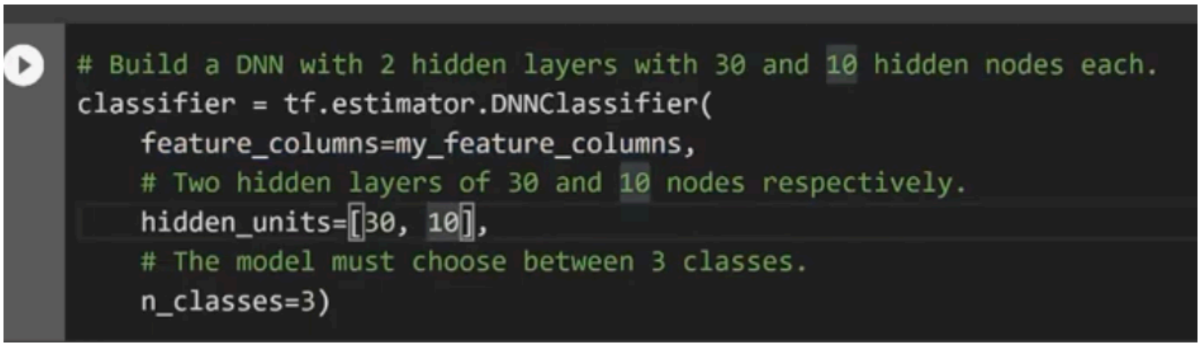
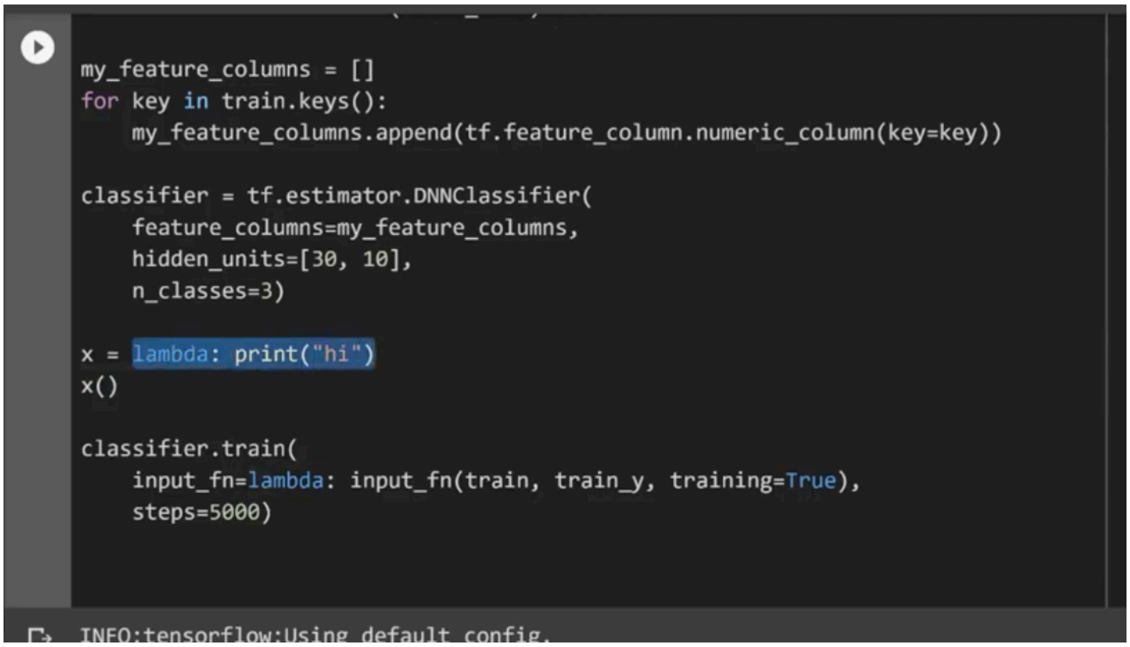
- ***Fitting the classification model***
    - -> this is a linear classifier
    - -> in tensor flow
        - a DNNClassifier
            - -> deep neural network classifier
        - a linear classifier
            - -> this does classification rather than regressoin
            - -> it divides things into categories, rather than predicts the values using a regression analysis
    - -> this model is using a DNN with two layers to classify the species of plant
- ***To make the model***



```
# Build a DNN with 2 hidden layers with 30 and 10 hidden nodes each.
classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    # Two hidden layers of 30 and 10 nodes respectively.
    hidden_units=[30, 10],
    # The model must choose between 3 classes.
    n_classes=3)
```

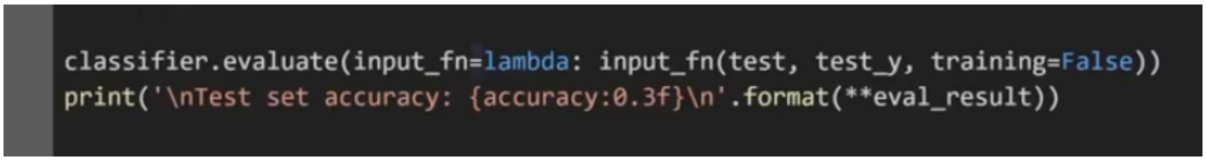
    - -> build a deep neutral network with two hidden nodes each
    - -> .estimator <- this module stores pre-made models
    - -> then defining the hidden units -> to set the hidden architecture of the model
    - -> then the number of classes <- in other words the number of categories the model splits the data into
- ***To train the model***
    - -> this is a DNN which is a more complex model than the previous example
    - -> there is an input function
    - -> this function was returned from another function
    - -> this is being used to train the model
        - -> a lambda is an anonymous function
        - -> it's a one line function in Python
        - 

```
my_feature_columns = []
for key in train.keys():
    my_feature_columns.append(tf.feature_column.numeric_column(key=key))

classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    hidden_units=[30, 10],
    n_classes=3)

x = lambda: print("hi")
x()

classifier.train(
    input_fn=lambda: input_fn(train, train_y, training=True),
    steps=5000)
```

```
INFO:tensorflow:Using default config.
```

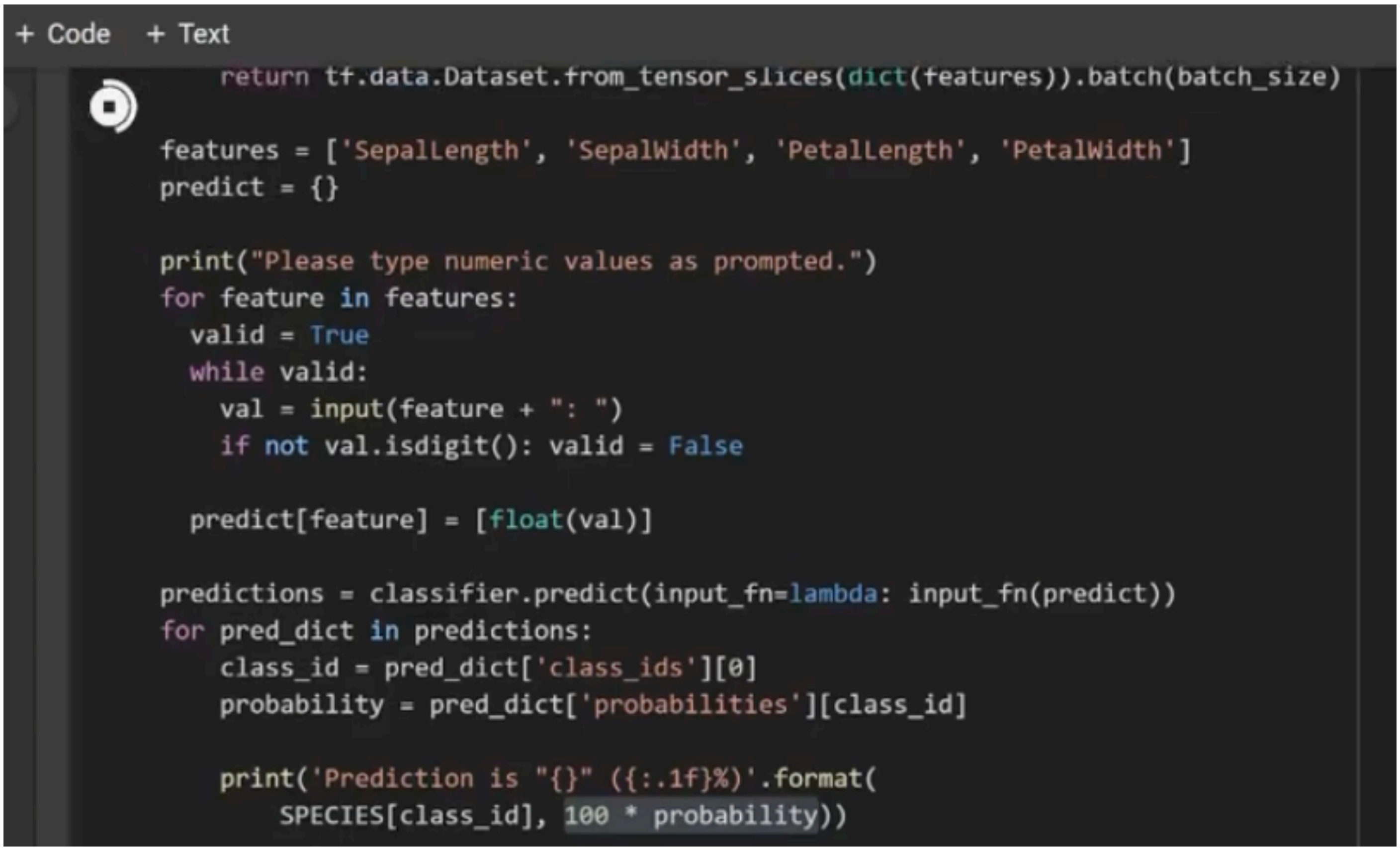
        - -> (highlighted) the lambda function -> this works like a function which is embdedded inside another function, and it defined in one line
        - -> this is done because the function wasn't embedded in its definition -> these would have been created in the form of an interior function

        - -> iterating through the dataset x number of times
            - he's doing this in a loop and iterating through the dataset
            - -> then printing out the value which comes out after each iteration
            - -> this is training the model
- ***To evaluate the model***
    - -> to see how the accurate trained model is
    - -> he passes the input function into the model for this data
    - -> classifier.evaluate
    - -> a lambda is being passed into this function <- the lambda is a dummy function



```
classifier.evaluate(input_fn=lambda: input_fn(test, test_y, training=False))
print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
```

        - -> then the model trains
        - -> he's stored the first line of code in the picture above in a variable
        - -> this gives the accuracy of the model which in this case is 80%
- ***To use the model to make predictions***
    - -> the script allows us to make predictions on blocks
    - -> he's running the prediction on one data entry
    - -> he's batching the data -> these are the data for predictions, so he's removed the results in the data set which we're predicting
    - -> the feature is predicted, then it's set equal to a list and added to a dictionary
        - -> tensor flow predicts for multiple values in a list
    - -> then for prediction dictionaries -> they add ids to the classes
    - -> then he iterates through the dictionaries with the predictions and calculates their accuracy



```
    return tf.data.Dataset.from_tensor_slices(dict(features)).batch(batch_size)

features = ['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']
predict = {}

print("Please type numeric values as prompted.")
for feature in features:
  valid = True
  while valid:
    val = input(feature + ": ")
    if not val.isdigit(): valid = False

  predict[feature] = [float(val)]

predictions = classifier.predict(input_fn=lambda: input_fn(predict))
for pred_dict in predictions:
    class_id = pred_dict['class_ids'][0]
    probability = pred_dict['probabilities'][class_id]

    print('Prediction is "{}" ({:.1f}%)'.format(
        SPECIES[class_id], 100 * probability))
```

    - -> the then runs the code and it takes input values for a flower -> to predict which type it is
        - -> there were three probabilities <- one for each of the classes
        - -> each of those predictions was a percentages
        - -> each of the plant species had a probability (a probability it was this species of flower, this one, this one -> and then it returns the most likely)

        - -> this example makes the predictions for one of the species of plant
    - -> he knows wha the outputs are for each of those plant types and has the prediction compared against what the actual value was <- this is called evaluating the model
- **if you are using tensor flow for classification problems (e.g which species does the plant belong to) -> then use a DNN (deep neural network) classifier**