Introduction to Data Science (S2-22_DSECLZG532)-ASSIGNMENT

Group No

33

Group Member Names:

1. Prashant Kumar: 2022DC04013@wilp.bits-pilani.ac.in
2. Rahul Kumar: 2022DC04053@wilp.bits-pilani.ac.in
3. Ramesh Kumar: 2022DA04563@wilp.bits-pilani.ac.in

1. Business Understanding

Students are expected to identify an analytical problem of your choice. You have to detail the Business Understanding part of your problem under this heading which basically addresses the following questions.

1. What is the business problem that you are trying to solve?
2. What data do you need to answer the above problem?
3. What are the different sources of data?
4. What kind of analytics task are you performing?

Score: 1 Mark in total (0.25 mark each)

--------------Type the answers below this line--------------

1. Our objective is to anticipate instances of payment defaults within a financial institution. We are utilizing a dataset obtained from a bank in Taiwan, made accessible through UCI_Credit_Card.

   Our **central goal** is **to construct a predictive model with the capability to assess the probability of customers defaulting on their payments in a monthly cycle**.

   This predictive proficiency will aid the bank in segregating customers into two categories: those considered dependable and those perceived as less reliable concerning payments. Through this classification, the bank can take proactive actions to minimize potential losses and sustain its financial stability.

1. In order to enhance the precision of our model, we necessitate particular customer details including age, educational history, marital status, assets, existing loans, income range, dependents, credit limit, expenditure and loan repayment tendencies, spending trends, payment behaviors, and pertinent variables. While the existing dataset encompasses a portion of these particulars, we are of the opinion that the integration of supplementary data facets will amplify the prognostic potential of our model. Notwithstanding, we hold the conviction that the prevailing dataset encompasses ample information to proficiently foresee customer credibility.

1. We've sourced our dataset from the UCI website (https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients) for our analysis. The dataset provides the following information:

The dataset encompasses 25 variables:

- **ID**: Client identification
- **LIMIT_BAL**: Credit amount in NT dollars (inclusive of both individual and family/supplementary credit)
- **SEX**: Gender (1=male, 2=female)
- **EDUCATION**: Educational background (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
- **MARRIAGE**: Marital status (1=married, 2=single, 3=others)
- **AGE**: Age in years
- **PAY_0**: Payment status in September, 2005 (-1=paying duly, 1=one-month payment delay, 2=two-month payment delay, ... 8=eight-month payment delay, 9=nine months and above payment delay)
- **PAY_2** - **PAY_6**: Payment status in August, July, June, May, and April 2005, respectively (using the same scale as above)
- **BILL_AMT1**: Bill statement amount in September, 2005 (in NT dollars)
- **BILL_AMT2** - **BILL_AMT6**: Bill statement amount in August, July, June, May, and April 2005, respectively (in NT dollars)
- **PAY_AMT1**: Previous payment amount in September, 2005 (in NT dollars)
- **PAY_AMT2** - **PAY_AMT6**: Previous payment amount in August, July, June, May, and April 2005, respectively (in NT dollars)
- **default payment next month**: Default payment (1=yes, 0=no)

1. Our analysis delves into an intricate examination of multiple attributes and their respective distributions across each feature. Our primary focus lies in investigating the correlation between demographic characteristics and the extent of credit extended to clients. Moreover, we are dedicated to scrutinizing the interconnections among diverse features, their distribution patterns, and their overarching significance in forecasting the target attribute. Through this comprehensive analysis, our objective is to pinpoint noteworthy variables and forge a resilient predictive model that precisely anticipates the probability of payment defaults.

2. Data Acquisition

For the problem identified , find an appropriate data set (Your data set must be unique with minimum **20 features and 10k rows**) from any public data source.

2.1 Download the data directly

In [2]:
```
##---------Type the code below this line------------------##

# Importing all libraries here

import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20cre

original_df = pd.read_excel(url) #to keep a separate copy of the dataframe
df = original_df.copy() #the dataframe on which the analysis would be done
df
```

Out[2]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X
0	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AM
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	34
3	3	90000	2	2	2	34	0	0	0	0	...	14331	149
4	4	50000	2	2	1	37	0	0	0	0	...	28314	289
...	...	...	...	...	...	...	...	...	...	...	...	...	
29996	29996	220000	1	3	1	39	0	0	0	0	...	88004	312
29997	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	51
29998	29998	30000	1	2	2	37	4	3	2	-1	...	20878	205
29999	29999	80000	1	3	1	41	1	-1	0	0	...	52774	118
30000	30000	50000	1	2	1	46	0	0	0	0	...	36535	324

30001 rows × 25 columns

Inference

1. Data downloaded directly from the UCI website as mentioned earlier.
2. It has 30k rows and 25 columns.
3. We need to promote row 0 as the index.

2.2 Code for converting the above downloaded data into a dataframe

In [3]:

```
##---------Type the code below this line------------------##
df.columns = df.iloc[0] #Promoting the index = 0 as headers using iloc function of pandas.
df = df.drop(df.index[0])
df
```

Out[3]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_A
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_A
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	
3	3	90000	2	2	2	34	0	0	0	0	...	14331	1
4	4	50000	2	2	1	37	0	0	0	0	...	28314	2
5	5	50000	1	2	1	57	-1	0	-1	0	...	20940	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
29996	29996	220000	1	3	1	39	0	0	0	0	...	88004	3
29997	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	
29998	29998	30000	1	2	2	37	4	3	2	-1	...	20878	2
29999	29999	80000	1	3	1	41	1	-1	0	0	...	52774	1
30000	30000	50000	1	2	1	46	0	0	0	0	...	36535	3

30000 rows × 25 columns

Inference

1. The above step is one time run, if we run it again (without loading the original data, the first row (which is actual data) would be promoted as headers.
2. The dataset is converted into dataframe with proper headers, which would be further useful for analysis

2.3 Confirm the data has been correctly by displaying the first 5 and last 5 records.

In [4]:

```
##---------Type the code below this line------------------##
# displaying the first 5 row of the data
df.head()
```

Out[4]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	B
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	
3	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	
4	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	
5	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	

5 rows × 25 columns

In [5]:

```
# displaying the last 5 row of the data
df.tail()
```

Out[5]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_
29996	29996	220000	1	3	1	39	0	0	0	0	...	88004	[illegible]
29997	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	
29998	29998	30000	1	2	2	37	4	3	2	-1	...	20878	[illegible]
29999	29999	80000	1	3	1	41	1	-1	0	0	...	52774	[illegible]
30000	30000	50000	1	2	1	46	0	0	0	0	...	36535	[illegible]

5 rows × 25 columns

2.4 Display the column headings, statistical information, description and statistical summary of the data.

In [6]:

```
##---------Type the code below this line------------------##

print(f'Columns headings for dataset are: \n{list(df.columns)}')
```

```
Columns headings for dataset are:
['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', 'B
ILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'defaul
t payment next month']
```

Inference

1. The dataset comprises a total of 25 attributes or features.
2. Our target variable is 'default payment next month'.

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 1 to 30000
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   ID                          30000 non-null  object
 1   LIMIT_BAL                   30000 non-null  object
 2   SEX                         30000 non-null  object
 3   EDUCATION                   30000 non-null  object
 4   MARRIAGE                    30000 non-null  object
 5   AGE                         30000 non-null  object
 6   PAY_0                       30000 non-null  object
 7   PAY_2                       30000 non-null  object
 8   PAY_3                       30000 non-null  object
 9   PAY_4                       30000 non-null  object
 10  PAY_5                       30000 non-null  object
 11  PAY_6                       30000 non-null  object
 12  BILL_AMT1                   30000 non-null  object
 13  BILL_AMT2                   30000 non-null  object
 14  BILL_AMT3                   30000 non-null  object
 15  BILL_AMT4                   30000 non-null  object
 16  BILL_AMT5                   30000 non-null  object
 17  BILL_AMT6                   30000 non-null  object
 18  PAY_AMT1                    30000 non-null  object
 19  PAY_AMT2                    30000 non-null  object
```

```
 20   PAY_AMT3                   30000 non-null  object
 21   PAY_AMT4                   30000 non-null  object
 22   PAY_AMT5                   30000 non-null  object
 23   PAY_AMT6                   30000 non-null  object
 24   default payment next month  30000 non-null  object
dtypes: object(25)
memory usage: 6.0+ MB
```

Inference

It is evident from this observation that the data types of the attributes are classified as 'object'. Despite this classification, the values contained within these attributes are in integral form. To facilitate our statistical comprehension, it is necessary to transform these attributes into either floating-point or integral format.

In [8]:

```
#converting data types to float for further statistical studies
df = df.convert_dtypes(float)
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
ID	30000.0	15000.500000	8660.398374	1.0	7500.75	15000.5	22500.25	30000.0
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	1000000.0
SEX	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	2.0
EDUCATION	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	6.0
MARRIAGE	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	3.0
AGE	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	79.0
PAY_0	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	8.0
PAY_2	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	8.0
PAY_3	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	8.0
PAY_4	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	8.0
PAY_5	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	8.0
PAY_6	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	8.0
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	964511.0
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	983931.0
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	1664089.0
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	891586.0
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	927171.0
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	961664.0
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	873552.0
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	1684259.0
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	896040.0
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	621000.0
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	426529.0
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	528666.0

	count	mean	std	min	25%	50%	75%	max
default payment next month	30000.0	0.221200	0.415062	0.0	0.00	0.0	0.00	1.0

Inference

Regarding Data Preprocessing:

1. We observed that the 'EDUCATION' feature contains values such as 0, 5, and 6. These values are not explicitly mentioned with clear definitions in the original problem statement.
2. Similarly, the 'MARRIAGE' feature includes an unassigned value of 0, which is not explicitly explained in the problem statement.
3. To maintain consistency and accuracy in our analysis, we've decided to preprocess this data.
4. Our preprocessing aims to assign appropriate labels to the 'EDUCATION' and 'MARRIAGE' features before moving forward with building our predictive model.
5. This preprocessing step is crucial to ensure that our data is correctly interpreted and to prevent any potential misleading or erroneous results.

Regarding Attribute Renaming:

1. Within our dataset, the term 'PAY_0' signifies the repayment status for September 2005.
2. This 'PAY_0' attribute directly corresponds with the 'BILL_AMT_1' attribute.
3. Hence, we have chosen to rename 'PAY_0' as 'PAY_1' - which will help us in better clarity and consistency throughout the analysis
4. Moreover, change the 'default payment next month' attribute with a shorter name.
5. This renaming adjustment is intended to streamline future interactions with the dataset and enhance its overall readability.

In [9]:
```
#renaming PAY_0 to PAY_1
df.rename(columns ={'PAY_0':'PAY_1'}, inplace =True)
# renaming default payment next month to def_payment
df.rename(columns ={'default payment next month':'def_payment'}, inplace =True)
```

In [10]:
```
df['MARRIAGE'].describe()
```

Out[10]:
```
count    30000.000000
mean         1.551867
std          0.521970
min          0.000000
25%          1.000000
50%          2.000000
75%          2.000000
max          3.000000
Name: MARRIAGE, dtype: float64
```

Code to find out answer for Section 2.5

In [11]:
```
print('shape of the data set: {}'.format(df.shape))
```

```
shape of the data set: (30000, 25)
```

In [12]:
```
print('statistical infomation about SEX, EDUCATION and MARRIAGE: \n{}'
      .format(df[['SEX', 'EDUCATION', 'MARRIAGE', 'AGE']].describe()))
print('----------------------------------------------------------------------')
```

```python
print('Statistical information about PAY attribute: \n{}'
      .format(df[['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']].describe()))
print('-------------------------------------------------------------------------')
print('Statistical information about BILL_AMT attribute: \n{}'
      .format(df[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_A
              .describe()))
```

```
statistical infomation about SEX, EDUCATION and MARRIAGE:
                SEX     EDUCATION      MARRIAGE           AGE
count  30000.000000  30000.000000  30000.000000  30000.000000
mean       1.603733      1.853133      1.551867     35.485500
std        0.489129      0.790349      0.521970      9.217904
min        1.000000      0.000000      0.000000     21.000000
25%        1.000000      1.000000      1.000000     28.000000
50%        2.000000      2.000000      2.000000     34.000000
75%        2.000000      2.000000      2.000000     41.000000
max        2.000000      6.000000      3.000000     79.000000
-------------------------------------------------------------------------
Statistical information about PAY attribute: 
              PAY_1         PAY_2         PAY_3         PAY_4         PAY_5  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000   
mean      -0.016700     -0.133767     -0.166200     -0.220667     -0.266200   
std        1.123802      1.197186      1.196868      1.169139      1.133187   
min       -2.000000     -2.000000     -2.000000     -2.000000     -2.000000   
25%       -1.000000     -1.000000     -1.000000     -1.000000     -1.000000   
50%        0.000000      0.000000      0.000000      0.000000      0.000000   
75%        0.000000      0.000000      0.000000      0.000000      0.000000   
max        8.000000      8.000000      8.000000      8.000000      8.000000   

              PAY_6  
count  30000.000000  
mean      -0.291100  
std        1.149988  
min       -2.000000  
25%       -1.000000  
50%        0.000000  
75%        0.000000  
max        8.000000  
-------------------------------------------------------------------------
Statistical information about BILL_AMT attribute: 
           BILL_AMT1      BILL_AMT2     BILL_AMT3      BILL_AMT4  \
count   30000.000000   30000.000000  3.000000e+04   30000.000000   
mean    51223.330900   49179.075167  4.701315e+04   43262.948967   
std     73635.860576   71173.768783  6.934939e+04   64332.856134   
min   -165580.000000  -69777.000000 -1.572640e+05 -170000.000000   
25%      3558.750000    2984.750000  2.666250e+03    2326.750000   
50%     22381.500000   21200.000000  2.008850e+04   19052.000000   
75%     67091.000000   64006.250000  6.016475e+04   54506.000000   
max    964511.000000  983931.000000  1.664089e+06  891586.000000   

           BILL_AMT5      BILL_AMT6  
count   30000.000000   30000.000000  
mean    40311.400967   38871.760400  
std     60797.155770   59554.107537  
min    -81334.000000 -339603.000000  
25%      1763.000000    1256.000000  
50%     18104.500000   17071.000000  
75%     50190.500000   49198.250000  
max    927171.000000  961664.000000  
```

In [13]:

```python
df.isna().sum()
```

Out[13]:

```
ID           0
LIMIT_BAL    0
SEX          0
```

```
EDUCATION      0
MARRIAGE       0
AGE            0
PAY_1          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
def_payment    0
dtype: int64
```

2.5 Write your observations from the above.

1. Size of the dataset
2. What type of data attributes are there?
3. Is there any null data that has to be cleaned?

Score: 2 Marks in total (0.25 marks for 2.1, 0.25 marks for 2.2, 0.5 marks for 2.3, 0.25 marks for 2.4, 0.75 marks for 2.5)

--------------Type the answers below this line--------------

1. Our dataset is made up of **30000** rows and **25** columns, each representing a unique piece of information.

2. The data attributes primarily consist of integers, contributing to the numerical foundation of our dataset.

3. While our dataset shows no signs of missing values, a few peculiar aspects deserve attention:

   - In the 'EDUCATION' category, we've encountered the presence of categories 5 and 6, which currently lack proper labeling. Additionally, the category 0 remains unspecified.

   - The 'MARRIAGE' category includes an unexplained label, 0, which requires further clarity.

   - Upon closer examination of the 'PAY_AMT' attribute, it's apparent that there's an unfamiliar label, -2. To enhance comprehensibility, we plan to designate 0 as 'paid on time' and treat all negative values as zeros. However, this adjustment will be addressed in due course.

   - Similar to the 'PAY_AMT', the 'BILL_AMT' attribute also contains an unfamiliar label, -2. In line with our approach, we'll consider 0 as 'paid on time' and treat any negative values as zeros. But this rectification will be tackled later in our process.

3. Data Preparation

If input data is numerical or categorical, do 3.1, 3.2 and 3.4 If input data is text, do 3.3 and 3.4

3.1 Check for

- duplicate data
- missing data
- data inconsistencies

In [14]:
```
##---------Type the code below this line------------------##
print('Duplicate value in LIMIT_BAL: {}'.format(df['LIMIT_BAL'].is_unique))
print('Duplicate value in SEX: {}'.format(df['SEX'].is_unique))
print('Duplicate value in EDUCATION: {}'.format(df['EDUCATION'].is_unique))
print('Duplicate value in MARRIAGE: {}'.format(df['MARRIAGE'].is_unique))
print('Duplicate value in AGE: {}'.format(df['AGE'].is_unique))
```

```
Duplicate value in LIMIT_BAL: False
Duplicate value in SEX: False
Duplicate value in EDUCATION: False
Duplicate value in MARRIAGE: False
Duplicate value in AGE: False
```

Inference

There are no duplicate values in dataframe for the categorical values

In [15]:
```
df.isna().sum()
```

Out[15]:
```
ID             0
LIMIT_BAL      0
SEX            0
EDUCATION      0
MARRIAGE       0
AGE            0
PAY_1          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
def_payment    0
dtype: int64
```

Inference

There are no missing data in the dataset

In [16]:
```
df.describe().T
```

Out[16]:

	count	mean	std	min	25%	50%	75%	max
ID	30000.0	15000.500000	8660.398374	1.0	7500.75	15000.5	22500.25	30000.0

	count	mean	std	min	25%	50%	75%	max
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	1000000.0
SEX	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	2.0
EDUCATION	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	6.0
MARRIAGE	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	3.0
AGE	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	79.0
PAY_1	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	8.0
PAY_2	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	8.0
PAY_3	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	8.0
PAY_4	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	8.0
PAY_5	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	8.0
PAY_6	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	8.0
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	964511.0
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	983931.0
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	1664089.0
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	891586.0
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	927171.0
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	961664.0
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	873552.0
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	1684259.0
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	896040.0
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	621000.0
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	426529.0
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	528666.0
def_payment	30000.0	0.221200	0.415062	0.0	0.00	0.0	0.00	1.0

In [17]:

```
print('value count of EDUCATION: \n{}'.format(df['EDUCATION'].value_counts()))
print('----------------------------------------------------------------------')
print('value count of MARRIAGE: \n{}'.format(df['MARRIAGE'].value_counts()))
```

```
value count of EDUCATION:
2    14030
1    10585
3     4917
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: Int64
----------------------------------------------------------------------
value count of MARRIAGE:
2    15964
1    13659
3      323
```

```
0        54
Name: MARRIAGE, dtype: Int64
```

Inferences

1. There was a data inconsistency, where data types was object for integeral values, this has been solved above in the section 2.4
2. EDUCATION has category 5 and 6 that are unlabelled, moreover the category 0 is undocumented. (To be addressed subsequently)
3. MARRIAGE has a label 0 that is undocumented.(To be addressed subsequently)

3.2 Apply techiniques

- to remove duplicate data
- to impute or remove missing data
- to remove data inconsistencies

In [18]:
```
##---------Type the code below this line------------------##
```

Inferences

- Duplicate Data: There are no instances of duplicate data - Analysis performed in Sec 3.1
- Missing Data: Similar to the above, no instances of missing data - Analysis performed in Sec 3.1
- Inconsistencies: We changed the data type 'object' to 'float' in Sec 2.4

While carefully examining the earlier mentioned code, we noticed that the EDUCATION attribute had some values (0, 5, and 6) that didn't have clear labels in the dataset's description. To fix this, we decided to put these values into a broader category called 'Others,' which is represented by the number 4 as suggested by the problem's description. This helps us keep our analysis clear and consistent.

In [19]:
```
df.loc[:,'EDUCATION'] = df.loc[:,'EDUCATION'].replace(0,4)
df.loc[:,'EDUCATION'] = df.loc[:,'EDUCATION'].replace(5,4)
df.loc[:,'EDUCATION'] = df.loc[:,'EDUCATION'].replace(6,4)
```

In [20]:
```
print('value count of EDUCATION after operation: \n{}'.format(df['EDUCATION'].value_counts
```

```
value count of EDUCATION after operation:
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: Int64
```

Checking the values for MARRIAGE Column

In [21]:
```
print('value count of MARRIAGE: \n{}'.format(df['MARRIAGE'].value_counts()))
```

```
value count of MARRIAGE:
2    15964
1    13659
3      323
0       54
Name: MARRIAGE, dtype: Int64
```

As we can see above, the MARRIAGE attribute contains an unlabelled value (0) which is not specified in the dataset description. To address this matter, we've chosen to categorize this value under 'divorced' (denoted by the number 3), considering the relatively low frequency of instances associated with this value. Our decision aligns with our interpretation of the problem description.

In [22]:
```
df.loc[:,'MARRIAGE'] = df.loc[:,'MARRIAGE'].replace(0,3)
```

In [23]:
```
print('value count of MARRIAGE after operation: \n{}'.format(df['MARRIAGE'].value_counts()
```

```
value count of MARRIAGE after operation:
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: Int64
```

3.3 Encode categorical data

In [24]:
```
##---------Type the code below this line------------------##
```

This step is not required as there are no categorical data in the dataset

3.4 Text data

1. Remove special characters
2. Change the case (up-casing and down-casing).
3. Tokenization — process of discretizing words within a document.
4. Filter Stop Words.

In [25]:
```
##---------Type the code below this line------------------##
```

In [26]:
```
##---------Type the code below this line------------------##
```

1. Given that the dataset exclusively contains integral values and lacks any text data, there is no requirement to undertake tasks like eliminating special characters, altering case, tokenizing, or filtering.
2. Hence, we have skipped these steps for this project

3.4 Report

Mention and justify the method adopted

- to remove duplicate data, if present
- to impute or remove missing data, if present
- to remove data inconsistencies, if present

OR for textdata

- How many tokens after step 3?
- how may tokens after stop words filtering?

If the any of the above are not present, then also add in the report below.

Score: 2 Marks (based on the dataset you have, the data prepreation you had to do and report typed, marks will be distributed between 3.1, 3.2, 3.3 and 3.4)

In [27]:

```
##---------Type the code below this line------------------##
```

In [28]:

```
##---------Type the code below this line------------------##
```

Inferences

1. No duplicate data in the dataset.
2. No missing data in the dataset, hence we didn't apply any data imputation techniques
3. We did find some inconsistenceis with respect to data types and some values were not matching with given definition in the problem statement. We addressed those in earlier sections.

3.5 Identify the target variables.

- Separate the data from the target such that the dataset is in the form of (X,y) or (Features, Label)
- Discretize / Encode the target variable or perform one-hot encoding on the target or any other as and if required.
- Report the observations

Score: 1 Mark

In [29]:

```
##---------Type the code below this line------------------##
# Importing the necessary library

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.metrics import f1_score, precision_score, recall_score, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.feature_selection import mutual_info_classif
```

In [30]:

```
y = df['def_payment'].copy().astype(float)
print(f'data types of target variable, y: {y.dtype}')
print(f'target variable y: \n{y.sample(5)}')
```

```
data types of target variable, y: float64
target variable y:
18800    0.0
26097    0.0
14831    0.0
24163    0.0
11264    0.0
Name: def_payment, dtype: float64
```

Inferences

1. def_payment is the target variable in this dataset
2. The values of def_payment are binary, 1 ==defaulter and 0 == non-defaulters

In [31]:
```
# create the features, which now will be everything in the original df
features = ['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1', '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']
X = df[features].copy()
```

In [32]:
```
print(f'features columns, X: {X.columns}')
```

```
features columns, X: Index(['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1',
'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'],
      dtype='object')
```

Inferences

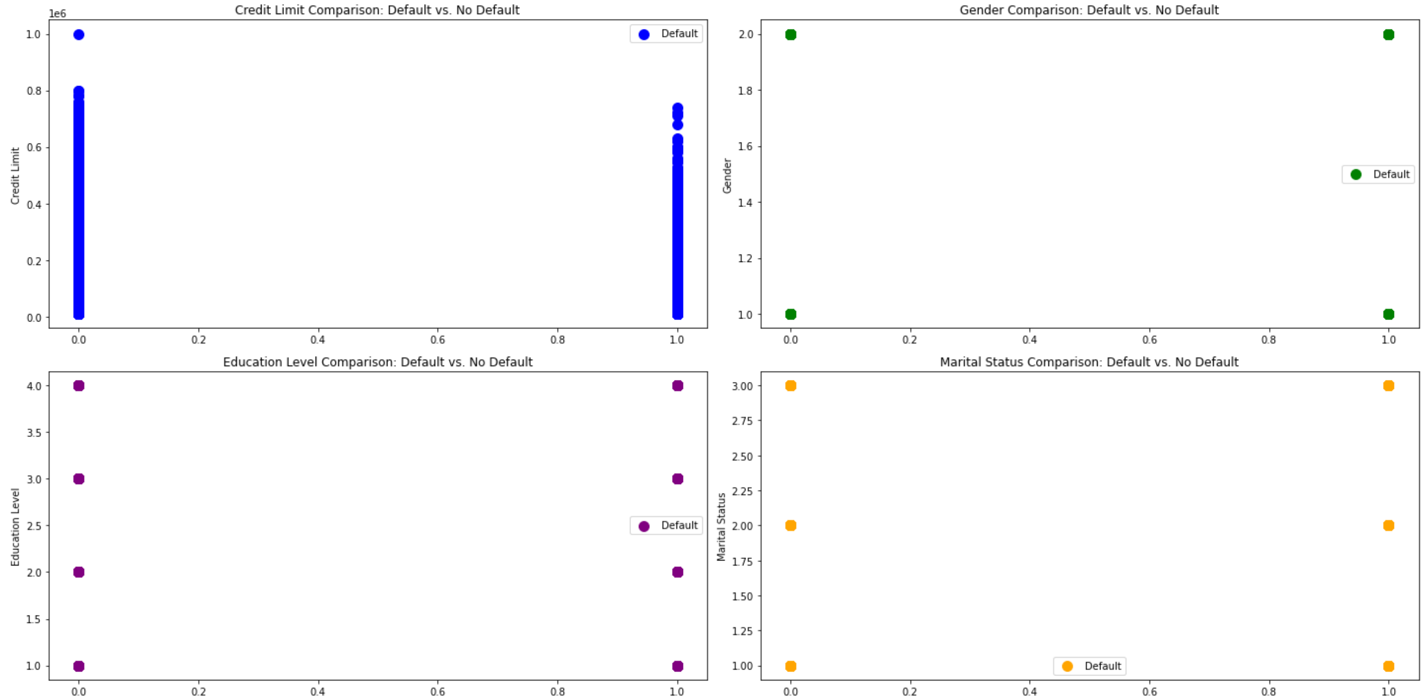
1. Target variable 'def_payment' is not part of feature variable
2. The 'ID' attribute from our feature variables is removed as it does not contribute to our analysis.
3. All other relevant features have been retained for our analysis.

4. Data Exploration using various plots

4.1 Scatter plot of each quantitative attribute with the target.

Score: 1 Mark

In [33]:
```
##---------Type the code below this line------------------##
plt.subplots(figsize=(20, 10))

plt.subplot(221)
plt.scatter(x=df.def_payment, y=df.LIMIT_BAL, c='blue', marker='o', s=100)
plt.ylabel('Credit Limit')
plt.legend(['Default', 'No Default'])
plt.title('Credit Limit Comparison: Default vs. No Default')

plt.subplot(222)
plt.scatter(x=df.def_payment, y=df.SEX, c='green', marker='o', s=100)
plt.ylabel('Gender')
plt.legend(['Default', 'No Default'])
plt.title("Gender Comparison: Default vs. No Default")

plt.subplot(223)
plt.scatter(x=df.def_payment, y=df.EDUCATION, c='purple', marker='o', s=100)
plt.ylabel('Education Level')
plt.legend(['Default', 'No Default'])
plt.title('Education Level Comparison: Default vs. No Default')

plt.subplot(224)
plt.scatter(x=df.def_payment, y=df.MARRIAGE, c='orange', marker='o', s=100)
plt.ylabel('Marital Status')
plt.legend(['Default', 'No Default'])
plt.title('Marital Status Comparison: Default vs. No Default')

plt.tight_layout()
plt.show()
```



Inferences

- Seeing the above scatter plots, its tough to analyze the data. We are not able to understand anything out of it.
- This is happening because our target variable is binary, hence in this case, scatter plots may not be the effective way to analyze the data
- We will generate distribution plots to show the relationship between each variable and the target variable
- Distribution plots would helps us in getting a clearer picture - to understand how each attribute is related to the target variable - further helping in analyzing the data

In [34]:

```
df
```

Out[34]:

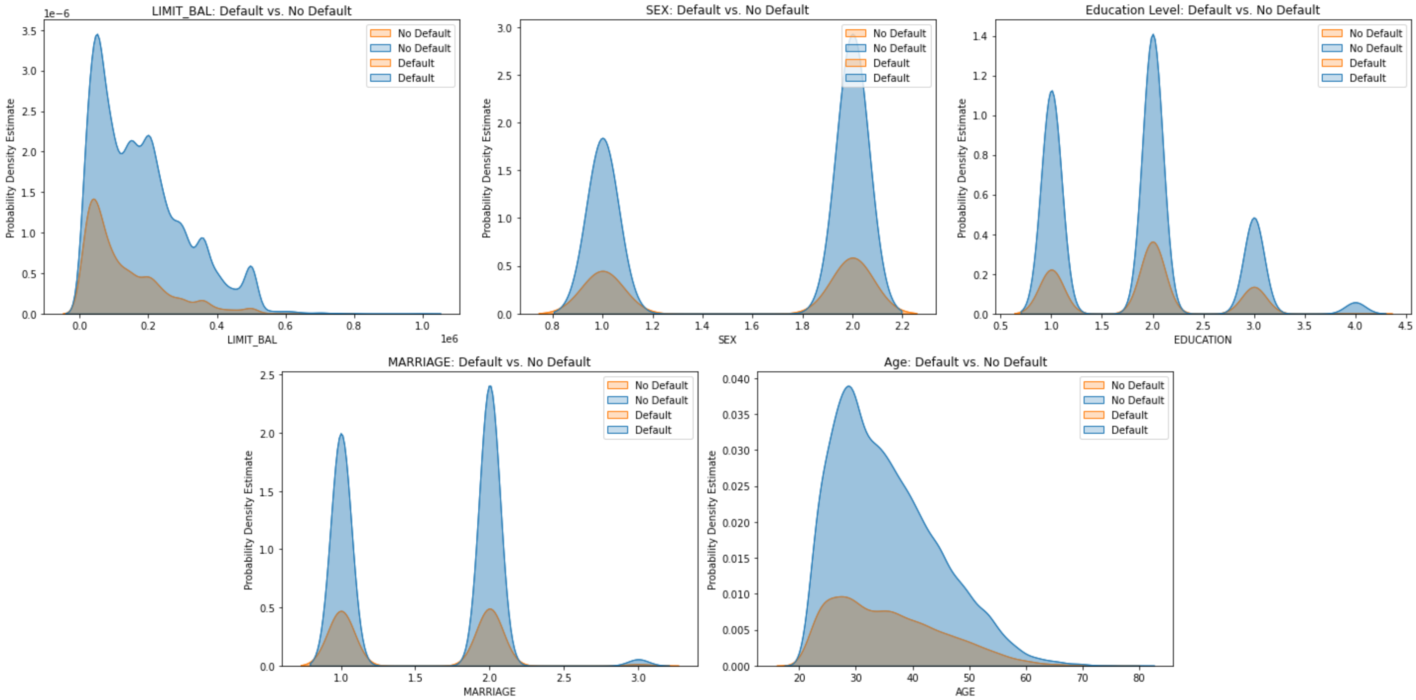
	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_A
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	
3	3	90000	2	2	2	34	0	0	0	0	...	14331	
4	4	50000	2	2	1	37	0	0	0	0	...	28314	
5	5	50000	1	2	1	57	-1	0	-1	0	...	20940	
...	...	...	...	...	...	...	...	...	...	...	...	...	
29996	29996	220000	1	3	1	39	0	0	0	0	...	88004	
29997	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	
29998	29998	30000	1	2	2	37	4	3	2	-1	...	20878	
29999	29999	80000	1	3	1	41	1	-1	0	0	...	52774	
30000	30000	50000	1	2	1	46	0	0	0	0	...	36535	

30000 rows × 25 columns

In [35]:

```
plt.subplots(figsize=(20, 10))
```

```python
plt.subplot2grid(shape=(2, 6), loc=(0, 0), colspan=2)
sns.kdeplot(data=df, x='LIMIT_BAL', hue='def_payment',
            label='No Default', shade=True, color='blue')
sns.kdeplot(data=df, x='LIMIT_BAL', hue='def_payment',
            label='Default', shade=True, color='orange')
plt.ylabel('Probability Density Estimate')
plt.legend()
plt.title('LIMIT_BAL: Default vs. No Default')

ax2 = plt.subplot2grid((2, 6), (0, 2), colspan=2)
sns.kdeplot(data=df, x='SEX', hue='def_payment',
            label='No Default', shade=True, color='green')
sns.kdeplot(data=df, x='SEX', hue='def_payment',
            label='Default', shade=True, color='red')
plt.ylabel('Probability Density Estimate')
plt.legend()
plt.title("SEX: Default vs. No Default")

ax2 = plt.subplot2grid((2, 6), (0, 4), colspan=2)
sns.kdeplot(data=df, x='EDUCATION', hue='def_payment',
            label='No Default', shade=True, color='purple')
sns.kdeplot(data=df, x='EDUCATION', hue='def_payment',
            label='Default', shade=True, color='pink')
plt.ylabel('Probability Density Estimate')
plt.legend()
plt.title('Education Level: Default vs. No Default')

ax2 = plt.subplot2grid((2, 6), (1, 1), colspan=2)
sns.kdeplot(data=df, x='MARRIAGE', hue='def_payment',
            label='No Default', shade=True, color='cyan')
sns.kdeplot(data=df, x='MARRIAGE', hue='def_payment',
            label='Default', shade=True, color='gray')
plt.ylabel('Probability Density Estimate')
plt.legend()
plt.title('MARRIAGE: Default vs. No Default')

ax2 = plt.subplot2grid((2, 6), (1, 3), colspan=2)
sns.kdeplot(data=df, x='AGE', hue='def_payment',
            label='No Default', shade=True, color='brown')
sns.kdeplot(data=df, x='AGE', hue='def_payment',
            label='Default', shade=True, color='olive')
plt.ylabel('Probability Density Estimate')
plt.legend()
plt.title('Age: Default vs. No Default')

plt.tight_layout()

plt.show()
```



Inferences

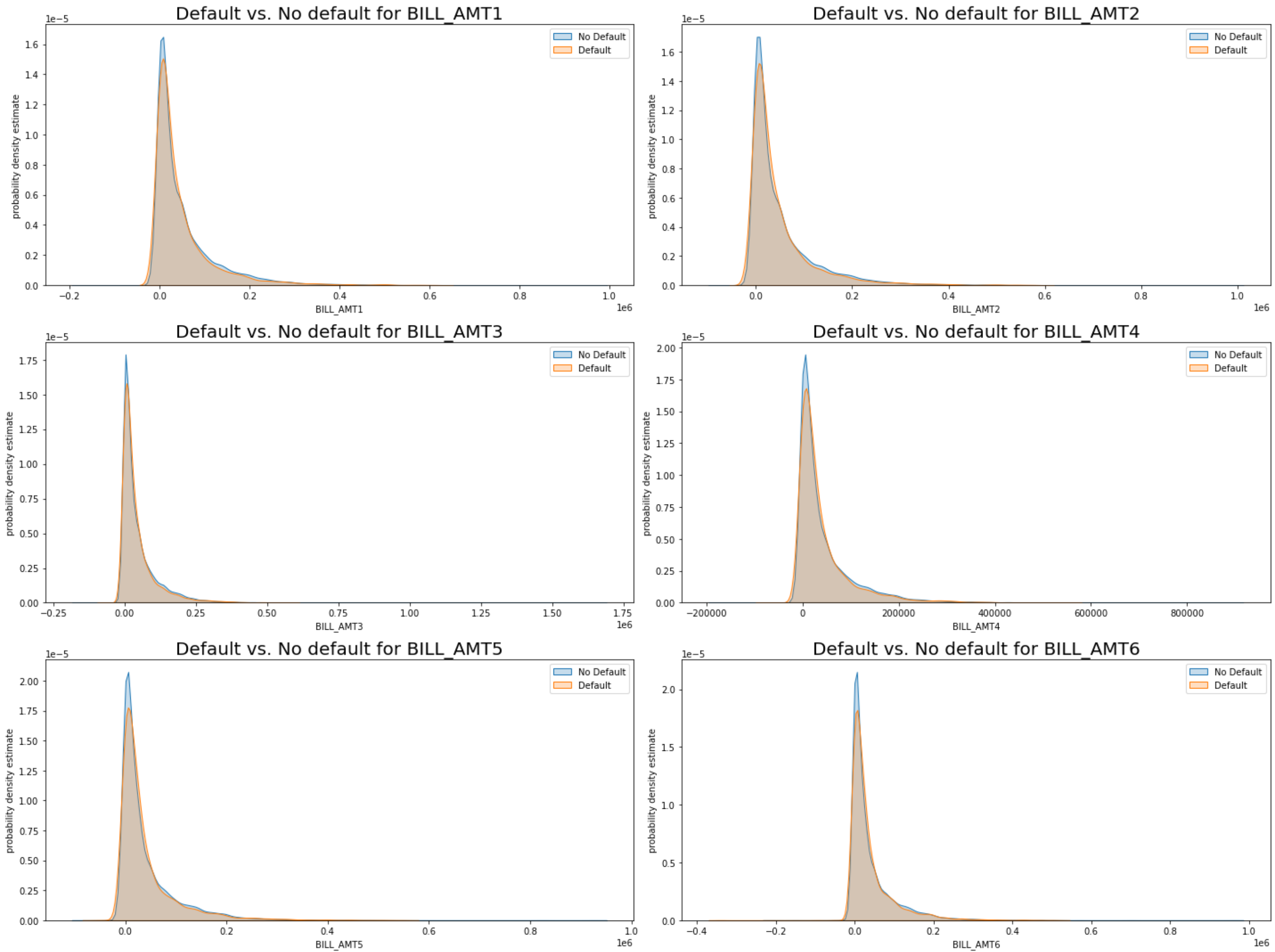
1. 'LIMIT_BAL' and 'AGE' are skewed a. LIMIT_BAL: Most of the clients have a credit limit between 0-20000 unit b. AGE: They fall between primarily between 20-40 years of age c. Conclusion: Most clients are from the middle-aged group
2. SEX: We see that there are high volume of Female Customers - further analysis to be done later - to understand the probability of defaulter among men and women

3. High volume of customers holding university degree, less volume of divorced customers as compared to single and married.

4. For all of the above groups, we will do deeper analysis in later sections to understand the impact on default probability.

In [36]:

```
#plotting some of the dist plots to understand data better
bill_amtx_fts = ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_A
plt.figure(figsize=(20,15))

for i,col in enumerate(bill_amtx_fts):
    plt.subplot(3,2,i + 1)
    sns.kdeplot(df.loc[(df['def_payment'] == 0), col],
                label = 'No Default', shade = True)
    sns.kdeplot(df.loc[(df['def_payment'] == 1), col],
                label = 'Default', shade = True)
    plt.ylabel('probability density estimate')
    plt.legend()
    plt.tight_layout()
    plt.title('Default vs. No default for {}'.format(col), size =20)

plt.show()
```

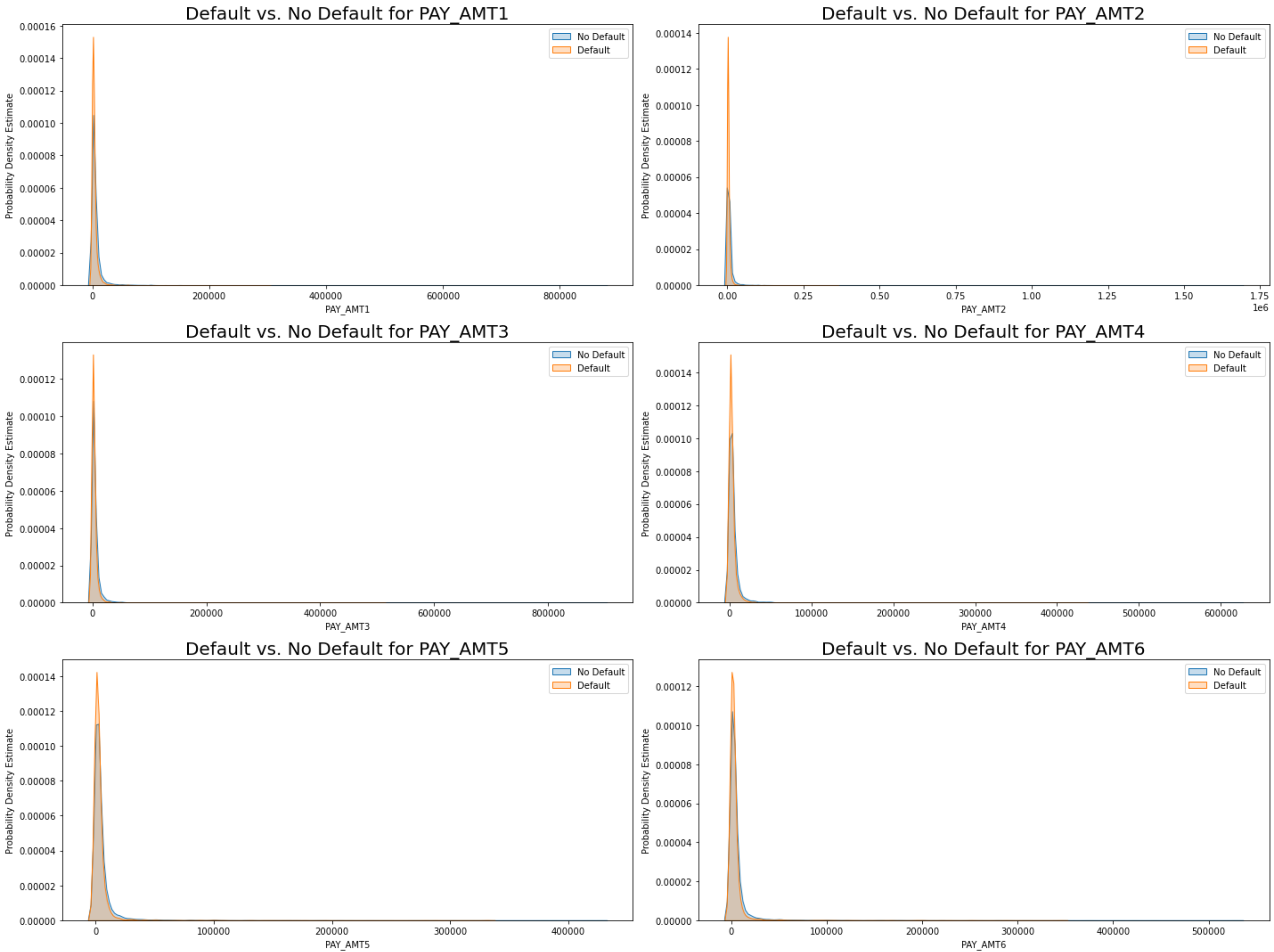


Inferences

1. From the above distribution plots, we can see that a larger portion of bill amounts is concentrated towards lower values.
2. Also, the tail of the distribution extends more towards the right side.
3. The above two observations implies that - a significant number of individuals have relatively low outstanding balances on their credit cards.

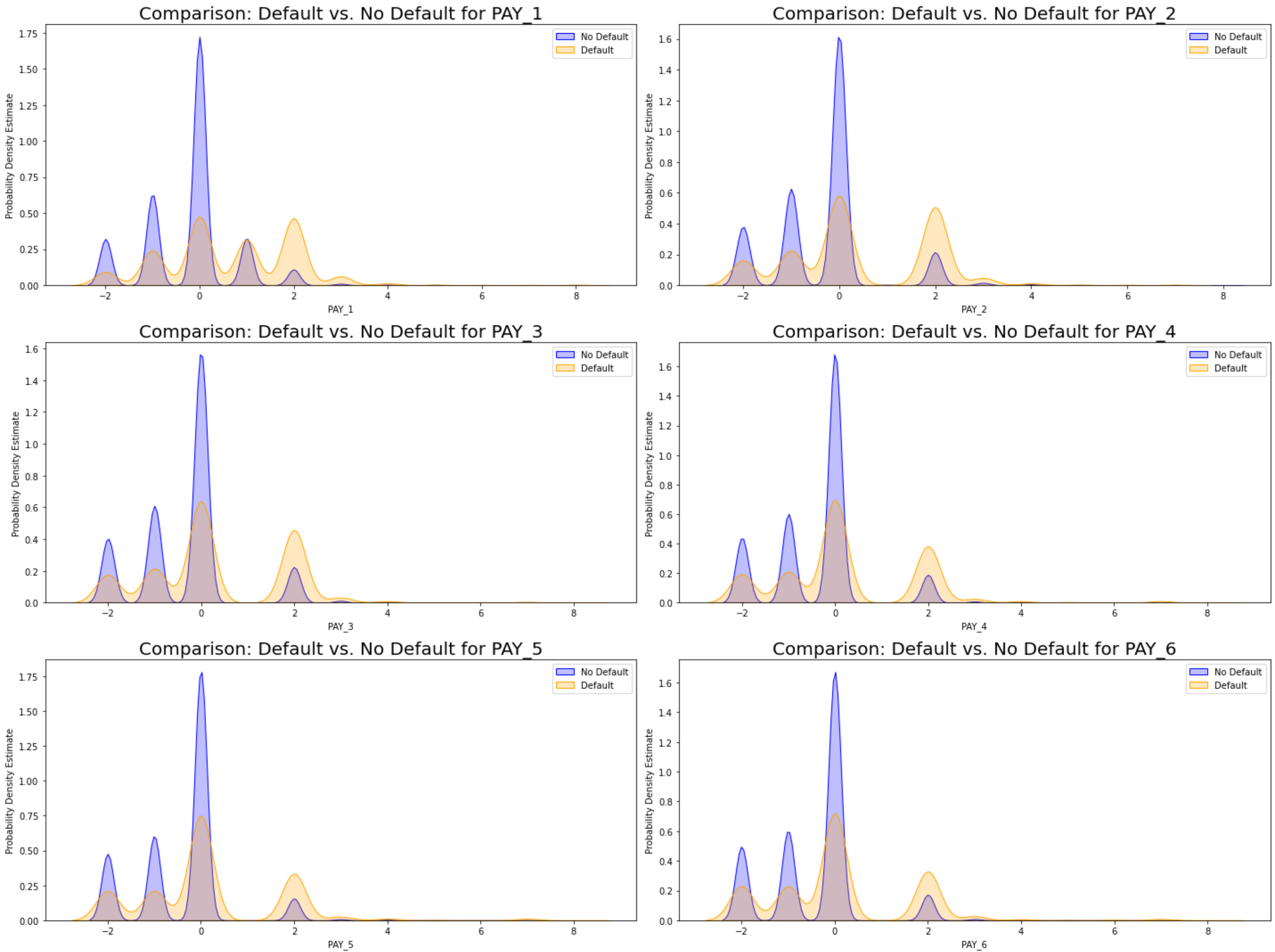
In [37]:

```
pay_amtx_fts = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
plt.figure(figsize=(20, 15))

for i, col in enumerate(pay_amtx_fts):
    plt.subplot(3, 2, i + 1)
    sns.kdeplot(data=df[df['def_payment'] == 0], x=col, label='No Default', shade=True)
    sns.kdeplot(data=df[df['def_payment'] == 1], x=col, label='Default', shade=True)
    plt.ylabel('Probability Density Estimate')
    plt.legend()
    plt.tight_layout()
    plt.title('Default vs. No Default for {}'.format(col), size=20)

plt.show()
```



In [38]:

```
pay_fts = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
plt.figure(figsize=(20, 15))

for i, col in enumerate(pay_fts):
    plt.subplot(3, 2, i + 1)
    sns.kdeplot(data=df[df['def_payment'] == 0], x=col, label='No Default', shade=True, co
    sns.kdeplot(data=df[df['def_payment'] == 1], x=col, label='Default', shade=True, color
    plt.ylabel('Probability Density Estimate')
    plt.legend()
    plt.tight_layout()
    plt.title('Comparison: Default vs. No Default for {}'.format(col), size=20)

plt.show()
```



Inference

1. The data that revolves at 0, implies duly paid. Such customers are more as per the above graph

4.2 EDA using visuals

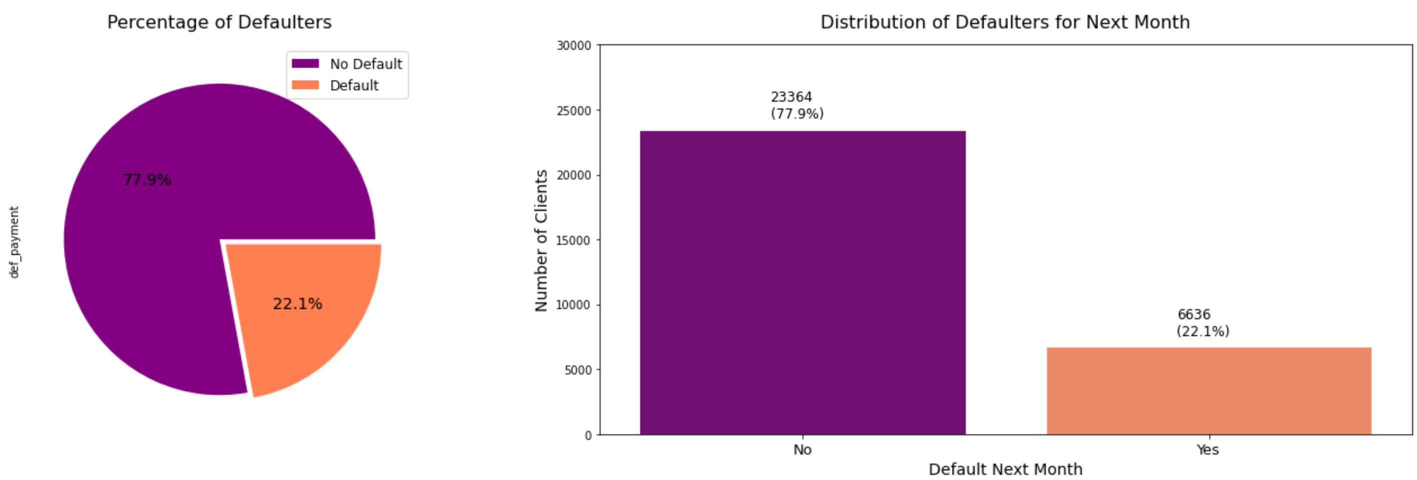
- Use (minimum) 2 plots (pair plot, heat map, correlation plot, regression plot...) to identify the optimal set of attributes that can be used for classification.
- Name them, explain why you think they can be helpful in the task and perform the plot as well. Unless proper justification for the choice of plots given, no credit will be awarded.

Score: 2 Marks

In [39]:

```
plt.subplots(figsize=(20, 6))

plt.subplot(1, 2, 1)
defaulter_counts = df['def_payment'].value_counts()
defaulter_counts.plot(kind='pie', labels=['', ''], autopct='%1.1f%%',
                      colors=['purple', 'coral'], explode=[0, 0.05],
                      textprops={'fontsize': 14})
plt.legend(labels=['No Default', 'Default'], fontsize=12)
plt.title('Percentage of Defaulters', fontsize=16, pad=15)

plt.subplot(1, 2, 2)
ax = sns.countplot(x="def_payment", data=df, palette=['purple', 'coral'])
plt.xlabel("Default Next Month", fontsize=14)
plt.ylabel('Number of Clients', fontsize=14)
plt.ylim(0, 30000)
```

```
plt.xticks([0, 1], ['No', 'Yes'], fontsize=13)
plt.title('Distribution of Defaulters for Next Month', fontsize=16, pad=15)

for p in ax.patches:
    count = p.get_height()
    percentage = (count / len(df)) * 100
    ax.annotate(f"{count}\n({percentage:.1f}%)", (p.get_x() + 0.32, p.get_height() + 1000)
                fontsize=12)

plt.tight_layout()
plt.show()
```



Inferences From the above graphs, we can infer the following:

- 78% of clients are expected to do payment on time - avoiding default payment in upcoming month.
- 22% of clients are anticipated to default, indicating significant credit risk.
- Hence, in-depth exploration is necessary to identify factors driving this high default likelihood.
- We will do further analysis to identify key factors and pinpointing key contributors that contribute to the high probability.
- Accordingly, the bank can strategize effective risk mitigation actions.

Understanding impact of Gender

In [40]:

```
print('Value Count of SEX: \n{}'.format(df['SEX'].value_counts()))
```

```
Value Count of SEX:
2    18112
1    11888
Name: SEX, dtype: Int64
```

In [41]:

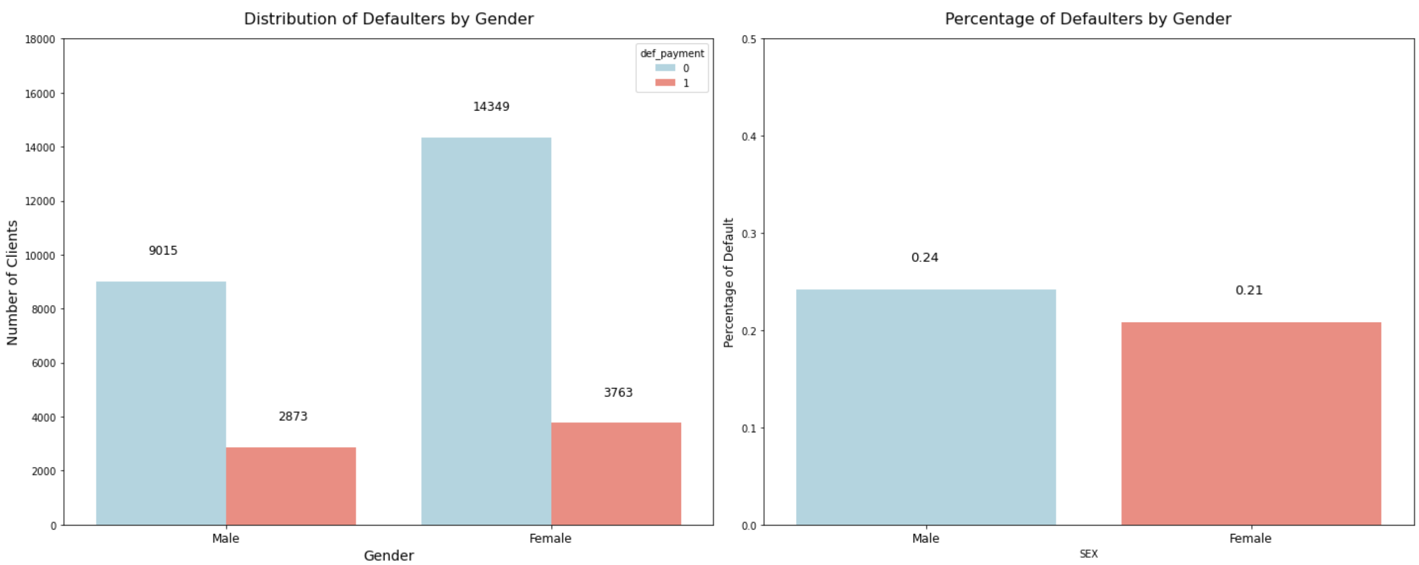
```
print('Value Count OF SEX with respect to def_payment: \n{}'
      .format(df['def_payment'].groupby(df['SEX']).value_counts(normalize =True)))
```

```
Value Count OF SEX with respect to def_payment:
SEX  def_payment
1    0              0.758328
     1              0.241672
2    0              0.792237
     1              0.207763
Name: def_payment, dtype: float64
```

In [42]:

```
plt.figure(figsize=(20, 8))

plt.subplot(1, 2, 1)
ax = sns.countplot(data=df, x='SEX', hue='def_payment', palette=['lightblue', 'salmon'])
plt.xlabel('Gender', fontsize=14)
```

```
plt.ylabel('Number of Clients', fontsize=14)
plt.ylim(0, 18000)
plt.xticks([0, 1], ['Male', 'Female'], fontsize=12)
plt.title('Distribution of Defaulters by Gender', fontsize=16, pad=15)

for i in ax.patches:
    ax.annotate(str(i.get_height()), (i.get_x() + 0.16, i.get_height() + 1000), fontsize=1

plt.subplot(1, 2, 2)
ax = sns.barplot(x="SEX", y="def_payment", data=df, palette=['lightblue', 'salmon'], ci=No

plt.ylabel("Percentage of Default", fontsize=12)
plt.ylim(0, 0.5)
plt.xticks([0, 1], ['Male', 'Female'], fontsize=12)
plt.title('Percentage of Defaulters by Gender', fontsize=16, pad=15)

for p in ax.patches:
    ax.annotate("%.2f" % (p.get_height()), (p.get_x() + 0.35, p.get_height() + 0.03), font

plt.tight_layout()
plt.show()
```



Inferences

1. The total number of data points for women is greater than that of men.
2. Also, the % of male defaulters are slightly higher than that of the female defaulters
3. Conclusion: Gender may play a signigicant role in credit risk, which has to be inveistigated further for better understanding.

Understanding Impact of Education Level

In [43]:
```
print('Value Count of EDUCATION: \n{}'.format(df['EDUCATION'].value_counts()))
```

```
Value Count of EDUCATION:
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: Int64
```

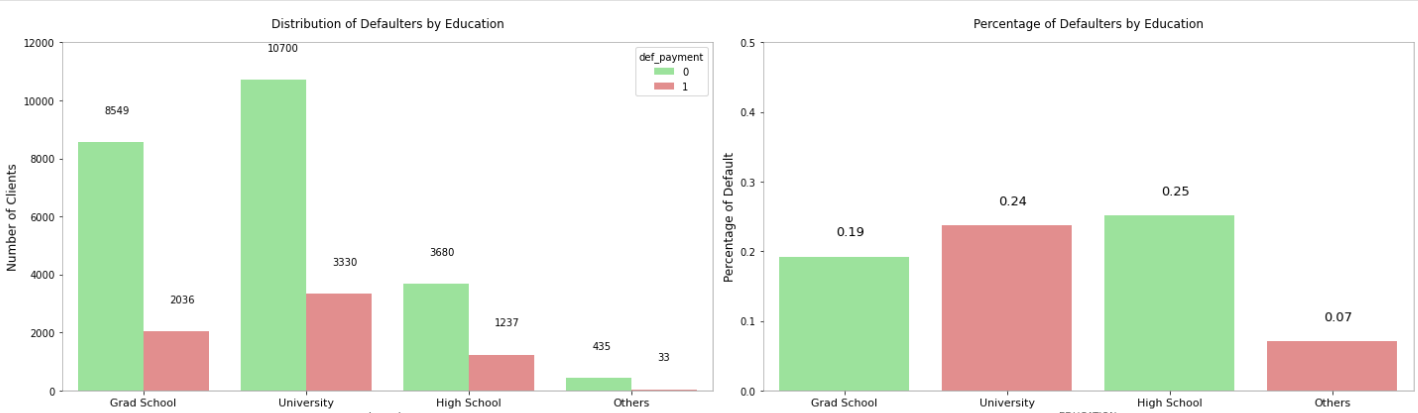
In [44]:
```
print('Value Count OF EDUCATION with respect to def_payment: \n{}'
      .format(df['def_payment'].groupby(df['EDUCATION']).value_counts(normalize =True)))
```

```
Value Count OF EDUCATION with respect to def_payment:
```

```
EDUCATION  def_payment
1          0              0.807652
           1              0.192348
2          0              0.762651
           1              0.237349
3          0              0.748424
           1              0.251576
4          0              0.929487
           1              0.070513
Name: def_payment, dtype: float64
```

In [45]:

```python
plt.figure(figsize=(20, 6))

plt.subplot(121)
ax = sns.countplot(data=df, x='EDUCATION', hue='def_payment',
                   palette=['lightgreen', 'lightcoral'])
plt.xlabel('Education', fontsize=12)
plt.ylabel('Number of Clients', fontsize=12)
plt.ylim(0, 12000)
plt.xticks([0, 1, 2, 3], ['Grad School', 'University', 'High School', 'Others'], fontsize=
plt.title('Distribution of Defaulters by Education', pad=15)

for i in ax.patches:
    ax.annotate(i.get_height(), (i.get_x() + 0.16, i.get_height() + 1000))

    for spine in ax.spines.values():
        spine.set_visible(True)
        spine.set_linewidth(0.5)
        spine.set_color('gray')

plt.subplot(122)
ax = sns.barplot(x="EDUCATION", y="def_payment", data=df,
                 palette=['lightgreen', 'lightcoral'], ci=None)

plt.ylabel("Percentage of Default", fontsize=12)
plt.ylim(0, 0.5)
plt.xticks([0, 1, 2, 3], ['Grad School', 'University', 'High School', 'Others'], fontsize=
plt.title('Percentage of Defaulters by Education', pad=15)

for p in ax.patches:
    ax.annotate("%.2f" % (p.get_height()), (p.get_x() + 0.35, p.get_height() + 0.03), font

    for spine in ax.spines.values():
        spine.set_visible(True)
        spine.set_linewidth(0.5)
        spine.set_color('gray')

plt.tight_layout()
plt.show()
```



Inferences

1. % of deaulters decreases as level of education increases.
2. Highest defaulters for High School
3. Others contain low % of defaulters.

Understanding Impact of Marriage

In [46]:

```
print('Value Count of MARRIAGE: \n{}'.format(df['MARRIAGE'].value_counts()))
```

```
Value Count of MARRIAGE:
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: Int64
```

In [47]:

```
df
```

Out[47]:

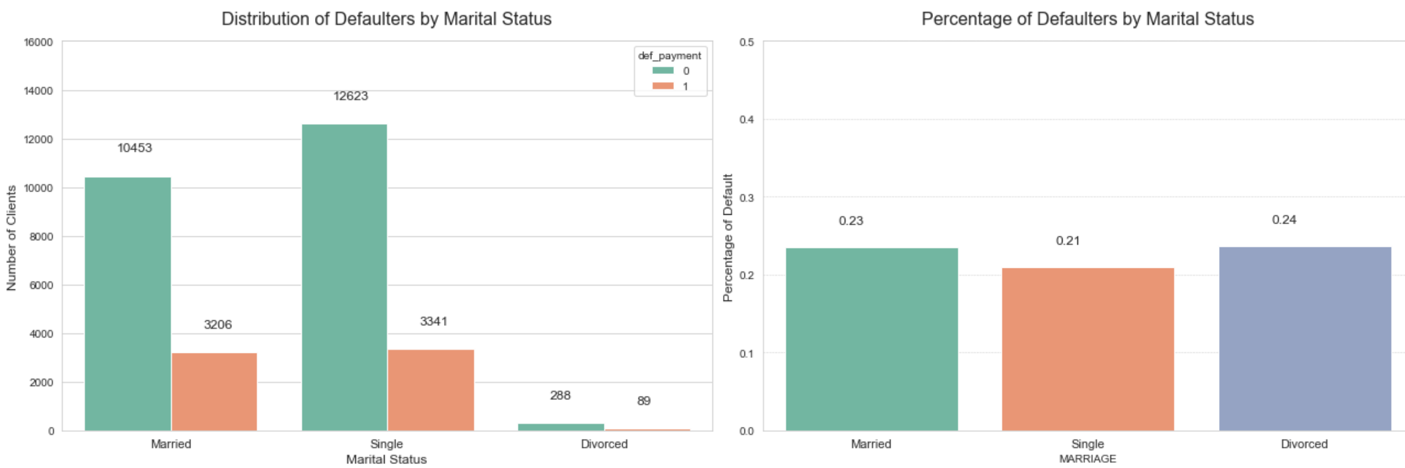
	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_/
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	
3	3	90000	2	2	2	34	0	0	0	0	...	14331	
4	4	50000	2	2	1	37	0	0	0	0	...	28314	
5	5	50000	1	2	1	57	-1	0	-1	0	...	20940	
...	...	...	...	...	...	...	...	...	...	...	...	...	
29996	29996	220000	1	3	1	39	0	0	0	0	...	88004	
29997	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	
29998	29998	30000	1	2	2	37	4	3	2	-1	...	20878	
29999	29999	80000	1	3	1	41	1	-1	0	0	...	52774	
30000	30000	50000	1	2	1	46	0	0	0	0	...	36535	

30000 rows × 25 columns

In [48]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style
sns.set_style("whitegrid")
sns.set_palette("Set2")

# Create a figure
plt.figure(figsize=(18, 6))

# Left subplot
plt.subplot(121)
ax1 = sns.countplot(data=df, x='MARRIAGE', hue='def_payment')
plt.xlabel('Marital Status', fontsize=12)
plt.ylabel('Number of Clients', fontsize=12)
plt.ylim(0, 16000)
plt.xticks([0, 1, 2], ['Married', 'Single', 'Divorced'], fontsize=11)
plt.title('Distribution of Defaulters by Marital Status', fontsize=16, pad=15)

for i in ax1.patches:
    ax1.annotate(f"{i.get_height()}", (i.get_x() + 0.15, i.get_height() + 1000), fontsize=
```

```
# Right subplot
plt.subplot(122)
ax2 = sns.barplot(x="MARRIAGE", y="def_payment", data=df, ci=None)
plt.ylabel("Percentage of Default", fontsize=12)
plt.ylim(0, 0.5)
plt.xticks([0, 1, 2], ['Married', 'Single', 'Divorced'], fontsize=11)
plt.title('Percentage of Defaulters by Marital Status', fontsize=16, pad=15)

for p in ax2.patches:
    ax2.annotate(f"{p.get_height():.2f}", (p.get_x() + 0.25, p.get_height() + 0.03), fonts

# Adjust layout and add horizontal gridlines
plt.tight_layout()
plt.gca().yaxis.grid(True, linestyle='--', linewidth=0.5)

plt.show()
```



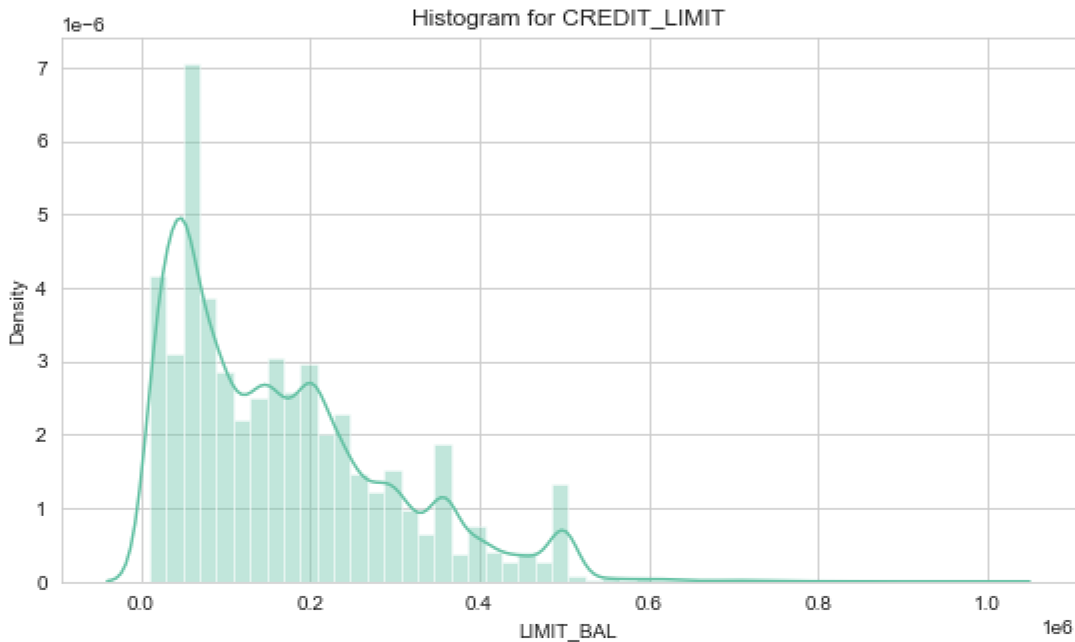
Inferences

1. We can see that % of defaulters are highest among divorced category as compared to married/single

Understanding Impact of CREDIT LIMIT

In [49]:
```
plt.subplots(figsize =(20,5))
plt.subplot(121)
sns.distplot(df.LIMIT_BAL)
plt.title('Histogram for CREDIT_LIMIT')
```

Out[49]:
```
Text(0.5, 1.0, 'Histogram for CREDIT_LIMIT')
```



Inferences

Based on the above graph, we can conclude that:

1. Majority of customers have a credit limit ≤ 200k.
2. Higher default rate observed within this credit limit range.
3. Customers with ≤ 200k credit limit more likely to default.
4. This group forms the majority of our customer base.

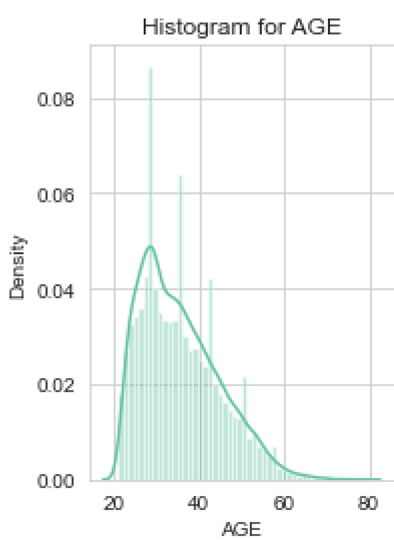
Understanding Impact of AGE

In [50]:

```
plt.subplot(122)
sns.distplot(df.AGE)
plt.title("Histogram for AGE")
plt.show
```

Out[50]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Inferences Based on the above graph, we can conclude that

1. Majority of customers are aged 25 to 40 years old.
2. Lower default likelihood within this age range.
3. Default probability relatively lower for customers in this age range.

Binning

In [51]:
```
df
```

Out[51]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_A
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	
3	3	90000	2	2	2	34	0	0	0	0	...	14331	1
4	4	50000	2	2	1	37	0	0	0	0	...	28314	2
5	5	50000	1	2	1	57	-1	0	-1	0	...	20940	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
29996	29996	220000	1	3	1	39	0	0	0	0	...	88004	3
29997	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	
29998	29998	30000	1	2	2	37	4	3	2	-1	...	20878	2
29999	29999	80000	1	3	1	41	1	-1	0	0	...	52774	1
30000	30000	50000	1	2	1	46	0	0	0	0	...	36535	3

30000 rows × 25 columns

In [52]:
```
df['LimitBin'] = pd.cut(df['LIMIT_BAL'],[5000, 50000, 100000, 150000, 200000, 300000, 4000
print('For LIMIT_BAL : \n{}'.format(df['LimitBin'].value_counts()))

df['AgeBin'] = pd.cut(df['AGE'],[20, 25, 30, 35, 40, 50, 60, 80])
print('For Age : \n{}'.format(df['AgeBin'].value_counts()))
```

```
For LIMIT_BAL :
(5000, 50000]        7676
(200000, 300000]     5059
(50000, 100000]      4822
(150000, 200000]     3978
(100000, 150000]     3902
(300000, 400000]     2759
(400000, 500000]     1598
(500000, 1100000]     206
Name: LimitBin, dtype: int64
For Age :
(25, 30]    7142
(40, 50]    6005
(30, 35]    5796
(35, 40]    4917
(20, 25]    3871
(50, 60]    1997
(60, 80]     272
Name: AgeBin, dtype: int64
```

In [53]:
```
df.head()
```

Out[53]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT6	PAY_AMT1	P
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BILL_AMT6	PAY_AMT1	P
2	2	120000	2	2	2	26	-1	2	0	0	...	3261	0	
3	3	90000	2	2	2	34	0	0	0	0	...	15549	1518	
4	4	50000	2	2	1	37	0	0	0	0	...	29547	2000	
5	5	50000	1	2	1	57	-1	0	-1	0	...	19131	2000	

5 rows × 27 columns

In [54]:
```
print('Default percentage for LIMIT_BAL : \n{}'
      .format(df['def_payment'].groupby(df['LimitBin']).value_counts(normalize = True)))
```

```
Default percentage for LIMIT_BAL :
LimitBin          def_payment
(5000, 50000]     0              0.682126
                  1              0.317874
(50000, 100000]   0              0.742016
                  1              0.257984
(100000, 150000]  0              0.788570
                  1              0.211430
(150000, 200000]  0              0.821518
                  1              0.178482
(200000, 300000]  0              0.839494
                  1              0.160506
(300000, 400000]  0              0.859369
                  1              0.140631
(400000, 500000]  0              0.878598
                  1              0.121402
(500000, 1100000] 0              0.888350
                  1              0.111650
Name: def_payment, dtype: float64
```

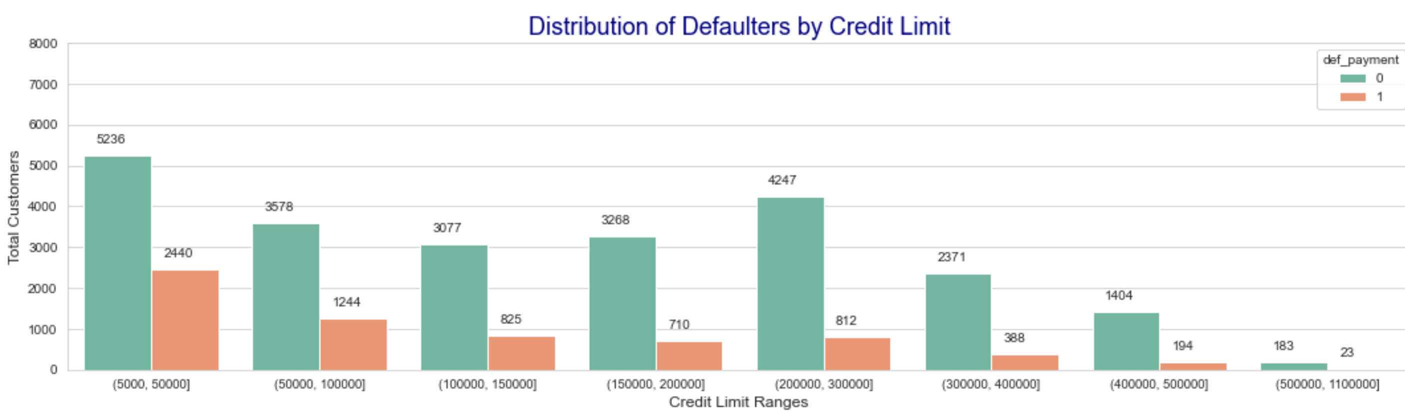
In [55]:
```
print('Default percentage for AGE : \n{}'
      .format(df['def_payment'].groupby(df['AgeBin']).value_counts(normalize = True)))
```

```
Default percentage for AGE :
AgeBin    def_payment
(20, 25]  0              0.733402
          1              0.266598
(25, 30]  0              0.798516
          1              0.201484
(30, 35]  0              0.805728
          1              0.194272
(35, 40]  0              0.783811
          1              0.216189
(40, 50]  0              0.767027
          1              0.232973
(50, 60]  0              0.747621
          1              0.252379
(60, 80]  0              0.731618
          1              0.268382
Name: def_payment, dtype: float64
```

Defaulters by Age Analysis

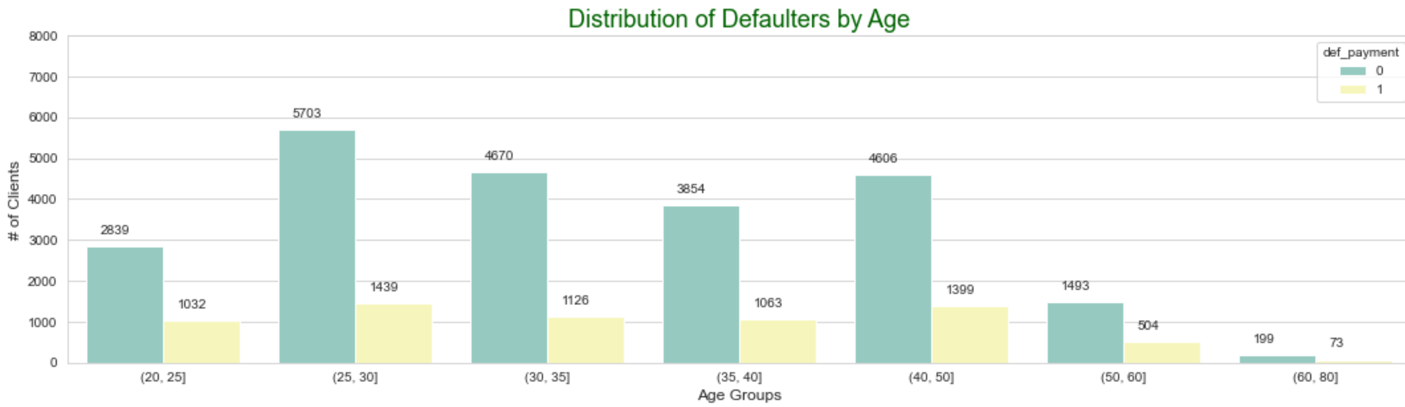
In [56]:
```
plt.subplots(figsize=(15, 8))  # Adjust the size of the plot
```

```python
plt.subplot(211)
df['LimitBin'] = df['LimitBin'].astype('str')
LimitBin_order = ['(5000, 50000]', '(50000, 100000]', '(100000, 150000]',
                  '(150000, 200000]','(200000, 300000]', '(300000, 400000]',
                  '(400000, 500000]', '(500000, 1100000]']

# Change the color palette
ax = sns.countplot(data=df, x='LimitBin', hue="def_payment", palette='Set2', order=LimitBi

plt.xlabel("Credit Limit Ranges", fontsize=12)
plt.ylabel("Total Customers", fontsize=12)
plt.ylim(0, 8000)
ax.tick_params(axis="x", labelsize=9.5)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x() + 0.075, p.get_height() + 300))

# Change the plot title size and color
plt.title('Distribution of Defaulters by Credit Limit', size=18, color='navy')

# Display the plot
plt.tight_layout()
plt.show()
```



In [57]:

```python
plt.subplots(figsize=(15, 8))   # Adjust the size of the plot

plt.subplot(212)
df['AgeBin'] = df['AgeBin'].astype('str')
AgeBin_order = ['(20, 25]', '(25, 30]', '(30, 35]',
                '(35, 40]', '(40, 50]', '(50, 60]', '(60, 80]']

# Change the color palette
ax = sns.countplot(data=df, x='AgeBin', hue="def_payment", palette='Set3', order=AgeBin_or

plt.xlabel("Age Groups", fontsize=12)
plt.ylabel("# of Clients", fontsize=12)
plt.ylim(0, 8000)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x() + 0.075, p.get_height() + 300))

# Change the plot title size and color
plt.title("Distribution of Defaulters by Age", size=18, color='darkgreen')

# Display the plot
plt.tight_layout()
plt.show()
```



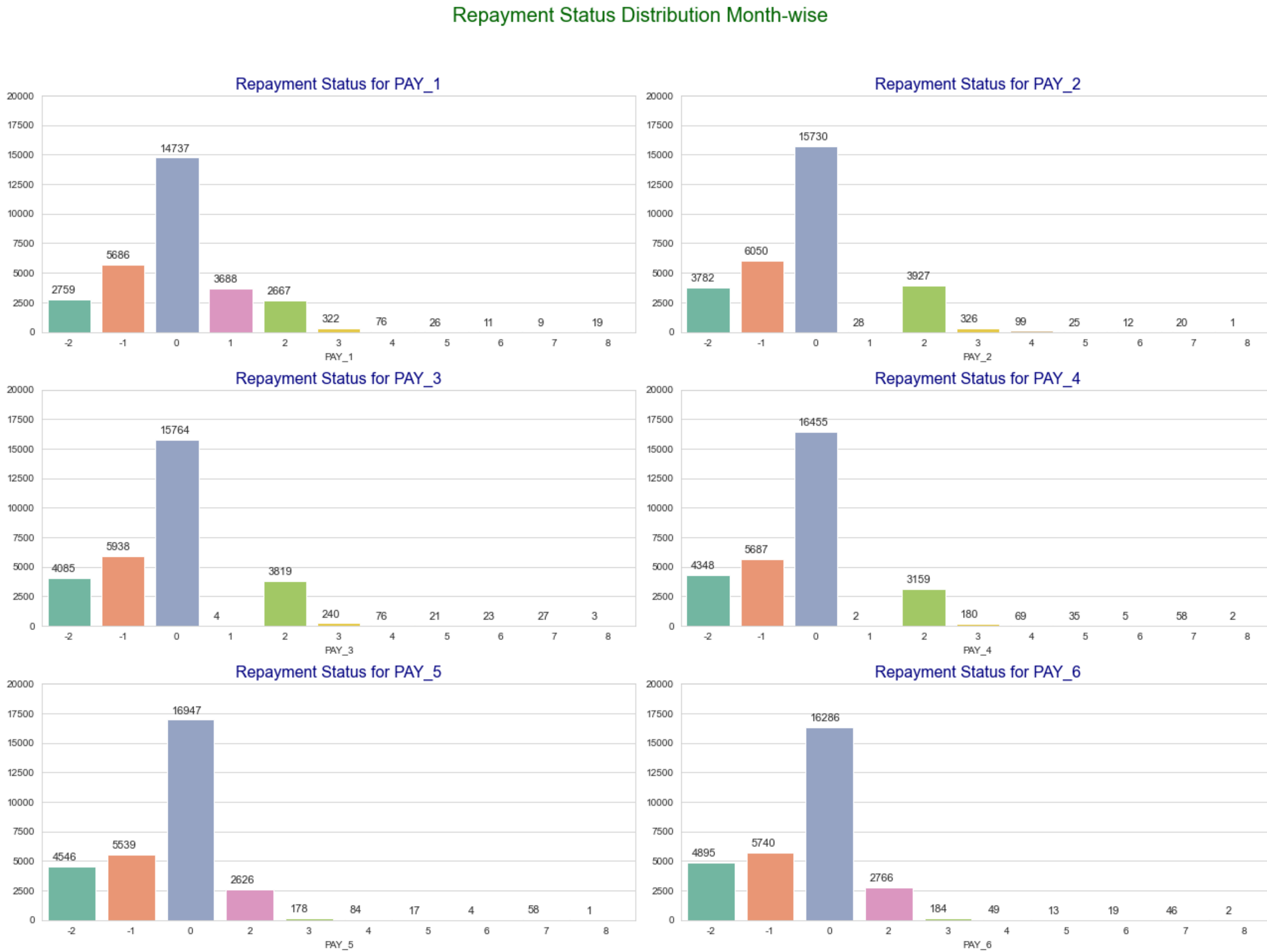
Inferences

On the basis of above analysis with graphs, we can say the following:

1. Majority of customers consistently pay their credit card bills on time.
2. It also indicates that there is a significantly lower default likelihood for timely payers compared to non-timely payers.

In [58]:

```
pay_x_fts = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
plt.figure(figsize=(18, 14))  # Adjust the overall size of the figure

for i, col in enumerate(pay_x_fts):
    plt.subplot(3, 2, i + 1)
    ax = sns.countplot(data=df, x=col, palette='Set2')  # Change the color palette
    plt.ylim(0, 20000)
    plt.ylabel('')
    plt.tight_layout()

    # Change individual subplot titles and annotation positions
    plt.title(f"Repayment Status for {col}", size=16, color='navy')
    for p in ax.patches:
        ax.annotate(p.get_height(), (p.get_x()
                                     + 0.08, p.get_height() + 500), fontsize=11)

# Change the main figure title
plt.suptitle("Repayment Status Distribution Month-wise", size=20, color='darkgreen')

# Display the plot
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust the position of the main title
plt.show()
```



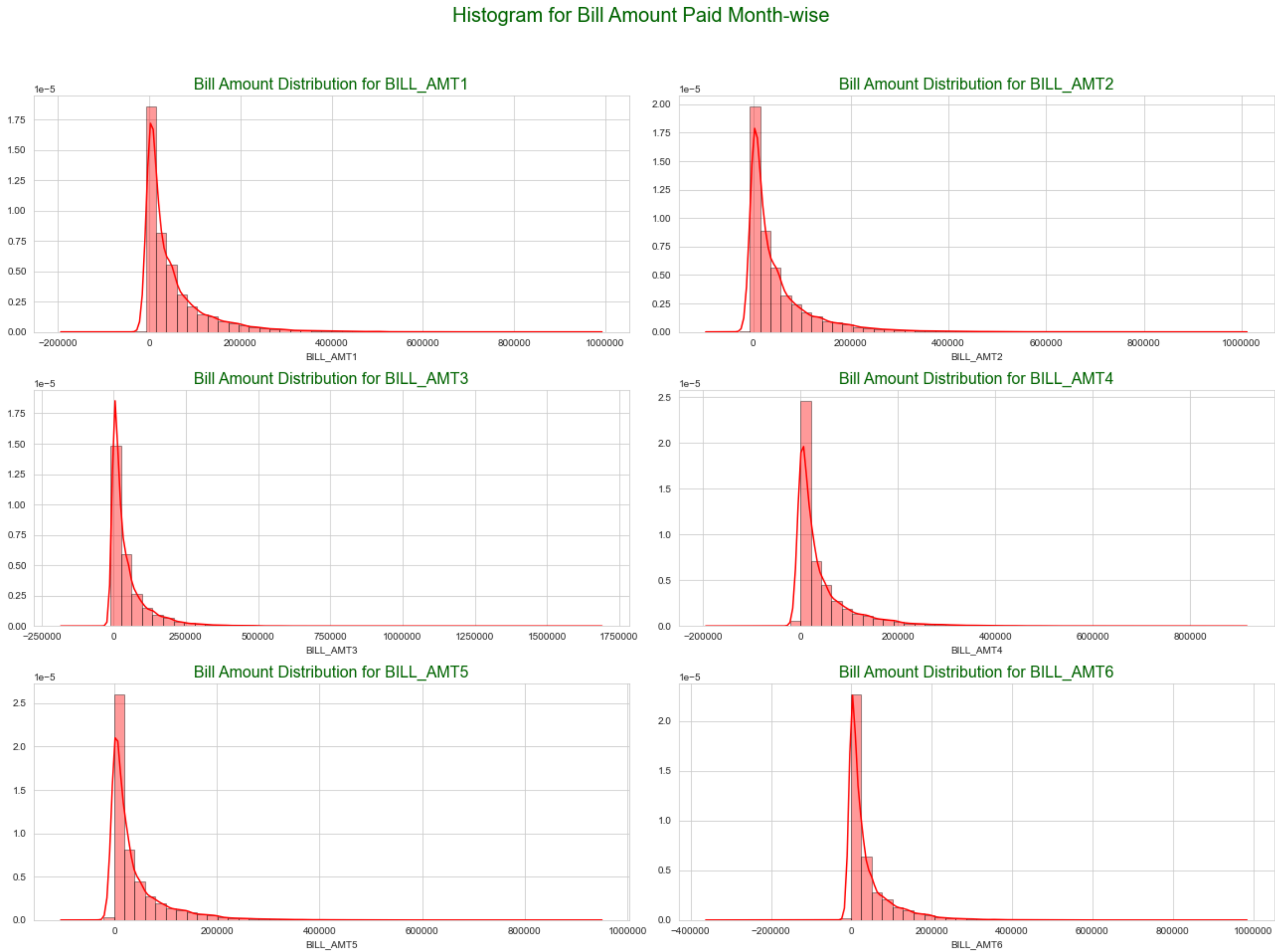
Inferences Basis the above analysis, we can say the following:

1. A small handful of customers exhibit delays of 4+ months in all PAY_X features.
2. Further, we can do a separate analysis for this group to improve the average default rate accuracy.

Bill Statement Analysis

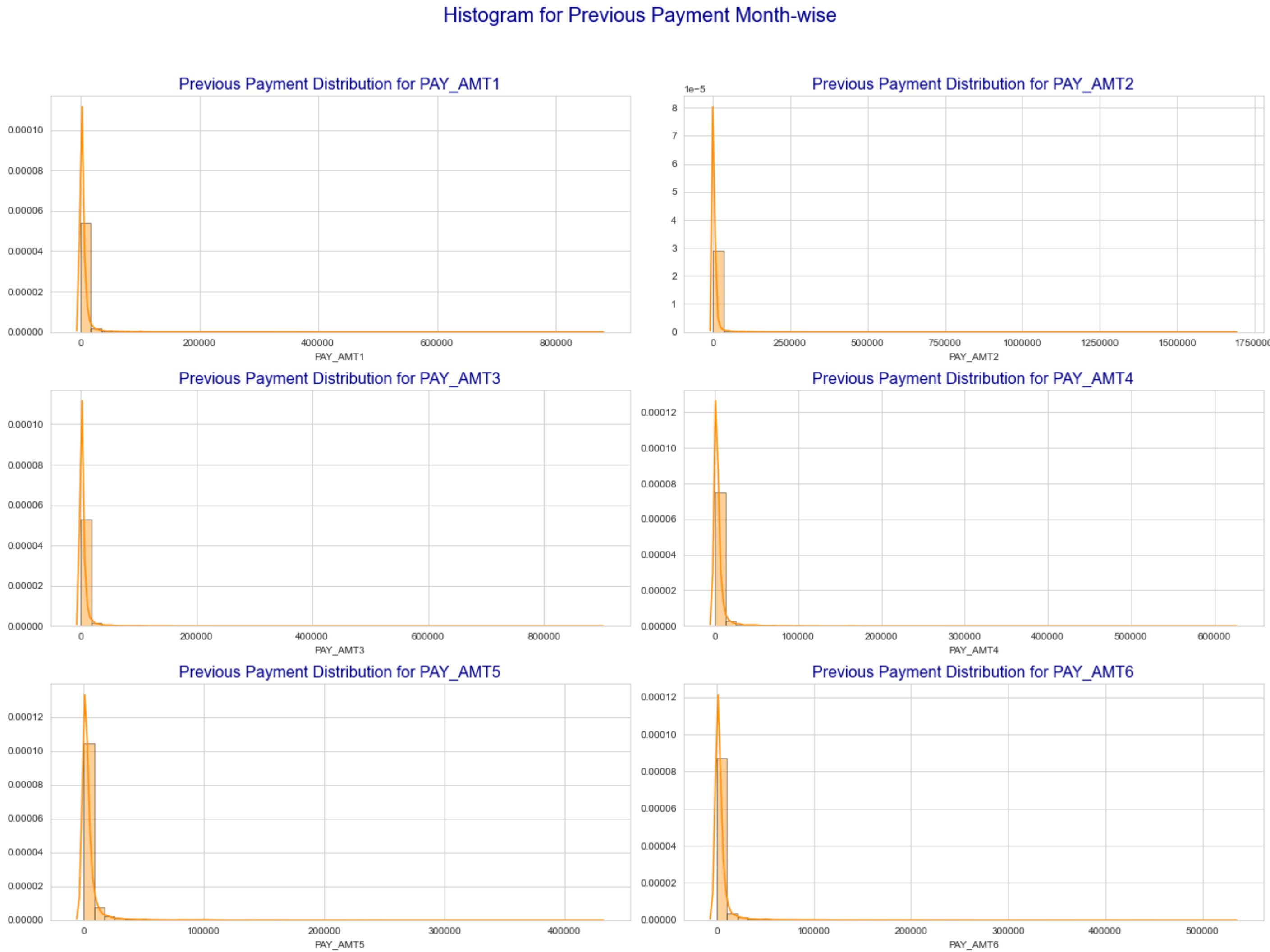
In [59]:

```
bill_amtx_fts = ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AM
plt.figure(figsize=(18, 14))  # Adjust the overall size of the figure

for i, col in enumerate(bill_amtx_fts):
    plt.subplot(3, 2, i + 1)
    sns.distplot(df.loc[:, col], color='red',
                 hist_kws={'edgecolor':'black'})  # Change the color to dark blue for the
    plt.ticklabel_format(style='plain', axis='x')  # Suppress scientific notation
    plt.ylabel('')
    plt.tight_layout()

    # Change individual subplot titles
    plt.title(f'Bill Amount Distribution for {col}', size=16, color='darkgreen')

# Change the main figure title
plt.suptitle("Histogram for Bill Amount Paid Month-wise", size=20, color='darkgreen')

# Display the plot
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust the position of the main title
plt.show()
```



In [60]:

```
pay_amtx_fts = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
plt.figure(figsize=(18, 14))  # Adjust the overall size of the figure

for i, col in enumerate(pay_amtx_fts):
    plt.subplot(3, 2, i + 1)
    sns.distplot(df.loc[:, col],
                 color='darkorange', hist_kws={'edgecolor':'black'})  # Change the color
    plt.ticklabel_format(style='plain', axis='x')  # Suppress scientific notation
    plt.ylabel('')
    plt.tight_layout()

    # Change individual subplot titles
    plt.title(f'Previous Payment Distribution for {col}', size=16, color='darkblue')

# Change the main figure title
plt.suptitle("Histogram for Previous Payment Month-wise", size=20, color='darkblue')

# Display the plot
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust the position of the main title
plt.show()
```



Inferences From the above analysis, we can conclude/infer the following:

1. Default rate is higher for customers who haven't made recent payments.
2. Customers with payments over 25k NT dollars have a lower default rate.
3. We can see a clear relationship between payment history and default rates.
4. Overall, the predictability of these features reinforces known patterns, i.e., the relationship between payment history and default rates are consistent and predictable

5. Data Wrangling

5.1 Univariate Filters

Numerical and Categorical Data

- Identify top 5 significant features by evaluating each feature independently with respect to the target variable by exploring

1. Mutual Information (Information Gain)
2. Gini index
3. Gain Ratio
4. Chi-Squared test
5. Fisher Score (From the above 5 you are required to use only any **two**)

For Text data

3. Stemming / Lemmatization.
4. Forming n-grams and storing them in the document vector.
5. TF-IDF (From the above 2 you are required to use only any **two**)

Score: 3 Marks

In [61]:

```
##---------Type the code below this line------------------##
# Separating target variables
df.head
feature_X = df.iloc[:, :-3]
feature_y = df.iloc[:, -3].astype('int')
print("_________________________________")
print(feature_X)
print("=________________________________")
print(feature_y)
```

```
________________________________
          ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_1  PAY_2  PAY_3  \
1          1      20000    2          2         1   24      2      2     -1
2          2     120000    2          2         2   26     -1      2      0
3          3      90000    2          2         2   34      0      0      0
4          4      50000    2          2         1   37      0      0      0
5          5      50000    1          2         1   57     -1      0     -1
...      ...        ...  ...        ...       ...  ...    ...    ...    ...
29996  29996     220000    1          3         1   39      0      0      0
29997  29997     150000    1          3         2   43     -1     -1     -1
29998  29998      30000    1          2         2   37      4      3      2
29999  29999      80000    1          3         1   41      1     -1      0
30000  30000      50000    1          2         1   46      0      0      0

       PAY_4  ...  BILL_AMT3  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  \
1         -1  ...        689          0          0          0         0
2          0  ...       2682       3272       3455       3261         0
3          0  ...      13559      14331      14948      15549      1518
4          0  ...      49291      28314      28959      29547      2000
5          0  ...      35835      20940      19146      19131      2000
...      ...  ...        ...        ...        ...        ...       ...
29996      0  ...     208365      88004      31237      15980      8500
29997     -1  ...       3502       8979       5190          0      1837
29998     -1  ...       2758      20878      20582      19357         0
29999      0  ...      76304      52774      11855      48944     85900
30000      0  ...      49764      36535      32428      15313      2078

       PAY_AMT2  PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6
1           689         0         0         0         0
2          1000      1000      1000         0      2000
3          1500      1000      1000      1000      5000
4          2019      1200      1100      1069      1000
5         36681     10000      9000       689       679
...         ...       ...       ...       ...       ...
29996     20000      5003      3047      5000      1000
29997      3526      8998       129         0         0
29998         0     22000      4200      2000      3100
29999      3409      1178      1926     52964      1804
30000      1800      1430      1000      1000      1000

[30000 rows x 24 columns]
=________________________________
1         1
2         1
3         0
4         0
5         0
```

```
        ..
29996    0
29997    0
29998    1
29999    1
30000    1
Name: def_payment, Length: 30000, dtype: int32
```

In [62]:

```
##---------Code starts below this line------------------##

# Univariate Feature Selection

# Calculate Mutual Information (MI) for each feature
mi_scores = mutual_info_classif(feature_X, feature_y)

# Display MI scores for each feature
for i, score in enumerate(mi_scores):
    print(f'Feature {feature_X.columns[i]}: MI = {score}')

print('-------------------------------------------------------------------------')

# Define the threshold for selecting top features
top_features_count = 15

# Initialize a list for high-scored features
high_scored_features = []

# Select the top features based on MI scores
for score, feature_name in sorted(zip(mi_scores, feature_X.columns), reverse=True)[:top_fe
    print(feature_name, score)
    high_scored_features.append(feature_name)

# Create a new dataframe with the selected high-scored features
df_credit_norm_mic = feature_X[high_scored_features]
print(df_credit_norm_mic.columns)
```

```
Feature ID: MI = 0.004026365373763996
Feature LIMIT_BAL: MI = 0.015799871581723135
Feature SEX: MI = 0.0009984194620762388
Feature EDUCATION: MI = 0.007264898069836212
Feature MARRIAGE: MI = 0.0
Feature AGE: MI = 0.004132919105854338
Feature PAY_1: MI = 0.07848198487783042
Feature PAY_2: MI = 0.04948101183911313
Feature PAY_3: MI = 0.03668434203911164
Feature PAY_4: MI = 0.03450921838634846
Feature PAY_5: MI = 0.030251421276994872
Feature PAY_6: MI = 0.02567861130230531
Feature BILL_AMT1: MI = 0.011623750199065697
Feature BILL_AMT2: MI = 0.006710137708671349
Feature BILL_AMT3: MI = 0.008006245728655381
Feature BILL_AMT4: MI = 0.0040657221481992245
Feature BILL_AMT5: MI = 0.006585785346204798
Feature BILL_AMT6: MI = 0.0060965823375001005
Feature PAY_AMT1: MI = 0.020986752668029850
Feature PAY_AMT2: MI = 0.01761829346863686
Feature PAY_AMT3: MI = 0.018613710532520367
Feature PAY_AMT4: MI = 0.016436049289125565
Feature PAY_AMT5: MI = 0.019244222470158112
Feature PAY_AMT6: MI = 0.011528267489294475
-------------------------------------------------------------------------
PAY_1 0.07848198487783042
PAY_2 0.04948101183911313
PAY_3 0.03668434203911164
PAY_4 0.03450921838634846
```

```
PAY_5 0.030251421276994872
PAY_6 0.025678611302305311
PAY_AMT1 0.020986752668029855
PAY_AMT5 0.019244222470158112
PAY_AMT3 0.018613710532520367
PAY_AMT2 0.017618293468636866
PAY_AMT4 0.016436049289125565
LIMIT_BAL 0.015799871581723135
BILL_AMT1 0.011623750199065697
PAY_AMT6 0.011528267489294475
BILL_AMT3 0.008006245728655381
Index(['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'PAY_AMT1',
       'PAY_AMT5', 'PAY_AMT3', 'PAY_AMT2', 'PAY_AMT4', 'LIMIT_BAL',
       'BILL_AMT1', 'PAY_AMT6', 'BILL_AMT3'],
      dtype='object')
```

In [63]:

```python
#### Calculation of GINI Index
gini_features = {}

for feature in feature_X.columns:
    # Calculate class proportions for each feature value
    counts = df.groupby([feature, 'def_payment']).size().unstack()
    proportions = counts.div(counts.sum(axis=1), axis=0)

    # Compute Gini index for each feature value
    gini = 1 - (proportions ** 2).sum(axis=1)

    # Calculate weighted average of Gini index values
    weighted_gini = (counts.sum(axis=1) / len(df)) * gini
    feature_gini = weighted_gini.sum()

    # Display Gini index for the feature
    print(f'Feature {feature}: Gini index = {feature_gini}')
    gini_features[feature] = feature_gini

# Display the dictionary and sort it by value
dict(sorted(gini_features.items(), key=lambda item: item[1], reverse=True))
```

```
Feature ID: Gini index = 0.0
Feature LIMIT_BAL: Gini index = 0.33294135657880514
Feature SEX: Gini index = 0.34399094028066435
Feature EDUCATION: Gini index = 0.3426988591932808
Feature MARRIAGE: Gini index = 0.3442180515480303
Feature AGE: Gini index = 0.3427201868736329
Feature PAY_1: Gini index = 0.2829146005600383
Feature PAY_2: Gini index = 0.3046378973542323
Feature PAY_3: Gini index = 0.3144229187123528
Feature PAY_4: Gini index = 0.3176500307517461
Feature PAY_5: Gini index = 0.31930124458049813
Feature PAY_6: Gini index = 0.3228713749793728
Feature BILL_AMT1: Gini index = 0.08569584278282737
Feature BILL_AMT2: Gini index = 0.08934915345326486
Feature BILL_AMT3: Gini index = 0.0924734871472114
Feature BILL_AMT4: Gini index = 0.0977342102335754
Feature BILL_AMT5: Gini index = 0.10402965843922488
Feature BILL_AMT6: Gini index = 0.10864786625184605
Feature PAY_AMT1: Gini index = 0.26289384826890905
Feature PAY_AMT2: Gini index = 0.2675869538795681
Feature PAY_AMT3: Gini index = 0.2683262094538118
Feature PAY_AMT4: Gini index = 0.27474373989301043
Feature PAY_AMT5: Gini index = 0.2747789772371485
Feature PAY_AMT6: Gini index = 0.27346324193455185
```

Out[63]:

```
{'MARRIAGE': 0.3442180515480303,
 'SEX': 0.34399094028066435,
 'AGE': 0.3427201868736329,
```

```
'EDUCATION': 0.3426988591932808,
'LIMIT_BAL': 0.33294135657880514,
'PAY_6': 0.3228713749793728,
'PAY_5': 0.31930124458049813,
'PAY_4': 0.3176500307517461,
'PAY_3': 0.3144229187123528,
'PAY_2': 0.3046378973542323,
'PAY_1': 0.2829146005600383,
'PAY_AMT5': 0.2747789772371485,
'PAY_AMT4': 0.27474373989301043,
'PAY_AMT6': 0.27346324193455185,
'PAY_AMT3': 0.2683262094538118,
'PAY_AMT2': 0.2675869538795681,
'PAY_AMT1': 0.26289384826890905,
'BILL_AMT6': 0.10864786625184605,
'BILL_AMT5': 0.10402965843922488,
'BILL_AMT4': 0.0977342102335754,
'BILL_AMT3': 0.0924734871472114,
'BILL_AMT2': 0.08934915345326486,
'BILL_AMT1': 0.08569584278282737,
'ID': 0.0}
```

5.2 Report observations

Write your observations from the results of each method. Clearly justify your choice of the method.

Score 1 mark

We start the above task by **feature selection** process. Feature selection involves the human-guided task of pinpointing significant and vital attributes important for the analysis/ model building.

We use different **Filter Methods** to evaluate individual feature. We first rank each feature according to uni-variate metric and amongst them, we select the highest ranking features. Then, we compute the score for each feature, which should in turn reflect the predictive power of each feature.

We considered two filter methods:

1. **Mutual Information (Information Gain)**: To identify the features that provide valuable insights into predicting the outcome of interest, we can leverage a technique centered around mutual information. This approach quantifies the level of mutual information shared between each independent variable and the dependent variable, subsequently highlighting the attributes with the most substantial information gain. In essence, this method gauges the extent to which each feature aligns with the target variable. A heightened mutual information score signifies a more pronounced connection between the feature and the target variable.

   For example, in the context of the dataset , we can employ this method to discern the attributes that hold the greatest relevance in forecasting whether a customer will default on their payment in the upcoming month.

1. **Gini index** serves as a metric employed to gauge the level of disorder or imbalance within classification tasks, commonly applied in decision trees. By applying the Gini index to the credit card clients dataset, we can assess the level of impurity each feature introduces in relation to the prediction of whether a customer will default on their payment the following month. This grants us Gini index scores for every feature, which can be utilized to establish a hierarchy of feature significance for classification purposes. Typically, features sporting higher Gini index values hold greater importance in predicting the target variable.

Conclusion In summary, following the utilization of both mutual information and Gini index techniques, we reached the determination that attributes linked to payments significantly correlate with the probability of defaulting on the next month's payment. As a result, all attributes, barring the ID, were incorporated into our analysis due to their relevance to the target variable. While the option to enhance the model exists through feature reduction, we opted to encompass all attributes in our solution to meet our objectives.

Note: We don't have any text data, any text related processes are not performed

6. Implement Machine Learning Techniques

Use any 2 ML algorithms

1. Classification -- Decision Tree classifier
2. Clustering -- kmeans
3. Association Analysis
4. Anomaly detection
5. Textual data -- Naive Bayes classifier (not taught in this course)

A clear justification have to be given for why a certain algorithm was chosen to address your problem.

Score: 4 Marks (2 marks each for each algorithm)

6.1 ML technique 1 + Justification

Feature Scaling of Numerical Attributes We need to do feature scaling of numerical attributes. This step is crucial in machine learning to ensure all features share a common scale, preventing dominance by larger values and aiding algorithms like gradient descent to converge faster. Scaling avoids biases in regularization methods and aligns with distribution assumptions, potentially enhancing model performance.

If we use tree-based algorithm, then Feature Scaling is not required, such as Random Forest and Decision Tree won't require Feature scaling.

However, we will do for non-tree based algorithm - which we have done in ML Technique 3. Hence, we will do feature scaling post ML Technique 2.

Machine Learning Technique 1:

We've opted for a Decision Tree classifier for the task. This choice is rooted in its alignment with human thought processes. Just like humans, Decision Trees navigate through choices and implications. This classifier takes available customer data along with credit and spending behavior to distinguish between credible and non-credible individuals. The structure resembles a tree: nodes symbolize dataset features, branches signify decision criteria, and leaves denote outcomes.

In [64]:

```
##---------Type the code below this line------------------##
```

In [65]:

```
# Divide the dataframe into training and testing sets;
#it's crucial that these two remain isolated during the training process.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
# This arrangement entails using 80% of the data for training
#and reserving the remaining 20% for testing purposes.
```

Notes

1. Random state is a model hyperparameter used to control the randomness involved in machine learning models. Typically 0 and 42 are used. The default value is None, and when used the function will give different result in different run. Most popular are 0 and 42. But any positive integer value can be used. We are using 42.
2. test size = 0.2 taken as a standard value

In [66]:

```
# Establish the classifier
clf1 = DecisionTreeClassifier(max_depth=10, random_state=42)
#we fixed a value - static value for max_depth. Later we will perform hyperparameter as we
#by deafult Decision Tree Classifier uses "Gini" criterion
#using all other default values all listed in Scikit Learn Offical Documentation:
#https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htm
# Train the classifier
clf1.fit(X_train, y_train)
```

Out[66]:

```
DecisionTreeClassifier(max_depth=10, random_state=42)
```

Notes/Inferences

1. Max_depth can have any values between 0 to 100. It indicates how deep the decision tree can be
2. More deeper the tree, more splits and hence capturing more information about the data.
3. The deeper, more complex the model
4. We need to find an optimal to avoid overfitting and underfitting issues.
5. Hence we need to arrive to find right max depth using hyper parameter tuning using either grid search or random search.
6. In the above case, we have fixed a specific value for max_depth. We are first going to check the accuracy and then we will go for hyperparameter tuning

In [67]:

```
# Make predictions on the test set
predictions = clf1.predict(X_test)
```

In [68]:

```
# Evaluate the performance on the test set
accuracy_score(y_true=y_test, y_pred=predictions)
```

Out[68]:

```
0.8095
```

Inferences

1. We have got accuracy as 0.8095, but we need to see if this the best we can get
2. Hence we will perform hyperparameter tuning later.

In [69]:

```
#Before getting in hyperparameter tuning, let's check for ROC,Accuracy, Precision, Recall
pred1 = clf1.predict(X_test)
roc_auc_score_dtc1 = roc_auc_score( y_test, pred1)
accuracy_score_dtc1 = accuracy_score(y_true = y_test, y_pred = pred1)
precision_score_dtc1 = precision_score(y_true = y_test, y_pred = pred1)
recall_score_dtc1 = recall_score(y_true = y_test, y_pred = pred1)
f1_score_dtc1 = f1_score(y_true = y_test, y_pred = pred1)

print("ROC Score of Decision tree model: {}".format(round(roc_auc_score_dtc1,3)))
```

```
print("Accuracy of Decision tree model: {}".format(round(accuracy_score_dtc1,3)))
print("precision score of Decision tree model: {}".format(round(precision_score_dtc1,3)))
print("recall score of Decision tree model: {}".format(round(recall_score_dtc1,3)))
print("f1 score of Decision tree model: {}".format(round(f1_score_dtc1,3)))
```

```
ROC Score of Decision tree model: 0.644
Accuracy of Decision tree model: 0.81
precision score of Decision tree model: 0.614
recall score of Decision tree model: 0.349
f1 score of Decision tree model: 0.445
```

In [70]:

```
#displaying the confusion matrix for visualization purpose
confusion_matrix1 = metrics.confusion_matrix(y_test, pred1)
confusion_matrix1
```

Out[70]:

```
array([[4399,  288],
       [ 855,  458]], dtype=int64)
```

In [71]:

```
TN, FP, FN, TP = confusion_matrix1.ravel()
# Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
TPR_DT = TP / (TP + FN)
FPR_DT = FP / (FP + TN)
print("True Positive Rate (TPR):", TPR_DT)
print("False Positive Rate (FPR):", FPR_DT)
```

```
True Positive Rate (TPR): 0.3488194973343488
False Positive Rate (FPR): 0.06144655429912524
```

HyperParameter Tuning

In [72]:

```
#defining parameters for hyperparameter tuning
param_dist = {
    "criterion": ["gini","entropy","log_loss"],
    "max_depth" :[1,2,3,4,5,6,7,8,9,10,None],
    "splitter":["best", "random"]

}
```

In [73]:

```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(clf1,param_grid = param_dist, cv = 10, n_jobs = -1)
```

Notes for the above code:

1. cv is basically cross validation, it's a resmapling technique used to evaluate ML models on a sample data
2. We need to select value of k (k fold cross validation) in a way that reduces variance and bias. Also we need to ensure that the value of k is the representative of the broader dataset
3. Through multiple experiments, it;s a practice to use k = 10, hence we are using the same
4. n_jobs = -1 is to speed up the processing for parallel execution. -1 implies using all cores of the system

In [74]:

```
grid.fit(X_train,y_train)
```

Out[74]:

```
GridSearchCV(cv=10,
             estimator=DecisionTreeClassifier(max_depth=10, random_state=42),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, None],
                         'splitter': ['best', 'random']})
```

In [75]:
```
grid.best_score_
```

Out[75]:
```
0.8213333333333332
```

Inferences

1. So the best score that we can get is 0.8213333

In [76]:
```
#for the above accuracy, the params are
grid.best_params_
```

Out[76]:
```
{'criterion': 'gini', 'max_depth': 3, 'splitter': 'best'}
```

In [77]:
```
#classifer using max_depth = 3, criterion = 'gini', random_state = 42
clf_2 = DecisionTreeClassifier(max_depth=3,
                               criterion = "gini", random_state=42, splitter = "best")
clf_2.fit(X_train, y_train)
```

Out[77]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [78]:
```
pred2 = clf_2.predict(X_test)
roc_auc_score_dtc2 = roc_auc_score( y_test, pred2)
accuracy_score_dtc2 = accuracy_score(y_true = y_test, y_pred = pred2)
precision_score_dtc2 = precision_score(y_true = y_test, y_pred = pred2)
recall_score_dtc2 = recall_score(y_true = y_test, y_pred = pred2)
f1_score_dtc2 = f1_score(y_true = y_test, y_pred = pred2)

print("ROC Score of Decision tree model after tuning: {}".format(round(roc_auc_score_dtc2,
print("Accuracy of Decision tree model after tuning: {}".format(round(accuracy_score_dtc2,
print("precision score of Decision tree model after tuning: {}".format(round(precision_sco
print("recall score of Decision tree model after tuning: {}".format(round(recall_score_dtc
print("f1 score of Decision tree model after tuning: {}".format(round(f1_score_dtc2,3)))
```

```
ROC Score of Decision tree model after tuning: 0.656
Accuracy of Decision tree model after tuning: 0.821
precision score of Decision tree model after tuning: 0.669
recall score of Decision tree model after tuning: 0.362
f1 score of Decision tree model after tuning: 0.47
```

Inferences

From the below table, we can clearly see that, after hyper paramter, we are getting 82% accuracy.

Category	Before Tuning	After Tuning
ROC	0.644	0.656
Accuracy	0.81	0.821
Precision	0.614	0.669
Recall	0.349	0.362
F1 Score	0.445	0.47

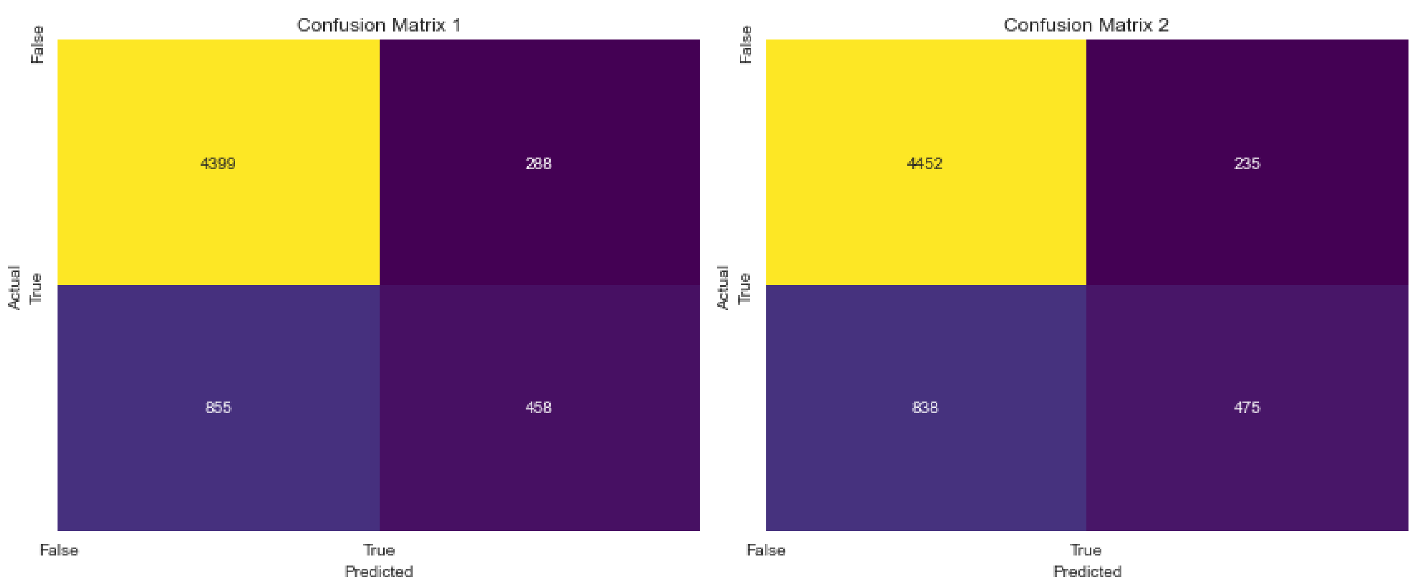
In [79]:
```
#displaying the confusion matrix for visualization purpose
confusion_matrix2 = metrics.confusion_matrix(y_test, pred2)
confusion_matrix2
```

Out[79]:
```
array([[4452,  235],
       [ 838,  475]], dtype=int64)
```

In [80]:
```
TN, FP, FN, TP = confusion_matrix2.ravel()
# Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
TPR_DT_Tuned = TP / (TP + FN)
FPR_DT_Tuned = FP / (FP + TN)
print("True Positive Rate (TPR):", TPR_DT_Tuned)
print("False Positive Rate (FPR):", FPR_DT_Tuned)
```

```
True Positive Rate (TPR): 0.3617669459253618
False Positive Rate (FPR): 0.05013868145935566
```

In [81]:
```
plt.figure(figsize=(12, 5))

# Plot the first confusion matrix
plt.subplot(1, 2, 1)
sns.heatmap(confusion_matrix1, annot=True, fmt="d", cmap="viridis", cbar=False)
plt.title("Confusion Matrix 1")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks([0, 1], ["False", "True"])
plt.yticks([0, 1], ["False", "True"])

plt.subplots_adjust(wspace=5)

# Plot the second confusion matrix
plt.subplot(1, 2, 2)
sns.heatmap(confusion_matrix2, annot=True, fmt="d", cmap="viridis", cbar=False)
plt.title("Confusion Matrix 2")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks([0, 1], ["False", "True"])
plt.yticks([0, 1], ["False", "True"])

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```



Inferences We can clearly see that the prediction accuracy has increased after hyperparameter tuning

6.2 ML technique 2 + Justification

In [82]:
```
##---------Type the code below this line------------------##
```

Next algorithm, that we want to use is **Random Forest Classifier**

Why **Random Forest Classifier**?

- **Team of Trees:** It has evolved from Decision tree where it's kind of a team of decision trees working together.
- **Mixed Samples:** Each tree learns from a group of training examples, where some examples might be repeated.
- **Handles Data Well:** It's good at dealing with different kinds of data, making it efficient for various tasks.
- **Better Guesses:** When it tries to predict something, it often gives better answers compared to just one tree.
- **Trees Help Each Other:** All these trees work as a group, helping each other out to make smarter predictions, especially when things are a bit tricky.

Hence, to summarize, we are using RF Classifier and expect better accuracy in terms of predicting outcomes as compared to Decision Tree algorithm. We will analyze and compare all algorithms in later sections

In [83]:
```
##---------Type the code below this line------------------##
from sklearn.ensemble import RandomForestClassifier
RFC_METRIC = 'gini'  #metric used for RandomForrestClassifier
NUM_ESTIMATORS = 100 #number of estimators used for RandomForrestClassifier
NO_JOBS = -1 #use all cores of the system for RandomForrestClassifier
RANDOM_STATE = 2018
rfc_1 = RandomForestClassifier(n_jobs=NO_JOBS,
                             random_state=RANDOM_STATE,
                             criterion=RFC_METRIC,
                             n_estimators=NUM_ESTIMATORS,
                             verbose=False)
```

In [84]:
```
rfc_1.fit(X_train, y_train.values)
```

Out[84]:
```
RandomForestClassifier(n_jobs=-1, random_state=2018, verbose=False)
```

In [85]:
```
rfc_pred1 = rfc_1.predict(X_test)
roc_auc_score_rfc1 = roc_auc_score(y_test, rfc_pred1)
accuracy_score_rfc1 = accuracy_score(y_true = y_test, y_pred = rfc_pred1)
precision_score_rfc1 = precision_score(y_true = y_test, y_pred = rfc_pred1)
recall_score_rfc1 = recall_score(y_true = y_test, y_pred = rfc_pred1)
f1_score_rfc1 = f1_score(y_true = y_test, y_pred = rfc_pred1)

print("ROC Score of Random Forest model: {}".format(round(roc_auc_score_rfc1,3)))
print("Accuracy of Random Forest model: {}".format(round(accuracy_score_rfc1,3)))
print("precision score of Random Forest model: {}".format(round(precision_score_rfc1,3)))
print("recall score of Random Forest model: {}".format(round(recall_score_rfc1,3)))
print("f1 score of Random Forest model: {}".format(round(f1_score_rfc1,3)))
```

```
ROC Score of Random Forest model: 0.649
Accuracy of Random Forest model: 0.812
precision score of Random Forest model: 0.624
recall score of Random Forest model: 0.358
f1 score of Random Forest model: 0.455
```

Hyper Parameter Tunning of RF Classifer

In [86]:
```
#defining parameters for hyperparameter tuning
```

```
param_grid = {
    'n_estimators': [100, 150,200],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [3, 6],
    'max_leaf_nodes': [3, 6, 9],
    'criterion' :["gini","entropy","log_loss"]

}
```

In [87]:
```
from sklearn.model_selection import GridSearchCV

CV_rfc = GridSearchCV(estimator = rfc_1, param_grid = param_grid, cv = 10, n_jobs = -1)
```

Alert

Time Consuming Steps Ahead Time Taken to run the below cell: 45 minutes-1 hour

In [88]:
```
CV_rfc.fit(X_train,y_train)
```

Out[88]:
```
GridSearchCV(cv=10,
             estimator=RandomForestClassifier(n_jobs=-1, random_state=2018,
                                              verbose=False),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [3, 6],
                         'max_features': ['sqrt', 'log2', None],
                         'max_leaf_nodes': [3, 6, 9],
                         'n_estimators': [100, 150, 200]})
```

In [89]:
```
CV_rfc.best_score_
```

Out[89]:
```
0.821625
```

In [90]:
```
CV_rfc.best_params_
```

Out[90]:
```
{'criterion': 'gini',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': 9,
 'n_estimators': 150}
```

In [91]:
```
#building the RF Classifier based on above parameters
rfc_2 = RandomForestClassifier(max_depth=3,
                               criterion = "gini", max_features = None, max_leaf_nodes = 9
                               n_estimators = 150)
rfc_2.fit(X_train, y_train)
```

Out[91]:
```
RandomForestClassifier(max_depth=3, max_features=None, max_leaf_nodes=9,
                       n_estimators=150)
```

In [92]:
```
rfc_pred2 = rfc_2.predict(X_test)

roc_auc_score_rfc2 = roc_auc_score( y_test, rfc_pred2)
accuracy_rfc2 = accuracy_score(y_true = y_test, y_pred = rfc_pred2)
precision_score_rfc2 = precision_score(y_true = y_test, y_pred = rfc_pred2)
recall_score_rfc2 = recall_score(y_true = y_test, y_pred = rfc_pred2)
f1_score_rfc2 = f1_score(y_true = y_test, y_pred = rfc_pred2)

print("ROC Score of Random Forest model after tuning: {}".format(round(roc_auc_score_rfc2,
```

```
print("Accuracy of Random Forest model after tuning: {}".format(round(accuracy_rfc2,3)))
print("precision score of Random Forest model after tuning: {}".format(round(precision_sc
print("recall score of Random Forest model after tuning: {}".format(round(recall_score_rf
print("f1 score of Random Forest model after tuning: {}".format(round(f1_score_rfc2,3)))
```

```
ROC Score of Random Forest model after tuning: 0.648
Accuracy of Random Forest model after tuning: 0.82
precision score of Random Forest model after tuning: 0.676
recall score of Random Forest model after tuning: 0.342
f1 score of Random Forest model after tuning: 0.454
```

Inferences

From the below table, we can clearly see that, after hyper paramter, we are getting 82% accuracy.

Category	Before Tuning	After Tuning
ROC	0.649	0.645
Accuracy	0.812	0.821
Precision	0.624	0.675
Recall	0.358	0.349
F1 Score	0.455	0.46

In [93]:
```
#displaying the confusion matrix for visualization purpose
rfc_confusion_matrix1 = metrics.confusion_matrix(y_test, rfc_pred1)
rfc_confusion_matrix1

#displaying the confusion matrix for visualization purpose
rfc_confusion_matrix2 = metrics.confusion_matrix(y_test, rfc_pred2)
rfc_confusion_matrix2
```

Out[93]:
```
array([[4472,  215],
       [ 864,  449]], dtype=int64)
```

In [94]:
```
TN, FP, FN, TP = rfc_confusion_matrix1.ravel()
# Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
TPR_RF = TP / (TP + FN)
FPR_RF = FP / (FP + TN)
print("True Positive Rate (TPR):", TPR_RF)
print("False Positive Rate (FPR):", FPR_RF)
```

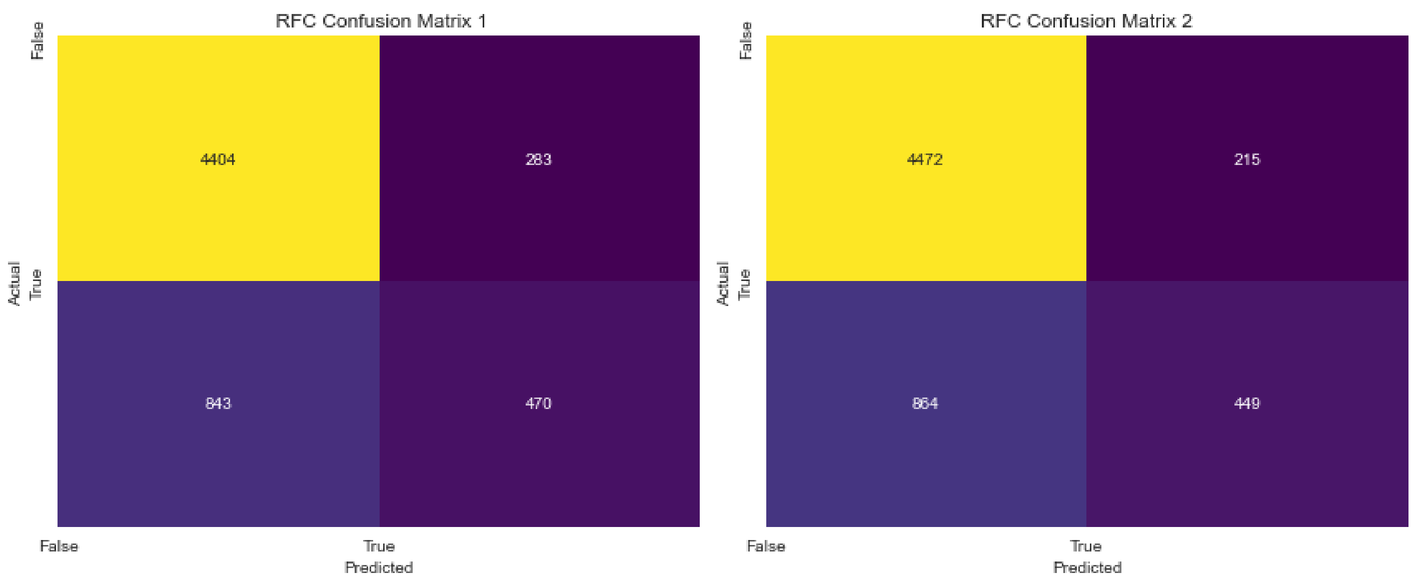
```
True Positive Rate (TPR): 0.357958872810358
False Positive Rate (FPR): 0.060379773842543207
```

In [95]:
```
TN, FP, FN, TP = rfc_confusion_matrix2.ravel()
# Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
TPR_RF_Tuned = TP / (TP + FN)
FPR_RF_Tuned = FP / (FP + TN)
print("True Positive Rate (TPR):", TPR_RF_Tuned)
print("False Positive Rate (FPR):", FPR_RF_Tuned)
```

```
True Positive Rate (TPR): 0.341964965727342
False Positive Rate (FPR): 0.045871559633027525
```

In [96]:
```
plt.figure(figsize=(12, 5))

# Plot the first confusion matrix
plt.subplot(1, 2, 1)
sns.heatmap(rfc_confusion_matrix1, annot=True, fmt="d", cmap="viridis", cbar=False)
```

```
plt.title(" RFC Confusion Matrix 1")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks([0, 1], ["False", "True"])
plt.yticks([0, 1], ["False", "True"])

plt.subplots_adjust(wspace=5)

# Plot the second confusion matrix
plt.subplot(1, 2, 2)
sns.heatmap(rfc_confusion_matrix2, annot=True, fmt="d", cmap="viridis", cbar=False)
plt.title("RFC Confusion Matrix 2")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks([0, 1], ["False", "True"])
plt.yticks([0, 1], ["False", "True"])

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```



6.3 ML technique 3 + Justification

Now, we will be working with KNN

Why KNN?

K-Nearest Neighbors (KNN) is a simple and interpretable algorithm for predicting credit card defaulters, suitable for quick model development and handling non-linear relationships. It excels in capturing local patterns and requires minimal feature engineering, making it valuable for smaller datasets with diverse subgroups of default patterns.

KNN is used for classification as well are regression predictive problems. However, industry experts uses KNN for classification problems.

However, KNN's computational intensity, sensitivity to noise and irrelevant features, optimal 'K' selection, and challenges with imbalanced or high-dimensional data need consideration.

Since, we don't have huge dataset, so we can apply KNN.

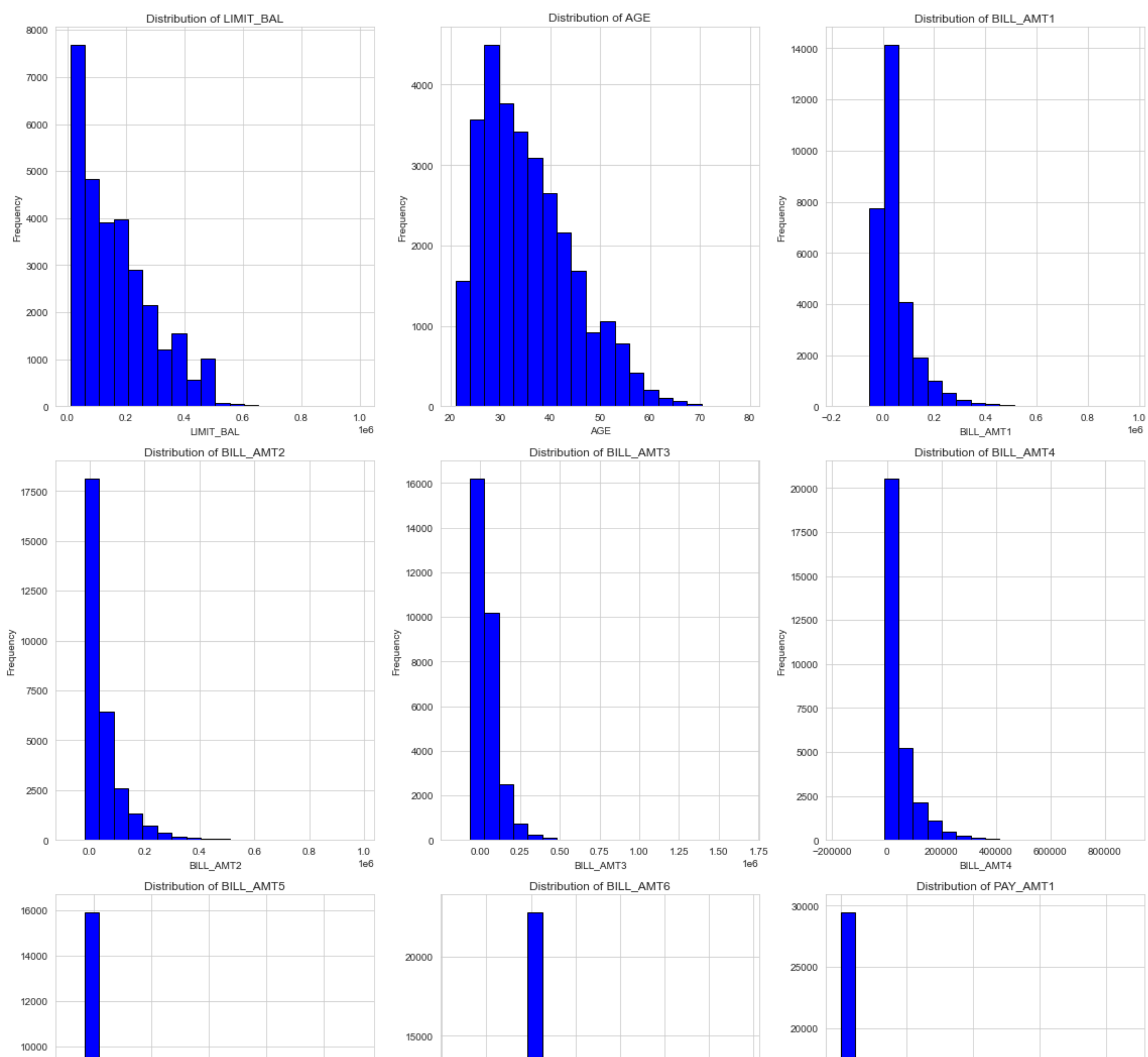
Feature Scaling of numerical attributes - required for KNN

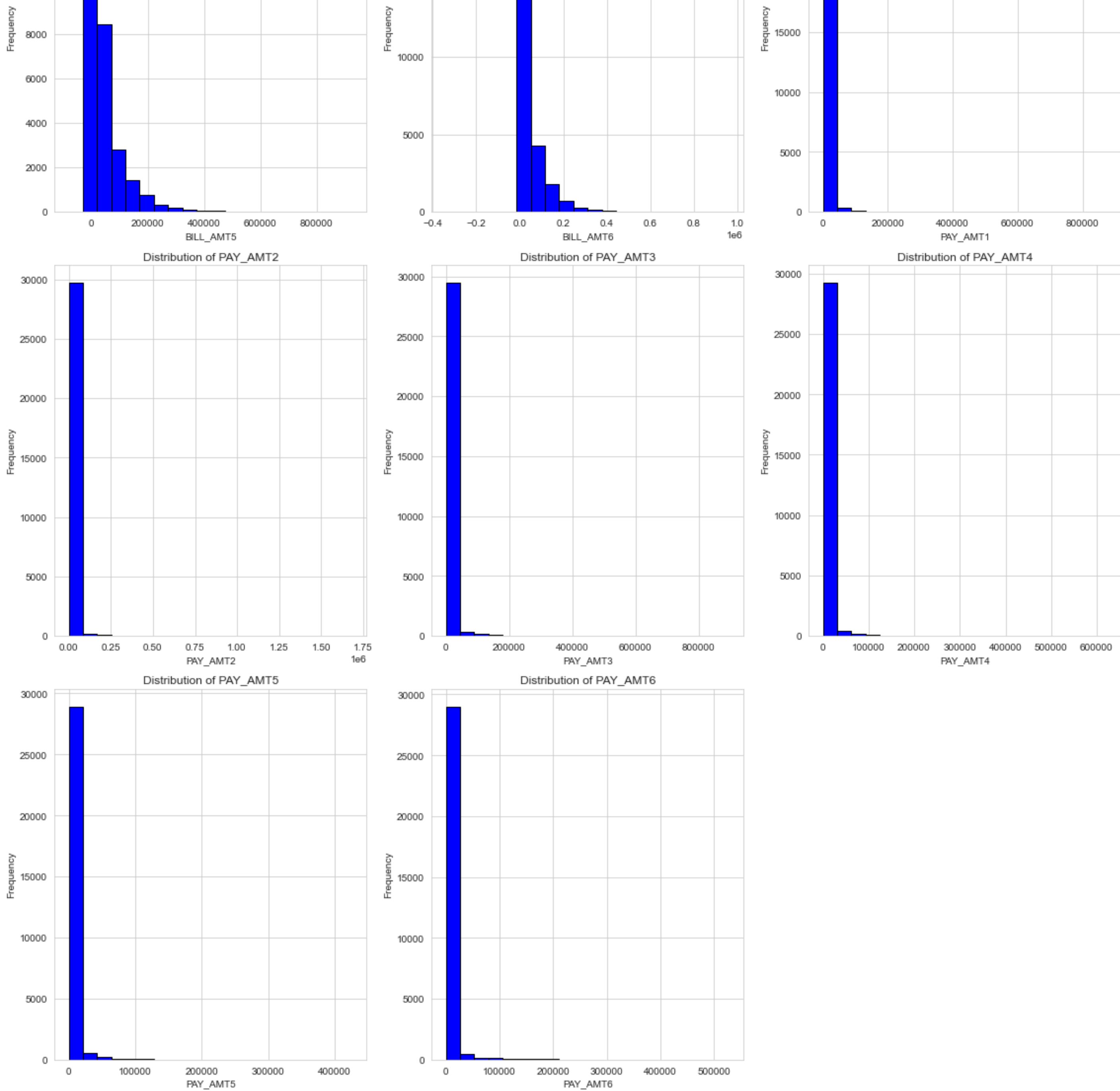
In [97]:

```python
# data standardization with  sklearn
from sklearn.preprocessing import StandardScaler
```

In [98]:

```python
#Let's check what's the distribution of the each column - whether it's normally distribute
#based on the distribution, we can selection the standrdization process.

columns_to_visualize = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_A
               'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
               'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
num_columns = len(columns_to_visualize)
num_rows = num_columns // 3 + (num_columns % 3 > 0) # Calculate the number of rows needed

plt.figure(figsize=(16, 6*num_rows))

for i, column in enumerate(columns_to_visualize, 1):
    plt.subplot(num_rows, 3, i)   # 3 graphs in a row
    plt.hist(df[column], bins=20, color='blue', edgecolor='black')
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```





Inferences

The above columns distribution are not properly normally distributed

In [101…

```
from scipy.stats import normaltest
dataset = df
# List of column names to analyze
col_to_analyze = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3',
                  'BILL_AMT4','BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
                  'PAY_AMT2', 'PAY_AMT3','PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']

# Create a DataFrame to store the statistics
statistics_df = pd.DataFrame(columns=['Column', 'Mean', 'Median', 'Mode', 'Is_Normal'])

for col in col_to_analyze:
    mean_val = dataset[col].mean()
    median_val = dataset[col].median()
    mode_val = dataset[col].mode().iloc[0]  # Mode can have multiple values, so we take th
    is_normal = normaltest(dataset[col]).pvalue > 0.05  # Perform normality test using sci
    mean_affected_by_outliers = abs(mean_val - median_val) > (2 * dataset[col].std())  # (
```

```
    statistics_df = statistics_df.append({'Column': col, 'Mean': mean_val, 'Median': media
                                          'Mode': mode_val, 'Is_Normal': is_normal,
                                          'Mean_Affected_By_Outliers': mean_affected_by_ou
                                         ignore_index=True)

# Print the statistics DataFrame
print(statistics_df)
```

```
       Column           Mean    Median   Mode  Is_Normal  \
0    LIMIT_BAL  167484.322667  140000.0  50000      False
1          AGE      35.485500      34.0     29      False
2    BILL_AMT1   51223.330900   22381.5      0      False
3    BILL_AMT2   49179.075167   21200.0      0      False
4    BILL_AMT3   47013.154800   20088.5      0      False
5    BILL_AMT4   43262.948967   19052.0      0      False
6    BILL_AMT5   40311.400967   18104.5      0      False
7    BILL_AMT6   38871.760400   17071.0      0      False
8     PAY_AMT1    5663.580500    2100.0      0      False
9     PAY_AMT2    5921.163500    2009.0      0      False
10    PAY_AMT3    5225.681500    1800.0      0      False
11    PAY_AMT4    4826.076867    1500.0      0      False
12    PAY_AMT5    4799.387633    1500.0      0      False
13    PAY_AMT6    5215.502567    1500.0      0      False

    Mean_Affected_By_Outliers
0                         0.0
1                         0.0
2                         0.0
3                         0.0
4                         0.0
5                         0.0
6                         0.0
7                         0.0
8                         0.0
9                         0.0
10                        0.0
11                        0.0
12                        0.0
13                        0.0
```

Inferences Since the 'Mean_Affected_By_Outliers' is consistently 0 for all columns, it suggests that mean and median for each columns are relatively close. This also means, outliers may not significantly impacting the mean value - but we will have to assess this

And hence we will start with Z-score normalization for standardization purpose.

Typically, inudstry experts considers z-score normalization as a default choice, that works well in many cases.

However, if we have outliers, skewness, we should use other standaridization such as

- Robust Scaling
- Log Transformation
- Power Transformation
- Min-max scaling etc.

In [102…

```
col_to_normalize = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4'
                    'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
                    'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
# Create a StandardScaler
scaler = StandardScaler()

X_train_normalized = X_train.copy()   # Create a copy of the training data
```

```
X_test_normalized = X_test.copy()     # Create a copy of the test data

# Apply Z-score normalization to the specified columns
X_train_normalized[col_to_normalize] = scaler.fit_transform(X_train[col_to_normalize])
X_test_normalized[col_to_normalize] = scaler.transform(X_test[col_to_normalize])
```

In [103…

```
#Performing a few checks to see if Z-score normalization would work for our dataset or not
from scipy.stats import normaltest
dataset = X_train_normalized
# List of column names to analyze
col_to_analyze = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
               'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
               'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']

# Create a DataFrame to store the statistics
statistics_df = pd.DataFrame(columns=['Column', 'Mean', 'Median', 'Mode', 'Is_Normal'])

for col in col_to_analyze:
    mean_val = dataset[col].mean()
    median_val = dataset[col].median()
    mode_val = dataset[col].mode().iloc[0]  # Mode can have multiple values, so we take th
    is_normal = normaltest(dataset[col]).pvalue > 0.05  # Perform normality test using sci

    statistics_df = statistics_df.append({'Column': col, 'Mean': mean_val, 'Median': media
                                        'Mode': mode_val, 'Is_Normal': is_normal},
                                       ignore_index=True)

# Print the statistics DataFrame
print(statistics_df)
```

```
       Column          Mean    Median      Mode Is_Normal
0   LIMIT_BAL -5.812769e-18 -0.209868 -0.903605     False
1         AGE  1.780107e-16 -0.161817 -0.703237     False
2   BILL_AMT1 -6.606290e-17 -0.389731 -0.693841     False
3   BILL_AMT2 -7.540265e-18 -0.391617 -0.689698     False
4   BILL_AMT3 -1.808738e-18 -0.385586 -0.675060     False
5   BILL_AMT4 -2.086757e-17 -0.374656 -0.671371     False
6   BILL_AMT5 -6.522560e-18 -0.363907 -0.662168     False
7   BILL_AMT6  7.308968e-19 -0.365296 -0.653898     False
8    PAY_AMT1 -1.094495e-16 -0.209015 -0.331937     False
9    PAY_AMT2 -7.299427e-17 -0.162830 -0.245478     False
10   PAY_AMT3 -5.441191e-17 -0.189574 -0.288246     False
11   PAY_AMT4 -2.865382e-16 -0.207358 -0.299358     False
12   PAY_AMT5  9.189408e-17 -0.212487 -0.308523     False
13   PAY_AMT6  7.780929e-17 -0.209612 -0.295532     False
```

Inferences

Although we have normalized we seethat the values of mean and median have significant difference. Hence we will assess other normalization methods too

We will now assess **Robust Scaling methods** which is used when mean and standard deviation are impacted by outliers.

In [104…

```
from sklearn.preprocessing import RobustScaler


col_to_scale = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
               'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
               'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
# Create a RobustScaler
scaler = RobustScaler()
```

```
# Fit the scaler on the training data and transform both training and test data
X_train_scaled = X_train.copy()   # Create a copy of the training data
X_test_scaled = X_test.copy()     # Create a copy of the test data

# Apply Robust Scaling to the specified columns
X_train_scaled[col_to_scale] = scaler.fit_transform(X_train[col_to_scale])
X_test_scaled[col_to_scale] = scaler.transform(X_test[col_to_scale])
```

In [105…

```
dataset = X_train_scaled
# List of column names to analyze
col_to_analyze = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
               'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
               'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']

# Create a DataFrame to store the statistics
statistics_df = pd.DataFrame(columns=['Column', 'Mean', 'Median', 'Mode', 'Is_Normal'])

for col in col_to_analyze:
    mean_val = dataset[col].mean()
    median_val = dataset[col].median()
    mode_val = dataset[col].mode().iloc[0]  # Mode can have multiple values, so we take th
    is_normal = normaltest(dataset[col]).pvalue > 0.05  # Perform normality test using sci

    statistics_df = statistics_df.append({'Column': col, 'Mean': mean_val, 'Median': media
                                          'Mode': mode_val, 'Is_Normal': is_normal},
                                         ignore_index=True)

# Print the statistics DataFrame
print(statistics_df)
```

```
       Column      Mean  Median      Mode Is_Normal
0   LIMIT_BAL  0.143298     0.0 -0.473684     False
1         AGE  0.106741     0.0 -0.357143     False
2   BILL_AMT1  0.455219     0.0 -0.355212     False
3   BILL_AMT2  0.462554     0.0 -0.352075     False
4   BILL_AMT3  0.469949     0.0 -0.352808     False
5   BILL_AMT4  0.465196     0.0 -0.368420     False
6   BILL_AMT5  0.457365     0.0 -0.374861     False
7   BILL_AMT6  0.452750     0.0 -0.357694     False
8    PAY_AMT1  0.891592     0.0 -0.524345     False
9    PAY_AMT2  0.955443     0.0 -0.484958     False
10   PAY_AMT3  0.841423     0.0 -0.437956     False
11   PAY_AMT4  0.910237     0.0 -0.403850     False
12   PAY_AMT5  0.877944     0.0 -0.396799     False
13   PAY_AMT6  0.941401     0.0 -0.385877     False
```

Inferences

We can see that, mean and median have closer values, hence we can now proceed with further steps of KNN

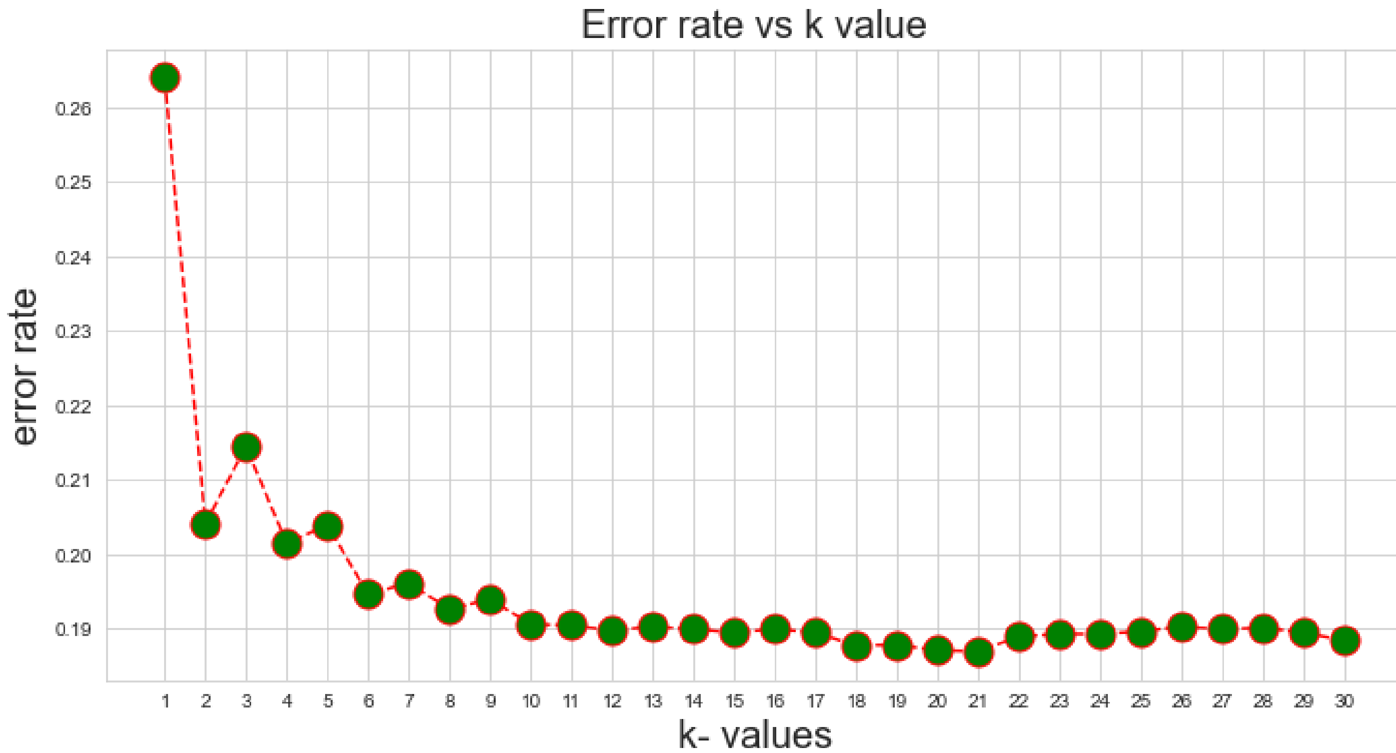
In [108…

```
from sklearn.neighbors import KNeighborsClassifier

error_rate = []

error_rate=[]
for i in range (1,31):  #Took the range of k from 1 to 30
    knn_clf=KNeighborsClassifier(n_neighbors=i)
    knn_clf.fit(X_train_scaled,y_train)
    predict_i=knn_clf.predict(X_test_scaled)
    error_rate.append(np.mean(predict_i!=y_test))
error_rate
```

Out[108… [0.264,
 0.20416666666666666,
 0.2145,
 0.2015,
 0.204,
 0.19466666666666665,
 0.19616666666666666,
 0.19266666666666668,
 0.194,
 0.19066666666666668,
 0.1905,
 0.18983333333333333,
 0.19033333333333333,
 0.19,
 0.1895,
 0.19,
 0.1895,
 0.18783333333333332,
 0.18783333333333332,
 0.18716666666666668,
 0.187,
 0.189,
 0.18933333333333333,
 0.18933333333333333,
 0.18966666666666668,
 0.19033333333333333,
 0.19,
 0.19016666666666668,
 0.1895,
 0.1885]

In [109…

```
#plotting the error rate vs k graph
plt.figure(figsize=(12,6))
plt.plot(range(1,31),error_rate,marker="o",markerfacecolor="green",
         linestyle="dashed",color="red",markersize=15)
plt.title("Error rate vs k value",fontsize=20)
plt.xlabel("k- values",fontsize=20)
plt.ylabel("error rate",fontsize=20)
plt.xticks(range(1,31))
plt.show()
```



Inferences We will k = 7, after which the error rate is not changing much

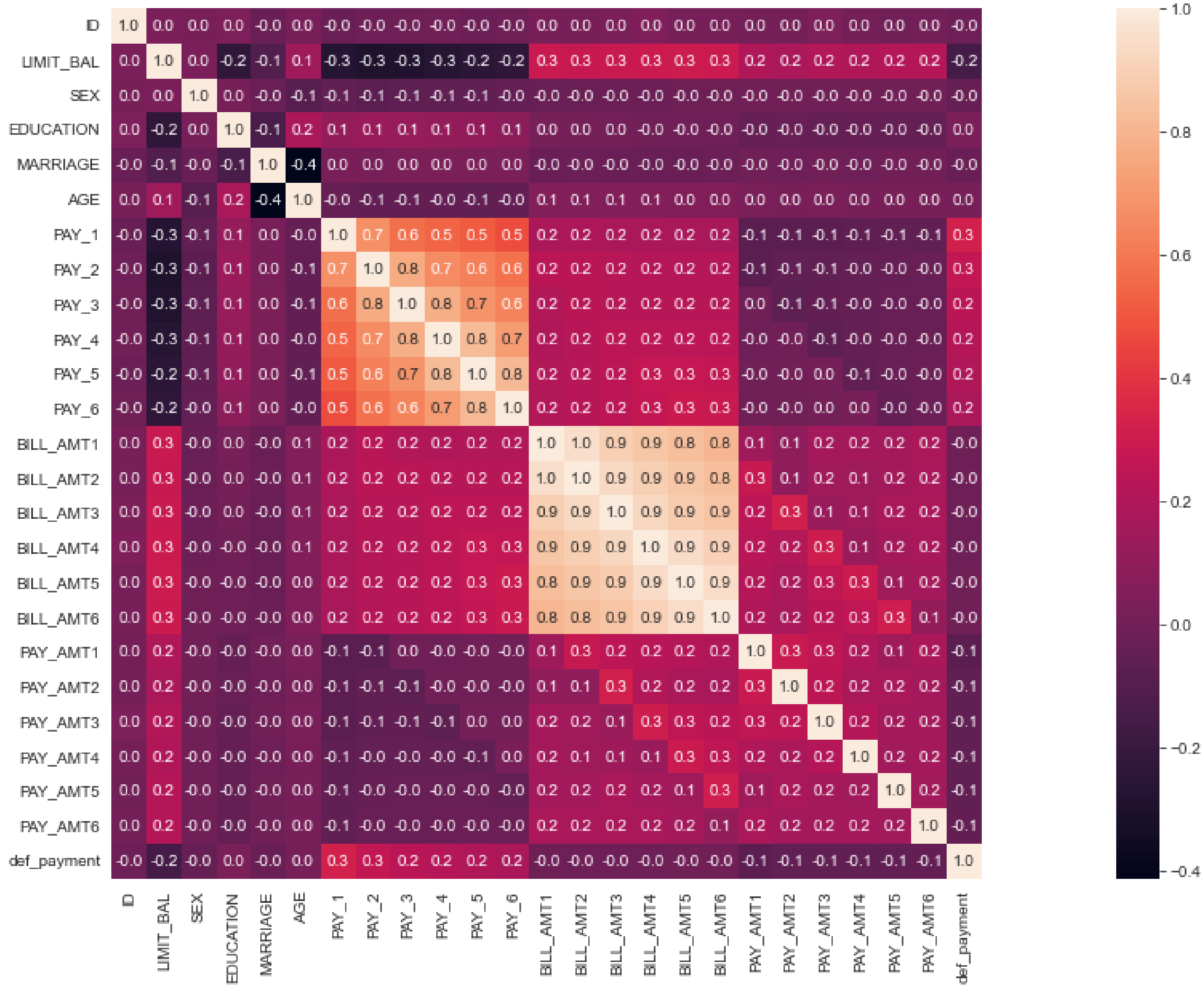
To see if the features are predictive - strong or weak, we will plot coorelation matrix

In [110…

```
correlation = df.corr()
plt.subplots(figsize=(30,10))
sns.heatmap( correlation, square=True, annot=True, fmt=".1f" )
```

Out[110…

```
<AxesSubplot:>
```



We see that for the target variable, the coorelation between feature variablle and target variable is on the lower side. In such a case, we would go ahead with Tree based algorithm.

In [111…

```
knn_clf = KNeighborsClassifier(n_neighbors=7)
knn_clf.fit(X_train_scaled,y_train)
y_pred = knn_clf.predict(X_test_scaled)
```

In [112…

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score

roc_knn = roc_auc_score(y_test,y_pred)
accuracy_knn = accuracy_score(y_test, y_pred)
precision_knn = precision_score(y_test, y_pred)
recall_knn = recall_score(y_test, y_pred)
f1_score_knn= f1_score(y_test, y_pred)

print("ROC of KNN model: {}".format(round(roc_knn,3)))
print("Accuracy of KNN model: {}".format(round(accuracy_knn,3)))
print("precision score of KNN model: {}".format(round(precision_knn,3)))
print("recall score of KNN model: {}".format(round(recall_knn,3)))
print("f1 score of KNN model: {}".format(round(f1_score_knn,3)))
```

```
ROC of KNN model: 0.642
Accuracy of KNN model: 0.804
precision score of KNN model: 0.585
```

```
recall score of KNN model: 0.355
f1 score of KNN model: 0.442
```

Inferences

We see that Accuracy of KNN model has gone down. This was exepected that KNN might not be a good fit for this problem statement because the predictors are not very strong - discussed earlier

We will comapare all 3 algorithms in next section.

In [113…

```
#displaying the confusion matrix for visualization purpose
knn_confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
knn_confusion_matrix
```

Out[113…

```
array([[4357,  330],
       [ 847,  466]], dtype=int64)
```

In [114…

```
TN, FP, FN, TP = knn_confusion_matrix.ravel()
# Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
TPR_KNN = TP / (TP + FN)
FPR_KNN= FP / (FP + TN)
print("True Positive Rate (TPR):", TPR_KNN)
print("False Positive Rate (FPR):", FPR_KNN)
```

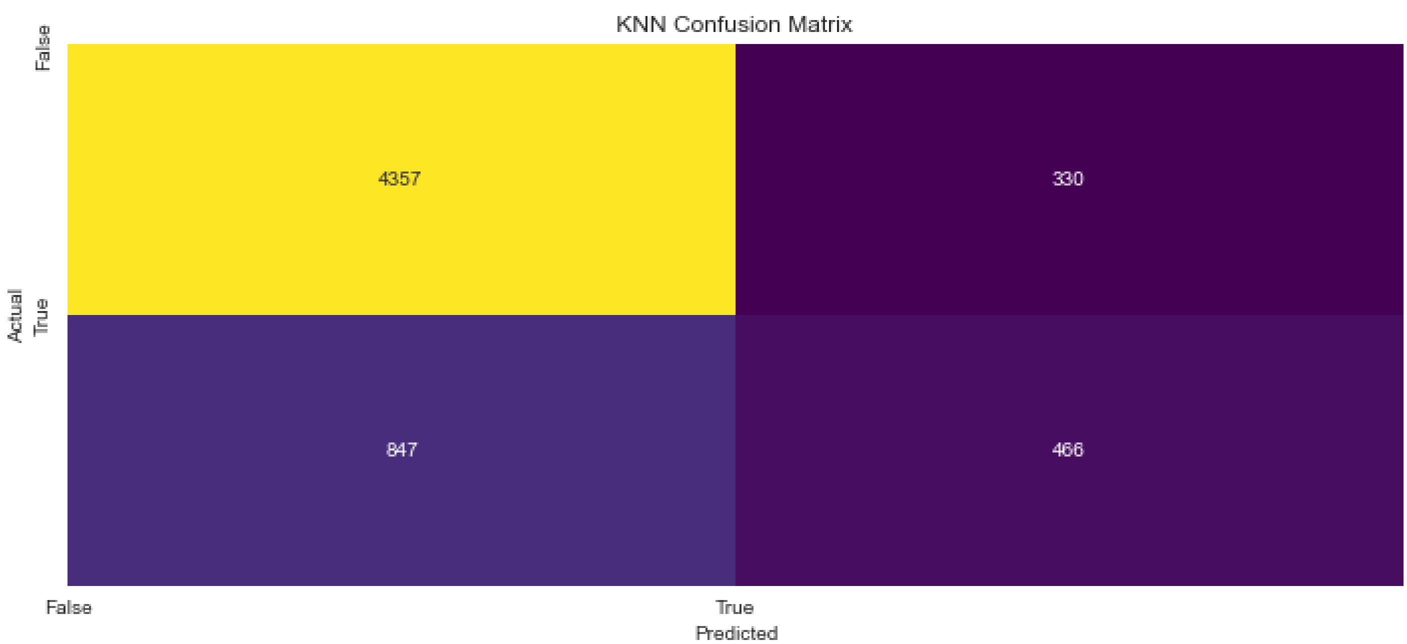
```
True Positive Rate (TPR): 0.3549124143183549
False Positive Rate (FPR): 0.07040751013441433
```

In [123…

```
plt.figure(figsize=(12, 5))

# Plot the first confusion matrix
sns.heatmap(knn_confusion_matrix, annot=True, fmt="d", cmap="viridis", cbar=False)
plt.title(" KNN Confusion Matrix ")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks([0, 1], ["False", "True"])
plt.yticks([0, 1], ["False", "True"])
```

Out[123…

```
([<matplotlib.axis.YTick at 0x138d601e460>,
  <matplotlib.axis.YTick at 0x138d8762e20>],
 [Text(0, 0, 'False'), Text(0, 1, 'True')])
```



7. Conclusion

Compare the performance of the ML techniques used.

Derive values for preformance study metrics like accuracy, precision, recall, F1 Score, AUC-ROC etc to compare the ML algos and plot them. A proper comparision based on different metrics should be done and not just accuracy alone, only then the comparision becomes authentic. You may use Confusion matrix, classification report, Word cloud etc as per the requirement of your application/problem.

Score 1 Mark

In [130…

```
from sklearn import metrics
##---------Type the code below this line------------------##
roc_auc_list = [roc_auc_score_dtc1,roc_auc_score_dtc2,
                roc_auc_score_rfc1,roc_auc_score_rfc2,roc_knn]
accuracy_list = [accuracy_score_dtc1, accuracy_score_dtc2,
                 accuracy_score_rfc1,accuracy_rfc2,accuracy_knn]
precision_list = [precision_score_dtc1,precision_score_dtc2,
                  precision_score_rfc1,precision_score_rfc2,precision_knn]
recall_list = [recall_score_dtc1, recall_score_dtc2,
               recall_score_rfc1, recall_score_rfc2, recall_knn]
f1_score_list = [f1_score_dtc1, f1_score_dtc2,
                 f1_score_rfc1,f1_score_rfc2,f1_score_knn]

all_metrices_dict = {'model' : ['DecisionTree','DecisionTree_Tuned', 'RandomForest',
                                'RandomForest_Tuned','KNN'],
                     'ROC':[roc_auc_score_dtc1,roc_auc_score_dtc2,
                            roc_auc_score_rfc1,roc_auc_score_rfc2,roc_knn],
                     'accuracy': [accuracy_score_dtc1, accuracy_score_dtc2,
                                  accuracy_score_rfc1,accuracy_rfc2,accuracy_knn],
                     'precision': [precision_score_dtc1,precision_score_dtc2,
                                   precision_score_rfc1,precision_score_rfc2,
                                   precision_knn],
                     'recall' :[recall_score_dtc1, recall_score_dtc2,
                                recall_score_rfc1, recall_score_rfc2, recall_knn],
                     'f1_score': [f1_score_dtc1, f1_score_dtc2,
                                  f1_score_rfc1,f1_score_rfc2,f1_score_knn]}
metrics_list_df = pd.DataFrame(all_metrices_dict)
metrics_list_df
```

Out[130…

	model	ROC	accuracy	precision	recall	f1_score
0	DecisionTree	0.643686	0.809500	0.613941	0.348819	0.444876
1	DecisionTree_Tuned	0.655814	0.821167	0.669014	0.361767	0.469600
2	RandomForest	0.648790	0.812333	0.624170	0.357959	0.454985
3	RandomForest_Tuned	0.648047	0.820167	0.676205	0.341965	0.454224
4	KNN	0.642252	0.803833	0.585427	0.354912	0.441916

Inferences

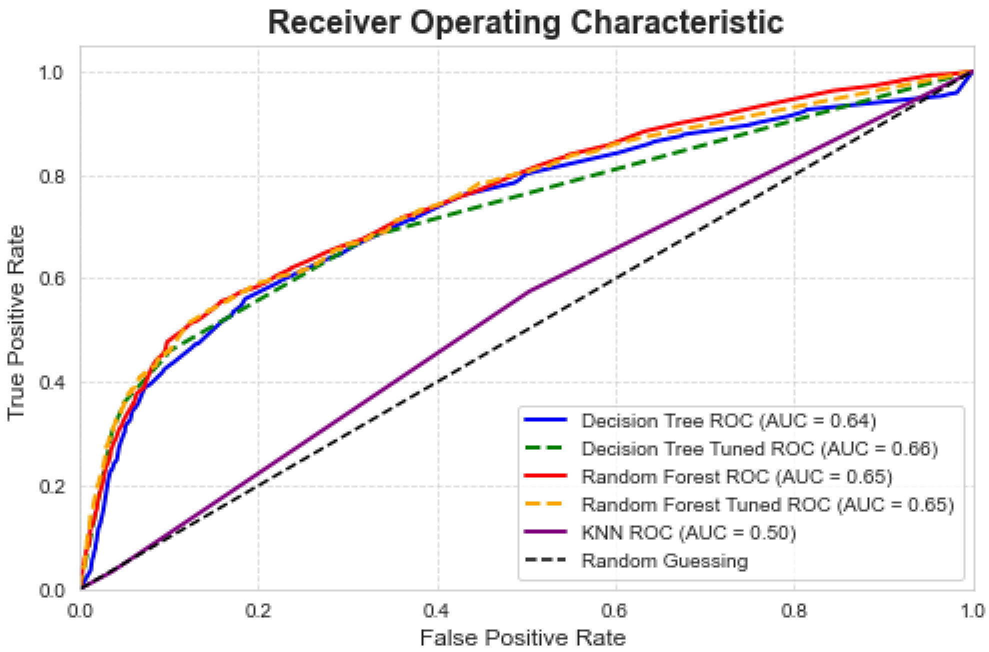
From the above table, we can clearly see that.

- **ROC is good for Decision Tree Tuned.**
- **Same goes with Accuracy of the model as well**, accuracy is highest for Decision Tree Tuned : 82.1167%

Plotting ROC Curve

In [132…

```
from sklearn import metrics

# Calculate predicted probabilities for multiple models
dc_pred = clf1.predict_proba(X_test)[:, 1]
dc_pred_tuned = clf_2.predict_proba(X_test)[:, 1]
rf_pred = rfc_1.predict_proba(X_test)[:, 1]
rf_pred_tuned = rfc_2.predict_proba(X_test)[:, 1]
knn_pred = knn_clf.predict_proba(X_test)[:, 1]

# List of models and their corresponding predicted probabilities
models = [clf1, clf_2, rfc_1, rfc_2, knn_clf]
model_predictions = [dc_pred, dc_pred_tuned, rf_pred, rf_pred_tuned, knn_pred]

# Labels for the models
model_labels = ['Decision Tree', 'Decision Tree Tuned', 'Random Forest',
                'Random Forest Tuned', 'KNN']

# Initialize an empty list to store ROC AUC values
roc_auc_values = []

# Plotting ROC curves
plt.figure(figsize=(8, 5))
m = range(5)

# Customize line styles and colors for the ROC curves
line_styles = ['-', '--', '-', '--', '-']
line_colors = ['blue', 'green', 'red', 'orange', 'purple']

for i in m:
    fpr, tpr, thresholds = metrics.roc_curve(y_test, model_predictions[i])
    auc = metrics.roc_auc_score(y_test, models[i].predict(X_test))
    roc_auc_values.append(auc)  # Append ROC AUC value to the list
    plt.plot(fpr, tpr, linestyle=line_styles[i], color=line_colors[i], linewidth=2,
             label='%s ROC (AUC = %0.2f)' % (model_labels[i], auc))  # Update label forma

# Add a dashed diagonal line for reference
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')

# Customize plot appearance
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate ', fontsize=12)
plt.ylabel('True Positive Rate ', fontsize=12)
plt.title('Receiver Operating Characteristic', fontsize=16, fontweight='bold')

# Add ROC AUC values to the legend
plt.legend(loc="lower right", fontsize=10)

plt.grid(True, linestyle='--', alpha=0.7)

# Show the ROC curve plot
plt.show()
```



Inferences

- We can see that, AUC is highest for Decision Tree Tuned model
- Higher the AUC value, higher is the discriminatory power
- Hence, we **recommend using Decision Tree (Tuned) Model**

8. Solution

What is the solution that is proposed to solve the business problem discussed in Section 1. Also share your learnings while working through solving the problem in terms of challenges, observations, decisions made etc.

Score 2 Marks

--------------Type the answers below this line--------------

Solution

We started our problem with the identification of deaulters next month. We worked on various models, and considering the accuracy, AUC value, F1 Score etc., we would like to fix ourselves with Decision Tree Model. We would **recommend the bank to use Decision Tree with hyper parameter tuning**. We have limited data, with large amount of data, the accuracy and other model parameters can be increased - with hyper paramter tuning. Further, given the computation power, tuning of all models with multiple iterations can be done

##---------Type the answer below this line------------------##

Learnings

There are multiple learnings that we had throughout this assignment

1. Importance of **business understanding** is the first thing to do - else we would have lost
2. **Data Cleaning, manipulation and pre-processing** has its own importance, and it's crucial without which we cannot proceed.
3. None of the pre-processing steps can be skipped in any ML project. Else the foundation would be lost
4. It's very important to **do the descriptive and diagonostic analytics** before getting into predictive analytics - this way we understood the dataset vis-a-vis business problem statement in a much better way
5. **Understanding of mathematics behind algorithm** was helpful, we could relate the things that were taught in other courses, and we could relate things here.

6. While model selection and building models might be an easy task - considering limited volume of data, but we need to be mindful of **computation cost and time**. In our project, while tuning parameters, it took significant amount of time, which means, computation cost would be high.
7. When the bank would deploy our solution, definitely they would get the accuracy that we have got, but at the same time, the team should **be mindful of computation cost**. Hence, that trade-off should be done (in consultation with business stakeholders) before deployment.
8. Overall, it was a great learning experience, we were able to apply almost all the concepts, that were taught in the class.

Thank you so much.

NOTE

All Late Submissions will incur a penalty of -2 marks. Do ensure on time submission to avoid penalty.

Good Luck!!!