Default of Credit Card Clients

Group-7:

Akshay Jambhulkar	23PGAI0074
Amrutha C	23PGAI0030
Gajendra Muley	23PGAI0048
Muskan Rath	23PGAI0019
Neha Rupesh Thakur	23PGAI0064
Nency Badiyani	23PGAI0018
Tanuja Tanushree	23PGAI0104

Objective:

The objective of this report is to develop a model that can accurately predict which clients are at risk of defaulting on their credit card payments next month. This will allow the financial institution to take proactive measures to mitigate potential losses and manage risk exposure.

Dataset:

In this report, we utilized a dataset obtained from the UCI repository, specifically the "Default of Credit Clients" dataset. This dataset contains information on 30,000 credit clients and includes 25 different features that were used to train and test our model.

Columns:

```
raw_data.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')
```

The dataset is well-structured, with no missing values and all columns are of integer type.So, no preprocessing required.

Methodology:

1. Proxy variable:

To check for the presence of proxy variables, we employed several methods, including correlation, heatmap, Variance Inflation Factor (VIF) and cosine similarity. The correlation method helped us to identify the linear relationship between the features, while the heatmap helped us to visualize the correlation between the features. VIF is used to check the multicollinearity between the features, while the cosine similarity method helped us to identify the similarity between the features. This allowed us to identify any proxy variables that may be present in the dataset and take appropriate measures to address them. By using these methods, we were able to ensure the quality of our dataset and reduce multicollinearity.

Code:

- Cosine similarity:

```
def circle(result):
        r = 1
        d = 10 * r * (1 - result)
        circle1=plt.Circle((0, 0), r, alpha=.2)
        circle2=plt.Circle((d, 0), r, alpha=.2)
        plt.ylim([-1.1, 1.1])
        plt.xlim([-1.1, 1.1 + d])
        fig = plt.gcf()
        fig.gca().add_artist(circle1)
        fig.gca().add_artist(circle2)
```

```
def cosine_similarity(df,n,lower_limit,upper_limit):
    for i in np.arange(lower_limit,upper_limit):
        result = 1 - spatial.distance.cosine(df[n], df.iloc[:,i])
        print ('cosine distance between {}  and {}'.format(n,df.columns[i]), result)
        circle(result)
```

- VIF:

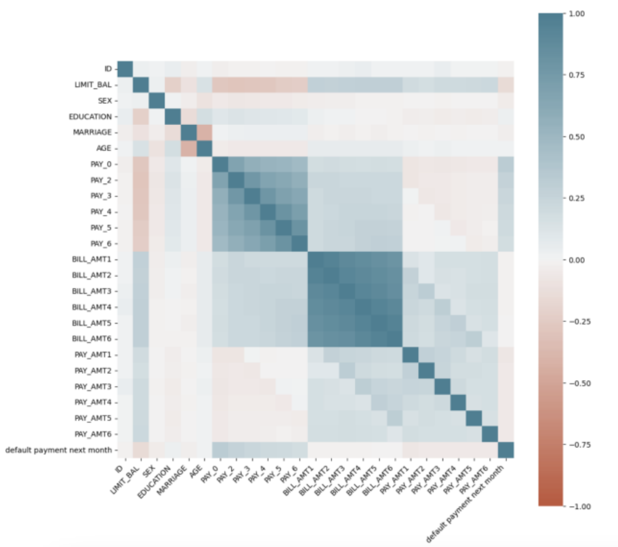
```
def calculate_vif(X,threshold=12.0):
    vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    for i, v in enumerate(vif):
        if v > threshold:
            print(f'VIF for variable {X.columns[i]}: {v}')
```

```
calculate_vif(raw_data)
```

```
VIF for variable BILL_AMT1: 20.844042295880307
VIF for variable BILL_AMT2: 38.22808248204284
VIF for variable BILL_AMT3: 31.78333010024126
VIF for variable BILL_AMT4: 29.69978758865143
VIF for variable BILL_AMT5: 36.07861319493973
VIF for variable BILL_AMT6: 21.4275958783812
```

- Heatmap:

```
def correlation_data(df):
    fig = plt.figure(figsize=(12,12))
    corr = df.corr()
    ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
    )
    ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
    );
    dict_columns={}
    corr_matrix = df.corr(method='pearson',min_periods=2)
    corr_matrix = corr_matrix.mask(np.tril(np.ones(corr_matrix.shape)).astype(np.bool))
    for col in df.columns:
        col_corrs = corr_matrix[col]
        strong_corrs = col_corrs[abs(col_corrs) > 0.75]
        if len(strong_corrs) != 0:
            print(f'Column {col}:')
            print(strong_corrs)
            print()
            dict_columns[col]=[strong_corrs]
    return dict_columns
```



```
Column BILL_AMT2:
BILL_AMT1     0.951484
Name: BILL_AMT2, dtype: float64

Column BILL_AMT3:
BILL_AMT1     0.892279
BILL_AMT2     0.928326
Name: BILL_AMT3, dtype: float64

Column BILL_AMT4:
BILL_AMT1     0.860272
BILL_AMT2     0.892482
BILL_AMT3     0.923969
Name: BILL_AMT4, dtype: float64

Column BILL_AMT5:
BILL_AMT1     0.829779
BILL_AMT2     0.859778
BILL_AMT3     0.883910
BILL_AMT4     0.940134
Name: BILL_AMT5, dtype: float64

Column BILL_AMT6:
BILL_AMT1     0.802650
BILL_AMT2     0.831594
BILL_AMT3     0.853320
BILL_AMT4     0.900941
BILL_AMT5     0.946197
Name: BILL_AMT6, dtype: float64
```

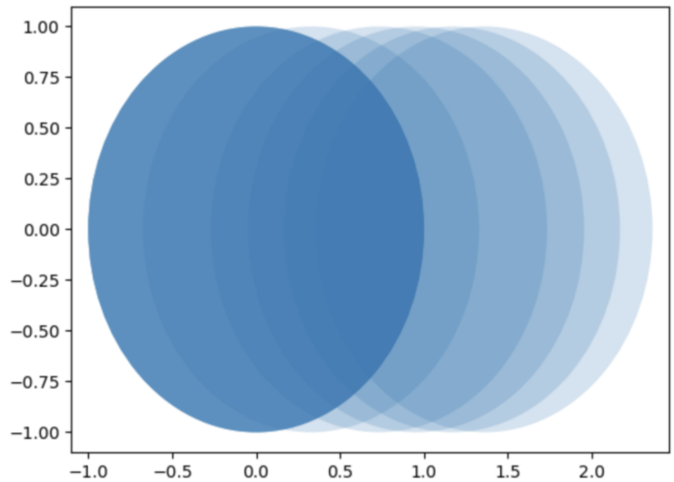
1. Bill_Amount

```
raw_data.iloc[:,12:18].corr(method ='pearson')
```

	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6
BILL_AMT1	1.000000	0.951484	0.892279	0.860272	0.829779	0.802650
BILL_AMT2	0.951484	1.000000	0.928326	0.892482	0.859778	0.831594
BILL_AMT3	0.892279	0.928326	1.000000	0.923969	0.883910	0.853320
BILL_AMT4	0.860272	0.892482	0.923969	1.000000	0.940134	0.900941
BILL_AMT5	0.829779	0.859778	0.883910	0.940134	1.000000	0.946197
BILL_AMT6	0.802650	0.831594	0.853320	0.900941	0.946197	1.000000

```
cosine_similarity(raw_data,'BILL_AMT1',12,18)
```

```
cosine distance between BILL_AMT1  and BILL_AMT1 1
cosine distance between BILL_AMT1  and BILL_AMT2 0.9672288987843305
cosine distance between BILL_AMT1  and BILL_AMT3 0.9267324088262381
cosine distance between BILL_AMT1  and BILL_AMT4 0.904693255417323
cosine distance between BILL_AMT1  and BILL_AMT5 0.8832902750921946
cosine distance between BILL_AMT1  and BILL_AMT6 0.8639006965838157
```



By observing the results of these methods, we found that the feature **"bill_amt1" can be used as a proxy variable** for "bill_amt2", "bill_amt3", "bill_amt4”, "bill_amt5” and "bill_amt6”. This means that "bill_amt1" has a strong linear relationship and high similarity with "bill_amt2", "bill_amt3", "bill_amt4”, "bill_amt5” and "bill_amt6”. Therefore, including all six features in the model may lead to multicollinearity and have a negative impact on the model's performance. To address this issue, we could remove one of the correlated variables and keep only "bill_amt1" in the model.

2. Differential privacy on "LIMIT_BAL":

We took the necessary steps to protect the clients' sensitive information by applying differential privacy techniques on the **"LIMIT_BAL"** feature. We recognize that this feature, which represents the limit of credit balance for each client, is considered sensitive information and its revelation can lead to identification and prediction of other details.

We employed a technique of adding noise to the raw data using different values of epsilon. It was found that a value of **epsilon=0.5** produced the best results. The mean of both the raw data and the differentially private data was calculated and it was found that there is no significant difference in the mean values of both datasets.

```
epsilon =0.5
dp_LIMIT_BAL=[]

original = raw_data['LIMIT_BAL']

for i in range(0,30000):
        value=original[i] +  np.random.laplace(loc=0, scale=sensitivity/epsilon)
        value = round(value)
        dp_LIMIT_BAL.append(value)
dp_limit_bal['LIMIT_BAL']=dp_LIMIT_BAL
```

```
: raw_data['LIMIT_BAL']

: 0         20000
  1        120000
  2         90000
  3         50000
  4         50000
            ...
  29995    220000
  29996    150000
  29997     30000
  29998     80000
  29999     50000
  Name: LIMIT_BAL, Length: 30000, dtype: int64
```

```
]: dp_limit_bal['LIMIT_BAL']

]: 0         20037
   1        120159
   2         89950
   3         50009
   4         49944
             ...
   29995    220015
   29996    149771
   29997     29985
   29998     80203
   29999     49876
   Name: LIMIT_BAL, Length: 30000, dtype: int64
```

Statistical significance:

```
raw_data.describe()['LIMIT_BAL']

count       30000.000000
mean       167484.322667
std        129747.661567
min         10000.000000
25%         50000.000000
50%        140000.000000
75%        240000.000000
max       1000000.000000
Name: LIMIT_BAL, dtype: float64
```

```
dp_limit_bal.describe()['LIMIT_BAL']

count     30000.000000
mean     167484.090967
std      129747.262429
min        9558.000000
25%       50155.000000
50%      140020.000000
75%      239968.000000
max      999915.000000
Name: LIMIT_BAL, dtype: float64
```