Classification

Tan Wen Tao Bryan
2214449
DAAA/FT/2A/01

Project Objective

- To build a classification model to predict which customer will have default payment in the next month from the perspective of risk management

Step 1: EDA

```
print(credit_ds.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600 entries, 0 to 1599
Data columns (total 13 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Customer ID                 1600 non-null   int64
 1   Credit Limit                1600 non-null   int64
 2   Gender                      1600 non-null   object
 3   Education                   1600 non-null   object
 4   Marriage Status             1600 non-null   object
 5   Age                         1600 non-null   int64
 6   Bill_Amount1                1600 non-null   int64
 7   Bill_Amount2                1600 non-null   int64
 8   Bill_Amount3                1600 non-null   int64
 9   Pay_Amount1                 1600 non-null   int64
 10  Pay_Amount2                 1600 non-null   int64
 11  Pay_Amount3                 1600 non-null   int64
 12  Default payment next month  1600 non-null   int64
dtypes: int64(10), object(3)
memory usage: 162.6+ KB
None
```

Observations:

- 1600 rows and 13 columns
- 3 categorical columns, 10 numerical columns
- All columns has an equal number of observations
- Customer_ID has a unique value for every row so it can be removed later
- Default payment next month is the target variable as it fulfils the project objective

Unique Values for Columns with Categorical Data

```
#cols with categorical data
categorical_col = credit_ds.dtypes[credit_ds.dtypes=='object']
print(categorical_col)
print()

#Change spelling error in data to university
credit_ds["Education"] = credit_ds["Education"].replace("univeresity", "university")

#prints unique categorical values
for cat_val in categorical_col.index:
    print(f'{cat_val} - {credit_ds[cat_val].unique()}')
```

```
Gender             object
Education          object
Marriage Status    object
dtype: object

Gender - ['female' 'male']
Education - ['university' 'post-graduate' 'high school']
Marriage Status - ['married' 'single']
```

Observations:

- Have to encode these categorical data later in feature engineering
- Spelling error with 'univeresity' in Education column, have to clean that data too

Missing Values

```
#check for missing values in the dataframe
print(credit_ds.isna().sum().sort_values())
```

```
Customer ID                 0
Credit Limit                0
Gender                      0
Education                   0
Marriage Status             0
Age                         0
Bill_Amount1                0
Bill_Amount2                0
Bill_Amount3                0
Pay_Amount1                 0
Pay_Amount2                 0
Pay_Amount3                 0
Default payment next month  0
dtype: int64
```

Observations:

- No missing values for the dataset so there is no need for imputation

Step 1: EDA (Target Distribution)

Distribution of Target Variable

```
#Temporarily turns data of target variable to categorical value
##Check whether data is balanced by calculating percentage of freq
credit_ds["Default payment next month"] = credit_ds["Default payment next month"].replace(0, "Non default").replace(1, "Default")
print(f'{"Default payment next month"} - {credit_ds["Default payment next month"].unique()}')
```

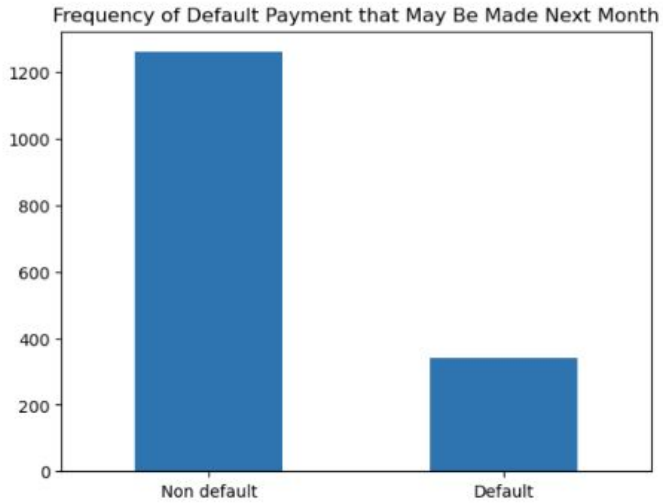
```
Default payment next month - ['Default' 'Non default']
```

```
#Descriptive Stats
cred_stats=credit_ds.describe(include="all").T


cred_stats["percentage of most freq val"] = cred_stats["freq"]/len(credit_ds)*100
display(cred_stats)
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	percentage of most freq val
Customer ID	1600.0	NaN	NaN	NaN	800.5	462.02453	1.0	400.75	800.5	1200.25	1600.0	NaN
Credit Limit	1600.0	NaN	NaN	NaN	166787.5	129944.635707	10000.0	50000.0	140000.0	230000.0	700000.0	NaN
Gender	1600	2	female	940	NaN	NaN	NaN	NaN	NaN	NaN	NaN	58.75
Education	1600	3	university	718	NaN	NaN	NaN	NaN	NaN	NaN	NaN	44.875
Marriage Status	1600	2	single	899	NaN	NaN	NaN	NaN	NaN	NaN	NaN	56.1875
Age	1600.0	NaN	NaN	NaN	35.224375	9.40455	21.0	28.0	34.0	41.0	75.0	NaN
Bill_Amount1	1600.0	NaN	NaN	NaN	49263.408125	72687.106917	-14386.0	3138.0	20320.5	61602.75	507726.0	NaN
Bill_Amount2	1600.0	NaN	NaN	NaN	47726.644375	71476.217397	-13543.0	3288.25	19769.5	59891.75	509229.0	NaN
Bill_Amount3	1600.0	NaN	NaN	NaN	44531.051875	67164.064239	-9850.0	2321.5	19332.5	53857.25	499936.0	NaN
Pay_Amount1	1600.0	NaN	NaN	NaN	5547.64625	14085.471957	0.0	1000.0	2176.5	5021.0	239104.0	NaN
Pay_Amount2	1600.0	NaN	NaN	NaN	5230.494375	15569.605419	0.0	468.5	1904.0	4592.5	285138.0	NaN
Pay_Amount3	1600.0	NaN	NaN	NaN	4451.610625	12491.715387	0.0	226.5	1313.5	4000.0	222750.0	NaN
Default payment next month	1600	2	Non default	1260	NaN	NaN	NaN	NaN	NaN	NaN	NaN	78.75

Observations:
- Non-default value occurred 78.75% the time which shows that the default payment next month of the customers is not balanced
- Negative values can be seen on Bill_Amount which shows the balance credited in the account
- Suggests either bill was offset or overpaid by customer
- Customer credit card bill amount was the greatest 1 month ago and the least 3 months ago based on mean value
- Amount customer paid was the greatest 1 month ago and the least 3 months ago based on mean value

```
#plot a barplot to show comparison of target variable frequency
ax1 = credit_ds["Default payment next month"].value_counts().plot(
    kind='bar', title="Frequency of Default Payment that May Be Made Next Month", rot=0
)
```

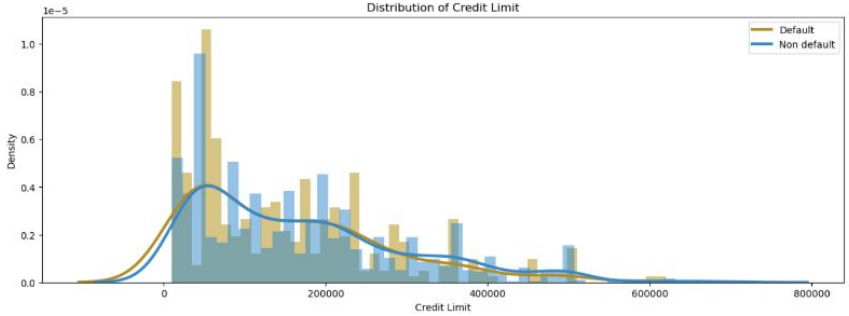


Observations:
- Barplot further proves that the target variable is unbalanced

Step 1: EDA (Numerical Variables)

Credit Limit

```
#Credit Limit Distribution
plt.figure(figsize=(15,5))
plt.hist(
    credit_df[credit_df["Default payment next month"] == 1]["Credit Limit"],
    density = True,
    bins = 50,
    color = '#b5890080'
)
plt.hist(
    credit_df[credit_df["Default payment next month"] == 0]["Credit Limit"],
    density = True,
    bins = 50,
    color='#268bd280'
)
sns.kdeplot(credit_df[credit_df["Default payment next month"] == 1]["Credit Limit"], lw=3, color='#b58900')
sns.kdeplot(credit_df[credit_df["Default payment next month"] == 0]["Credit Limit"], lw=3, color='#268bd2')
plt.legend(np.unique(credit_ds["Default payment next month"]), loc="upper right")
plt.title("Distribution of Credit Limit")
plt.show()
```

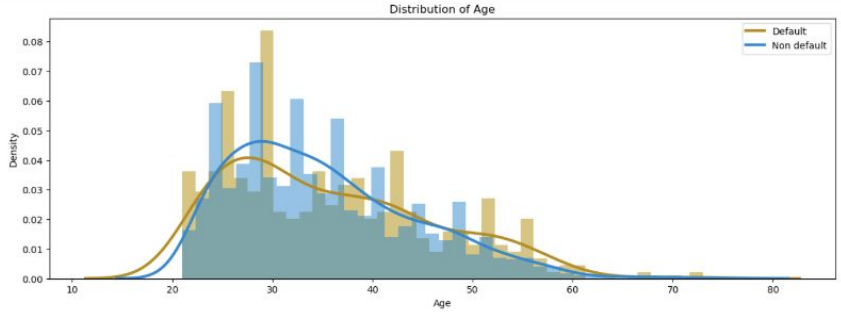


Observations:

- Distribution is skewed to the left, more customers with low credit limit than high credit limit
- Not much difference between data wiith customers that will default and not default payment next month

Age

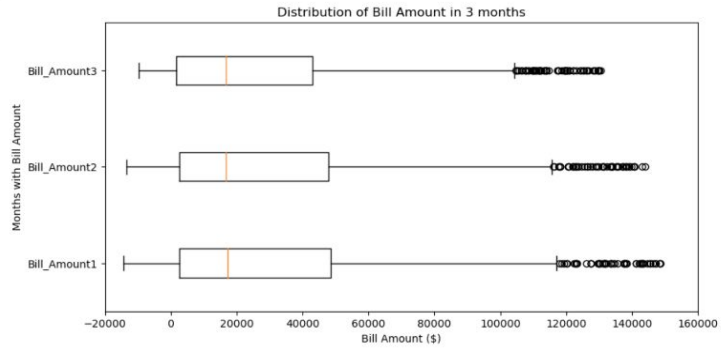
```
#Age Distribution
plt.figure(figsize=(15,5))
plt.hist(
    credit_df[credit_df["Default payment next month"] == 1]["Age"],
    density = True,
    bins = 40,
    color = '#b5890080'
)
plt.hist(
    credit_df[credit_df["Default payment next month"] == 0]["Age"],
    density = True,
    bins = 40,
    color='#268bd280'
)
sns.kdeplot(credit_df[credit_df["Default payment next month"] == 1]["Age"], lw=3, color='#b58900')
sns.kdeplot(credit_df[credit_df["Default payment next month"] == 0]["Age"], lw=3, color='#268bd2')
plt.legend(np.unique(credit_ds["Default payment next month"]), loc="upper right")
plt.title("Distribution of Age")
plt.show()
```



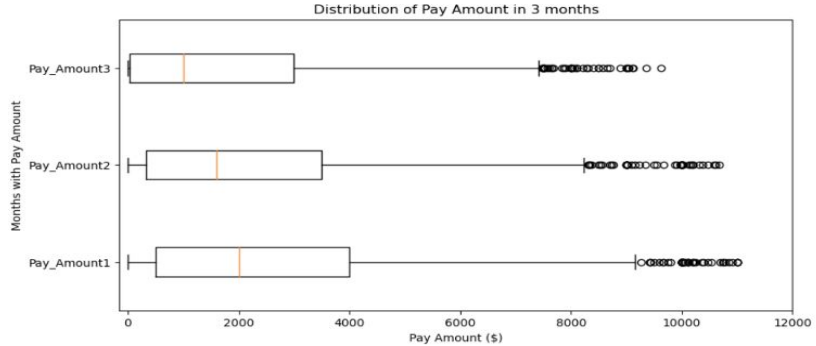
Observations:

- Distribution is skewed to the left
- For both customers who default and don't default payment, age of customers peaked at a range from 27-30 years old

Step 1: EDA (Numerical Variables)



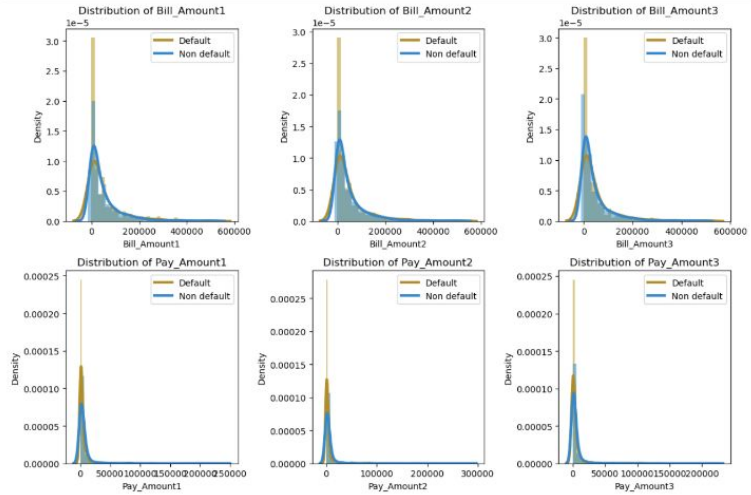
Observations:
- Distribution is extremely skewed to the left
- Median of bill amounts in 3 months are the same which shows that the bills are equal in amount
- IQR of Bill_Amount3 is slightly smaller than the others while IQR of Bill_Amount1 and Bill_Amount2 which shows that the bill amount in the 3rd month is less spread out than the 1st and 2nd month



Observations:
- Distribution is extremely skewed to the left
- Median of pay amount decreases in each month which shows that the amount paid is getting lesser in each month
- IQR of pay amount decreases within every month which shows that the total amount paid is getting more consistent in each month

Bill_Amount & Pay_Amount Distribution

```
#Bill_Amount & Pay_Amount Distribution
columns=["Bill_Amount1","Bill_Amount2","Bill_Amount3","Pay_Amount1","Pay_Amount2","Pay_Amount3"]
fig, axs = plt.subplots(2,3, figsize=(12, 8))
axs=axs.flatten()

for i, col in enumerate(columns):
    ax=axs[i]
    ax.hist(
    credit_df[credit_df["Default payment next month"] == 1][col],
    density = True,
    bins = 35,
    color = '#b5890080'

    )
    ax.hist(
        credit_df[credit_df["Default payment next month"] == 0][col],
        density = True,
        bins = 35,
        color='#268bd280'
    )
    sns.kdeplot(credit_df[credit_df["Default payment next month"] == 1][col], lw=3, color='#b58900', ax=ax)
    sns.kdeplot(credit_df[credit_df["Default payment next month"] == 0][col], lw=3, color='#268bd2', ax=ax)
    ax.legend(np.unique(credit_ds["Default payment next month"]), loc="upper right")
    ax.set_title(f'Distribution of {col}')

plt.tight_layout()
plt.show()
```



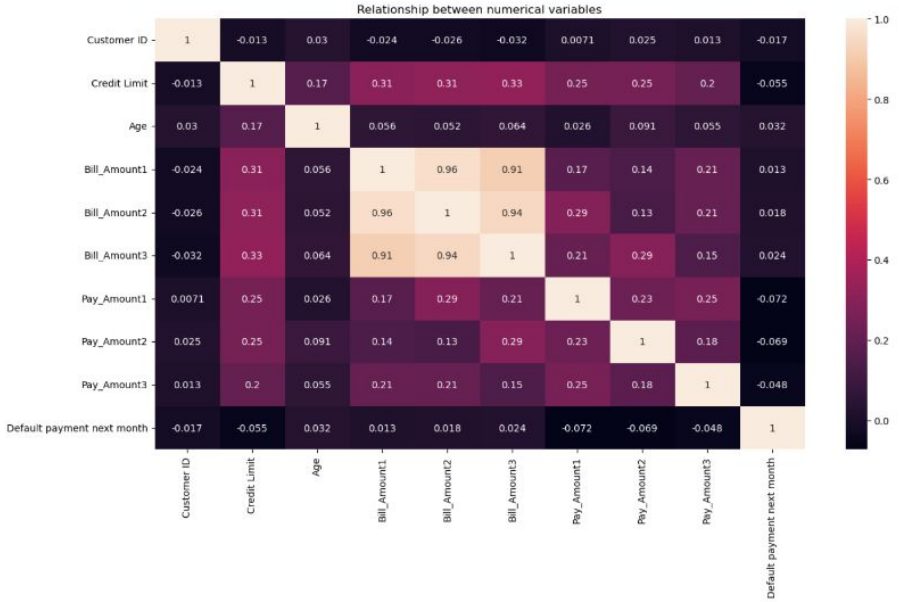
Observations:
- Distribution is very skewed to the left
- Larger amount of customers with below 100000 dollars as bill amount
- Larger amount of customers paid amounts below 50000 dollars
- No obvious pattern to determine distribution

Step 1: EDA (Bivariate Relationship)

Numerical Variables (Relationship)

```
plt.figure(figsize=(15,8))
sns.heatmap(credit_df.corr(), annot=True)
plt.title("Relationship between numerical variables")
plt.show()
```

```
C:\Users\bryan\AppData\Local\Temp\ipykernel_29324\3865247049.py:2: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  sns.heatmap(credit_df.corr(), annot=True)
```



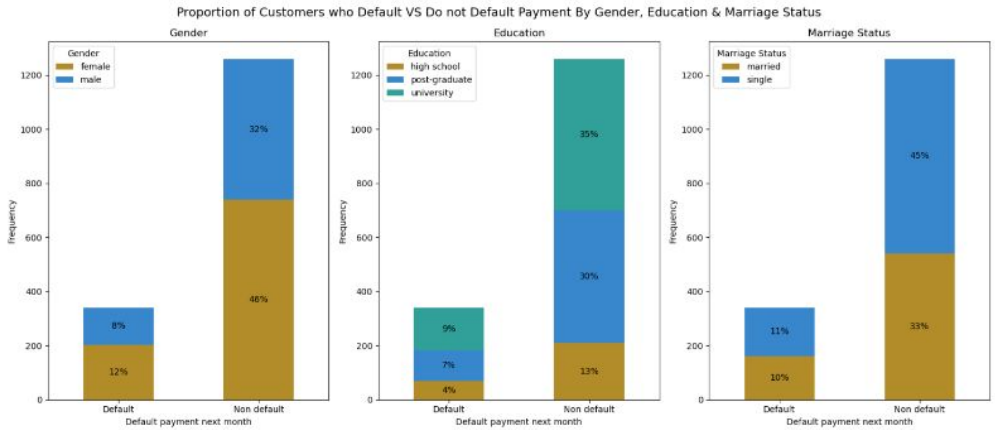
Observations:

- The 3 Bill_Amounts have a very strong correlation with one another, might possibly mean they are the cumulative amount in each month
- Bill Amount & Pay Amounty - Rest of the variables have weak relationship with one another

Step 1: EDA (Categorical Variables)

Gender, Marriage Status & Education

```
#Re-extract unique values from Default payment next month column
unique_payment = np.unique(credit_ds['Default payment next month'])

#Group data by default payment status
grouped= credit_ds.groupby(['Default payment next month'])

#Create subplots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(16,7))

#Calculate total count of customers who default and do not default payment
total_count = credit_ds["Default payment next month"].value_counts()

#1st subplot: Proportion of customers genders who default payment VS do not default
gender_count=grouped["Gender"].value_counts().unstack()
gender_count.plot(kind='bar', stacked=True, ax=axes[0], rot=0, color=['#b58900','#268bd2'])
axes[0].set_title('Gender')
axes[0].set_xlabel('Default payment next month')
axes[0].set_ylabel('Frequency')

#Add percentage labels
for p in axes[0].patches:
    width, height = p.get_width(), p.get_height()
    x, y=p.get_xy()
    axes[0].annotate(f'{int(height/total_count.sum() * 100)}%',
                     (x+width/2, y+height/2), ha='center', va='center')

#2nd subplot: Proportion of customers education who default VS do not default payment
education_count=grouped["Education"].value_counts().unstack()
education_count.plot(kind='bar', stacked=True, ax=axes[1], rot=0, color=['#b58900','#268bd2','#2aa198'])
axes[1].set_title('Education')
axes[1].set_xlabel('Default payment next month')
axes[1].set_ylabel('Frequency')

#Add percentage labels
for p in axes[1].patches:
    width, height = p.get_width(), p.get_height()
    x, y=p.get_xy()
    axes[1].annotate(f'{int(height/total_count.sum() * 100)}%',
                     (x+width/2, y+height/2), ha='center', va='center')

#3rd subplot: Proportion of customers marriage status who default VS do not default payment
marriage_count=grouped["Marriage Status"].value_counts().unstack()
marriage_count.plot(kind="bar", stacked=True, ax=axes[2], rot=0, color=['#b58900','#268bd2'])
axes[2].set_title('Marriage Status')
axes[2].set_xlabel('Default payment next month')
axes[2].set_ylabel('Frequency')

#Add percentage labels
for p in axes[2].patches:
    width, height = p.get_width(), p.get_height()
    x, y=p.get_xy()
    axes[2].annotate(f'{int(height/total_count.sum() * 100)}%',
                     (x+width/2, y+height/2), ha='center', va='center')

fig.suptitle("Proportion of Customers Who Default VS Do not Default Payment By Gender, Education & Marriage Status", fontsize=14)
plt.tight_layout()
plt.show()
```



Observations:

- For each category, the proportion of customers who do not default payment are much greater than those who default payment
- For Gender, proportion of female customers is much larger than the male customers for both default and non default payment
- For Education, proportion of customers in university is largest, followed by post-graduate then high school for both default and non-default payment
- For Marriage Status, proportion of single customers is larger than married customers for both default and non default payment

Step 2: Data Cleaning/Feature Engineering

```
#Change spelling error in data to university
credit_df["Education"] = credit_df["Education"].replace("univeresity", "university")
print(credit_df["Education"].unique())
```

```
['university' 'post-graduate' 'high school']
```

Drop Unwanted Columns

- Drop Customer ID as it is unique for every row

```
credit_df.drop("Customer ID", axis=1, inplace=True)
display(credit_df.head())
```

	Credit Limit	Gender	Education	Marriage Status	Age	Bill_Amount1	Bill_Amount2	Bill_Amount3	Pay_Amount1	Pay_Amount2	Pay_Amount3	Default payment next month
0	20000	female	university	married	24	3913	3102	689	0	689	0	1
1	120000	female	university	single	26	2682	1725	2682	0	1000	1000	1
2	90000	female	university	single	34	29239	14027	13559	1518	1500	1000	0
3	50000	female	university	married	37	46990	48233	49291	2000	2019	1200	0
4	50000	male	university	married	57	8617	5670	35835	2000	36681	10000	0

Separate Features & Target labels

```
X, y = credit_df.drop("Default payment next month", axis=1), credit_df["Default payment next month"]
```

Split Testing & Training data

- Split data 80/20 training/testing data to train and test the model
- Stratify parameter will maintain the distribution of sample values

```
#Split test and training data for features and target label
#random_state sets a seed to the random generator
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, shuffle=True, random_state=42)
print(X_train.shape)
print(X_test.shape)

X_trainSubset = X_train.copy()
```

```
(1280, 11)
(320, 11)
```

- Stratify retains the same proposition of class in the train and test datasets

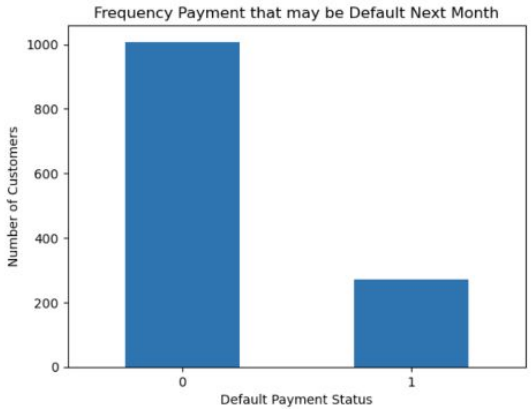
Step 2: Feature Engineering

Handling Imbalanced Data

Mentioned before in the Data Exploration portion, the target data is imbalanced and we shall use y_train data to visualise this distribution again

BEFORE

```
#plot a barplot to show comparison of target variable frequency
ax1 = y_train.value_counts().plot(
    kind='bar',
    title="Frequency Payment that may be Default Next Month",
    rot=0,
    xlabel='Default Payment Status',
    ylabel='Number of Customers'
)
```



- For me, using the Random Resampling technique from IMBLEARN will allow each customer to have an equal chance of being picked, solving the issue of imbalanced data

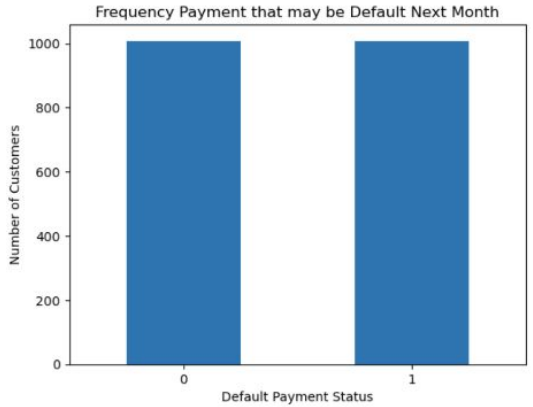
```
x_train_resampled, y_train_resampled = RandomOverSampler(random_state=15).fit_resample(X_trainSubset, y_train)
x_train_resampled
```

	Credit Limit	Gender	Education	Marriage Status	Age	Bill_Amount1	Bill_Amount2	Bill_Amount3	Pay_Amount1	Pay_Amount2	Pay_Amount3
0	30000	1.0	1.0	1.0	24	27391	28412	28575	1767	1700	1010
1	200000	0.0	1.0	0.0	38	104978	105924	102663	4500	0	3500
2	360000	1.0	0.0	1.0	29	3280	0	0	0	0	0
3	380000	0.0	0.0	1.0	43	8201	7882	-2	39	0	0
4	130000	0.0	2.0	1.0	43	1018	6377	-2	6433	2	937
...	...	...	...	...	...	...	...	...	...	...	...
2011	160000	1.0	0.0	1.0	29	160432	163916	159676	7500	0	11558
2012	20000	1.0	1.0	0.0	35	18648	18061	21034	0	3292	0
2013	500000	1.0	2.0	0.0	28	22848	23638	18878	1516	1300	1000
2014	50000	1.0	0.0	1.0	24	36166	37188	37680	1900	1400	1700
2015	230000	0.0	1.0	1.0	32	189567	189023	187521	9000	6300	6200

2016 rows × 11 columns

AFTER

```
#plot a barplot to show comparison of target variable frequency
ax1 = y_train_resampled.value_counts().plot(
    kind='bar',
    title="Frequency Payment that may be Default Next Month",
    rot=0,
    xlabel='Default Payment Status',
    ylabel='Number of Customers'
)
```



Step 2: Feature Engineering (Pipeline)

Tried **Discretisation of Age** and **creating new columns** but they **worsened my score** so were not included

View the Jupyter File for **more info**

Pipeline

- To link the steps of data engineering and the model implementation together

```
#Categorical transformer
categoricalTransformer = ColumnTransformer([
        ("oneHotEncoding", OneHotEncoder(sparse_output=False, categories="auto"),['Gender','Marriage Status']),
        ("ordinalEncoding", OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value=-1),['Education'])
    ],
    remainder="passthrough")

#Numerical transformer
num_cols = ['Credit Limit', 'Age', 'Bill_Amount1', 'Bill_Amount2',
            'Bill_Amount3', 'Pay_Amount1','Pay_Amount2', 'Pay_Amount3']

numericalTransformer = ColumnTransformer([
    ("standardise", StandardScaler(), num_cols)
],
remainder="passthrough")

#Combining two transformers into preprocessor
preprocessor=ColumnTransformer([
    ("categorical", categoricalTransformer, ["Gender", "Marriage Status", "Education"]),
    ("numerical", numericalTransformer, num_cols)
],
remainder="passthrough")

#OverSampling transformer
overSamplingTransformer = RandomOverSampler(random_state=42)

#Building the steps
steps = [
    ("preprocessing", preprocessor),
    ("oversampling", overSamplingTransformer),
    ("model")
]

model_step=len(steps)-1
```

OneHotEncode: Gender, Marriage Status
OrdinalEncode: Education
StandardScaler: All numerical variables
Oversampling

Step 3: Model Selection

- Tried out 7 models and chose the **top 3** based on **F2 score** and **another 1** model based on its **highest balanced accuracy** score

Model Selection

Here are some reasons and benefits of why I have selected these models for classification.

Logistic Regression

- Measures the probability of whether the customer default payment using a sigmoid function
- Predicted y-value lies below threshold y=0 and lie above threshold y=1 to show whether payment is default or not
- Easy model to implement

Naive Bayes

- Treat each feature independently and uses Bayes Theorem to find out how the target variable is classified

KNN

- Uses the value of the nearest data points to find the closest data points in training set
- Forecast the label of the test data based on the labels of the closest neighbours in training set

SVM

- Finds the boundary that separates the classes as wide a margin as possible
- Uses that hyperplane to separate the classes of the test data set
- Easy model to implement

Decision Tree

- Predict the customers who default and do not default payment by subsetting data into smaller groups based on different attributes (ex. gender, age, marital_status)

Random Forest

- Build multiple decision trees on different subsets of input features and training data and aggregate predictions of individual trees to make a final prediction
- Reduces risk of overfitting
- Robust to noisy data, irrelevant features

Gradient Boosting

- Ensemble model that combines several weak models to create a strong model (Prediction based on aggregation of all predictions of all trees in the sequence)

```
#Defining the models used with default hyperparameters
models = [('Logistic Regression', LogisticRegression(random_state=42)),
          ('Naive Bayes', GaussianNB()),
          ('KNN', KNeighborsClassifier()),
          ('SVM', SVC(random_state=42)),
          ('Decision Tree', DecisionTreeClassifier(random_state=42)),
          ('Random Forest', RandomForestClassifier(random_state=42)),
          ('Gradient Boosting', GradientBoostingClassifier(random_state=42))
         ]
```

Step 4: Model Evaluation

```
#Use make_scorer to make own score, beta is given as value 2

#Store scoring metrics
f2 = make_scorer(fbeta_score, beta =2)
score_metrics={
    'f2': f2,
    'balanced_accuracy':'balanced_accuracy',
    'roc_auc':'roc_auc'
}
```

- Prioritise F2 score & Balanced Accuracy

F2 Score

- Variation of F1 Score
- Weighted harmonic mean of the precision and recall
- Gives more weight to recall instead of precision
- Beta determines the weight of recall
  - Higher Beta gives more weight to recall

$$F2 = \frac{(1+\beta^2)\cdot(precision\cdot recall)}{(\beta^2\cdot precision)+recall}$$

Balanced Accuracy

- Suits the imbalanced testing data

$$Sensitivity = Recall = \frac{TP}{TP+FN}$$

$$Specificity = \frac{TN}{FP+TN}$$

$$BalancedAcc = \frac{Sensitivity + Specificity}{2}$$

ROC_AUC

- Area under the ROC helps us observe proportion of the classification

True Positive (TP)

- Customer default payment in the next month and prediction is positive

True Negative (TN)

- Customer does not default payment in the next month and prediction is negative

False Positive (FP)

- Customer does not default payment in the next month but prediction is positive

False Negative (FN)

- Customer default payment in the next month but prediction is negative

Terms below can also help with understanding the formulas on top

Accuracy

- Shows the percentage of correct predictions model can make

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Recall

- Shows the percentage of predicted positives out of the total number of postives labels
- High Recall Score shows:
  - Low False Negative rate
  - Predicted most customers who will default payment next month correctly

$$Recall = \frac{TP}{TP+FN}$$

Precision

- Shows the percentage of predicted positives that was actually correct
- High Precision Score shows:
  - Low False Positive rate

$$Precision = \frac{TP}{TP+FP}$$

F1 Score

- Harmonic mean between precision and recall
- Metrics does not consider how many True Negatives are classified

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$$

Step 4: Model Evaluation

Models Selection

- Evaluation metrics of several models to try out to see which model gets the best score based on the score metrics
- Based on their default parameters
- Typically looking at f2 score and balanced accuracy score

```
#Function for the model
def evaluate_Model(X_train, y_train, models, scoring):
    avgModelScores=[]
    #Iterate through the models
    for name, model in models:
        steps[model_step]=('model', model)
        score=cross_validate(
            Pipeline(steps=steps),
            X_train,
            y_train,
            scoring=scoring,
            cv=10,
            return_train_score=True
        )
        avgModelScores.append(pd.Series(score, name=name).apply(np.mean))
    return pd.DataFrame(avgModelScores).sort_values(by=[
        "test_f2",
        "test_balanced_accuracy"
    ],ascending=False)

model_Scores=evaluate_Model(X_train, y_train, models, score_metrics)
```

```
model_Scores
```

	fit_time	score_time	test_f2	train_f2	test_balanced_accuracy	train_balanced_accuracy	test_roc_auc	train_roc_auc
Naive Bayes	0.011167	0.012796	0.575768	0.575950	0.532803	0.532232	0.556845	0.583191
Logistic Regression	0.031858	0.013346	0.474965	0.525326	0.546983	0.588878	0.583473	0.628375
SVM	0.169309	0.050013	0.436353	0.599533	0.524082	0.654736	0.551975	0.731221
Gradient Boosting	0.432679	0.015439	0.352269	0.776381	0.565562	0.842814	0.617446	0.926425
KNN	0.014307	0.024108	0.343388	0.737393	0.491907	0.791417	0.497019	0.885393
Decision Tree	0.020452	0.015384	0.266191	1.000000	0.531878	1.000000	0.531878	1.000000
Random Forest	0.369181	0.033164	0.201443	1.000000	0.551943	1.000000	0.594055	1.000000

- Naive Bayes
- Logistic Regression
- SVM
- Gradient Boosting

Random Forest & Decision Tree **overfit training data**

Observations:

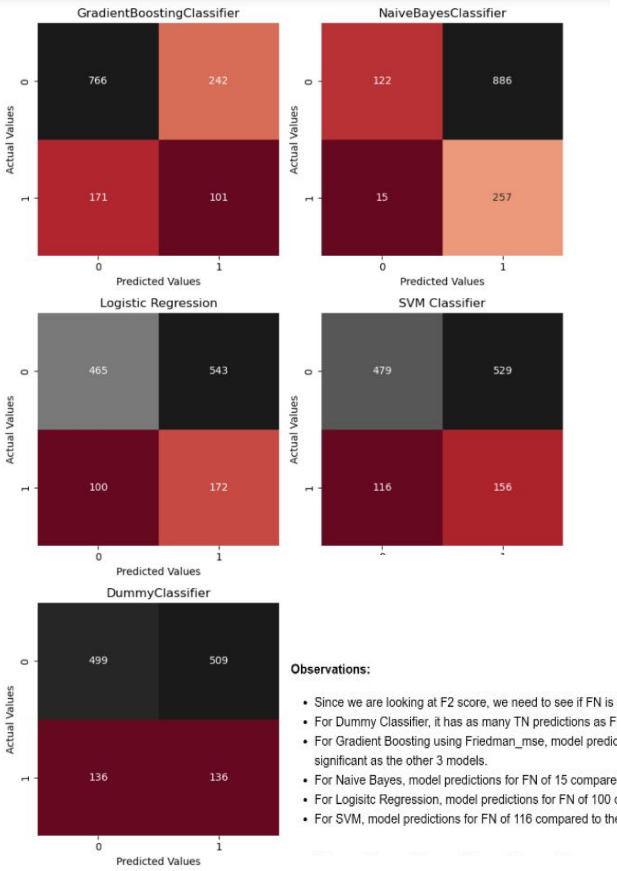
- Best 3 models with the highest test_f2 scores include Naive Bayes, Logistic Regression & SVM
- Top model with highest test_balanced_accuracy include Gradient Boosting
- Trees based model like Decision Tree and Random Forest got 1 for every score metrics which implies:
  - Both Decision Tree and Random Forest have overfitted the training data and are unable to generalize the new, unseen data, hence have a low score for all the test data.

Step 4: Model Evaluation (Dummy Classifier)

Confusion Matrix

- Determine the performance for each classifier based on the true values and false values

```
# Confusion Matrix
fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2, figsize=(8, 12))

# GradientBoostingClassifier
gradientBoost_cm = confusion_matrix(
    y_train, cross_val_predict(gradientBoost_clf, X_train, y_train, cv=10)
)
sns.heatmap(gradientBoost_cm, annot=True, fmt="", ax=ax1, cbar=False, cmap="RdGy")
ax1.set_title("GradientBoostingClassifier")
ax1.set_ylabel("Actual Values")
ax1.set_xlabel("Predicted Values")

# Naive Bayes
nb_cm = confusion_matrix(
    y_train, cross_val_predict(naivebayes_clf, X_train, y_train, cv=10)
)
sns.heatmap(nb_cm, annot=True, fmt="", ax=ax2, cbar=False, cmap="RdGy")
ax2.set_title("NaiveBayesClassifier")
ax2.set_ylabel("Actual Values")
ax2.set_xlabel("Predicted Values")

# Logistic Regression
logisticRegress_cm = confusion_matrix(
    y_train, cross_val_predict(logisticRegress_clf, X_train, y_train, cv=10)
)
sns.heatmap(logisticRegress_cm, annot=True, fmt="", ax=ax3, cbar=False, cmap="RdGy")
ax3.set_title("Logistic Regression")
ax3.set_ylabel("Actual Values")
ax3.set_xlabel("Predicted Values")

# SVM
svm_cm = confusion_matrix(
    y_train, cross_val_predict(svm_clf, X_train, y_train, cv=10)
)
sns.heatmap(svm_cm, annot=True, fmt="", ax=ax4, cbar=False, cmap="RdGy")
ax4.set_title("SVM Classifier")
ax4.set_ylabel("Actual Values")
ax4.set_xlabel("Predicted Values")

# Baseline
dummy_cm = confusion_matrix(y_train, cross_val_predict(dummy, X_train, y_train, cv=10))
sns.heatmap(dummy_cm, annot=True, fmt="", ax=ax5, cbar=False, cmap="RdGy")
ax5.set_title("DummyClassifier")
ax5.set_ylabel("Actual Values")
ax5.set_xlabel("Predicted Values")
plt.tight_layout()
plt.show()
```



Observations:

- Since we are looking at F2 score, we need to see if FN is smaller than FP.
- For Dummy Classifier, it has as many TN predictions as FP predictions but FN predictions is still fewer than FP.
- For Gradient Boosting using Friedman_mse, model predictions for FN of 171 compared to the FP of 242, can predict data with minor errors but not as significant as the other 3 models.
- For Naive Bayes, model predictions for FN of 15 compared to the FP of 886, can predict data with minor errors.
- For Logisitc Regression, model predictions for FN of 100 compared to the FP of 543, can predict data with minor errors.
- For SVM, model predictions for FN of 116 compared to the FP of 529, can predict data with minor errors.

Step 5: Model Improvement (GridSearchCV)

5) Model Improvement

Hyperparameter Tuning (GridSearchCV)

Run through the parameters to see which parameters give the best f2 score

Parameters to be tuned (GradientBoostingClassifier):

1. max_depth - maximum depth of the tree
2. max_leaf_nodes - max number of leaves nodes
3. n_estimators - numbers of trees in the forest
4. criterion - measures the quality {"friedman_mse", "squared_error"}

Parameters to be tuned (LogisticRegression):

1. max_iter - max number of iteration taken for solvers to converge
2. solver - algorithmn to use for the optimization problem {'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}
3. C - inverse of regularization strength
4. penalty - penalty terms {'l1', 'l2', 'elasticnet', None}
5. l1_ratio - the Elastic-Net mixing parameter
   - 0 <= l1_ratio <= 1 for penalty='elasticnet'
   - l1_ratio = 0 for penalty='l2'
   - l1_ratio = 1 for penalty='l1'

Parameters to be tuned (Naive Bayes):

1. var_smoothing - largest variance of all features added to variances for calculation stability

Parameters to be tuned (SVM):

1. C - regularization parameter
2. kernel - kernel type is used in the algorithm {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}
3. degree - degree of the polynomial kernel function ('poly')

Why GridSearchCV and not RandomizedSearchCV?

- GridSearchCV exhaustively considers all hyperparameter combination but
- RandomizedSearchCV can sample a given number of candidates from a hyperparameter space with a specified distribution.

Step 5: Model Improvement (GridSearchCV)

Gradient Boosting Classifier

```
list(GradientBoostingClassifier().get_params().keys())
```

```
['ccp_alpha',
 'criterion',
 'init',
 'learning_rate',
 'loss',
 'max_depth',
 'max_features',
 'max_leaf_nodes',
 'min_impurity_decrease',
 'min_samples_leaf',
 'min_samples_split',
 'min_weight_fraction_leaf',
 'n_estimators',
 'n_iter_no_change',
 'random_state',
 'subsample',
 'tol',
 'validation_fraction',
 'verbose',
 'warm_start']
```

```
# Create the parameter grid
params_grid = {
    "max_depth": [5,10,20,30,40,50,60,70,80,90,100],
    "max_leaf_nodes": np.arange(10, 16),
    "n_estimators": np.arange(50, 501, 50),
    "criterion": ["friedman_mse",'squared_error']
}

# Creating a model based on the pipeline
steps[model_step] = (
    "hyper",
    GridSearchCV(
        GradientBoostingClassifier(
            min_samples_split=2, min_samples_leaf=1
        ),
        params_grid,
        cv=10,
        verbose=1,
        n_jobs=-1,
        scoring=f2,
    ),
)

g_boost_random_search = Pipeline(steps=steps)
# Fitting Model
g_boost_random_search.fit(X_train, y_train)
print(g_boost_random_search.named_steps["hyper"].best_estimator_)
print(g_boost_random_search.named_steps["hyper"].best_params_)
print(g_boost_random_search.named_steps["hyper"].best_score_)
```

```
Fitting 10 folds for each of 1320 candidates, totalling 13200 fits
GradientBoostingClassifier(max_depth=5, max_leaf_nodes=15, n_estimators=500)
{'criterion': 'friedman_mse', 'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 500}
0.9596078777676711
```

Naive Bayes

```
list(GaussianNB().get_params().keys())
```

```
['priors', 'var_smoothing']
```

```
# Create the parameter grid
params_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]
}

# Creating a model based on the pipeline
steps[model_step] = (
    "hyper",
    GridSearchCV(
        GaussianNB(),
        params_grid,
        cv=10,
        verbose=1,
        n_jobs=-1,
        scoring=f2,
    ),
)

gaussian_search = Pipeline(steps=steps)
# Fitting Model
gaussian_search.fit(X_train, y_train)
print(gaussian_search.named_steps["hyper"].best_estimator_)
print(gaussian_search.named_steps["hyper"].best_params_)
print(gaussian_search.named_steps["hyper"].best_score_)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
GaussianNB()
{'var_smoothing': 1e-09}
0.8099164400636507
```

Logistic Regression

```
list(LogisticRegression().get_params().keys())
```

```
['C',
 'class_weight',
 'dual',
 'fit_intercept',
 'intercept_scaling',
 'l1_ratio',
 'max_iter',
 'multi_class',
 'n_jobs',
 'penalty',
 'random_state',
 'solver',
 'tol',
 'verbose',
 'warm_start']
```

```
import warnings
warnings.filterwarnings('ignore')

# Create the parameter grid
params_grid = {
    "max_iter": np.arange(100, 501, 100),
    "solver": ['newton-cholesky', 'saga', 'lbfgs', 'newton-cg', 'liblinear'],
    "C": [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1],
    'penalty':['l1','l2','elasticnet'],
    'l1_ratio': np.arange(0, 1.01, 0.02)
}

# Creating a model based on the pipeline
steps[model_step] = (
    "hyper",
    GridSearchCV(
        LogisticRegression(
            random_state=42
        ),
        params_grid,
        cv=10,
        verbose=1,
        n_jobs=-1,
        scoring=f2,
    ),
)

logisticRegress_search = Pipeline(steps=steps)
# Fitting Model
logisticRegress_search.fit(X_train, y_train)
print(logisticRegress_search.named_steps["hyper"].best_estimator_)
print(logisticRegress_search.named_steps["hyper"].best_params_)
print(logisticRegress_search.named_steps["hyper"].best_score_)
```

```
Fitting 10 folds for each of 26775 candidates, totalling 267750 fits
LogisticRegression(C=0.01, l1_ratio=0.44, penalty='elasticnet', random_state=42,
                   solver='saga')
{'C': 0.01, 'l1_ratio': 0.44, 'max_iter': 100, 'penalty': 'elasticnet', 'solver': 'saga'}
0.7640289453992228
```

SVC

```
list(SVC().get_params().keys())
```

```
['C',
 'break_ties',
 'cache_size',
 'class_weight',
 'coef0',
 'decision_function_shape',
 'degree',
 'gamma',
 'kernel',
 'max_iter',
 'probability',
 'random_state',
 'shrinking',
 'tol',
 'verbose']
```

```
# Create the parameter grid
params_grid = {
    "C": [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1],
    'kernel':['linear', 'rbf', 'poly', 'sigmoid'],
    'degree':[2,3,4],
}

# Creating a model based on the pipeline
steps[model_step] = (
    "hyper",
    GridSearchCV(
        SVC(random_state=42),
        params_grid,
        cv=10,
        verbose=1,
        n_jobs=-1,
        scoring=f2,
    ),
)

svc_search = Pipeline(steps=steps)
# Fitting Model
svc_search.fit(X_train, y_train)
print(svc_search.named_steps["hyper"].best_estimator_)
print(svc_search.named_steps["hyper"].best_params_)
print(svc_search.named_steps["hyper"].best_score_)
```

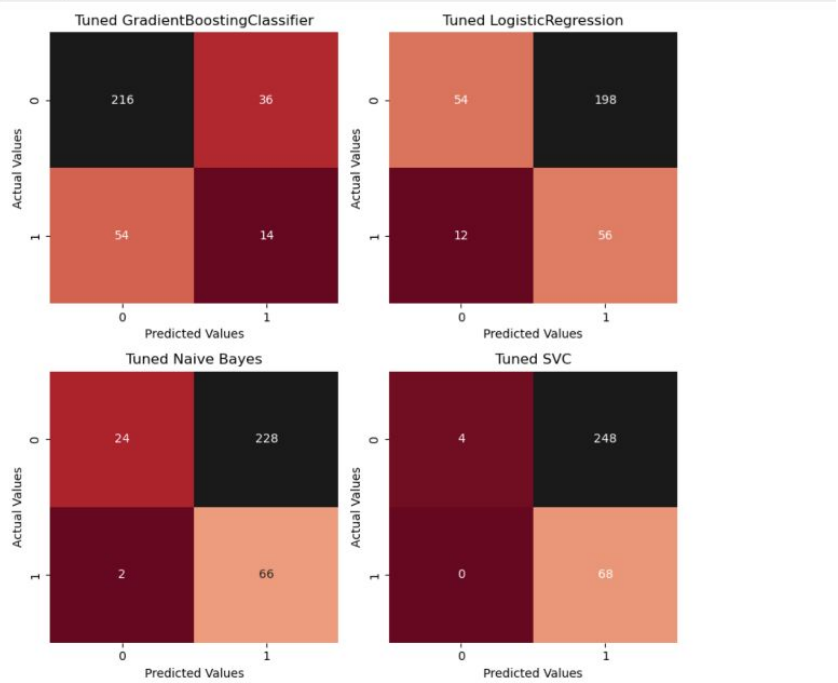
```
Fitting 10 folds for each of 84 candidates, totalling 840 fits
SVC(C=0.001, degree=4, kernel='poly', random_state=42)
{'C': 0.001, 'degree': 4, 'kernel': 'poly'}
0.8361134568123288
```

Step 5: Evaluated Improved Model (Confusion Matrix)

- Determine **performance for each classifier** based on true and false values

Confusion Matrix

```
# Confusion Matrix
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(8, 8))

# GradientBoostingClassifier
gradientBoost_tuned_cm = confusion_matrix(y_test, gradientBoost_y_pred)
sns.heatmap(gradientBoost_tuned_cm, annot=True, fmt="", ax=ax1, cbar=False, cmap="RdGy")
ax1.set_title("Tuned GradientBoostingClassifier", fontsize=12)
ax1.set_ylabel("Actual Values")
ax1.set_xlabel("Predicted Values")

# Logistic Regression
logisticRegress_tuned_cm = confusion_matrix(y_test, logisticRegress_y_pred)
sns.heatmap(logisticRegress_tuned_cm, annot=True, fmt="", ax=ax2, cbar=False, cmap="RdGy")
ax2.set_title("Tuned LogisticRegression", fontsize=12)
ax2.set_ylabel("Actual Values")
ax2.set_xlabel("Predicted Values")

# Naive Bayes
gaussian_tuned_cm = confusion_matrix(y_test, gaussian_y_pred)
sns.heatmap(gaussian_tuned_cm, annot=True, fmt="", ax=ax3, cbar=False, cmap="RdGy")
ax3.set_title("Tuned Naive Bayes", fontsize=12)
ax3.set_ylabel("Actual Values")
ax3.set_xlabel("Predicted Values")

# SVC
svc_tuned_cm = confusion_matrix(y_test, svc_y_pred)
sns.heatmap(svc_tuned_cm, annot=True, fmt="", ax=ax4, cbar=False, cmap="RdGy")
ax4.set_title("Tuned SVC", fontsize=12)
ax4.set_ylabel("Actual Values")
ax4.set_xlabel("Predicted Values")

plt.tight_layout()
plt.show()
```



Observations:

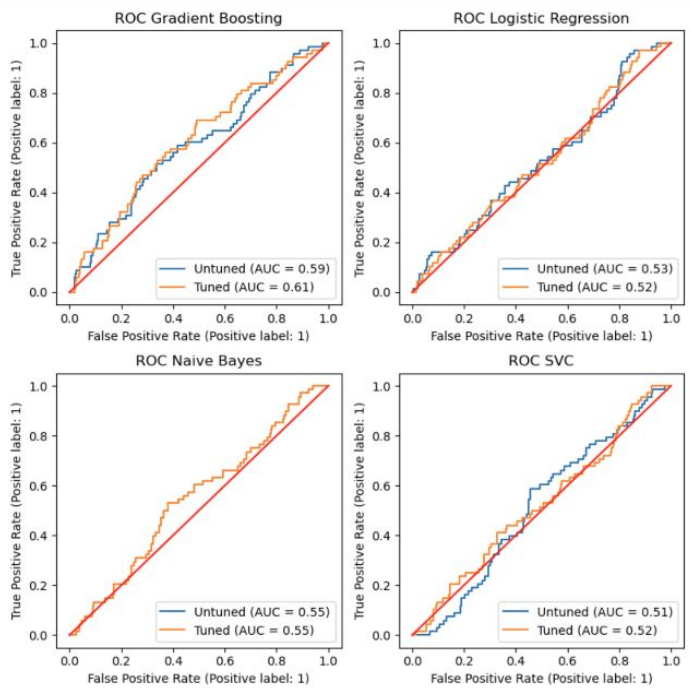
- Even though the f2 score for logistic regression is high, a score of 0.96, it is not a good model to use as the FN prediction of 54 is greater than the FP prediction of 36, shows that it gives more weight to precision instead of recall.
- For the other models, the FN prediction are way lower than FP prediction, which shows that these able to predict whether the customer will default payment or not with high recall.
- Tuned SVC has a the lowest FN prediction compared to the other models hence it has the best F2 score.

Step 5: Evaluated Improved Model (ROC curve)

- Find **performance** of a classification model at **all classification thresholds**

ROC Curve

```
# ROC Curve
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(8, 8))

# Logistic Regression
RocCurveDisplay.from_estimator(gradientBoost_clf, X_test, y_test, ax=ax1, name="Untuned")
RocCurveDisplay.from_estimator(tuned_gradientBoost_clf, X_test, y_test, ax=ax1, name="Tuned")

# Logistic Regression
RocCurveDisplay.from_estimator(logisticRegress_clf, X_test, y_test, ax=ax2, name="Untuned")
RocCurveDisplay.from_estimator(tuned_logisticRegress_clf, X_test, y_test, ax=ax2, name="Tuned")

# Naive Bayes
RocCurveDisplay.from_estimator(naivebayes_clf, X_test, y_test, ax=ax3, name="Untuned")
RocCurveDisplay.from_estimator(tuned_gaussian_clf, X_test, y_test, ax=ax3, name="Tuned")

# SVC
RocCurveDisplay.from_estimator(svm_clf, X_test, y_test, ax=ax4, name="Untuned")
RocCurveDisplay.from_estimator(tuned_svc_clf, X_test, y_test, ax=ax4, name="Tuned")

#50% Line
x = np.linspace(0,1,2)
ax1.plot(x,x,color="red")
ax2.plot(x,x,color="red")
ax3.plot(x,x,color="red")
ax4.plot(x,x,color="red")

ax1.set_title("ROC Gradient Boosting")
ax2.set_title("ROC Logistic Regression")
ax3.set_title("ROC Naive Bayes")
ax4.set_title("ROC SVC")

plt.tight_layout()
plt.show()
```



Observations:

- Overall, as there is an imbalanced set of data where more customer will choose not to default payment than default payment, the AUC-ROC is weak for all 3 models as a result
- Naive Bayes has a similar ROC for untuned and tuned as the parameter chose the default value even after tuning but still has the best performance than the other 2 tuned models.
- ROC_AUC of SVC improved a little after tuning
- Overall, even though from the confusion matrix, we have determined that SVC has the best f2 score, it does not have a better roc_auc score compared to Naive Bayes which makes SVC less reliable than Naive Bayes

Step 5: Evaluated Improved Model (Using test data)

```
#Show the F2 scores & Balanced Accuracy of test data

pd.DataFrame([
    [balanced_accuracy_score(y_test,logisticRegress_y_pred),
     balanced_accuracy_score(y_test, gaussian_y_pred),
     balanced_accuracy_score(y_test, svc_y_pred)]],
    columns=["Logisitc Regression", "Naive Bayes", "SVC"],
    index=["Balanced Accuracy"],
)
```

	Logisitc Regression	Naive Bayes	SVC
Balanced Accuracy	0.518908	0.532913	0.507937

Observations:

- Based on the balanced accuracy score, Naive Bayes performed the best.
- Shows that Naive Bayes is the best at predicting whether the customer will default payment or not.

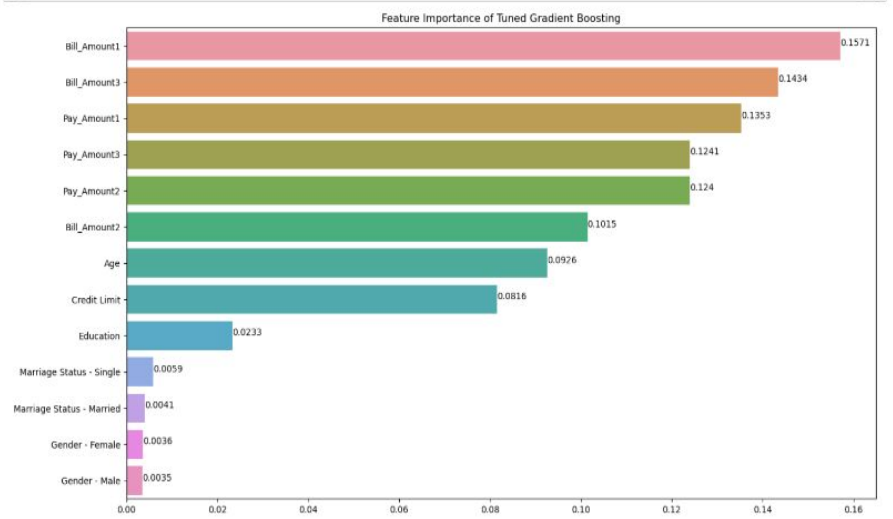
- Balanced Accuracy score is **moderate** as the models were **not tuned according to that metrics**

Step 5: Evaluated Improved Model (Feature Importance)

Feature Importance

- Find out which features the model considers important using Gradient Boosting as it is an ensemble model
- Only Gradient Boosting is able to determine feature importance

```
feature_names=["Gender - Female","Gender - Male",  "Marriage Status - Married",  "Marriage Status - Single", "Education",
               'Credit Limit', 'Age', 'Bill_Amount1', 'Bill_Amount2', 'Bill_Amount3', 'Pay_Amount1','Pay_Amount2', 'Pay_Amount3']

# Create feature importance and sort the features based on the coefficient
importance = pd.Series(
tuned_gradientBoost_clf[-1].feature_importances_, index=feature_names
).sort_values(ascending=False)
fig, ax = plt.subplots(figsize=(16, 9))
sns.barplot(
    x=importance.values, y=importance.index, ax=ax
)
ax.set_title("Feature Importance of Tuned {}".format("Gradient Boosting"))

for i, v in zip(np.arange(0, len(importance)), importance.values):
        ax.text(x=v, y=i, s=round(v, 4))

plt.show()
```



Observations:

- Top 3 features Gradient Boosting has considered most important are Bill_Amount1, Bill_Amount3 and Pay_Amount1

Summary

- **Naive Bayes** is the **best model** based on the **balanced_auc score & roc_auc score**.
- It also has quite a **strong f2 score** which show that the model is performing well for **both precision and recall**, with **higher emphasis on recall**.
- Naive Bayes strikes a good balance between **correctly identifying customers** who will **default** and **minimizing the number of customers falsely predicted to default**.
- Make sense as features are **independent** of one another, features **do not have strong relationship** with one another

Thank You!

References

1. Julia, K., 2023. Default Risk: Definition, Types, and Ways to Measure. [online]. Investopia. Available from: https://www.investopedia.com/terms/d/defaultrisk.asp [Accessed at 9 Jun 2022].

2. Amy., 2022. Four Oversampling and Under-Sampling Methods for Imbalanced Classification Using Python. [online]. Medium. Available from: https://medium.com/grabngoinfo/four-oversampling-and-under-sampling-methods-for-imbalanced-classification-using-python-7304aedf9037 [Accessed at 9 Jun 2022]

3. H20.ai. 2023. Scorers: Classification or Regression. [online]. DriverlessAI. Available from: https://docs.h2o.ai/driverless-ai/1-10-lts/docs/userguide/scorers.html#:~:text=The%20F2%20score%20is%20the,to%20recall%20than%20to%20precision. [Acessed at 9 Jun 2022].

4. The Investopia Team., 2023. Credit Utilization Ratio: Definition, Calculation, and How To Improve. [online]. Investopia. Available from: https://www.investopedia.com/terms/c/credit-utilization-rate.asp [Accessed at 9 Jun 2023].

5. Zach., 2021. What is Balanced Accuracy? (Definition & Example). [online]. Statology. Available from: https://www.statology.org/balanced-accuracy/#:~:text=Balanced%20accuracy%20is%20a%20metric,model%20is%20able%20to%20detect . [Accessed at 9 Jun 2022].

6. Classical ML Equations in LaTeX.[online] GitHub. https://blmoistawinde.github.io/ml_equations_latex/ [Accessed at 9 Jun 2023].

7. Confusion Matrix. [online] w3schools. https://www.w3schools.com/python/python_ml_confusion_matrix.asp [Accessed at 9 Jun 2023]

8. Bhandari, A., 2023. Guide to AUC ROC Curve in Machine Learning : What Is Specificity? [online]. Analytics Vidhya. Available from: https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/ [Accessed at 9 Jun 2023]

9. Shin, T., 2023. Understanding Feature Importance and How to Implement it in Python. [online]. Medium. Available from: https://towardsdatascience.com/understanding-feature-importance-and-how-to-implement-it-in-python-ff0287b20285 [Accessed at 9 Jun 2023]

Sklearn References

1. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html