In [8]:
```
# Define a threshold value for anomalies
threshold_lstm = np.percentile(mse_lstm, 95)

# Predict anomalies
y_pred_lstm = mse_lstm > threshold_lstm

# Ensure y_pred_lstm has the same length as y_test
y_pred_lstm = y_pred_lstm[:len(y_test)]

# Flatten predictions for evaluation
y_pred_lstm = y_pred_lstm.flatten()

print(model_lstm.summary())

# Evaluate the performance
from sklearn.metrics import classification_report
print("LSTM Model Performance")
print(classification_report(y_test, y_pred_lstm))
```

**Model: "LSTM_AnomalyDetectionModel"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| LSTM_Layer_1 (LSTM) | (None, 1, 64) | 24,320 |
| Dropout_Layer_1 (Dropout) | (None, 1, 64) | 0 |
| LSTM_Layer_2 (LSTM) | (None, 32) | 12,416 |
| Dropout_Layer_2 (Dropout) | (None, 32) | 0 |
| Output_Layer (Dense) | (None, 30) | 990 |

**Total params:** 113,180 (442.11 KB)

**Trainable params:** 37,726 (147.37 KB)

**Non-trainable params:** 0 (0.00 B)

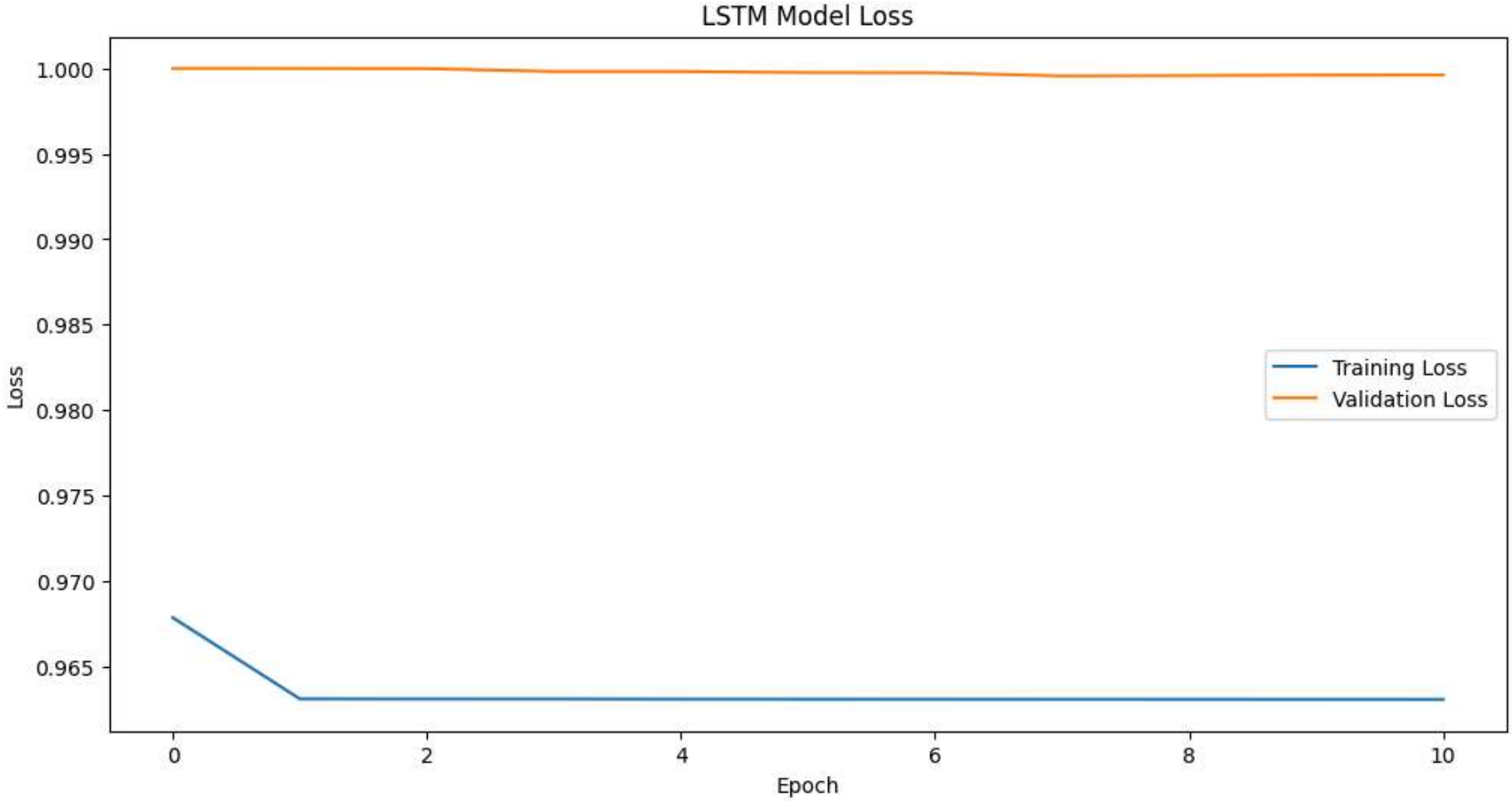
**Optimizer params:** 75,454 (294.75 KB)

```
None
LSTM Model Performance
              precision    recall  f1-score   support

           0       1.00      0.94      0.97    284315
           1       0.00      0.08      0.00       492

    accuracy                           0.94    284807
   macro avg       0.50      0.51      0.49    284807
weighted avg       1.00      0.94      0.97    284807
```

In [9]:
```python
# Plot LSTM model loss
plt.figure(figsize=(12, 6))
plt.plot(history_lstm.history['loss'], label='Training Loss')
plt.plot(history_lstm.history['val_loss'], label='Validation Loss')
plt.title('LSTM Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




# Recurrent Neural Networks (RNN)

- **Core Functionality**: Basic sequential pattern recognition
- **Comparative Characteristics**:
  - Simpler architectural design
  - Faster computational processing
  - Limited long-term dependency capture

## Building the RNN Model Architecture

### Layer Breakdown

- **Input Layer** (SimpleRNN with 64 units):
  - **Purpose**: This layer is responsible for reading the input features and capturing the dependencies over time.
  - **64 Units**: The number of units determines the network's capacity to learn complex patterns. We chose 64 units to balance model complexity and performance.
  - **return_sequences=True**: This ensures that the output from each time step is fed to the next layer, preserving the temporal sequences.
- **Dropout Layer** (0.2):
  - **Purpose**: Dropout helps prevent overfitting by randomly setting a fraction of the input units to 0 at each update during training time.
  - **0.2**: This means 20% of the units will be dropped randomly, promoting generalization in the model.
- **Second SimpleRNN Layer** (32 units):
  - **Purpose**: This layer further processes the output from the previous RNN layer to capture higher-level temporal patterns.
  - **32 Units**: A smaller number of units are used as we move deeper into the network,focusing on refining and condensing information.
  - **return_sequences=False**: Since this is the final RNN layer, we set it to return only the last output in the sequence, which is then passed to the dense output layer.
- **Second Dropout Layer** (0.2):
  - **Purpose**: This dropout layer continues to prevent overfitting by dropping 20% of the units, further ensuring that the model does not rely too heavily on specific neurons.
- **Output Layer** (Dense with Sigmoid Activation):
  - **Purpose**: Thefinaldenselayerwithasingle unit outputs the probability of a transaction being fraudulent.
  - **Sigmoid Activation**: The sigmoid function maps the output to a value between 0 and 1, making it suitable for binary classification.

## Summary

- **Stacked RNN Layers**: The combination of stacked SimpleRNN layers helps the model learn both immediate and long-term dependencies in transaction sequences.
- **Dropout Layers**: These are crucial for preventing overfitting, especially when working with sequential data, ensuring that the model generalizes well to unseen data.
- **Output Layer**: The dense layer with sigmoid activation provides the final binary classification decision.

In [18]:

```
#STEP 1: Import libraries
# Already done

#STEP 2: Preprocess the Data
# nothing to do, we reuse the same data as for LSTM model also for the RNN model

#STEP3: Build the RNN Model
# Define the RNN model
model_rnn = Sequential(name='RNN_AnomalyDetectionModel')
model_rnn.add(Input(shape=(1, X_train_reshaped.shape[2]), name='Input_Layer'))
model_rnn.add(SimpleRNN(64, return_sequences=True, name='RNN_Layer_1'))
model_rnn.add(Dropout(0.2, name='Dropout_Layer_1'))
model_rnn.add(SimpleRNN(32, return_sequences=False, name='RNN_Layer_2'))
model_rnn.add(Dropout(0.2, name='Dropout_Layer_2'))
model_rnn.add(Dense(X_train_reshaped.shape[2], activation='sigmoid', name='Output_Layer'))

#STEP4: Train the Model
# Train the LSTM model
model_rnn.compile(loss='mean_squared_error', optimizer=Adam(learning_rate=0.001))
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history_rnn = model_rnn.fit(X_train_reshaped, X_train_reshaped, epochs=20, batch_size=64, validation_data=(X_test_reshaped, X_tes
print(model_rnn.summary())

#STEP5: Evaluate the Model
# Calculate reconstruction error for RNN
# Evaluate in Batches
batch_size = 1000
#mse_rnn = np.array([], dtype=np.float64)
mse_rnn = np.array([])
for start in range(0, X_test_reshaped.shape[0], batch_size):
    end = min(start + batch_size, X_test_reshaped.shape[0])
    reconstructed_rnn_batch = model_rnn.predict(X_test_reshaped[start:end])
    mse_rnn_batch = np.mean(np.power(X_test_reshaped[start:end]-reconstructed_rnn_batch, 2), axis=2).flatten() # Ensure mse_rnn_b
    mse_rnn = np.concatenate((mse_rnn, mse_rnn_batch), axis=0)
```

```
Epoch 1/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 27s 5ms/step - loss: 0.9805 - val_loss: 1.0000
Epoch 2/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 21s 5ms/step - loss: 0.9419 - val_loss: 0.9999
Epoch 3/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 20s 5ms/step - loss: 0.9675 - val_loss: 0.9998
Epoch 4/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 21s 5ms/step - loss: 0.9570 - val_loss: 0.9998
Epoch 5/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 23s 5ms/step - loss: 0.9611 - val_loss: 0.9997
Epoch 6/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 25s 6ms/step - loss: 0.9742 - val_loss: 0.9997
Epoch 7/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 22s 5ms/step - loss: 0.9663 - val_loss: 0.9997
Epoch 8/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 24s 5ms/step - loss: 0.9556 - val_loss: 0.9997
Epoch 9/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 24s 5ms/step - loss: 0.9692 - val_loss: 0.9996
Epoch 10/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 26s 6ms/step - loss: 0.9652 - val_loss: 0.9996
Epoch 11/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 22s 5ms/step - loss: 0.9674 - val_loss: 0.9996
Epoch 12/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 23s 5ms/step - loss: 0.9661 - val_loss: 0.9996
Epoch 13/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 21s 5ms/step - loss: 0.9727 - val_loss: 0.9996
Epoch 14/20
4443/4443 ━━━━━━━━━━━━━━━━━━━━ 25s 6ms/step - loss: 0.9620 - val_loss: 0.9996
Model: "RNN_AnomalyDetectionModel"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| RNN_Layer_1 (SimpleRNN) | (None, 1, 64) | 6,080 |
| Dropout_Layer_1 (Dropout) | (None, 1, 64) | 0 |
| RNN_Layer_2 (SimpleRNN) | (None, 32) | 3,104 |
| Dropout_Layer_2 (Dropout) | (None, 32) | 0 |
| Output_Layer (Dense) | (None, 30) | 990 |

**Total params:** 30,524 (119.24 KB)

**Trainable params:** 10,174 (39.74 KB)

**Non-trainable params:** 0 (0.00 B)

**Optimizer params:** 20,350 (79.50 KB)