Group No 46

Group Member Names:

1. Satya Prakash Pandit - 2022AC05040
2. Pushkar Kumar Verma – 2022AC05272
3. G Krishna Sameera – 2022AC05407

Journal used for the implemetation

Journal title:Proposal of a model for credit risk prediction based on deep learning methods and SMOTE techniques for imbalanced dataset

Authors:Adaleta Gicić - Engineering Faculty, University of Sarajevo, Bosnia and Herzegovina Dženana Đonko - Engineering Faculty, University of Sarajevo, Bosnia and Herzegovina

Journal Name:The architecture of the deep learning models discussed in the paper involves Stacked LSTM and Stacked BiLSTM networks: Number of Layers: Both Stacked LSTM and Stacked BiLSTM architectures consist of three hidden layers.

Year:2021

1. Import the required libraries

In [1]:
```
##---------Type the code below this line-----------------##
import numpy as np
import pandas as pd
import seaborn as sn
import random
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras import layers
from imblearn.over_sampling import SMOTE
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Dropout
```

In [2]:
```
random.seed(42)          # Initialize the random number generator.
np.random.seed(42)       # With the seed reset, the same set of
                         # numbers will appear every time.
tf.random.set_seed(42)   # sets the graph-level random seed
```

2. Data Acquisition

For the problem identified by you, students have to find the data source themselves from any data source.

Provide the URL of the data used.

Write Code for converting the above downloaded data into a form suitable for DL

URL for data : https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data

In [3]:
```
##---------Type the code below this line-----------------##
##Load the original dataset :
data1=pd.read_csv('german.data',header = None,delim_whitespace = True)
data1.head()
#assigning columns names to data varible
data1.columns = ["chek_acc","mon","credit_his","purpose","Credit_amo","saving_amo",
data1.head()
# Identify NULL or Missing Values
missing_values = data1.isnull().sum()
print("Missing Values:")
print(missing_values)
# Handle Missing Values
df = data1.dropna()  # Drop rows with missing values
```

```
Missing Values:
chek_acc          0
mon               0
credit_his        0
purpose           0
Credit_amo        0
saving_amo        0
Pre_employ        0
instalrate        0
p_status          0
guatan            0
pre_res           0
property          0
age               0
installment       0
Housing           0
existing_cards    0
job               0
no_people         0
telephn           0
for_work          0
status            0
dtype: int64
```

In [4]:
```
data1.describe()
```

Out[4]:

	mon	Credit_amo	instalrate	pre_res	age	existing_cards	no_peo
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000
mean	20.903000	3271.258000	2.973000	2.845000	35.546000	1.407000	1.155
std	12.058814	2822.736876	1.118715	1.103718	11.375469	0.577654	0.362
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000	1.000
25%	12.000000	1365.500000	2.000000	2.000000	27.000000	1.000000	1.000
50%	18.000000	2319.500000	3.000000	3.000000	33.000000	1.000000	1.000
75%	24.000000	3972.250000	4.000000	4.000000	42.000000	2.000000	1.000
max	72.000000	18424.000000	4.000000	4.000000	75.000000	4.000000	2.000

In [5]:
```
##Loading the dataset numeric
data2=pd.read_csv('german.data-numeric',header = None,delim_whitespace = True)
data2.head()
#assigning columns names to data varible
data2.columns = ["chek_acc","mon","credit_his","purpose","Credit_amo","saving_amo",
data2.head()
```

Out[5]:

	chek_acc	mon	credit_his	purpose	Credit_amo	saving_amo	Pre_employ	instalrate	p_status
0	1	6	4	12	5	5	3	4	1
1	2	48	2	60	1	3	2	2	1
2	4	12	4	21	1	4	3	3	1
3	1	42	2	79	1	4	3	4	2
4	1	24	3	49	1	3	3	4	4

5 rows × 25 columns

3. Data Preparation

Perform the data prepracessing that is required for the data that you have downloaded.

This stage depends on the dataset that is used.

In [6]:
```
#load the dataset

##pre process the data
X = data2.drop('status', axis=1)  # Replace 'target_variable' with your actual targ
##to make output value  binary represenations 0-good 1-bad
y = data2['status']-1
type(data2)
```

Out[6]:
```
pandas.core.frame.DataFrame
```

In [7]:
```
X.head()
y.head()
```

Out[7]:
```
0    0
1    1
2    0
3    0
4    1
Name: status, dtype: int64
```

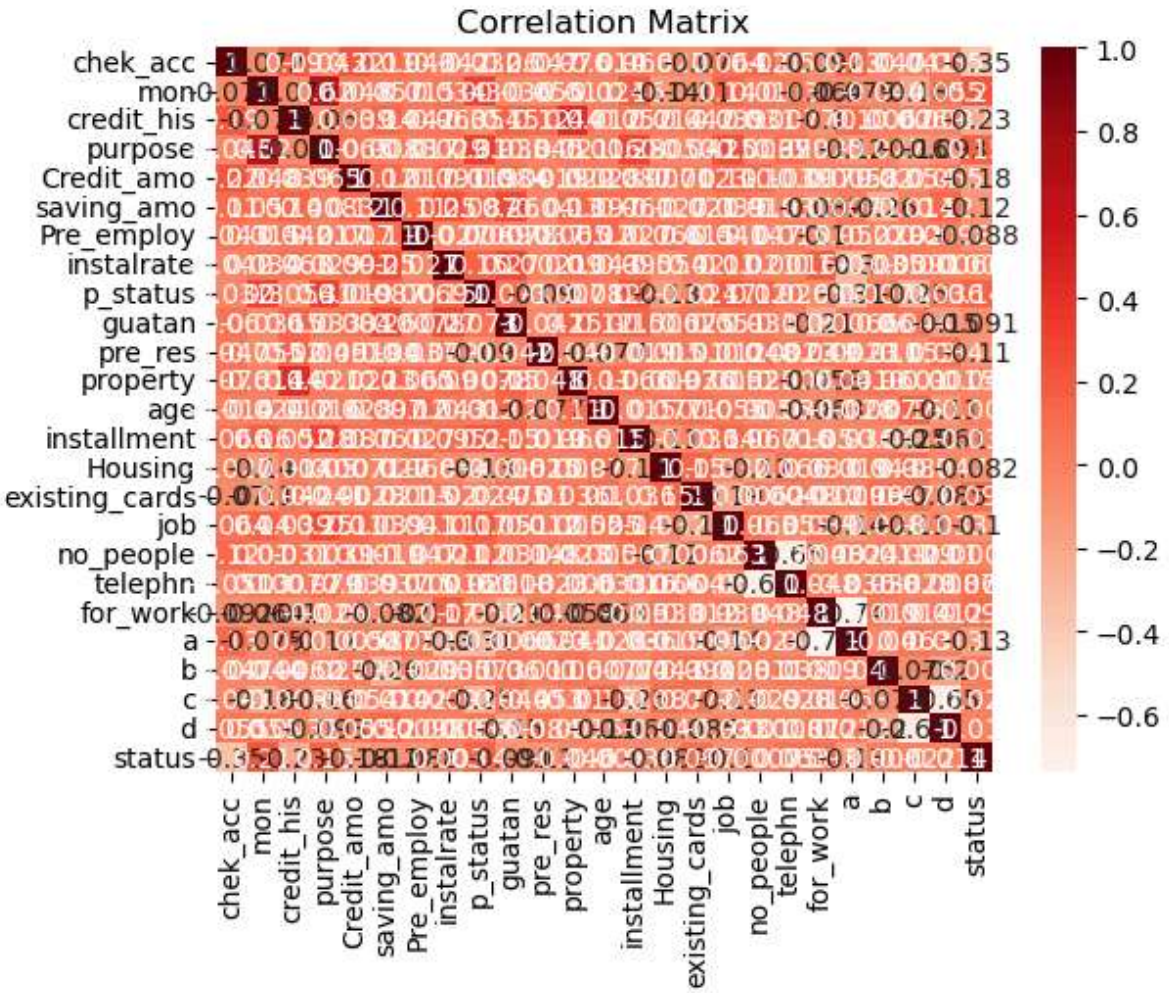
In [8]:
```
##---------Type the code below this line-----------------##

## Split the data into training set and testing set
##---------Type the code below this line-----------------##
# Splitting the dataset for training and testing the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
# Let us check shape of the training & test set
print(X_train.shape, y_test.shape)
## Identify the target variables.
##---------Type the code below this line-----------------##
#created histogram for credit amount
```

```
#sn.distplot( data2.Credit_amo, kde = False )
#plt.title( "Histogram of Credit Amount ", fontsize = 15)
#plt.ylabel( "Frequency")
#created pairplot to view data

#sn.pairplot(data2)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
(800, 24) (200,)
```

In [9]:
```
##Draw a a correlation matrix
sn.heatmap(data2.corr(), cmap='Reds', annot=True)
plt.title('Correlation Matrix')
```

Out[9]:
```
Text(0.5, 1.0, 'Correlation Matrix')
```



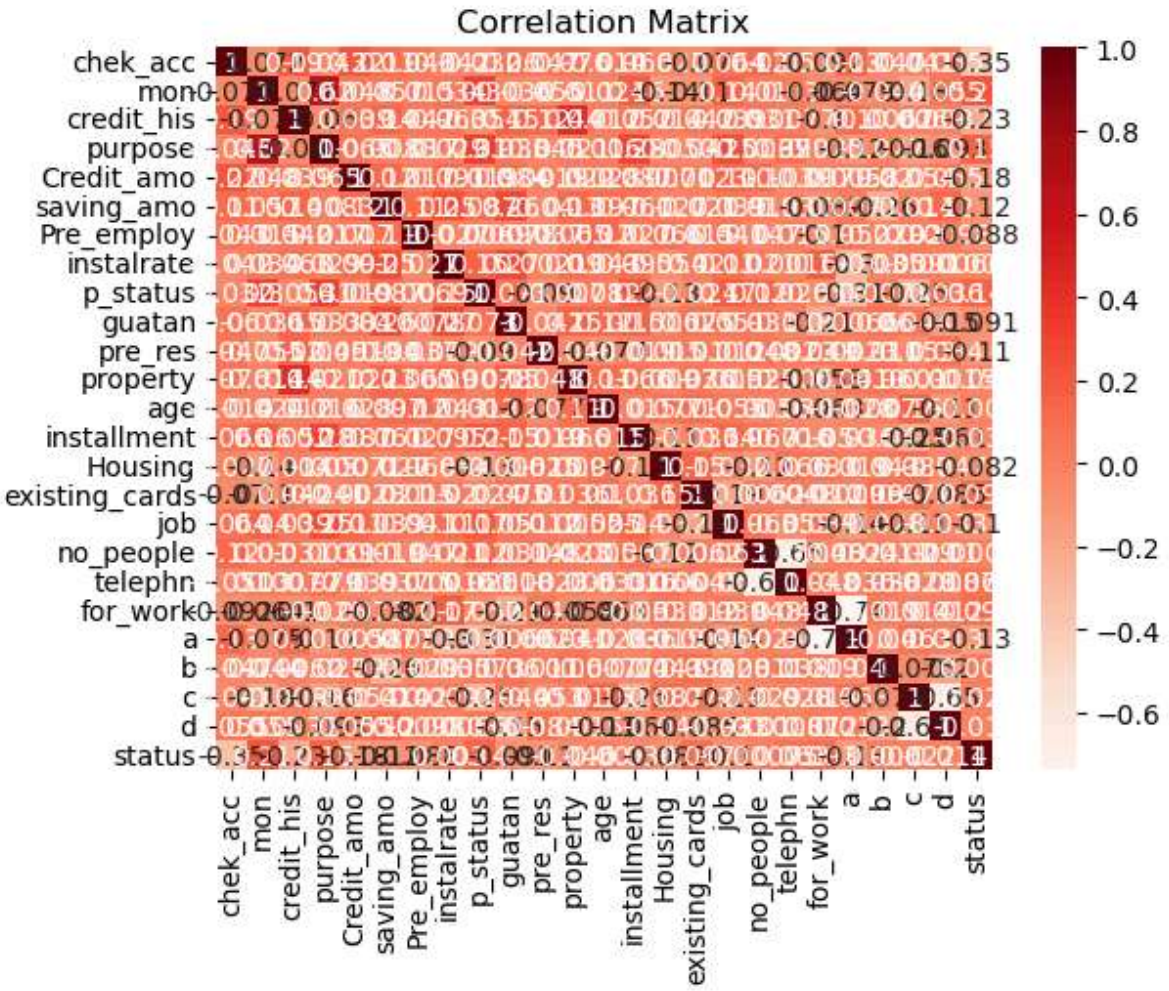
Report the feature representation that is being used for training the model. ##---------Type below this line----------## ## Display the feature representation being used for training the model print("Feature Representation for Training the Model:") print(X_train.columns)

In [10]:
```
## Display the feature representation being used for training the model
print("Feature Representation for Training the Model:")
print(X_train.columns)
##Draw a a correlation matrix
sn.heatmap(data2.corr(), cmap='Reds', annot=True)
plt.title('Correlation Matrix')
```

```
Feature Representation for Training the Model:
Index(['chek_acc', 'mon', 'credit_his', 'purpose', 'Credit_amo', 'saving_amo',
       'Pre_employ', 'instalrate', 'p_status', 'guatan', 'pre_res', 'property',
       'age', 'installment', 'Housing', 'existing_cards', 'job', 'no_people',
       'telephn', 'for_work', 'a', 'b', 'c', 'd'],
      dtype='object')
```

Out[10]:

```
Text(0.5, 1.0, 'Correlation Matrix')
```



4. Deep Neural Network Architecture

Create a model object

dnnModel = models.Sequential()

4.1 Design the architecture that you will be using

- CNN / RNN / Transformer as per the journal referenced

In [11]:

```
##---------Type the code below this line-----------------##
dnnModel = models.Sequential()
# Apply SMOTE to address class imbalance
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)

# Reshape the data for LSTM models
X_train_reshaped = X_train_resampled.reshape((X_train_resampled.shape[0], 1, X_trai
```

```
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.s

# Build Stacked LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(60, activation='relu', input_shape=(X_train_reshaped.shape[1],
model_lstm.add(LSTM(80, activation='relu', return_sequences=True))
model_lstm.add(LSTM(120, activation='relu'))
model_lstm.add(Dense(1, activation='sigmoid'))


# Compile the model
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy

# Evaluate the model
accuracy_lstm_train = model_lstm.evaluate(X_train_reshaped, y_train_resampled)[1]
accuracy_lstm_test = model_lstm.evaluate(X_test_reshaped, y_test)[1]
model_lstm.summary()
print(f"Training Accuracy (Stacked LSTM): {accuracy_lstm_train}")
print(f"Testing Accuracy (Stacked LSTM): {accuracy_lstm_test}")
```

```
35/35 [==============================] - 1s 6ms/step - loss: 0.6932 - accuracy: 0.
4329
7/7 [==============================] - 0s 4ms/step - loss: 0.6932 - accuracy: 0.38
00
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 60)             20400

 lstm_1 (LSTM)               (None, 1, 80)             45120

 lstm_2 (LSTM)               (None, 120)               96480

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 162121 (633.29 KB)
Trainable params: 162121 (633.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training Accuracy (Stacked LSTM): 0.43291592597961426
Testing Accuracy (Stacked LSTM): 0.3799999952316284
```

4.2 DNN Report

Report the following and provide justification for the same.

- Number of layers
- Number of units in each layer
- Total number of trainable parameters

##---------Type the answer below this line----------## model_lstm.summary()

5. Training the model

In [12]:
```
# Configure the training, by using appropriate optimizers, regularizations and loss
##---------Type the code below this line----------##
# Compile the model
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy
```

```
# Train the model
history=model_lstm.fit(X_train_reshaped, y_train_resampled, epochs=50, batch_size=3
```

```
Epoch 1/50
28/28 [==============================] - 8s 44ms/step - loss: 0.6879 - accuracy:
0.6141 - val_loss: 0.7422 - val_accuracy: 0.0000e+00
Epoch 2/50
28/28 [==============================] - 0s 18ms/step - loss: 0.6534 - accuracy:
0.6253 - val_loss: 0.9112 - val_accuracy: 0.0000e+00
Epoch 3/50
28/28 [==============================] - 0s 14ms/step - loss: 0.5638 - accuracy:
0.6253 - val_loss: 0.9180 - val_accuracy: 0.0045
Epoch 4/50
28/28 [==============================] - 0s 17ms/step - loss: 0.5137 - accuracy:
0.6957 - val_loss: 0.8266 - val_accuracy: 0.6205
Epoch 5/50
28/28 [==============================] - 0s 11ms/step - loss: 0.4859 - accuracy:
0.7752 - val_loss: 0.7483 - val_accuracy: 0.7277
Epoch 6/50
28/28 [==============================] - 0s 10ms/step - loss: 0.4571 - accuracy:
0.7919 - val_loss: 0.5911 - val_accuracy: 0.8036
Epoch 7/50
28/28 [==============================] - 0s 11ms/step - loss: 0.4350 - accuracy:
0.7886 - val_loss: 0.5163 - val_accuracy: 0.8036
Epoch 8/50
28/28 [==============================] - 0s 11ms/step - loss: 0.4158 - accuracy:
0.7919 - val_loss: 0.5334 - val_accuracy: 0.7902
Epoch 9/50
28/28 [==============================] - 0s 11ms/step - loss: 0.4018 - accuracy:
0.8098 - val_loss: 0.4775 - val_accuracy: 0.8170
Epoch 10/50
28/28 [==============================] - 0s 12ms/step - loss: 0.3861 - accuracy:
0.8132 - val_loss: 0.4382 - val_accuracy: 0.8348
Epoch 11/50
28/28 [==============================] - 0s 10ms/step - loss: 0.3656 - accuracy:
0.8277 - val_loss: 0.4161 - val_accuracy: 0.8527
Epoch 12/50
28/28 [==============================] - 0s 10ms/step - loss: 0.3508 - accuracy:
0.8412 - val_loss: 0.4279 - val_accuracy: 0.8393
Epoch 13/50
28/28 [==============================] - 0s 12ms/step - loss: 0.3375 - accuracy:
0.8479 - val_loss: 0.3932 - val_accuracy: 0.8616
Epoch 14/50
28/28 [==============================] - 0s 11ms/step - loss: 0.3121 - accuracy:
0.8546 - val_loss: 0.3588 - val_accuracy: 0.8839
Epoch 15/50
28/28 [==============================] - 0s 10ms/step - loss: 0.2962 - accuracy:
0.8747 - val_loss: 0.3445 - val_accuracy: 0.8839
Epoch 16/50
28/28 [==============================] - 0s 11ms/step - loss: 0.2822 - accuracy:
0.8859 - val_loss: 0.2746 - val_accuracy: 0.9196
Epoch 17/50
28/28 [==============================] - 0s 10ms/step - loss: 0.2683 - accuracy:
0.8837 - val_loss: 0.3246 - val_accuracy: 0.8884
Epoch 18/50
28/28 [==============================] - 0s 10ms/step - loss: 0.2594 - accuracy:
0.8926 - val_loss: 0.2668 - val_accuracy: 0.9420
Epoch 19/50
28/28 [==============================] - 0s 10ms/step - loss: 0.2394 - accuracy:
0.9038 - val_loss: 0.2106 - val_accuracy: 0.9554
Epoch 20/50
28/28 [==============================] - 0s 12ms/step - loss: 0.2429 - accuracy:
0.8971 - val_loss: 0.2810 - val_accuracy: 0.9107
Epoch 21/50
28/28 [==============================] - 0s 12ms/step - loss: 0.2173 - accuracy:
0.9116 - val_loss: 0.2869 - val_accuracy: 0.9107
Epoch 22/50
```

```
28/28 [==============================] - 0s 10ms/step - loss: 0.2057 - accuracy:
0.9128 - val_loss: 0.2344 - val_accuracy: 0.9464
Epoch 23/50
28/28 [==============================] - 0s 11ms/step - loss: 0.1907 - accuracy:
0.9239 - val_loss: 0.1873 - val_accuracy: 0.9688
Epoch 24/50
28/28 [==============================] - 0s 11ms/step - loss: 0.1805 - accuracy:
0.9284 - val_loss: 0.2038 - val_accuracy: 0.9464
Epoch 25/50
28/28 [==============================] - 0s 11ms/step - loss: 0.1702 - accuracy:
0.9340 - val_loss: 0.1511 - val_accuracy: 0.9732
Epoch 26/50
28/28 [==============================] - 0s 10ms/step - loss: 0.1606 - accuracy:
0.9385 - val_loss: 0.1519 - val_accuracy: 0.9643
Epoch 27/50
28/28 [==============================] - 0s 10ms/step - loss: 0.1537 - accuracy:
0.9385 - val_loss: 0.1256 - val_accuracy: 0.9732
Epoch 28/50
28/28 [==============================] - 0s 11ms/step - loss: 0.1419 - accuracy:
0.9418 - val_loss: 0.1735 - val_accuracy: 0.9420
Epoch 29/50
28/28 [==============================] - 0s 10ms/step - loss: 0.1307 - accuracy:
0.9485 - val_loss: 0.1306 - val_accuracy: 0.9688
Epoch 30/50
28/28 [==============================] - 0s 10ms/step - loss: 0.1219 - accuracy:
0.9530 - val_loss: 0.1349 - val_accuracy: 0.9732
Epoch 31/50
28/28 [==============================] - 0s 14ms/step - loss: 0.1342 - accuracy:
0.9441 - val_loss: 0.0861 - val_accuracy: 0.9866
Epoch 32/50
28/28 [==============================] - 0s 11ms/step - loss: 0.1500 - accuracy:
0.9396 - val_loss: 0.1093 - val_accuracy: 0.9777
Epoch 33/50
28/28 [==============================] - 0s 11ms/step - loss: 0.1205 - accuracy:
0.9474 - val_loss: 0.1208 - val_accuracy: 0.9688
Epoch 34/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0979 - accuracy:
0.9609 - val_loss: 0.1012 - val_accuracy: 0.9732
Epoch 35/50
28/28 [==============================] - 0s 14ms/step - loss: 0.0878 - accuracy:
0.9676 - val_loss: 0.0789 - val_accuracy: 0.9911
Epoch 36/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0854 - accuracy:
0.9642 - val_loss: 0.0882 - val_accuracy: 0.9821
Epoch 37/50
28/28 [==============================] - 0s 12ms/step - loss: 0.1262 - accuracy:
0.9474 - val_loss: 0.1141 - val_accuracy: 0.9598
Epoch 38/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0984 - accuracy:
0.9575 - val_loss: 0.0675 - val_accuracy: 0.9955
Epoch 39/50
28/28 [==============================] - 0s 12ms/step - loss: 0.0782 - accuracy:
0.9676 - val_loss: 0.0784 - val_accuracy: 0.9911
Epoch 40/50
28/28 [==============================] - 0s 13ms/step - loss: 0.0753 - accuracy:
0.9687 - val_loss: 0.0857 - val_accuracy: 0.9732
Epoch 41/50
28/28 [==============================] - 0s 10ms/step - loss: 0.0727 - accuracy:
0.9664 - val_loss: 0.0763 - val_accuracy: 0.9821
Epoch 42/50
28/28 [==============================] - 0s 10ms/step - loss: 0.0609 - accuracy:
0.9709 - val_loss: 0.0556 - val_accuracy: 0.9955
Epoch 43/50
28/28 [==============================] - 0s 14ms/step - loss: 0.0559 - accuracy:
```

```
0.9743 - val_loss: 0.0412 - val_accuracy: 1.0000
Epoch 44/50
28/28 [==============================] - 0s 14ms/step - loss: 0.0516 - accuracy:
0.9754 - val_loss: 0.0462 - val_accuracy: 1.0000
Epoch 45/50
28/28 [==============================] - 0s 14ms/step - loss: 0.0488 - accuracy:
0.9765 - val_loss: 0.0451 - val_accuracy: 0.9955
Epoch 46/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0453 - accuracy:
0.9776 - val_loss: 0.0430 - val_accuracy: 1.0000
Epoch 47/50
28/28 [==============================] - 0s 12ms/step - loss: 0.0427 - accuracy:
0.9810 - val_loss: 0.0440 - val_accuracy: 0.9955
Epoch 48/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0405 - accuracy:
0.9810 - val_loss: 0.0452 - val_accuracy: 0.9955
Epoch 49/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0378 - accuracy:
0.9821 - val_loss: 0.0397 - val_accuracy: 0.9955
Epoch 50/50
28/28 [==============================] - 0s 11ms/step - loss: 0.0353 - accuracy:
0.9821 - val_loss: 0.0355 - val_accuracy: 0.9955
```

6. Test the model

In [13]:
```
##---------Type the code below this line------------------##
# Evaluate the model
accuracy_lstm_train = model_lstm.evaluate(X_train_reshaped, y_train_resampled)[1]
accuracy_lstm_test = model_lstm.evaluate(X_test_reshaped, y_test)[1]
```

```
35/35 [==============================] - 1s 5ms/step - loss: 0.0320 - accuracy: 0.
9884
7/7 [==============================] - 0s 5ms/step - loss: 3.1874 - accuracy: 0.74
00
```

7. Report the result

1. Plot the training and validation accuracy history.
2. Plot the training and validation loss history.
3. Report the testing accuracy and loss.
4. Show Confusion Matrix for testing dataset.
5. Report values for preformance study metrics like accuracy, precision, recall, F1 Score.

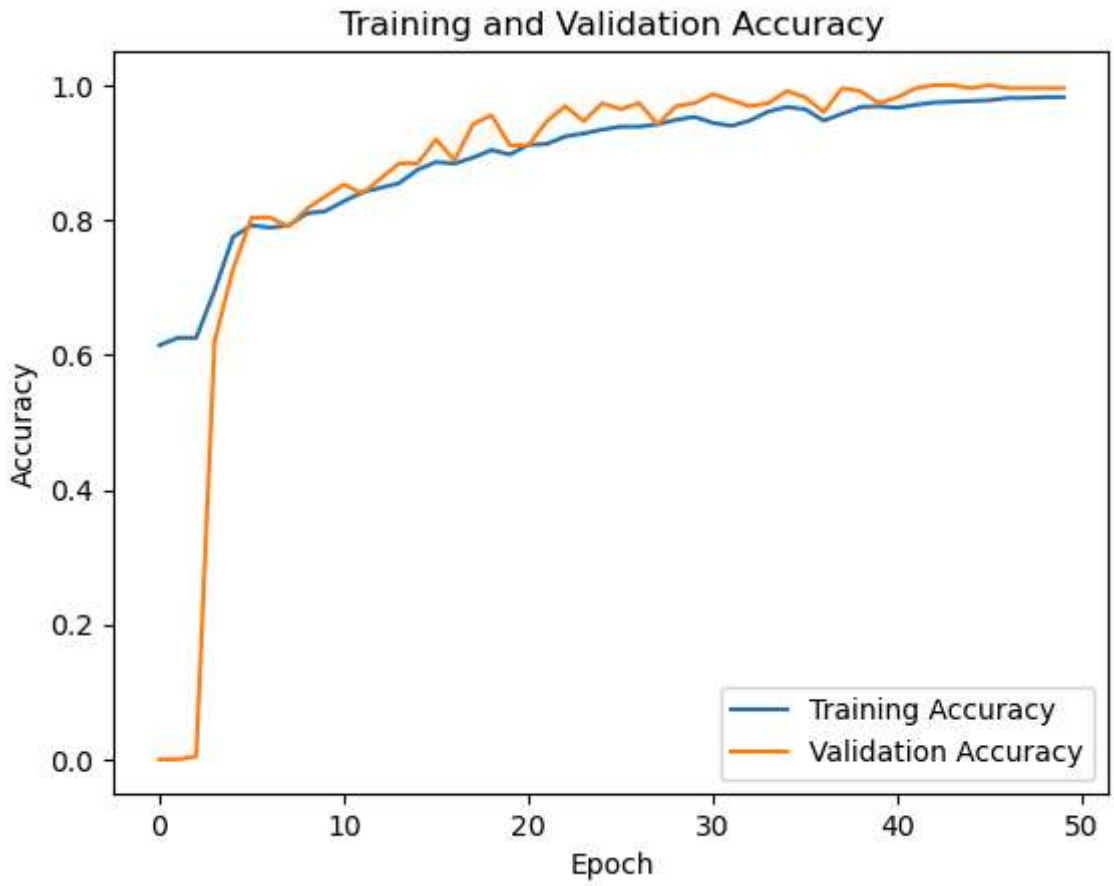
In [14]:
```
##---------Type the code below this line------------------##
model_lstm.summary()
print(f"Training Accuracy (Stacked LSTM): {accuracy_lstm_train}")
print(f"Testing Accuracy (Stacked LSTM): {accuracy_lstm_test}")
```

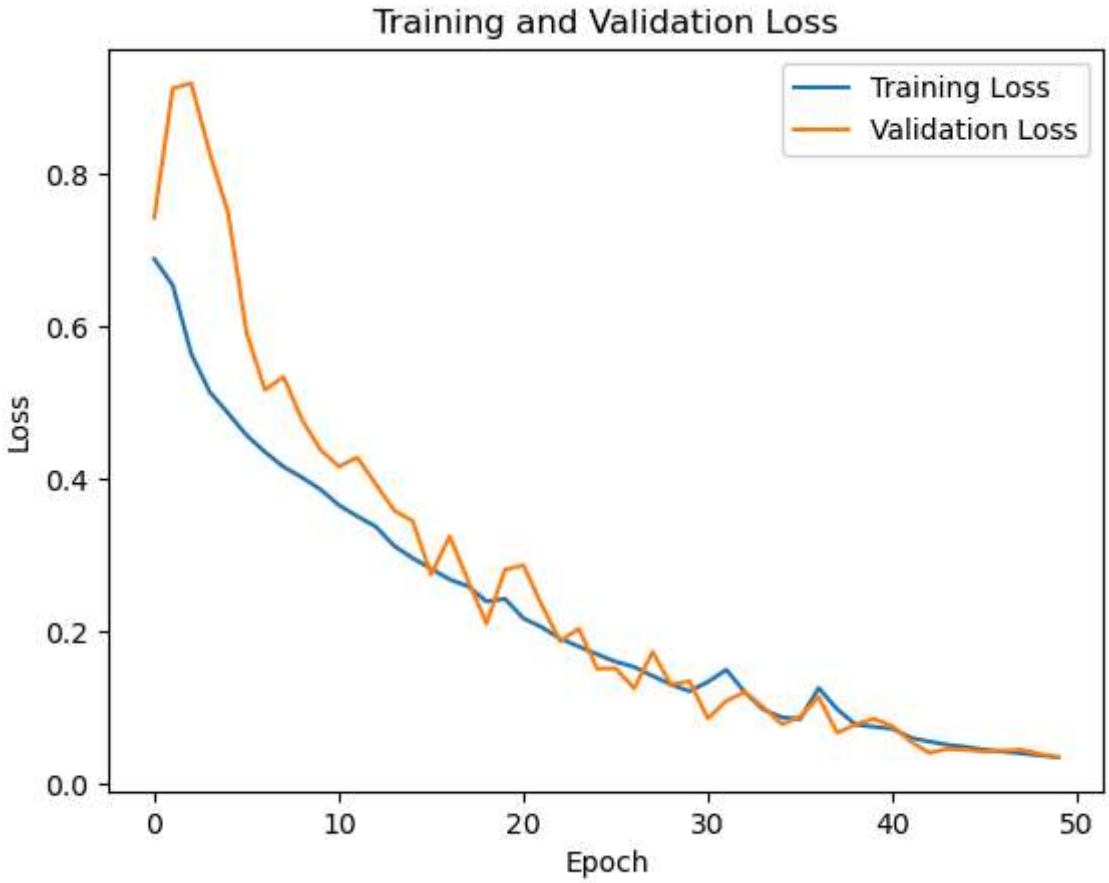
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 60)             20400

 lstm_1 (LSTM)               (None, 1, 80)             45120

 lstm_2 (LSTM)               (None, 120)               96480

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 162121 (633.29 KB)
Trainable params: 162121 (633.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training Accuracy (Stacked LSTM): 0.9883720874786377
Testing Accuracy (Stacked LSTM): 0.7400000095367432
```

In [23]:

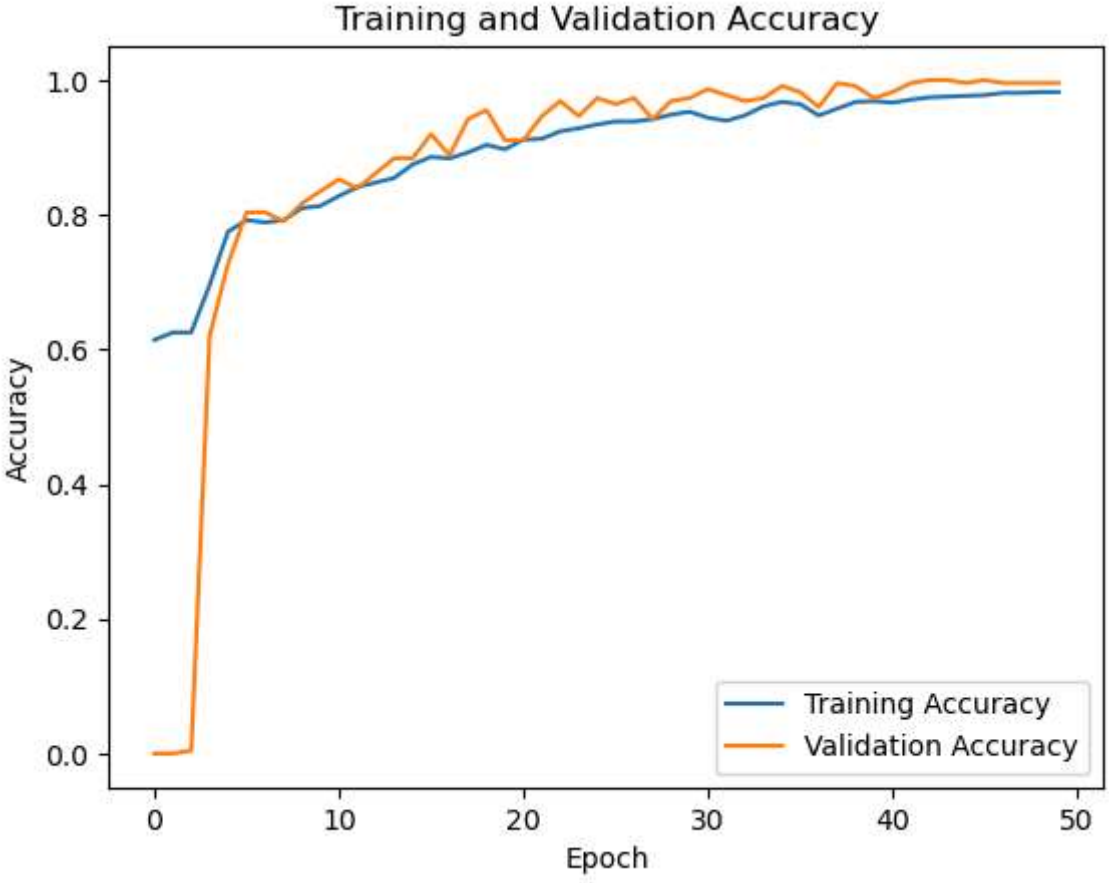
```python
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
# Train the model and store the history
#history = model_lstm.fit(X_train_reshaped, y_train_resampled, epochs=50, batch_siz
# Plot the training and validation accuracy history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Report the result
print("Testing Accuracy: {:.2f}%".format(accuracy_lstm_test * 100))
print("Testing Loss: {:.4f}".format(model_lstm.evaluate(X_test_reshaped, y_test)[0]

# Plot the training and validation accuracy history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

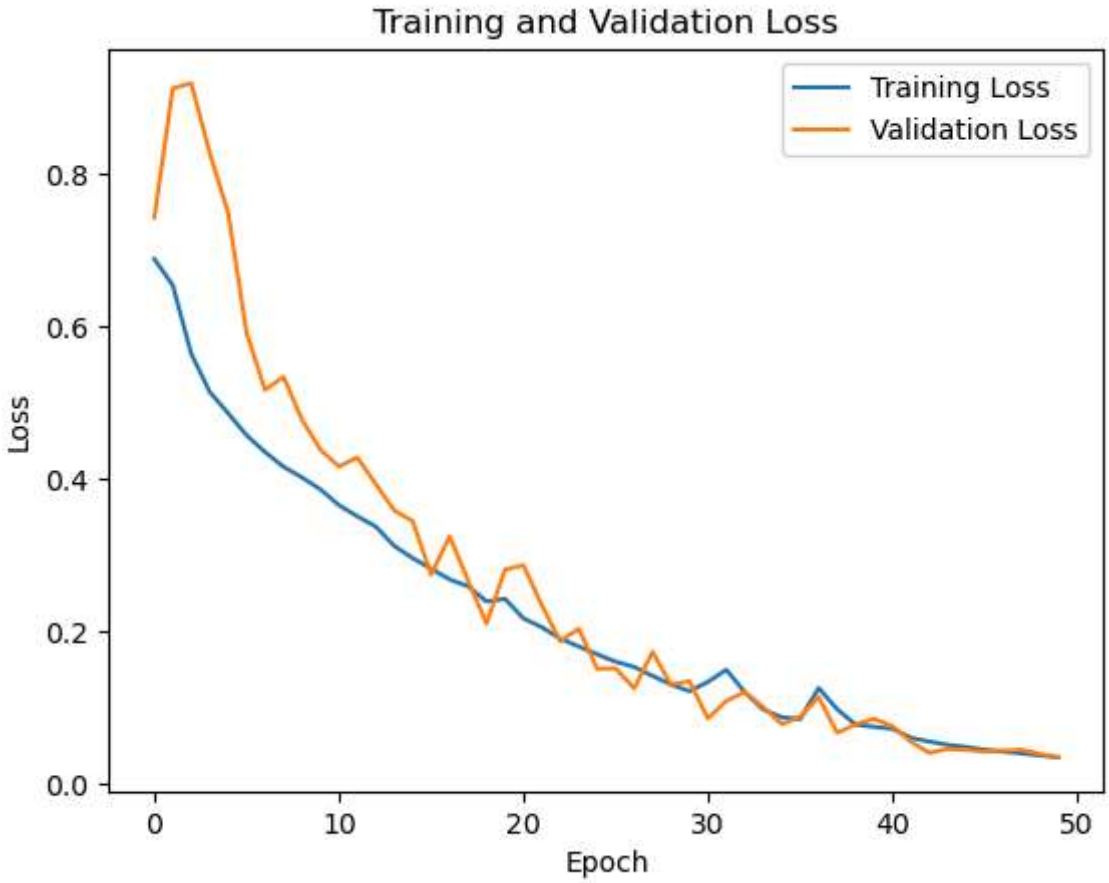
```
# Assuming binary classification
threshold = 0.5  # Adjust this threshold based on your needs

# Obtain binary predictions based on the threshold
y_pred_binary = (model_lstm.predict(X_test_reshaped) > threshold).astype(int)

# Confusion Matrix for testing dataset
conf_matrix = confusion_matrix(y_test, y_pred_binary)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Good', 'Bad'], yticklabels=['Good', 'Bad'])
plt.title('Confusion Matrix for Testing Dataset')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Performance Study Metrics
classification_rep = classification_report(y_test, y_pred_binary, target_names=['Go
print("Performance Study Metrics:")
print(classification_rep)
```



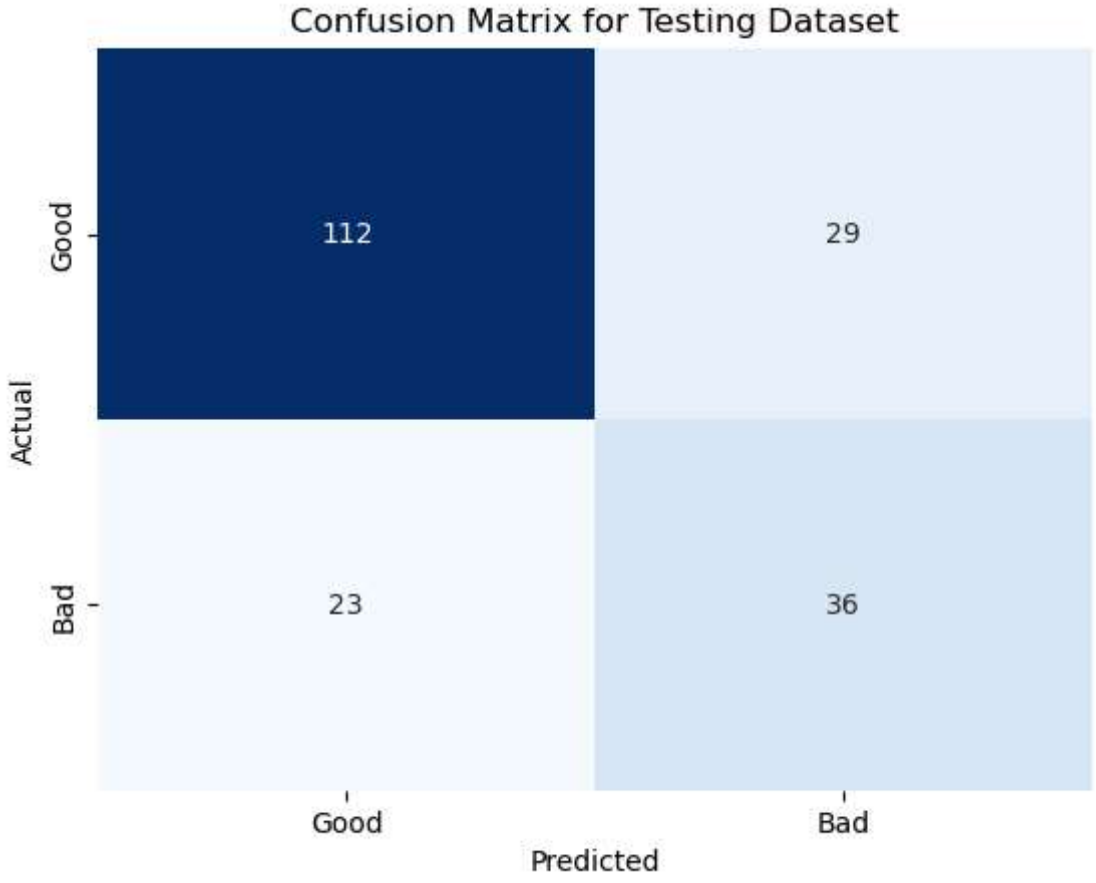



```
Testing Accuracy: 74.00%
7/7 [==============================] - 0s 3ms/step - loss: 3.1874 - accuracy: 0.74
00
Testing Loss: 3.1874
```






```
7/7 [==============================] - 0s 3ms/step
```



```
Performance Study Metrics:
              precision    recall  f1-score   support

        Good       0.83      0.79      0.81       141
         Bad       0.55      0.61      0.58        59

    accuracy                           0.74       200
   macro avg       0.69      0.70      0.70       200
weighted avg       0.75      0.74      0.74       200
```

NOTE

All Late Submissions will incur a **penalty of -2 marks** . So submit your assignments on time.

Good Luck

In [ ]: