```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
import seaborn as sns
```

```
 from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
  df=pd.read_csv(r"/content/drive/MyDrive/Credit_Card.csv")
```

```
  df.head()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PA
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	0.0	689.0	
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0	
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0	
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0	
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131.0	2000.0	36681.0	1

5 rows × 25 columns

```
  df["MARRIAGE"].value_counts()
```

```
    2    15964
    1    13659
    3      323
    0       54
    Name: MARRIAGE, dtype: int64
```

```
  df["SEX"].value_counts()
```

```
    2    18112
    1    11888
    Name: SEX, dtype: int64
```

```
df.columns
```

```
    Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
           'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
           'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
           'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
           'default.payment.next.month'],
          dtype='object')
```

```
# ID - Identification for a Person
# LIMIT_BAL- What is total LIMIT of credit Card
# SEX - Gender
# Education - what label of education - 1 School, 2 College , 3 batchler etc
# Marriage - is guy married or not married
# Age - (years)
# Pay_0,2,3,4,5,6  - Paid on time or not if yes then how many days befor the last day or after how many days.
# - last six month -
# Bill_amt1 - bill amount from last month
# bill_amt2 - bill amount from more one month
# PAY_AMT1 - paid amount for that month  (Pay_AMT1 to 6 )
# default.payment.next.month - will this guy will default in next month ( Target Variable)
```

```
df.drop(columns=["ID"], inplace=True)
```

```
  df.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 30000 entries, 0 to 29999
```

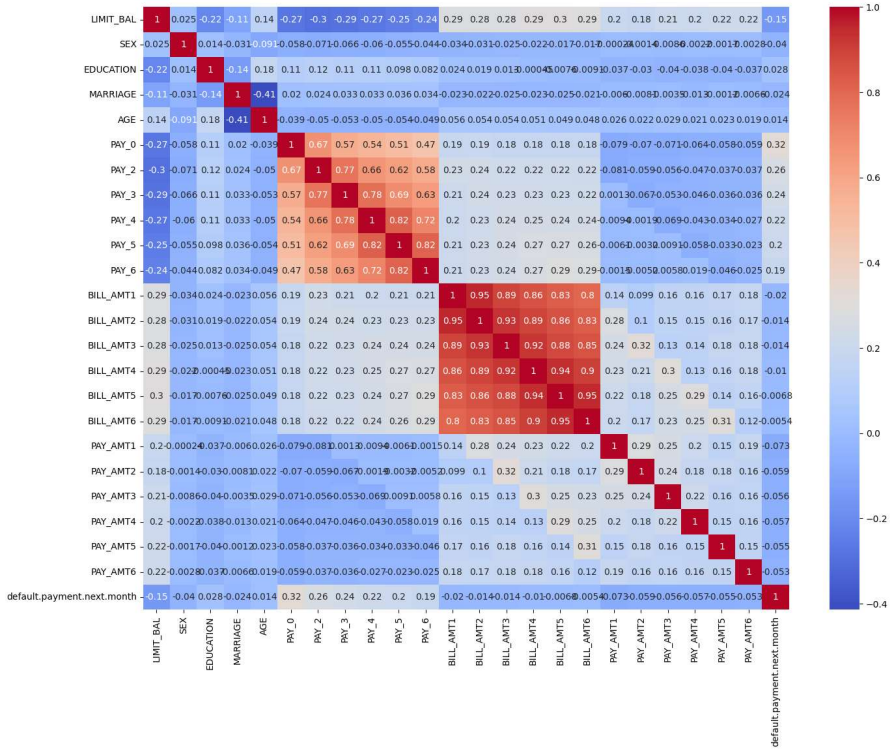
```
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   LIMIT_BAL                   30000 non-null  float64
 1   SEX                         30000 non-null  int64  
 2   EDUCATION                   30000 non-null  int64  
 3   MARRIAGE                    30000 non-null  int64  
 4   AGE                         30000 non-null  int64  
 5   PAY_0                       30000 non-null  int64  
 6   PAY_2                       30000 non-null  int64  
 7   PAY_3                       30000 non-null  int64  
 8   PAY_4                       30000 non-null  int64  
 9   PAY_5                       30000 non-null  int64  
 10  PAY_6                       30000 non-null  int64  
 11  BILL_AMT1                   30000 non-null  float64
 12  BILL_AMT2                   30000 non-null  float64
 13  BILL_AMT3                   30000 non-null  float64
 14  BILL_AMT4                   30000 non-null  float64
 15  BILL_AMT5                   30000 non-null  float64
 16  BILL_AMT6                   30000 non-null  float64
 17  PAY_AMT1                    30000 non-null  float64
 18  PAY_AMT2                    30000 non-null  float64
 19  PAY_AMT3                    30000 non-null  float64
 20  PAY_AMT4                    30000 non-null  float64
 21  PAY_AMT5                    30000 non-null  float64
 22  PAY_AMT6                    30000 non-null  float64
 23  default.payment.next.month  30000 non-null  int64  
dtypes: float64(13), int64(11)
memory usage: 5.5 MB
```

Missing Values

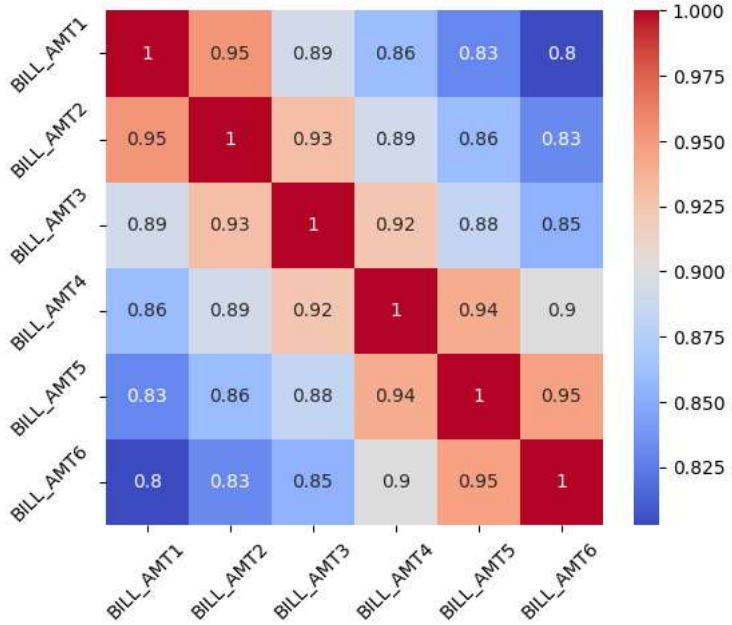
```
df.isnull().sum()
```

```
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default.payment.next.month    0
dtype: int64
```

```
plt.figure(figsize=(16,12))
cr=df.corr()
sns.heatmap(cr, annot=True, cmap="coolwarm")
plt.show()
```




```
cf=df[['BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',]].corr()
plt.figure(figsize=(6,5))
sns.heatmap(cf, annot=True, cmap="coolwarm")
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
```




```
# As we have cleaned data that means we can run classifier on this data directly
```

```
# ML Approach
# 1. Split x, y
# 2. Get train data and test data ( x_train, x_test, y_train, y_test)
# 3. get model and get its object and set the hyperparameters
#             from sklearn.ensembple import RandomForestClassifier
#             rf=RandomForestClassifier(max_depth=12, n_estimator=100, min_sample_split=30)
# 4. fit the model
#             rf.fit(x_train, y_train)
# 5. Evaluate the model
#             rf.score(x_train, y_train)
#             confusion matrix, accuracy recall , etc



  # DL approach
  # 1. Split x, y
  # 2. Get train data and test data ( x_train, x_test, y_train, y_test,x_val, y_val)
  # 3. get the model and make its object
  #           from keras.model import Sequential                                                    - importing model from keras package
  #           model=Sequential()                                                                   - creating object for the model
  #           model.add(Dense(units=8,activation="linear",kernel_initializer="uniform",input_dim=11 )) - input Layer ( input_dim= No of x var
  #           model.add(Dense(units=16,activation="linear",kernel_initializer="uniform"))          - hidden layer
  #           model.add(Dense(units=32,activation="linear",kernel_initializer="uniform"))          - hidden layer
  #           model.add(Dense(units=16,activation="linear",kernel_initializer="uniform"))          - hidden layer
  #           model.add(Dense(units=1, activation="sigmoid",kernel_initializer="uniform" ))         - Output Layer

  #4. compilation step
  #          model.compile(optimizer="sgd", loss='binary_crossentropy', metrics=["accuracy"])      - Compilation stage
  # 5. train the model/ fit the model
  #          model.fit(x_train, y_train , epoch=10, batch_size=32, validation =(x_val, y_val))     - Fitting the model
  # 6. Evaluate the model
  #       Evaluation of the model is same as ML models


df.columns

     Index(['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
            'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
            'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
            'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
            'default.payment.next.month'],
           dtype='object')


# In any Deeplearning Model make sure data is standardized / normalized
y=df['default.payment.next.month']
x=df.drop(columns=['default.payment.next.month'])



x_train, x_test, y_train, y_test=train_test_split(x,y, test_size=.3, random_state=88)


# Standardization of X Variables
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
st_x=st.fit(x_train)
x_train_st=st_x.transform(x_train)
x_test_st=st_x.transform(x_test)


# Model for Deeplearning

#  packages for Deeplearning

import keras
from keras import Sequential
from keras import activations, initializers, regularizers, constraints
from keras.layers import Dense, Activation


x_train.shape

     (21000, 23)


y.nunique()

     2
```

```
# Model for Deeplearning
model=Sequential()
model.add(Dense(units=23, activation="relu", kernel_initializer="uniform", input_dim=23))  # input layer
model.add(Dense(units=64, activation="relu", kernel_initializer="uniform"))  # hidden layer
# model.add(Dense(units=64, activation="relu", kernel_initializer="uniform"))  # hidden layer
# model.add(Dense(units=128, activation="relu", kernel_initializer="uniform"))  # hidden layer
# model.add(Dense(units=256, activation="relu", kernel_initializer="uniform"))  # hidden layer
# model.add(Dense(units=32, activation="relu", kernel_initializer="uniform"))  # hidden layer
model.add(Dense(units=8, activation="relu", kernel_initializer="uniform"))  # hidden layer
model.add(Dense(units=1, activation="sigmoid", kernel_initializer="uniform"))     # Output Layer


# if y has 2 category : then final layer looks : model.add(Dense(units=1, activation="sigmoid", kernel_initializer="uniform"))
# if y has more than 2 category (for example 5 category) : model.add(Dense(units=5, activation="softmax", kernel_initializer="uniform"))
# if y is continuous : model.add(Dense(units=1, activation="linear", kernel_initializer="uniform"))



opt = keras.optimizers.Adam(learning_rate=0.00000001)
# model.compile(loss='categorical_crossentropy', optimizer=opt)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=["accuracy"])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 23)                552

 dense_1 (Dense)             (None, 64)                1536

 dense_2 (Dense)             (None, 8)                 520

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 2617 (10.22 KB)
Trainable params: 2617 (10.22 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# fitting the model

model.fit(x_train_st, y_train, validation_data=(x_test_st,y_test), epochs=50, batch_size=32)
```

```
Epoch 40/50
657/657 [==============================] - 3s 4ms/step - loss: 0.6931 - accuracy: 0.7785 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 41/50
657/657 [==============================] - 3s 4ms/step - loss: 0.6931 - accuracy: 0.7785 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 42/50
657/657 [==============================] - 3s 5ms/step - loss: 0.6931 - accuracy: 0.7785 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 43/50
657/657 [==============================] - 3s 5ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 44/50
657/657 [==============================] - 3s 5ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 45/50
657/657 [==============================] - 3s 4ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 46/50
657/657 [==============================] - 3s 5ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 47/50
657/657 [==============================] - 4s 5ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 48/50
657/657 [==============================] - 3s 4ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6931 - val_accuracy: 0.7792
Epoch 49/50
657/657 [==============================] - 3s 4ms/step - loss: 0.6931 - accuracy: 0.7786 - val_loss: 0.6930 - val_accuracy: 0.7792
Epoch 50/50
657/657 [==============================] - 3s 4ms/step - loss: 0.6930 - accuracy: 0.7786 - val_loss: 0.6930 - val_accuracy: 0.7792
<keras.src.callbacks.History at 0x7a14a573be20>
```

```
y_test_prob=model.predict(x_test_st)  # Prob
```

```
282/282 [==============================] - 1s 2ms/step
```

```
y_test_prob
```

```
array([[0.49990955],
       [0.4999138 ],
       [0.4999144 ],
       ...,
       [0.49990252],
       [0.4999036 ],
       [0.49991643]], dtype=float32)
```

```
pred_test=np.where(y_test_prob>=.4999199,1,0)
```

```
pred_test
```

```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]])
```

```
metrics.confusion_matrix(y_test,pred_test)
```

```
array([[6267,  746],
       [1723,  264]])
```

```
print(metrics.classification_report(y_test,pred_test))
```

```
                  precision    recall  f1-score   support

               0       0.78      0.89      0.84      7013
               1       0.26      0.13      0.18      1987

        accuracy                           0.73      9000
       macro avg       0.52      0.51      0.51      9000
    weighted avg       0.67      0.73      0.69      9000
```