# credit-card-default-pranali

May 20, 2024

# 1 Project Name Machine Learning Project on Credit Card data

Credit risk refers to the possibility of loss that a lender or investor may face due to the failure of a borrower to repay a loan or fulfill other financial obligations. It is the risk of default on a debt that may arise from a borrower's inability or unwillingness to pay back the money borrowed.

Credit risk is a major concern for banks, financial institutions, and investors who lend money or invest in securities, as it can lead to a reduction in the value of their investments or even to a loss of principal. To manage credit risk, lenders and investors often use credit scoring models, perform due diligence on borrowers, and set credit limits and collateral requirements.

Machine Learning models have been helping these companies to improve the accuracy of their credit risk analysis, providing a scientific method to identify potential debtors in advance.

**In this project, I'll built a credit risk model to predict the risk of client default**

## 1.1 The progress of notebook is organized as follow:

- Data Preprocessing; data prepration; and data visualization
- Features Engineering and Features selection
- Model Developement
- Model Evaluations

**Project Type - Classification** Pranali Yadav

# 2 Project Summary -

## 2.1 Variables:

There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender (1=male, 2=female)
- EDUCATION: (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: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment (1=yes, 0=no)

## 2.2 Import Libraries:

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', 100)

import warnings
warnings.filterwarnings("ignore")

from imblearn.over_sampling import SMOTE
from collections import Counter

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
```

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from xgboost import XGBClassifier

from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, roc_auc_score, plot_confusion_matrix,␣
 ↪plot_precision_recall_curve
```

### 2.3 Read the Data:

[2]:
```
df=pd.read_csv('d:\default of credit card clients (1).csv')
df
```

[2]:
```
          ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  \
0          1      20000    2          2         1   24      2      2     -1
1          2     120000    2          2         2   26     -1      2      0
2          3      90000    2          2         2   34      0      0      0
3          4      50000    2          2         1   37      0      0      0
4          5      50000    1          2         1   57     -1      0     -1
…        …          …  …          …         …  …    …      …      …
29995  29996     220000    1          3         1   39      0      0      0
29996  29997     150000    1          3         2   43     -1     -1     -1
29997  29998      30000    1          2         2   37      4      3      2
29998  29999      80000    1          3         1   41      1     -1      0
29999  30000      50000    1          2         1   46      0      0      0

       PAY_4  …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  \
0         -1  …          0          0          0         0       689
1          0  …       3272       3455       3261         0      1000
2          0  …      14331      14948      15549      1518      1500
3          0  …      28314      28959      29547      2000      2019
4          0  …      20940      19146      19131      2000     36681
…        …  …          …          …          …         …
29995      0  …      88004      31237      15980      8500     20000
29996     -1  …       8979       5190          0      1837      3526
29997     -1  …      20878      20582      19357         0         0
29998      0  …      52774      11855      48944     85900      3409
29999      0  …      36535      32428      15313      2078      1800

       PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6  default payment next month
0             0         0         0         0                           1
1          1000      1000         0      2000                           1
2          1000      1000      1000      5000                           0
3          1200      1100      1069      1000                           0
4         10000      9000       689       679                           0
…           …        …        …        …                         …
```

```
29995     5003     3047     5000     1000                           0
29996     8998      129        0        0                           0
29997    22000     4200     2000     3100                           1
29998     1178     1926    52964     1804                           1
29999     1430     1000     1000     1000                           1

[30000 rows x 25 columns]
```

## 2.4 Data Pre-processing and Data Visualization:

## 2.5 1) Data Pre-processing:

[4]:
```python
# First, we check if there are missing data:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
 5   AGE                         30000 non-null  int64
 6   PAY_0                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

- There are no missing data for our database. Next, we take look into more details to the data.
- "default.payment.next.month" is a feature and is the target variable we are trying to predict.

[5]:
```
df.describe().T
```

[5]:

| | count | mean | std | min |
|---|---|---|---|---|
| ID | 30000.0 | 15000.500000 | 8660.398374 | 1.0 |
| LIMIT_BAL | 30000.0 | 167484.322667 | 129747.661567 | 10000.0 |
| SEX | 30000.0 | 1.603733 | 0.489129 | 1.0 |
| EDUCATION | 30000.0 | 1.853133 | 0.790349 | 0.0 |
| MARRIAGE | 30000.0 | 1.551867 | 0.521970 | 0.0 |
| AGE | 30000.0 | 35.485500 | 9.217904 | 21.0 |
| PAY_0 | 30000.0 | -0.016700 | 1.123802 | -2.0 |
| PAY_2 | 30000.0 | -0.133767 | 1.197186 | -2.0 |
| PAY_3 | 30000.0 | -0.166200 | 1.196868 | -2.0 |
| PAY_4 | 30000.0 | -0.220667 | 1.169139 | -2.0 |
| PAY_5 | 30000.0 | -0.266200 | 1.133187 | -2.0 |
| PAY_6 | 30000.0 | -0.291100 | 1.149988 | -2.0 |
| BILL_AMT1 | 30000.0 | 51223.330900 | 73635.860576 | -165580.0 |
| BILL_AMT2 | 30000.0 | 49179.075167 | 71173.768783 | -69777.0 |
| BILL_AMT3 | 30000.0 | 47013.154800 | 69349.387427 | -157264.0 |
| BILL_AMT4 | 30000.0 | 43262.948967 | 64332.856134 | -170000.0 |
| BILL_AMT5 | 30000.0 | 40311.400967 | 60797.155770 | -81334.0 |
| BILL_AMT6 | 30000.0 | 38871.760400 | 59554.107537 | -339603.0 |
| PAY_AMT1 | 30000.0 | 5663.580500 | 16563.280354 | 0.0 |
| PAY_AMT2 | 30000.0 | 5921.163500 | 23040.870402 | 0.0 |
| PAY_AMT3 | 30000.0 | 5225.681500 | 17606.961470 | 0.0 |
| PAY_AMT4 | 30000.0 | 4826.076867 | 15666.159744 | 0.0 |
| PAY_AMT5 | 30000.0 | 4799.387633 | 15278.305679 | 0.0 |
| PAY_AMT6 | 30000.0 | 5215.502567 | 17777.465775 | 0.0 |
| default.payment.next.month | 30000.0 | 0.221200 | 0.415062 | 0.0 |

| | 25% | 50% | 75% | max |
|---|---|---|---|---|
| ID | 7500.75 | 15000.5 | 22500.25 | 30000.0 |
| LIMIT_BAL | 50000.00 | 140000.0 | 240000.00 | 1000000.0 |
| SEX | 1.00 | 2.0 | 2.00 | 2.0 |
| EDUCATION | 1.00 | 2.0 | 2.00 | 6.0 |
| MARRIAGE | 1.00 | 2.0 | 2.00 | 3.0 |
| AGE | 28.00 | 34.0 | 41.00 | 79.0 |
| PAY_0 | -1.00 | 0.0 | 0.00 | 8.0 |
| PAY_2 | -1.00 | 0.0 | 0.00 | 8.0 |
| PAY_3 | -1.00 | 0.0 | 0.00 | 8.0 |
| PAY_4 | -1.00 | 0.0 | 0.00 | 8.0 |
| PAY_5 | -1.00 | 0.0 | 0.00 | 8.0 |
| PAY_6 | -1.00 | 0.0 | 0.00 | 8.0 |
| BILL_AMT1 | 3558.75 | 22381.5 | 67091.00 | 964511.0 |
| BILL_AMT2 | 2984.75 | 21200.0 | 64006.25 | 983931.0 |

```
BILL_AMT3                    2666.25    20088.5    60164.75   1664089.0
BILL_AMT4                    2326.75    19052.0    54506.00    891586.0
BILL_AMT5                    1763.00    18104.5    50190.50    927171.0
BILL_AMT6                    1256.00    17071.0    49198.25    961664.0
PAY_AMT1                     1000.00     2100.0     5006.00    873552.0
PAY_AMT2                      833.00     2009.0     5000.00   1684259.0
PAY_AMT3                      390.00     1800.0     4505.00    896040.0
PAY_AMT4                      296.00     1500.0     4013.25    621000.0
PAY_AMT5                      252.50     1500.0     4031.50    426529.0
PAY_AMT6                      117.75     1500.0     4000.00    528666.0
default.payment.next.month      0.00        0.0        0.00         1.0
```

- Total 30,000 clients in the dataset;
- The average amount of credit card is (NT dollar) 167,484. Minimal amount a credit card can receive is (NT dollar)10000 while max value is (NT dollar)1M
- In average, the client who owns a credit card is 35.48 year old in our sample, the yougest age one can get a credit card is 21 while the most senior age is 79.
- 22.1% of credit card contracts that will default next month

### 2.5.1 Drop ID and rename "default.payment.next.month"

[6]:
```
df.drop(["ID"], axis=1, inplace= True)
df.rename(columns={"default.payment.next.month": "Default"}, inplace=True)
```

[7]:
```
df.columns
```

[7]:
```
Index(['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
       'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'Default'],
      dtype='object')
```

### 2.5.2 Drop repeated Categories:

Some categorical data have repeated categories. First, let check which features contain repeated catigories and then drop the repeated one:

[8]:
```
ca=['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2',
       'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
for i in ca:
    print (i,df[i].unique())
```

```
SEX [2 1]
EDUCATION [2 1 3 5 4 6 0]
MARRIAGE [1 2 3 0]
PAY_0 [ 2 -1  0 -2  1  3  4  8  7  5  6]
```

```
PAY_2 [ 2  0 -1 -2  3  5  7  4  1  6  8]
PAY_3 [-1  0  2 -2  3  4  6  7  1  5  8]
PAY_4 [-1  0 -2  2  3  4  5  7  6  1  8]
PAY_5 [-2  0 -1  2  3  5  4  7  8  6]
PAY_6 [-2  2  0 -1  3  6  4  7  8  5]
```

#### 2.5.3 EDUCATION FEATURE:

- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
- category 4,5,6 and 0 can Group into class 4

[9]:
```
df['EDUCATION'].replace({0:4,5:4,6:4}, inplace=True)
df.EDUCATION.value_counts()
```

[9]:
```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

#### 2.5.4 MARRIAGE FEATURE:

- Marital status (1=married, 2=single, 3=others)
- We'll group categories 1 into category 3

[10]:
```
df['MARRIAGE'].replace({0:3}, inplace=True)
df.MARRIAGE.value_counts()
```

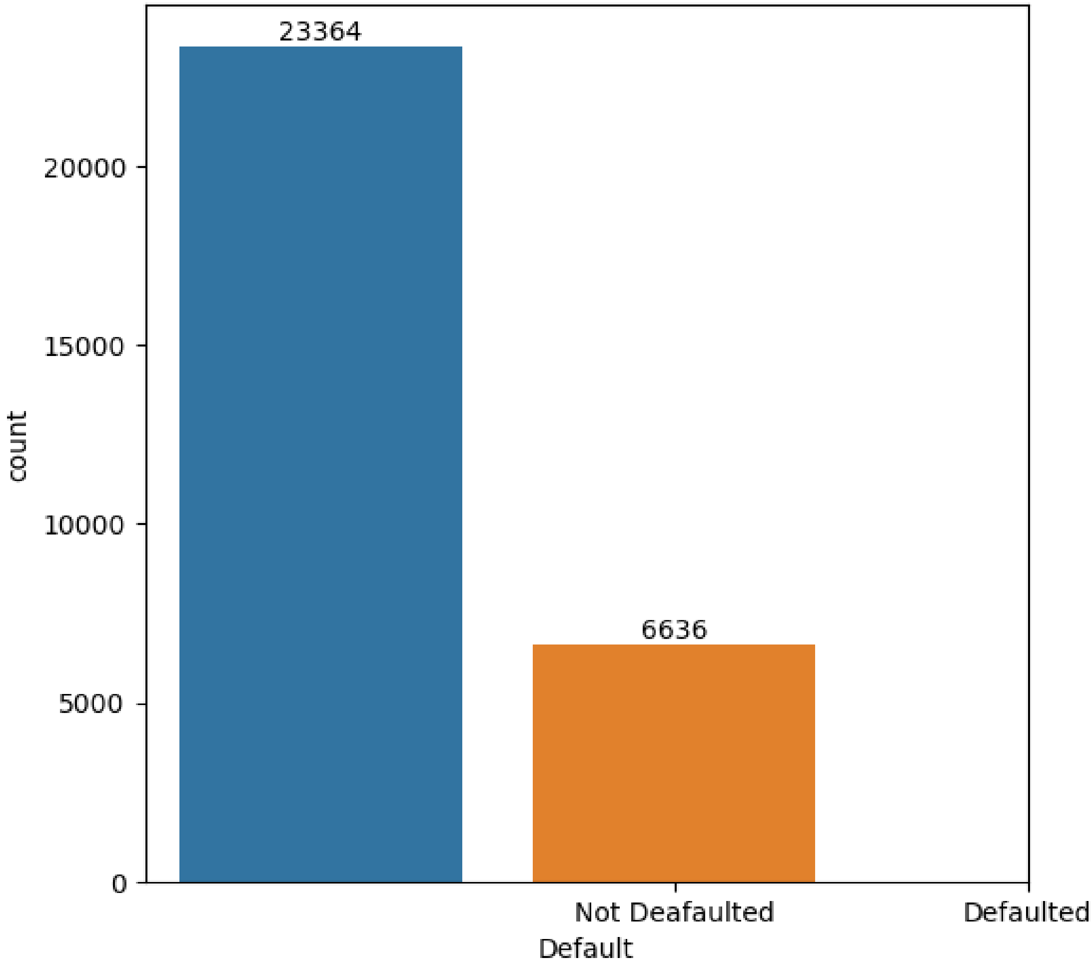
[10]:
```
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: int64
```

### 2.6 2) Data Visualization:

#### 2.6.1 Target Variable:

We'll visualize the target column "default" to figure out how imblance (balance) the data is

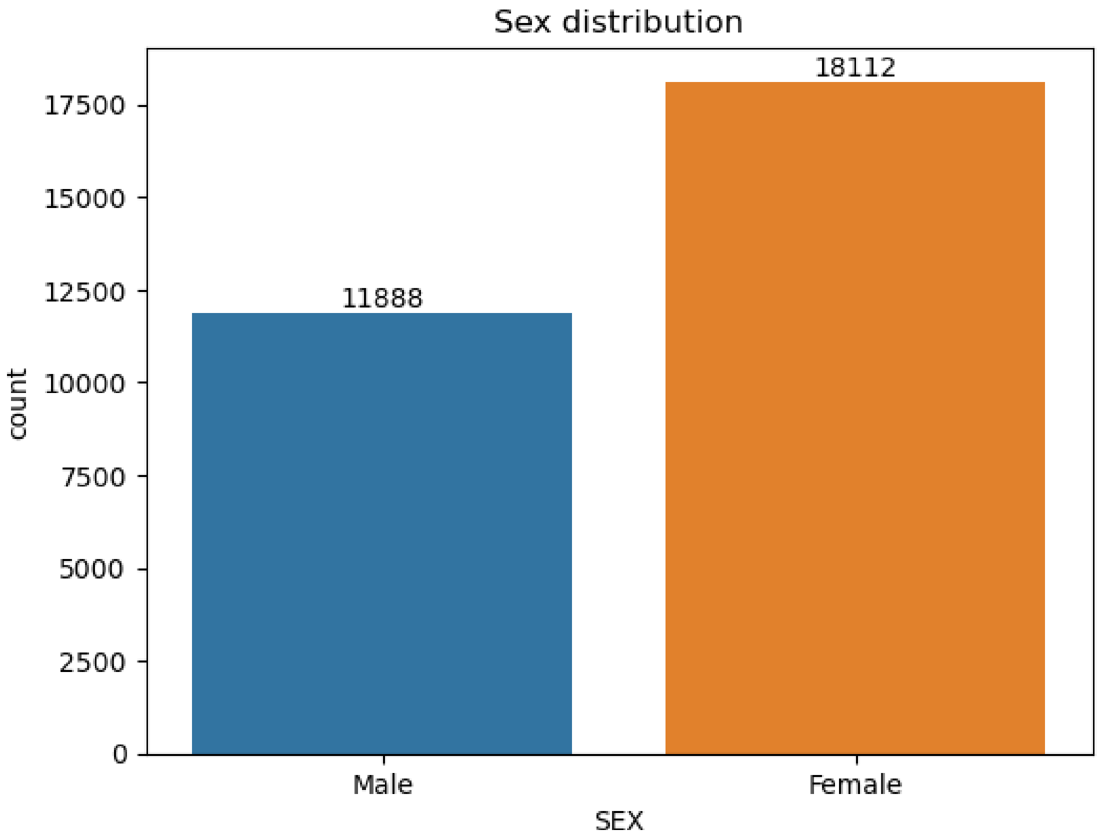
[11]:
```
plt.figure(figsize=(6,6))
ax=sns.countplot(x= df['Default'])
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([1,2], labels=["Not Deafaulted", "Defaulted"])
plt.show()
```

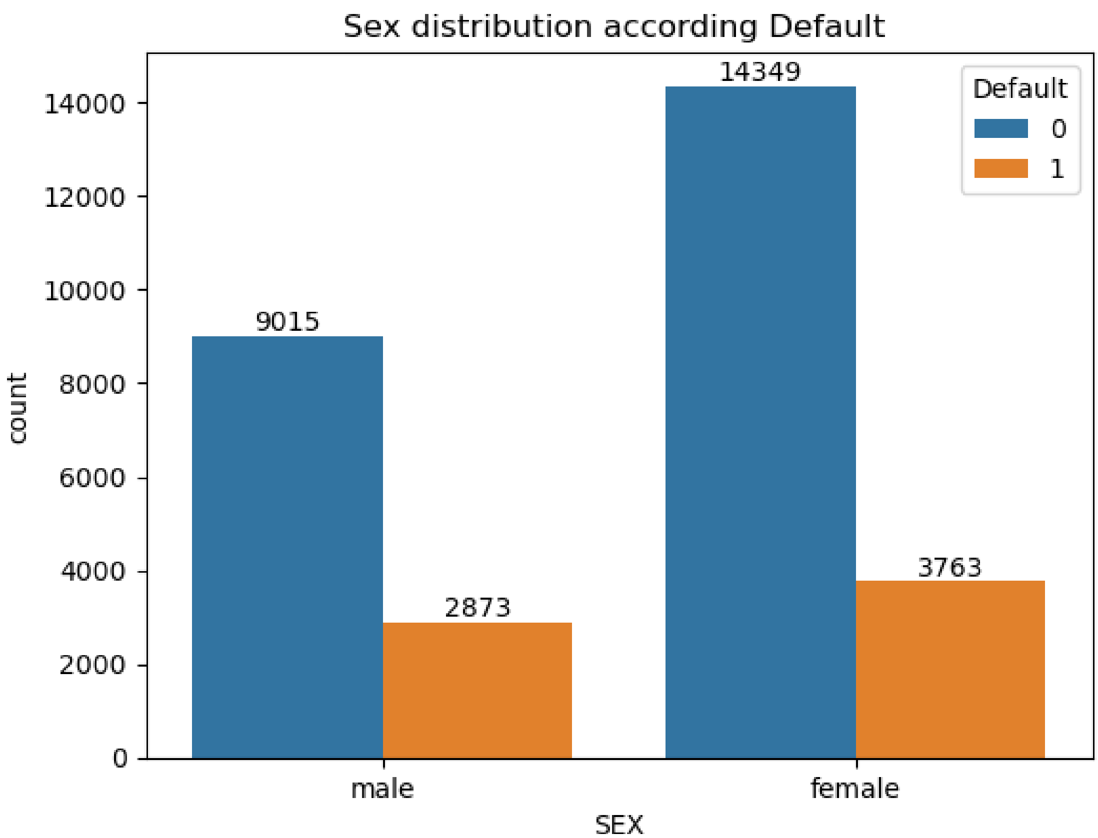



The data is quite imbalance which about 22% of clients will default next month.

#### 2.6.2 SEX Variable:

[12]:
```
ax=sns.countplot(x= df['SEX'])
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([0,1], labels=["Male", "Female"])
plt.title("Sex distribution")
plt.show()

ax=sns.countplot(data=df, x="SEX", hue="Default")
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([0,1], labels=["male", "female"])
plt.title("Sex distribution according Default ")
plt.show()
```

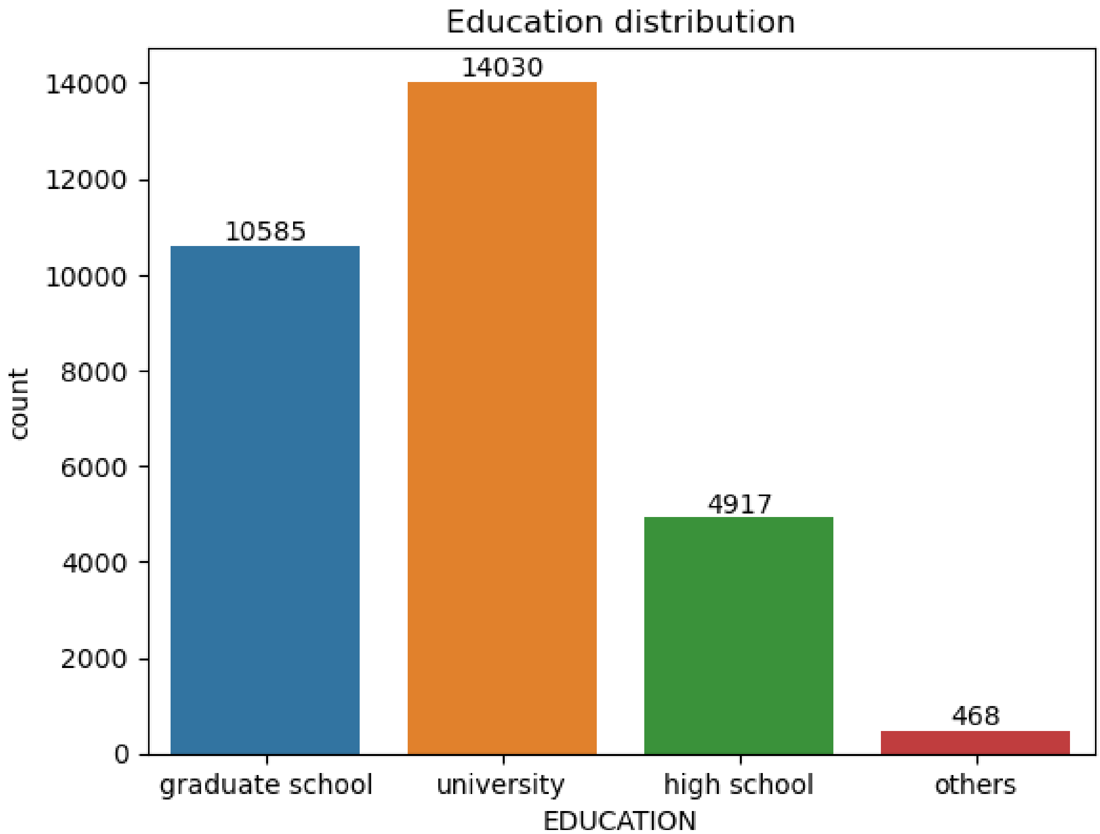

Sex distribution
18112
11888
17500
15000
12500
10000
7500
5000
2500
0
count
Male
Female
SEX

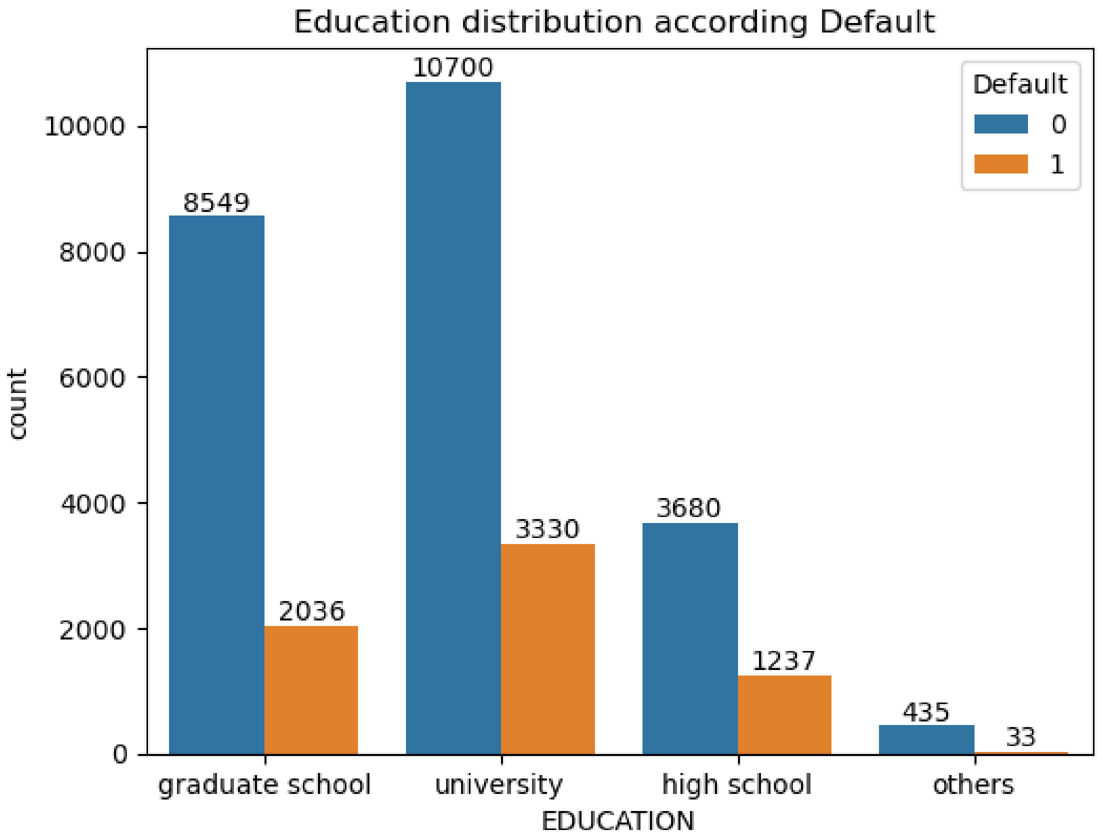



- More Female clients own credit card than Male client.
- 24% of male clients fraud credit card while the ratio for female is around 20%

### 2.6.3 EDUCATION VARIABLE:

[13]:
```
ax=sns.countplot(x= df['EDUCATION'])
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([0,1,2,3], labels=["graduate school", "university",'high␣
 ↪school','others'])
plt.title("Education distribution")
plt.show()

ax=sns.countplot(data=df, x="EDUCATION", hue="Default")
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([0,1,2,3], labels=["graduate school", "university",'high␣
 ↪school','others'])
plt.title("Education distribution according Default ")
plt.show()
```

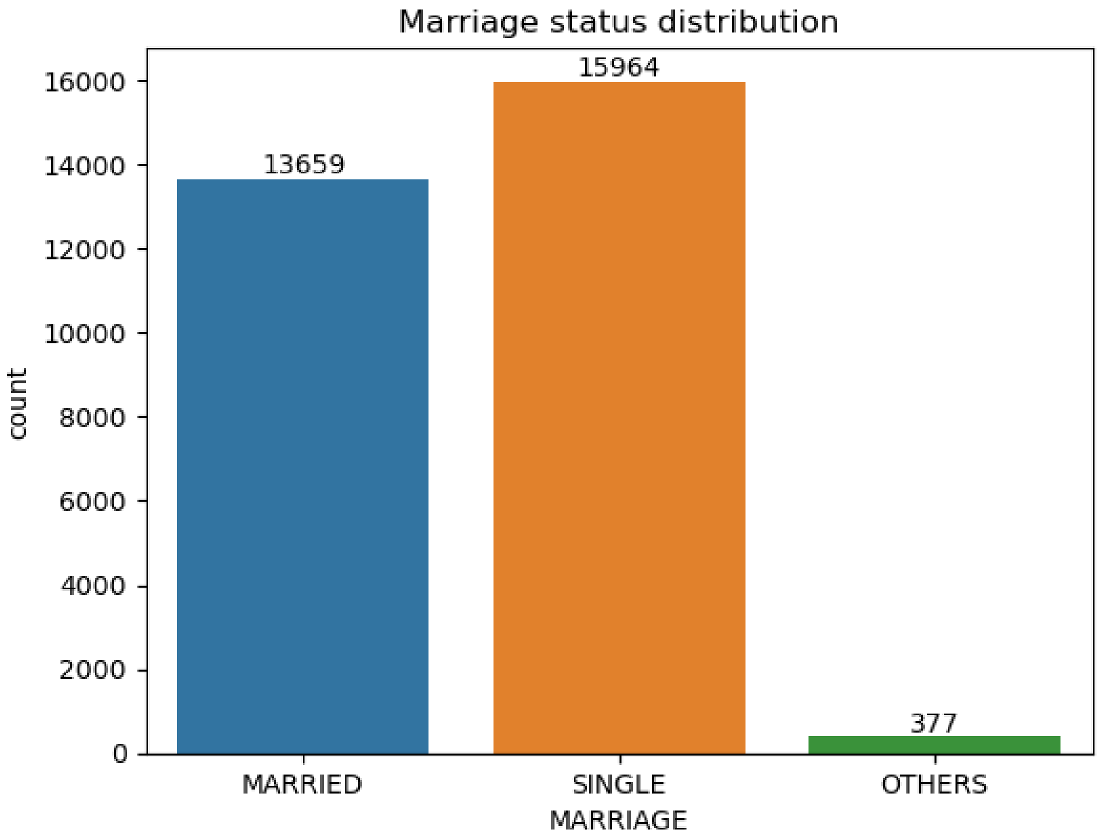

Education distribution
14030
10585
4917
468
count
14000
12000
10000
8000
6000
4000
2000
0
graduate school
university
high school
others
EDUCATION

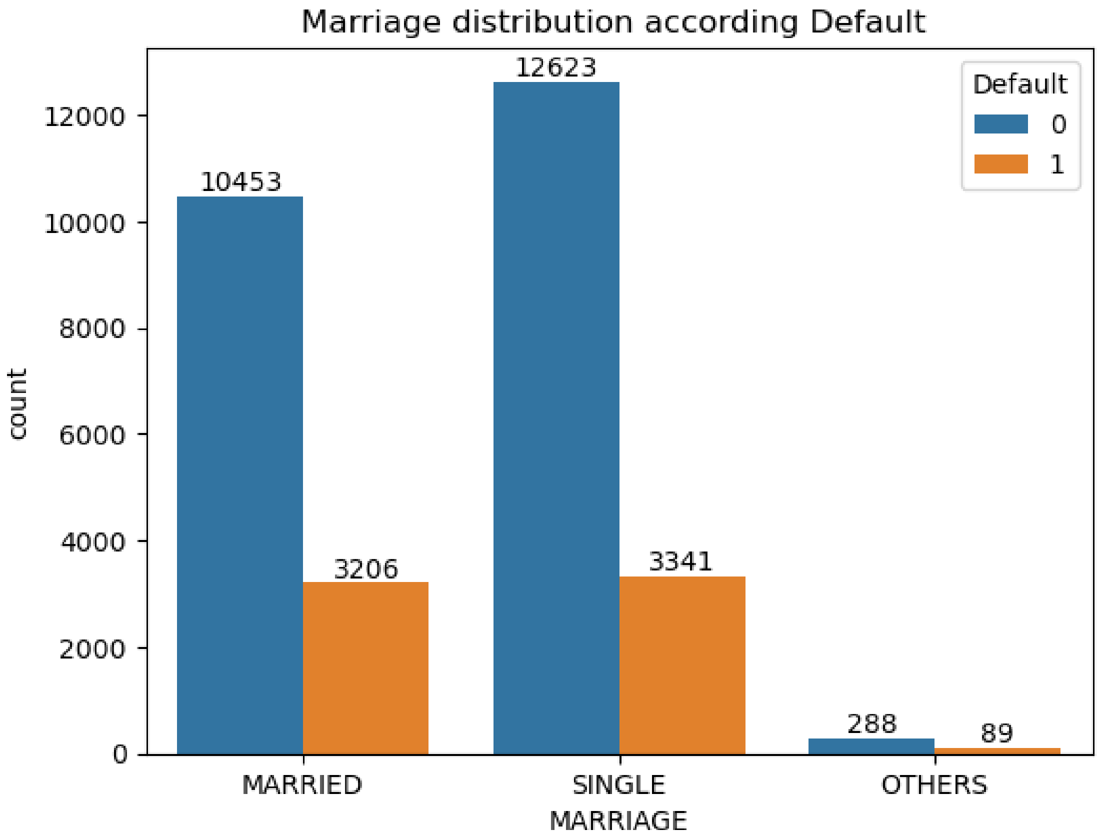



- University students are the group which highest number customers using credit cards (47%)
- High school students are the group which has highest fraud cases(25%), follow by university student(23%)

### 2.6.4 MARRIAGE STATUS VARIABLE:

[14]:
```
ax=sns.countplot(x= df['MARRIAGE'])
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([0,1,2], labels=["MARRIED", "SINGLE",'OTHERS'])
plt.title("Marriage status distribution")
plt.show()

ax=sns.countplot(data=df, x="MARRIAGE", hue="Default")
for label in ax.containers:
    ax.bar_label(label)
plt.xticks([0,1,2], labels=["MARRIED", "SINGLE",'OTHERS'])
plt.title("Marriage distribution according Default ")
plt.show()
```

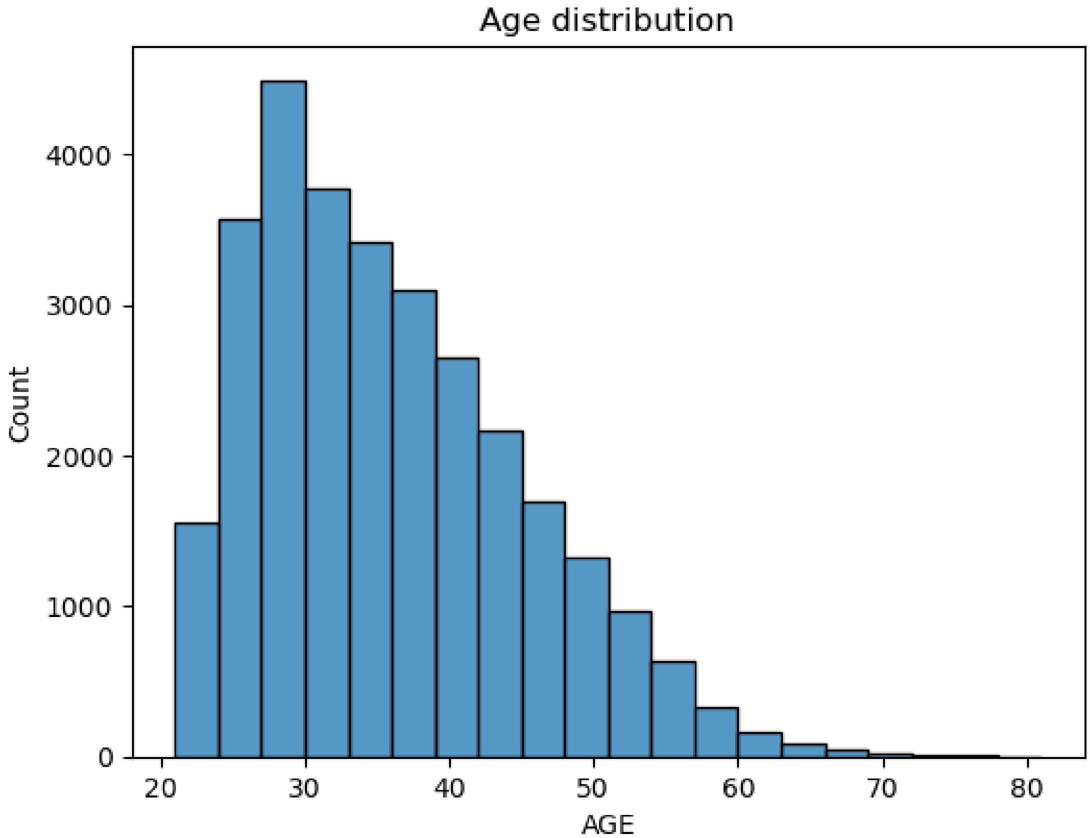

Marriage status distribution
16000
14000
12000
10000
8000
6000
4000
2000
0
count
15964
13659
377
MARRIED
SINGLE
OTHERS
MARRIAGE

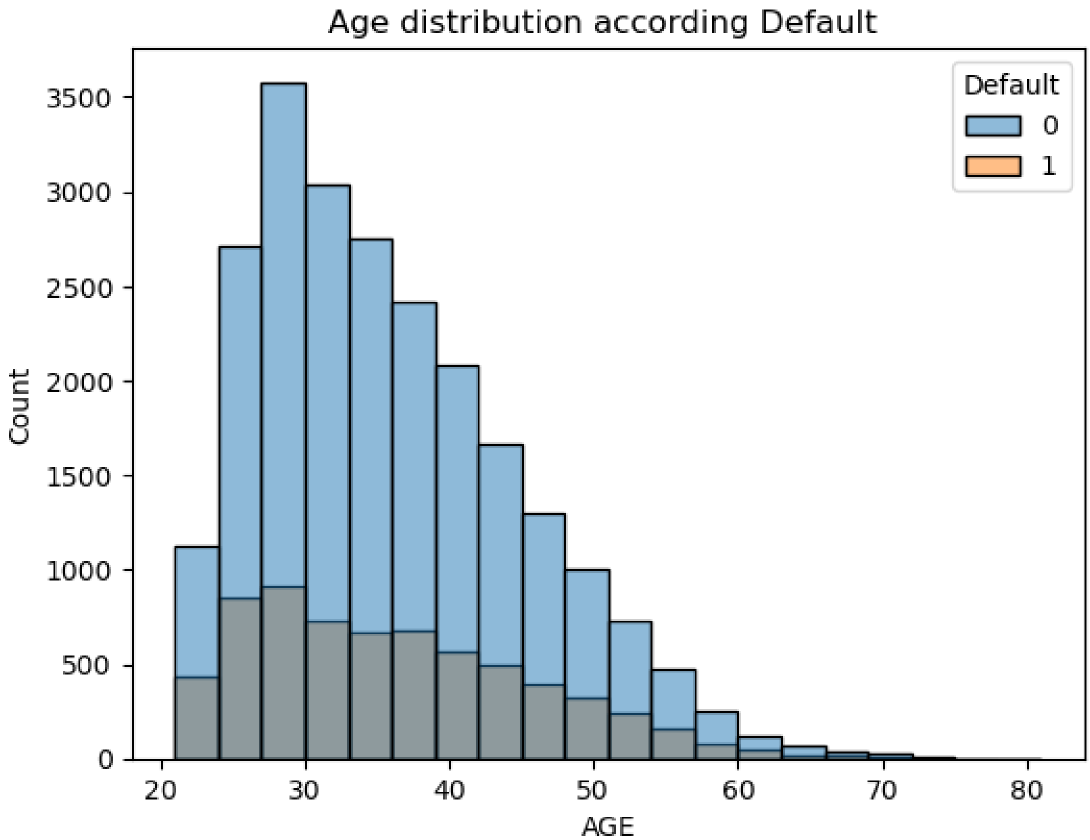



- Single is the group which highest number of customers using credit cards (53%)
- Married people are the group which has highest fraud cases(30%)

### 2.6.5 AGE VARIABLE:

[15]:
```
sns.histplot(data=df, x="AGE",binwidth=3)
plt.title("Age distribution")
plt.show()

sns.histplot(data=df, x="AGE", hue="Default", binwidth=3)
plt.title("Age distribution according Default ")
plt.show()
```


Age distribution
Count
AGE
0
1000
2000
3000
4000
20
30
40
50
60
70
80

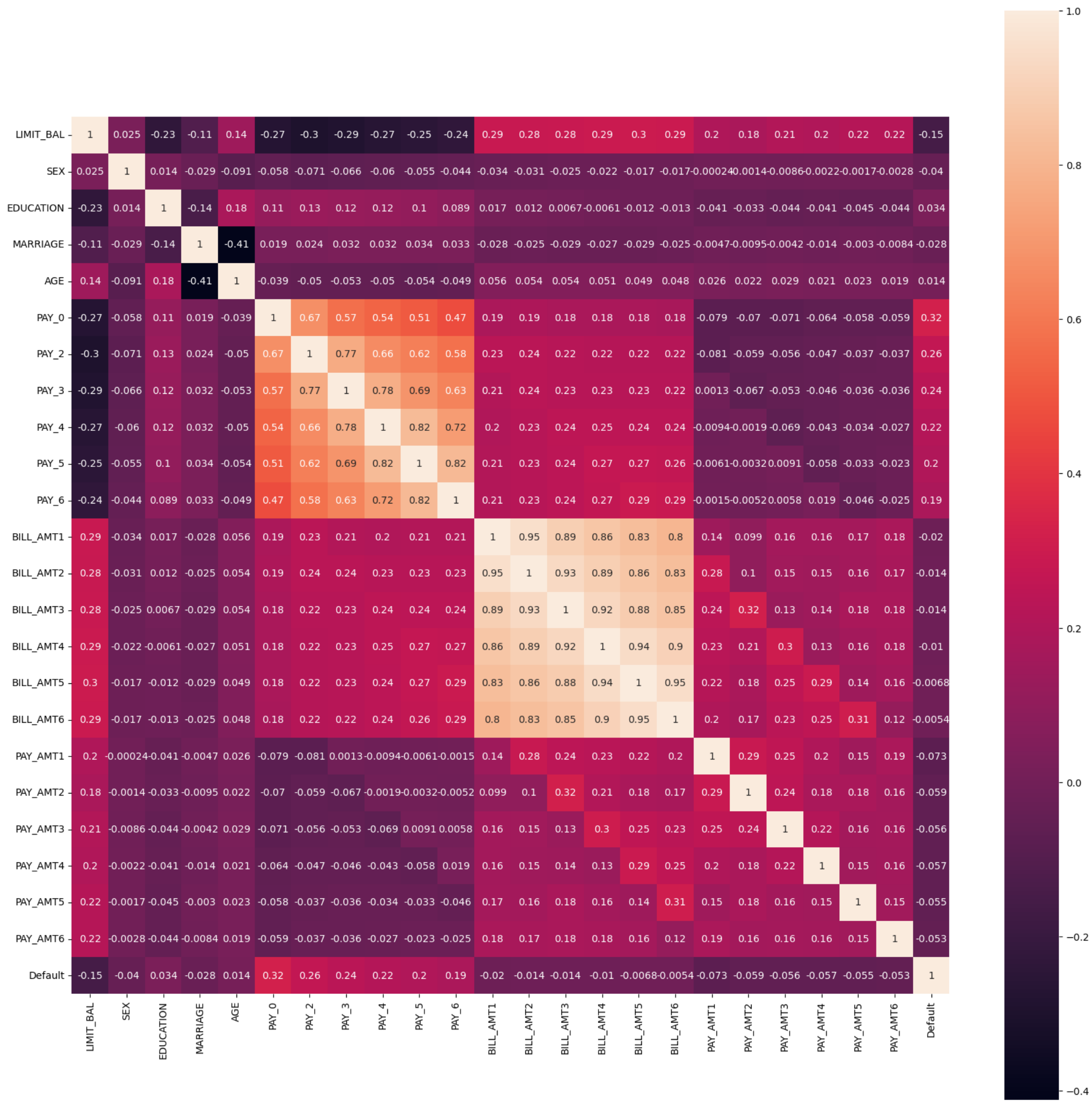



- Histogram is right-skewed meaning the older customers are less likely to use credit cards
- The main client is in their 30s
- Customers in their 30s are also the most prone to credit fraud

### 2.7 Correlation Analysis:

[18]:
```
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True,square = True)
```

[18]: <AxesSubplot:>




Correlation is high among PAY_0,2,3,4,5,6 and BILL_AMT1,2,3,4,5,6.

# 3 MODELLING:

[19]:
```
df.columns
```

[19]:
```
Index(['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
       'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'Default'],
      dtype='object')
```

### 3.0.1 Create target feaure and independent feature:

[20]:
```
X = df.drop(['Default'], axis=1)

y = df['Default']
X.head()
```

[20]:
```
   LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0    20000.0    2          2         1   24      2      2     -1     -1
1   120000.0    2          2         2   26     -1      2      0      0
2    90000.0    2          2         2   34      0      0      0      0
3    50000.0    2          2         1   37      0      0      0      0
4    50000.0    1          2         1   57     -1      0     -1      0

   PAY_5  PAY_6  BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4  BILL_AMT5  \
0     -2     -2     3913.0     3102.0      689.0        0.0        0.0
1      0      2     2682.0     1725.0     2682.0     3272.0     3455.0
2      0      0    29239.0    14027.0    13559.0    14331.0    14948.0
3      0      0    46990.0    48233.0    49291.0    28314.0    28959.0
4      0      0     8617.0     5670.0    35835.0    20940.0    19146.0

   BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6
0        0.0       0.0     689.0       0.0       0.0       0.0       0.0
1     3261.0       0.0    1000.0    1000.0    1000.0       0.0    2000.0
2    15549.0    1518.0    1500.0    1000.0    1000.0    1000.0    5000.0
3    29547.0    2000.0    2019.0    1200.0    1100.0    1069.0    1000.0
4    19131.0    2000.0   36681.0   10000.0    9000.0     689.0     679.0
```

[21]:
```
### Feature Engineering:

scaler= StandardScaler()
X= scaler.fit_transform(X)
```

## 3.1 Balancing the data

The SMOTE algorithm works like this:

You select a random sample from the minority group. You will determine the k nearest neighbours for the observations in this sample. Then, using one of those neighbours, you will determine the vector between the current data point and the chosen neighbour. The vector is multiplied by a random number between 0 and 1. You add this to the current data point to get the synthetic data point. This operation is essentially the same as moving the data point slightly in the direction of its neighbour. This ensures that your synthetic data point is not an exact replica of an existing data point, while also ensuring that it is not too dissimilar from known observations in your minority class.

[22]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.
↪20,random_state=42)
```

[23]:
```
# summarize class distribution

print("Before oversampling: ",Counter(y_train))
SMOTE= SMOTE()


X_train,y_train= SMOTE.fit_resample(X_train,y_train)

# summarize class distribution
print("After oversampling: ",Counter(y_train))
```

```
Before oversampling:  Counter({0: 18677, 1: 5323})
After oversampling:  Counter({0: 18677, 1: 18677})
```

## 3.2 Building Model:

### 3.2.1 Logistic Regression

### 3.2.2 Random Forest Classifier

### 3.2.3 Decision Tree

### 3.2.4 XGBoost Classifier

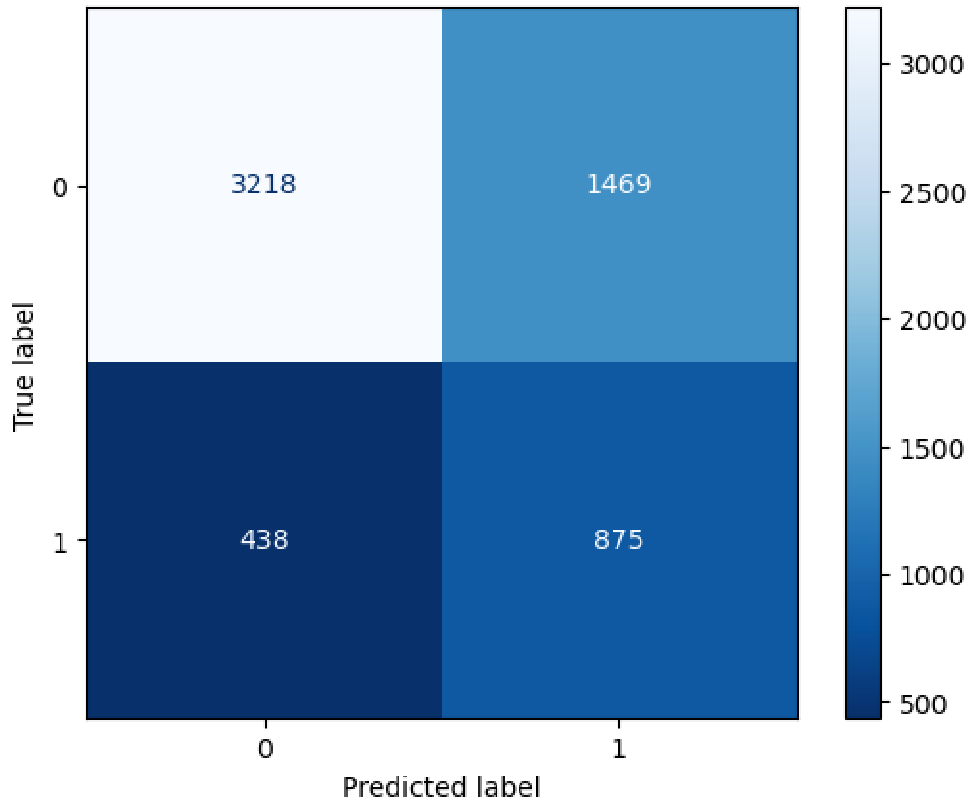
### 3.2.5 a) Logistic Regression

[24]:
```
logit= LogisticRegression()
logit.fit(X_train, y_train)

pred_logit= logit.predict(X_test)



print("Logit model's accuracy:", accuracy_score(y_test, pred_logit))

print(classification_report(y_test, pred_logit))

print('confusion matrix of logistic regression')
plot_confusion_matrix(logit, X_test, y_test, cmap="Blues_r")
```

```
Logit model's accuracy: 0.6821666666666667
              precision    recall  f1-score   support

           0       0.88      0.69      0.77      4687
           1       0.37      0.67      0.48      1313

    accuracy                           0.68      6000
   macro avg       0.63      0.68      0.62      6000
weighted avg       0.77      0.68      0.71      6000
```

```
confusion matrix of logistic regression
```

[24]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x75a267e6ec50>



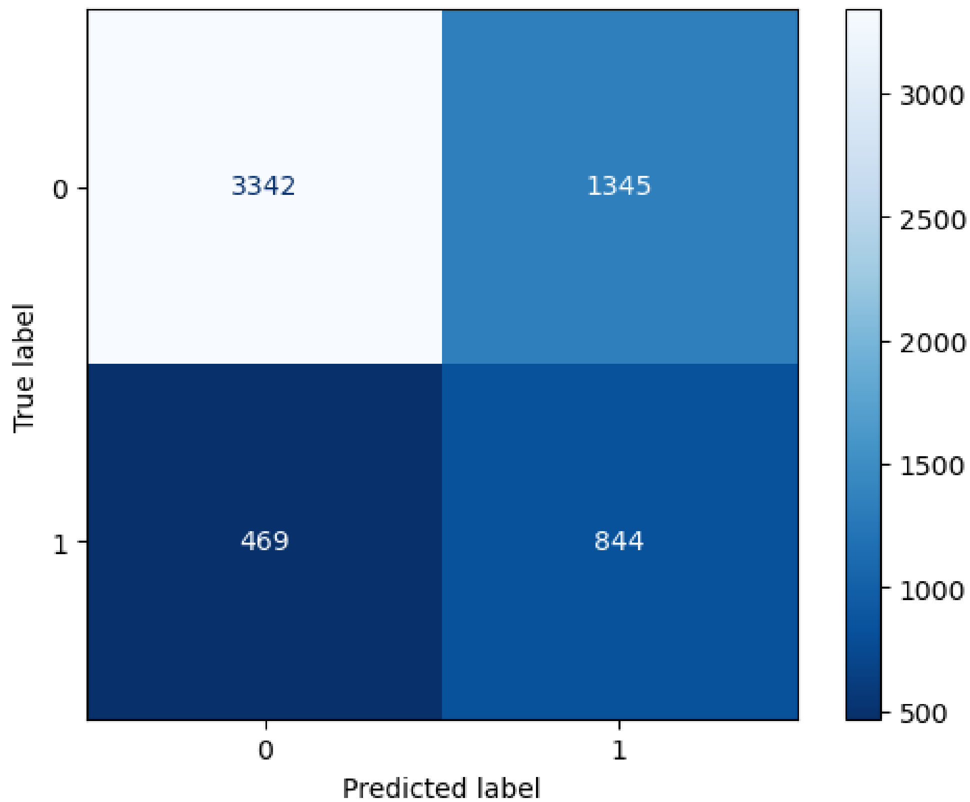

### 3.2.6 b) Decision Tree Classifier:

[25]:
```
Tree = DecisionTreeClassifier(criterion= 'gini', max_depth= 7,
                                     max_features= 9, min_samples_leaf= 2,
                                     random_state=0)
Tree.fit(X_train, y_train)
pred_tree = Tree.predict(X_test)
print('Decision Tree Accuracy :', accuracy_score(y_test,pred_tree))

print(classification_report(y_test, pred_tree))

print('confusion matrix of decision tree')
plot_confusion_matrix(Tree, X_test, y_test, cmap="Blues_r")
```

```
Decision Tree Accuracy : 0.6976666666666667
              precision    recall  f1-score   support
```

```
              0       0.88      0.71      0.79      4687
              1       0.39      0.64      0.48      1313

       accuracy                           0.70      6000
      macro avg       0.63      0.68      0.63      6000
   weighted avg       0.77      0.70      0.72      6000

confusion matrix of decision tree
```

[25]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x75a2672a85d0>



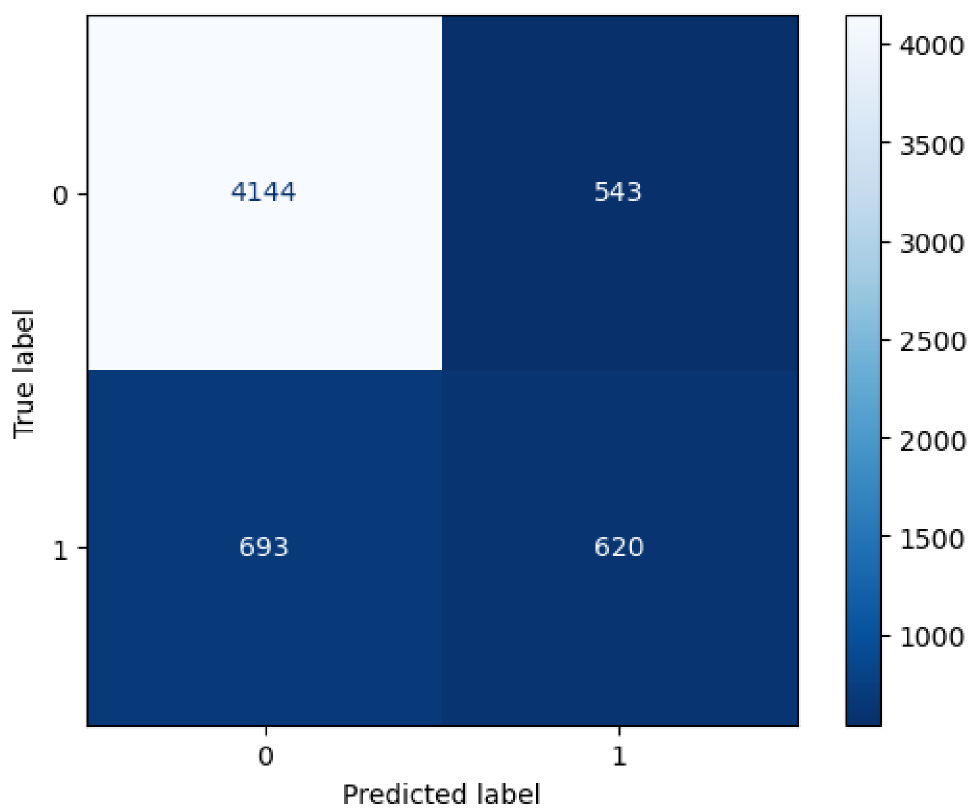

### 3.2.7 c) Random Forest:

[26]:
```
rf= RandomForestClassifier()

rf.fit(X_train,y_train)

pred_rf= rf.predict(X_test)
```

```python
print("Random Forest Accuracy is:", accuracy_score(y_test, pred_rf))

print(classification_report(y_test,pred_rf ))

plot_confusion_matrix(rf, X_test, y_test, cmap="Blues_r")
```

```
Random Forest Accuracy is: 0.794
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      4687
           1       0.53      0.47      0.50      1313

    accuracy                           0.79      6000
   macro avg       0.69      0.68      0.69      6000
weighted avg       0.79      0.79      0.79      6000
```

[26]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x75a267358e10>

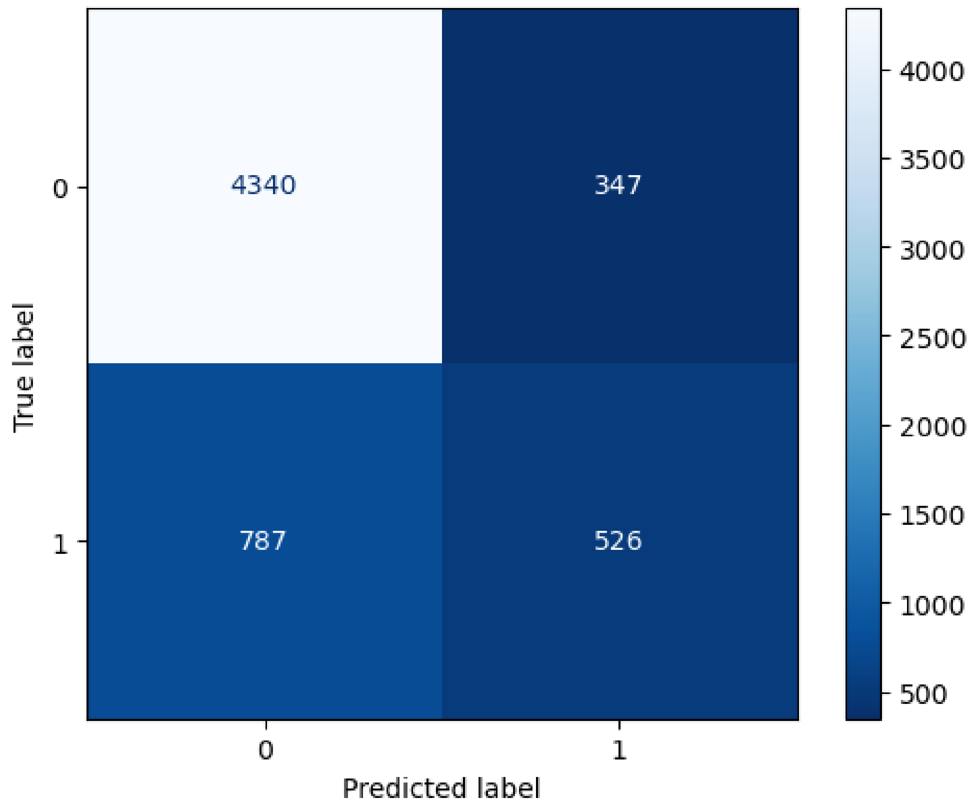


### 3.2.8 d) XGBoost:

[27]:
```
xgboost= xgb.XGBClassifier()


xgboost.fit(X_train,y_train)

xgboost_pred= xgboost.predict(X_test)

print("XGBoost Accuracy:", accuracy_score(y_test, xgboost_pred))
print(classification_report(y_test,xgboost_pred ))
plot_confusion_matrix(xgboost, X_test, y_test, cmap="Blues_r")
```

```
XGBoost Accuracy: 0.811
              precision    recall  f1-score   support

           0       0.85      0.93      0.88      4687
           1       0.60      0.40      0.48      1313

    accuracy                           0.81      6000
   macro avg       0.72      0.66      0.68      6000
weighted avg       0.79      0.81      0.80      6000
```

[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x75a26730e750>
```




### 3.2.9 Hyper parameter turning:

[28]:
```
## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

[29]:
```
random_search=RandomizedSearchCV(xgboost,param_distributions=params,n_iter=5,scoring='roc_auc'


random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

[29]:
```
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           callbacks=None, colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1,
                                           early_stopping_rounds=None,
                                           enable_categorical=False,
                                           eval_metric=None, gamma=0, gpu_id=-1,
                                           grow_policy='depthwise',
                                           importance_type=None,
                                           interaction_constraints='',
                                           learning_rate=0.300000012,
                                           max_bin=256,…
                                           n_estimators=100, n_jobs=0,
                                           num_parallel_tree=1,
                                           predictor='auto', random_state=0,
                                           reg_alpha=0, reg_lambda=1, …),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='roc_auc', verbose=3)
```

[30]:
```
# Best estimators:
random_search.best_estimator_
```

[30]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.4,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.15, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=12, max_leaves=0, min_child_weight=5,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
```

[31]:
```
#  best param
random_search.best_params_
```

[31]:
```
{'min_child_weight': 5,
 'max_depth': 12,
```

```
 'learning_rate': 0.15,
 'gamma': 0.0,
 'colsample_bytree': 0.4}
```

[32]:
```
classifier=XGBClassifier(objective='binary:logistic',
                                  min_child_weight=3,
                                 max_depth=10,
                                 learning_rate=0.25,
                                 gamma=0.1,
                                  colsample_bynode=1,
                                  colsample_bytree=0.4,
                                  use_label_encoder=False)

# Fitting the model
classifier.fit(X_train,y_train)
```

[32]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.4,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.1, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.25, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=10, max_leaves=0, min_child_weight=3,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
```

[33]:
```
# Predicting model
hyper_pred= classifier.predict(X_test)

print("The accuracy of the model is:", accuracy_score(y_test, hyper_pred))
```

```
The accuracy of the model is: 0.8058333333333333
```

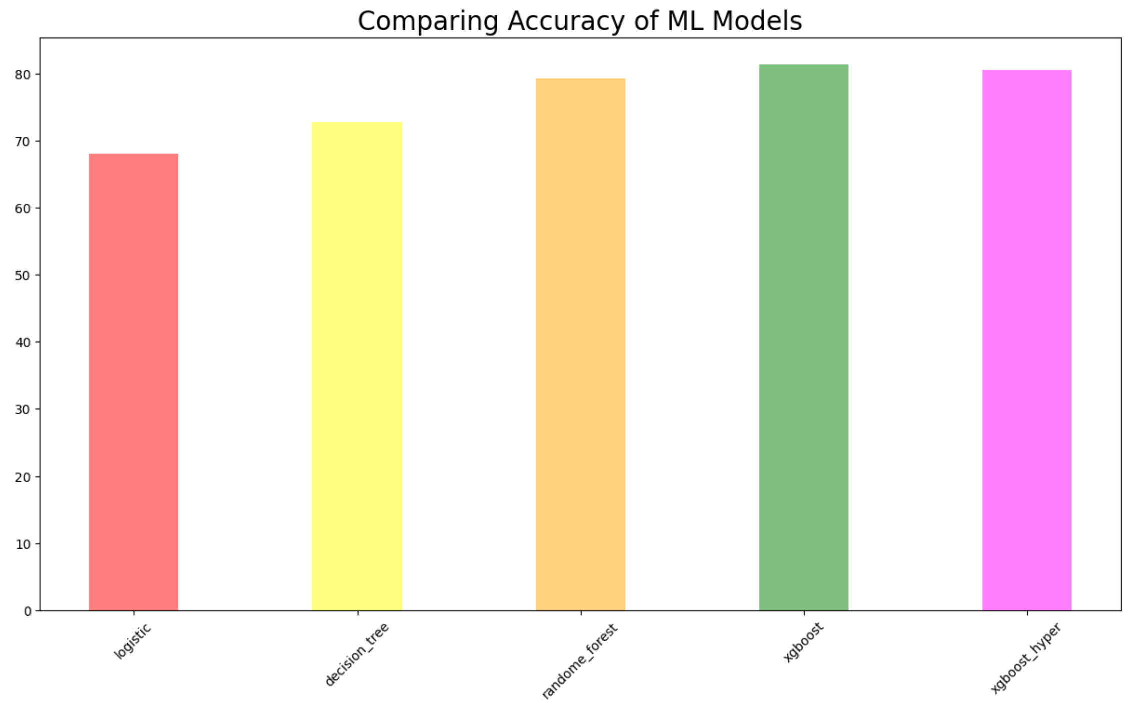
### 3.3 Compare Model Performance:

[35]:
```
data = {'logistic':68.08,
                'decision_tree':72.83,
               'randome_forest': 79.35,
                'xgboost': 81.35,
               'xgboost_hyper':80.58}
courses = list(data.keys())
values = list(data.values())
```

[36]:
```
plt.figure(figsize=(15,8))
plt.title('Comparing Accuracy of ML Models',fontsize=20)
colors=['red','yellow','orange','green','magenta'
```

```
    ]
plt.bar(courses, values, color =colors,alpha=0.5,width = 0.4)
plt.xticks(rotation = 45)
```

[36]:
```
([0, 1, 2, 3, 4],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```




### 3.3.1 Conclusion:

- In this project, we first check for data unbalancing, visualize the feaure and investigate in the relationship between different feature to find the strongest predictors of default payment
- We then run different 5 ML models in order to find the best model for detecting credit default:
- Logistic model with 68.08% accuracy,
- Decision_tree model with 72.83% accuracy,
- Randome_forest model with 79.35% accuracy,
- XGboost model with 81.35% accuracy,
- XGboost hyperparameter model with 80.58% accuracy

Among all the ML model we use to predict the default credit card, XGboost is the best model with highest accuracy score

[ ]:
```

```