Data Mining Project
Credit Card Cancellation Classification

By

6205046	Thitiwat	Worapraphinchai
6205071	Akepawin	Pornserikul
6205263	Metee	Baopimpa

Present to

Asst. Prof. Pannapa Changpetch

This project associates with SCMA 447 Data Mining

Department of Mathematics, Faculty of Science, Mahidol University

Table of Contents

1. Introduction

Credit Card is a solution of payment when you do not have money in hand and in need of spending now. Providers of credit cards can set conditions and benefits for using their services and make profit. However, customers can choose a provider that gives conditions and benefits suitable for their lifestyle. If the current one does not serve their purposes, then they can choose to stop the service and sign with a more suitable one. This leads to one of the most important classification jobs in most businesses to keep their customers satisfied and use their products, called churn prediction.

In this task, a manager of a bank wants to classify customers who are likely to cancel their credit card subscription. With the prediction they can use this information to provide those people with better services and change customers' decision to continue their credit card subscription. The data that the bank has collected contains many features, such as their salary, marital status, income range, card category, etc. as they can use to keep track of every customers' behavior.

The downside of this data is having an imbalanced dataset. The bank has only 16.1% from over 10,000 customers that would cancel the subscription. This is a challenge we need to solve for the bank manager with our data mining technique. We will first visualize the data we obtain to get insight. Then, we will build machine learning models to learn from the dataset and try to predict who would cancel a credit card subscription. The models we will build in this report are Classification Tree (Decision Tree), Naïve Bayes Classifier, and k Nearest Neighbor.

2. Dataset overview

- Raw data contains 10127 instance and 20 features
- Source: https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers
- Features of dataset contains
    1. Attrition_Flag - Internal event(if customer closes account then Attrited Customer else Existing Customer)
    2. Customer_Age - Customer's age in year(Quantitative)
    3. Gender - Customer's gender(M is male, F is female)
    4. Dependent_count - Number of dependents(Categorical)
    5. Education_Level - Customer's educational qualification(high school, college, graduate, etc)
    6. Marital_Status - Customer's marital status(Categorical)
    7. Income_Category - Customer's income(Categorical)
    8. Card_Category - The type of credit card that customer holds(Categorical)
    9. Months_on_book - How long customer have been part of the service(Quantitative)
    10. Total_Relationship_Count - Number of card customer holds(Categorical)
    11. Months_Inactive_12_mon - Number of inactive months in the last 12 months(Categorical)
    12. Contacts_Count_12_mon - Number of contacts in the last 12 months(Categorical)
    13. Credit_Limit - Credit limit of the card that customer holds(Quantitative)
    14. Total_Revolving_Bal - Total revolving balance on the credit card(Quantitative)
    15. Avg_Open_To_Buy - Open to buy credit card which is average in the last 12 months(Quantitative)
    16. Total_Amt_Chng_Q4_Q1 - Change in transaction amount from Q4 to Q1(Quantitative)
    17. Total_Trans_Amt - family educational support in the last 12 months(Quantitative)
    18. Total_Trans_Ct - Total transaction count in the last 12 months (Quantitative)
    19. Total_Ct_Chng_Q4_Q1 - Change in transaction count from Q4 to Q1 (Quantitative)
    20. Avg_Utilization_Ratio - Average credit card utilization ratio (Quantitative)

Below are a few example rows from the dataset.

1	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon
2	Existing Customer	45	M	3	High School	Married	$60K - $80K	Blue	39	5	1
3	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44	6	1
4	Existing Customer	51	M	3	Graduate	Married	$80K - $120K	Blue	36	4	1
5	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	34	3	4
6	Existing Customer	40	M	3	Uneducated	Married	$60K - $80K	Blue	21	5	1
7	Existing Customer	44	M	2	Graduate	Married	$40K - $60K	Blue	36	3	1
8	Existing Customer	51	M	4	Unknown	Married	$120K +	Gold	46	6	1
9	Existing Customer	32	M	0	High School	Unknown	$60K - $80K	Silver	27	2	2
10	Existing Customer	37	M	3	Uneducated	Single	$60K - $80K	Blue	36	5	2
11	Existing Customer	48	M	2	Graduate	Single	$80K - $120K	Blue	36	6	3
12	Existing Customer	42	M	5	Uneducated	Unknown	$120K +	Blue	31	5	3
13	Existing Customer	65	M	1	Unknown	Married	$40K - $60K	Blue	54	6	2
14	Existing Customer	56	M	1	College	Single	$80K - $120K	Blue	36	3	6
15	Existing Customer	35	M	3	Graduate	Unknown	$60K - $80K	Blue	30	5	1
16	Existing Customer	57	F	2	Graduate	Married	Less than $40K	Blue	48	5	2
17	Existing Customer	44	M	4	Unknown	Unknown	$80K - $120K	Blue	37	5	1
18	Existing Customer	48	M	4	Post-Graduate	Single	$80K - $120K	Blue	36	6	2
19	Existing Customer	41	M	3	Unknown	Married	$80K - $120K	Blue	34	4	4
20	Existing Customer	61	M	1	High School	Married	$40K - $60K	Blue	56	2	2
21	Existing Customer	45	F	2	Graduate	Married	Unknown	Blue	37	6	1
22	Existing Customer	47	M	1	Doctorate	Divorced	$60K - $80K	Blue	42	5	2
23	Attrited Customer	62	F	0	Graduate	Married	Less than $40K	Blue	49	2	3

Contacts_Count_12_mon	Credit_Limit	Total_Revolving_Bal	Avg_Open_To_Buy	Total_Amt_Chng_Q4_Q1	Total_Trans_Amt	Total_Trans_Ct	Total_Ct_Chng_Q4_Q1	Avg_Utilization_Ratio
3	12691	777	11914	1.335	1144	42	1.625	0.061
2	8256	864	7392	1.541	1291	33	3.714	0.105
0	3418	0	3418	2.594	1887	20	2.333	0
1	3313	2517	796	1.405	1171	20	2.333	0.76
0	4716	0	4716	2.175	816	28	2.5	0
2	4010	1247	2763	1.376	1088	24	0.846	0.311
3	34516	2264	32252	1.975	1330	31	0.722	0.066
2	29081	1396	27685	2.204	1538	36	0.714	0.048
0	22352	2517	19835	3.355	1350	24	1.182	0.113
3	11656	1677	9979	1.524	1441	32	0.882	0.144
2	6748	1467	5281	0.831	1201	42	0.68	0.217
3	9095	1587	7508	1.433	1314	26	1.364	0.174
0	11751	0	11751	3.397	1539	17	3.25	0
3	8547	1666	6881	1.163	1311	33	2	0.195
2	2436	680	1756	1.19	1570	29	0.611	0.279
2	4234	972	3262	1.707	1348	27	1.7	0.23
3	30367	2362	28005	1.708	1671	27	0.929	0.078
1	13535	1291	12244	0.653	1028	21	1.625	0.095
3	3193	2517	676	1.831	1336	30	1.143	0.788
2	14470	1157	13313	0.966	1207	21	0.909	0.08
0	20979	1800	19179	0.906	1178	27	0.929	0.086
3	1438.3	0	1438.3	1.047	692	16	0.6	0

By using df.describe() we got statistical values for each column as shown below

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Limit
count	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000
mean	46.325960	2.346203	35.928409	3.812580	2.341167	2.455317	8631.953698
std	8.016814	1.298908	7.986416	1.554408	1.010622	1.106225	9088.776650
min	26.000000	0.000000	13.000000	1.000000	0.000000	0.000000	1438.300000
25%	41.000000	1.000000	31.000000	3.000000	2.000000	2.000000	2555.000000
50%	46.000000	2.000000	36.000000	4.000000	2.000000	2.000000	4549.000000
75%	52.000000	3.000000	40.000000	5.000000	3.000000	3.000000	11067.500000
max	73.000000	5.000000	56.000000	6.000000	6.000000	6.000000	34516.000000

Total_Revolving_Bal	Avg_Open_To_Buy	Total_Amt_Chng_Q4_Q1	Total_Trans_Amt	Total_Trans_Ct	Total_Ct_Chng_Q4_Q1	Avg_Utilization_Ratio
10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000
1162.814061	7469.139637	0.759941	4404.086304	64.858695	0.712222	0.274894
814.987335	9090.685324	0.219207	3397.129254	23.472570	0.238086	0.275691
0.000000	3.000000	0.000000	510.000000	10.000000	0.000000	0.000000
359.000000	1324.500000	0.631000	2155.500000	45.000000	0.582000	0.023000
1276.000000	3474.000000	0.736000	3899.000000	67.000000	0.702000	0.176000
1784.000000	9859.000000	0.859000	4741.000000	81.000000	0.818000	0.503000
2517.000000	34516.000000	3.397000	18484.000000	139.000000	3.714000	0.999000

Then we continue to check if our dataset has duplicates and missing values which we do not have.

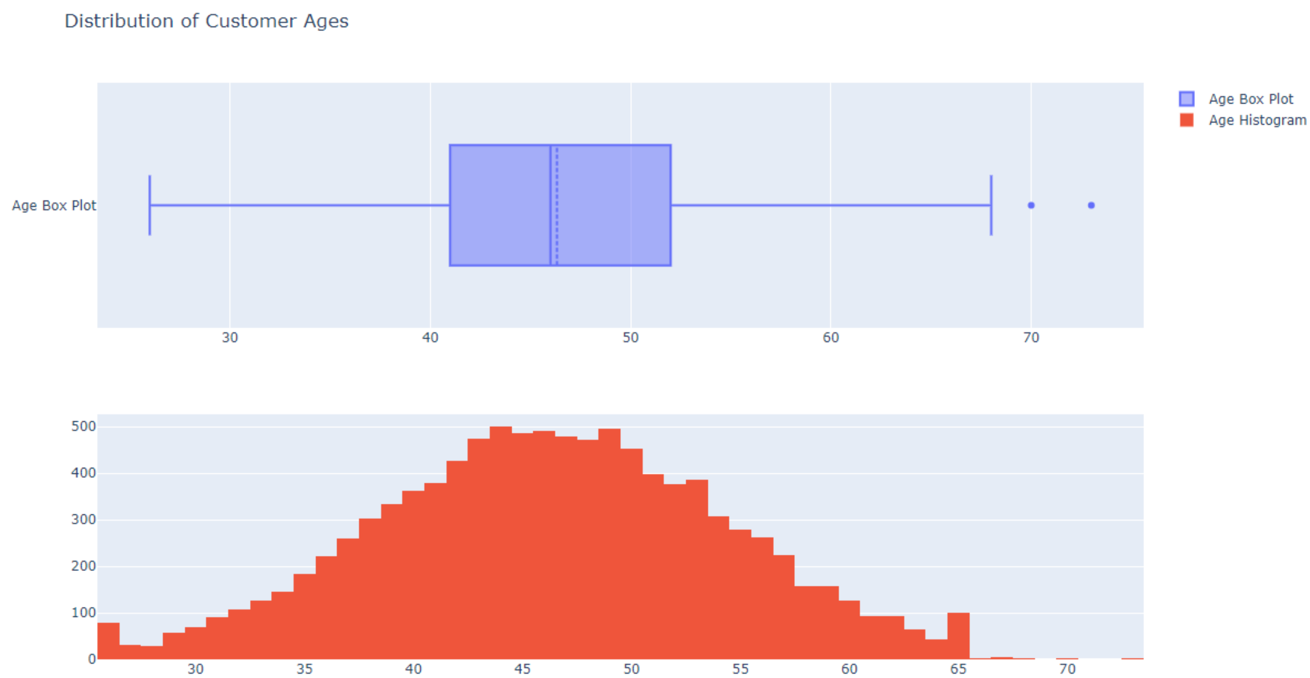
```
counts = df.Attrition_Flag.value_counts()
perc_attri = (counts[1]/(counts[0]+counts[1]))*100
duplicates = len(df[df.duplicated()])
missing_values = df.isnull().sum().sum()
types = df.dtypes.value_counts()

print("Existing Customer %d"%counts[0])
print("Attrited Customer %d"%counts[1])
print("Attrited Rate = %.1f %% \n"%(perc_attri))

print('Number of Duplicate Entries: %d'%(duplicates))
print('Number of Missing Values: %d \n'%(missing_values))

print('Number of Features: %d'%(df.shape[1]))
print('Number of Customers: %d \n'%(df.shape[0]))

print('Data Types in Dataset:')
print(types)
```

```
Existing Customer 8500
Attrited Customer 1627
Attrited Rate = 16.1 %

Number of Duplicate Entries: 0
Number of Missing Values: 0

Number of Features: 20
Number of Customers: 10127

Data Types in Dataset:
int64      9
object     6
float64    5
dtype: int64
```

3. Data visualization

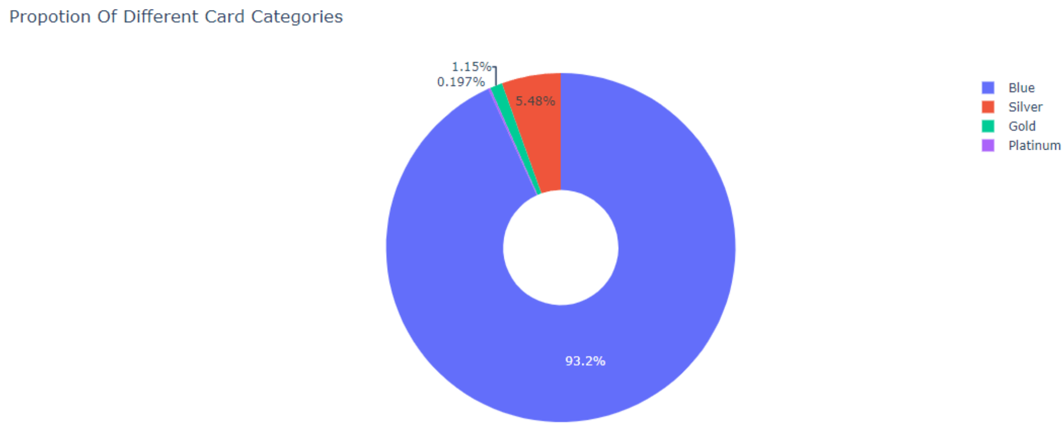
In this chapter, we visualize our dataset by using different appropriate plots for each purpose, to get some insights and overview of our dataset. We also look for interesting plots that may improve our models by doing some method depending on its result.

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Customer_Age'],name='Age Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Customer_Age'],name='Age Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of Customer Ages")
fig.show()
```



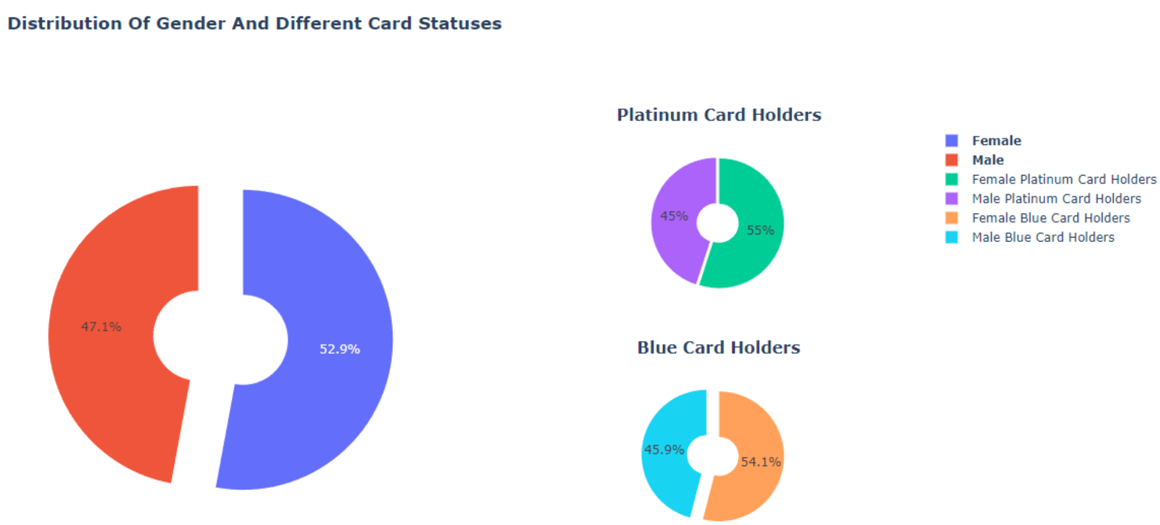
We can see that the distribution of customer ages follows a fairly normal distribution. Thus, further use of the age feature can be done with the normality assumption.

```
ex.pie(df,names='Card_Category',title='Propotion Of Different Card Categories',hole=0.33)
```



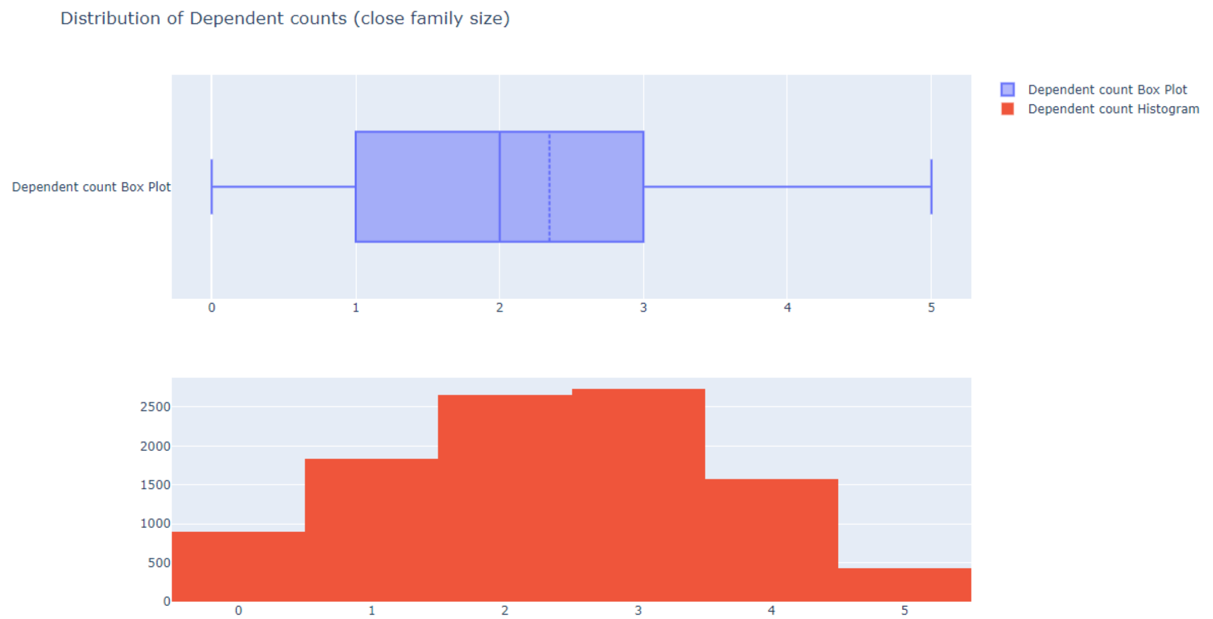
This feature may not be very useful since the difference in proportion is drastic, so we can't determine the pattern inside.

```
fig = make_subplots(
    rows=2, cols=2,subplot_titles=('','<b>Platinum Card Holders','<b>Blue Card Holders<b>','Residuals'),
    vertical_spacing=0.09,
    specs=[[{"type": "pie","rowspan": 2}       ,{"type": "pie"}] ,
           [None                               ,{"type": "pie"}]            ,
          ]
)

fig.add_trace(
    go.Pie(values=df.Gender.value_counts().values,labels=['<b>Female<b>','<b>Male<b>'],hole=0.3,pull=[0,0.3]),
    row=1, col=1
)

fig.add_trace(
    go.Pie(
        labels=['Female Platinum Card Holders','Male Platinum Card Holders'],
        values=df.query('Card_Category=="Platinum"').Gender.value_counts().values,
        pull=[0,0.05,0.5],
        hole=0.3

    ),
    row=1, col=2
)

fig.add_trace(
    go.Pie(
        labels=['Female Blue Card Holders','Male Blue Card Holders'],
        values=df.query('Card_Category=="Blue"').Gender.value_counts().values,
        pull=[0,0.2,0.5],
        hole=0.3
    ),
    row=2, col=2
)



fig.update_layout(
    height=800,
    showlegend=True,
    title_text="<b>Distribution Of Gender And Different Card Statuses<b>",
)

fig.show()
```



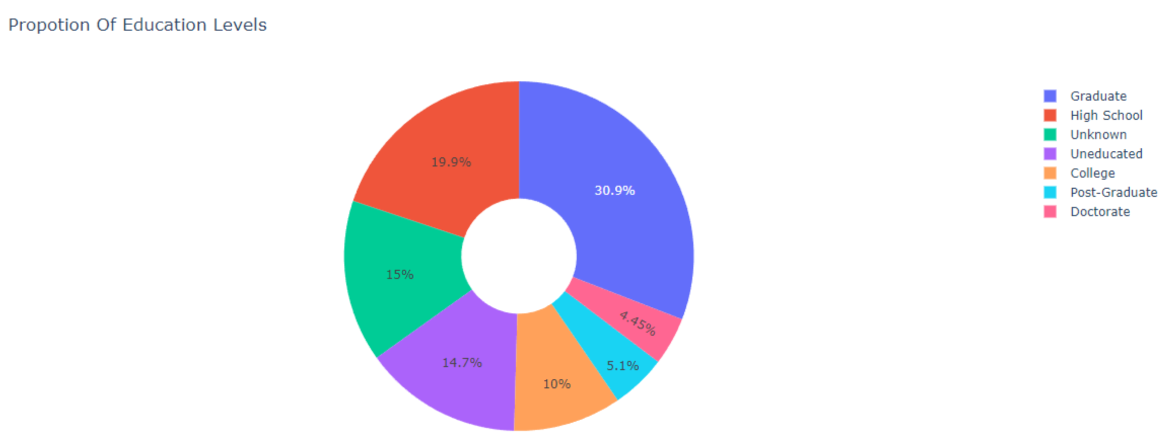
We have more samples of females compared to males samples, but the difference is not that significant, so we can say that genders are uniformly distributed which is good.

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Dependent_count'],name='Dependent count Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Dependent_count'],name='Dependent count Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of Dependent counts (close family size)")
fig.show()
```



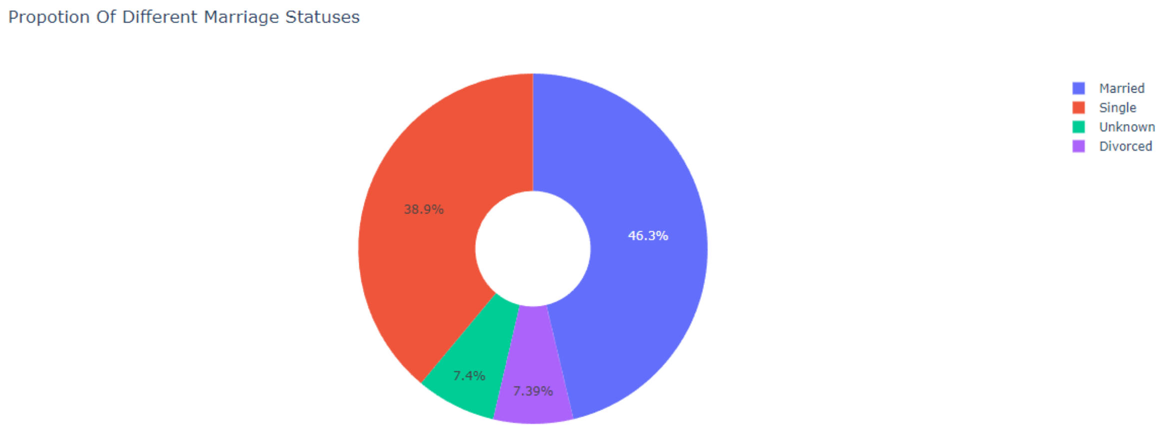
The distribution of Dependent counts is fairly normally distributed with a slight right skew.

```
ex.pie(df,names='Education_Level',title='Propotion Of Education Levels',hole=0.33)
```



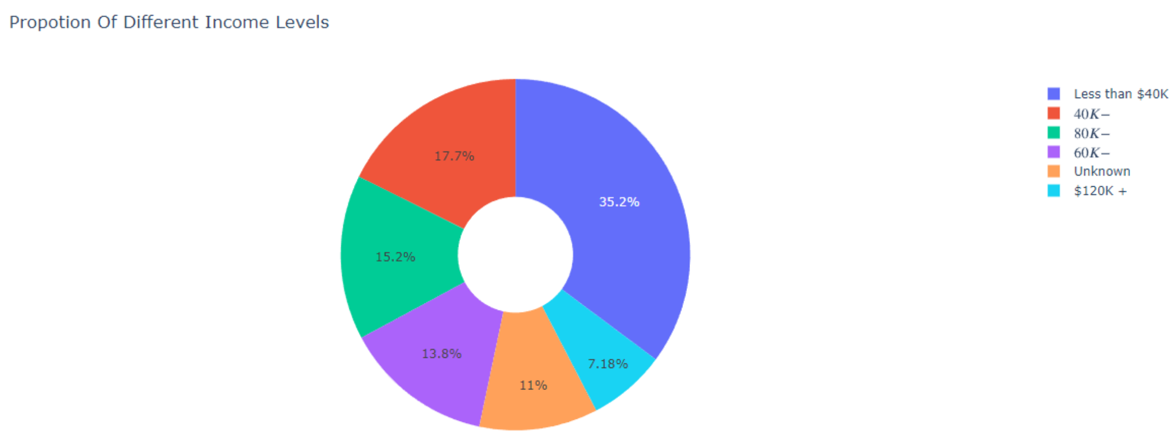
Suppose most unknown education customers lack education, then we can state that more than 70% of the customers have a formal education level, and about 35% have a higher level of education.

```
ex.pie(df,names='Marital_Status',title='Propotion Of Different Marriage Statuses',hole=0.33)
```



The feature is divided into 2 halves which are Married half and Single half; both have percentages at 46% and 38% respectively. Only about 7% of the customers are divorced and unknown.

```
ex.pie(df,names='Income_Category',title='Propotion Of Different Income Levels',hole=0.33)
```



Income less than 40k is the majority of this feature at 35% and the more income the lower percentage of records. However, the proportion of each type is not drastically different, so we can say that this feature should not be a problem in the future.

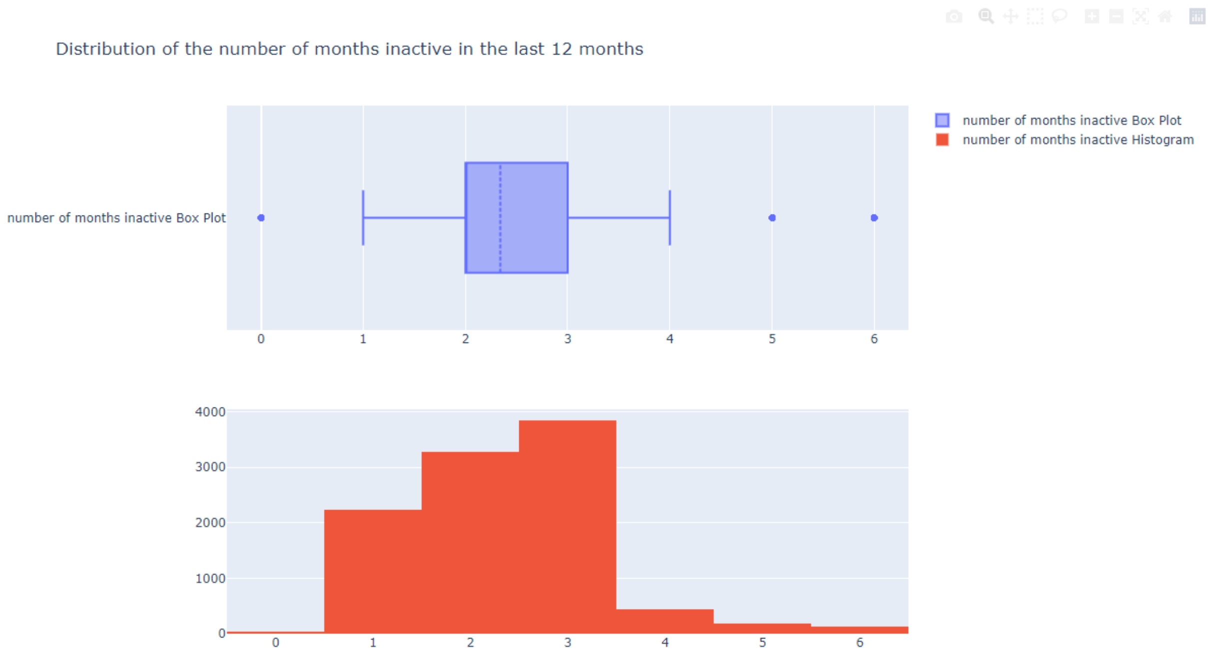
```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Months_on_book'],name='Months on book Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Months_on_book'],name='Months on book Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of months the customer is part of the bank")
fig.show()
```

Obviously by looking at the extremely high value, we can say that this feature is not normal distribution.

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Total_Relationship_Count'],name='Total no. of products Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Total_Relationship_Count'],name='Total no. of products Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of Total no. of products held by the customer")
fig.show()
```

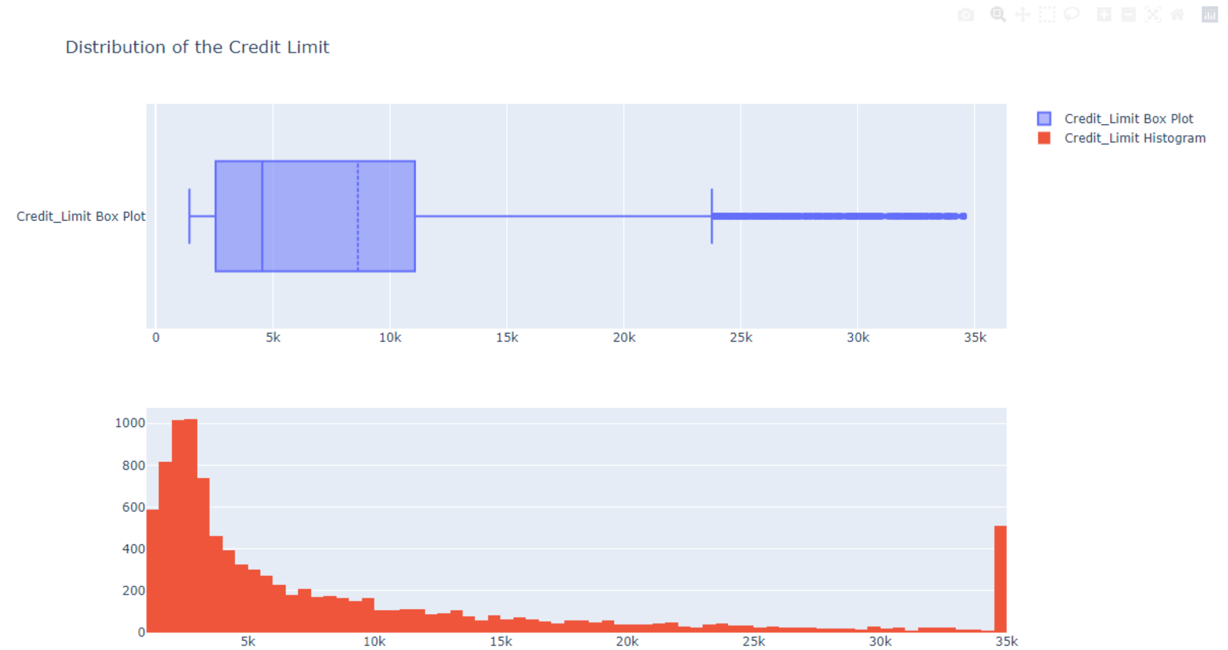
The distribution of the total number of products held by the customer seems closer to a uniform distribution.

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Months_Inactive_12_mon'],name='number of months inactive Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Months_Inactive_12_mon'],name='number of months inactive Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of the number of months inactive in the last 12 months")
fig.show()
```



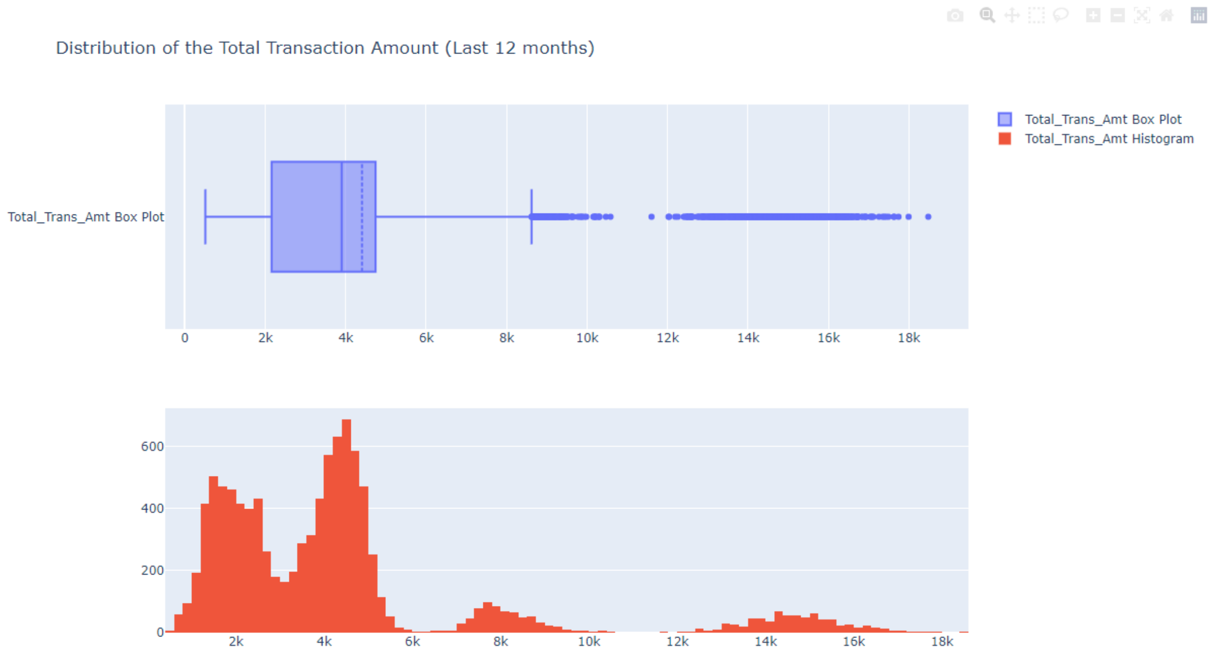
The data records are very dense at values around 1, 2 and 3 and the distribution is right skewed.

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Credit_Limit'],name='Credit_Limit Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Credit_Limit'],name='Credit_Limit Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of the Credit Limit")
fig.show()
```



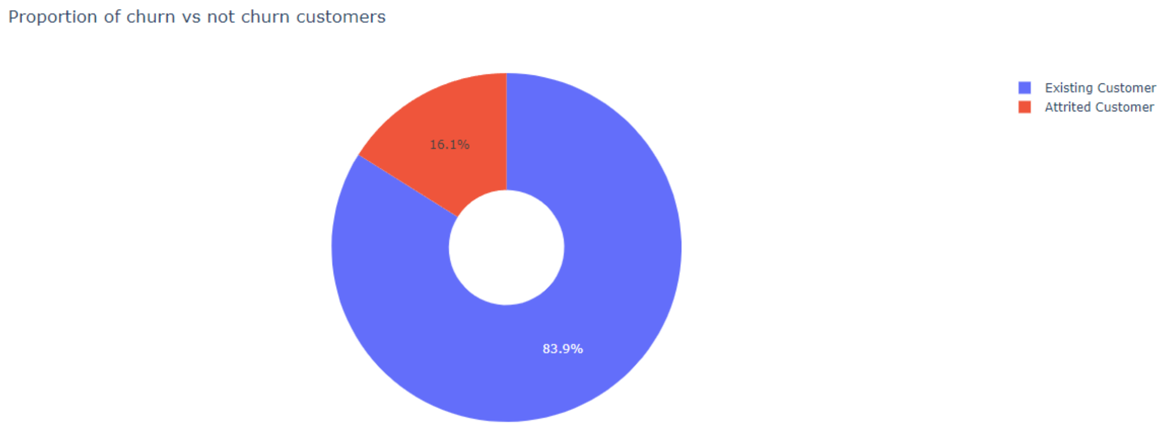
The distribution is right skewed but it has a lot of records that hold a credit limit around 35k. We tried to find some insight of it, but did not find anything useful for our models.

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=df['Total_Trans_Amt'],name='Total_Trans_Amt Box Plot',boxmean=True)
tr2=go.Histogram(x=df['Total_Trans_Amt'],name='Total_Trans_Amt Histogram')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=700, width=1200, title_text="Distribution of the Total Transaction Amount (Last 12 months)")
fig.show()
```



We see that the distribution of the total transactions (Last 12 months) is shown as a multimodal distribution, which means that we have some underlying groups in our data, therefore, we can experiment to try and cluster the different groups and view the similarities between them.

```
ex.pie(df,names='Attrition_Flag',title='Proportion of churn vs not churn customers',hole=0.33)
```



We have only 16% of the data samples representing Attrited Customers samples compared to 83% of the Existing Customer samples, which means that our class feature has an imbalanced data problem. Therefore, we try to upsample the Attrited Customer sample to match the number of another sample and we decided to use SMOTE(Synthetic Minority Oversampling Technique) in the upsampling process. This experiment will be in the Decision Tree model section.

4. Methodology

This chapter will provide the process of training model, testing model, model evaluation, background ideas, and code implementation in Python and R program.

4.1 Decision Tree

In the Decision Tree section, we preprocessed our dataset, then we used the preprocessed dataset in 3 different experiments including Basic Decision Tree and Random Forest models, models from experiment 1 with SMOTE, and models from experiment 1 with Feature Selection.

The evaluation metrics are F1-score and Accuracy with 10-fold cross validation. 3 experiments provide both metrics but the interpretations of models are based on F1-score more, because F1-score is more sensitive to incorrect prediction from models. However, the SMOTE experiment will provide F1-score and Accuracy with testing data from train_test_split, because when combining SMOTE and cross validation, the process is too complicated.

4.1.1 Preprocessing dataset

We label categorical features by using One-hot encoder which basically is creating new columns for different values and label as 0 or 1. For numerical features, we apply MinMaxScaler on each column. Code and result are shown below. Additionally for features that have 4 different values, we can delete 1 column and have 3 columns, since 3 columns are enough for 4 combinations. (1,0,0) (0,1,0) (0,0,1) (0,0,0)

```
# preprocessing
df = pd.read_csv("D:/! Work/term8/447/project/BankChurners2.csv")

df.Attrition_Flag = df.Attrition_Flag.replace({'Attrited Customer':1,'Existing Customer':0})
df.Gender = df.Gender.replace({'F':1,'M':0})
df = pd.concat([df,pd.get_dummies(df['Education_Level']).drop(columns=['Unknown'])],axis=1)
df = pd.concat([df,pd.get_dummies(df['Income_Category']).drop(columns=['Unknown'])],axis=1)
df = pd.concat([df,pd.get_dummies(df['Marital_Status']).drop(columns=['Unknown'])],axis=1)
df = pd.concat([df,pd.get_dummies(df['Card_Category']).drop(columns=['Platinum'])],axis=1)
df.drop(columns = ['Education_Level','Income_Category','Marital_Status','Card_Category'],inplace=True)

# minmax scaling numeric features
floatcols = df.select_dtypes(include = ['Float64']).columns
for col in df[floatcols]:
    df[col] = MinMaxScaler().fit_transform(df[[col]])

intcols = df.select_dtypes(include = ['int64']).columns
for col in df[intcols]:
    df[col] = MinMaxScaler().fit_transform(df[[col]])

print('New Number of Features: %d'%(df.shape[1]))
df.head(10)
```

```
New Number of Features: 33
```

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon
0	0.0	0.404255	0.0	0.6	0.604651	0.8	0.166667	0.500000
1	0.0	0.489362	1.0	1.0	0.720930	1.0	0.166667	0.333333
2	0.0	0.531915	0.0	0.6	0.534884	0.6	0.166667	0.000000
3	0.0	0.297872	1.0	0.8	0.488372	0.4	0.666667	0.166667
4	0.0	0.297872	0.0	0.6	0.186047	0.8	0.166667	0.000000
5	0.0	0.382979	0.0	0.4	0.534884	0.4	0.166667	0.333333
6	0.0	0.531915	0.0	0.8	0.767442	1.0	0.166667	0.500000
7	0.0	0.127660	0.0	0.0	0.325581	0.2	0.333333	0.333333
8	0.0	0.234043	0.0	0.6	0.534884	0.8	0.333333	0.000000
9	0.0	0.468085	0.0	0.4	0.534884	1.0	0.500000	0.500000

Credit_Limit	Total_Revolving_Bal	Avg_Open_To_Buy	Total_Amt_Chng_Q4_Q1	Total_Trans_Amt	Total_Trans_Ct	Total_Ct_Chng_Q4_Q1	Avg_Utilization_Ratio
0.340190	0.308701	0.345116	0.392994	0.035273	0.248062	0.437534	0.061061
0.206112	0.343266	0.214093	0.453636	0.043452	0.178295	1.000000	0.105105
0.059850	0.000000	0.098948	0.763615	0.076611	0.077519	0.628164	0.000000
0.056676	1.000000	0.022977	0.413600	0.036775	0.077519	0.628164	0.760761
0.099091	0.000000	0.136557	0.640271	0.017025	0.139535	0.673129	0.000000
0.077747	0.495431	0.079970	0.405063	0.032158	0.108527	0.227787	0.311311
1.000000	0.899484	0.934402	0.581395	0.045621	0.162791	0.194400	0.066066
0.835690	0.554629	0.802075	0.648808	0.057194	0.201550	0.192246	0.048048
0.632260	1.000000	0.574624	0.987636	0.046734	0.108527	0.318255	0.113113
0.308900	0.666269	0.289051	0.448631	0.051797	0.170543	0.237480	0.144144

College	Doctorate	Graduate	High School	Post-Graduate	Uneducated	$120K +	40K – 60K	60K – 80K	80K – 120K	Less than $40K	Divorced	Married	Single	Blue	Gold	Silver
0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0	0	0	1	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0

4.1.2 Experiment1: Basic Decision Tree and Random Forest model

In this experiment, we create 3 models including a non-tuning parameter Decision Tree, a tuning parameter Decision Tree, and non-tuning parameter Random Forest. Note that this experiment does not select any features manually which means we provide models with the whole dataset. The code below shows the parameter tuning process by looping.

4 DT with preprocessed dataset

```
x = df.iloc[:,1:]
y = df.iloc[:,0]

def eval_model_10randomState(model, model_name, X, y):
    accA=[]
    f1A=[]
    for k in range(1,11):
        train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.20, random_state = k)

        # fit model
        model.fit(train_x,train_y)
        model.score(test_x, test_y)
        pred_test = model.predict(test_x)

        # get metrics
        f1 = metrics.f1_score(test_y, pred_test)
        test_acc = metrics.accuracy_score(test_y, pred_test)
        con = metrics.confusion_matrix(test_y, pred_test)
        accA.append(test_acc)
        f1A.append(f1)

    accA = np.mean(accA)
    f1A = np.mean(f1A)
#     print(con,'%s:        %.4f F1-score        %.4f accuracy'%(model_name, f1, test_acc))
    print(model_name, '        %.4f F1-score'%f1A, '        %.4f accuracy'%accA)
```

4.1 base dt model result

```
dt = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2, random_state=0)
name = 'base dt model'
eval_model_10randomState(dt, name, x, y)
```

```
base dt model         0.8059 F1-score         0.9392 accuracy
```

```
# base dt model         0.8059 F1-score         0.9392 accuracy
```

4.2 base rf model result

```
rf = RandomForestClassifier(random_state=0)
name = 'base rf model'
eval_model_10randomState(rf, name, x, y)
```

```
base rf model         0.8442 F1-score         0.9552 accuracy
```

```
# base rf model         0.8442 F1-score         0.9552 accuracy
```

4.3 parameter tuning ¶

```
for i in range(1,21):
    dt = tree.DecisionTreeClassifier(max_depth=i, min_samples_split=2, random_state=0)
    name = 'max_depth '+str(i)
    eval_model_10randomState(dt, name, x, y)
    print('- - - - - - - - - - - - - - - - - - - - - - - - - - ')
# max_depth 8         0.8197 F1-score
```

```
max_depth 1         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 2         0.5991 F1-score         0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 3         0.7093 F1-score         0.9171 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 4         0.7264 F1-score         0.9235 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 5         0.7892 F1-score         0.9338 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 6         0.8058 F1-score         0.9417 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 7         0.8154 F1-score         0.9437 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 8         0.8197 F1-score         0.9441 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 9         0.8172 F1-score         0.9438 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 10         0.8131 F1-score         0.9421 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 11         0.8145 F1-score         0.9427 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 12         0.8031 F1-score         0.9387 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 13         0.8064 F1-score         0.9397 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 14         0.8025 F1-score         0.9384 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 15         0.8008 F1-score         0.9376 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 16         0.8043 F1-score         0.9386 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 17         0.8031 F1-score         0.9384 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 18         0.8050 F1-score         0.9389 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 19         0.8057 F1-score         0.9390 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
max_depth 20         0.8058 F1-score         0.9392 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
for i in range(1,21):
    minsplit = 5+(i*5)
    dt = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=minsplit, random_state=0)
    name = 'minsplit '+str(minsplit)
    eval_model_10randomState(dt, name, x, y)
    print('- - - - - - - - - - - - - - - - - - - - - - - - - - ')
# minsplit 35         0.8146 F1-score
```

```
minsplit 10         0.8045 F1-score         0.9393 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 15         0.8074 F1-score         0.9406 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 20         0.8103 F1-score         0.9414 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 25         0.8124 F1-score         0.9422 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 30         0.8122 F1-score         0.9420 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 35         0.8146 F1-score         0.9425 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 40         0.8132 F1-score         0.9423 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 45         0.8128 F1-score         0.9422 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 50         0.8136 F1-score         0.9424 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 55         0.8099 F1-score         0.9418 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 60         0.8119 F1-score         0.9422 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 65         0.8110 F1-score         0.9418 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 70         0.8064 F1-score         0.9408 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 75         0.8059 F1-score         0.9413 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 80         0.8041 F1-score         0.9410 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 85         0.8000 F1-score         0.9398 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 90         0.7976 F1-score         0.9389 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 95         0.7931 F1-score         0.9373 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 100         0.7890 F1-score         0.9365 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
minsplit 105         0.7847 F1-score         0.9354 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
for j in range(1,21):
    for i in range(1,21):
        minsplit = 5+(i*5)
        dt = tree.DecisionTreeClassifier(max_depth=j, min_samples_split=minsplit, random_state=0)
        name = 'depth'+str(j)+'   minsplit'+str(minsplit)
        eval_model_10randomState(dt, name, x, y)
        print('- - - - - - - - - - - - - - - - - - - - - - - - - - ')
    print('\n- - - - - - - - - - - - - - - - - - - - - - - - - - \n')
# depth8   minsplit20         0.8210 F1-score         0.9446 accuracy
```

```
depth1   minsplit10         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit15         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit20         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit25         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit30         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit35         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit40         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit45         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit50         0.0000 F1-score         0.8431 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - -
depth1   minsplit55         0.0000 F1-score         0.8431 accuracy
```

Code below did the 10-fold cross validation for each model, then we had results in the table below.

```
# 'threshold no' means whole preprocessed dataset
# 'threshold 2'  means not whole dataset(only features appearing > 2 times in feature selection)

# select columns
X = df.iloc[:,1:]
y = df.iloc[:,0]

# select different X
x_no_feature_selection = X
x_threshold1 = X[get_features(1)]
x_threshold2 = X[get_features(2)]

# split x y for traning models
train_x_noFS, test_x_noFS, train_y_noFS, test_y_noFS = train_test_split(x_no_feature_selection, y, test_size = 0.20, random_state = 0)
train_x_th1, test_x_th1, train_y_th1, test_y_th1 = train_test_split(x_threshold1, y, test_size = 0.20, random_state = 0)
train_x_th2, test_x_th2, train_y_th2, test_y_th2 = train_test_split(x_threshold2, y, test_size = 0.20, random_state = 0)
```

6.3 result 10cv each models

6.3.1 tree

```
# tree     threshold no    base model              0.8059 F1-score         0.9392 accuracy
tree1 = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2, random_state=0)
tree1.fit(train_x_noFS,train_y_noFS)

pred_test = tree1.predict(test_x_noFS)
f1 = metrics.f1_score(test_y_noFS, pred_test)
test_acc = metrics.accuracy_score(test_y_noFS, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(tree1, x_no_feature_selection, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

```
test score: f1=0.7718   acc=0.9314

10cv score:
f1 = 0.7348705291039229
acc = 0.9045019489716687
```

```
# tree     threshold no   depth8  minsplit20      0.8210 F1-score         0.9446 accuracy
tree2 = tree.DecisionTreeClassifier(max_depth=8, min_samples_split=20, random_state=0)
tree2.fit(train_x_noFS,train_y_noFS)

pred_test = tree2.predict(test_x_noFS)
f1 = metrics.f1_score(test_y_noFS, pred_test)
test_acc = metrics.accuracy_score(test_y_noFS, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(tree2, x_no_feature_selection, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

```
test score: f1=0.8247   acc=0.9467

10cv score:
f1 = 0.7539037822149429
acc = 0.9140826371791221
```

6.3.2 forest

```
# forest   threshold no    base model              0.8442 F1-score         0.9552 accuracy
forest1 = RandomForestClassifier(random_state=0)
forest1.fit(train_x_noFS,train_y_noFS)

pred_test = forest1.predict(test_x_noFS)
f1 = metrics.f1_score(test_y_noFS, pred_test)
test_acc = metrics.accuracy_score(test_y_noFS, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(forest1, x_no_feature_selection, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

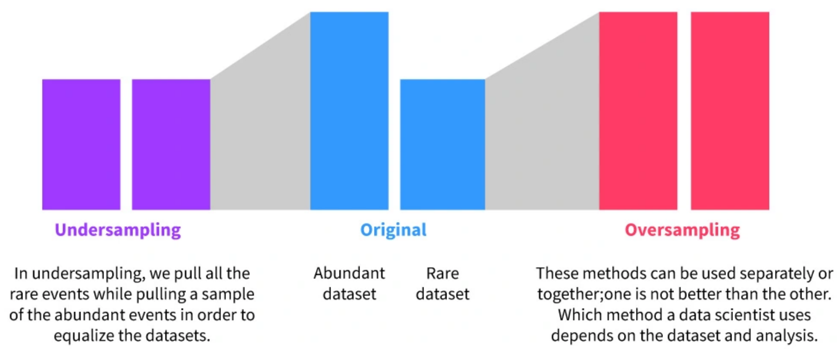
```
test score: f1=0.8144   acc=0.9492

10cv score:
f1 = 0.7941790810140734
acc = 0.9413414153553215
```

Model	10cv F1-score	10cv Accuracy
Base DT	0.7349	0.9045
Parameter tuned DT	0.7539	0.914
Base RF	0.7942	0.9413

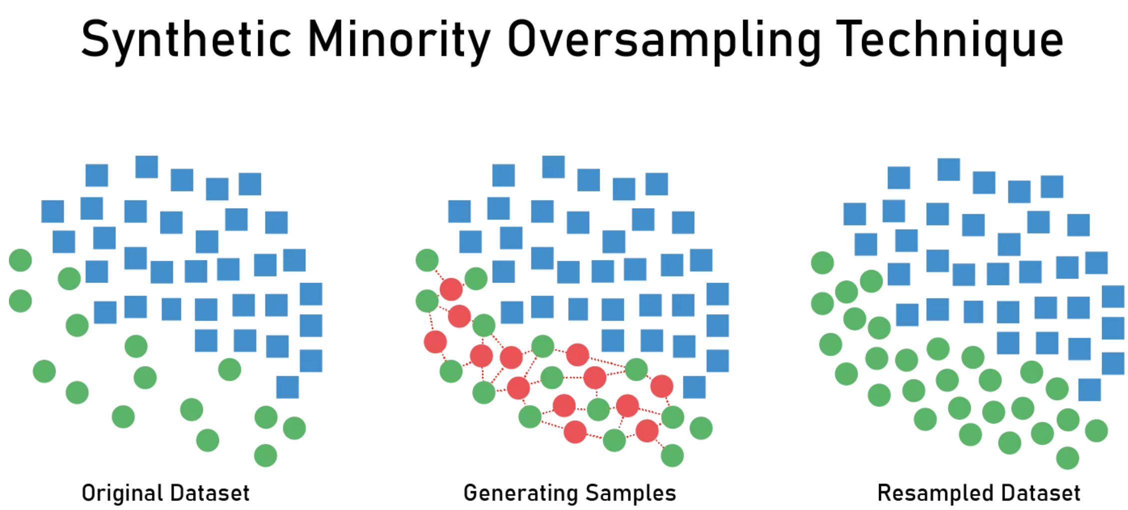
This experiment shows that Random Forest, which is a more advanced version of Decision Tree, performs better than Decision Tree even after tuning parameters, but tuning parameters does improve the performance of the Decision Tree model. Depth 8 minsplit 20 is the best parameter for Decision Tree.

4.1.3 Experiment2: Models with SMOTE

SMOTE(Synthetic Minority Oversampling Technique) is an upsampling technique. The concept of upsampling is to generate new rows to help models learn better and find more patterns inside. Common resampling technique is random upsampling but SMOTE adds a new concept and prevents randomly generating new rows. SMOTE uses a pair of points in class that we want to generate and then create new points between both points. By doing this, the new point will not lose the characteristic and pattern of the feature.



Picture from https://www.mastersindatascience.org/learning/statistics-data-science/undersampling/



Picture from https://medium.com/analytics-vidhya/bank-data-smote-b5cb01a5e0a2

We import SMOTE to do the upsampling process as we mentioned earlier and compare it with models from experiment 1. However, in this experiment, we used both evaluation metrics based on the testing set instead of 10-cv cross validation because of complexity. Code below shows upsampling, training, and evaluating processes of models.

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
```

```
# select columns
X = df.iloc[:,1:]
y = df.iloc[:,0]
```

```
sm = SMOTE(random_state = 0)

# split x y for traning models
train_x, test_x, train_y, test_y = train_test_split(X,
                                                    y,
                                                    test_size = 0.20,
                                                    random_state = 0)

# Oversampling splitted training set
train_x_oversampled, train_y_oversampled = sm.fit_resample(train_x, train_y)
```

```
## tree
# tree     threshold no   base model              0.8059 F1-score         0.9392 accuracy
tree1 = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2, random_state=0)
tree1.fit(train_x_oversampled,train_y_oversampled)

pred_test = tree1.predict(test_x)
f1 = metrics.f1_score(test_y, pred_test)
test_acc = metrics.accuracy_score(test_y, pred_test)
print('test score: f1 = %.4f   acc = %.4f'%(f1,test_acc))
```

```
test score: f1 = 0.7688   acc = 0.9240
```

```
# tree     threshold no   depth8  minsplit20      0.8210 F1-score         0.9446 accuracy
tree2 = tree.DecisionTreeClassifier(max_depth=8, min_samples_split=20, random_state=0)
tree2.fit(train_x_oversampled,train_y_oversampled)

pred_test = tree2.predict(test_x)
f1 = metrics.f1_score(test_y, pred_test)
test_acc = metrics.accuracy_score(test_y, pred_test)
print('test score: f1 = %.4f   acc = %.4f'%(f1,test_acc))
```

```
test score: f1 = 0.7827   acc = 0.9255
```

```
## forest

# forest   threshold no   base model              0.8442 F1-score         0.9552 accuracy
forest1 = RandomForestClassifier(random_state=0)
forest1.fit(train_x_oversampled,train_y_oversampled)

pred_test = forest1.predict(test_x)
f1 = metrics.f1_score(test_y, pred_test)
test_acc = metrics.accuracy_score(test_y, pred_test)
print('test score: f1 = %.4f   acc = %.4f'%(f1,test_acc))
```

```
test score: f1 = 0.8534   acc = 0.9556
```

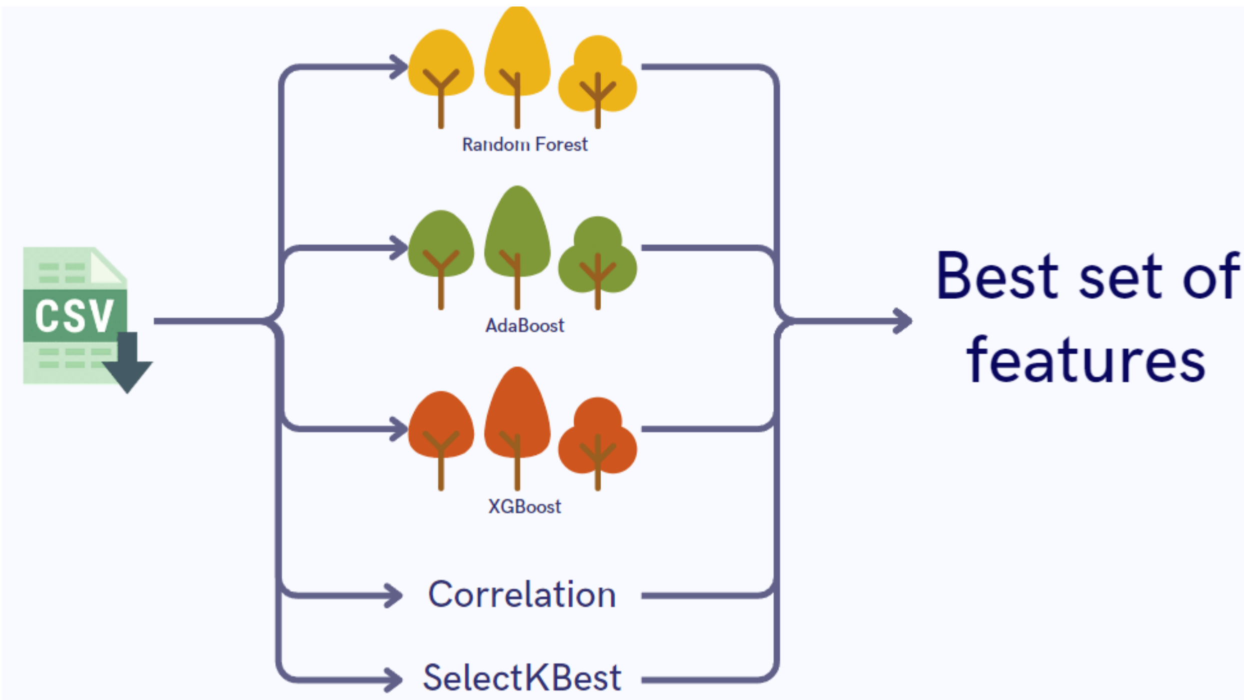
Model	testing set F1-score	testing set Accuracy	SMOTE F1-score	SMOTE Accuracy
Base DT	0.8059	0.9392	0.7688	0.9240
Parameter tuned DT	0.8210	0.9446	0.7827	0.9255
Base RF	0.8442	0.9552	0.8534	0.9556

This experiment shows that SMOTE did not help all models to perform better, which is unexpected. It only helps Random Forest to increase both metrics slightly and decrease performance of Decision Tree models significantly. Therefore, in experiment 3, we will not include the SMOTE models version with Feature Selection.

4.1.4 Experiment3: Models with Feature Selection

The concept of this feature selection is to feed our dataset into ensemble models including Random Forest, AdaBoost, and XGBoost and use the attribute feature_importances_ of models to find the best set of features. The reason is that these ensemble advanced models have the ability to choose features by themselves, which should be more appropriate than manually selecting. Other than selecting by ensemble models, we also use correlation between features and SelectKBest function to find sets of features.

After getting 5 different sets, we combine them and count the number of times a feature appears. We use these count values to represent how important the feature is, so we can select only features that are important based on its appearance. Below is the overview of this process.



Below are codes that show the process of training models and selecting features.

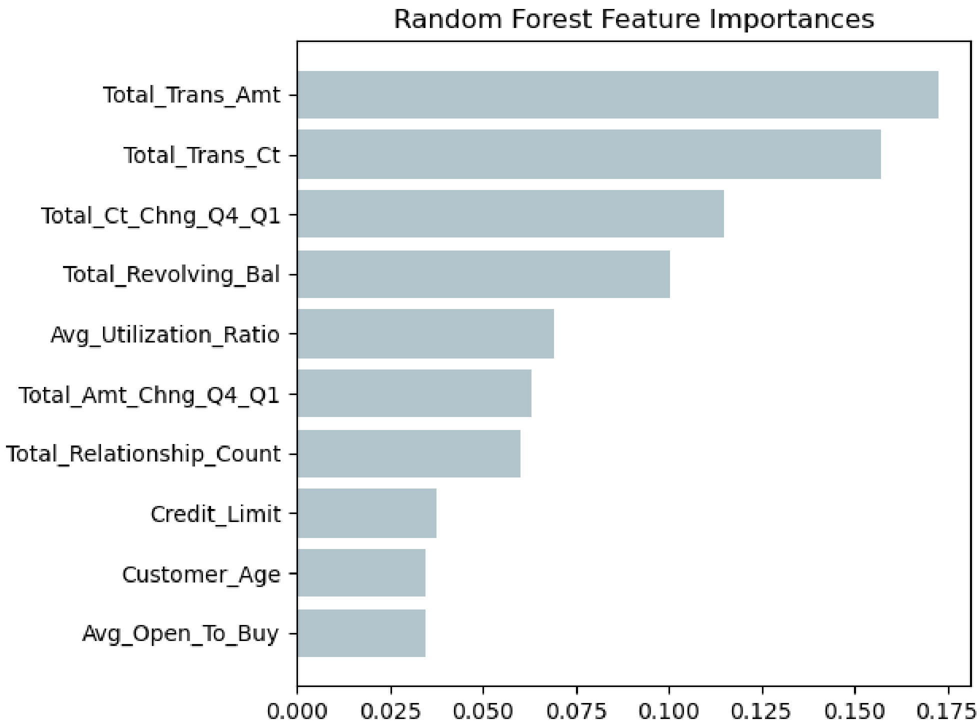
5 DT with feature selection RF+Ada+XG+Corr+KBest

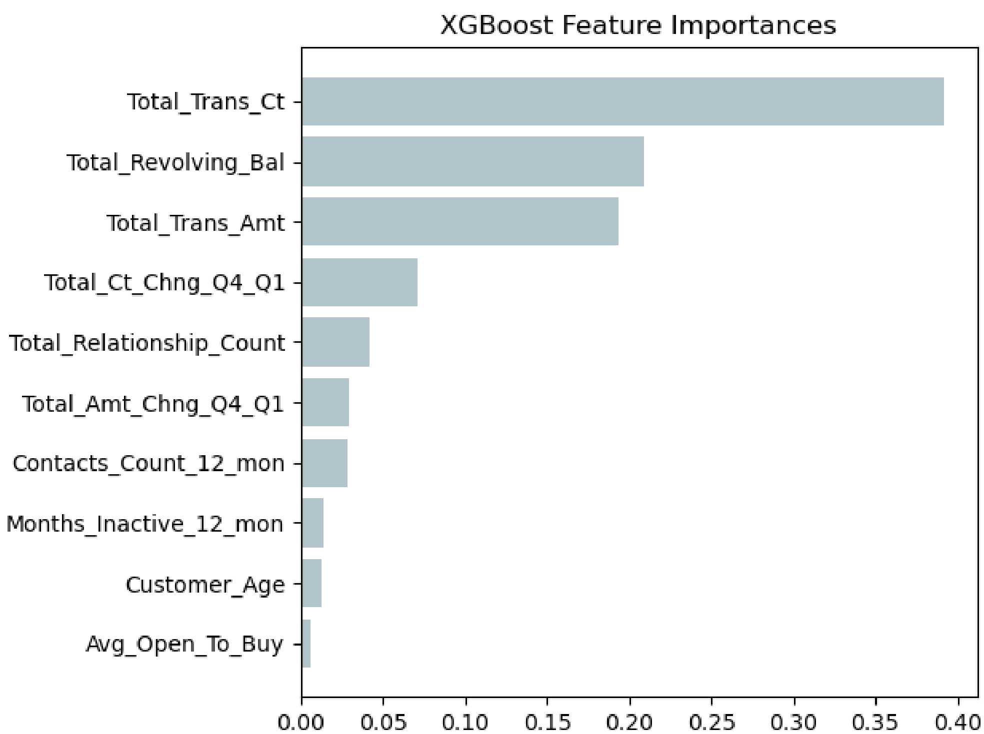
```
def plot_importances(model, model_name, features_to_plot, feature_names):
    #fit model and performances
    model.fit(x,y)
    importances = model.feature_importances_

    # sort and rank importances
    indices = np.argsort(importances)
    best_features = np.array(feature_names)[indices][-features_to_plot:]
    values = importances[indices][-features_to_plot:]

    # plot a graph
    y_ticks = np.arange(0, features_to_plot)
    fig, ax = plt.subplots()
    ax.barh(y_ticks, values, color = '#b2c4cc')
    ax.set_yticklabels(best_features)
    ax.set_yticks(y_ticks)
    ax.set_title("%s Feature Importances"%(model_name))
    fig.tight_layout()
    plt.show()

def best_features(model, features_to_plot, feature_names):
    # get list of best features
    model.fit(x,y)
    importances = model.feature_importances_

    indices = np.argsort(importances)
    best_features = np.array(feature_names)[indices][-features_to_plot:]
    return best_features
```

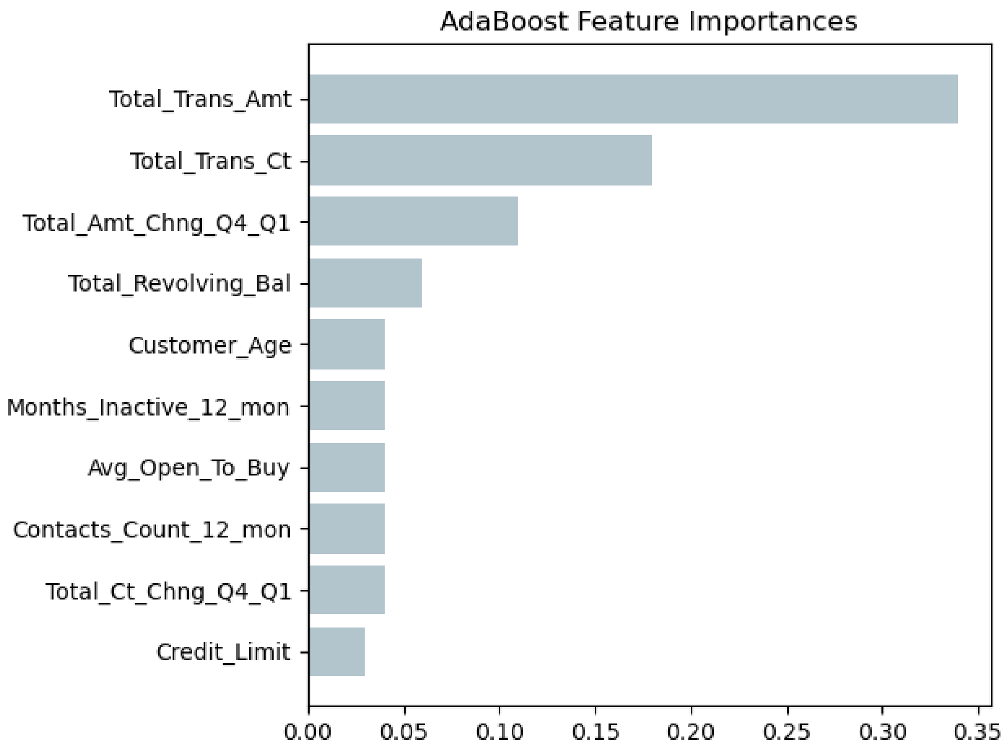
```
feature_names = list(x.columns)

model1 = RandomForestClassifier(random_state = 0)
plot_importances(model1, 'Random Forest', 10, feature_names)

model2 = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0, max_depth = 1, random_state = 0)
plot_importances(model2, 'XGBoost', 10, feature_names)

model3 = AdaBoostClassifier(n_estimators = 100, learning_rate = 1.0, random_state = 0)
plot_importances(model3, 'AdaBoost', 10, feature_names)
```

```
forest_best = list(best_features(model1, 10, feature_names))
XG_best = list(best_features(model2, 10, feature_names))
ada_best = list(best_features(model3, 10, feature_names))
```

The 3 plots below are the set of features that have been selected by ensemble models.



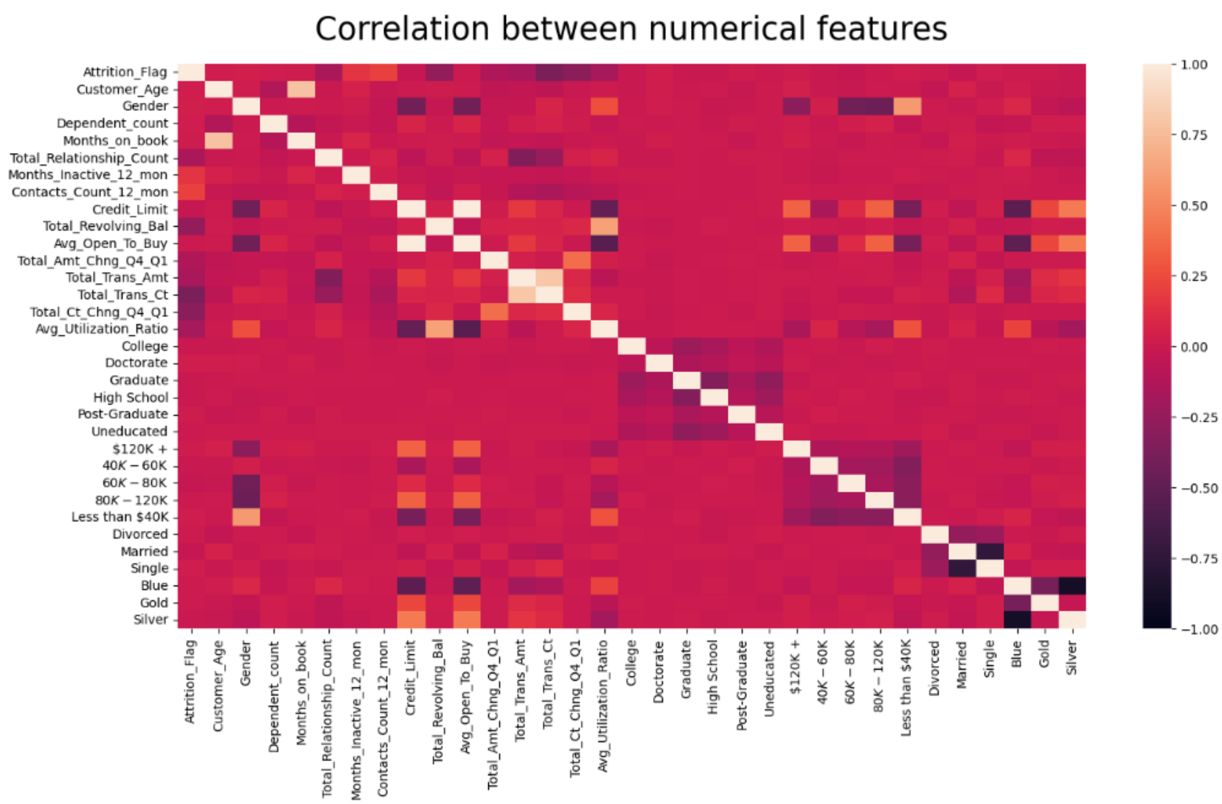



AdaBoost Feature Importances
Total_Trans_Amt
Total_Trans_Ct
Total_Amt_Chng_Q4_Q1
Total_Revolving_Bal
Customer_Age
Months_Inactive_12_mon
Avg_Open_To_Buy
Contacts_Count_12_mon
Total_Ct_Chng_Q4_Q1
Credit_Limit
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35

Below codes were used to select features set by SelectKBest and by correlation. We only select features that have correlation more than 0.1 or less than -0.1.

```
f_selector = SelectKBest(f_classif, k = 10)
f_selector.fit_transform(x, y)
f_selector_best = f_selector.get_feature_names_out()
f_selector_best = list(f_selector_best)
```

```
heat = df.corr()
plt.figure(figsize = [16,8])
plt.title("Correlation between numerical features", size = 25, pad = 20)
sns.heatmap(heat, vmin=-1.0, vmax=1.0, annot = False)
plt.show()

print("Correlation Coefficient of all the Features")
corr = df.corr()
corr.sort_values(["Attrition_Flag"], ascending = False, inplace = True)
correlations = corr.Attrition_Flag
a = correlations[correlations > 0.1]
b = correlations[correlations < -0.1]
top_corr_features = pd.concat([a,b])
print(top_corr_features[1:])
top_corr_features = list(top_corr_features.index[1:])
```




```
Correlation Coefficient of all the Features
Contacts_Count_12_mon         0.204491
Months_Inactive_12_mon        0.152449
Total_Amt_Chng_Q4_Q1         -0.131063
Total_Relationship_Count     -0.150005
Total_Trans_Amt              -0.168598
Avg_Utilization_Ratio        -0.178410
Total_Revolving_Bal          -0.263053
Total_Ct_Chng_Q4_Q1          -0.290054
Total_Trans_Ct               -0.371403
Name: Attrition_Flag, dtype: float64
```

After getting 5 sets from above, we combine them as shown in code below and count the appearance. We also create a function to help us select only features that appear more than a threshold that we can control. The example of threshold 4 is shown below.

```
best_features_overall = forest_best+XG_best+ada_best+f_selector_best+top_corr_features

from collections import Counter
count_best_features = dict(Counter(best_features_overall)) # count duplicate value in list

features_no_repeats = list(dict.fromkeys(best_features_overall)) # name no repeat

display(count_best_features)
```

```
{'Avg_Open_To_Buy': 3,
 'Customer_Age': 3,
 'Credit_Limit': 2,
 'Total_Relationship_Count': 4,
 'Total_Amt_Chng_Q4_Q1': 5,
 'Avg_Utilization_Ratio': 3,
 'Total_Revolving_Bal': 5,
 'Total_Ct_Chng_Q4_Q1': 5,
 'Total_Trans_Ct': 5,
 'Total_Trans_Amt': 5,
 'Months_Inactive_12_mon': 4,
 'Contacts_Count_12_mon': 4,
 'Gender': 1}
```

```
def get_features(threshold):
    # remove features below a certain number of appearances
    chosen_features = []
    for i in features_no_repeats:
        if count_best_features[i] > threshold:
#            print(count_best_features[i] ,'>', threshold,i)
            chosen_features.append(i)
    return chosen_features
```

```
chosen_features = get_features(4)
# chosen_features.remove('Avg_Open_To_Buy')
# chosen_features.remove('Avg_Utilization_Ratio')
chosen_features
```

```
['Total_Amt_Chng_Q4_Q1',
 'Total_Revolving_Bal',
 'Total_Ct_Chng_Q4_Q1',
 'Total_Trans_Ct',
 'Total_Trans_Amt']
```

Then, we train models with Feature Selection and tune parameters simultaneously in loops below.

```
def eval_model_threshold_10randomState(model, model_name, x, y, threshold):
    # make x the chosen subset
    chosen_features = get_features(threshold)
    x = x[chosen_features]

    accA=[]
    f1A=[]
    for k in range(1,11):
        train_x, test_x, train_y, test_y = train_test_split(x, y,
                                                            test_size = 0.20,
                                                            random_state = k)

        # fit model
        model.fit(train_x,train_y)
        model.score(test_x, test_y)
        pred_test = model.predict(test_x)

        # get metrics
        f1 = metrics.f1_score(test_y, pred_test)
        test_acc = metrics.accuracy_score(test_y, pred_test)
        con = metrics.confusion_matrix(test_y, pred_test)
        accA.append(test_acc)
        f1A.append(f1)

    accA = np.mean(accA)
    f1A = np.mean(f1A)

#    print(con,'%s:       %.4f F1-score       %.4f accuracy'%(model_name, f1, test_acc))
    print(model_name, '       %.4f F1-score'%f1A, '       %.4f accuracy'%accA)
```

```
model0 = tree.DecisionTreeClassifier(max_depth=8, min_samples_split=20, random_state=0)
model1 = RandomForestClassifier(random_state = 0)
model2 = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0,
                                    max_depth = 1, random_state = 0)
model3 = AdaBoostClassifier(n_estimators = 100, learning_rate = 1.0, random_state = 0)

# run ranges of possible thresholds
for i in range(0,5):
    eval_model_threshold_10randomState(model0, 'tree', x, y, i)

for i in range(0,5):
    eval_model_threshold_10randomState(model1, 'forest', x, y, i)

for i in range(0,5):
    eval_model_threshold_10randomState(model2, 'xGBoost', x, y, i)

for i in range(0,5):
    eval_model_threshold_10randomState(model3, 'AdaBoost', x, y, i)
```

```
tree        0.8214 F1-score        0.9447 accuracy
tree        0.8227 F1-score        0.9451 accuracy
tree        0.8214 F1-score        0.9444 accuracy
tree        0.8162 F1-score        0.9428 accuracy
tree        0.7859 F1-score        0.9347 accuracy
forest        0.8811 F1-score        0.9646 accuracy
forest        0.8834 F1-score        0.9652 accuracy
forest        0.8851 F1-score        0.9657 accuracy
forest        0.8763 F1-score        0.9627 accuracy
forest        0.8414 F1-score        0.9518 accuracy
xGBoost        0.8658 F1-score        0.9590 accuracy
xGBoost        0.8651 F1-score        0.9588 accuracy
xGBoost        0.8643 F1-score        0.9586 accuracy
xGBoost        0.8562 F1-score        0.9562 accuracy
xGBoost        0.8224 F1-score        0.9462 accuracy
AdaBoost        0.8723 F1-score        0.9609 accuracy
AdaBoost        0.8681 F1-score        0.9593 accuracy
AdaBoost        0.8680 F1-score        0.9593 accuracy
AdaBoost        0.8583 F1-score        0.9567 accuracy
AdaBoost        0.8202 F1-score        0.9454 accuracy
```

```
BDT = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2, random_state=0)
for i in range(0,5):
    eval_model_threshold_10randomState(BDT, 'tree', x, y, i)
```

```
tree        0.8069 F1-score        0.9394 accuracy
tree        0.8091 F1-score        0.9405 accuracy
tree        0.8084 F1-score        0.9402 accuracy
tree        0.7962 F1-score        0.9364 accuracy
tree        0.7678 F1-score        0.9275 accuracy
```

```
for theshold in range(0,5):
    for j in range(2,11):
        for i in range(1,19):
            minsplit = 5+(i*5)
            dt = tree.DecisionTreeClassifier(max_depth=j, min_samples_split=minsplit, random_state=0)
            name = 'threshold'+str(theshold)+'   depth'+str(j)+'   minsplit'+str(minsplit)
            eval_model_threshold_10randomState(dt, name, x, y, theshold)
            print('- - - - - - - - - - - - - - - - - - - - - - - - - - ')
        print('\n\n- - - - - - - - - - - - - - - - - - - - - - - - - - \n\n')
    print('\n\n\n\n- - - - - - - - - - - - theshold',theshold,'- - - - - - - - - - - - - - \n\n\n\n')
```

```
threshold0   depth2   minsplit10        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit15        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit20        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit25        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit30        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit35        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit40        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit45        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit50        0.5991 F1-score        0.8935 accuracy
- - - - - - - - - - - - - - - - - - - - - - - - - - 
threshold0   depth2   minsplit55        0.5991 F1-score        0.8935 accuracy
```

After finishing tuning parameters, we create models and test them using 10-fold cross validation which are shown below.

```
# 'threshold no' means whole preprocessed dataset
# 'threshold 2'  means not whole dataset(only features appearing > 2 times in feature selection)

# select columns
X = df.iloc[:,1:]
y = df.iloc[:,0]

# select different X
x_no_feature_selection = X
x_threshold1 = X[get_features(1)]
x_threshold2 = X[get_features(2)]

# split x y for traning models
train_x_noFS, test_x_noFS, train_y_noFS, test_y_noFS = train_test_split(x_no_feature_selection, y, test_size = 0.20, random_state =
train_x_th1, test_x_th1, train_y_th1, test_y_th1 = train_test_split(x_threshold1, y, test_size = 0.20, random_state = 0)
train_x_th2, test_x_th2, train_y_th2, test_y_th2 = train_test_split(x_threshold2, y, test_size = 0.20, random_state = 0)
```

```
# tree     threshold 1     base model              0.8091 F1-score         0.9405 accuracy
Btree = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2, random_state=0)
Btree.fit(train_x_th1,train_y_th1)

pred_test = Btree.predict(test_x_th1)
f1 = metrics.f1_score(test_y_th1, pred_test)
test_acc = metrics.accuracy_score(test_y_th1, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(Btree, x_threshold1, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

```
test score: f1=0.7865   acc=0.9344

10cv score:
f1 = 0.7439336130583534
acc = 0.9102324914452045
```

```
# tree     threshold 1    depth8  minsplit20      0.8227 F1-score         0.9451 accuracy
tree3 = tree.DecisionTreeClassifier(max_depth=8, min_samples_split=20, random_state=0)
tree3.fit(train_x_th1,train_y_th1)

pred_test = tree3.predict(test_x_th1)
f1 = metrics.f1_score(test_y_th1, pred_test)
test_acc = metrics.accuracy_score(test_y_th1, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(tree3, x_threshold1, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

```
test score: f1=0.8260   acc=0.9472

10cv score:
f1 = 0.7538307745373767
acc = 0.9137862920375046
```

```
# tree     threshold 1    depth9  minsplit25      0.8246 F1-score         0.9461 accuracy
tree4 = tree.DecisionTreeClassifier(max_depth=9, min_samples_split=25, random_state=0)
tree4.fit(train_x_th1,train_y_th1)

pred_test = tree4.predict(test_x_th1)
f1 = metrics.f1_score(test_y_th1, pred_test)
test_acc = metrics.accuracy_score(test_y_th1, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(tree4, x_threshold1, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

```
test score: f1=0.8013   acc=0.9418

10cv score:
f1 = 0.7574727831176593
acc = 0.9157618937995778
```

```
# forest   threshold 2   base model              0.8851 F1-score         0.9657 accuracy
forest2 = RandomForestClassifier(random_state=0)
forest2.fit(train_x_th2,train_y_th2)

pred_test = forest2.predict(test_x_th2)
f1 = metrics.f1_score(test_y_th2, pred_test)
test_acc = metrics.accuracy_score(test_y_th2, pred_test)
print('test score: f1=%.4f   acc=%.4f'%(f1,test_acc))

cv_results = cross_validate(forest2, x_threshold2, y, scoring = ('f1', 'accuracy'), cv = 10)
sorted(cv_results.keys())
print('\n10cv score:\nf1 =',np.mean(cv_results['test_f1']),'\nacc =',np.mean(cv_results['test_accuracy']))
```

```
test score: f1=0.8874   acc=0.9674

10cv score:
f1 = 0.8320315121531887
acc = 0.9483529336023006
```

Model	10cv F1-score	10cv Accuracy	Best threshold
Base DT	0.7349	0.9045	-
Base DT with FS	0.7439	0.9102	1
Parameter tuned DT	0.7539	0.9140	-
Parameter tuned DT with FS	0.7575	0.9158	1
Base RF	0.7942	0.9413	-
Base RF with FS	0.8320	0.9484	2

We found that the best threshold for both tuned and non-tuning Decision Trees is 1 and for Random Forest is 2. This experiment shows that Feature Selection improves performance of all models from base model to the most advanced one.

The best model of 3 experiments is Random Forest with Feature Selection threshold 2, that has F1-score at 0.8320 and Accuracy at 0.9484.

4.2 Naïve Bayes Classifier

The most obvious weak point for the naïve bayes classifier is happening to deal with an imbalance dataset. However, it is better to show results after trying to find the best combination of predictors for this classification.

4.2.1 Experiment1: All Categorical predictors in the original dataset

First experiment, we tried to use all categorical variables, which consisted of Gender, Education_Level, Marital_Status, Income_Category, and Card_Category, in the naïve bayes classifier. The result was expected to perform badly. The model was unable to detect attrited customers from all customers in the dataset. (Recall = 0 and F1 = 0) Thus, we needed to try out any combination of variables with other techniques to improve the recall score.

```
library(tidyverse)
library(caret)
library(klaR)
library(e1071)
library(arules)
library(rpart)
library(rpart.plot)

## Data Manipulation
churn <- read_csv("BankChurners.csv")
churn <- churn[,- c(1,22,23)]
glimpse(churn)

churn$Attrition_Flag <- factor(churn$Attrition_Flag)
churn$Gender <- factor(churn$Gender)
churn$Education_Level <- factor(churn$Education_Level)
churn$Marital_Status <- factor(churn$Marital_Status)
churn$Income_Category <- factor(churn$Income_Category)
churn$Card_Category <- factor(churn$Card_Category)
glimpse(churn)
```

```
###### Training Models
## NB originally discrete
nb <- naiveBayes(Attrition_Flag ~ Gender + Education_Level + Marital_Status + Income_Category + Card_Category,
                 data = churn)
nb
pred <- predict(nb, newdata = churn)
confusionMatrix(table(pred, churn$Attrition_Flag),
                mode = "prec_recall")
```

Data Preprocessing for the naïve bayes classification.

```
Confusion Matrix and Statistics


pred                   Attrited Customer Existing Customer
  Attrited Customer                    0                 0
  Existing Customer                 1627              8500

               Accuracy : 0.8393
                 95% CI : (0.832, 0.8464)
    No Information Rate : 0.8393
    P-Value [Acc > NIR] : 0.5066

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

              Precision :     NA
                 Recall : 0.0000
                     F1 :     NA
             Prevalence : 0.1607
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : Attrited Customer
```

Confusion Matrix and statistics for all categorical variables in the original dataset.

4.2.2 Experiment2: 10-fold Naïve Bayes Cross Validation with some Categorical Predictors

For the second experiment, we selected some categorical variables, Card_Category and Marital_Status, together with k-fold cross validation. The result was still the same as the previous experiment. (Recall = 0 and F1 = 0)

```
## NB w/ k-fold + 2 selected discrete values
ctrl <- trainControl(method = "cv",
                     number = 10)

set.seed(4042)
nb_caret <- train(Attrition_Flag ~ Card_Category + Marital_Status,
                  data = churn,
                  method = "nb",
                  trControl = ctrl)
nb_caret

pred_cv <- predict(nb_caret, newdata = churn)
confusionMatrix(table(pred_cv, churn$Attrition_Flag),
                mode = "prec_recall")
```

Setting for 10-fold naïve bayes cross validation with Card_Category and Marital_Status.

```
Confusion Matrix and Statistics


pred_cv              Attrited Customer Existing Customer
  Attrited Customer                  0                 0
  Existing Customer               1627              8500

               Accuracy : 0.8393
                 95% CI : (0.832, 0.8464)
    No Information Rate : 0.8393
    P-Value [Acc > NIR] : 0.5066

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

              Precision :     NA
                 Recall : 0.0000
                     F1 :     NA
             Prevalence : 0.1607
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : Attrited Customer
```

Confusion Matrix and statistics for the second experiment.

4.2.3 Experiment3: Applying EFD before training Naïve Bayes with all predictors

The third experiment began with discretizing all continuous variables with Equal-Frequency Discretization (EFD). Then, we used all discretized variables and originally categorical variables. The result was impressive for the fact that the model was able to classify attrited customers with a pretty good recall value at 0.6472 and F1 value at 0.6480. Hence, this showed that all continuous variables dropped in the first experiment were improving the performance of the model.

```
## Discretizing
dis_churn <- discretizeDF(churn)
glimpse(dis_churn)
```

Performing EFD with "arules" library in R.

```
Confusion Matrix and Statistics


pred_dis             Attrited Customer Existing Customer
  Attrited Customer               1053               570
  Existing Customer                574              7930

               Accuracy : 0.887
                 95% CI : (0.8807, 0.8931)
    No Information Rate : 0.8393
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.5807

 Mcnemar's Test P-Value : 0.9293

              Precision : 0.6488
                 Recall : 0.6472
                     F1 : 0.6480
             Prevalence : 0.1607
         Detection Rate : 0.1040
   Detection Prevalence : 0.1603
      Balanced Accuracy : 0.7901

       'Positive' Class : Attrited Customer
```

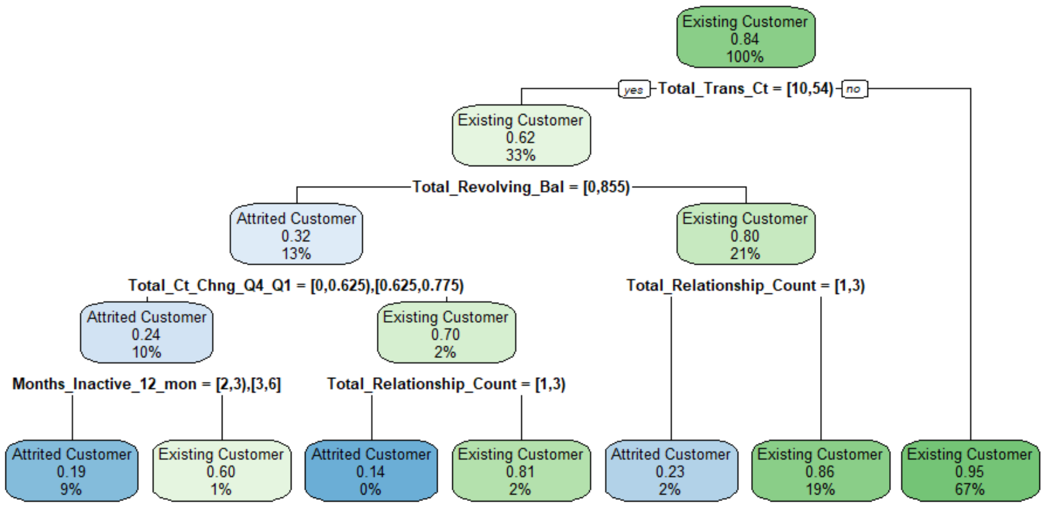
Confusion Matrix and statistics for the third experiment containing all variables after discretizing.

4.2.4 Experiment4: Feature Selection with Decision Tree in Discretized Dataset

The fourth experiment focused on feature selection with a decision tree model. Then, we used all variables in the decision tree model to construct a naïve bayes classifier. The overall performance was increasing. (Accuracy = 0.9062) However, the model's recall dropped to 0.5519 and F1 increased to 0.6540. It was not very good for identifying attrited customers. But it improved in identifying existing customers.

```
## feature selection w/ dt on discretized churn
dt_ctrl <- rpart.control(minsplit = 20,
                         xval = 10)
dt <- rpart(Attrition_Flag ~ .,
            data = dis_churn,
            control = dt_ctrl)
rpart.plot(dt)
printcp(dt)
```

Constructing a decision tree with "rpart" library for feature selection.



The Decision Tree used only Total_Trans_Ct, Total_Revolving_Bal, Total_Ct_Chng_Q4_Q1, Months_Inactive_12_mon, and Total_Relationship_Count for classification.

```
nb_dt <- naiveBayes(Attrition_Flag ~ Total_Trans_Ct + Total_Revolving_Bal + Total_Ct_Chng_Q4_Q1 + Months_Inactive_12_mon + Total_Relationship_Count,
                    data = dis_churn)
pred_nb_dt <- predict(nb_dt, newdata = dis_churn)
confusionMatrix(table(pred_nb_dt, dis_churn$Attrition_Flag),
                mode = "prec_recall")
```

Constructing a naïve bayes model with all features in decision tree criterias.

```
Confusion Matrix and Statistics


pred_nb_dt          Attrited Customer Existing Customer
  Attrited Customer               898               221
  Existing Customer               729              8279

               Accuracy : 0.9062
                 95% CI : (0.9003, 0.9118)
    No Information Rate : 0.8393
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6019

 Mcnemar's Test P-Value : < 2.2e-16

              Precision : 0.80250
                 Recall : 0.55194
                     F1 : 0.65404
             Prevalence : 0.16066
         Detection Rate : 0.08867
   Detection Prevalence : 0.11050
      Balanced Accuracy : 0.76297

       'Positive' Class : Attrited Customer
```

Confusion Matrix and statistics for the fourth experiment containing all variables in the decision tree.

From all experiments with naïve bayes classifiers, we will give the third and the fourth experiment to be the best model. Since this dataset is imbalanced, using accuracy makes us overlook the objective of this classification. Hence, it is better to rate the performance with recall and F1. The third experiment is good for detecting attrited customers. On the other hand, the fourth experiment is performing well in identifying both classes.

4.3 k-Nearest Neighbor

KNN (K-nearest neighbor algorithm) classification algorithm is a non-parametric leaning method. The advantages of the algorithm are its simple principle and few influencing factors but it also has many shortcomings, such as too much time consuming and difficulty in choosing K value.

Its basic idea is when entering new data of unknow category to be classified, classified, the category of the data to be classified should be determined according to the category of other samples. Firstly, the characteristics of the data to be classified should be extracted and compared with the characteristics of each known category data in the test set. Then, the nearest neighbor data of K should be taken from the test set to count the categories in which most of the data are located. Finally, the data to be classified should be classified into this category.

Propose Method: KNN Classifier Framework

Step 1: Select only quantitative predictors.
Step 2: Standardized variables.
Step 3: Split data set to train and test
Step 4: Hyper-parameter tuning.
Step 5: KNN classification using Euclidean distance.

4.3.1 Experiment1: Use All Quantitative predictors in the original dataset

First experiment, we use all quantitative variables, which consisted of Customer_Age, Credit_Limit, Total_Revolving_Bal, Avg_Open_To_Buy, Total_Amt_Chng_Q4_Q1, Total_Trans_Amt, Total_Trans_Ct, Total_Ct_Chng_Q4_Q1, Avg_Utilization_Ratio, Months_on_book and Predicting class of Attrition_Flag. In the experiment1 we get the best model with accuracy score is 0.9112 and f1-score is 0.6904 when k is equal to 5. The result is quite good performance but it can be better than by using combination techniques in data mining to improve the model.

```
use = ['Customer_Age','Credit_Limit','Total_Revolving_Bal',
     'Avg_Open_To_Buy','Total_Amt_Chng_Q4_Q1',
     'Total_Trans_Amt','Total_Trans_Ct','Total_Ct_Chng_Q4_Q1',
     'Avg_Utilization_Ratio','Months_on_book'] # all of quntitative predictors.
x = df[use]
y = df['Attrition_Flag']
# re-scal and prepare to model
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3,random_state = 42)
X_train = scaler.fit_transform(X_train)
X_test =  scaler.fit_transform(X_test)
Y_test = Y_test.map({'Attrited Customer': 1, 'Existing Customer': 0}).astype(int)
Y_train = Y_train.map({'Attrited Customer': 1, 'Existing Customer': 0}).astype(int)
```

Data Preprocessing for the KNN classification.

```
k = [1,3,5,7,9,11,13,15]
for i in range(len(k)):
    kNN1 = KNeighborsClassifier(n_neighbors = k[i],metric = 'euclidean')
    kNN1.fit(X_train,Y_train)
    y_pred = kNN1.predict(X_test)
    print('When k = %d      accuracy=%.4f     f1_score=%.4f'%(k[i],accuracy_score(Y_test,y_pred),f1_score(Y_test,y_pred))
```

Hyper-parameter tuning and using Euclidean distance.

```
When k = 1     accuracy=0.9069    f1_score=0.6967
When k = 3     accuracy=0.9092    f1_score=0.6913
When k = 5     accuracy=0.9112    f1_score=0.6904
When k = 7     accuracy=0.9085    f1_score=0.6782
When k = 9     accuracy=0.9065    f1_score=0.6674
When k = 11     accuracy=0.9062    f1_score=0.6635
When k = 13     accuracy=0.9085    f1_score=0.6698
When k = 15     accuracy=0.9072    f1_score=0.6602
```

Confusion Matrix for all quantitative variables in the original dataset.

```
kNN1
```

```
KNeighborsClassifier

KNeighborsClassifier(algorithm='auto', leaf_size=30, metr
ic='euclidean',
                     metric_params=None, n_jobs=None, n_neig
hbors=15, p=1,
                     weights='uniform')
```

KNN seting for the frist experiment.

4.3.2 Experiment2: Using Feature Selection Techniques.

From the result in part 4.1.4, we use feature selection techniques RF+Ada+XG to get important feature. So, the quantitative variable in used as the same 4.3.1 except we don't use Months_on_book to be predictor.In the experiment2 we get the best model with accuracy score is 0.9220 and f1-score is 0.7310 when k is equal to 5. The result is impovre from 4.3.1 manifestly.

```
use = ['Customer_Age','Credit_Limit','Total_Revolving_Bal',
       'Avg_Open_To_Buy','Total_Amt_Chng_Q4_Q1',
       'Total_Trans_Amt','Total_Trans_Ct','Total_Ct_Chng_Q4_Q1',
       'Avg_Utilization_Ratio',]
x = df[use]
y = df['Attrition_Flag']
```

Select the predictors from feature selection and continuous data preprocessing

```
# model 2 use feature selection
k = [1,3,5,7,9,11,13,15]
for i in range(len(k)):
    kNN2 = KNeighborsClassifier(n_neighbors = k[i],metric = 'euclidean')
    kNN2.fit(X_train,Y_train)
    y_pred = kNN2.predict(X_test)
    print('When k = %d     accuracy=%.4f     f1_score=%.4f'%(k[i],accuracy_score(Y_test,y_pred),f1_score(Y_test,y_pred)))
```

```
When k = 1    accuracy=0.9098    f1_score=0.7116
When k = 3    accuracy=0.9194    f1_score=0.7281
When k = 5    accuracy=0.9220    f1_score=0.7310
When k = 7    accuracy=0.9167    f1_score=0.7102
When k = 9    accuracy=0.9161    f1_score=0.7038
When k = 11    accuracy=0.9138    f1_score=0.6946
When k = 13    accuracy=0.9128    f1_score=0.6908
When k = 15    accuracy=0.9121    f1_score=0.6870
```

Confusion Matrix for the second experiment.

```
1  kNN2
```

```
KNeighborsClassifier

KNeighborsClassifier(algorithm='auto', leaf_size=30, metr
ic='euclidean',
                     metric_params=None, n_jobs=None, n_neig
hbors=15, p=1,
                     weights='uniform')
```

KNN seting for the second experiment.

4.3.3 Experiment3: Using K-fold and Random Grid Search

From the result 4.3.2, we decide to improve the model by using K-fold and Random Grid Search from library Pycaret. So, we use k is equal 5 and same predictor from second experiment and set up data preprocessing as all experiment. Thus, we get mean of accuracy is 0.9224 and mean of f1-score is 0.9547 after 10-fold and tune model by using randomgrid search.The result are impovre from all experimant obviously.

```
In [86]: s = setup(data_py,target ='Attrition_Flag',normalize = True)
```

	Description	Value
0	Session id	5312
1	Target	Attrition_Flag
2	Target type	Binary
3	Target mapping	Attrited Customer: 0, Existing Customer: 1
4	Original data shape	(10127, 10)
5	Transformed data shape	(10127, 10)
6	Transformed train set shape	(7088, 10)
7	Transformed test set shape	(3039, 10)
8	Numeric features	9
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	constant
13	Low variance threshold	0
14	Normalize	True

Seting of model using Pycaret.

```
In [105]:  1  knn = create_model('knn',n_neighbors=5,metric = 'euclidean',p=1)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9013	0.9165	0.9479	0.9353	0.9416	0.6235	0.6241
1	0.9196	0.9242	0.9613	0.9439	0.9525	0.6900	0.6912
2	0.9097	0.9166	0.9782	0.9194	0.9479	0.6134	0.6303
3	0.9182	0.9327	0.9765	0.9296	0.9525	0.6608	0.6712
4	0.9111	0.9074	0.9731	0.9249	0.9484	0.6301	0.6408
5	0.9168	0.9282	0.9681	0.9351	0.9513	0.6668	0.6716
6	0.9168	0.9241	0.9681	0.9351	0.9513	0.6668	0.6716
7	0.9055	0.9101	0.9664	0.9244	0.9449	0.6128	0.6204
8	0.9181	0.9253	0.9748	0.9310	0.9524	0.6606	0.6696
9	0.9322	0.9514	0.9747	0.9461	0.9602	0.7320	0.7357
Mean	0.9149	0.9237	0.9689	0.9325	0.9503	0.6557	0.6626
Std	0.0082	0.0119	0.0086	0.0080	0.0048	0.0355	0.0335

Confusion Matrix of 10-fold with the same set up for the third experiment.

```
1  tuned_knn = tune_model(knn)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9097	0.9465	0.9613	0.9331	0.9470	0.6427	0.6460
1	0.9295	0.9461	0.9681	0.9489	0.9584	0.7271	0.7286
2	0.9182	0.9529	0.9765	0.9296	0.9525	0.6608	0.6712
3	0.9379	0.9538	0.9882	0.9408	0.9639	0.7427	0.7544
4	0.9140	0.9343	0.9748	0.9265	0.9500	0.6418	0.6527
5	0.9196	0.9472	0.9765	0.9311	0.9532	0.6680	0.6777
6	0.9210	0.9606	0.9681	0.9396	0.9536	0.6874	0.6909
7	0.9083	0.9490	0.9697	0.9247	0.9467	0.6214	0.6304
8	0.9223	0.9438	0.9782	0.9327	0.9549	0.6768	0.6869
9	0.9435	0.9611	0.9865	0.9482	0.9670	0.7715	0.7788
Mean	0.9224	0.9495	0.9748	0.9355	0.9547	0.6840	0.6918
Std	0.0109	0.0076	0.0080	0.0081	0.0064	0.0461	0.0457

Confusion Matrix after tune model by using Random Grid Search

```
KNeighborsClassifier
KNeighborsClassifier(algorithm='auto', leaf_size=30, metr
ic='euclidean',
                     metric_params=None, n_jobs=-1, n_neighbo
rs=5, p=1,
                     weights='uniform')
```

KNN seting for the third experiment.

From all experiments with KNN classifiers, we will give the third experiment to be the best model.With, KNN using euclidean distance with k=5 after feature selection and tune model give highest accuracy = 0.9224 and F1-score =0.9547, Hence, it prove it that feature selection and tune model by using random grid search are effective.

5. Conclusion and Discussion

From 3 experiments in the Decision Tree section, the best model is Random Forest with Feature Selection threshold 2, which has Accuracy at 0.9484 and F1-score at 0.8320. Compared to the non-tuning parameter Decision Tree that has Accuracy at 0.9045 and F1-score at 0.7349, we are safe to say that our ideas do improve model performance.

From 4 experiments in the Naïve Bayes section, the best two model are applying EFD before training Naïve Bayes with all predictors with Accuracy at 0.887, Recall at 0.6472, and F1 at 0.6480 for identifying attrited customers and feature selection with decision tree in Discretized Dataset before training with Naïve Bayes classifier with Accuracy at 0.9062, Recall at 0.5519, and F1 at 0.6540 for overall performance. However, various combinations of predictors can still be used to improve prediction. We recommend trying different sets of predictors and techniques to improve the performance of the model.

From 3 experiments in the KNN section, the best model is KNN using Feature Selction and tune hyperameter. Which has Accuracy at 0.9224 and F1-score at 0.9547, from all of experiment it prove it that feature selection and tune model by using random grid search are effective.

Lastly, the best model of all 3 sections is Random Forest with Feature Selection threshold 2. We guess the reason that Random Forest performs the best is that it is strong with imbalanced data, has a lot of Decision Trees to aggregate the final output, and is flexible to use both quantitative and categorical features.

References

https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers

https://www.kaggle.com/code/rebeccapringle/bank-churners-96-6-accuracy#Import-Libraries

https://www.kaggle.com/code/thomaskonstantin/bank-churn-data-exploration-and-churn-prediction

https://stackoverflow.com/questions/55591063/how-to-perform-smote-with-cross-validation-in-sklearn-in-python

https://pycaret.gitbook.io/docs/get-started/functions/optimize

https://stackoverflow.com/questions/65588010/how-to-determine-number-of-neighbors-in-knn-in-pycaret?fbclid=IwAR0a7Q2Xug-iskfwTcgcaSBi-UltqyH5id3dODMgZ8na8LeLBNbhTT3zPhc

Huang, Jie, et al. "An improved knn based on class contribution and feature weighting." *2018 10th international conference on measuring technology and mechatronics automation (ICMTMA)*. IEEE, 2018.

SCMA447/SCIM323 Data Mining Lecture Slides. *(Data Visualization, kNN, Classification Tree, and Naive Bayes.)* Semester 02/2565. By Asst. Prof. Pannapa Changpetch, Department of Mathematics, Faculty of Science, Mahidol Unversity.

Member responsibility

Thithiwat Worapraphinchai 6205046

- Decision Tree

Akepawin Pornserikul 6205071

- Naïve Bayes Classifier

Metee Baopimpa 6205263

- k-Nearest Neighbor

All 3 members have contributions in chapter Introduction, Dataset overview, Data visualization, Conclusion and Discussion.