Importing Libraries

```
[7] import tensorflow as tf
    from tensorflow.keras.layers import *
    from tensorflow.keras.models import *
    from tensorflow.keras.datasets import imdb
```

Loading and Preprocessing Data

```
[9] (train_x, train_y), (test_x, test_y) = imdb.load_data(num_words=1000)
    print("Review is: ", train_x[5])
    print("Label is: ", train_y[5])

    word_index = imdb.get_word_index()
    print(word_index)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
Review is:  [1, 778, 128, 74, 12, 630, 163, 15, 4, 2, 2, 2, 2, 32, 85, 156, 45, 40, 148, 139, 121, 664, 665, 10, 10, 2, 173, 4, 749, 2, 16, 2, 8, 4, 226,
Label is:  0
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 1s 1us/step
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders': 16115, 'hanging': 2345, 'woody': 2289, 'traw
```

```
[12] from keras.preprocessing import sequence
     max_words=500
     train_x=sequence.pad_sequences(train_x,maxlen=max_words)
     test_x=sequence.pad_sequences(test_x,maxlen=max_words)
     embedding_size=32
```

Creating Model

```
[14] model=Sequential()
     model.add(Embedding(1000,embedding_size,input_length=(max_words)))
     model.add(SimpleRNN(100,return_sequences='true'))
     model.add(SimpleRNN(50,return_sequences='true'))
     model.add(SimpleRNN(25))
     model.add(Dense(1,activation='sigmoid'))
     model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 500, 32)           32000

 simple_rnn_3 (SimpleRNN)    (None, 500, 100)          13300

 simple_rnn_4 (SimpleRNN)    (None, 500, 50)           7550

 simple_rnn_5 (SimpleRNN)    (None, 25)                1900

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 54776 (213.97 KB)
Trainable params: 54776 (213.97 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling and Training Model

```
[16] return_sequences=True
     model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
```

```
[17] his=model.fit(train_x,train_y,epochs=10,batch_size=128,validation_split=0.2)
```

```
Epoch 1/10
157/157 [==============================] - 180s 1s/step - loss: 0.7032 - acc: 0.4994 - val_loss: 0.6918 - val_acc: 0.5078
Epoch 2/10
157/157 [==============================] - 149s 951ms/step - loss: 0.6947 - acc: 0.5295 - val_loss: 0.6718 - val_acc: 0.5876
Epoch 3/10
157/157 [==============================] - 123s 784ms/step - loss: 0.5956 - acc: 0.6817 - val_loss: 0.7122 - val_acc: 0.6238
Epoch 4/10
157/157 [==============================] - 133s 847ms/step - loss: 0.5114 - acc: 0.7603 - val_loss: 0.6056 - val_acc: 0.6478
Epoch 5/10
157/157 [==============================] - 131s 838ms/step - loss: 0.5009 - acc: 0.7662 - val_loss: 0.5382 - val_acc: 0.7634
Epoch 6/10
157/157 [==============================] - 123s 786ms/step - loss: 0.4799 - acc: 0.7832 - val_loss: 0.4477 - val_acc: 0.8004
Epoch 7/10
157/157 [==============================] - 120s 765ms/step - loss: 0.5355 - acc: 0.7397 - val_loss: 0.5215 - val_acc: 0.7420
Epoch 8/10
157/157 [==============================] - 119s 761ms/step - loss: 0.4700 - acc: 0.7893 - val_loss: 0.6124 - val_acc: 0.7610
Epoch 9/10
157/157 [==============================] - 121s 773ms/step - loss: 0.5610 - acc: 0.7150 - val_loss: 0.6462 - val_acc: 0.5960
Epoch 10/10
157/157 [==============================] - 119s 758ms/step - loss: 0.5049 - acc: 0.7584 - val_loss: 0.4753 - val_acc: 0.7816
```

Visualization

```
[21] import matplotlib.pyplot as plt

     # Accessing training and validation metrics from history object
     acc = his.history['acc']
     val_acc = his.history['val_acc']
     loss = his.history['loss']
     val_loss = his.history['val_loss']

     epochs = range(1, len(acc) + 1)

     # Plotting Training and Validation Accuracy
     plt.plot(epochs, acc, 'b', label='Training Accuracy')
     plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
     plt.title('Training and Validation Accuracy')
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.legend()
```

```
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting Training and Validation Loss
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```