In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
```

In [3]:

```
# Load the IMDB dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

# Display dataset shape
print(f'Training data shape: {len(train_data)}')
print(f'Training labels shape: {len(train_labels)}')
print(f'Testing data shape: {len(test_data)}')
print(f'Testing labels shape: {len(test_labels)}')
```

```
Training data shape: 25000
Training labels shape: 25000
Testing data shape: 25000
Testing labels shape: 25000
```

In [4]:

```
# Pad sequences to a maximum length of 500
max_length = 500
train_data = pad_sequences(train_data, maxlen=max_length)
test_data = pad_sequences(test_data, maxlen=max_length)

# Check padded data shape
print(f'Padded training data shape: {train_data.shape}')
print(f'Padded testing data shape: {test_data.shape}')
```

```
Padded training data shape: (25000, 500)
Padded testing data shape: (25000, 500)
```

In [5]:

```
def create_rnn_model():
    model = Sequential()
    model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_length))
    model.add(SimpleRNN(64))  # Simple RNN layer
    model.add(Dense(1, activation='sigmoid'))  # Binary classification
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

# Create RNN model
rnn_model = create_rnn_model()
rnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 500, 128)          1280000   
                                                                 
 simple_rnn (SimpleRNN)      (None, 64)                12352     
                                                                 
 dense (Dense)               (None, 1)                 65        
                                                                 
=================================================================
Total params: 1,292,417
Trainable params: 1,292,417
Non-trainable params: 0
_________________________________________________________________
```

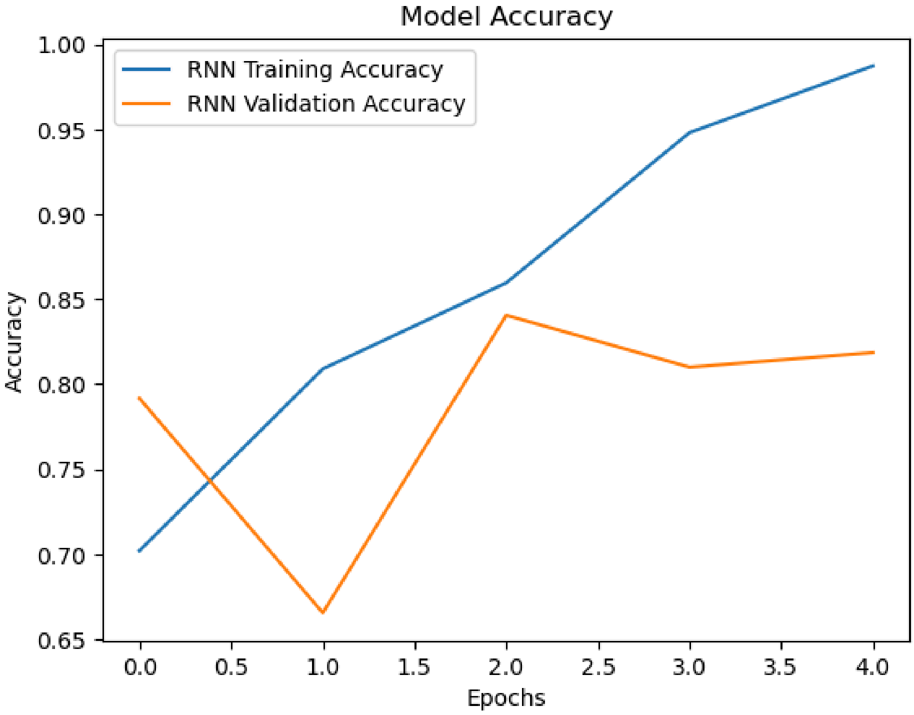
In [6]:
```
# Train RNN model
rnn_history = rnn_model.fit(train_data, train_labels, epochs=5, batch_size=64, validation_split=0.2
```

```
Epoch 1/5
313/313 [==============================] - 171s 509ms/step - loss: 0.5445 - accuracy: 0.7020 - val
_loss: 0.4560 - val_accuracy: 0.7916
Epoch 2/5
313/313 [==============================] - 149s 475ms/step - loss: 0.4227 - accuracy: 0.8090 - val
_loss: 0.5982 - val_accuracy: 0.6654
Epoch 3/5
313/313 [==============================] - 144s 460ms/step - loss: 0.3343 - accuracy: 0.8596 - val
_loss: 0.4208 - val_accuracy: 0.8406
Epoch 4/5
313/313 [==============================] - 145s 463ms/step - loss: 0.1450 - accuracy: 0.9481 - val
_loss: 0.4842 - val_accuracy: 0.8100
Epoch 5/5
313/313 [==============================] - 265s 849ms/step - loss: 0.0473 - accuracy: 0.9873 - val
_loss: 0.5961 - val_accuracy: 0.8186
```

In [7]:
```
# Evaluate RNN model
rnn_loss, rnn_accuracy = rnn_model.evaluate(test_data, test_labels)
print(f'RNN Test Accuracy: {rnn_accuracy:.2f}')
```

```
782/782 [==============================] - 63s 81ms/step - loss: 0.5948 - accuracy: 0.8194
RNN Test Accuracy: 0.82
```

In [8]:
```
# Plot training & validation accuracy
plt.plot(rnn_history.history['accuracy'], label='RNN Training Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='RNN Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


Model Accuracy
RNN Training Accuracy
RNN Validation Accuracy
Accuracy
Epochs