Evaluation Mode

Chapter Goals:

- Set up the regression function's evaluation code

A. Evaluating the model

When evaluating the model, we use mean absolute error as the metric. This is because our goal is to get the model's sales predictions as close to the actual labels as possible, which is equivalent to minimizing the mean absolute error between predictions and labels.

Since we use the same evaluation metric as the loss function, this makes the evaluation code extremely easy. We just need to return an `EstimatorSpec` containing the model's loss on the evalution set.

```
mode = tf.estimator.ModeKeys.TRAIN
estimator_spec = tf.estimator.EstimatorSpec(
    mode, loss=loss, train_op=train_op)
```

Creating the EstimatorSpec for training. The train_op variable is used for minimizing the loss

Time to Code!

All code for this chapter goes in the `regression_fn` function.

The code for this chapter focuses on model evaluation. Since we evaluate the model using mean absolute error, which is the same as the loss function, we don't need to do anything other than return the `EstimatorSpec`.

Outside the `if` block from the previous chapter, create another `if` block. This one should check if `mode` is equal to `tf.estimator.ModeKeys.EVAL`.

Inside the `if` block, return `tf.estimator.EstimatorSpec` initialized with `mode` as the required argument and `loss` as the `loss` keyword argument.

```
class SalesModel(object):
  def __init__(self, hidden_layers):

    self.hidden_layers = hidden_layers

  def regression_fn(self, features, labels, mode, params):
    feature_columns = create_feature_columns()
    inputs = tf.feature_column.input_layer(features, feature_columns)
    batch_predictions = self.model_layers(inputs)
    predictions = tf.squeeze(batch_predictions)
    if labels is not None:
      loss = tf.losses.absolute_difference(labels, predictions)

    if mode == tf.estimator.ModeKeys.TRAIN:
      global_step = tf.train.get_or_create_global_step()
      adam = tf.train.AdamOptimizer()
      train_op = adam.minimize(
        loss, global_step=global_step)
      return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
    # CODE HERE

  def model_layers(self, inputs):
    layer = inputs
    for num_nodes in self.hidden_layers:
      layer = tf.layers.dense(layer, num_nodes,
        activation=tf.nn.relu)
    batch_predictions = tf.layers.dense(layer, 1)
    return batch_predictions
```