for training a model. Here we select a few columns arbitrarily to train our model. If your aim is to build an accurate model you should take a larger data set and think carefully about what features are more meaningful for your model and then include only those features or construct meaningful features from the from the given representation.

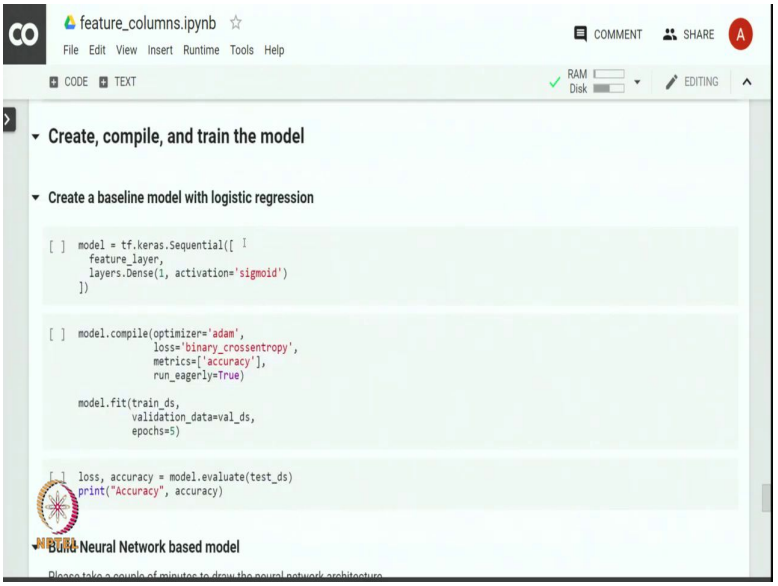
We will define feature_columns as a list to hold the features that we are going to use. So, here what we do is for all the numeric columns we use numeric underscore column function and construct numeric feature columns, then we construct bucketized feature columns for age based on the boundaries given over here. Then we can then we construct indicator feature columns for thal and we also construct an embedding feature column for thal and we cross the age buckets with thal and construct call and construct crossed column feature columns.

(Refer Slide Time: 22:27)



Now, that we have defined our feature columns we will use a dense features layer to input them to our keras model. The dense features layer takes feature columns as an input.

(Refer Slide Time: 22:39)

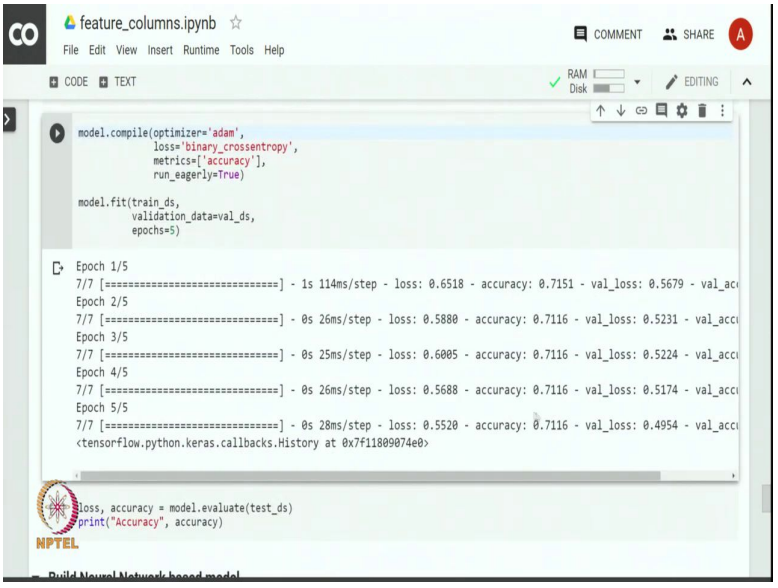


Let us create a model based on the feature columns defined earlier. First we will create a baseline model with logistic regression. The logistic regression model is constructed with

tf.keras.Sequential in which we specify the feature layer and then there is an output unit which is which has got one unit with a sigmoid activation.

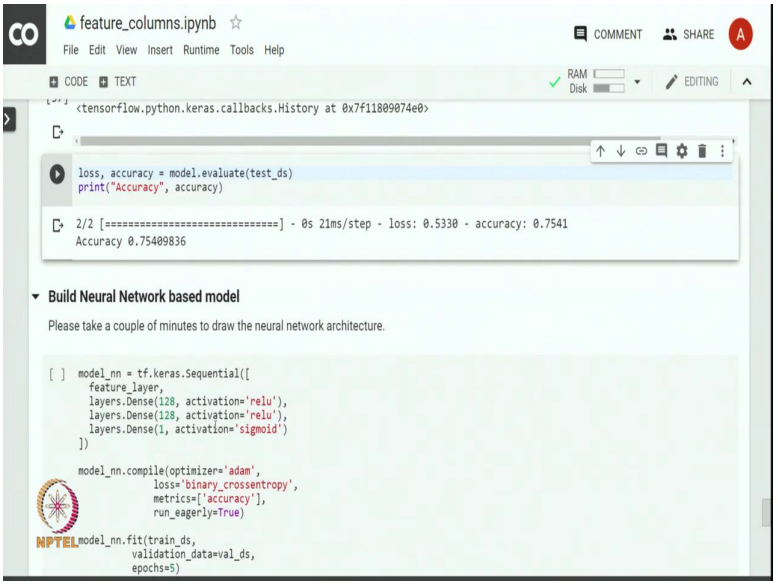
Once you define a logistic regression model, we compile it and we compile it with adam optimizer, we use binary cross entropy as a loss because we are solving a binary classification problem and we will use accuracy as a metric to track during the training. Finally, we will find out the loss and accuracy of the model based on the test data. Let us run this particular code.

(Refer Slide Time: 23:35)



So, let us construct the model then we will compile it and you can see that the model is getting trained and after 5 epochs the model has an accuracy of 71 percent on training set and validation accuracy is slightly higher, it is 77 percent.

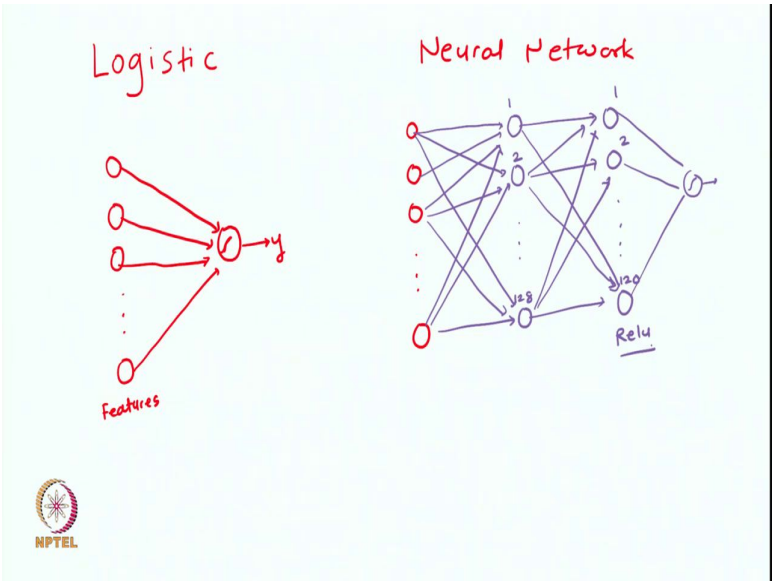
(Refer Slide Time: 23:59)



Let us look at the accuracy on the test set. On test set we got accuracy of 75 percent. So, this is our baseline model let us try to build a neural network model. It is always a good idea to build a baseline model for a classification problems logistic regression serves as a good baseline model. If you are solving a regression model always start with a linear regression model as a baseline model.

Baseline model also helps us to understand what kind of performance we can obtain just using the data that is given to us and then we can and then we can use a bunch of strategies to improve the performance and doing the base lining also helps us to understand how each of these new strategies help you to improve the performance of the model further. Let us build a neural network model. In this particular case let us look at the structure of this neural network model.

(Refer Slide Time: 25:03)



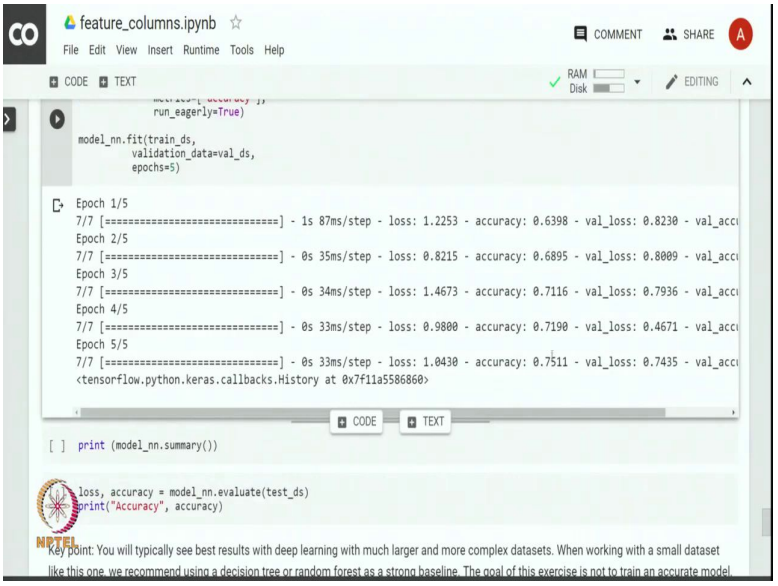
So, we had a logistic regression model as a baseline model and we have a neural network model here. In case of logistic regression model what we did is we had a bunch of feature columns and we had exactly one output. These are all features and we had sigmoid as an activation function here. Your sigmoid as an activation function here which gives us a probability of a patient having a heart disease.

In case of neural network what we do is we take these features and we define we set up a neural network with two hidden layers each containing 128 units and each of this input. And finally, we have a single output node. So, you got sigmoid activation for the output later and use relu as an activation for the hidden layers. So, this is the neural network architecture.

We are using a feed forward neural network with two hidden layers each containing 128 units and relu has an activation and we have an output layer with a sigmoid activation as we can see over here. We use adam as an optimizer, we use binary cross entropy as a loss because we are solving a binary classification problem and we will track accuracy as a metric.

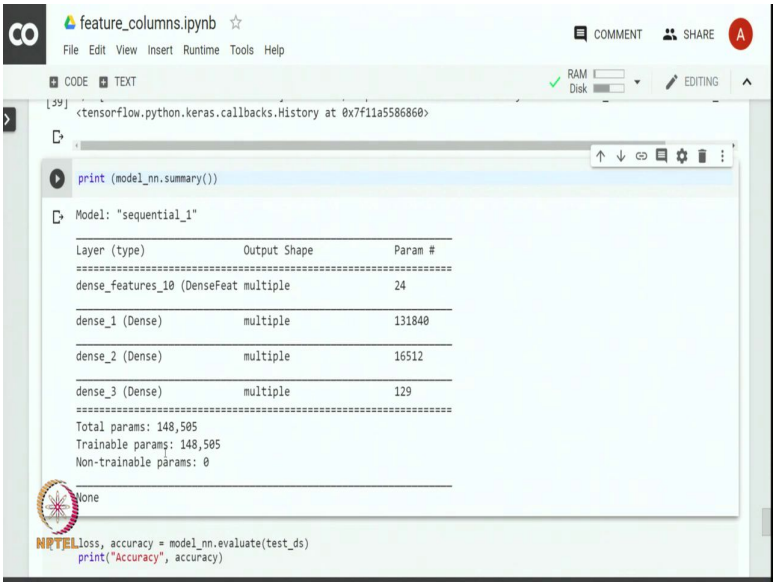
After compiling the model we will we will run a training loop with train_ds as a training data and val_ds as a validation data and we train for five epochs.

(Refer Slide Time: 27:41)



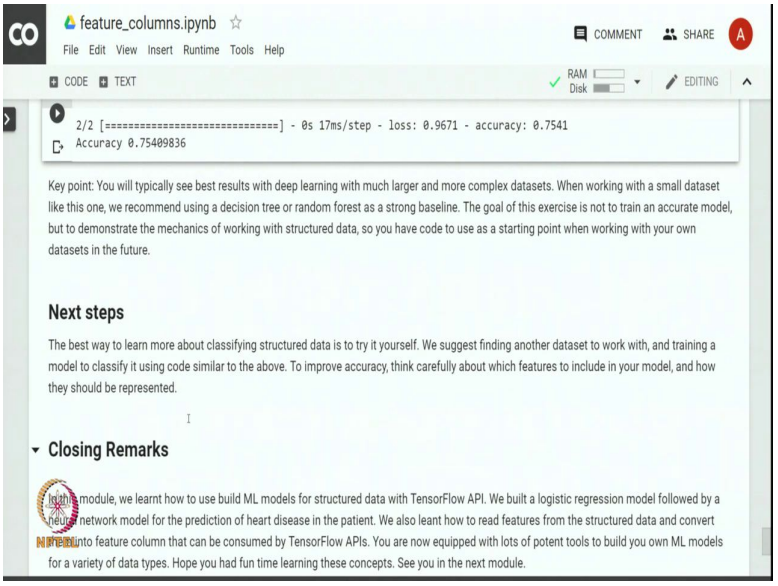
You can see that we get accuracy of about 75 percent at the end and validation accuracy is still slightly higher and let us look at the model summary.

(Refer Slide Time: 27:55)



Model summery helps us to see what kind of model we have setup and we can also see the number of parameters of the model. We can see that the total number of parameters in the model is very large. We have about 148k parameters.

(Refer Slide Time: 28:11)



And, let us evaluate the accuracy of model with the test data we get an accuracy of 75, 0.75 or 75 percent accuracy of the test data. You will typically see best results with deep learning with much larger and more complex data sets. When working with a small dataset we recommend using other classifiers like decision tree or random forest as the strong baseline. The goal of this exercise was to demonstrate the mechanics of working with structured data, so that you have some idea of how to work with structured data when you start working on your own.

The best way to learn more about classifying structured data is to try it yourself with some datasets. I would strongly encourage you to find a structured data and apply the concepts that we studied in this particular session. To improve the accuracy you should think carefully about which features to include in the model and how they should be represented.

In this module, we learnt how to use machine learning models for structured data with TensorFlow API. We build a logistic regression model followed by neural network model for prediction of heart disease in a patient. We also learnt how to read features from the structured data and convert them into feature columns. You are now equipped with lots of potent tools to build your own machine learning models for a variety of data types. Hope you had a fun time learning these concepts.