Credit Card Default Prediction

Machine learning approach

Adarsh Yadav - 22112007

Problem Overview

Banks are facing problem with credit card default so they want to know whether the customer will default next month or not

Solution Approach

We have the data of almost 25000 customers so based on this data we will train our machine learning classification model .

Dataset Overview

Summary:

Dataset of **25,000 credit card users** across 6 months, with attributes like payment history, demographic data, and default status.

Key Columns:

Column Name	Description
CustomerId	Unique ID per customer
Marriage	Marital Status: `1=Married`, `2=Single`, `3=Others`
Sex	Gender: `1=Male`, `0=Female`
Education	`1=Graduate School`, `2=University`, `3=High School`, `4=Others`
Limit_balance	Credit limit (in currency units)
Age	Age of customer

Dataset Overview

Monthly Behavior:

- **PAY_X (X = 0 to 6)**: Payment status
  - `-2`: No credit use
  - `-1`: Full repayment
  - `0`: Minimum/partial payment
  - `1+`: Delayed payments
- **BILL_AMT_X (X = 1 to 6)**: Bill amount
  - `> 0`: Money owed
  - `= 0`: No activity
  - `< 0`: Overpaid (credit balance)
- **PAY_AMT_X (X = 1 to 6)**: Payments made

Engineered Features:

Column Name	Description
Avg_bill_amt	Average bill across 6 months
PAY_TO_BILL_ratio	Total payment / total bill (6 months)

Target:

Column	Description
next_month_default	`1`: Will default next month `0`: Will not default

```
df.shape
```

```
(25121, 27)
```

Dataset Overview

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25247 entries, 0 to 25246
Data columns (total 27 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Customer_ID         25247 non-null  int64
 1   marriage            25247 non-null  int64
 2   sex                 25247 non-null  int64
 3   education           25247 non-null  int64
 4   LIMIT_BAL           25247 non-null  int64
 5   age                 25121 non-null  float64
 6   pay_0               25247 non-null  int64
 7   pay_2               25247 non-null  int64
 8   pay_3               25247 non-null  int64
 9   pay_4               25247 non-null  int64
 10  pay_5               25247 non-null  int64
 11  pay_6               25247 non-null  int64
 12  Bill_amt1           25247 non-null  float64
 13  Bill_amt2           25247 non-null  float64
 14  Bill_amt3           25247 non-null  float64
 15  Bill_amt4           25247 non-null  float64
 16  Bill_amt5           25247 non-null  float64
 17  Bill_amt6           25247 non-null  float64
 18  pay_amt1            25247 non-null  float64
 19  pay_amt2            25247 non-null  float64
 20  pay_amt3            25247 non-null  float64
 21  pay_amt4            25247 non-null  float64
 22  pay_amt5            25247 non-null  float64
 23  pay_amt6            25247 non-null  float64
 24  AVG_Bill_amt        25247 non-null  float64
 25  PAY_TO_BILL_ratio   25247 non-null  float64
 26  next_month_default  25247 non-null  int64
dtypes: float64(15), int64(12)
memory usage: 5.2 MB
```

Exploratory Data Analysis

Age: There are some null values around 126 so removing those rows as they less so it will not affect the result

Removing rows with null values

```
df.isnull().sum()
# Age is having some null values
```

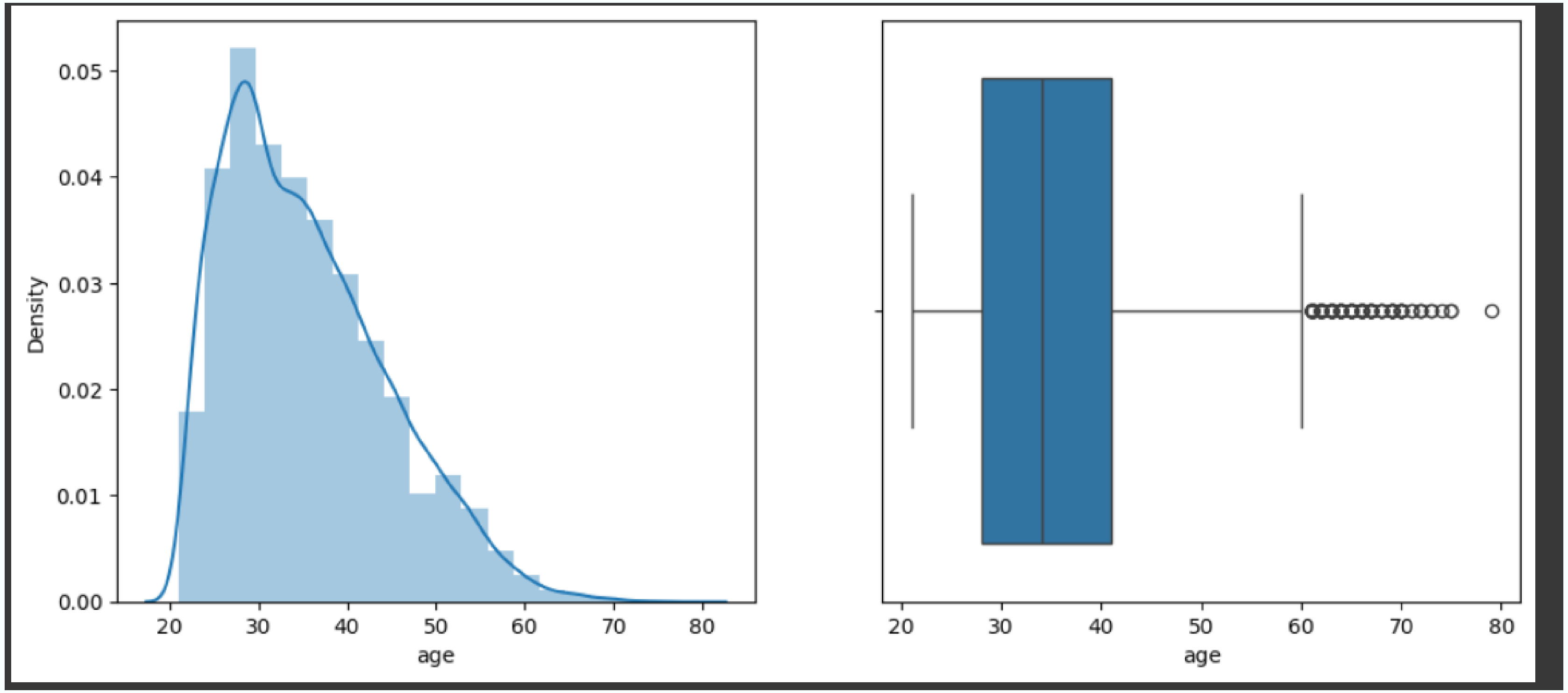
	0
Customer_ID	0
marriage	0
sex	0
education	0
LIMIT_BAL	0
age	126

```
[34] # removing them
     df.dropna(subset=['age'],inplace=True)
```

Replacing Categorical Columns with original value for better understanding

```
df['sex'] = df['sex'].replace({1:'Male',0:'Female'})
df['education']= df['education'].replace({1:'Graduate School', 2:'University', 3:'High School',0:'Others',4:'Others',5:'Others',6:'Others' })
df['marriage'] = df['marriage'].replace({1:'Married',2:'No',3:'Others',0:'Others'})
df.head()
```

Age :-



Observations:

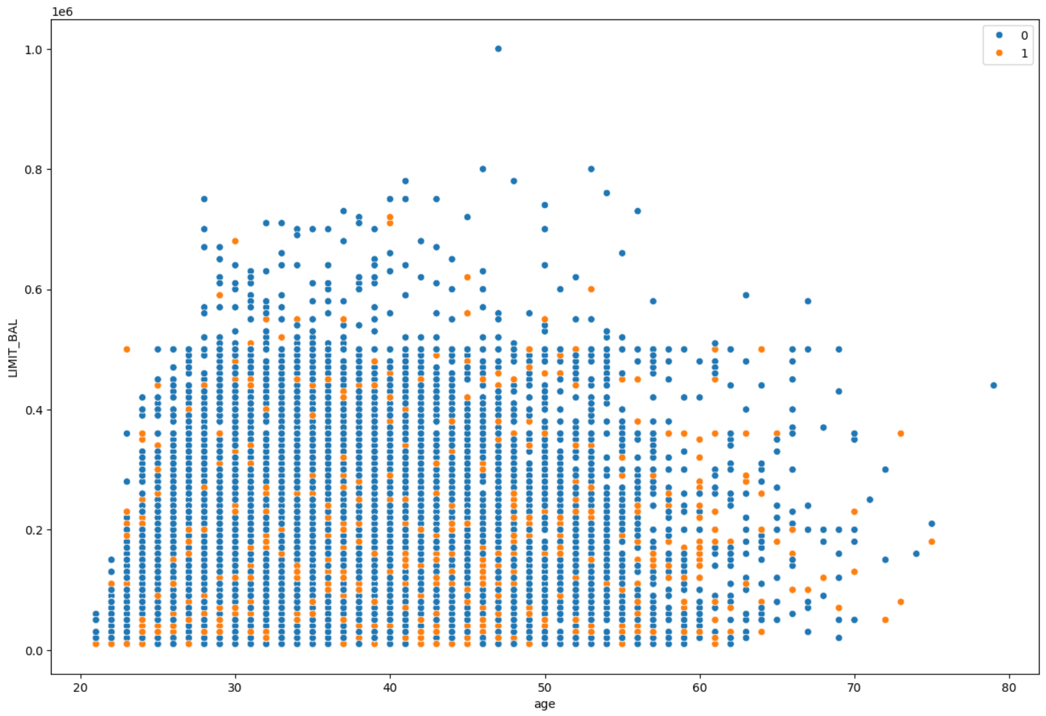
- Age is between 21 to 79
- Mostly are between 25 to 35
- Age data is rightly skewed

```
# Skewness
df['age'].skew()
```

```
np.float64(0.7384976486065989)
```

Age :- Multivariate Analysis

Age with Limit Balance

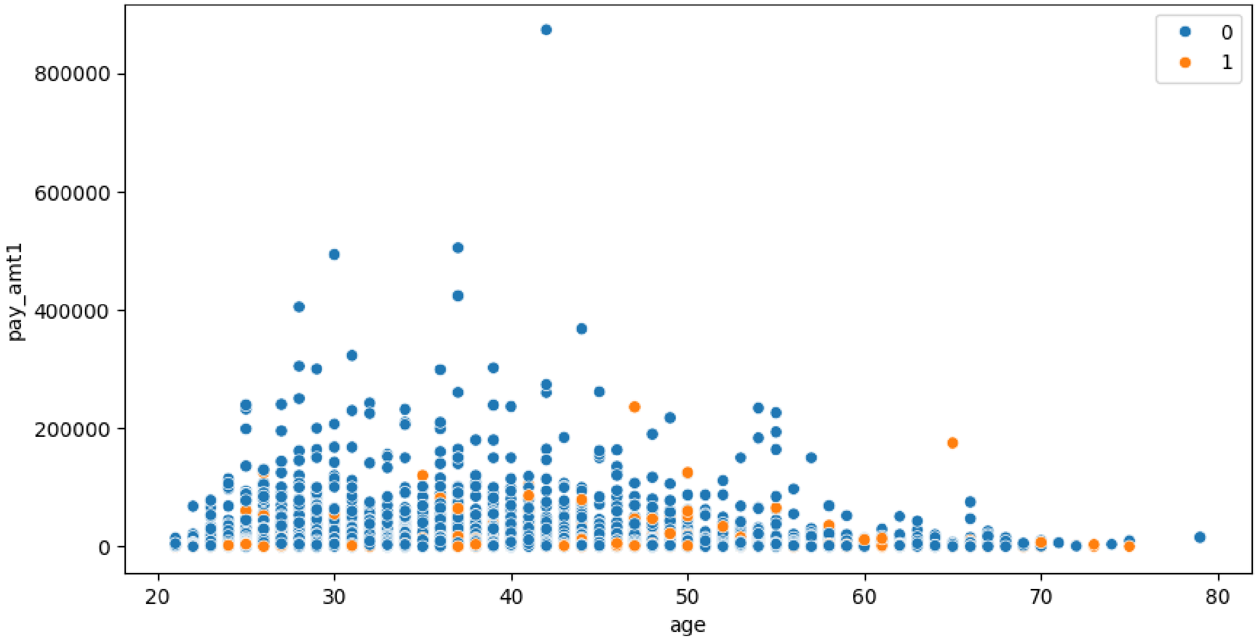


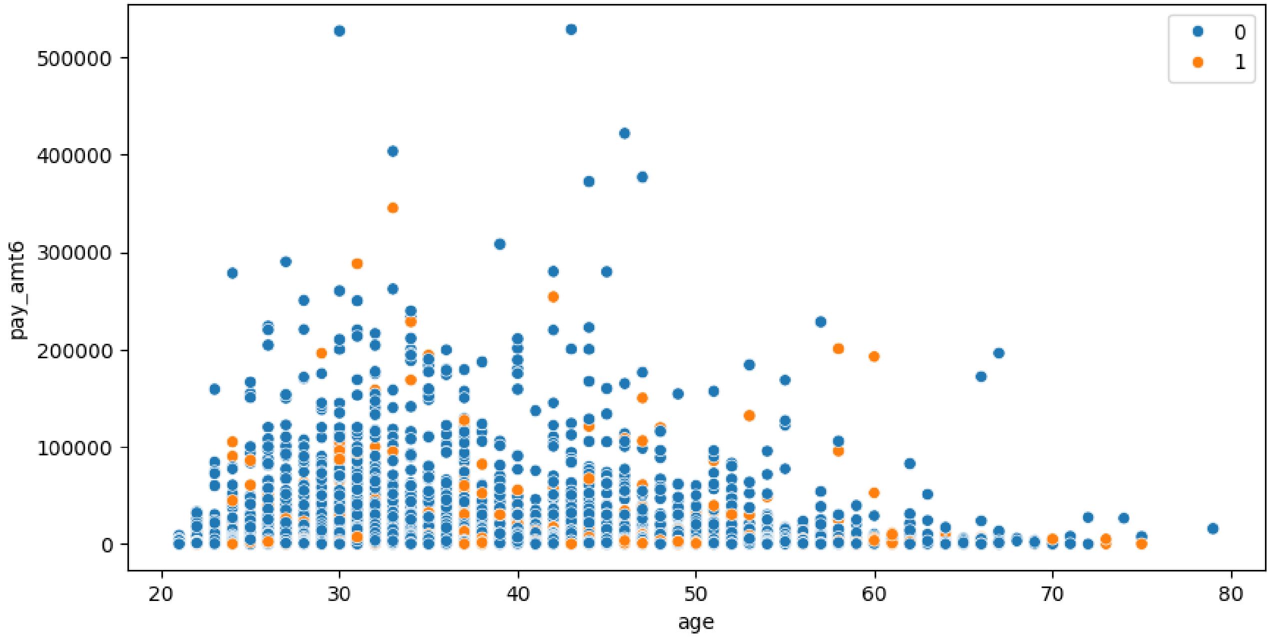
Observations:

- Shows that if age and limit balance are higher chance of defaulting is low

Age :- Multivariate Analysis

Age with Pay amount

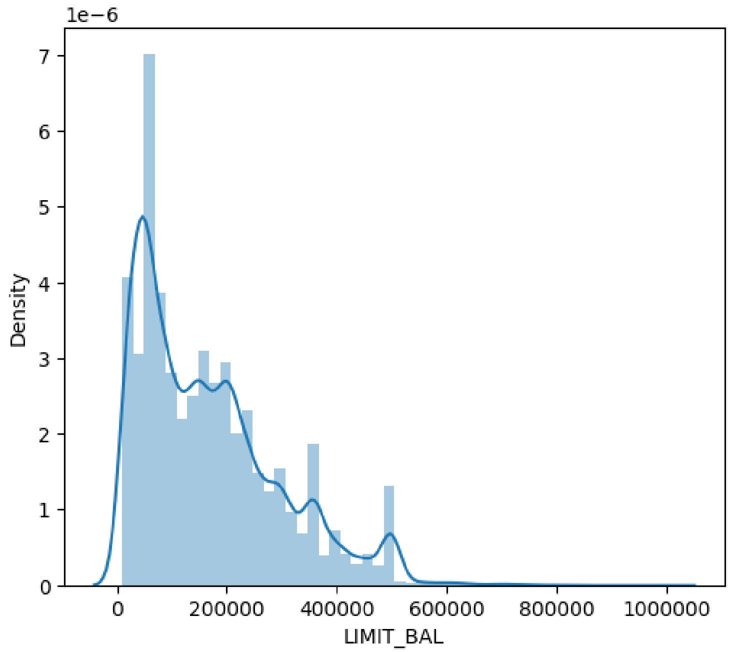


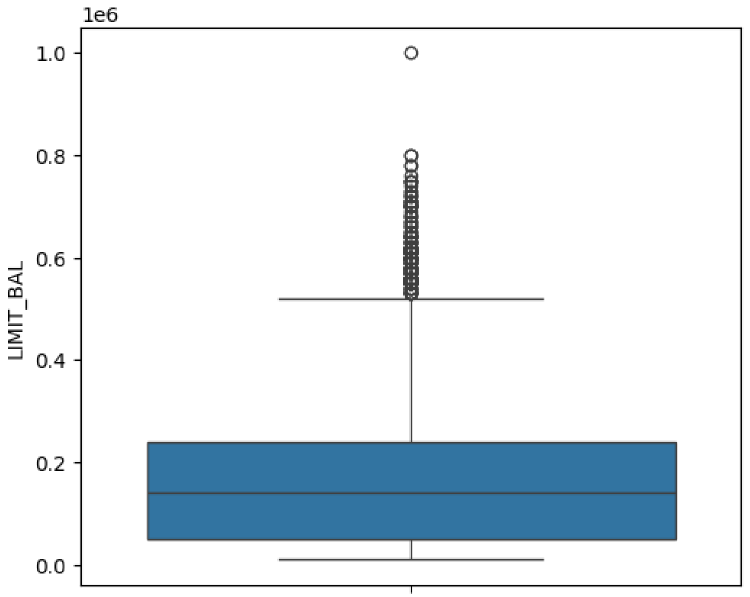


Observations:

- Customer with high payment amount are mostly between 25-50 ,
- Below 35 - Defaulter are decresing as we move from month1 to month6

Limit Balance :-






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

	LIMIT_BAL
count	25121.000000
mean	168358.823295
std	129866.750911
min	10000.000000
25%	50000.000000
50%	140000.000000
75%	240000.000000
max	1000000.000000

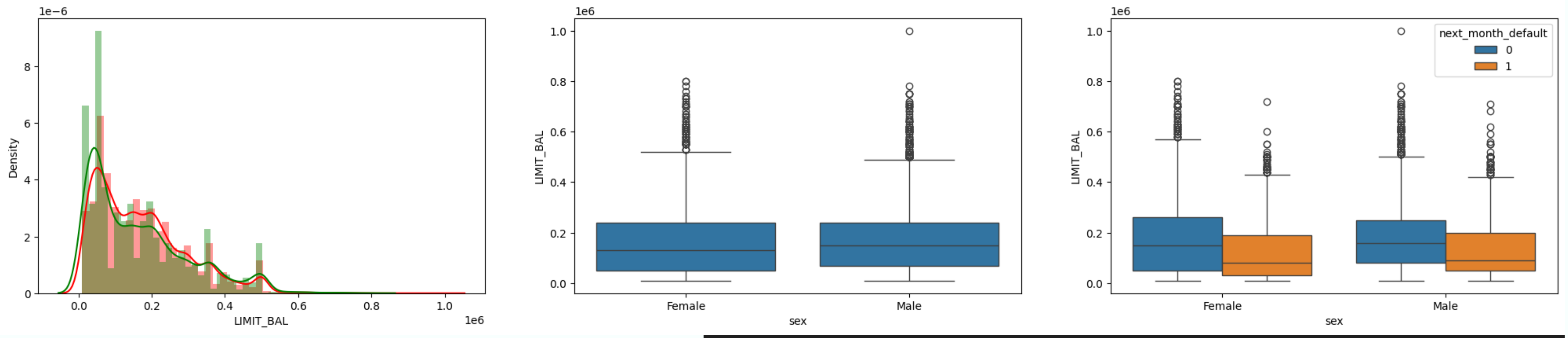
dtype: float64

Observations:

- Most Customer have limit balance between 0-175000
- Default rate is higher below 175000
- Rightly skewed

Limit Balance :-Multivariate

Limit Balance with Sex




```
df.groupby(['sex'])['LIMIT_BAL'].describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9,.99])
```

sex	count	mean	std	min	10%	20%	30%	40%	50%	60%	70%	80%	90%	99%	max
Female	9930.0	164483.987915	136518.286753	10000.0	20000.0	50000.0	50000.0	90000.0	130000.0	170000.0	210000.0	280000.0	360000.0	500000.0	800000.0
Male	15191.0	170891.712198	125267.964752	10000.0	30000.0	50000.0	80000.0	110000.0	150000.0	180000.0	220000.0	270000.0	360000.0	500000.0	1000000.0

```
[52] df.groupby(['sex','next_month_default'])['LIMIT_BAL'].describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9,.99])
```

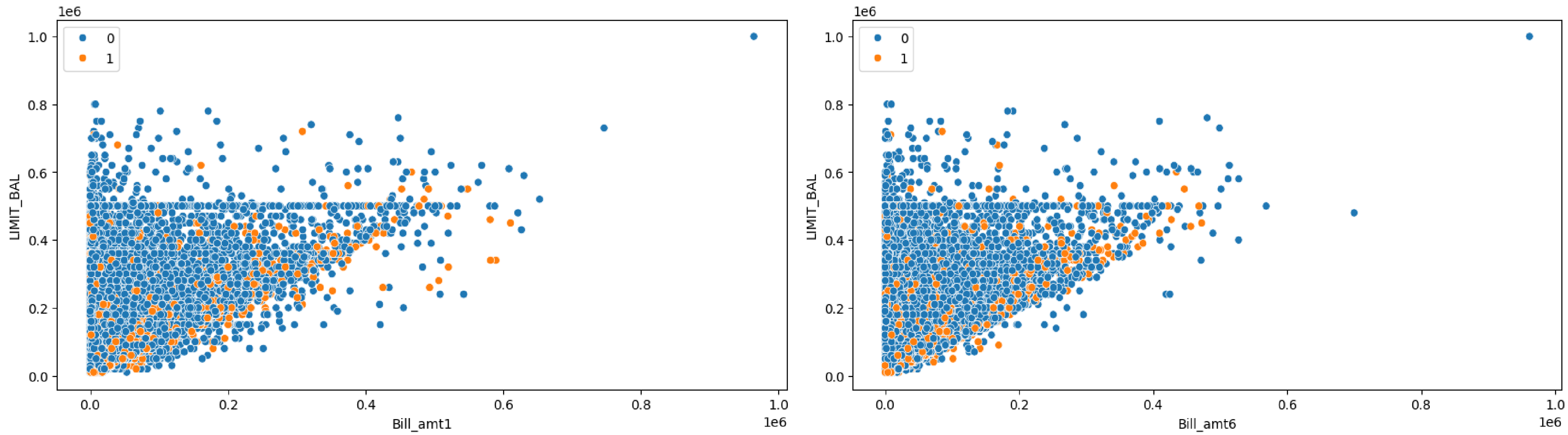
sex	next_month_default	count	mean	std	min	10%	20%	30%	40%	50%	60%	70%	80%	90%	99%	
Female	0	7858.0	174846.780351	139523.143559	10000.0	30000.0	50000.0	60000.0	100000.0	150000.0	180000.0	230000.0	290000.0	380000.0	500000.0	
	1	2072.0	125183.397683	116390.114564	10000.0	20000.0	30000.0	50000.0	50000.0	80000.0	120000.0	160000.0	200000.0	300000.0	500000.0	
Male	0	12479.0	179229.585704	126337.685866	10000.0	40000.0	60000.0	90000.0	120000.0	160000.0	200000.0	230000.0	280000.0	360000.0	500000.0	1
	1	2712.0	132525.811209	112545.737566	10000.0	30000.0	40000.0	50000.0	70000.0	90000.0	130000.0	170000.0	220000.0	300000.0	500000.0	

Observations:

- Mean Limit is almost same for both gender
- When seperate on the basis of default , mean limits differs significantly for defaulters and non defaulters

Limit Balance :-

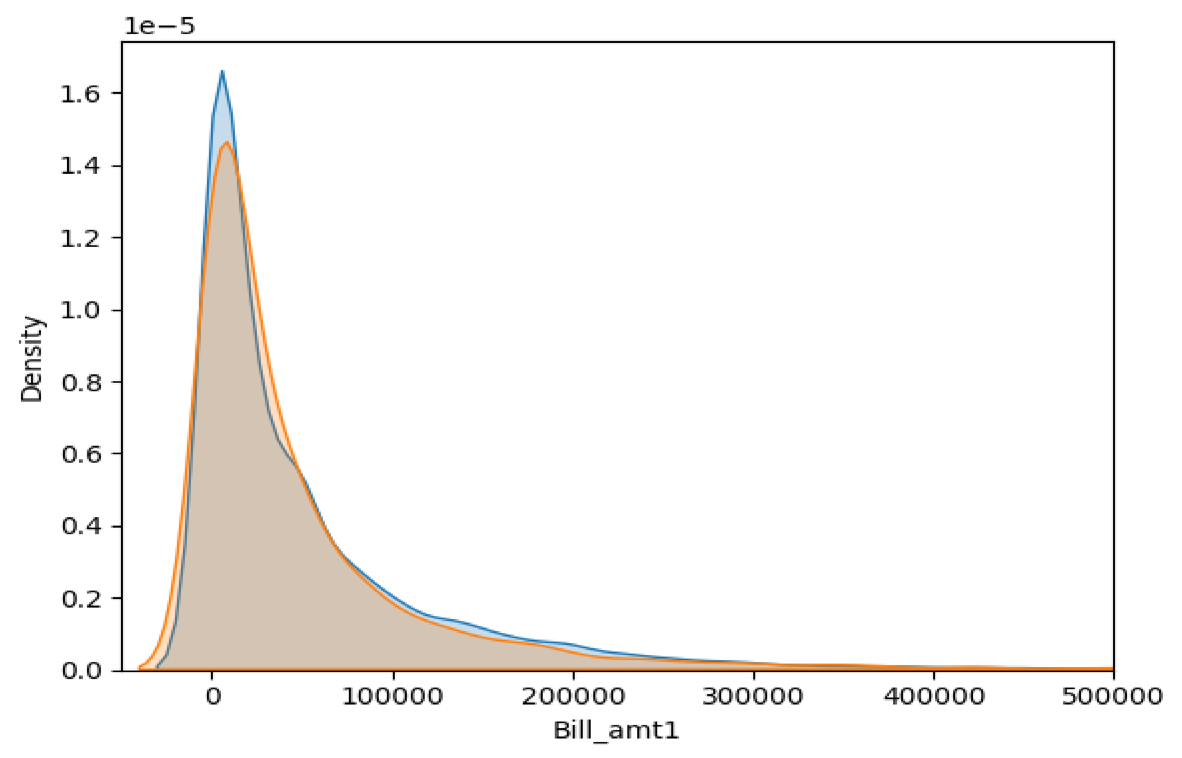
Limit Balance with Bill amount

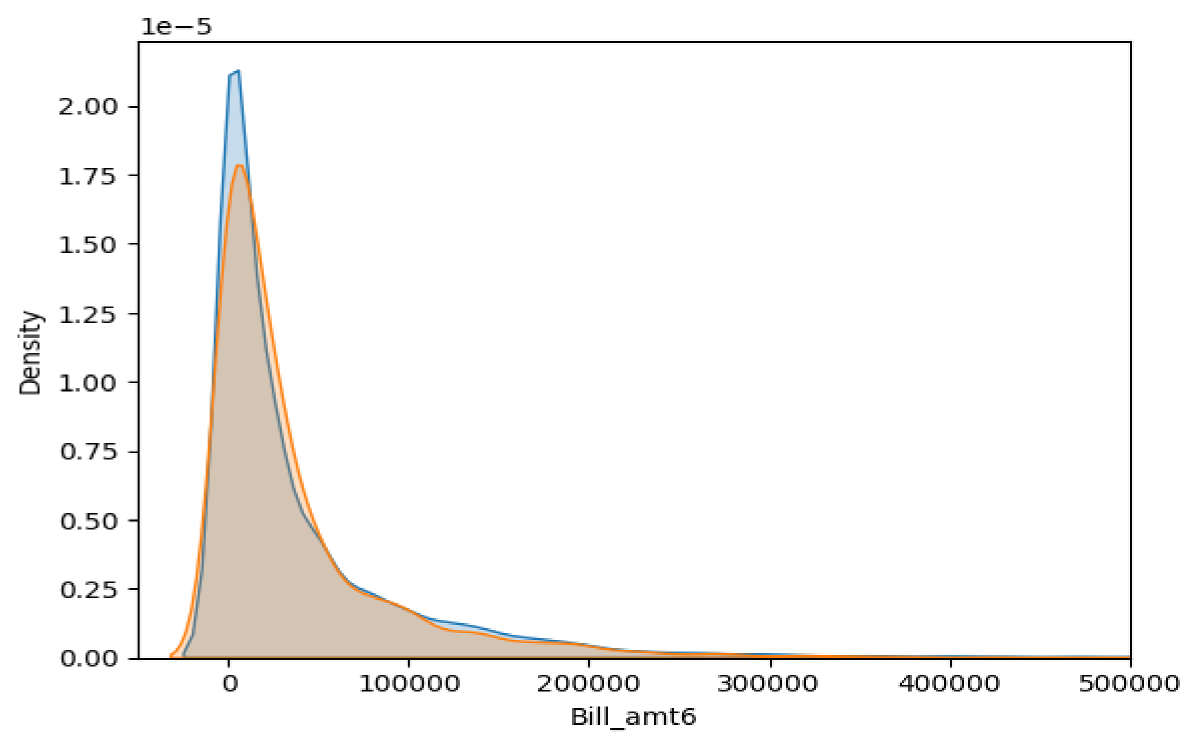


Observations:

- Number of customer drastically decreases after 500000
- Limit balance increase and bill amount decreases

Bill Amount:-



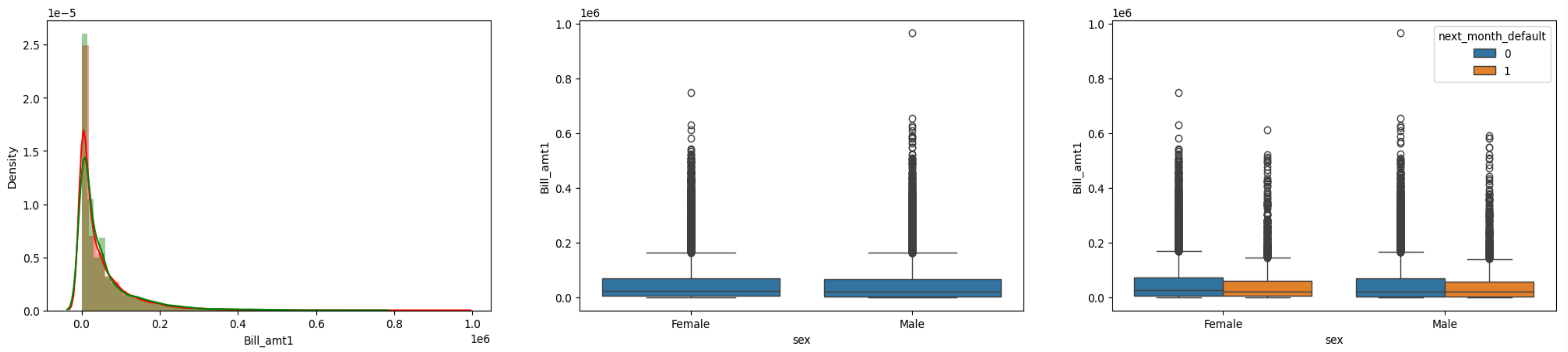


Observations:

- Rightly skewed data
- There is some relation from month 1 to month 6

Bill Amount:-Multivaraite

Bill amount with sex and default



Observations:

- From the graphs we might come to an assumption that bill amounts has a significance effect on gender.
- Mean amouns between male and female shows some difference

```
for col in bill_amt_cols:
  print(col )
  print(df.groupby(['sex'])[col].describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9,.99]))
```

```
Bill_amt1
          count          mean           std  min      10%       20%       30%  \
sex
Female   9930.0  54177.635872  76854.252962  0.0  389.798  2501.386  7722.329
Male    15191.0  49266.369535  70878.224942  0.0  139.860  1603.630  5024.350
```

Bill Amount:-Multivariate

Bill amount with education

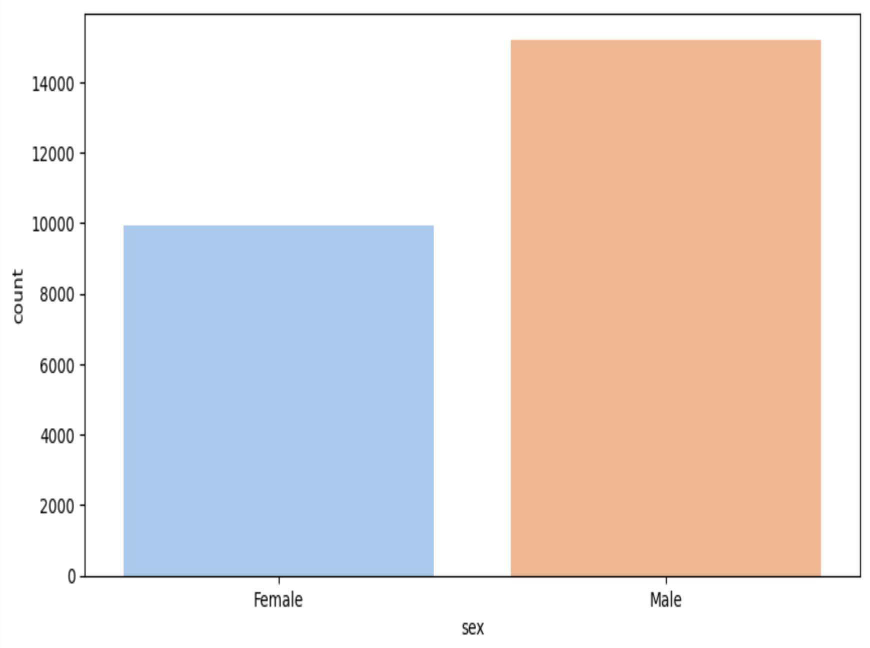
```
for col in bill_amt_cols:
  print(col )
  print(df.groupby(['education'])[col].describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9,.99]))
```

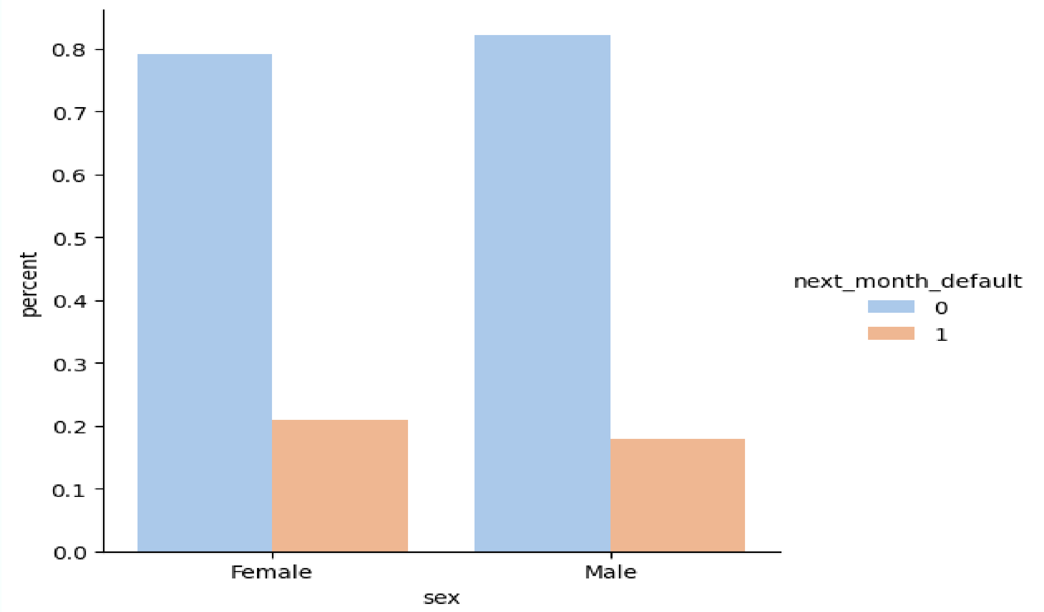
```
Bill_amt1
                   count          mean           std  min      10%       20%  \
education
Graduate School   8944.0  48717.536096  78350.806939  0.0    0.590   795.058
High School       4096.0  47455.568801  64983.865689  0.0  389.730  2565.270
Others             424.0  70014.542524  88573.547889  0.0  319.795  2955.308
University       11657.0  53752.725842  71297.852296  0.0  491.218  3533.254

                      30%        40%       50%        60%        70%  \
education
Graduate School  2773.729   6457.104  14217.43  26854.704  47926.980
High School      7824.655  15968.880  24930.56  38243.410  50392.780
Others           8351.540  17872.266  30639.68  53840.382  88624.442
University       9891.824  18156.004  27527.00  44303.040  58206.748

                       80%         90%          99%        max
education
Graduate School   81251.568  149612.218  376876.4477  964511.16
High School       75268.330  126646.550  285091.1735  746813.18
Others           136150.440  192473.288  372203.4185  626647.12
University        86710.552  140750.082  339377.2104  610722.35
```

Observations:

- Median bill amounts of each educational category, displays only a significantly small difference as it passes through each month
- When each category is subcategorized based on 'default' value, we see that the median bill amount for 'Others' category shows some difference for defaulters and non defaulters in every month.

Sex:-






```
[61] df['sex'].value_counts()
```

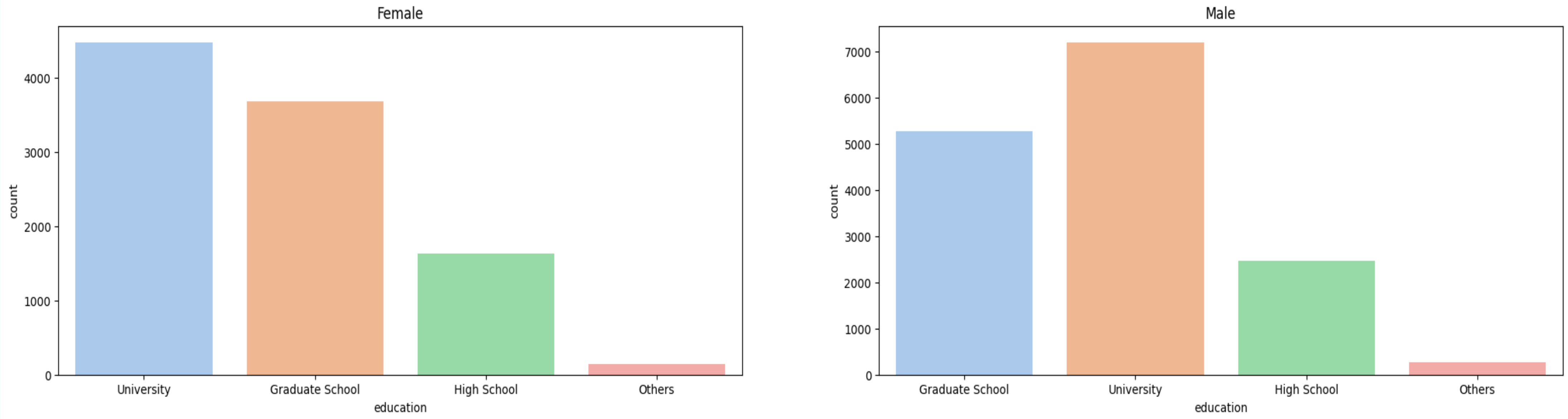
sex	count
Male	15191
Female	9930

Observations:

- More Male customers than female
- More percentage of female defaulters than male.

Sex:- Multivariate

Sex with Education

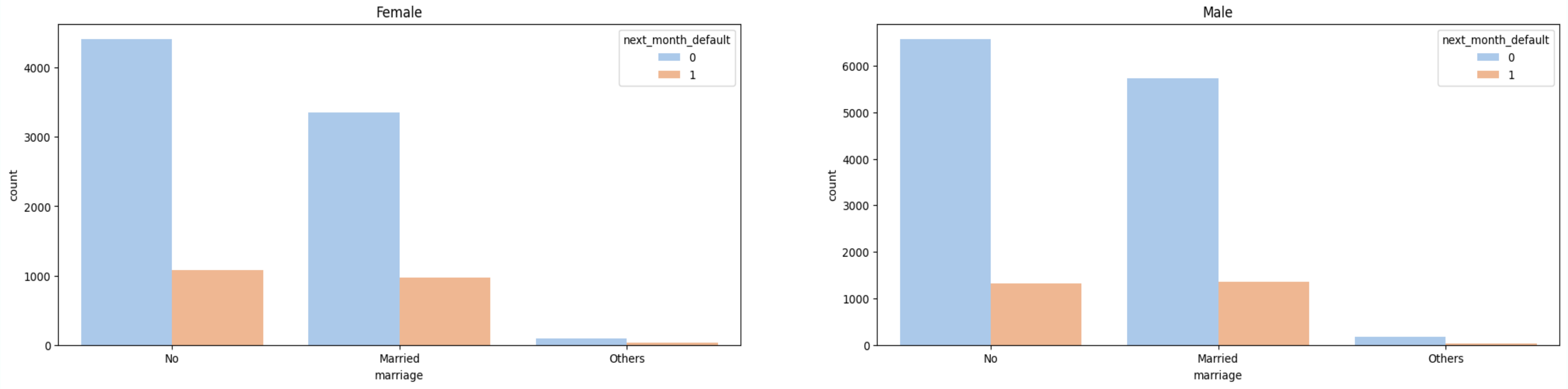


Observations:

- Among Both gender mostly are university educated

Sex:- Multivariate

Sex with Marriage

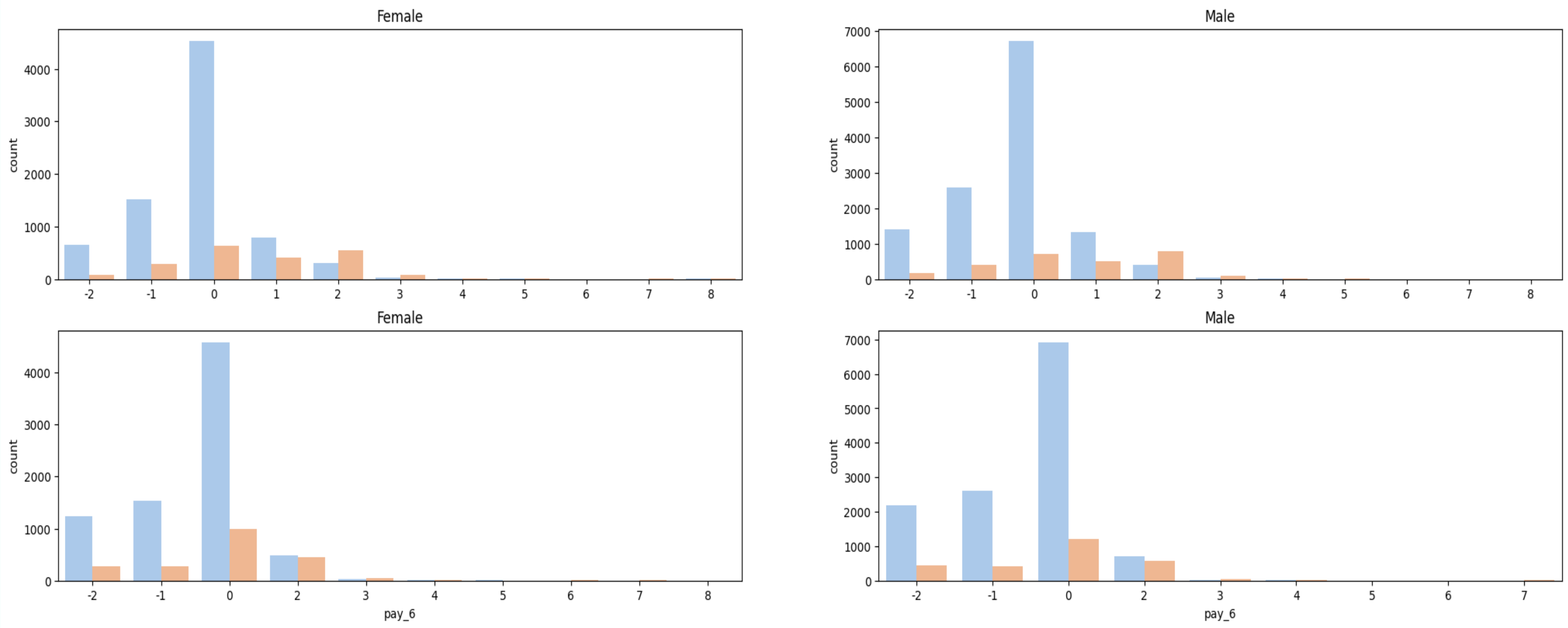


Observations:

- Both married and unmarried almost have same amount of defaulters

Sex:- Multivariate

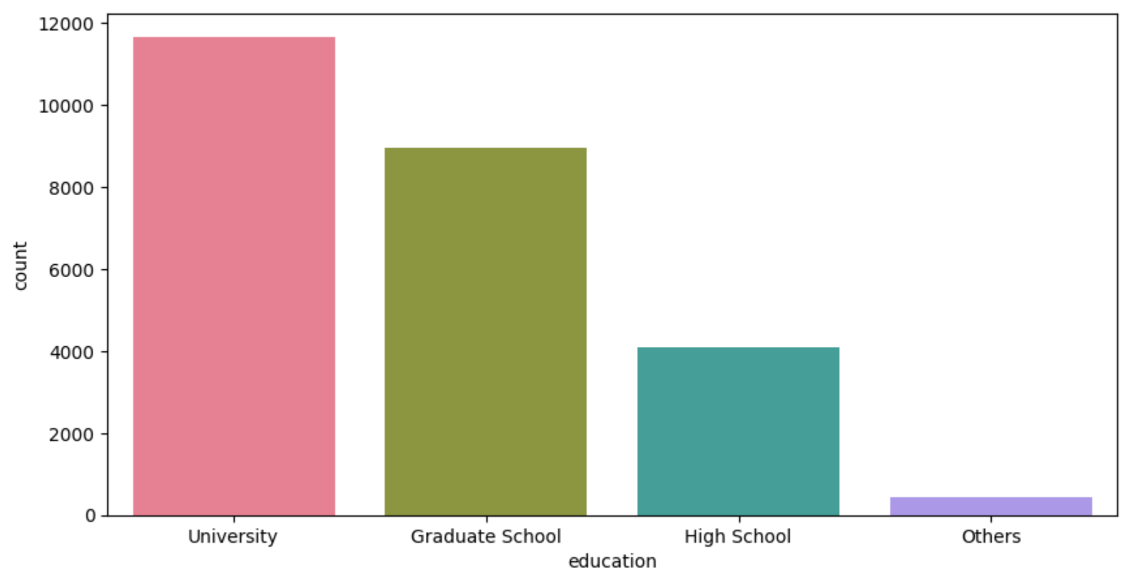
Sex with Repayment

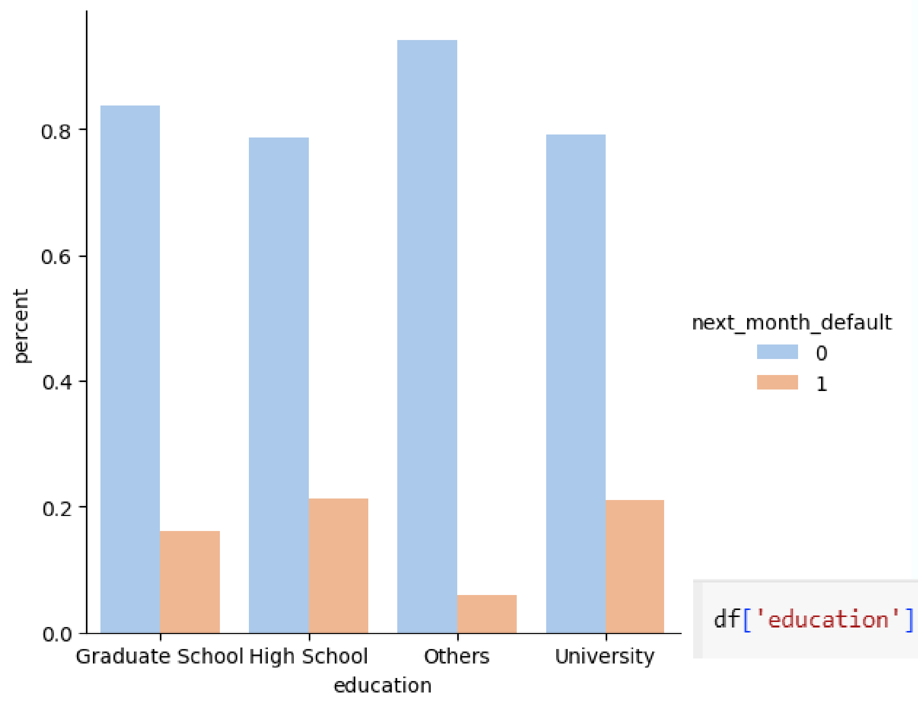


Observations:

- the number of defaulters have seems to be almost on same level for the first 5 months, but during the last month has shown a small dip.

Education:-Univariate






```
df['education'].value_counts()
```

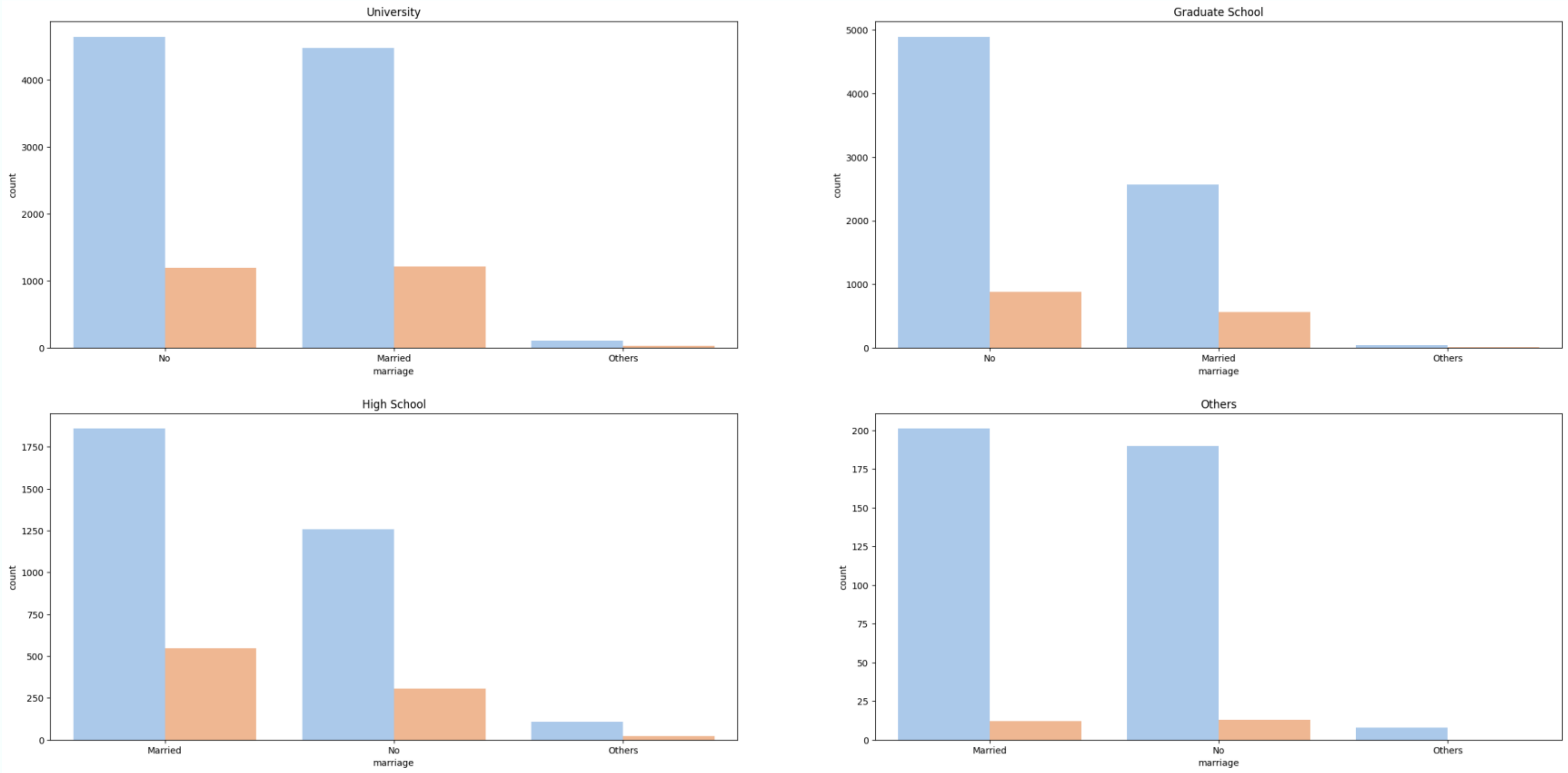
education	count
University	11657
Graduate School	8944
High School	4096
Others	424

Observations:

- University students are the majority of customers
- Majority of defaulter are from University and High School

Education:-Multivariate

Education with Marriage

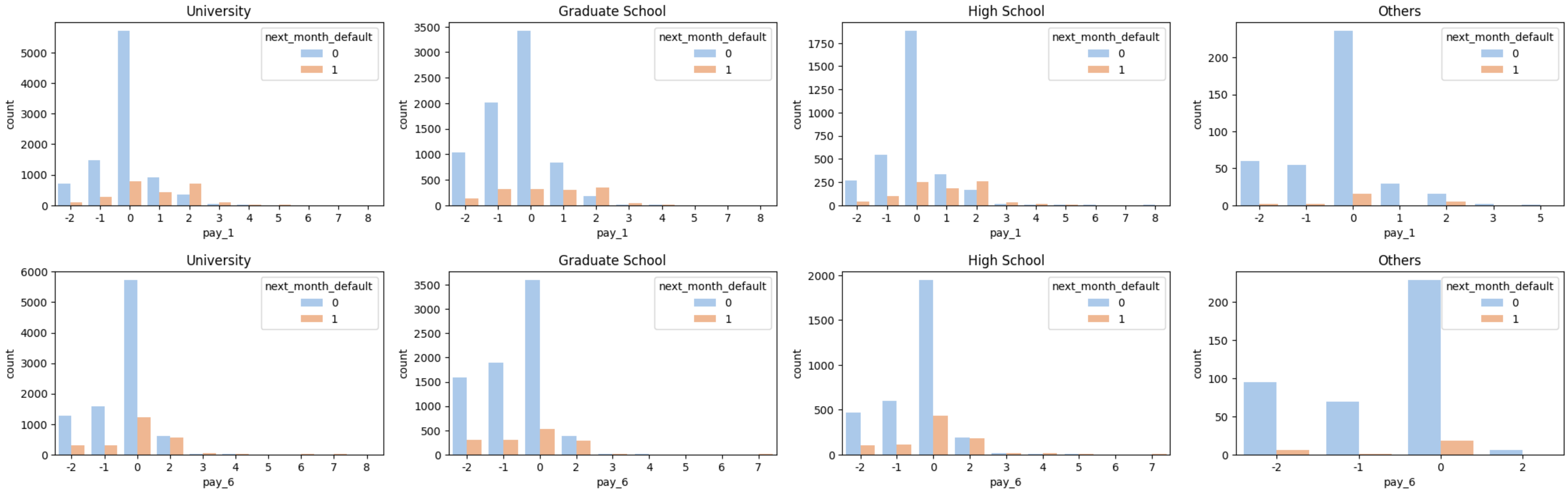


Observations:

- Proportion of defaulter to Non defaulter is almost same in all education categories

Education:-Multivariate

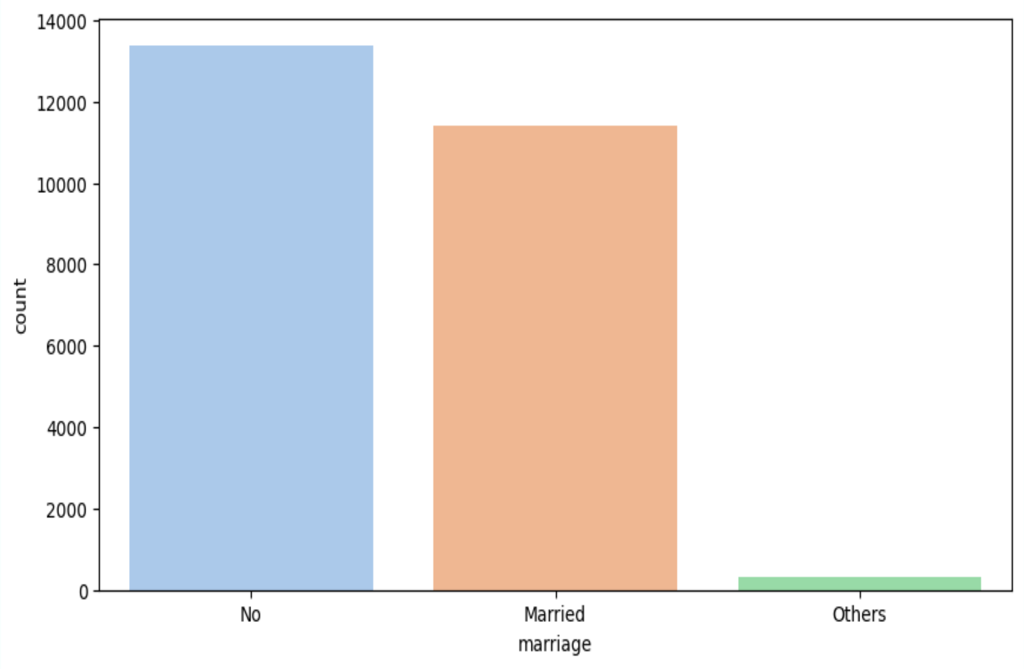
Education with Repayment status

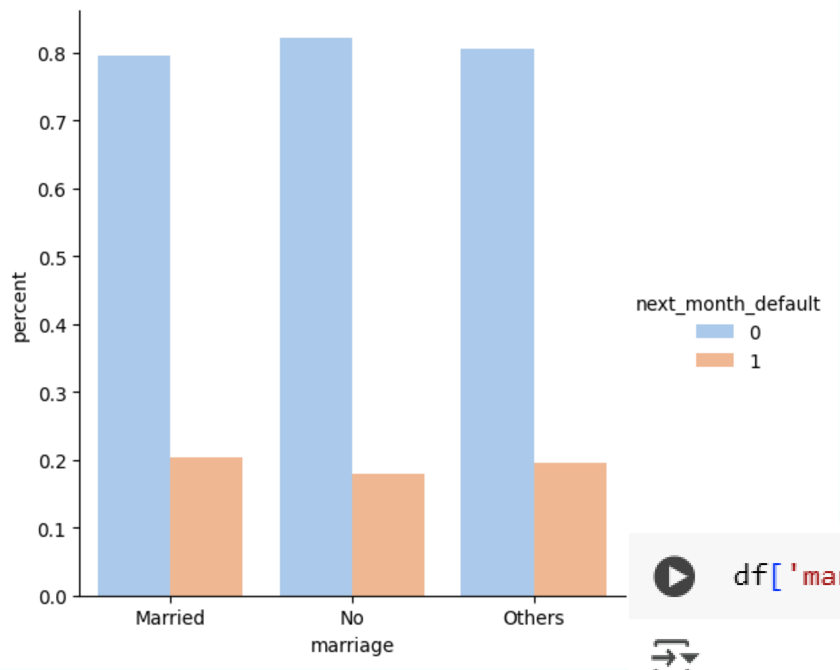


Observations:

- We can see that, for every education category when we move from Month1 to Month6 , there seems to be a sudden increase in the number of customer with 1 month repayment delay in the month 6, when all the other months, there were almost none.

Marriage:-Univariate






```
df['marriage'].value_counts()
```

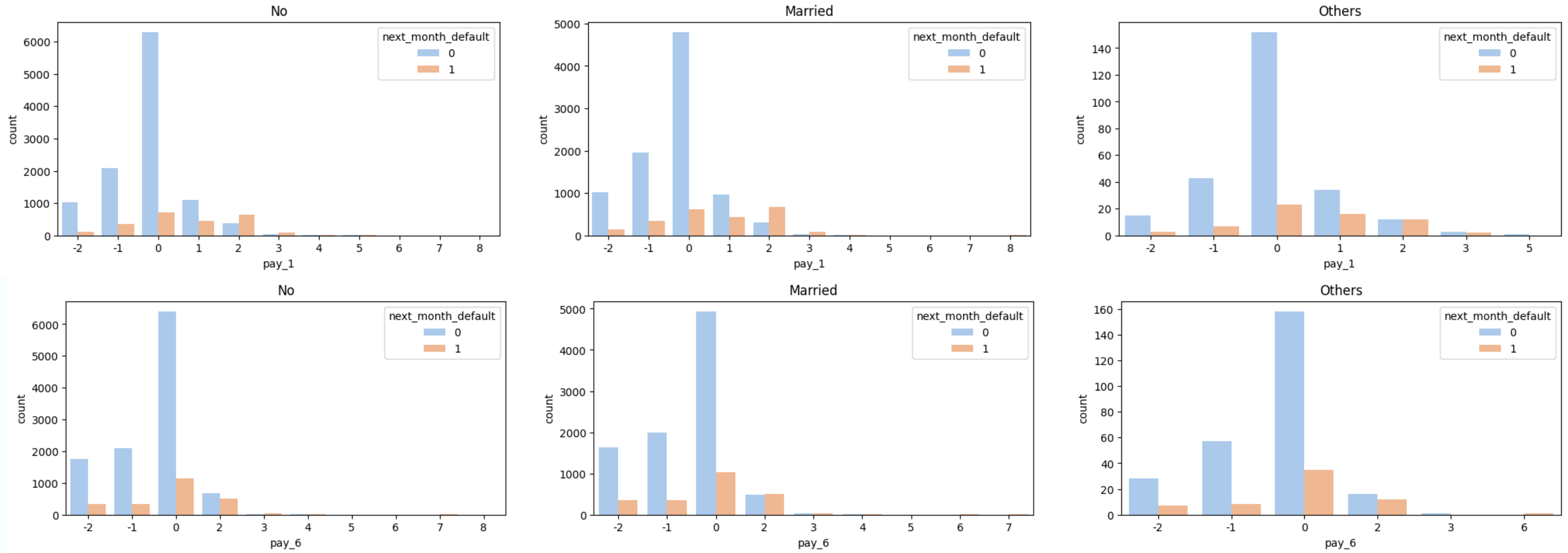
marriage	count
No	13374
Married	11424
Others	323

Observations:

- Unmarried people are more in count
- Default proportion is almost same in all categories

Marriage:-Multivariate

Marriage with repayment amount



Observations:

- Trend is almost same above categories with small variation

Feature Engineering:-

Removing Null values

Removing Null value

```
[3] ### Dropping age with null values as they are only 126 and dropping them will not effect data
    df.dropna(subset=['age'],inplace=True)
```

Feature Engineering:-

Handling Class Imbalance using SMOTE-

Synthetic Minority Oversampling Technique is a machine learning technique used to address the problem of imbalanced datasets. It works by **creating synthetic data** points for the minority class, effectively increasing its representation in the dataset and helping models learn to classify it more accurately.

```
[4]  smote = SMOTE()
```

```
[5]  X = df.iloc[:,:-1]
     y = df['next_month_default']
```

```
[6]  x_smote, y_smote = smote.fit_resample(X, y)
```

```
[7]  df_final = pd.DataFrame(x_smote, columns=df.columns[:-1])
     df_final['next_month_default'] = y_smote

     df_final.head()
```

Feature Engineering:-

Changing Column Names :

Replacing Months with original name

+ Code + Text

```
[13] df_final.rename(columns= {'pay_6':'Pay_January','pay_5':'Pay_February','pay_4':'Pay_March','pay_3':'Pay_
     df_final.rename(columns= {'pay_amt6':'Pay_amt_January','pay_amt5':'Pay_amt_February','pay_amt4':'Pay_am
     df_final.rename(columns= {'Bill_amt6':'Bill_amt_January','Bill_amt5':'Bill_amt_February','Bill_amt4':'Bi

     df2.rename(columns= {'pay_6':'Pay_January','pay_5':'Pay_February','pay_4':'Pay_March','pay_3':'Pay_April
     df2.rename(columns= {'pay_amt6':'Pay_amt_January','pay_amt5':'Pay_amt_February','pay_amt4':'Pay_amt_Marc
     df2.rename(columns= {'Bill_amt6':'Bill_amt_January','Bill_amt5':'Bill_amt_February','Bill_amt4':'Bill_am
```

Feature Engineering:-

Encoding Categorical Data :

- **One Hot Encoding** is a ***method for converting categorical variables into a binary format.*** It creates new columns for each category where **1** means the category is present and **0** means it is not. The primary purpose of One Hot Encoding is to ensure that categorical data can be effectively used in machine learning models.
- There are three columns named sex , education and marriage we applied OHE on then

```
[16] df_final = pd.get_dummies(df_final, columns=['marriage','education','sex'])
     df2 = pd.get_dummies(df2, columns=['marriage','education','sex'])
```

Feature Engineering:-

Train Test Split

- Without adding any additional features lets check model performance

Train Test Split

+ Code + Text

```
X= df_final.drop(columns='next_month_default',axis=1)
y= df_final['next_month_default']


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

X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

Feature Engineering:-

Feature Selection :

Feature Selection

```
[23] rf.feature_importances_

array([0.03731503, 0.04014882, 0.04086808, 0.07932968, 0.04745844,
       0.03486227, 0.02722628, 0.02083799, 0.01582563, 0.04241196,
       0.03739891, 0.03554222, 0.03456232, 0.03391281, 0.03449143,
       0.0509076 , 0.04532754, 0.04139054, 0.03973833, 0.0388632 ,
       0.04014879, 0.03938345, 0.04103   , 0.0225007 , 0.01804734,
       0.00073806, 0.00840582, 0.01165492, 0.0010338 , 0.0053993 ,
       0.01489638, 0.01834236])
```

```
[24] feature_scores = pd.Series(rf.feature_importances_, index=df_final.drop(columns='next_month_default',axi
     feature_scores
```

Model With Top important features

```
df_new = df_final[feature_scores.index[:20]]
df2_new = df2[feature_scores.index[:20]]
df_new['default'] = df_final['next_month_default']
X = df_new.drop('default', axis=1)
y = df_new['default']
```

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)

print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
```

```
Accuracy:  0.8735095267363245
Precision:  0.9025720966484801
Recall:  0.8415697674418605
F1 Score:  0.871004136893569
```

- Selecting only important features for training model
- Scores are improved from the previous ones.

Feature Engineering:-

Creating new features :

1-Bill amount average

```
[33] df_final['Bill_amt_avg']=(df_final['Bill_amt_January']+df_final['Bill_amt_February']+df_final['Bill_amt_
     df2['Bill_amt_avg']=(df2['Bill_amt_January']+df2['Bill_amt_February']+df2['Bill_amt_March']+df2['Bill_ar
```

2-Bill Pay Value

```
[34] df_final['Bill_pay_value'] = ((df_final['Pay_amt_January'] - df_final['Bill_amt_January']) + (df_final['
     df2['Bill_pay_value']=((df2['Pay_amt_January'] - df2['Bill_amt_January']) + (df2['Pay_amt_February'] - c
```

Model Performance

Gradient Boosting

- Gradient Boosting doest not give satisfacting results
- Lets try another model

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)
f2 = fbeta_score(y_test,test_pred,beta=2)
roc_auc = roc_auc_score(y_test,test_pred)

print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
print('F2 Score', f2)
print('roc_auc',roc_auc)
```

```
Accuracy:  0.8183159188690842
Precision:  0.8455920709441836
Recall:  0.7853682170542635
F1 Score:  0.8143682491836222
F2 Score 0.7967167993708837
roc_auc 0.8188133822980327
```

Model Performance

CatBoost

- As we can scores are improved compared to previous model
- Lets improve it more

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)
f2 = fbeta_score(y_test,test_pred,beta=2)
roc_auc = roc_auc_score(y_test,test_pred)

print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
print('F2 Score', f2)
print('roc_auc',roc_auc)
```

```
Accuracy:  0.8484326982175784
Precision:  0.8852808091562416
Recall:  0.8057170542635659
F1 Score:  0.8436271401395053
F2 Score 0.8204647491242785
roc_auc 0.8490776436778273
```

Model Performance

Applying RandomForest :

- Applying rf and evaluating performance metrics
- Results are quite good and improved
- Let try some more models

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)
f2 = fbeta_score(y_test,test_pred,beta=2)
roc_auc = roc_auc_score(y_test,test_pred)

print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
print('F2 Score', f2)
print('roc_auc',roc_auc)
```

```
Accuracy:  0.8695759065765212
Precision:  0.8865641542727501
Recall:  0.8519864341085271
F1 Score:  0.8689314391599753
F2 Score 0.8586845060793984
roc_auc 0.8698414825895767
```

Model Performance

Ensemble methods - Combining random forest and GradientBoosting

- As we can see results are degraded from the previous model .
- Lets try another methods

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)
f2 = fbeta_score(y_test,test_pred,beta=2)
roc_auc = roc_auc_score(y_test,test_pred)

print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
print('F2 Score', f2)
print('roc_auc',roc_auc)
```

```
Accuracy:  0.8544560540872772
Precision:  0.8758937691521961
Recall:  0.8309108527131783
F1 Score:  0.8528095474888115
F2 Score 0.8395339729782652
roc_auc 0.8548115531347273
```

Model Performance

Using LGBM

- Performance metrics are good
- F2 score is also increased

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)
f2 = fbeta_score(y_test,test_pred,beta=2)
roc_auc = roc_auc_score(y_test,test_pred)
print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
print('F2 Score', f2)
print('roc_auc',roc_auc)
```

```
Accuracy:  0.8867854947756607
Precision:  0.9325600215807931
Recall:  0.8374515503875969
F1 Score:  0.8824505424377792
F2 Score 0.8548889658242247
roc_auc 0.8875303671578614
```

Model Performance

Using XGBoost:

- **XGBoost** is a gradient boosting algorithm, meaning it iteratively builds a model by adding new trees that correct the errors of the previous ones.
- From this we get the highest F2 score
- Selecting this as our final model for evaluation

```
[58] f2_scorer = make_scorer(fbeta_score, beta=2)
     xgb_model = xgb.XGBClassifier()
     optimization_dict = {
         'max_depth': [2,4,6],
         'n_estimators': [50, 100, 200]
     }
     model = GridSearchCV(xgb_model, optimization_dict, scoring='accuracy', verbose=1)
     model.fit(X_train, y_train)
```

```
acc = accuracy_score(y_test,test_pred)
prec = precision_score(y_test,test_pred)
rec = recall_score(y_test,test_pred)
f1 = f1_score(y_test,test_pred)
f2 = fbeta_score(y_test,test_pred,beta=2)
roc_auc = roc_auc_score(y_test,test_pred)

print('Accuracy: ', acc)
print('Precision: ', prec)
print('Recall: ', rec)
print('F1 Score: ', f1)
print('F2 Score', f2)
print('roc_auc',roc_auc)
```

```
Accuracy:  0.8813767670559312
Precision:  0.9117417339234575
Recall:  0.8483527131782945
F1 Score:  0.8789057598192997
F2 Score 0.8603154326143566
roc_auc 0.8818753832924164
```

Model Selection

- Apply many models and comparing them on the basis of performance
- **Selecting Xgboost as out final model for output validation as it has highest f2 score**

	Accuracy	Precision	Recall	F1 Score	F2 Score	ROC AUC
Gradient Boosting	0.818316	0.845592	0.785368	0.814368	0.796717	0.818813
RandomForest	0.869576	0.886564	0.851986	0.868931	0.858685	0.869841
Ensemble (GB + RF)	0.854456	0.875894	0.830911	0.852810	0.839534	0.854812
CatBoost	0.848433	0.885281	0.805717	0.843627	0.820465	0.849078
XGBoost	0.881377	0.911742	0.848353	0.878906	0.860315	0.881875
LightGBM	0.886785	0.932560	0.837452	0.882451	0.854889	0.887530

Thank You