# ARTIFICIAL INTELLIGENCE AND DEEP LEARNING

**Credit Card Default Case**

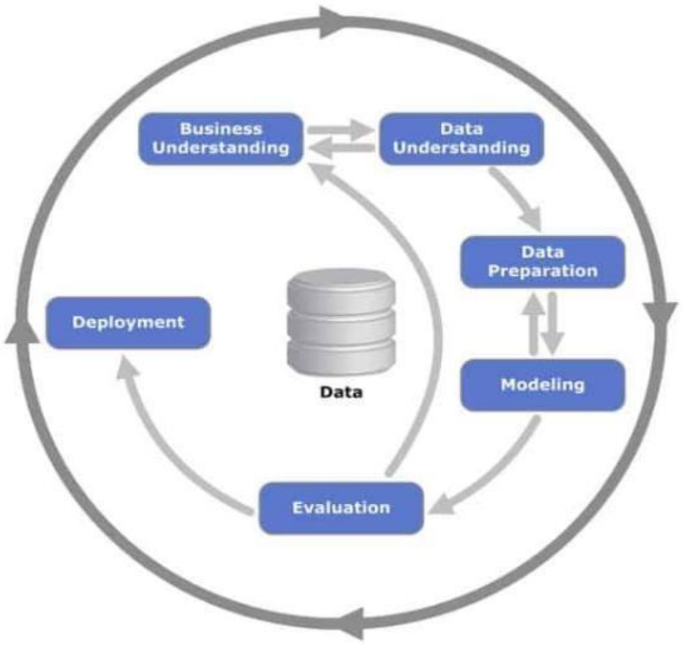
**Submitted by:**
Subham Sarangi

# Table of Contents

## CRISP DM Structure

CRISP-DM is a proven, extensively robust, and commonly used method by data scientists for planning, organizing, and executing data mining projects. This method is organised into six steps, which are as follows:

- Business Understanding
- Data Understanding
- Data Preparation
- Data Modelling
- Evaluation
- Deployment



## Business Understanding

Financial institutions increasingly rely on models and algorithms to forecast losses due to client defaults in order to achieve these criteria, for internal risk management and regulatory obligations, financial institutions must assess the risks in their credit portfolios. Therefore, one of the main initiatives of quantitative risk management groups inside these organisations is to build models that are sufficiently accurate and reliable.

**Credit Card Default Issue**

A single missed credit card payment does not constitute a default. A payment default happens when you fail to pay the Minimum Amount Due on your credit card for several months in a row. The card issuer usually sends the default notification after 6 consecutive missed payments. The bank, however, has the last say.

**Problem Statement**

A Taiwanese credit card company wants to improve its ability to predict customer default and identify the major factors that affect this risk. This study aims to discover the main factors that influence the chance of credit card default by utilising certain supervised machine learning methods. When you have fallen far behind on your credit card payments, you enter credit card default. Taiwan's card-issuing banks overissued cash and credit cards to ineligible applicants in an effort to gain market dominance. In addition, most cardholders abused their credit cards for consumption, regardless of their capacity to pay back the balance, and racked up significant debt.

The objective is to pinpoint the crucial elements and forecast a credit card default using client data and previous transactions. The issuer would then be allowed to choose who gets credit cards and what credit limits they'll give. Additionally, it aids issuers or companies in better comprehending their present and potential clients, leading future strategies and plans for providing their clients with specialised financing solutions. Based on the customer's payment history, demographics, and default status during the previous six months, this may be projected.

## Data Understanding

The dataset was downloaded in.csv file from UCI repository (URL: https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients), a dataset repository, where it was obtained. This dataset contains billing statements, credit history, payment history, and default information for Taiwanese credit card users or customers from April, 2005 to September, 2005.

Here, we are preparing to understand the problems faced by the banks, when a credit card is being issued to the customers to avoid the problem of default. The project is being prepared to understand, whether the credit card customer will make payment default in the next month or not based on number of features:

The dataset employs the binary variable **default.payment.next.month** as response variable. It indicates whether or not the credit card holders are defaulters next month ({Yes}=1, {No}=0). Then, the following 25 variables are used as explanatory variables.

**Attributes of the dataset:**

**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 of clients in September, 2005 (-2: No consumption; -1: Paid in full; 0: The use of revolving credit; 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 of clients in August, 2005 (scaling as above)

**PAY_3:** Repayment status of clients in July, 2005 (scaling as above)

**PAY_4:** Repayment status of clients in June, 2005 (scaling as above)

**PAY_5:** Repayment status of clients in May, 2005 Status (scaling as above)

**PAY_6:** Repayment status of clients in April, 2005 (scaling as above)

**BILL_AMT1:** Billed amount of clients for September, 2005 (NT dollars)

**BILL_AMT2:** Billed amount of clients for August, 2005 (NT dollars)

**BILL_AMT3:** Billed amount of clients for July, 2005 (NT dollars)

**BILL_AMT4:** Billed amount of clients for June, 2005 (NT dollars)

**BILL_AMT5:** Billed amount of clients for May, 2005 (NT dollars)

**BILL_AMT6:** Billed mount of clients for April, 2005 (NT dollars)

**PAY_AMT1:** Previous payment done in September, 2005 (NT dollars)

**PAY_AMT2:** Previous payment done in August, 2005 (NT dollars)

**PAY_AMT3:** Previous payment done in July, 2005 (NT dollars)

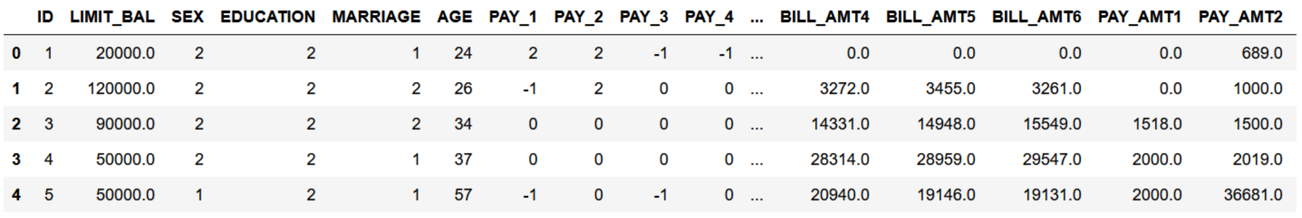
**PAY_AMT4:** Previous payment done in June, 2005 (NT dollars)

**PAY_AMT5:** Previous payment done in May, 2005 (NT dollars)

**PAY_AMT6:** Previous payment done in April, 2005 (NT dollars)

**default.payment.next.month:** default payment (1 = yes, 0 = no)

## DataSet Showcasing:



| | ID | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_1 | PAY_2 | PAY_3 | PAY_4 | ... | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT1 | PAY_AMT2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 20000.0 | 2 | 2 | 1 | 24 | 2 | 2 | -1 | -1 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 689.0 |
| **1** | 2 | 120000.0 | 2 | 2 | 2 | 26 | -1 | 2 | 0 | 0 | ... | 3272.0 | 3455.0 | 3261.0 | 0.0 | 1000.0 |
| **2** | 3 | 90000.0 | 2 | 2 | 2 | 34 | 0 | 0 | 0 | 0 | ... | 14331.0 | 14948.0 | 15549.0 | 1518.0 | 1500.0 |
| **3** | 4 | 50000.0 | 2 | 2 | 1 | 37 | 0 | 0 | 0 | 0 | ... | 28314.0 | 28959.0 | 29547.0 | 2000.0 | 2019.0 |
| **4** | 5 | 50000.0 | 1 | 2 | 1 | 57 | -1 | 0 | -1 | 0 | ... | 20940.0 | 19146.0 | 19131.0 | 2000.0 | 36681.0 |

**Rows and Columns:**

```
print("Default Credit Card Clients data -  rows:",dataSet.shape[0]," columns:", dataSet.shape[1])
```

```
Default Credit Card Clients data -  rows: 30000  columns: 25
```

```
print("Total number of elements in DataSet:",dataSet.size)
```

```
Total number of elements in DataSet: 750000
```

**Dataset.shape**:  showcases the dimensionality of the DataFrame.

The dataset has 30000 rows and 25 columns

**Dataset.size:**  exhibits an integer representing the number of elements present in this object.

A total number of elements in the dataset:750000

## Exploratory Data Analysis

The initial phase in the data analysis process is exploratory data analysis (EDA). EDA is used by researchers and data analysts to comprehend and summarize the contents of a dataset, either in order to answer a specific question, or to prepare for more advanced statistical modelling in later phases of data analysis.

### Descriptive Statistics

**Use of Describe () to understand the variables further**

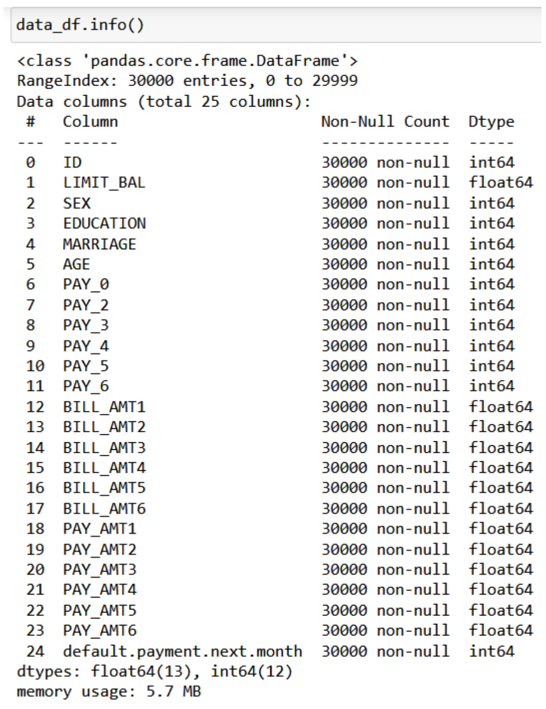
The dataset.describe() method returns a description of the data in the DataFrame dataset.

```
dataSet.describe()
```

| | ID | LIMIT_BAL | AGE | BILL_AMT1 | BILL_AMT2 | BILL_AMT3 | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 30000.000000 | 30000.000000 | 30000.000000 | 27992.000000 | 27494.000000 | 2.713000e+04 | 26805.000000 | 26494.000000 | 25980.000000 | 24751.00000 |
| **mean** | 15000.500000 | 167484.322667 | 35.485500 | 54897.825343 | 53661.608169 | 5.198653e+04 | 48419.640701 | 45645.883181 | 44886.559353 | 6864.66870 |
| **std** | 8660.398374 | 129747.661567 | 9.217904 | 74896.515914 | 72711.059216 | 7.113061e+04 | 66199.329394 | 62784.900318 | 61850.740349 | 18007.76437 |
| **min** | 1.000000 | 10000.000000 | 21.000000 | -165580.000000 | -69777.000000 | -1.572640e+05 | -170000.000000 | -81334.000000 | -339603.000000 | 1.00000 |
| **25%** | 7500.750000 | 50000.000000 | 28.000000 | 6059.750000 | 6239.000000 | 6.570500e+03 | 6569.000000 | 5962.500000 | 5418.500000 | 1610.00000 |
| **50%** | 15000.500000 | 140000.000000 | 34.000000 | 26732.500000 | 26848.000000 | 2.592550e+04 | 23437.000000 | 21319.000000 | 20818.500000 | 3000.00000 |
| **75%** | 22500.250000 | 240000.000000 | 41.000000 | 72184.250000 | 70286.500000 | 6.801950e+04 | 62630.000000 | 58516.250000 | 57462.500000 | 6005.00000 |
| **max** | 30000.000000 | 1000000.000000 | 79.000000 | 964511.000000 | 983931.000000 | 1.664089e+06 | 891586.000000 | 927171.000000 | 961664.000000 | 873552.00000 |

Using the describe() function to get the descriptive statistics summary of the chosen data set will allow us to understand the variables or features of the dataset in terms of mean, median (central tendency), dispersion, and quantiles. It works on numeric data, excluding NaN values.

**Getting all the information from the dataset using the info() function**

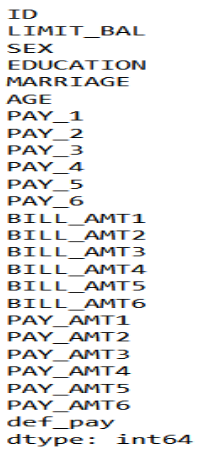


```
data_df.info()

<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
```

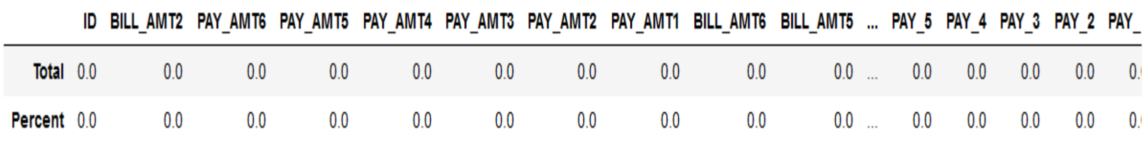
**Interpretation:** We can infer that our dataset has 25 columns and 3000 rows (as previously shown in the report). The information on the dataset reported below show that there are no missing features for any of the $30,000$ samples.

**Null or Missing values in dataset:**

**Interpretation:** This returns the number of null values in each column. As we can see, There is no missing data in the entire dataset. Also, we determined what percentage of the null values are present in each column. There are columns that have 0% null values.



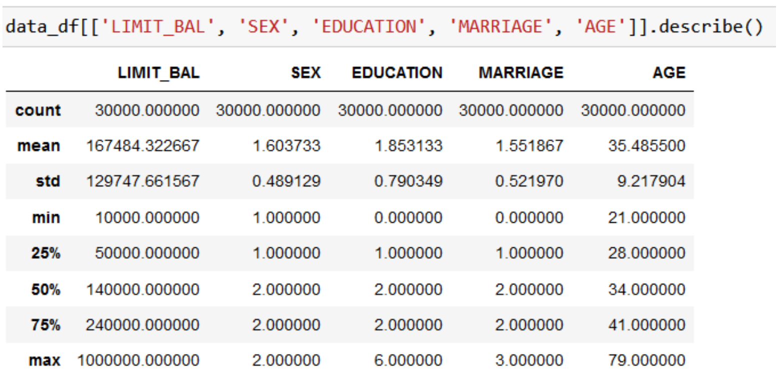
```
ID           0
LIMIT_BAL    0
SEX          0
EDUCATION    0
MARRIAGE     0
AGE          0
PAY_1        0
PAY_2        0
PAY_3        0
PAY_4        0
PAY_5        0
PAY_6        0
BILL_AMT1    0
BILL_AMT2    0
BILL_AMT3    0
BILL_AMT4    0
BILL_AMT5    0
BILL_AMT6    0
PAY_AMT1     0
PAY_AMT2     0
PAY_AMT3     0
PAY_AMT4     0
PAY_AMT5     0
PAY_AMT6     0
def_pay      0
dtype: int64
```



| | ID | BILL_AMT2 | PAY_AMT6 | PAY_AMT5 | PAY_AMT4 | PAY_AMT3 | PAY_AMT2 | PAY_AMT1 | BILL_AMT6 | BILL_AMT5 | ... | PAY_5 | PAY_4 | PAY_3 | PAY_2 | PAY_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Total** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0. |
| **Percent** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0. |

## Data Preprocessing

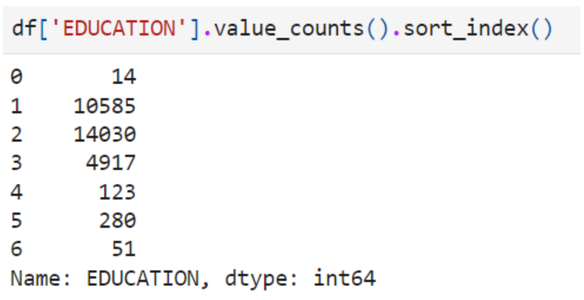
### Client Personal Information



```
data_df[['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE']].describe()
```

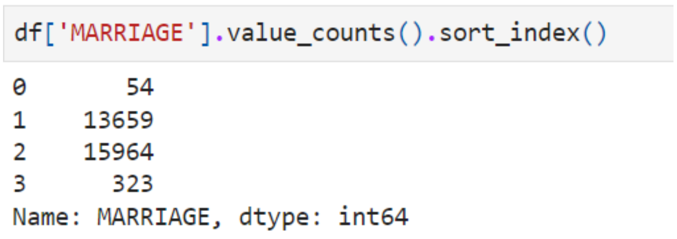
| | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE |
|---|---|---|---|---|---|
| **count** | 30000.000000 | 30000.000000 | 30000.000000 | 30000.000000 | 30000.000000 |
| **mean** | 167484.322667 | 1.603733 | 1.853133 | 1.551867 | 35.485500 |
| **std** | 129747.661567 | 0.489129 | 0.790349 | 0.521970 | 9.217904 |
| **min** | 10000.000000 | 1.000000 | 0.000000 | 0.000000 | 21.000000 |
| **25%** | 50000.000000 | 1.000000 | 1.000000 | 1.000000 | 28.000000 |
| **50%** | 140000.000000 | 2.000000 | 2.000000 | 2.000000 | 34.000000 |
| **75%** | 240000.000000 | 2.000000 | 2.000000 | 2.000000 | 41.000000 |
| **max** | 1000000.000000 | 2.000000 | 6.000000 | 3.000000 | 79.000000 |

We print the description of the client personal information features. The LIMIT_BAL, SEX, and AGE attributes seem to be consistent with the description provided earlier, while EDUCATION and MARRIAGE have some undocumented categories. EDUCATION ranges from 0 to 6, while MARRIAGE starts at category 0.



```
df['EDUCATION'].value_counts().sort_index()
```

```
0       14
1    10585
2    14030
3     4917
4      123
5      280
6       51
Name: EDUCATION, dtype: int64
```



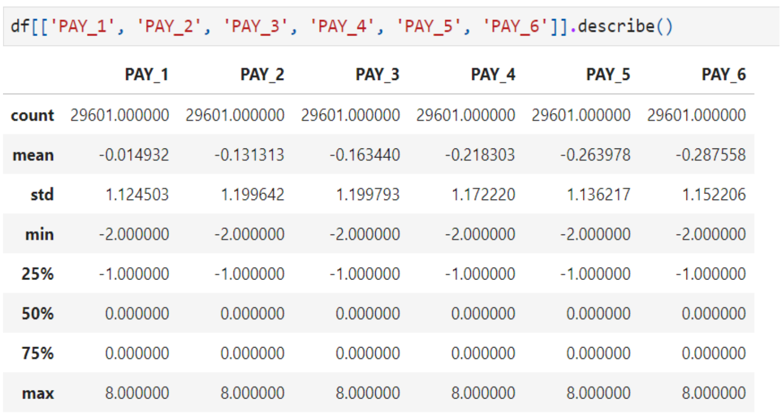
```
df['MARRIAGE'].value_counts().sort_index()
```

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

By changing the incorrect attribute or removing the rows involved in the issue, errors in the dataset can be fixed. We could be cautious and put the undocumented categories into another category, but since there aren't many anomalous entries (399, or 1.33% of total), we choose to remove the anomalies in the MARRIAGE column and combine the 0, 5, and 6 categories in EDUCATION into category 4, or "others" only.

### History of Past Payments

We print the description of the history of past payments features. We do not find any anomalies.



```
df[['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']].describe()
```

| | PAY_1 | PAY_2 | PAY_3 | PAY_4 | PAY_5 | PAY_6 |
|---|---|---|---|---|---|---|
| **count** | 29601.000000 | 29601.000000 | 29601.000000 | 29601.000000 | 29601.000000 | 29601.000000 |
| **mean** | -0.014932 | -0.131313 | -0.163440 | -0.218303 | -0.263978 | -0.287558 |
| **std** | 1.124503 | 1.199642 | 1.199793 | 1.172220 | 1.136217 | 1.152206 |
| **min** | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 | -2.000000 |
| **25%** | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 | -1.000000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **max** | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 |

**Amount of Bill Statement and Previous Payment**

We print the description of the features related to the amount of bill statement and the amount of previous payment. We do not find any anomalies.

```
data_df[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].describe()
```

| | BILL_AMT1 | BILL_AMT2 | BILL_AMT3 | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 |
|---|---|---|---|---|---|---|
| **count** | 29946.000000 | 29946.000000 | 2.994600e+04 | 29946.000000 | 29946.000000 | 29946.000000 |
| **mean** | 51278.911841 | 49224.542744 | 4.706321e+04 | 43306.688005 | 40352.140252 | 38911.533393 |
| **std** | 73682.871378 | 71219.298988 | 6.939321e+04 | 64374.889734 | 60836.076370 | 59592.166712 |
| **min** | -165580.000000 | -69777.000000 | -1.572640e+05 | -170000.000000 | -81334.000000 | -339603.000000 |
| **25%** | 3570.250000 | 2988.250000 | 2.684500e+03 | 2335.000000 | 1770.250000 | 1261.000000 |
| **50%** | 22400.000000 | 21221.000000 | 2.010800e+04 | 19066.000000 | 18121.000000 | 17098.500000 |
| **75%** | 67263.000000 | 64108.000000 | 6.024075e+04 | 54601.750000 | 50244.750000 | 49248.500000 |
| **max** | 964511.000000 | 983931.000000 | 1.664089e+06 | 891586.000000 | 927171.000000 | 961664.000000 |

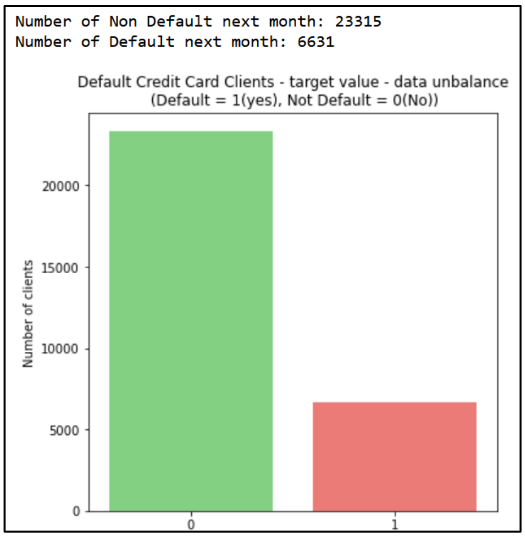
```
data_df[['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].describe()
```

| | PAY_AMT1 | PAY_AMT2 | PAY_AMT3 | PAY_AMT4 | PAY_AMT5 | PAY_AMT6 |
|---|---|---|---|---|---|---|
| **count** | 29946.000000 | 2.994600e+04 | 29946.000000 | 29946.000000 | 29946.000000 | 29946.000000 |
| **mean** | 5659.736826 | 5.926824e+03 | 5227.841314 | 4829.614573 | 4804.211080 | 5220.871035 |
| **std** | 16552.642231 | 2.306022e+04 | 17618.433069 | 15677.788192 | 15290.655486 | 17791.413753 |
| **min** | 0.000000 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 1000.000000 | 8.360000e+02 | 390.000000 | 298.000000 | 255.250000 | 122.000000 |
| **50%** | 2100.000000 | 2.010000e+03 | 1800.000000 | 1500.000000 | 1500.000000 | 1500.000000 |
| **75%** | 5007.000000 | 5.000000e+03 | 4511.500000 | 4015.000000 | 4040.750000 | 4000.000000 |
| **max** | 873552.000000 | 1.684259e+06 | 896040.000000 | 621000.000000 | 426529.000000 | 528666.000000 |

## Data Exploration

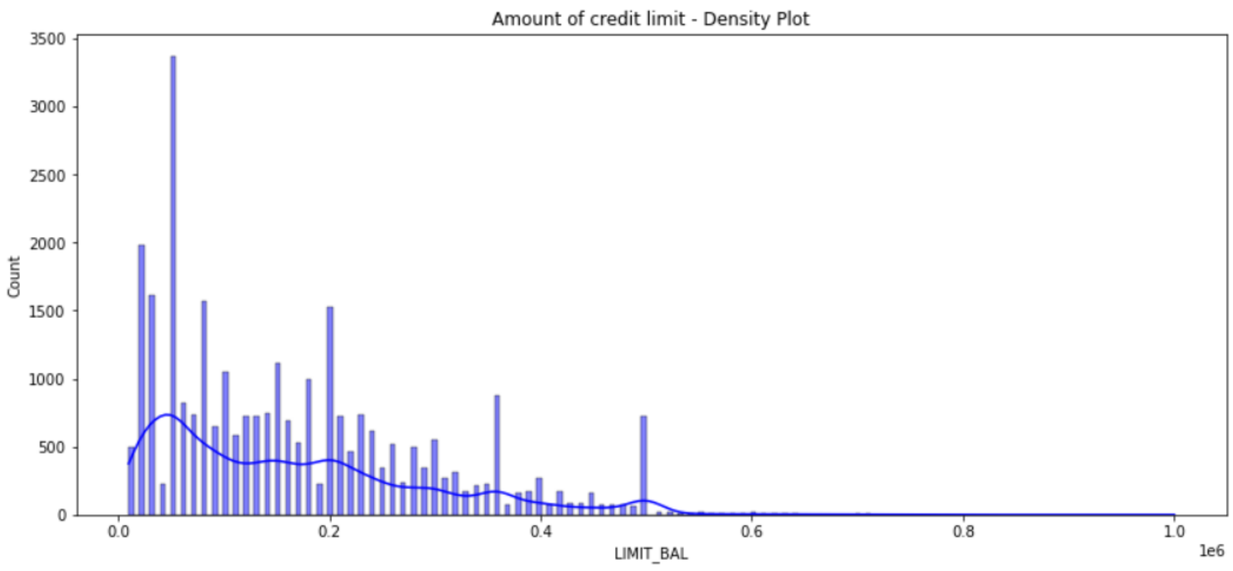
**Number of defaulters and non-defaulters in next month Interpretation:**

The above bar graph shows the number of clients who will default on the credit cards issued to them. In the chart, we found out that there are 23315 non-defaulters next month and 6631 defaulters next month, which is 3.5 times lower than the non-defaulters. So, the number of defaulters who are likely to default next month forms a very small group out of the total number of clients. Hence, we can say that there is no significant imbalance.

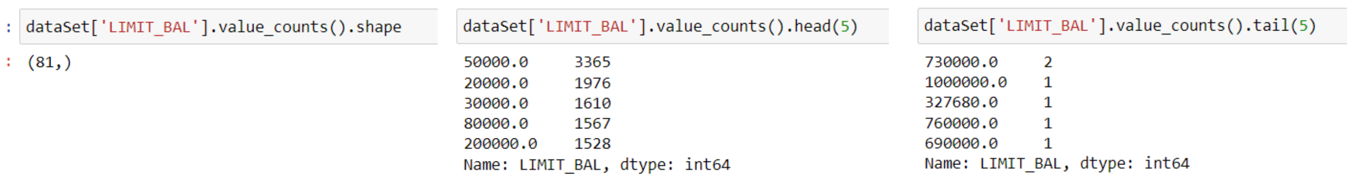


```
Number of Non Default next month: 23315
Number of Default next month: 6631
```

## Credit card limit distribution across data



**Interpretation:** Based on the maps depicting the relationship between the count of people using credit cards of a particular limit and the credit card spending limit balances for them, we can see that as the credit limit balance increases, there is a gradual decline in the number of people opting for those credit cards with respective limit balances. Plotting the continuous variables, we observe that dataset consists of skewed data of limiting balance. We have more number of clients having limiting balance between 0 to 200000 currency.



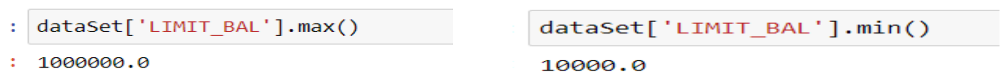
```
: dataSet['LIMIT_BAL'].value_counts().shape
: (81,)
```

```
dataSet['LIMIT_BAL'].value_counts().head(5)
50000.0     3365
20000.0     1976
30000.0     1610
80000.0     1567
200000.0    1528
Name: LIMIT_BAL, dtype: int64
```

```
dataSet['LIMIT_BAL'].value_counts().tail(5)
730000.0     2
1000000.0    1
327680.0     1
760000.0     1
690000.0     1
Name: LIMIT_BAL, dtype: int64
```

**Interpretation:** Credit limit has 81 different values, which means we can find out which category of people is using a higher limit and which is using a lower limit.

Credit cards with a limit of 50000, 20000, 30000, 80000, 200000 are used by the maximum number of people. There are 3365 people using credit card with a limit of 50000, 1976 people using credit card with limit of 20000 and so on. Credit cards with a limit of 690000, 760000, 327680, 1000000 are used by very few people. There is only one client or user using a credit card with those limits. This means that the company can either reduce the use of those credit cards or offer customers incentives to increase the usage of credit cards with these limits.

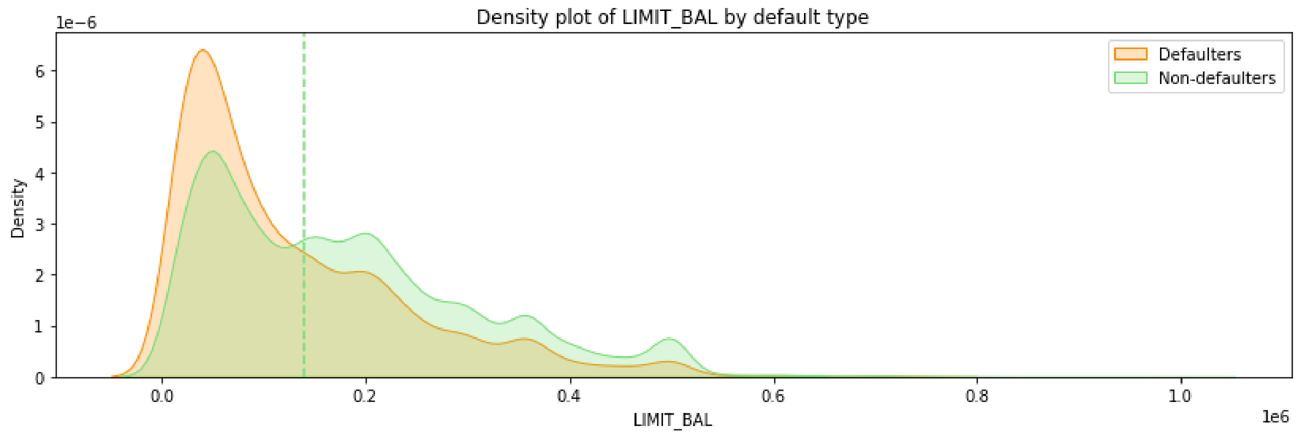


```
: dataSet['LIMIT_BAL'].max()
: 1000000.0
```

```
dataSet['LIMIT_BAL'].min()
10000.0
```

**Interpretation:** The maximum credit limit used by any client is of 1000000 and the minimum credit limit used is 10000.

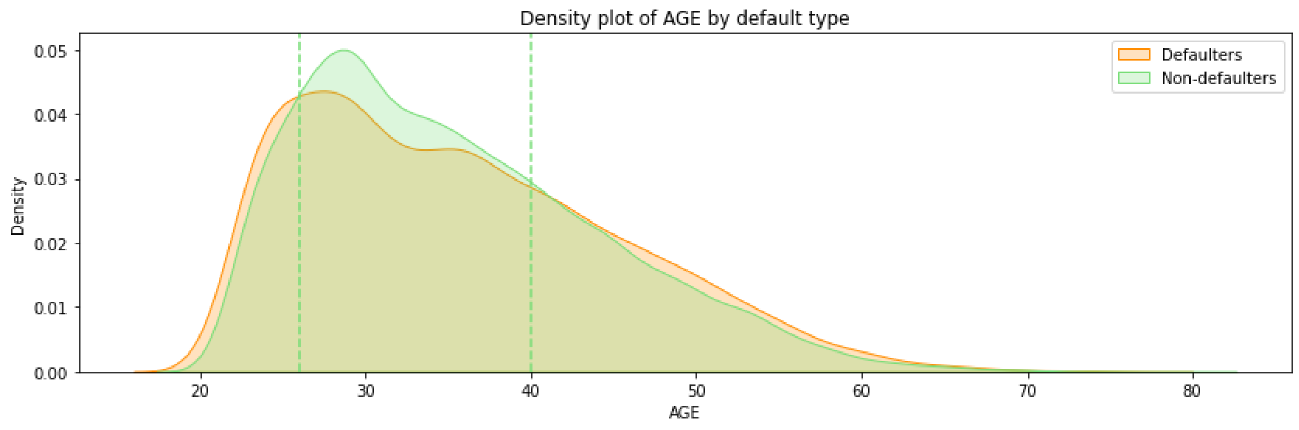
**Relation between Credit card limit and defaulters and non-defaulters of next month**



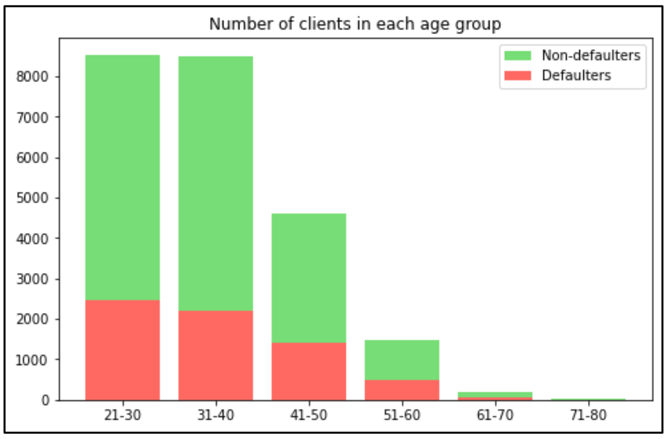
**Interpretation:** For the feature AGE, we perform a similar visual analysis, as shown above on the right. The probability of non-default of age between approximately $25$ and $40$ is higher, which indicates that consumers in this age group are more capable of repaying credit card loans. This may be because their work and family tend to be stable without too much pressure.

The features SEX, EDUCATION, and MARRIAGE are plotted according to the target variable using histograms, as shown below. Whether it is male or female, the proportion of defaulters is in line with the general situation, and we can say the same for the categories of the other two features.

**AGE Distribution by default type**

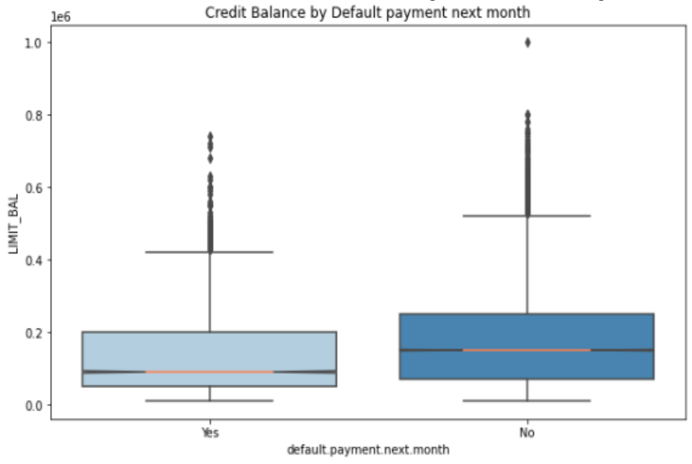


**Interpretation:** This graph helps us to find the relationship between the age and the clients who will default or not default. We can see that the defaulter count for the age in the range of 20-40 years is higher than the default counts for the higher age balances.



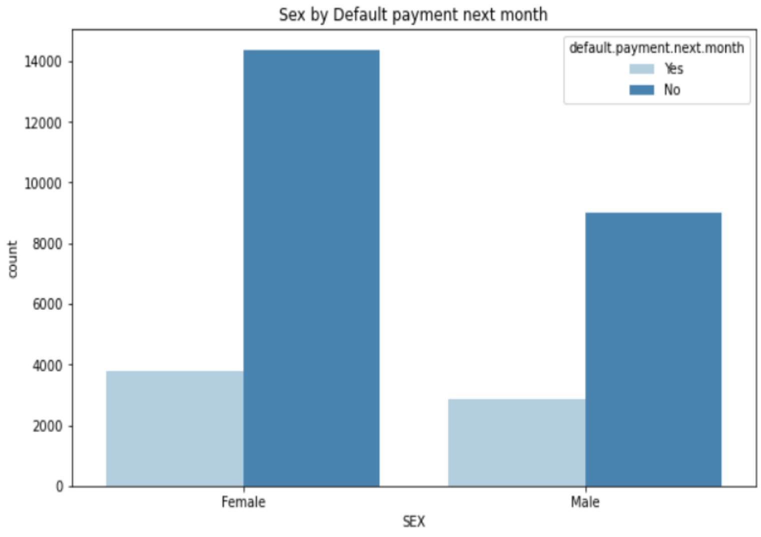
We have more number of clients from age bracket of 20 to 40, i.e., clients from mostly young to mid aged groups. We have maximum clients from 21-30 age group followed by 31-40. Hence with increasing age group the number of clients that will default the payment next month is decreasing. Hence we can see that Age is important feature to predict the default payment for next month.

## Relation: Limit or Credit Bal by Default Payment next month



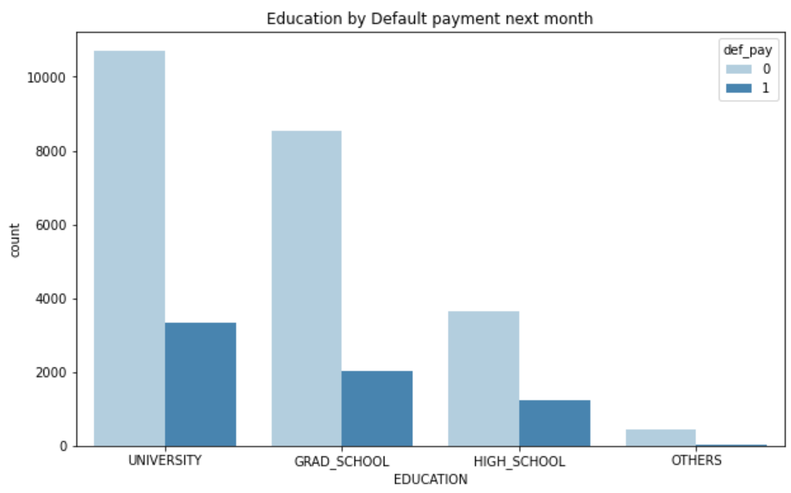
**Interpretation:** From the graph which shows the relation between credit card limit balance and default payment next month, it can be interpreted that the median credit card limit balance of the customers who will default next month is less than the median credit card limit balance of the customers who will not default. Also, the credit card with the maximum limit is used by the customer who will not default. And Q4(Max) value of non-defaulters is higher as compared to defaulters.

## Relation: Sex by Default payment next month



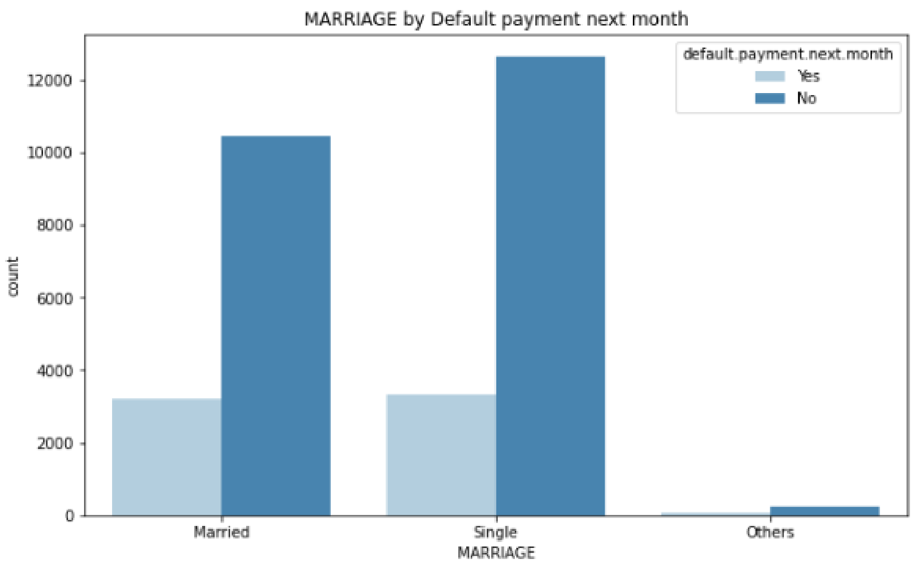
**Interpretation:** This graph maps the relation between the gender and the count of the defaulter payments and the non-defaulter payments for the next month. We can see that the number of defaults for the next month is almost the same for men and women, however non-defaulter payments for the next month are highest for the women compared to men by 7000. Hence, we can say that women default less than men. Also, among the females, the count of non-defaulter females is very high as compared to the defaulter females. So, a smaller number of females default compared to total females.

**Relation: Education by Default payment next month**



**Interpretation:** This graph maps the relationship between education of the people and them being categorized based on defaulters and non-defaulters for next month. We can see that as the qualification increases the non-defaulter list is increasing as well as the defaulter list, although defaulters are 1/3rd of the count compared to the non-defaulters. Also, it can be interpreted that university students default the highest during credit card payments and approximately 3000 university students will default next month.

**Relation: Marriage by Default payment next month**

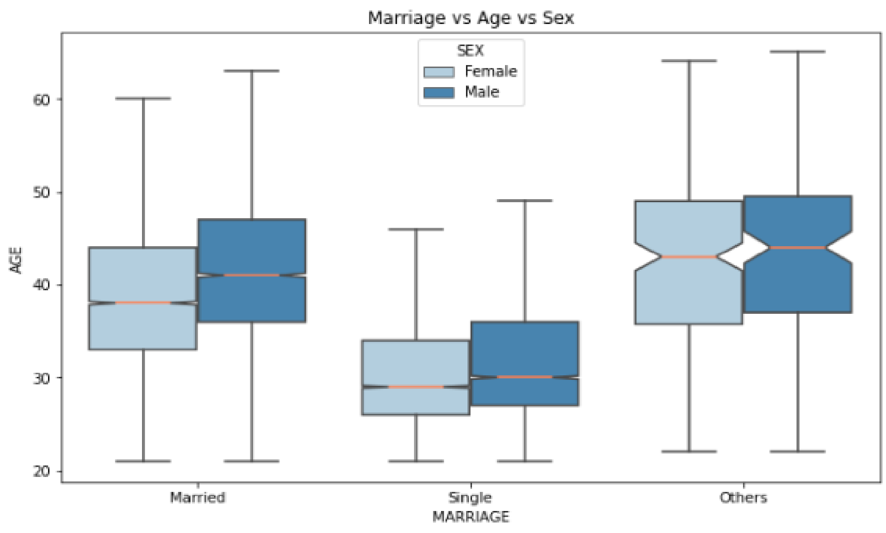


**Interpretation:** From the graph, we can interpret that the number of defaulters in credit card payments is nearly the same for married credit card users as compared to the credit card users who are single. Number of non-defaulters is highest for the credit card users who are single. And in every category, the number of non-defaulters is very high compared to the defaulters.

**Understanding relation between Sex, Education, Age, Marriage and Credit card amount limit**

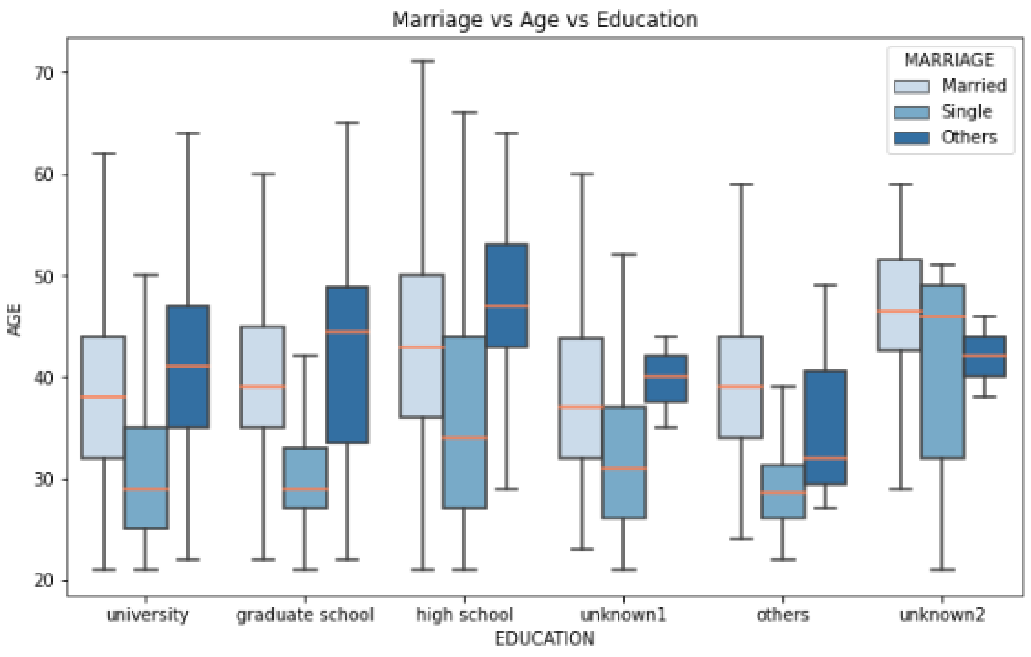
In this we are analysis the relation between parameters like Sex, Education, Age, Marriage and Credit card limit, and how are they related with each other and whether they are affecting our data for further analysis and prediction.

**Relation: Marriage vs Sex vs Age**



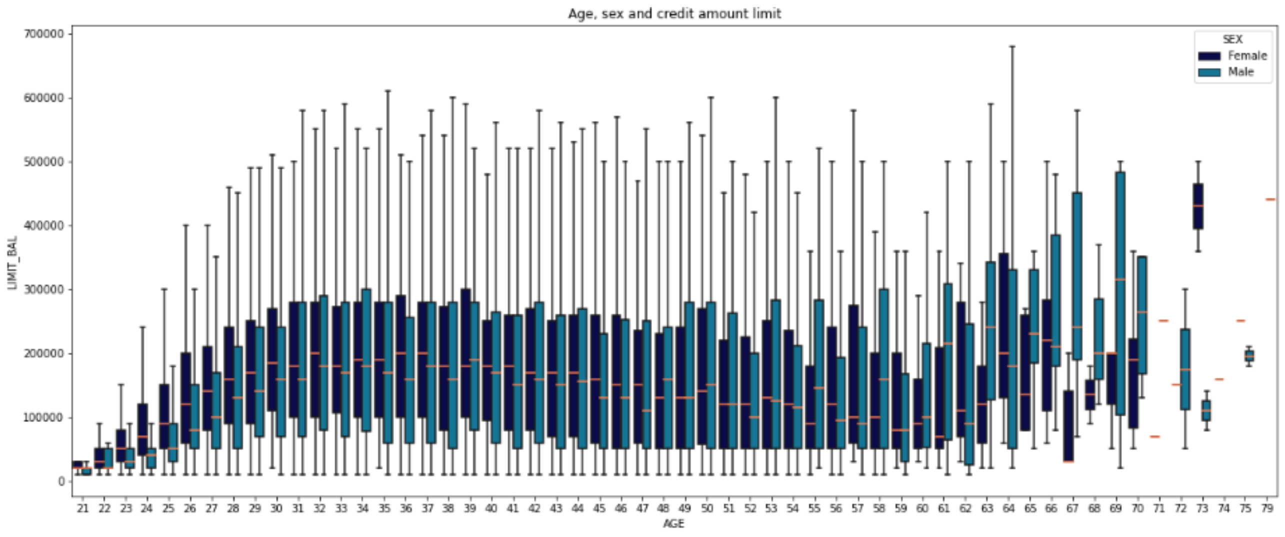
**Interpretation:** We can see that the mean age for both men and women is nearly the same in the case of singles and others but in the case of the Married category there is a notable difference in the mean with mean age of men being more than that of women. The Q1 for the respective Marriage, single and others are higher for men than the women in those categories respectively. Married men's Q3 and Q4 (maximum) values are higher than the corresponding values for married women. Q0 (minimum) age of men and women is same for married and singles. People in others category could be divorced or widowed, as mean is above 40.

**Relation: Marriage vs Age vs Education**



**Interpretation:** Here we can see the distribution of the people based on Qualification and Marriage, here we can say that 40 to 50 years age are in the major mean areas, the highest distribution of age was for high school followed by university and graduate school.

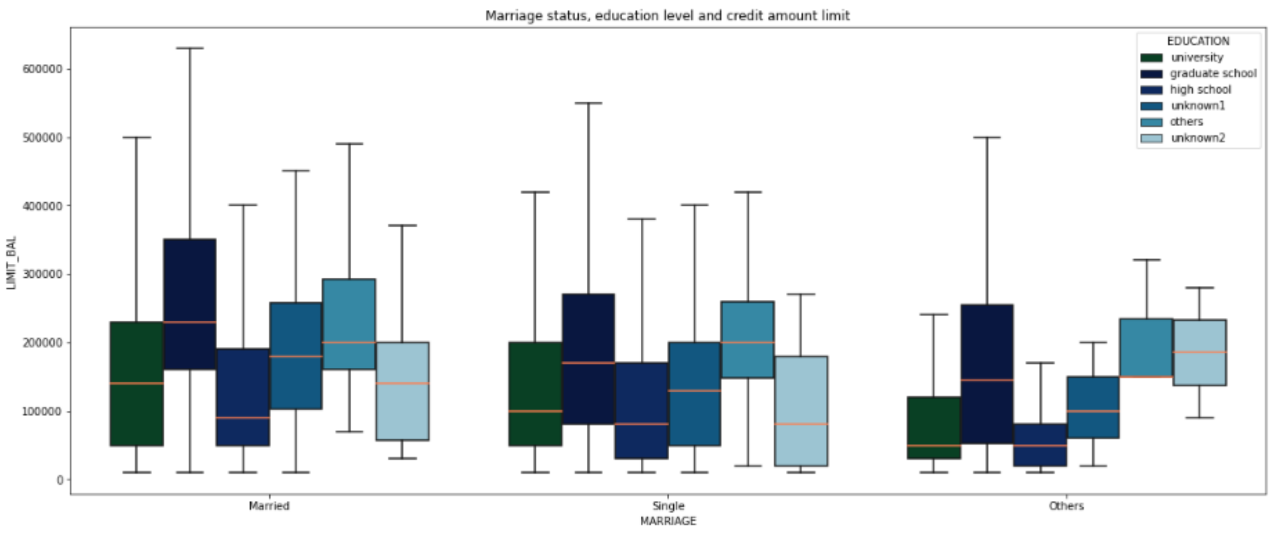
**Relation: Age vs Sex vs Credit amount limit**



**Interpretation:** The graph maps the relationship between credit card limit balances with the respective individual ages shown by male and female, and the distribution across the credit card limit balances spread over the respective range for those ages. we can see that for 69 years ages the spread for the limit balances is high ranging from 1 lakh to 5 lakhs, the maximum being for 64 years where the balance amount is reaching a high of 7 lakhs. Mean, Q3 and Q4 values increase with age for both males and females up to around 35 years of age, after which they oscillate and reach a peak for Q4 for males at 64 years of age.

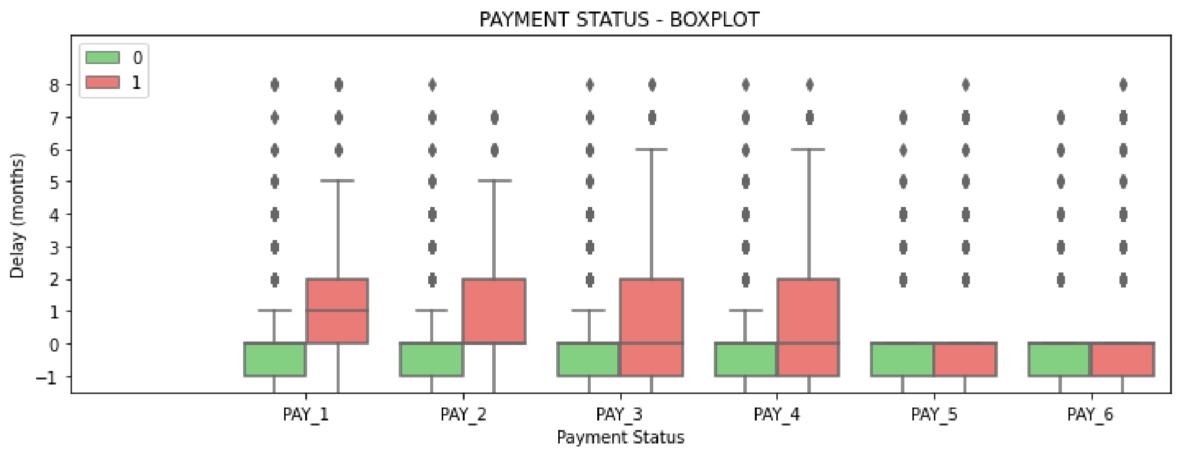
Males normally have smaller mean values than females, on average, with a few exceptions, such as around ages 39, 48, and up until roughly 60, where male mean values are typically larger than female mean values.

**Relation: Marriage status vs Education vs Credit limit**



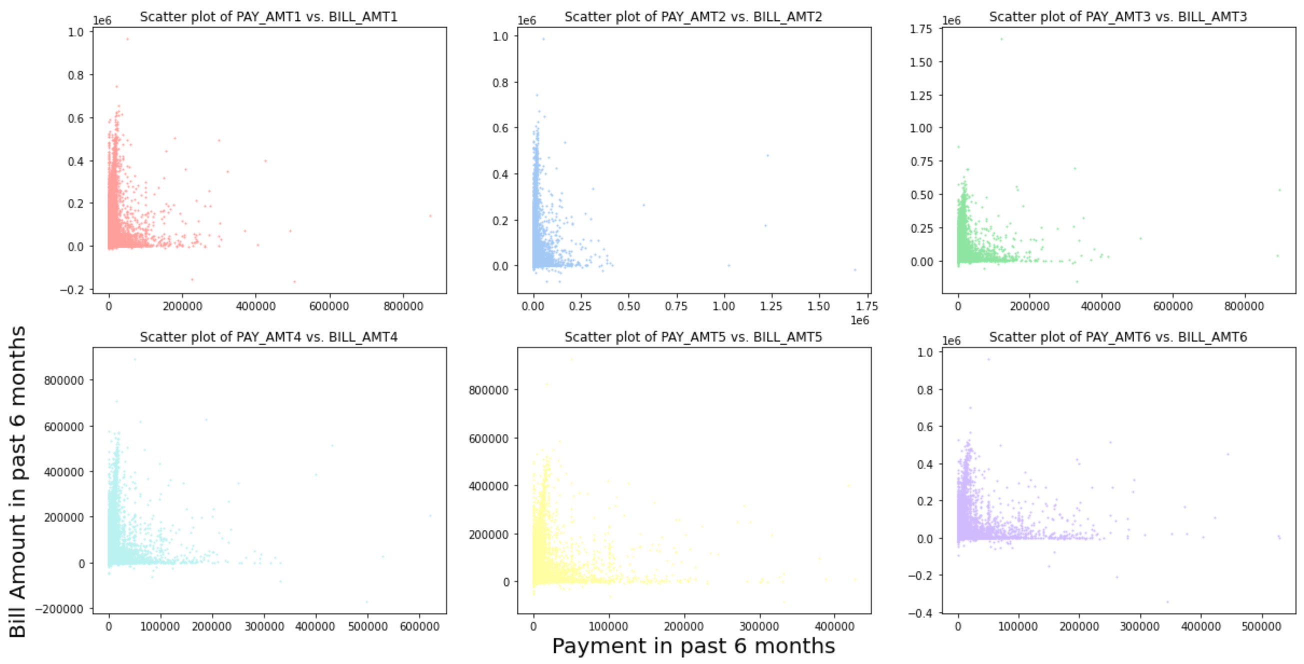
**Interpretation:** The graph maps relationship between limit balance ranges with respect to the marriage categories that has been subdivided on the basis of education qualification. We can see that the averages for the different age groups for the limit balances do not coincide for the respective marital statuses. Also, the credit limit is highest for the married. And graduate school people are having the highest credit limits in all categories.

## PAYMENT STATUS FEATURES (BY TARGET)



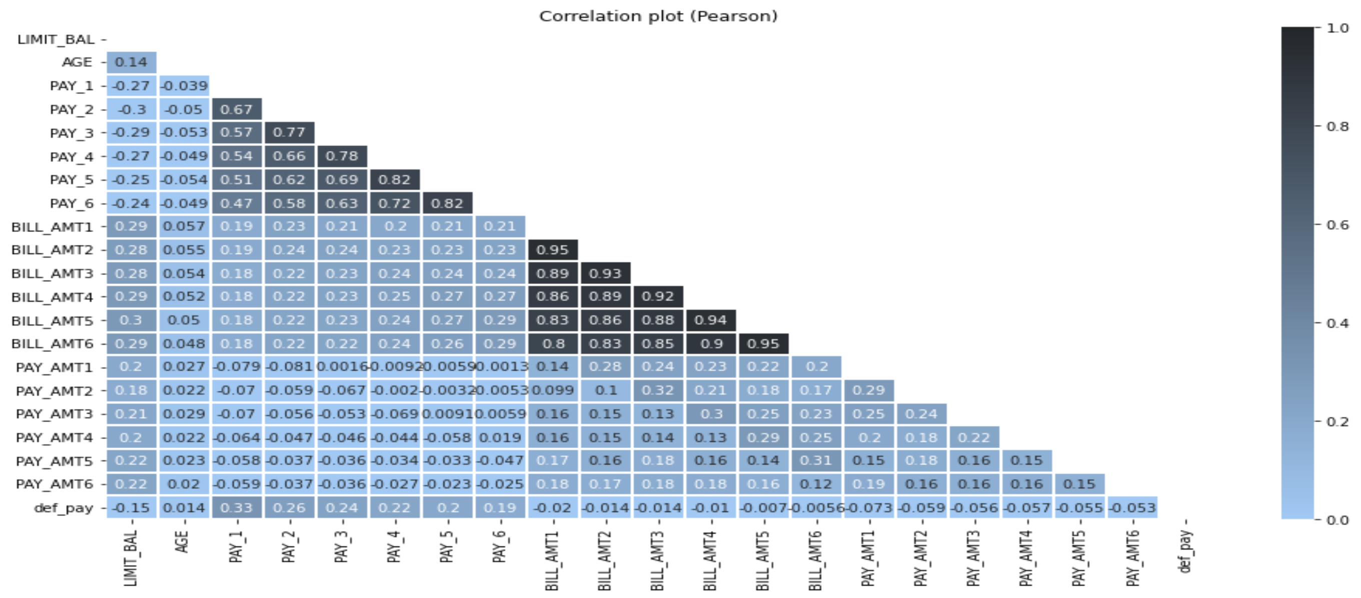
The payment status feature set is shown below via boxplots. It can be seen that clients who delay payment by one month or less have fewer credit card defaults. In particular, the repayment status in September, i.e., PAY_1, holds a greater discriminatory power than the repayment status in the other months.

## Past six months bill amount effect on the payment default next month or not:



Above plot indicates that there is higher proportion of clients for whom the bill amount is high but payment done against the same is very low. This we can infer since maximum number of datapoints are closely packed along the Y-axis near to 0 on X-axis.

**Correlation between variables**



**Interpretation:** We note that there is a strong positive correlation between the BILL_AMTn features, which may indicate a redundancy of information. We see that despite BILL_AMTX variables having high correlation, it would be a mistake to just drop all of them because they are important for the explanation of dependent variables.

## Data Preparation and Transformation

The data preparation and transformation stage will consist of the following steps

1. **Missing Data Analysis & Imputation:** As there is no missing values, missing value analysis and imputation is not needed.
2. **Outlier Removal:** Though there are outliers in some of the variables, we tend to keep those outliers as a company has customers with diverse ranges. Hence, some customer has higher credit limit and some have lower credit limits.
3. **Feature Encoding:** This step will include encoding categorical features like 'EDUCATION', 'MARRIAGE', 'SEX' using one-hot encoding. Also, non-categorical features have been standardised in this step since non-categorical variables may have different units so different scales and magnitude will be present.

4. **Train Validation Split:** For Data Modelling purpose, we took 70% of the data for training and the rest 30% for testing.

## Data Modelling

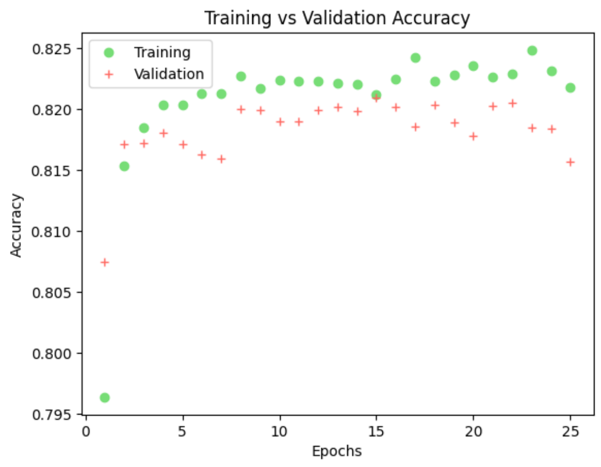
Since the output variable is categorical, Classification techniques can be used for prediction.
For Data Modelling purpose, we took 70% of the data for training and the rest 30% for testing.

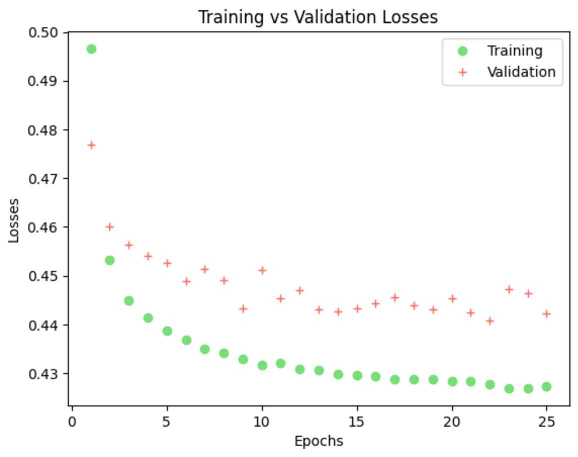
### Deep Learning Models

A standard Dense Neural Network framework is used using the Keras library for a binary classification task initially after which hyperparameter tuning.

The dataset is split into training and testing sets of 70% and 30% data respectively, and both feature and target variables are pre-processed by standardizing them using mean and standard deviation.

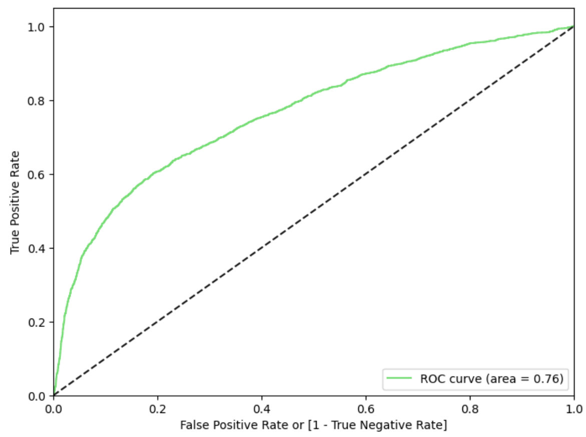
The neural network architecture consists of an input layer with 16 neurons, a hidden layer with 8 neurons, and an output layer with a single neuron using a sigmoid activation function for binary classification. The model is compiled with binary cross-entropy loss and the RMSprop optimizer. Training is performed for 25 epochs with a batch size of 32, and the training/validation accuracy and loss metrics were plotted. The plots are shown below.





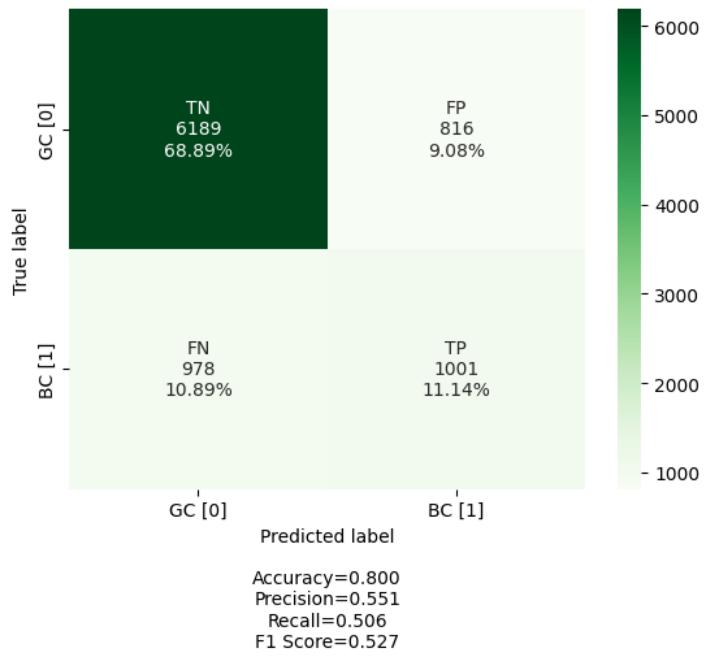
It shows that the training and validation accuracy increased with increase in epochs and the loss values showed a decline.

**Plotting the ROC to assess the model performance:**



The ROC score for model performance of a standard DNN framework is 76%.

**Confusion Matrix:**



**Interpretation**

From all 7005 NO's, 6189 NO's are correctly predicted, and 816 NO's are predicted as YES.

From all 1979 YES, 1001 YES are correctly predicted, and 978 YES are predicted as NO.

The confusion matrix shows a solid performance for model accuracy (0.8) but moderate scores for precision (0.55) and recall (0.5). This shows that while the model in general can identify labels well, the rate of false positives over totality of predicted positives is quite high. That can be a significant model shortcoming in the case of identifying defaulting customers.

The standard model with default set of parameters may not be the ideal model architecture and a different set of hyperparameters may yield a more powerful and robust model. There are different approaches by which we can select a set of hyperparameters that shows best performance on a smaller data and use that architecture to train our model. The following section takes an in-depth understanding of how the different parameter optimizing techniques influence the model performance.

## Hyperparameter Tuning Methods

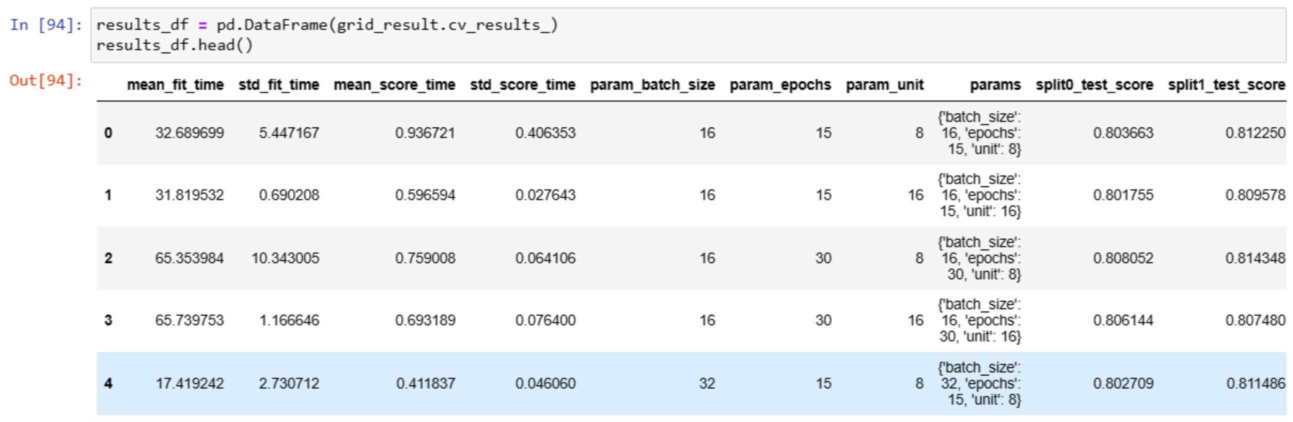
Data Preprocessing Steps Explained:

1. Standardization:
We used MinMaxScaler to transform the data into a range between 0 and 1. This ensures features with larger scales don't dominate the model training.
The code shows the transformed data X where each value now falls between 0 and 1.

2. Dividing Data:
We used train_test_split to split the data into training and testing sets (70% and 30%, respectively). This helps evaluate the model's performance on unseen data.
The output shows the shapes of each set:
X_train: (20962 rows, 26 columns) for training features
X_test: (8984 rows, 26 columns) for testing features
y_train: (20962 rows) for training labels
y_test: (8984 rows) for testing labels

## Hyperparameter Tuning - Grid Search

Grid Search uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters. This makes the processing time-consuming and expensive based on the number of hyperparameters involved.


```
In [94]: results_df = pd.DataFrame(grid_result.cv_results_)
         results_df.head()
Out[94]:
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_batch_size | param_epochs | param_unit | params | split0_test_score | split1_test_score |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 32.689699 | 5.447167 | 0.936721 | 0.406353 | 16 | 15 | 8 | {'batch_size': 16, 'epochs': 15, 'unit': 8} | 0.803663 | 0.812250 |
| **1** | 31.819532 | 0.690208 | 0.596594 | 0.027643 | 16 | 15 | 16 | {'batch_size': 16, 'epochs': 15, 'unit': 16} | 0.801755 | 0.809578 |
| **2** | 65.353984 | 10.343005 | 0.759008 | 0.064106 | 16 | 30 | 8 | {'batch_size': 16, 'epochs': 30, 'unit': 8} | 0.808052 | 0.814348 |
| **3** | 65.739753 | 1.166646 | 0.693189 | 0.076400 | 16 | 30 | 16 | {'batch_size': 16, 'epochs': 30, 'unit': 16} | 0.806144 | 0.807480 |
| **4** | 17.419242 | 2.730712 | 0.411837 | 0.046060 | 32 | 15 | 8 | {'batch_size': 32, 'epochs': 15, 'unit': 8} | 0.802709 | 0.811486 |

results_df summarizes the results as a DataFrame. Each row represents a different parameter combination and shows:
1. Training and scoring times.
2. Hyperparameters used.
3. Test scores for each cross-validation fold and the average (mean_test_score).
4. Rank according to mean_test_score (lower is better).

1. Obtaining Best Parameters:

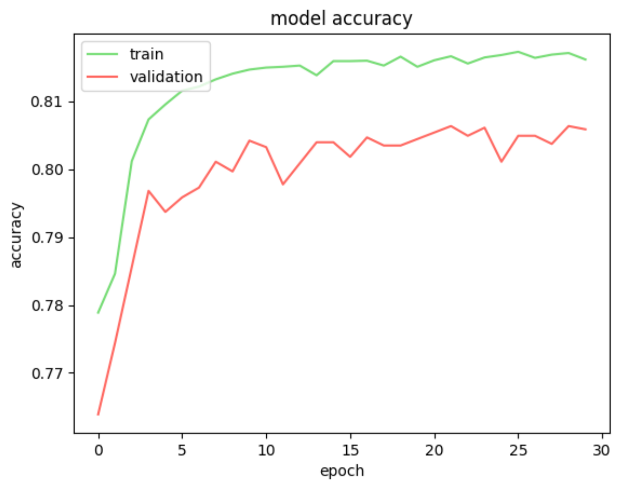
- grid_result.best_score_ returns the highest average test score across all parameter combinations evaluated by the Grid Search. In this case, it's 0.816764.
- grid_result.best_params_ returns a dictionary containing the hyperparameters corresponding to the best score. Here, it's **{'batch_size': 16, 'epochs': 30, 'unit': 8}**

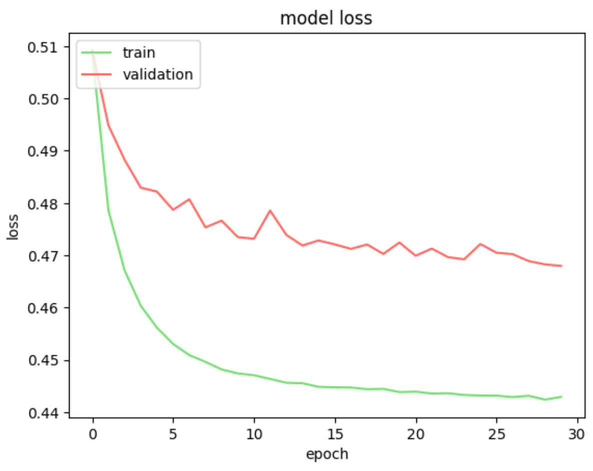
The loop iterates through all combinations of test scores (means) and hyperparameters (params) from the Grid Search results. For each combination, it prints the score and the corresponding hyperparameters.

**Interpretation:**
The Grid Search identified a combination of batch_size=16, epochs=30, and unit=8 as the one offering the highest average test score of 0.816764.

**Model Building with Identified Hyperparameters**

- Learning Over Time: Both training and validation accuracy steadily increased over the first 10-15 epochs, indicating that the model was learning effectively. After this point, the improvements plateaued, suggesting that the model had reached its peak performance within this configuration.
- Accuracy: The final training accuracy reached 81.61%, while the validation accuracy peaked at 80.63%. This suggests that the model is able to generalize reasonably well to unseen data, with a slight overfitting tendency (as the training accuracy is slightly higher than validation accuracy).
- Convergence: Both training and validation loss steadily decreased over the epochs, converging towards a minimum value. This further supports that the model was learning and improving its ability to minimize errors.
- Early Stopping: The validation accuracy exhibited minor fluctuations after around 20 epochs, suggesting that training could have potentially been stopped earlier to prevent potential overfitting.



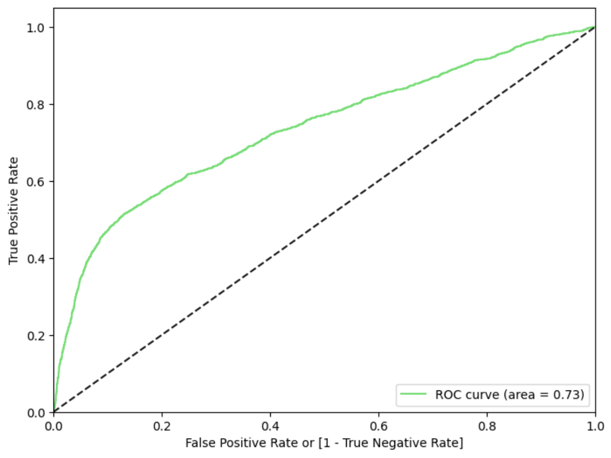


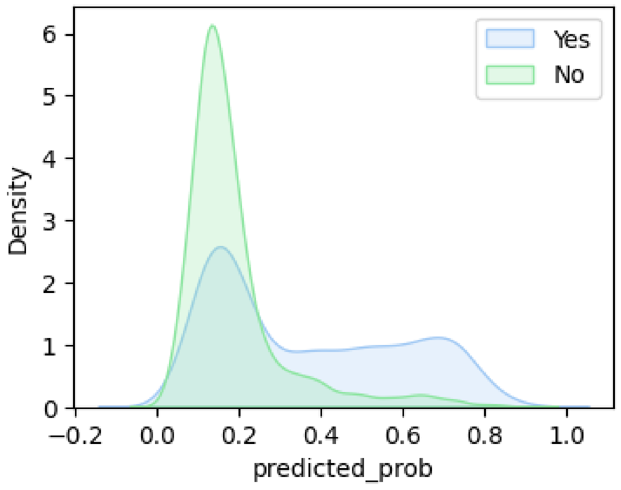
Overall Assessment:

The model has achieved a reasonable level of accuracy, demonstrating its ability to learn from the data and make predictions. There's a slight indication of overfitting, which could be addressed through techniques like regularization or early stopping. It's worth exploring model architecture adjustments (e.g., number of layers, units per layer) and hyperparameter tuning to enhance performance potentially further.

**Model Validation**

The AUC score of 0.7333 indicates moderate performance.

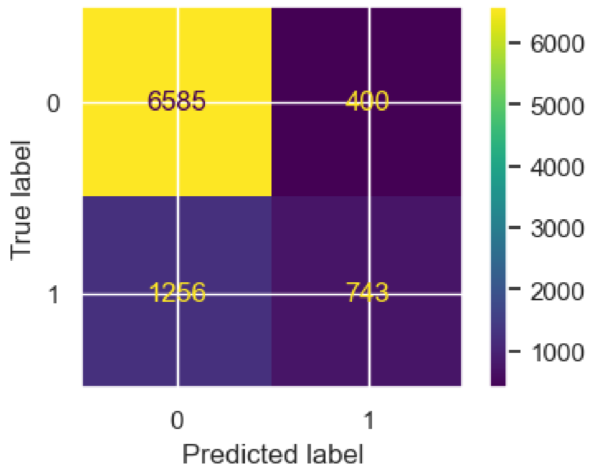




Confusion Matrix:

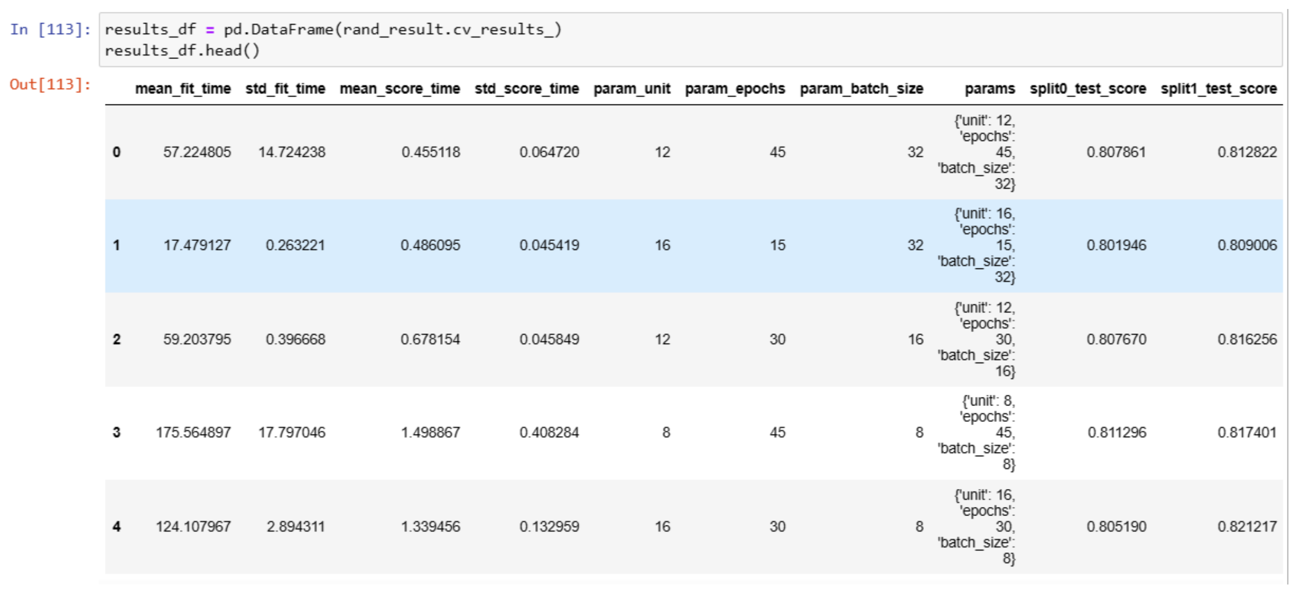
Two kernel density plots show the distribution of predicted probabilities for both classes (Yes and No).
The confusion matrix and associated display show the actual vs. predicted counts, revealing:

- High accuracy for class 0 (84% recall and precision).
- Lower accuracy for class 1 (37% recall and 65% precision).
- Overall accuracy is 82% but weighted towards the larger class (0).



## Hyperparameter Tuning - Random Search

RandomizedSearchCV is used with an early stopping callback to prevent overfitting. It trains the model with different hyperparameter combinations from the grid using k-fold cross-validation (4 folds in this case)



```
In [113]: results_df = pd.DataFrame(rand_result.cv_results_)
          results_df.head()
```

Out[113]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_unit | param_epochs | param_batch_size | params | split0_test_score | split1_test_score |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 57.224805 | 14.724238 | 0.455118 | 0.064720 | 12 | 45 | 32 | {'unit': 12, 'epochs': 45, 'batch_size': 32} | 0.807861 | 0.812822 |
| 1 | 17.479127 | 0.263221 | 0.486095 | 0.045419 | 16 | 15 | 32 | {'unit': 16, 'epochs': 15, 'batch_size': 32} | 0.801946 | 0.809006 |
| 2 | 59.203795 | 0.396668 | 0.678154 | 0.045849 | 12 | 30 | 16 | {'unit': 12, 'epochs': 30, 'batch_size': 16} | 0.807670 | 0.816256 |
| 3 | 175.564897 | 17.797046 | 1.498867 | 0.408284 | 8 | 45 | 8 | {'unit': 8, 'epochs': 45, 'batch_size': 8} | 0.811296 | 0.817401 |
| 4 | 124.107967 | 2.894311 | 1.339456 | 0.132959 | 16 | 30 | 8 | {'unit': 16, 'epochs': 30, 'batch_size': 8} | 0.805190 | 0.821217 |

The first few rows, but presumably, RandomSearchCV explored various combinations and identified the one with the highest average test score (rank 1 in the shown rows).

- In this case, unit=12, epochs=45, batch_size=32 achieved the best average score of 0.8158 with some variation across folds.
- Other combinations also performed well, indicating that the model might be sensitive to various parameter choices.

**Obtaining best Hyperparameters**

Best Hyperparameters:

- rand_result.best_score_ shows the highest average test score achieved during the Random Search, which is 0.8190538436174393.
- rand_result.best_params_ indicates the specific hyperparameter combination that led to this best score: {'unit': 8, 'epochs': 30, 'batch_size': 8}. This means the model performed best with 8 neurons in the first layer, trained for 30 epochs, and using a batch size of 8 samples.

Individual Score and Parameter Combinations:

- The for loop iterates through all the combinations of mean test scores and corresponding hyperparameters explored by Random SearchCV.
- Each printout shows a "Score: " followed by the average test score for a specific parameter combination enclosed in "Hyperparameters: "

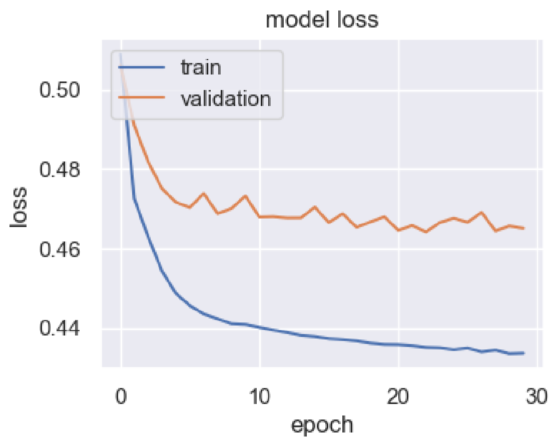
Observations:

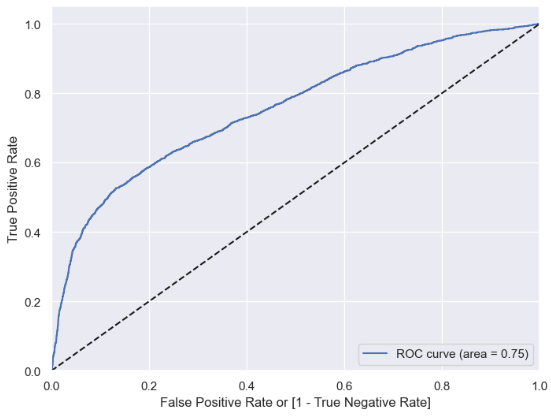
- While the best overall score was achieved with unit=8, epochs=30, batch_size=8, several other combinations come close in terms of performance. This suggests that the model might be less sensitive to specific parameter choices within a certain range.
- There's some variation in scores across different folds for each parameter combination. This is common in cross-validation and highlights the importance of using an average score as a more robust measure of performance.

- Overall, the results indicate that the model can achieve good accuracy with various hyperparameter settings. Further analysis might be needed to determine the most suitable combination based on specific needs and priorities (e.g., favoring higher accuracy even if it requires more training time or computational resources).

**Model Building with Identified Hyperparameters**

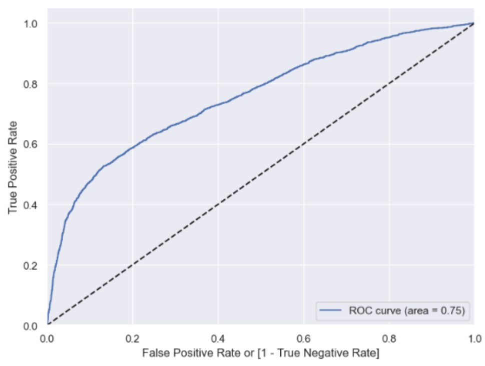
- Learning Over Time: Both training and validation accuracy showed a clear upward trend in the first 15-20 epochs, indicating effective learning. After this point, they plateaued, suggesting the model had reached its peak performance within this configuration.
- Final Accuracies: The model achieved a training accuracy of 82.23% and a validation accuracy of 80.80%, demonstrating its ability to learn and generalize to unseen data reasonably well.
- Slight Overfitting: The training accuracy consistently being slightly higher than validation accuracy hints at a minor overfitting tendency.
- Convergence: Both training and validation loss steadily decreased over the epochs, converging towards a minimum value, further confirming the model's learning process.
- Potential for Early Stopping: The validation accuracy exhibited some fluctuations after around 20 epochs, suggesting that training could have potentially been stopped earlier to prevent overfitting and potentially save training time.

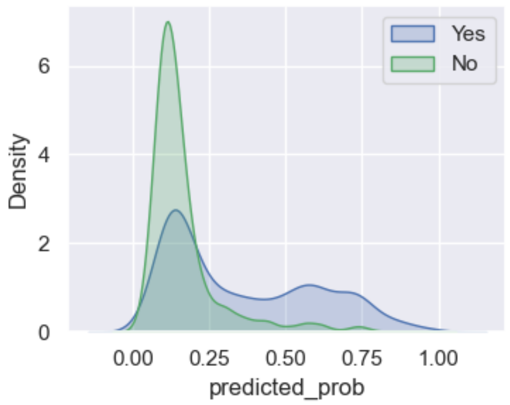


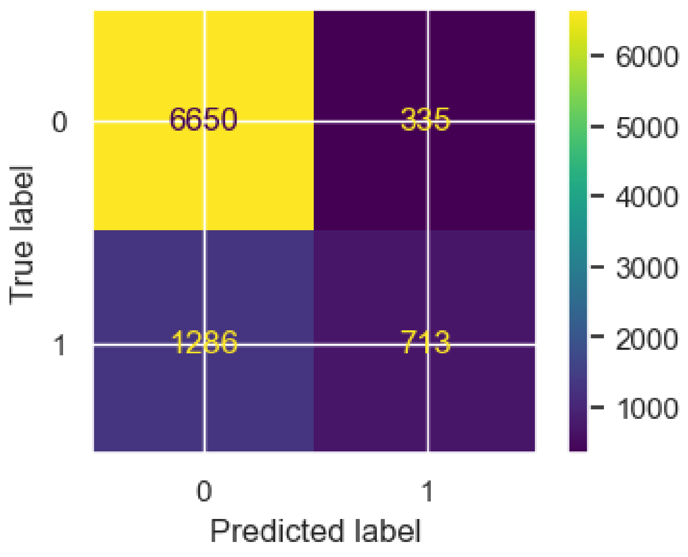


**Model Validation**

The AUC (Area Under the Curve) summarizes the overall performance, with higher values indicating better discrimination between classes. Here, the AUC score of 0.7547 suggests moderate performance, meaning the model can reasonably distinguish between positive and negative cases.







The confusion matrix provides a clearer breakdown of the classification results:

True Positives (TP): 6985 (correctly predicted positive cases)
True Negatives (TN): Not shown, but can be derived from the table (correctly predicted negative cases)
False Positives (FP): Not shown, but can be derived from the table (incorrectly predicted positive cases)
False Negatives (FN): 1999 (incorrectly predicted negative cases)

**Performance Measures:**

- Precision, Recall, F1-score, and Support are shown for both classes (0 and 1):
- Class 0: High accuracy (0.84 precision, 0.95 recall) due to the larger number of samples in this class.
- Class 1: Lower accuracy (0.68 precision, 0.36 recall) due to a higher rate of false negatives.
- Overall Accuracy: 0.82, meaning the model correctly classifies 82% of the samples.
- Macro and Weighted Average: Summarize overall performance across both classes, also suggesting moderate accuracy.

**Interpretation:**

- The model achieves good performance for the majority class (0) but struggles with class 1, evidenced by lower recall and precision.
- The AUC score also supports this observation, indicating moderate class discrimination ability.
- While the overall accuracy might seem acceptable, depending on the specific context, the performance difference between classes might be a crucial consideration

## Hyperparameter Tuning - Bayesian Optimization

The model architecture with several hyperparameters chosen by Bayesian optimization:
units: The number of neurons in the first Dense layer, ranging from 8 to 64 with an increment of 8.
activation: The activation function for the first Dense layer, chosen between **sigmoid and relu**.

Bayesian Tuner and Objective:

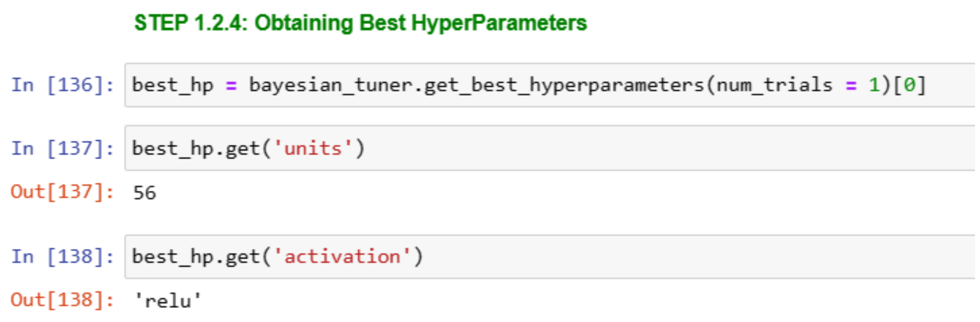
bayesian_tuner defines the optimization parameters:
objective: Optimize for the highest accuracy on the validation set.
max_trials: Perform a maximum of 5 hyperparameter configurations.
directory: Save results in the specified directory.
project_name: Identify the project for organizing results.
overwrite: Allow re-running optimization over existing results.

Execution and Results:

- The code runs the search using bayesian_tuner.search, passing the training and validation data, training parameters, and early stopping callback.
- The output shows the completion of the 5th trial and its accuracy score (0.8163).
- It also reports the current best accuracy achieved throughout the search (0.8247) and the total elapsed time (6 minutes 58 seconds).

**Interpretation:**

Bayesian optimization explored 5 different hyperparameter configurations, with the best one achieving an accuracy of 0.8247 on the validation set. This suggests the optimization process successfully identified a promising combination for this model.

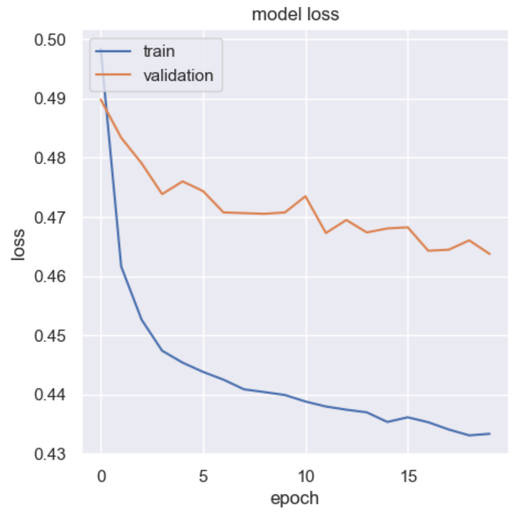

STEP 1.2.4: Obtaining Best HyperParameters

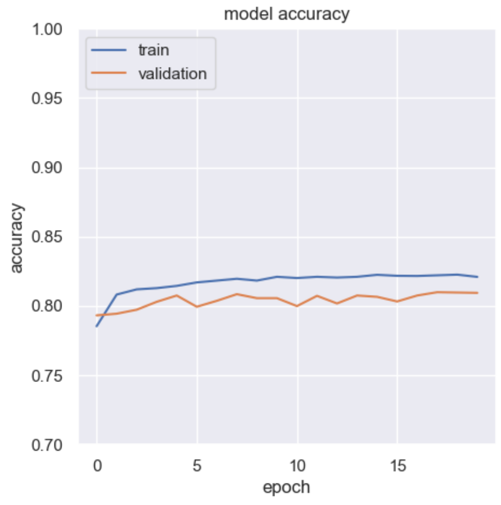
```
In [136]: best_hp = bayesian_tuner.get_best_hyperparameters(num_trials = 1)[0]

In [137]: best_hp.get('units')
Out[137]: 56

In [138]: best_hp.get('activation')
Out[138]: 'relu'
```

**Model Building with Identified Hyperparameters**

- Learning and Convergence: Both training and validation accuracy and loss demonstrate clear learning patterns, converging towards stable values over the 20 epochs. This indicates the model's ability to learn and improve its predictions.
- Final Accuracies: The model achieved a training accuracy of 82.10% and a validation accuracy of 80.94%, suggesting a reasonable capability to generalize to unseen data.
- Minimal Overfitting: The slight gap between training and validation accuracy is relatively small, hinting at minimal overfitting tendencies.

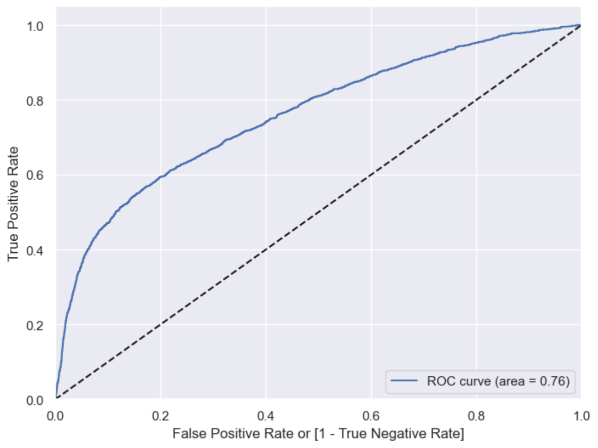
- Dropout Effectiveness: The inclusion of a dropout layer with a 5% rate appears to have contributed to mitigating overfitting, as evidenced by the close alignment of training and validation accuracy curves.

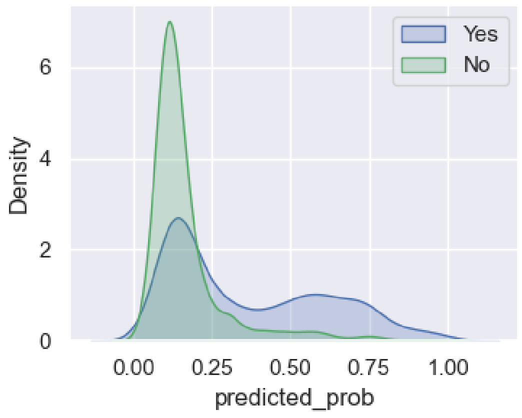


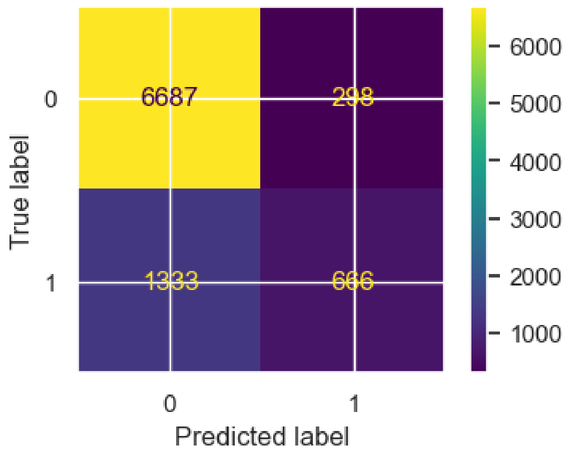


**Model Validation**

The AUC score has improved slightly, from 0.7547 to 0.7595, suggesting a marginal increase in the model's ability to discriminate between positive and negative cases.







Confusion Matrix:

- The performance for class 0 remains similar, with high precision and recall.
- Class 1 still shows lower recall (33%) compared to the previous analysis (36%). This means the model is still struggling to correctly identify some true positives.

Performance Measures:

Overall accuracy remains around 82%, similar to the previous analysis.
Precision and F1-score for class 1 are slightly lower than before, while the gap between recall for both classes has widened.

**Interpretation:**

While Bayesian optimization led to a minor improvement in the AUC score, the overall performance on class 1 has regressed slightly. This suggests that the chosen hyperparameters might not be optimal for addressing the class imbalance or the specific characteristics of class 1 samples.

### Hyperparameter Tuning - Hyperband Optimisation

Hyperband tuner is used with specific parameters:

- Objective: Maximize accuracy.
- Max_epochs: Maximum training epochs for any trial (300)
- Hyperband_iterations: Number of rounds of successive brackets reducing resource allocation.
- Factor: Reduction factor for resources between brackets (10x reduction per round).

Hyperband Optimization Execution:

- Early stopping prevents overfitting by stopping trials with poor validation performance.
- The output shows the completion of the 16th trial and its accuracy (0.7857), along with the current best accuracy (0.8125).
- results_summary displays the top 10 trials with their hyperparameters and scores.

Best Hyperparameters and Interpretation:

- get_best_hyperparameters retrieves the top configuration based on accuracy:
- units: 56 neurons in the first layer.
- activation: relu activation function.

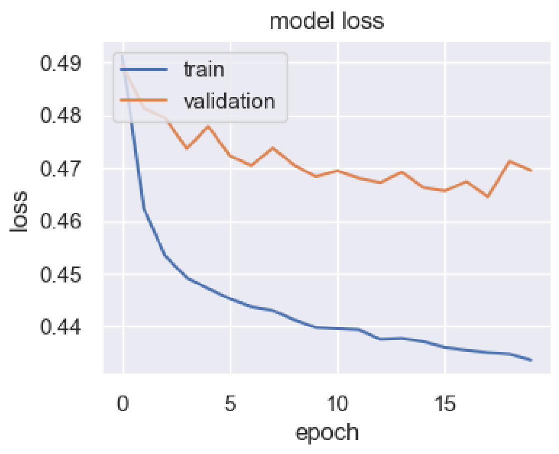
Interpretation:

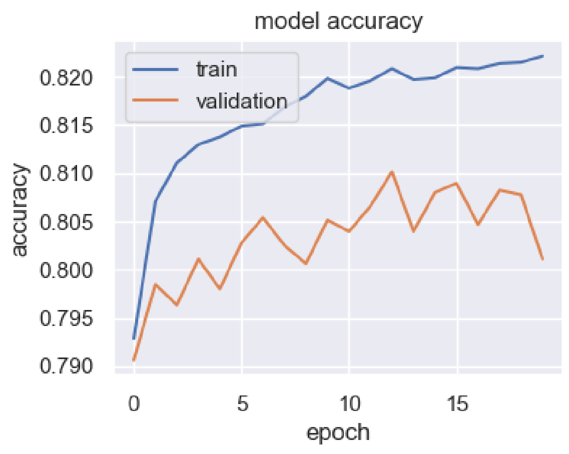
Although Hyperband explored various configurations, the best accuracy achieved (0.8125) is slightly lower than both Bayesian optimization and the previous default parameters. However, Hyperband identified that a higher number of units (56) might be beneficial compared to previous configurations.

The top trials mostly favor relu activation, suggesting its potential suitability for this task.

Building Models with Identified Hyperparameters

- Learning and Convergence: Both training and validation accuracy and loss demonstrate clear learning patterns, converging towards stable values over the 20 epochs. This indicates the model's ability to learn from the data and improve its predictions.
- Final Accuracies: The model achieved a training accuracy of 82.22% and a validation accuracy of 80.11%, suggesting a moderate capability to generalize to unseen data.
- Overfitting: The gap between training and validation accuracy is slightly larger in this run compared to the previous one, hinting at a potential overfitting tendency.
- Dropout Effectiveness: While the dropout layer with a 5% rate has helped to mitigate overfitting to some extent, it might be worth exploring higher dropout rates or additional regularization techniques.

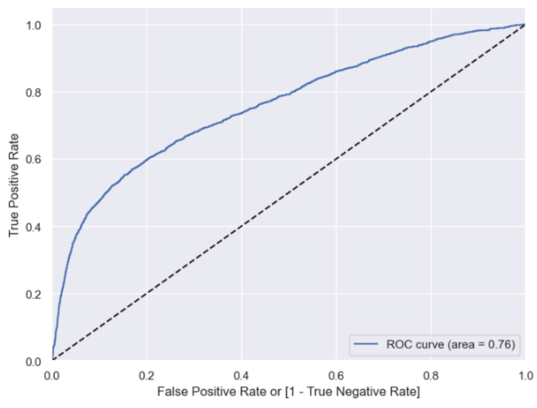


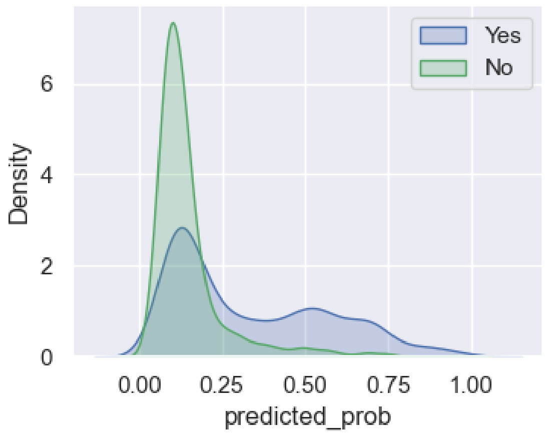


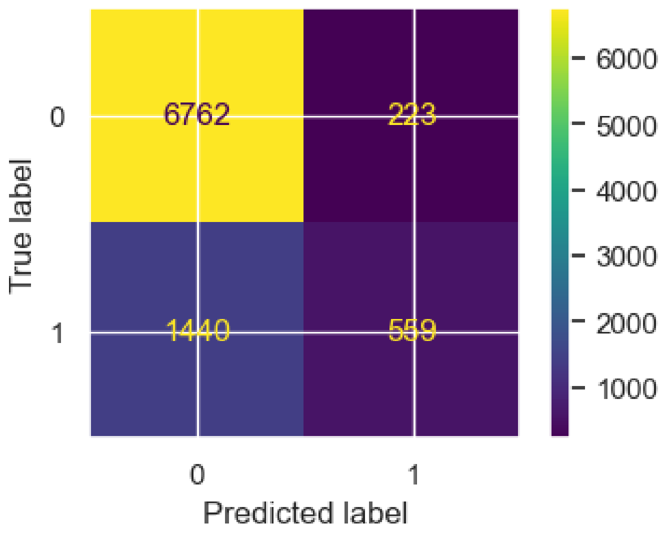
Model Validation

Performance Measures:

- Overall Accuracy: 81%, similar to the previous analyses.
- Class 0: High precision (0.82) and recall (0.97), indicating the model correctly identifies most negative cases.
- Class 1: Lower precision (0.71) and recall (0.28), suggesting the model struggles to accurately identify true positives.
- F1-score: Both classes have lower F1-scores compared to previous analyses, reflecting the imbalanced performance.
- AUC score has slightly decreased to 0.7572, indicating a marginal drop in the model's ability to discriminate between classes.







Confusion Matrix and Kernel Density Plots:

The confusion matrix confirms the class imbalance and the difficulty in identifying true positives. The kernel density plots show some overlap between the predicted probability distributions for both classes, highlighting the challenge of distinguishing them.

**Interpretation:**

While Hyperband helped identify a configuration with similar overall accuracy, it did not improve the performance on the minority class (1). The low recall for this class remains a critical concern. The slight decrease in AUC score and F1-scores further suggests that this hyperparameter configuration might not be optimal for addressing the class imbalance or the specific characteristics of class 1 samples.

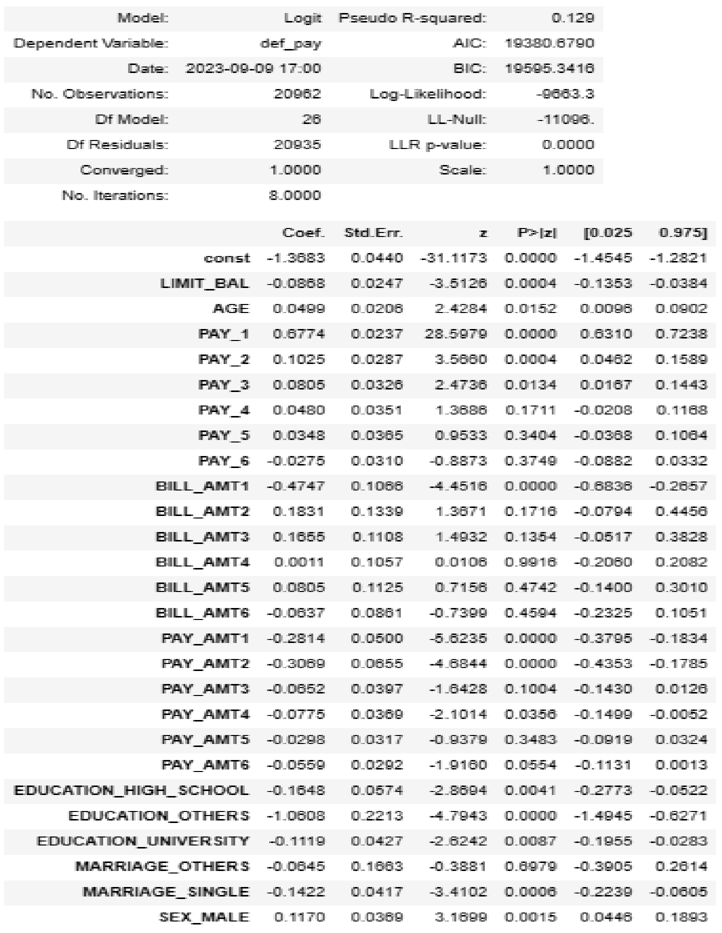
## Logistic Regression Model

Logistic Regression is a ML technique that is used to predict the log-odds of the probability of an event as a linear combination of independent or in other words predictor variables.

- **Splitting dataset into train and test dataset**

Since logistic regression is a linear model, the constant must be included. As a result, a constant is introduced for variable X, and the data is then divided into train and test sets in an 80:20 ratio.

- **Built the logistic regression model and interpreting the summary**



| | | | |
|---|---|---|---|
| Model: | Logit | Pseudo R-squared: | 0.129 |
| Dependent Variable: | def_pay | AIC: | 19380.6790 |
| Date: | 2023-09-09 17:00 | BIC: | 19595.3416 |
| No. Observations: | 20962 | Log-Likelihood: | -9663.3 |
| Df Model: | 26 | LL-Null: | -11096. |
| Df Residuals: | 20935 | LLR p-value: | 0.0000 |
| Converged: | 1.0000 | Scale: | 1.0000 |
| No. Iterations: | 8.0000 | | |

| | Coef. | Std.Err. | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | -1.3683 | 0.0440 | -31.1173 | 0.0000 | -1.4545 | -1.2821 |
| LIMIT_BAL | -0.0868 | 0.0247 | -3.5126 | 0.0004 | -0.1353 | -0.0384 |
| AGE | 0.0499 | 0.0206 | 2.4284 | 0.0152 | 0.0096 | 0.0902 |
| PAY_1 | 0.6774 | 0.0237 | 28.5979 | 0.0000 | 0.6310 | 0.7238 |
| PAY_2 | 0.1025 | 0.0287 | 3.5660 | 0.0004 | 0.0462 | 0.1589 |
| PAY_3 | 0.0805 | 0.0326 | 2.4736 | 0.0134 | 0.0167 | 0.1443 |
| PAY_4 | 0.0480 | 0.0351 | 1.3686 | 0.1711 | -0.0208 | 0.1168 |
| PAY_5 | 0.0348 | 0.0365 | 0.9533 | 0.3404 | -0.0368 | 0.1064 |
| PAY_6 | -0.0275 | 0.0310 | -0.8873 | 0.3749 | -0.0882 | 0.0332 |
| BILL_AMT1 | -0.4747 | 0.1066 | -4.4516 | 0.0000 | -0.6836 | -0.2657 |
| BILL_AMT2 | 0.1831 | 0.1339 | 1.3671 | 0.1716 | -0.0794 | 0.4456 |
| BILL_AMT3 | 0.1655 | 0.1108 | 1.4932 | 0.1354 | -0.0517 | 0.3828 |
| BILL_AMT4 | 0.0011 | 0.1057 | 0.0106 | 0.9916 | -0.2060 | 0.2082 |
| BILL_AMT5 | 0.0805 | 0.1125 | 0.7156 | 0.4742 | -0.1400 | 0.3010 |
| BILL_AMT6 | -0.0637 | 0.0861 | -0.7399 | 0.4594 | -0.2325 | 0.1051 |
| PAY_AMT1 | -0.2814 | 0.0500 | -5.6235 | 0.0000 | -0.3795 | -0.1834 |
| PAY_AMT2 | -0.3069 | 0.0655 | -4.6844 | 0.0000 | -0.4353 | -0.1785 |
| PAY_AMT3 | -0.0652 | 0.0397 | -1.6428 | 0.1004 | -0.1430 | 0.0126 |
| PAY_AMT4 | -0.0775 | 0.0369 | -2.1014 | 0.0356 | -0.1499 | -0.0052 |
| PAY_AMT5 | -0.0298 | 0.0317 | -0.9379 | 0.3483 | -0.0919 | 0.0324 |
| PAY_AMT6 | -0.0559 | 0.0292 | -1.9160 | 0.0554 | -0.1131 | 0.0013 |
| EDUCATION_HIGH_SCHOOL | -0.1648 | 0.0574 | -2.8694 | 0.0041 | -0.2773 | -0.0522 |
| EDUCATION_OTHERS | -1.0608 | 0.2213 | -4.7943 | 0.0000 | -1.4945 | -0.6271 |
| EDUCATION_UNIVERSITY | -0.1119 | 0.0427 | -2.6242 | 0.0087 | -0.1955 | -0.0283 |
| MARRIAGE_OTHERS | -0.0645 | 0.1663 | -0.3881 | 0.6979 | -0.3905 | 0.2614 |
| MARRIAGE_SINGLE | -0.1422 | 0.0417 | -3.4102 | 0.0006 | -0.2239 | -0.0605 |
| SEX_MALE | 0.1170 | 0.0369 | 3.1699 | 0.0015 | 0.0446 | 0.1893 |

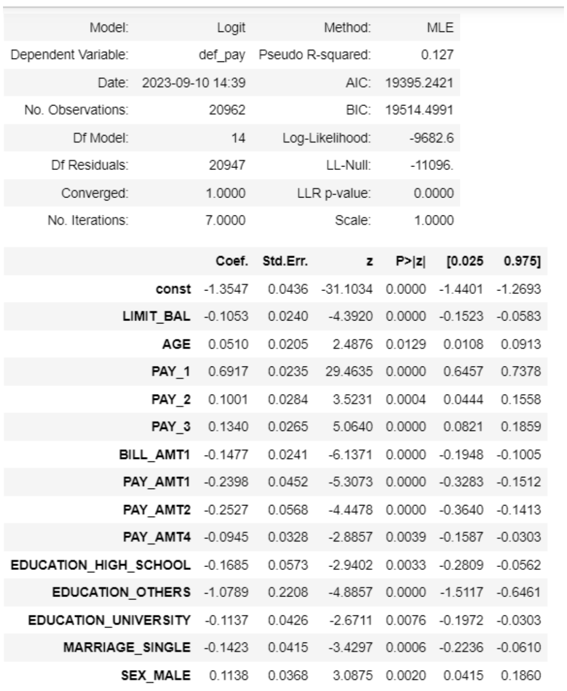
Below is the interpretation of few rows:

The score for the word "LIMIT_BAL" is -0.0808. This means that for every one-unit increase in credit limit balance, the log odds of credit default (e.g., the likelihood of something happening) go down by 0.0808 when all other factors stay the same. The p-value is very low (0.0004), suggesting this is a statistically significant feature.

For "AGE," the coefficient is 0.0499, and the p-value is 0.0152. This means that for every one-unit increase in age, the log odds of credit default (e.g., the likelihood of something happening) go up by 0.0499 when all other factors stay the same. The p-value is also low, suggesting this is a statistically significant feature.

- **Rebuilding model with only the significant variables**

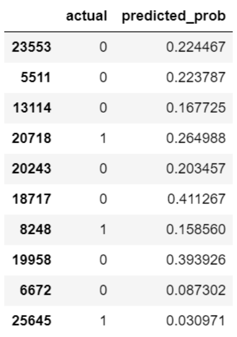
Significant variables are those that have a p-value of less than 0.05 and are utilised to rebuild the logistic regression model.



| | | | |
|---|---|---|---|
| Model: | Logit | Method: | MLE |
| Dependent Variable: | def_pay | Pseudo R-squared: | 0.127 |
| Date: | 2023-09-10 14:39 | AIC: | 19395.2421 |
| No. Observations: | 20962 | BIC: | 19514.4991 |
| Df Model: | 14 | Log-Likelihood: | -9682.6 |
| Df Residuals: | 20947 | LL-Null: | -11096. |
| Converged: | 1.0000 | LLR p-value: | 0.0000 |
| No. Iterations: | 7.0000 | Scale: | 1.0000 |

| | Coef. | Std.Err. | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -1.3547 | 0.0436 | -31.1034 | 0.0000 | -1.4401 | -1.2693 |
| **LIMIT_BAL** | -0.1053 | 0.0240 | -4.3920 | 0.0000 | -0.1523 | -0.0583 |
| **AGE** | 0.0510 | 0.0205 | 2.4876 | 0.0129 | 0.0108 | 0.0913 |
| **PAY_1** | 0.6917 | 0.0235 | 29.4635 | 0.0000 | 0.6457 | 0.7378 |
| **PAY_2** | 0.1001 | 0.0284 | 3.5231 | 0.0004 | 0.0444 | 0.1558 |
| **PAY_3** | 0.1340 | 0.0265 | 5.0640 | 0.0000 | 0.0821 | 0.1859 |
| **BILL_AMT1** | -0.1477 | 0.0241 | -6.1371 | 0.0000 | -0.1948 | -0.1005 |
| **PAY_AMT1** | -0.2398 | 0.0452 | -5.3073 | 0.0000 | -0.3283 | -0.1512 |
| **PAY_AMT2** | -0.2527 | 0.0568 | -4.4478 | 0.0000 | -0.3640 | -0.1413 |
| **PAY_AMT4** | -0.0945 | 0.0328 | -2.8857 | 0.0039 | -0.1587 | -0.0303 |
| **EDUCATION_HIGH_SCHOOL** | -0.1685 | 0.0573 | -2.9402 | 0.0033 | -0.2809 | -0.0562 |
| **EDUCATION_OTHERS** | -1.0789 | 0.2208 | -4.8857 | 0.0000 | -1.5117 | -0.6461 |
| **EDUCATION_UNIVERSITY** | -0.1137 | 0.0426 | -2.6711 | 0.0076 | -0.1972 | -0.0303 |
| **MARRIAGE_SINGLE** | -0.1423 | 0.0415 | -3.4297 | 0.0006 | -0.2236 | -0.0610 |
| **SEX_MALE** | 0.1138 | 0.0368 | 3.0875 | 0.0020 | 0.0415 | 0.1860 |

- **Extracting Predicted Probabilities**

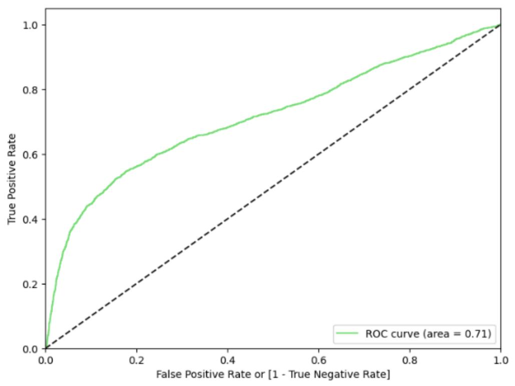


| | actual | predicted_prob |
|---|---|---|
| **23553** | 0 | 0.224467 |
| **5511** | 0 | 0.223787 |
| **13114** | 0 | 0.167725 |
| **20718** | 1 | 0.264988 |
| **20243** | 0 | 0.203457 |
| **18717** | 0 | 0.411267 |
| **8248** | 1 | 0.158560 |
| **19958** | 0 | 0.393926 |
| **6672** | 0 | 0.087302 |
| **25645** | 1 | 0.030971 |

In binary classification, the anticipated probabilities based on a predetermined threshold to create predictions. If we select a level of 0.5, for instance, any predicted probability above 0.5 is classified as "0" and any forecasted probability below 0.5 is classified as "1".

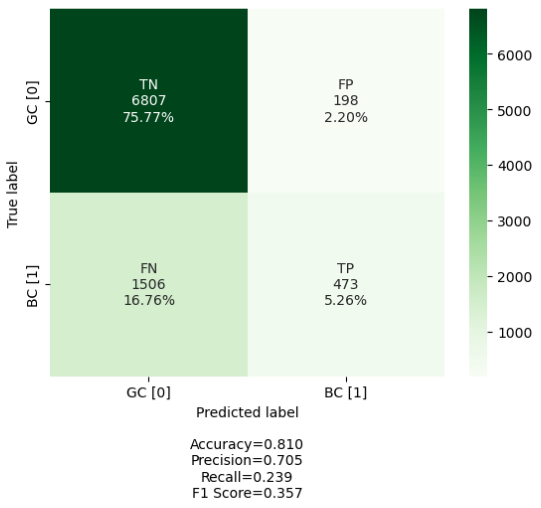
- **Measuring performance via the RoC Curve:**

The AUC achieved here = 0.713, which means the model is useful.



- **Classifying on default threshold 0.5 and building the Confusion Matrix**

  Probability above 0.5 will be categorised as class 1, and probabilities below 0.5 as class 0.



  *The accuracy at threshold 0.5 = 0.810*

- **Interpretation of the Performance of the Confusion Matrix(default threshold)**

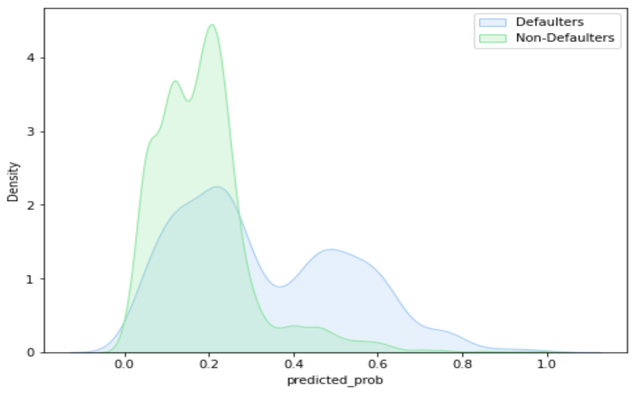
  TP = 473, TN = 6807, FP = 198, FN = 1506

  From all 8313 NO's, 6807 NO's are correctly predicted, and 1506 NO's are predicted as YES.

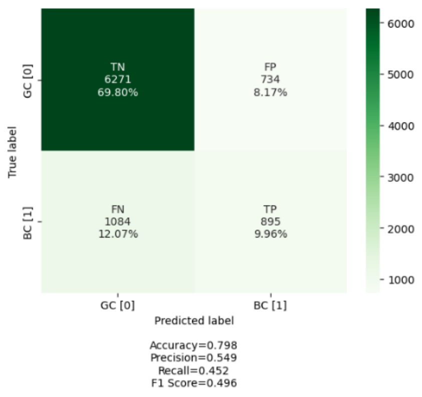
  From all 671 YES, 473 YES are correctly predicted, and 198 YES are predicted as NO.

  The model is found to be 81% accurate.

- **Density vs. Predicted probability plotted to obtain threshold frequency.**
  The threshold value is obtained by intersecting the Yes and No plots and is approximately 0.3.
  Therefore, probability exceeding 0.3 will be considered to be of class 1, and those below 0.3 will be considered to be of class 0.



- **Rebuilding the confusion matrix with a better threshold**



After rebuilding with better threshold, accuracy is 0.798

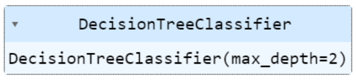
## Decision Tree Classifier

Decision trees are supervised machine learning models that are used to address classification and regression issues. They aid in decision-making by decomposing a problem into a series of if-else-then evaluations that produce a tree-like structure.

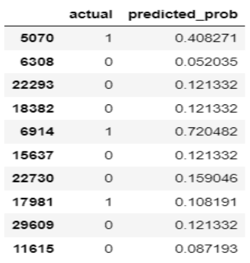
A decision tree constructs itself by dividing different aspects that provide the best information about the target feature until a pure final choice is archived in order to make quality and viable decisions.

### <u>Decision Tree using Gini Criteria</u>

- **Building the tree using Gini Criteria and extracting probabilities**

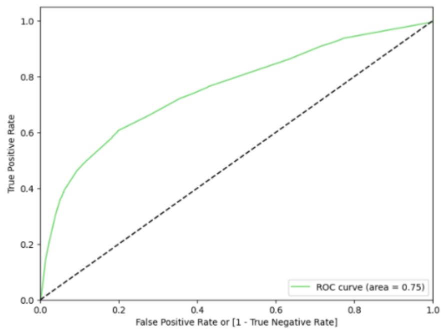


```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```



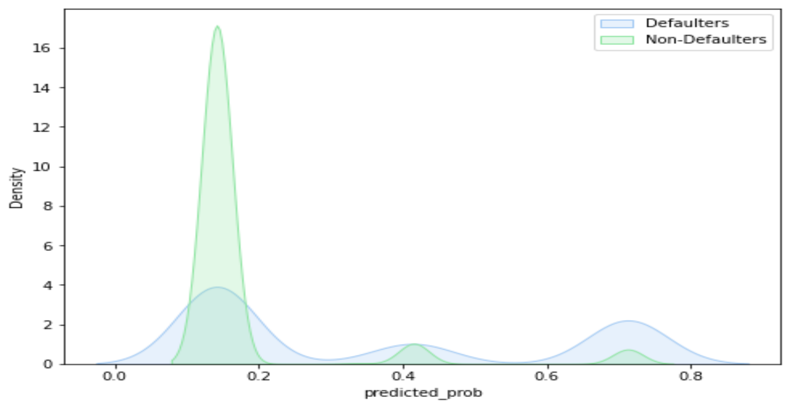
| | actual | predicted_prob |
|---|---|---|
| **5070** | 1 | 0.408271 |
| **6308** | 0 | 0.052035 |
| **22293** | 0 | 0.121332 |
| **18382** | 0 | 0.121332 |
| **6914** | 1 | 0.720482 |
| **15637** | 0 | 0.121332 |
| **22730** | 0 | 0.159046 |
| **17981** | 1 | 0.108191 |
| **29609** | 0 | 0.121332 |
| **11615** | 0 | 0.087193 |

- **Measuring performance via the RoC Curve:**



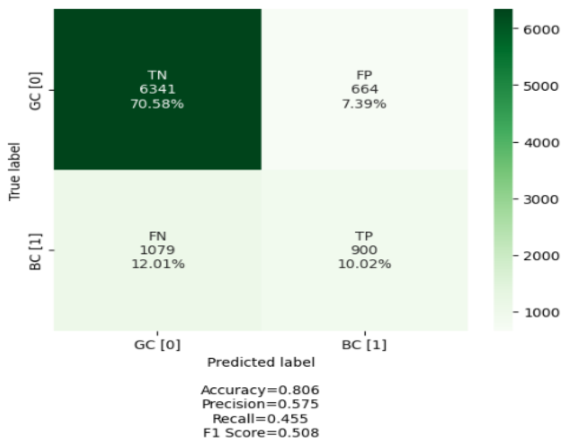
The AUC achieved here = 0.68, which means the model is useful.

- **Plotting Distributions and Identifying optimal probability**



The threshold value is obtained by intersecting the Yes and No plots and is approximately 0.18.
Therefore, probability exceeding 0.18 will be considered to be of class 1, and those below 0.18 will be considered to be of class 0.

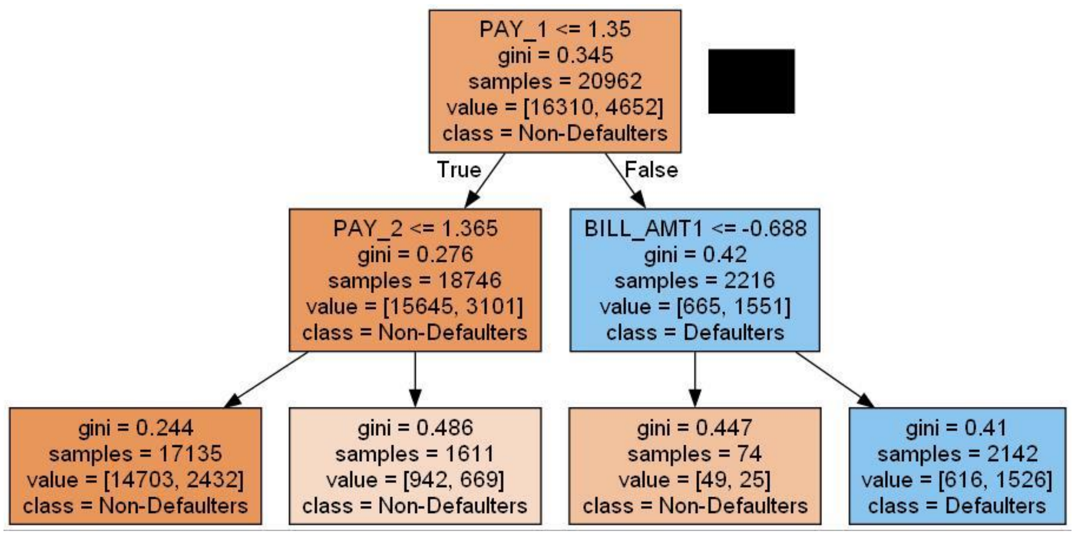
- **Confusion Matrix - Performance Measures**



Accuracy=0.806
Precision=0.575
Recall=0.455
F1 Score=0.508

Interpretation: 7241 were correct predictions and 1743 were incorrect.
Accuracy is 80.6%

- **Visualizing the decision tree**

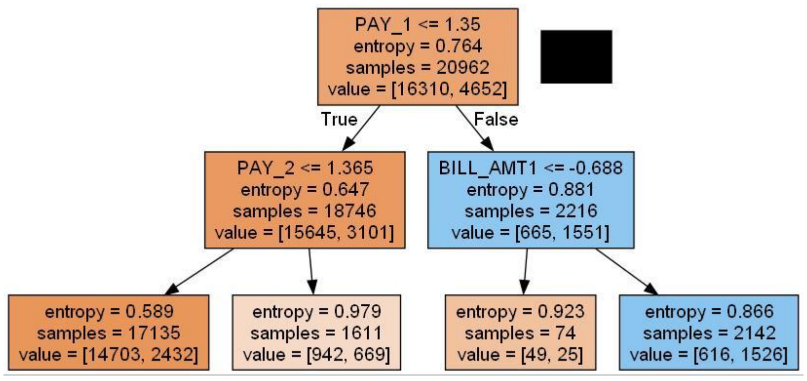

[redacted]

Gini impurity = 0.345

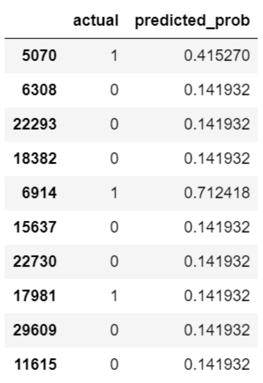
The node reveals some impurity, although only to a relatively low degree, according to the numerical value of 0.345. There is some mixing of different classes at this node.

## Decision Tree using Entropy Criteria

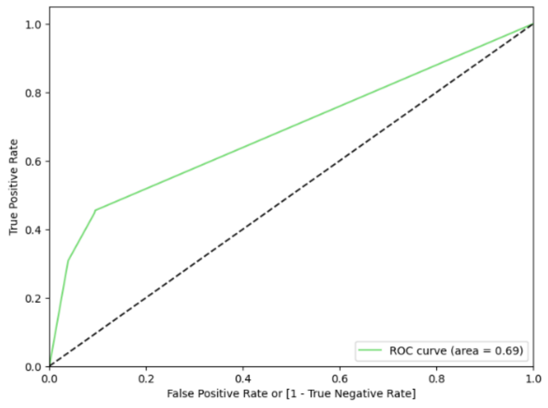
- **Building the tree with entropy criteria and displaying it**


[redacted]

- **Extracting Probabilities and measuring Classifier performance**

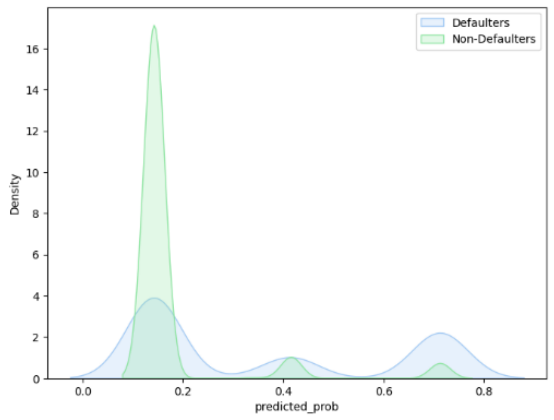


| | actual | predicted_prob |
|---|---|---|
| **5070** | 1 | 0.415270 |
| **6308** | 0 | 0.141932 |
| **22293** | 0 | 0.141932 |
| **18382** | 0 | 0.141932 |
| **6914** | 1 | 0.712418 |
| **15637** | 0 | 0.141932 |
| **22730** | 0 | 0.141932 |
| **17981** | 1 | 0.141932 |
| **29609** | 0 | 0.141932 |
| **11615** | 0 | 0.141932 |



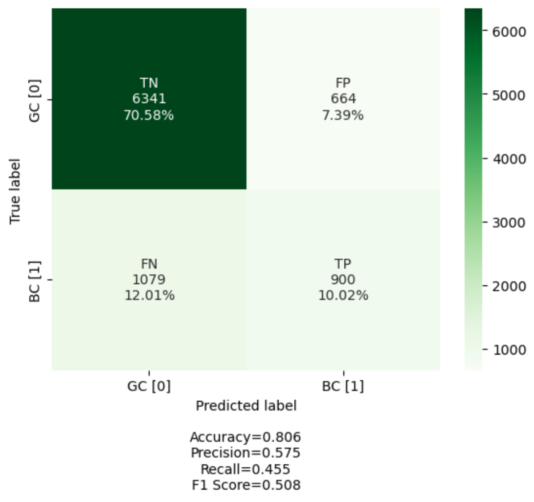
The Classifier performance is found to be 0.69

- **Plotting Distributions and Identifying Optimal Probability**

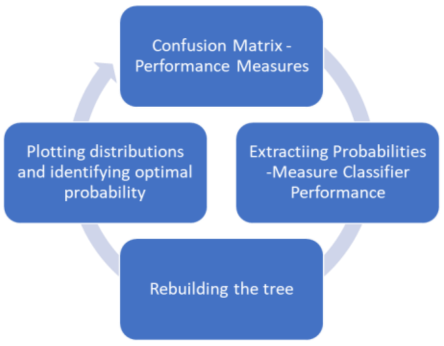


The threshold value is obtained by intersecting the Yes and No plots and is approximately 0.18.
Therefore, probability exceeding 0.18 will be considered to be of class 1, and those below 0.18 will be considered to be of class 0.

- **Confusion Matrix - Performance Measures**

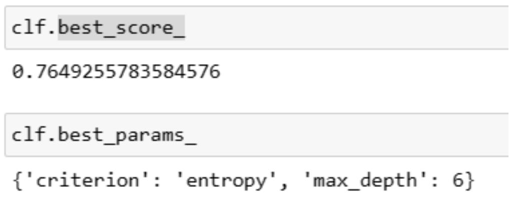


Accuracy=0.806
Precision=0.575
Recall=0.455
F1 Score=0.508


Confusion Matrix - Performance Measures
Extractiing Probabilities -Measure Classifier Performance
Rebuilding the tree
Plotting distributions and identifying optimal probability

**Interpretation:** 7241 were correct predictions and 1743 were incorrect. Accuracy is 80.6%

- **Finding optimal criteria & max depth**

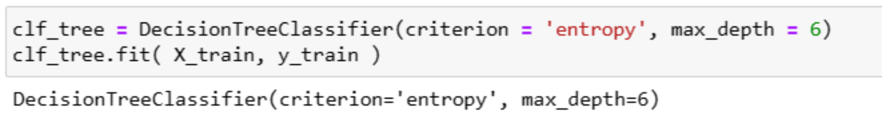


```
clf.best_score_
0.7649255783584576

clf.best_params_
{'criterion': 'entropy', 'max_depth': 6}
```

Best_score_ is a cross-validation score that represents a model's performance and predicts how well it will generalize to new data. We can conclude that the model has been created well in terms of generalizing with unknown data because the cross-validation score is 0.76.

best_params_: Through this, we can determine the tree's ideal depth, which is 6 for the tree to function correctly. As a result, we will classify using the decision tree's depth 6 and also the criteria is 'entropy'.

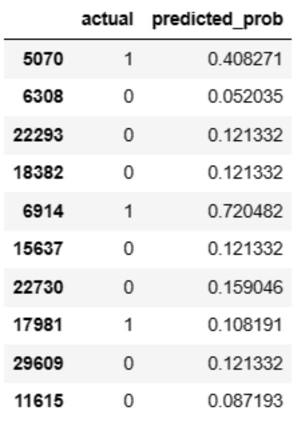
- **Rebuilding the tree with entropy criteria, max depth 6**



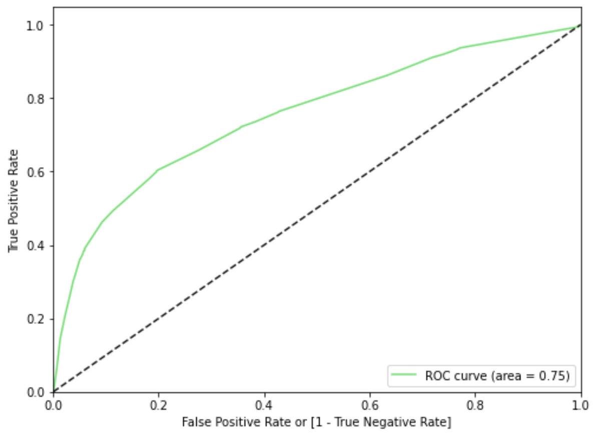
```
clf_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 6)
clf_tree.fit( X_train, y_train )
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=6)
```

- **Extracting Probabilities and Classifier's Performance**

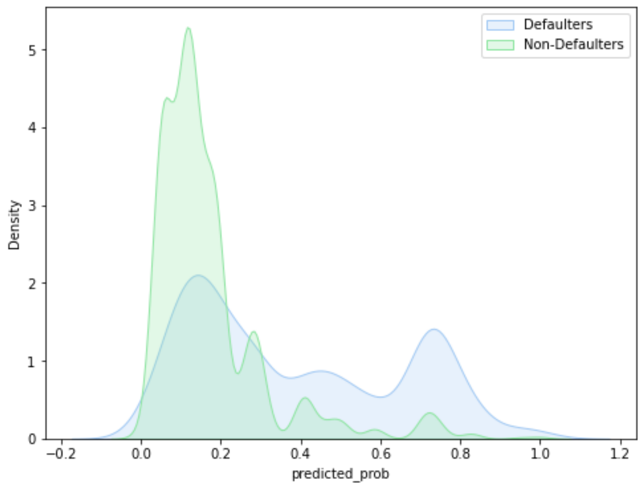


| | actual | predicted_prob |
|---|---|---|
| 5070 | 1 | 0.408271 |
| 6308 | 0 | 0.052035 |
| 22293 | 0 | 0.121332 |
| 18382 | 0 | 0.121332 |
| 6914 | 1 | 0.720482 |
| 15637 | 0 | 0.121332 |
| 22730 | 0 | 0.159046 |
| 17981 | 1 | 0.108191 |
| 29609 | 0 | 0.121332 |
| 11615 | 0 | 0.087193 |



The Classifier performance is found to be 0.75.

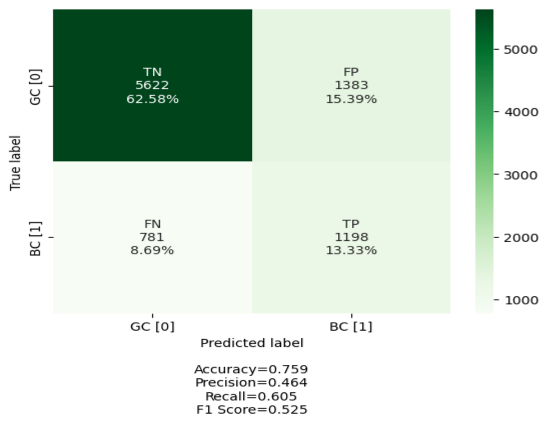
- **Plotting Distributions and Identifying optimal probability**

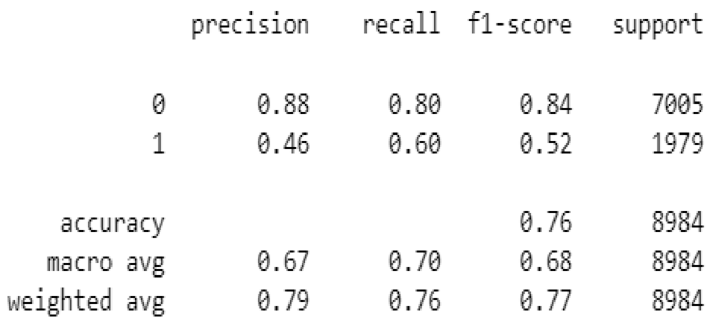


The threshold value is obtained by intersecting the Yes and No plots and is approximately 0.22.

Therefore, probability exceeding 0.22 will be considered to be of class 1, and those below 0.18 will be considered to be of class 0.

- **Final Confusion Matrix and Classifier Interpretations**





```
              precision    recall  f1-score   support

           0       0.88      0.80      0.84      7005
           1       0.46      0.60      0.52      1979

    accuracy                           0.76      8984
   macro avg       0.67      0.70      0.68      8984
weighted avg       0.79      0.76      0.77      8984
```

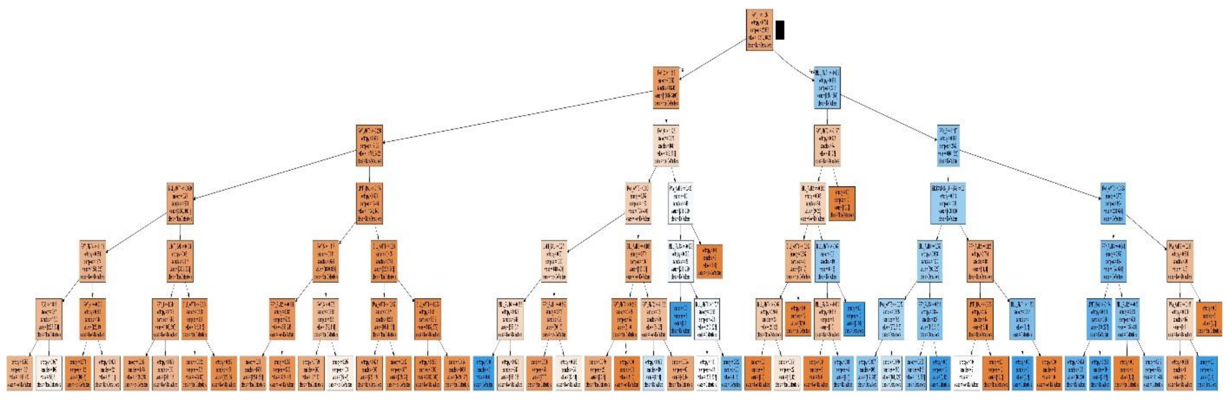
**Interpretation**: 6820 were correct predictions and 2164 were incorrect.
TP = 1198, TN = 5622, FP = 1383. FN = 781

The model is found to be 76% accurate.

**Precision:** Out of all the players that the model predicted would default next month (1) 46% actually did. Out of all the players that the model predicted would not default next month (0), 88% actually did not default.
**Recall:** Out of all the players that actually will default next month, the model only predicted this outcome correctly for 60% of those players. Out of all the players that actually will not default next month, the model predicted this outcome correctly for 80% of those players.
**F-Score:** Since this value is closer to 1, we can say that the model does a fair job of predicting whether or not clients will default.
**Support:** It shows that out of the total data available in the test set, 7005 would not default (having a value of 0), and 1979 would default (having a value of 1).

The model performs reasonably well for class 0 with high precision. However, its performance for class 1 is moderate, with moderate precision. The overall accuracy is 76%, but the F1 score suggests room for improvement, especially for class 1 predictions.

- **Visualizing Decision Tree with depth=6**



## Conclusion

Predictive modelling can help the credit card company predict client defaults. The models assessed show promise, but more refining and feature engineering are needed to find the most important variables. These models can help choose credit card receivers, set credit limits, and analyze consumers' financial behavior, improving risk management and customer satisfaction. Of the metrics used to assess the model performance, instead of accuracy the recall becomes lot more important to judge the robustness of the model as a model with high recall value has low propensity to miss out on credit defaults by discarding a positive as a false negative.

- **Key Variables**

  The evaluation provides insights into model performance, but it may not directly state the essential elements impacting credit card defaults. The primary variables are usually income, credit history, outstanding debt, utilization rate, and other financial indicators. Adding credit scores, work history, and demographic data to boost model predictive power can reveal client creditworthiness.

- **ML Model Selection and Performance**

  The model selection depended largely on the credit provider aims, computational resources, and interpretability. Logistic Regression is a strong candidate due to its AUC and threshold-adjusted performance. The Decision Tree model performs well and may be more interpretable. We also explored how building deep learning neural network frameworks compare with the other ML models for this problem statement.

  In credit default use cases, recall is equally crucial to accuracy to ensure that maximum capture of defaulting agents.

- **Neural networks with optimized hyperparameters**

  The exploration of deep learning models for the binary classification task of predicting credit card defaults has revealed valuable insights and trade-offs through hyperparameter tuning using various optimization methods. The standard Dense Neural Network (DNN) model, though exhibiting promising accuracy, demonstrated limitations in precision and recall, particularly for the identification of defaulting customers.

  Hyperparameter tuning performed using **Grid Search, Random Search, Bayesian Optimization, and Hyperband Optimization** gave us interesting insights. Each method provided a unique set of hyperparameters, influencing the model's performance differently. The results highlight the sensitivity of the model to parameter choices and the need for careful optimization to address the challenges posed by imbalanced data.

**Grid Search identified a combination of parameters (batch size=16, epochs=30, unit=8)** yielding the highest average test score, showcasing an average accuracy of 81.68%. Model validation revealed a moderate AUC score of 0.7333, emphasizing the need for improved discrimination between positive and negative cases. **Random Search identified (unit=12, epochs=45, batch size=32) as the best**, achieving an average score of 81.58%. While offering good performance, it highlighted the model's sensitivity to different parameter choices. Bayesian Optimization refined the model further, achieving an accuracy of 82.47% with a minor increase in AUC score to 0.7595. However, the **struggle to identify true positives in class 1 persisted, indicating the ongoing challenge of addressing class imbalances**. Hyperband Optimization provided insights into the importance of units, favoring a higher number (56) for the first layer.

Comparing the results, it's evident that different optimization methods lead to slightly varied performances. **Bayesian Optimization showed a marginal improvement in overall accuracy, while Hyperband favored a higher number of units**. However, these improvements came at the cost of decreased performance in identifying true positives, especially in class 1. The choice of hyperparameter tuning method should be guided by specific priorities—whether to prioritize overall accuracy, class-specific performance, or a balance between the two. The findings emphasize the need for continuous refinement and consideration of various optimization approaches to strike an optimal balance in credit risk prediction models. The trade-offs observed underscore the challenges posed by imbalanced data and the importance of thorough model evaluation in real-world applications.

- **Business Implications:**

Credit card providers must predict defaults to create educated customer approvals, credit limits, and risk assessments. By identifying high-risk defaulters, the credit issuing agency can reduce losses and personalize loan offerings to specific consumers.

The ML tools provide a strong mathematical backbone to regulate, evaluate and readminister credit related risks and the associated credit default rates. However, the choice of ML or DL model and the data refinements remain key to building strong models as explored in the report. The models should be monitored and updated with fresh data to accommodate for data shifts due to changing customer behaviour and economic conditions.