Evaluation and Quality Assurance

- The RNN model has been created and tested using different number of epochs in order to validate and calculate the Loss behavior of our model on our sample training batch dataset.

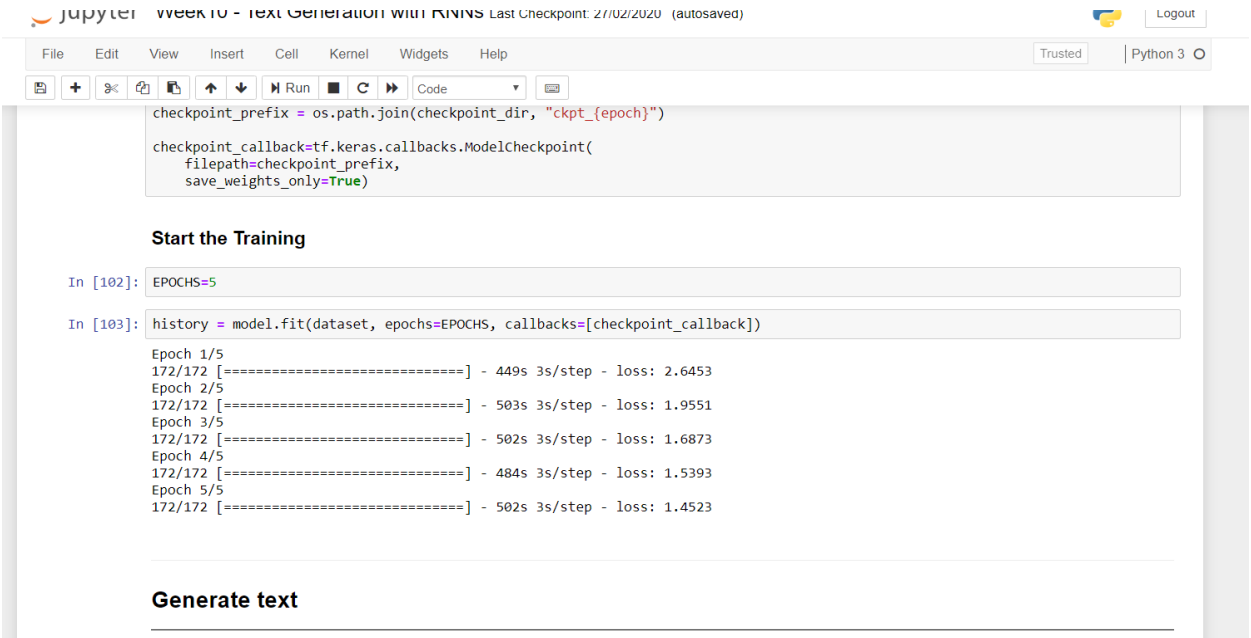
Initial Mean loss value of our dataset batch has been evaluated at 4.17577 Which is a very high value and this loss needs to be reduced minimally by setting an appropriate Epochs.

Train the model

```
[99]: def loss(labels, logits):
          return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

      example_batch_loss  = loss(target_example_batch, example_batch_predictions)
      print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, sequence_length, vocab_size)")
      print("scalar_loss:      ", example_batch_loss.numpy().mean())

      Prediction shape:  (64, 100, 65)  # (batch_size, sequence_length, vocab_size)
      scalar_loss:       4.17577

[100]: model.compile(optimizer='adam', loss=loss)
```

Two epochs numbers have been utilized and the loss values have been recorded for observation.

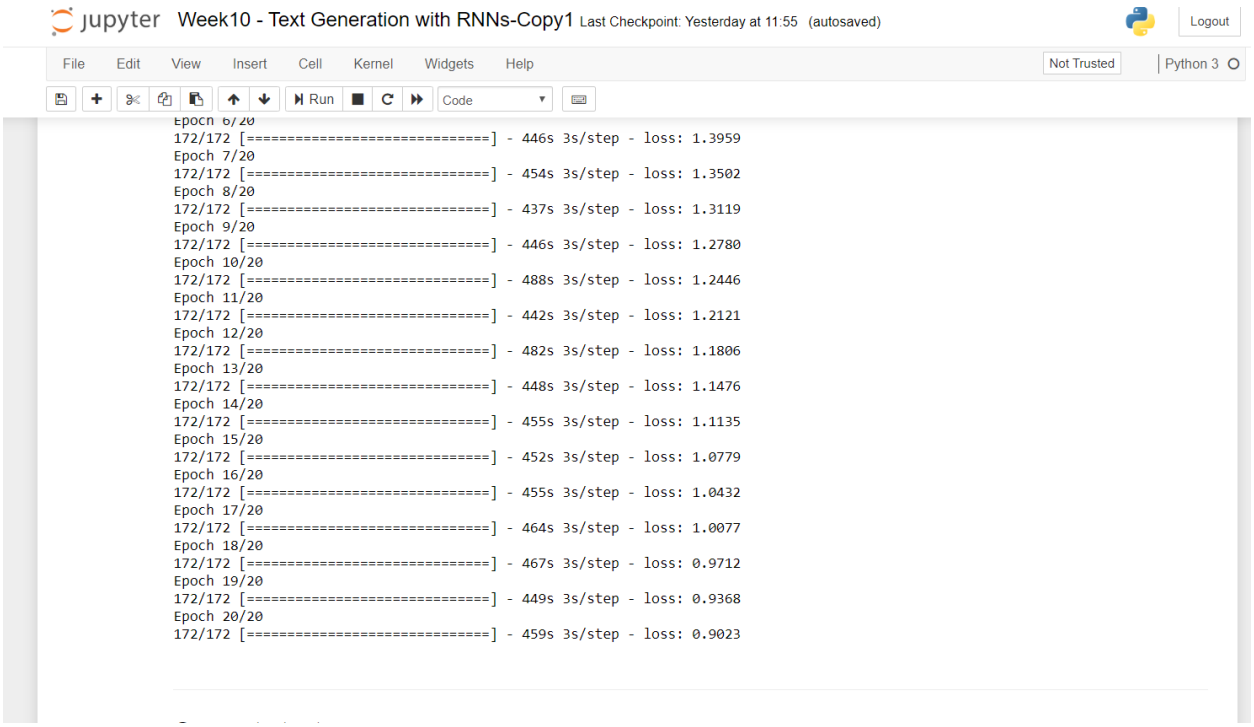
Working with “Epochs = 5”



After running 5 Epochs, the model was able to iterate really fast, but ended with a loss of 1.4523 which is on the high side for a loss value. At least we aim to have a loss value below “1”

After much research online I observed that most neural networks although usually have large datasets, require to be run on an epochs of at least 15 – 30, but not too high as this leads to vanishing gradient problem.

Therefore, I decided to work with “Epochs = 20”



From this iteration we can observe that the loss value of our RNN model got reduced over the 20 epochs and we ended at a value of 0.9023. This shows us that our model performs better when trained on larger epochs values than a smaller number.