APAUR REPORT

Credit Card Approval Dataset

GROUP - 3

Akash Kumar Singh - BA005-21
Gourab Dash - BA012-21
Shreyansh Mohanty - BA033-21
Amrita Singh - M009-21
Aakaash Vasu - M150-21

CONTENTS

Introduction

Credit score cards are a common risk control method in the financial industry. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk. Generally speaking, credit score cards are based on historical data. Once encountering large economic fluctuations, past models may lose their original predictive power. Logistic model is a common method for credit scoring.

Data Description

Dataset - 1: Application dataset

Feature name	Explanation	Remarks
ID	Client number	
CODE_GENDER	Gender	
FLAG_OWN_CAR	Is there a car	
FLAG_OWN_REALTY	Is there a property	
CNT_CHILDREN	Number of children	
AMT_INCOME_TOTAL	Annual income	
NAME_INCOME_TYPE	Income category	
NAME_EDUCATION_TYPE	Education level	
NAME_FAMILY_STATUS	Marital status	
NAME_HOUSING_TYPE	Way of living	
DAYS_BIRTH	Birthday	Count backwards from current day (0), -1 means yesterday
DAYS_EMPLOYED	Start date of employment	Count backwards from current day(0). If positive, it means the person currently unemployed.
FLAG_MOBIL	Is there a mobile phone	
FLAG_WORK_PHONE	Is there a work phone	
FLAG_PHONE	Is there a phone	
FLAG_EMAIL	Is there an email	
OCCUPATION_TYPE	Occupation	
CNT_FAM_MEMBERS	Family size	

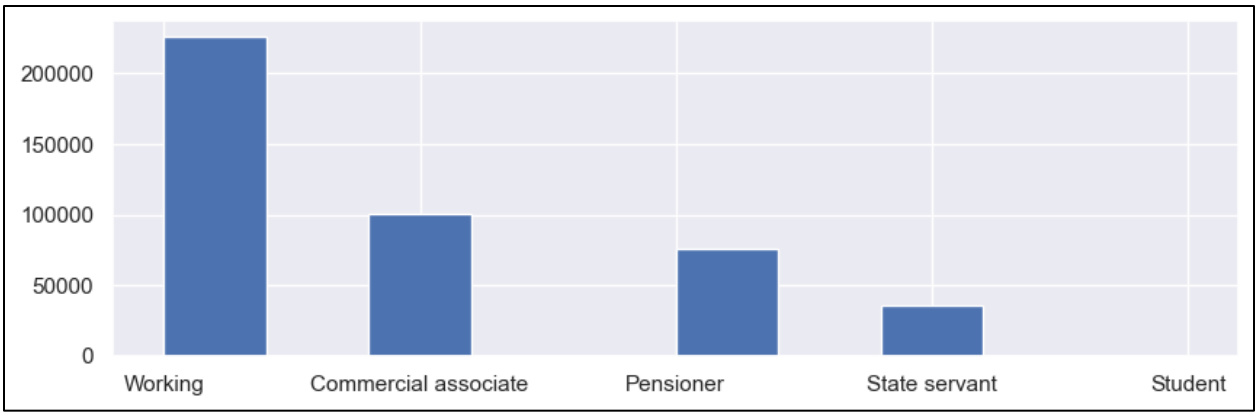
Dataset - 2: Credit Dataset

Feature name	Explanation	Remarks
ID	Client number	
MONTHS_BALANCE	Record month	The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on
STATUS	Status	0: 1-29 days past due 1: 30-59 days past due 2: 60-89 days overdue 3: 90-119 days overdue 4: 120-149 days overdue 5: Overdue or bad debts, write-offs for more than 150 days C: paid off that month X: No loan for the month

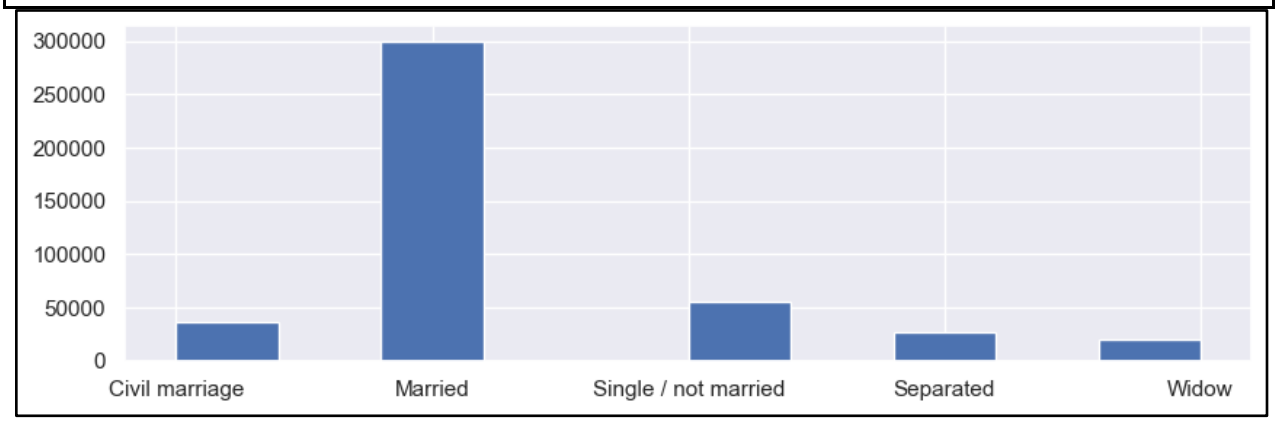
Exploratory Data Analysis

On Application Dataset

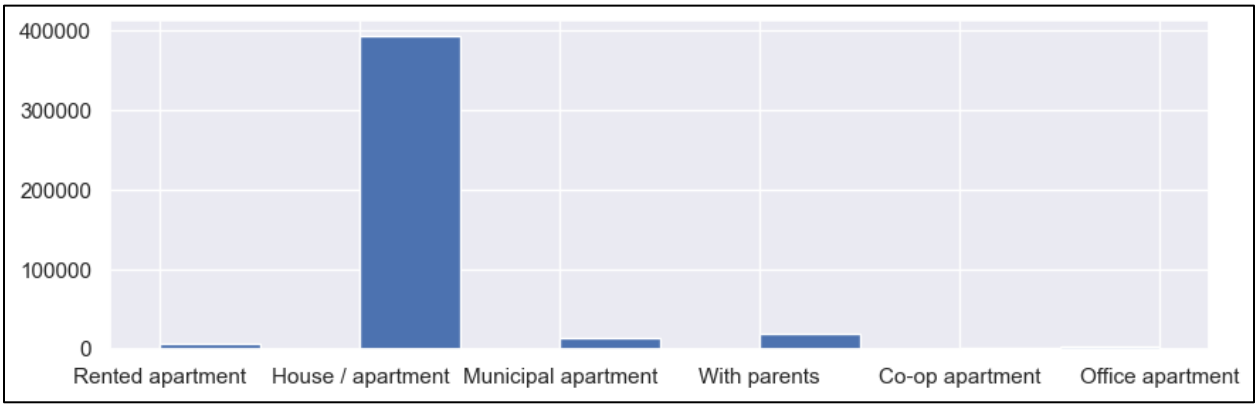
a. Number of applications per income type



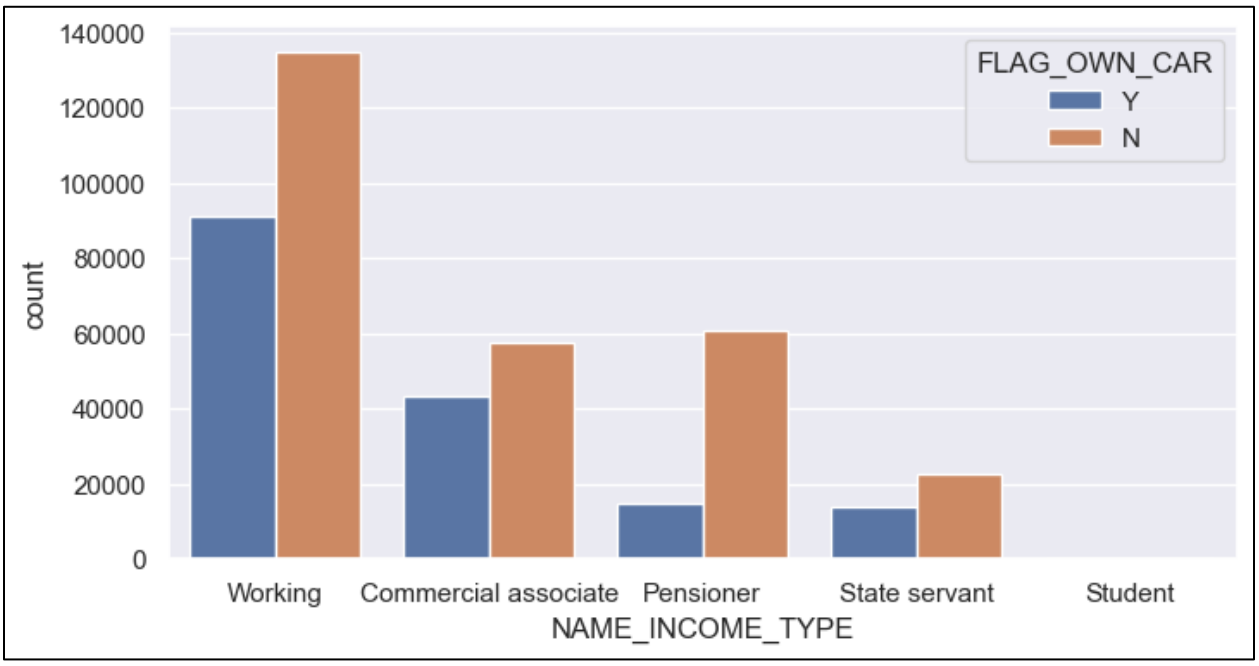
b. Number of applications per marital type



c. Number of applications per housing type

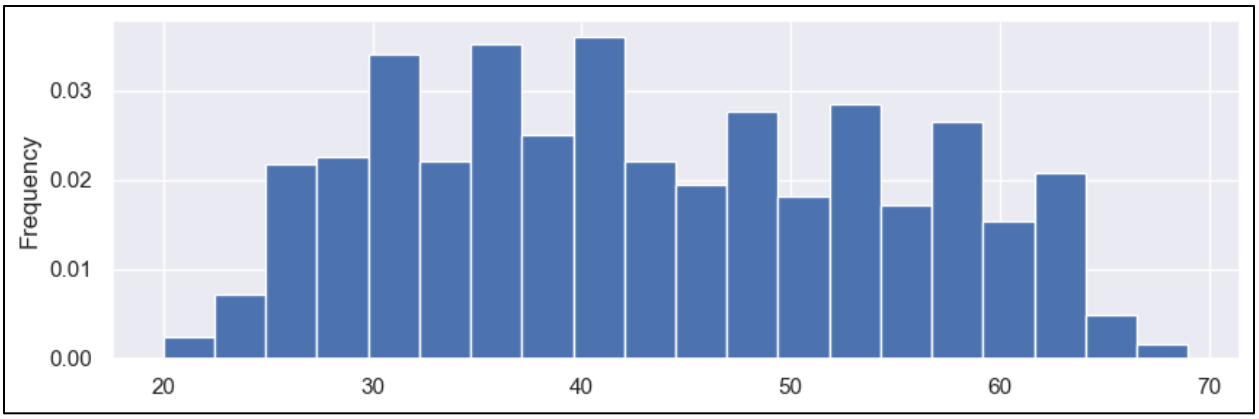


d. Applications by people who own a car by their income type

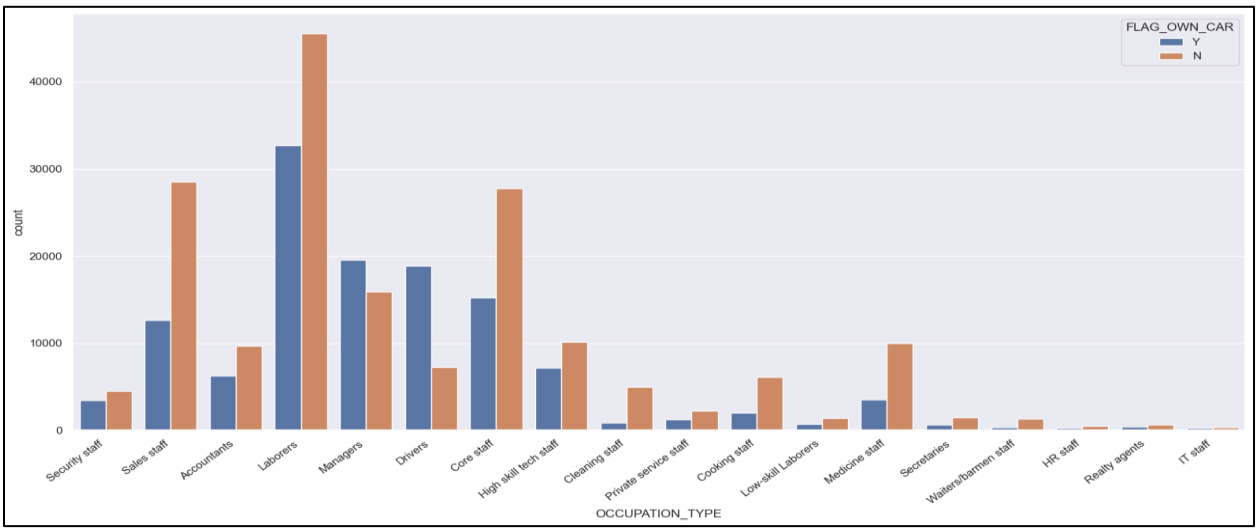


e. Age-wise distribution of applicants

```
sns.set(rc={'figure.figsize': (10, 3)})
application['Age'] = -(application['DAYS_BIRTH'])//365
print(application['Age'].value_counts(bins=10, normalize=True, sort=False))
application['Age'].plot(kind='hist', bins=20, density=True)
plt.show()
```



f. Applications by people owning a car by their occupation type



On Credit Dataset

Overall past-due ratio analysis: Calculating overall past-due rate. This analysis could help us to define who are bad customers.

```
def calculate_rate(pivot_tb, command):
    '''calculate bad customer rate
    '''
    credit0['status'] = None
    exec(command)  # excuate input code
    sumagg = credit0.groupby('ID')['status'].agg(sum)
    pivot_tb = pd.merge(pivot_tb, sumagg, on='ID', how='left')
    pivot_tb.loc[pivot_tb['status'] > 1, 'status'] = 1
    rate = pivot_tb['status'].sum() / len(pivot_tb)
    return round(rate, 5)


command = "credit0.loc[(credit0['STATUS'] == '0') | (credit0['STATUS'] == '1') | (credit0['STATUS'] == '2') | (credit0['STATUS']
morethan1 = calculate_rate(pivot_tb, command)
command = "credit0.loc[(credit0['STATUS'] == '1') | (credit0['STATUS'] == '2') | (credit0['STATUS'] == '3' )| (credit0['STATUS']
morethan30 = calculate_rate(pivot_tb, command)
command = "credit0.loc[(credit0['STATUS'] == '2') | (credit0['STATUS'] == '3' )| (credit0['STATUS'] == '4' )| (credit0['STATUS']
morethan60 = calculate_rate(pivot_tb, command)
command = "credit0.loc[(credit0['STATUS'] == '3' )| (credit0['STATUS'] == '4' )| (credit0['STATUS'] == '5'), 'status'] = 1"
morethan90 = calculate_rate(pivot_tb, command)
command = "credit0.loc[(credit0['STATUS'] == '4' )| (credit0['STATUS'] == '5'), 'status'] = 1"
morethan120 = calculate_rate(pivot_tb, command)
command = "credit0.loc[(credit0['STATUS'] == '5'), 'status'] = 1"
morethan150 = calculate_rate(pivot_tb, command)

summary_dt = pd.DataFrame({'situation': ['past due more than 1 day', 'past due more than 30 days', 'past due more than 60 days',
summary_dt
```

	situation	bad customer ratio
0	past due more than 1 day	0.87054
1	past due more than 30 days	0.11634
2	past due more than 60 days	0.01450
3	past due more than 90 days	0.00720
4	past due more than 120 days	0.00528
5	past due more than 150 days	0.00424

Findings: We could see that almost 87% of users have past due more than 1 day, which is too common, thus it's inappropriate to be a standard. Hence, we have considered users who're due for more than 30 days as the "bad customers" for our model.

Feature Engineering

Transforming the features available to identify “bad customers” for our model:
Based on our EDA of the credit dataset, we have taken 30+ days as the indicator for a customer to be at risk.
Now we have to assign statuses to the credit data set per ID based on the below mapping -

1 : Not at risk (repayment within 30 days or no dues/loans)
0 : At risk (repayment done post 30 days)

Steps:

a. Grouping credit dataset by ID and figuring out the count of repayment statuses per month

```
# Grouping credit dataset by ID & figuring out the count of repayment statuses per month...
df = pd.DataFrame(credit.groupby(["ID", "STATUS"])["STATUS"].count()).rename(columns={'STATUS': 'Freq'}).reset_index()
df
```

	ID	STATUS	Freq
0	5001711	0	3
1	5001711	X	1
2	5001712	0	10
3	5001712	C	9
4	5001713	X	22
...	...	...	...
94138	5150483	X	18
94139	5150484	0	12
94140	5150484	C	1
94141	5150485	0	2
94142	5150487	C	30

94143 rows × 3 columns

b. Mapping 0 & 1 based on our assignment of good and bad customers

```
# Based on our definition of Good/Bad customers, assigning 1 & 0 mapping...
df["STATUS"] = df["STATUS"].map({"X": 1, "C": 1, "0": 1, "1": 0, "2": 0, "3": 0, "4": 0, "5": 0})
df
```

	ID	STATUS	Freq
0	5001711	1	3
1	5001711	1	1
2	5001712	1	10
3	5001712	1	9
4	5001713	1	22
...	...	...	...
94138	5150483	1	18
94139	5150484	1	12
94140	5150484	1	1
94141	5150485	1	2
94142	5150487	1	30

94143 rows × 3 columns

c. Grouping similar statuses (1&0) together and finding aggregate frequency

```
# Grouping similar statuses (1 & 0) together per ID & finding aggregate frequency...
df2 = pd.DataFrame(df.groupby(["ID", "STATUS"])["Freq"].sum()).reset_index()
df2.head(20)
```

	ID	STATUS	Freq
0	5001711	1	4
1	5001712	1	19
2	5001713	1	22
3	5001714	1	15
4	5001715	1	60
5	5001717	1	22
6	5001718	0	2
7	5001718	1	37
8	5001719	1	43
9	5001720	0	7
10	5001720	1	29

d. For IDs with multiple statuses we are taking the status which occurs most frequently & dropping the other statuses. To achieve this, first we order the dataset in descending order of their frequencies per ID and drop the duplicates.

```
# For IDs with multiple statuses, we are taking the status which occurs most frequently & dropping the other status...
# To achieve this, first we order the dataset in descending order of frequencies per ID & drop the duplicates...
df3 = df2.sort_values(by=["ID", "Freq"], ascending=False)
df3.tail(20)
```

	ID	STATUS	Freq
19	5001732	1	36
18	5001731	1	11
17	5001730	1	61
16	5001729	1	7
15	5001728	1	1
14	5001726	1	39
13	5001725	1	8
12	5001724	1	31
11	5001723	1	31
10	5001720	1	29
9	5001720	0	7
8	5001719	1	43

e. Once the new variables are created, we then merge this with the application dataset

```
df4["STATUS"].value_counts()
```

```
1    45800
0      185
Name: STATUS, dtype: int64
```

```
# Creating a copy of the dataframe to be joined to the application dataframe, dropping the frequency
df5 = df4.drop("Freq",axis=1)
```

```
# inner join on ID...
final_df = pd.merge(left=application, right=df5, how="inner", on="ID")
final_df
```

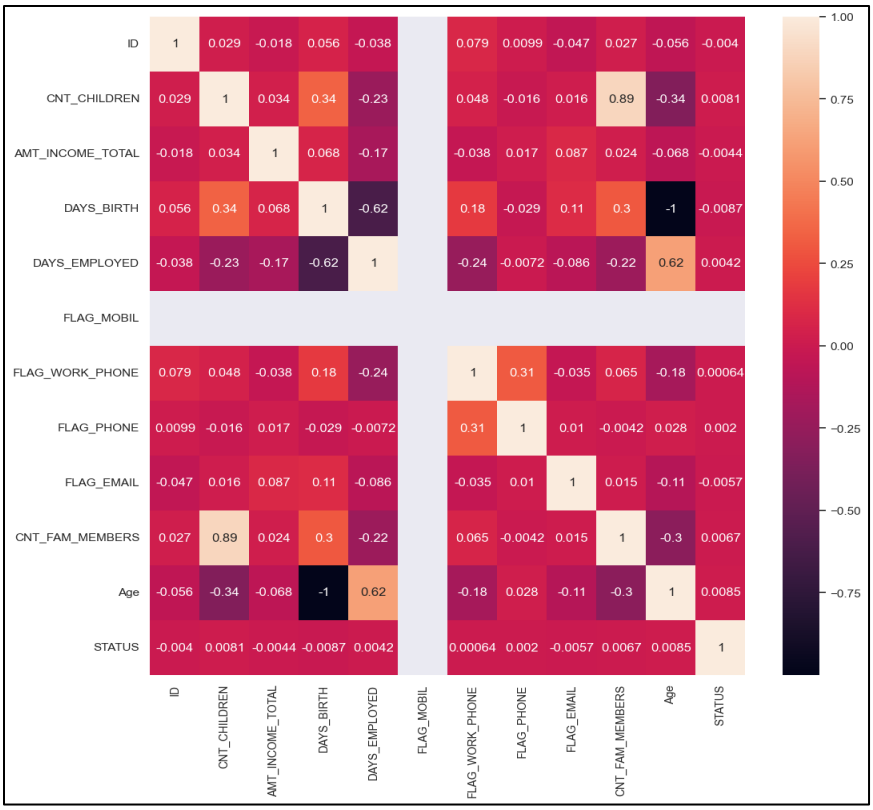
PE	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	OCCUPATION_TYPE	CNT_FAM_MEMBERS	Age	STATUS
nt	-12005	-4542	1	1	0	0	NaN	2.0	32	1
nt	-12005	-4542	1	1	0	0	NaN	2.0	32	1
nt	-21474	-1134	1	0	0	0	Security staff	2.0	58	1
nt	-19110	-3051	1	0	1	1	Sales staff	1.0	52	1

Correlation Analysis

```
# Correlation analysis on the feautres...  (numerical features)
final_df.corr()
```

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE
ID	1.000000	0.028878	-0.017667	0.056016	-0.038043	NaN	0.079215	0.009879
CNT_CHILDREN	0.028878	1.000000	0.033691	0.339357	-0.229379	NaN	0.048091	-0.016291
AMT_INCOME_TOTAL	-0.017667	0.033691	1.000000	0.067908	-0.168611	NaN	-0.037746	0.017245
DAYS_BIRTH	0.056016	0.339357	0.067908	1.000000	-0.616213	NaN	0.179054	-0.028659
DAYS_EMPLOYED	-0.038043	-0.229379	-0.168611	-0.616213	1.000000	NaN	-0.242869	-0.007233
FLAG_MOBIL	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FLAG_WORK_PHONE	0.079215	0.048091	-0.037746	0.179054	-0.242869	NaN	1.000000	0.311644
FLAG_PHONE	0.009879	-0.016291	0.017245	-0.028659	-0.007233	NaN	0.311644	1.000000
FLAG_EMAIL	-0.046979	0.015960	0.086681	0.105625	-0.085648	NaN	-0.034838	0.010455
CNT_FAM_MEMBERS	0.026624	0.889114	0.023750	0.304020	-0.221241	NaN	0.064527	-0.004221
Age	-0.056036	-0.339240	-0.067715	-0.999691	0.616077	NaN	-0.179464	0.028494
STATUS	-0.004003	0.008133	-0.004382	-0.008715	0.004165	NaN	0.000644	0.001990

```
# Heatmap of the correlation analysis...
fig, ax = plt.subplots(figsize=(12,12))
sns.heatmap(final_df.corr(),annot=True)
```



Dummies Creation

```
df6 = pd.get_dummies(temp, columns=['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
                                    'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
                                    'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE'], drop_first=True)
```

```
 #   Column                                               Non-Null Count  Dtype
---  ------                                               --------------  -----
 0   ID                                                   25134 non-null  int64
 1   CNT_CHILDREN                                         25134 non-null  int64
 2   AMT_INCOME_TOTAL                                     25134 non-null  float64
 3   DAYS_BIRTH                                           25134 non-null  int64
 4   DAYS_EMPLOYED                                        25134 non-null  int64
 5   FLAG_MOBIL                                           25134 non-null  int64
 6   FLAG_WORK_PHONE                                      25134 non-null  int64
 7   FLAG_PHONE                                           25134 non-null  int64
 8   FLAG_EMAIL                                           25134 non-null  int64
 9   CNT_FAM_MEMBERS                                      25134 non-null  float64
 10  Age                                                  25134 non-null  int64
 11  STATUS                                               25134 non-null  int64
 12  CODE_GENDER_M                                        25134 non-null  uint8
 13  FLAG_OWN_CAR_Y                                       25134 non-null  uint8
 14  FLAG_OWN_REALTY_Y                                    25134 non-null  uint8
 15  NAME_INCOME_TYPE_Pensioner                           25134 non-null  uint8
 16  NAME_INCOME_TYPE_State servant                       25134 non-null  uint8
 17  NAME_INCOME_TYPE_Student                             25134 non-null  uint8
 18  NAME_INCOME_TYPE_Working                             25134 non-null  uint8
 19  NAME_EDUCATION_TYPE_Higher education                 25134 non-null  uint8
 20  NAME_EDUCATION_TYPE_Incomplete higher                25134 non-null  uint8
 21  NAME_EDUCATION_TYPE_Lower secondary                  25134 non-null  uint8
 22  NAME_EDUCATION_TYPE_Secondary / secondary special    25134 non-null  uint8
 23  NAME_FAMILY_STATUS_Married                           25134 non-null  uint8
```

```
24  NAME_FAMILY_STATUS_Separated                    25134 non-null  uint8
25  NAME_FAMILY_STATUS_Single / not married         25134 non-null  uint8
26  NAME_FAMILY_STATUS_Widow                        25134 non-null  uint8
27  NAME_HOUSING_TYPE_House / apartment             25134 non-null  uint8
28  NAME_HOUSING_TYPE_Municipal apartment           25134 non-null  uint8
29  NAME_HOUSING_TYPE_Office apartment              25134 non-null  uint8
30  NAME_HOUSING_TYPE_Rented apartment              25134 non-null  uint8
31  NAME_HOUSING_TYPE_With parents                  25134 non-null  uint8
32  OCCUPATION_TYPE_Cleaning staff                  25134 non-null  uint8
33  OCCUPATION_TYPE_Cooking staff                   25134 non-null  uint8
34  OCCUPATION_TYPE_Core staff                      25134 non-null  uint8
35  OCCUPATION_TYPE_Drivers                         25134 non-null  uint8
36  OCCUPATION_TYPE_HR staff                        25134 non-null  uint8
37  OCCUPATION_TYPE_High skill tech staff           25134 non-null  uint8
38  OCCUPATION_TYPE_IT staff                        25134 non-null  uint8
39  OCCUPATION_TYPE_Laborers                        25134 non-null  uint8
40  OCCUPATION_TYPE_Low-skill Laborers              25134 non-null  uint8
41  OCCUPATION_TYPE_Managers                        25134 non-null  uint8
42  OCCUPATION_TYPE_Medicine staff                  25134 non-null  uint8
43  OCCUPATION_TYPE_Private service staff           25134 non-null  uint8
44  OCCUPATION_TYPE_Realty agents                   25134 non-null  uint8
45  OCCUPATION_TYPE_Sales staff                     25134 non-null  uint8
46  OCCUPATION_TYPE_Secretaries                     25134 non-null  uint8
47  OCCUPATION_TYPE_Security staff                  25134 non-null  uint8
48  OCCUPATION_TYPE_Waiters/barmen staff            25134 non-null  uint8
```

Model Creation & Selection

We have applied Logistic Regression and Decision Tree on this dataset.
First, we used Logistic Regression to figure out the significant features. After we got the significant features after looking at the p-values, we then tried to use the significant features only and build a regression model to check the accuracy of the same.
After that we used the same significant features to build a decision tree.
Finally, we compare the accuracy of both the results obtained to imply which one to use for our prediction model.

Logistic Regression - Model 1

```
# Logistic Regression Model-1...

import statsmodels.api as sm

X = sm.add_constant(X)

model = sm.Logit(y, X)
result = model.fit(method='newton')

# Evaluate Model...

display(result.summary())
```

Logit Regression Results

Dep. Variable:	STATUS	**No. Observations:**	25134
Model:	Logit	**Df Residuals:**	25087
Method:	MLE	**Df Model:**	46
Date:	Tue, 20 Dec 2022	**Pseudo R-squ.:**	0.1001
Time:	16:59:22	**Log-Likelihood:**	-689.67
converged:	False	**LL-Null:**	-766.39
Covariance Type:	nonrobust	**LLR p-value:**	1.755e-13

	coef	std err	z	P>|z|	[0.025	0.975]
CNT_CHILDREN	4.9942	0.627	7.960	0.000	3.764	6.224
AMT_INCOME_TOTAL	8.486e-08	9.59e-07	0.088	0.930	-1.8e-06	1.97e-06
DAYS_BIRTH	-0.0005	0.001	-0.571	0.568	-0.002	0.001
DAYS_EMPLOYED	-0.0002	5.72e-05	-2.662	0.008	-0.000	-4.01e-05
FLAG_MOBIL	24.3168	523.774	0.046	0.963	-1002.261	1050.895
FLAG_WORK_PHONE	0.0794	0.239	0.333	0.739	-0.389	0.548
FLAG_PHONE	0.1530	0.226	0.677	0.499	-0.290	0.596
FLAG_EMAIL	-0.2100	0.278	-0.755	0.450	-0.755	0.335
CNT_FAM_MEMBERS	-4.6871	0.595	-7.883	0.000	-5.852	-3.522
Age	-0.1726	0.326	-0.530	0.596	-0.811	0.466
CODE_GENDER_M	-0.6342	0.244	-2.598	0.009	-1.113	-0.156
FLAG_OWN_CAR_Y	0.1283	0.210	0.610	0.542	-0.284	0.541
FLAG_OWN_REALTY_Y	0.0706	0.206	0.342	0.732	-0.334	0.475
NAME_INCOME_TYPE_Pensioner	-4.1999	0.714	-5.882	0.000	-5.599	-2.800
NAME_INCOME_TYPE_State servant	1.0405	0.458	2.270	0.023	0.142	1.939
NAME_INCOME_TYPE_Student	24.6789	1.03e+06	2.4e-05	1.000	-2.02e+06	2.02e+06
NAME_INCOME_TYPE_Working	0.3317	0.203	1.637	0.102	-0.065	0.729
NAME_EDUCATION_TYPE_Higher education	-9.8698	523.771	-0.019	0.985	-1036.442	1016.702
NAME_EDUCATION_TYPE_Incomplete higher	-9.9928	523.771	-0.019	0.985	-1036.565	1016.580
NAME_EDUCATION_TYPE_Lower secondary	-10.5966	523.771	-0.020	0.984	-1037.170	1015.977
NAME_EDUCATION_TYPE_Secondary / secondary special	-9.8364	523.771	-0.019	0.985	-1036.409	1016.736
NAME_FAMILY_STATUS_Married	-0.5398	0.408	-1.322	0.186	-1.340	0.260
NAME_FAMILY_STATUS_Separated	-5.0276	0.850	-5.916	0.000	-6.693	-3.362
NAME_FAMILY_STATUS_Single / not married	-5.1837	0.662	-7.825	0.000	-6.482	-3.885
NAME_FAMILY_STATUS_Widow	-5.8979	0.898	-6.571	0.000	-7.657	-4.139

NAME_HOUSING_TYPE_House / apartment	0.1752	1.021	0.172	0.864	-1.826	2.176
NAME_HOUSING_TYPE_Municipal apartment	-0.3416	1.099	-0.311	0.756	-2.495	1.812
NAME_HOUSING_TYPE_Office apartment	-1.6687	1.118	-1.493	0.136	-3.860	0.523
NAME_HOUSING_TYPE_Rented apartment	1.1943	1.447	0.825	0.409	-1.642	4.030
NAME_HOUSING_TYPE_With parents	0.6522	1.102	0.592	0.554	-1.508	2.812
OCCUPATION_TYPE_Cleaning staff	-0.6513	0.862	-0.756	0.450	-2.341	1.038
OCCUPATION_TYPE_Cooking staff	-0.9305	0.793	-1.174	0.240	-2.484	0.623
OCCUPATION_TYPE_Core staff	-0.7698	0.643	-1.197	0.231	-2.031	0.491
OCCUPATION_TYPE_Drivers	-0.1777	0.713	-0.249	0.803	-1.575	1.219
OCCUPATION_TYPE_HR staff	13.8238	2048.478	0.007	0.995	-4001.120	4028.767
OCCUPATION_TYPE_High skill tech staff	-0.3159	0.784	-0.403	0.687	-1.852	1.220
OCCUPATION_TYPE_IT staff	-3.2600	0.779	-4.186	0.000	-4.786	-1.734
OCCUPATION_TYPE_Laborers	-0.0192	0.663	-0.029	0.977	-1.318	1.280
OCCUPATION_TYPE_Managers	-0.6016	0.658	-0.914	0.361	-1.892	0.689
OCCUPATION_TYPE_Medicine staff	-1.4977	0.705	-2.123	0.034	-2.880	-0.115
OCCUPATION_TYPE_Private service staff	-0.3606	1.169	-0.308	0.758	-2.652	1.931
OCCUPATION_TYPE_Realty agents	16.4092	8867.568	0.002	0.999	-1.74e+04	1.74e+04
OCCUPATION_TYPE_Sales staff	-0.5228	0.652	-0.801	0.423	-1.801	0.756
OCCUPATION_TYPE_Secretaries	-1.3049	1.172	-1.114	0.265	-3.602	0.992
OCCUPATION_TYPE_Security staff	-1.4095	0.722	-1.952	0.051	-2.825	0.006
OCCUPATION_TYPE_Waiters/barmen staff	-0.7706	1.178	-0.654	0.513	-3.079	1.538

Hypothesis to be considered:

H0 : The coefficient of the parameter is not significant in the LR model

HA : The coefficient of the parameter is significant in the LR model

Hypothesis testing:

If p-value < alpha (0.05) => Reject Null Hypothesis

Conclusions based on Model-1 p-value analysis:

1. Education type => Insignificant (along w/dummies)
2. Housing type => Insignificant (along w/dummies)
3. Occupation type => Insignificant (most of them; hence removing the entire feature)
4. Keeping Amount_Income => Since it seems to be a feature which realistically should be contributing to the model

Logistic Regression - Model 2

```
# Logistic Regression Model-2...
X2 = sm.add_constant(X2)

model = sm.Logit(y, X2)
result = model.fit(method='newton')

# Evaluate Model...

display(result.summary())
```

Logit Regression Results

Dep. Variable:	STATUS	**No. Observations:**	25134
Model:	Logit	**Df Residuals:**	25120
Method:	MLE	**Df Model:**	13
Date:	Tue, 20 Dec 2022	**Pseudo R-squ.:**	0.06148
Time:	19:43:32	**Log-Likelihood:**	-719.27
converged:	False	**LL-Null:**	-766.39
Covariance Type:	nonrobust	**LLR p-value:**	2.146e-14

	coef	std err	z	P>|z|	[0.025	0.975]
const	13.8897	1.240	11.204	0.000	11.460	16.320
CNT_CHILDREN	4.5861	0.603	7.604	0.000	3.404	5.768
AMT_INCOME_TOTAL	1.355e-07	9.21e-07	0.147	0.883	-1.67e-06	1.94e-06
DAYS_EMPLOYED	-0.0002	5.53e-05	-3.049	0.002	-0.000	-6.03e-05
CNT_FAM_MEMBERS	-4.3277	0.577	-7.497	0.000	-5.459	-3.196
CODE_GENDER_M	-0.5089	0.194	-2.627	0.009	-0.889	-0.129
NAME_INCOME_TYPE_Pensioner	-4.1312	0.688	-6.006	0.000	-5.479	-2.783
NAME_INCOME_TYPE_State servant	0.7472	0.443	1.688	0.091	-0.120	1.615
NAME_INCOME_TYPE_Student	15.9774	1.48e+04	0.001	0.999	-2.9e+04	2.9e+04
NAME_INCOME_TYPE_Working	0.3726	0.200	1.864	0.062	-0.019	0.764
NAME_FAMILY_STATUS_Married	-0.4135	0.399	-1.037	0.300	-1.195	0.368
NAME_FAMILY_STATUS_Separated	-4.5355	0.825	-5.495	0.000	-6.153	-2.918
NAME_FAMILY_STATUS_Single / not married	-4.7405	0.650	-7.289	0.000	-6.015	-3.466
NAME_FAMILY_STATUS_Widow	-5.3978	0.860	-6.278	0.000	-7.083	-3.712

Model Accuracy: Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

# Test train split of data...
x = pd.DataFrame(df6[["CNT_CHILDREN",
                      "AMT_INCOME_TOTAL",
                      "DAYS_EMPLOYED",
                      "CNT_FAM_MEMBERS",
                      "CODE_GENDER_M",
                      "NAME_INCOME_TYPE_Pensioner",
                      "NAME_INCOME_TYPE_State servant",
                      "NAME_INCOME_TYPE_Student",
                      "NAME_INCOME_TYPE_Working",
                      "NAME_FAMILY_STATUS_Married",
                      "NAME_FAMILY_STATUS_Separated",
                      "NAME_FAMILY_STATUS_Single / not married",
                      "NAME_FAMILY_STATUS_Widow"]])
y = pd.DataFrame(df6[['STATUS']])
trainX, testX, trainY, testY = train_test_split(x, y, test_size=0.2)
```

```
# Binomial Logistic Regression Model Defination...
log_reg = LogisticRegression(solver='newton-cg')
log_reg.fit(trainX, trainY)
y_pred = log_reg.predict(testX)

print('Accuracy: {:.2f}'.format(accuracy_score(testY, y_pred)))
print('Error rate: {:.2f}'.format(1 - accuracy_score(testY, y_pred)))
```

```
Accuracy: 1.00
Error rate: 0.00
```

```
df6["STATUS"].value_counts()

1    25013
0      121
```

We observed that due to the unbalanced nature of the dataset, we are getting a over-fitted model. To overcome this issue, we have applied SMOTE oversampling technique on the minority class.

Oversampling: SMOTE

One approach to addressing imbalanced datasets is to oversample the minority class. The simplest approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the Synthetic Minority Oversampling Technique, or SMOTE for short.

```
from imblearn.over_sampling import SMOTE

# y_train = trainY.astype('int')
X_balance, Y_balance = SMOTE().fit_resample(trainX, trainY)
X_balance = pd.DataFrame(X_balance, columns=trainX.columns)
Y_balance = pd.DataFrame(Y_balance, columns=["STATUS"])
```

```
Y_balance["STATUS"].value_counts()
```

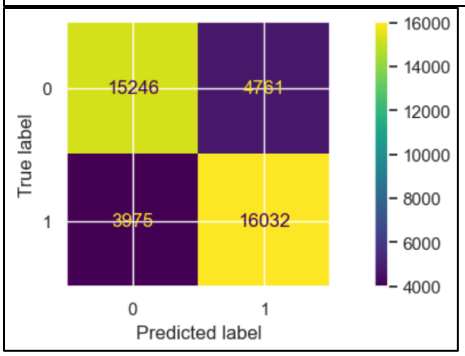
```
1    20010
0    20010
```

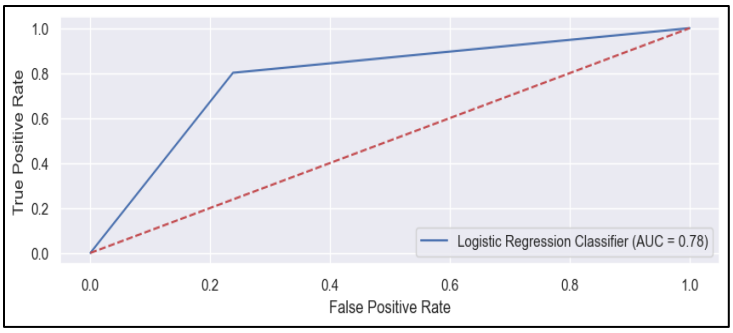
Model Accuracy on training data: Logistic regression

```
# Applying LR model again using balanced datasets...

# Binomial Logistic Regression Model Defination...
log_reg = LogisticRegression(solver='newton-cg')
log_reg.fit(X_balance, Y_balance)
y_pred_test = log_reg.predict(testX)
y_pred_train = log_reg.predict(X_balance)

print('Accuracy: {:.2f}'.format(accuracy_score(testY, y_pred_test)))
print('Error rate: {:.2f}'.format(1 - accuracy_score(testY, y_pred_test)))
```

```
# Plotting confusion matrix for training dataset from Logistic Regression Model...
plot_confusion_matrix(log_reg, X_balance, Y_balance)
print(classification_report(Y_balance, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.79      0.76      0.78     20007
           1       0.77      0.80      0.79     20007

    accuracy                           0.78     40014
   macro avg       0.78      0.78      0.78     40014
weighted avg       0.78      0.78      0.78     40014
```



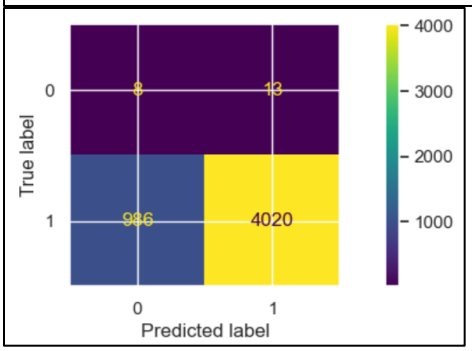


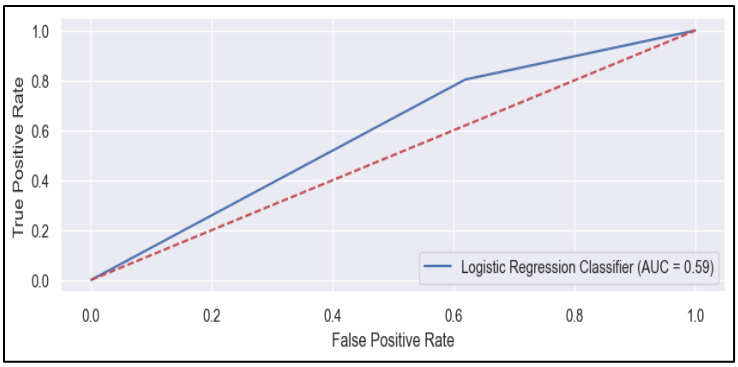
Model Accuracy on test data: Logistic regression

```
# Applying LR model again using balanced datasets...

# Binomial Logistic Regression Model Defination...
log_reg = LogisticRegression(solver='newton-cg')
log_reg.fit(X_balance, Y_balance)
y_pred_test = log_reg.predict(testX)
y_pred_train = log_reg.predict(X_balance)

print('Accuracy: {:.2f}'.format(accuracy_score(testY, y_pred_test)))
print('Error rate: {:.2f}'.format(1 - accuracy_score(testY, y_pred_test)))
```

```
# Plotting confusion matrix for test dataset from Logistic Regression Model...
plot_confusion_matrix(log_reg, testX, testY)
print(classification_report(testY, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.01      0.38      0.02        21
           1       1.00      0.80      0.89      5006

    accuracy                           0.80      5027
   macro avg       0.50      0.59      0.45      5027
weighted avg       0.99      0.80      0.89      5027
```



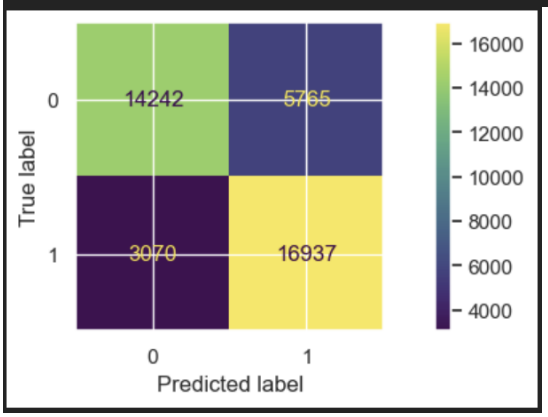


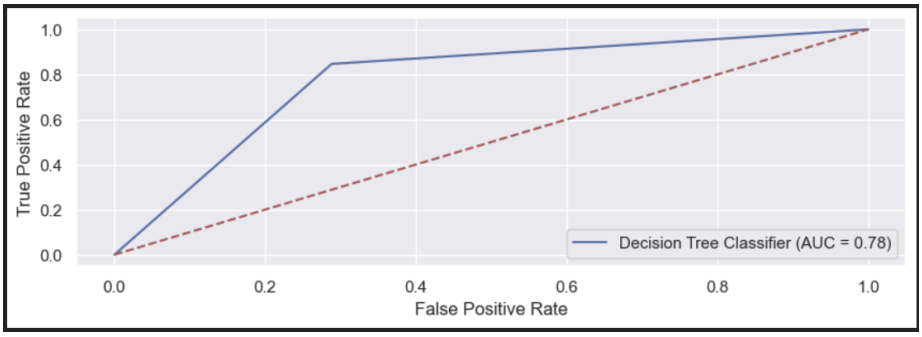
Model Accuracy on Training Data: Decision Tree

```
# Applying the model on the Train dataset...
y_pred_dt_train = dtclf.predict(X_balance)
print (classification_report(Y_balance, y_pred_dt_train))
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(Y_balance,y_pred_dt_train,
drop_intermediate=False)
roc_auc = metrics.auc(fpr_tr, tpr_tr)
disp = metrics.RocCurveDisplay(fpr=fpr_tr, tpr=tpr_tr, roc_auc=roc_auc,
estimator_name='Decision Tree Classifier')
disp.plot()  # doctest: +SKIP
plt.plot([0, 1], [0, 1],'r--')
plt.show()
cm_global = confusion_matrix(Y_balance, y_pred_dt_train)
cm_display_global = ConfusionMatrixDisplay(cm_global,display_labels=[0,1]).plot()
```

Python

	precision	recall	f1-score	support
0	0.82	0.71	0.76	20007
1	0.75	0.85	0.79	20007
accuracy			0.78	40014
macro avg	0.78	0.78	0.78	40014
weighted avg	0.78	0.78	0.78	40014



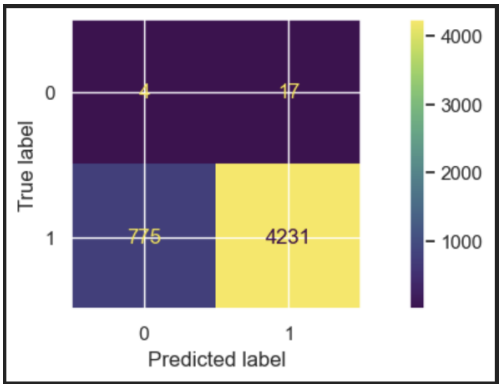


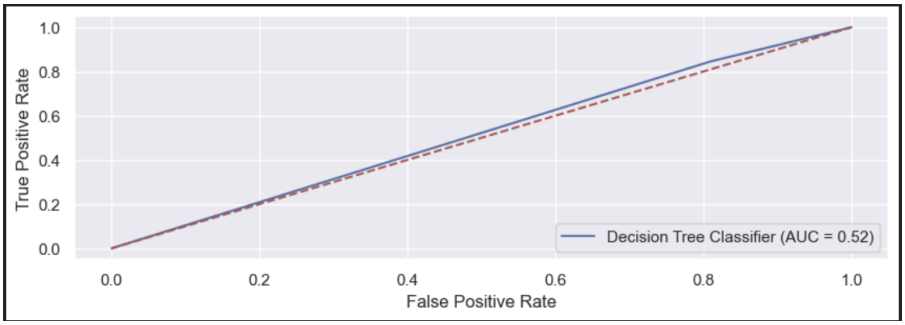
Model Accuracy on test data: Decision Tree

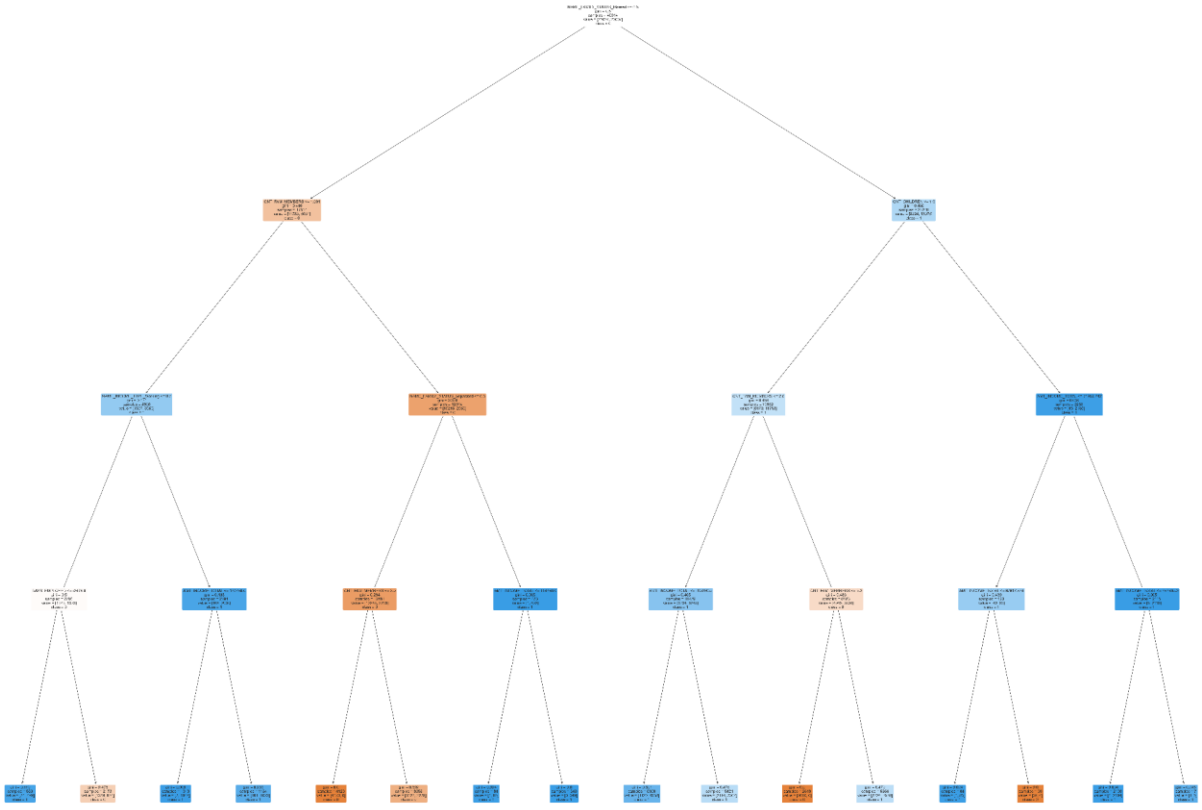
```
dtclf = DecisionTreeClassifier (max_depth = 4, random_state = 12)
dtclf = dtclf.fit(X_balance, Y_balance)

# Applying the model on the Test dataset...
y_pred_dt_test = dtclf.predict(testX)
print (classification_report(testY, y_pred_dt_test))
fpr_t, tpr_t, thresholds_t = metrics.roc_curve(testY,y_pred_dt_test,drop_intermediate=False)
roc_auc = metrics.auc(fpr_t, tpr_t)
disp = metrics.RocCurveDisplay(fpr=fpr_t, tpr=tpr_t, roc_auc=roc_auc,
estimator_name='Decision Tree Classifier')
disp.plot()  # doctest: +SKIP
plt.plot([0, 1], [0, 1],'r--')
plt.show()
cm_global = confusion_matrix(testY, y_pred_dt_test)
cm_display_global = ConfusionMatrixDisplay(cm_global,display_labels=[0,1]).plot()
```

Python

	precision	recall	f1-score	support
0	0.01	0.19	0.01	21
1	1.00	0.85	0.91	5006
accuracy			0.84	5027
macro avg	0.50	0.52	0.46	5027
weighted avg	0.99	0.84	0.91	5027







Conclusion : Summary Table

```python
classifiers = {"LogisticRegression" : LogisticRegression(solver="newton-cg"),
"DecisionTree" : DecisionTreeClassifier(max_depth= 4, random_state = 12)}
result_table = pd.DataFrame(columns=['classifiers','accuracy','presicion','recall','auc'])

for key, classifier in classifiers.items():
    classifier.fit(X_balance, Y_balance)
    y_predict = classifier.predict(testX)

    yproba = classifier.predict_proba(testX)[::,1]
    fpr, tpr, _ = roc_curve(testY, y_predict)
    auc = roc_auc_score(testY, y_predict)
    result_table = result_table.append({'classifiers': key,
                                        'accuracy': accuracy_score(testY, y_predict),
                                        'presicion': precision_score(testY, y_predict,
                                        average='weighted'),
                                        'recall': recall_score(testY, y_predict,
                                        average='weighted'),
                                        # 'f1_score': f1_score(testY, y_predict,
                                        average='weighted'),
                                        # 'fpr': fpr,
                                        # 'tpr': tpr,
                                        'auc': auc}, ignore_index=True)

result_table.set_index('classifiers', inplace=True)
```

Python

classifiers	accuracy	presicion	recall	auc
LogisticRegression	0.801273	0.992646	0.801273	0.591994
DecisionTree	0.842451	0.991859	0.842451	0.517831

- We observe that the accuracy of both Logistic Regression & Decision Tree models are comparable (around 80%)
- But going by the ROC curves, area under curve for Logistic Regression model is better (0.59) compared to Decision Tree model (0.51)
- Hence we conclude that Logistic Regression model would be more appropriate