Credit Card Default Prediction

About The Dataset

Attribute Information:

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/ her family (supplementary) credit.
- X2: Gender (1 = male; 2 = female).
- X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- X4: Marital status (1 = married; 2 = single; 3 = others).
- X5: Age (year).

- X6: The repayment status in September, 2005
- X7: The repayment status in August, 2005
- X8: The repayment status in July, 2005
- X9: The repayment status in June, 2005
- X10: The repayment status in May, 2005
- X11: The repayment status in April, 2005
  The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payent delay for eight months; 9 = payment delay for nine months and above.

- X12: The amount of bill statement in September, 2005
- X13: The amount of bill statement in August, 2005
- X14: The amount of bill statement in July, 2005
- X15: The amount of bill statement in June, 2005
- X16: The amount of bill statement in May, 2005
- X17: The amount of bill statement in April, 2005

- X12: The amount of previous payment in September, 2005
- X13: The amount of previous payment in August, 2005
- X14: The amount of previous payment in July, 2005
- X15: The amount of previous payment in June, 2005
- X16: The amount of previous payment in May, 2005
- X17: The amount of previous payment in April, 2005

Life Cycle of Machine Learning Project

- Understanding the Problem Statement
- Data Collection
- Data Checks to perform

- Exploratory data analysis
- Data Pre-Processing
- Model Training
- Choose best model

Data Ingestion

Importing The Required Packages

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Reading Data From Data Source
df= pd.read_excel("default of credit card clients.xls", skiprows=[0])
```

Explorative Data Analysis

```
# View The First 5 Rows
df.head()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default payment next month
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	0	689	0	0	0	0	1
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	0	1000	1000	1000	0	2000	1
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	2000	2019	1200	1100	1069	1000	0
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131	2000	36681	10000	9000	689	679	0

5 rows × 25 columns

```
# View The Last 5 Rows
df.tail()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	def pay m
29995	29996	220000	1	3	1	39	0	0	0	0	...	88004	31237	15980	8500	20000	5003	3047	5000	1000	
29996	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	5190	0	1837	3526	8998	129	0	0	

```
# Dataset Shape
df.shape
```

```
(30000, 25)
```

Insight

There are 30,000 records and 25 columns

```
# More Info About The Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  int64
 2   SEX                         30000 non-null  int64
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
 5   AGE                         30000 non-null  int64
 6   PAY_0                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  int64
 13  BILL_AMT2                   30000 non-null  int64
 14  BILL_AMT3                   30000 non-null  int64
 15  BILL_AMT4                   30000 non-null  int64
 16  BILL_AMT5                   30000 non-null  int64
 17  BILL_AMT6                   30000 non-null  int64
 18  PAY_AMT1                    30000 non-null  int64
 19  PAY_AMT2                    30000 non-null  int64
 20  PAY_AMT3                    30000 non-null  int64
 21  PAY_AMT4                    30000 non-null  int64
 22  PAY_AMT5                    30000 non-null  int64
 23  PAY_AMT6                    30000 non-null  int64
 24  default payment next month  30000 non-null  int64
dtypes: int64(25)
memory usage: 5.7 MB
```

```
# Statistical Summary Of Dataset
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ID	30000.0	15000.500000	8660.398374	1.0	7500.75	15000.5	22500.25	30000.0
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	1000000.0
SEX	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	2.0
EDUCATION	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	6.0
MARRIAGE	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	3.0
AGE	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	79.0
PAY_0	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	8.0
PAY_2	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	8.0
PAY_3	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	8.0
PAY_4	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	8.0
PAY_5	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	8.0
PAY_6	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	8.0
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	964511.0
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	983931.0
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	1664089.0
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	891586.0
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	927171.0
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	961664.0
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	873552.0
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	1684259.0
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	896040.0
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	621000.0
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	426529.0
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	528666.0

```
# Checking For Duplicate
df.duplicated().sum()
```

```
0
```

Insight

There are no duplicates.

```
# Checking For Missing Values
df.isnull().sum()
```

```
ID                          0
LIMIT_BAL                   0
SEX                         0
```

```
EDUCATION                   0
MARRIAGE                    0
AGE                         0
PAY_0                       0
PAY_2                       0
PAY_3                       0
PAY_4                       0
PAY_5                       0
PAY_6                       0
BILL_AMT1                   0
BILL_AMT2                   0
BILL_AMT3                   0
BILL_AMT4                   0
BILL_AMT5                   0
BILL_AMT6                   0
PAY_AMT1                    0
PAY_AMT2                    0
PAY_AMT3                    0
PAY_AMT4                    0
PAY_AMT5                    0
PAY_AMT6                    0
default payment next month  0
dtype: int64
```

Insight

There are no missing values.

```
# Checking For No of Unique
df.nunique()
```

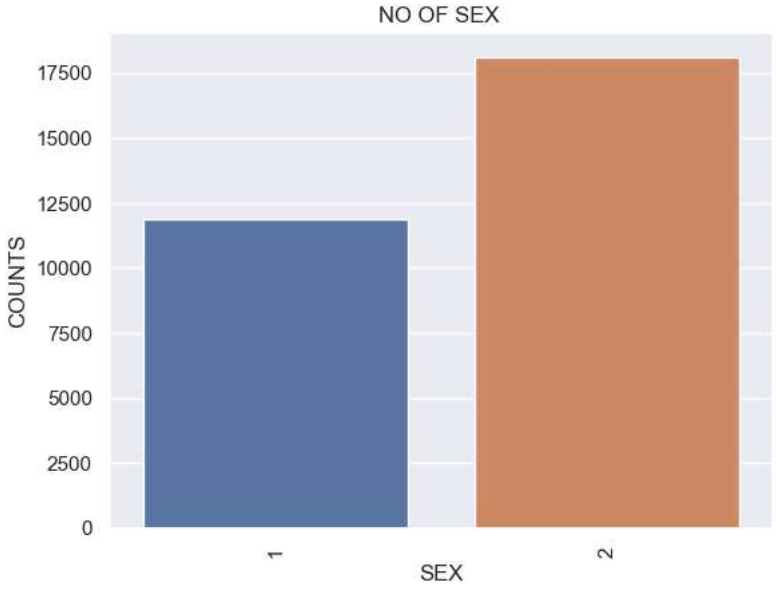
```
ID                          30000
LIMIT_BAL                      81
SEX                             2
EDUCATION                       7
MARRIAGE                        4
AGE                            56
PAY_0                          11
PAY_2                          11
PAY_3                          11
PAY_4                          11
PAY_5                          10
PAY_6                          10
BILL_AMT1                   22723
BILL_AMT2                   22346
BILL_AMT3                   22026
BILL_AMT4                   21548
BILL_AMT5                   21010
BILL_AMT6                   20604
PAY_AMT1                     7943
PAY_AMT2                     7899
PAY_AMT3                     7518
PAY_AMT4                     6937
PAY_AMT5                     6897
PAY_AMT6                     6939
default payment next month      2
dtype: int64
```

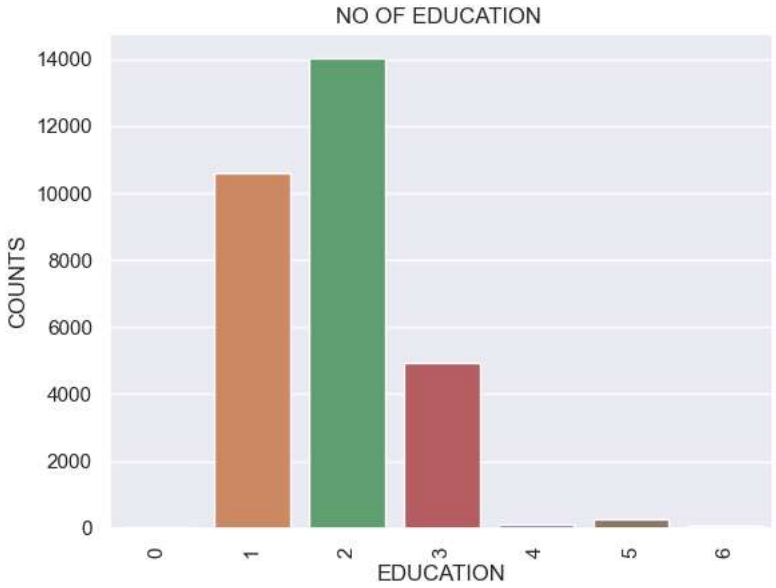
```
# Drop Columns Not Required For Analysis
df.drop("ID", axis=1, inplace=True)
```

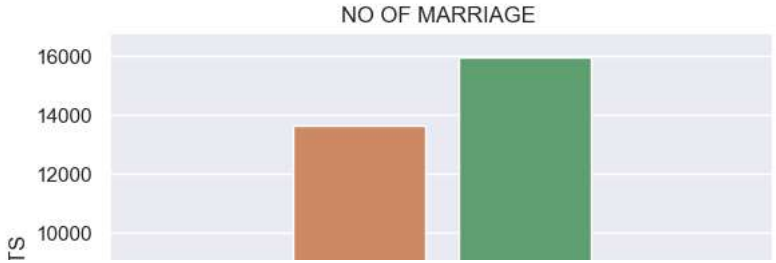
```
# Create A Copy Of The Dataset
data=df.copy()
```

Univariate Analysis

```
sns.set(style="darkgrid")
for i in data.columns[1:11]:
    sns.countplot(x=i, data=data)
    plt.title(f"NO OF {i} ")
    plt.ylabel("COUNTS")
    plt.xticks(rotation=90)
    plt.show()
```

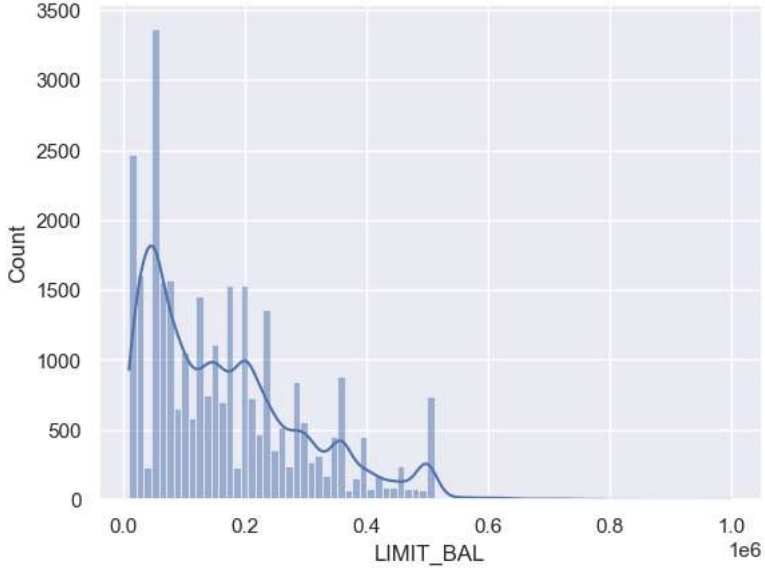

NO OF SEX
COUNTS
17500
15000
12500
10000
7500
5000
2500
0
1
2
SEX

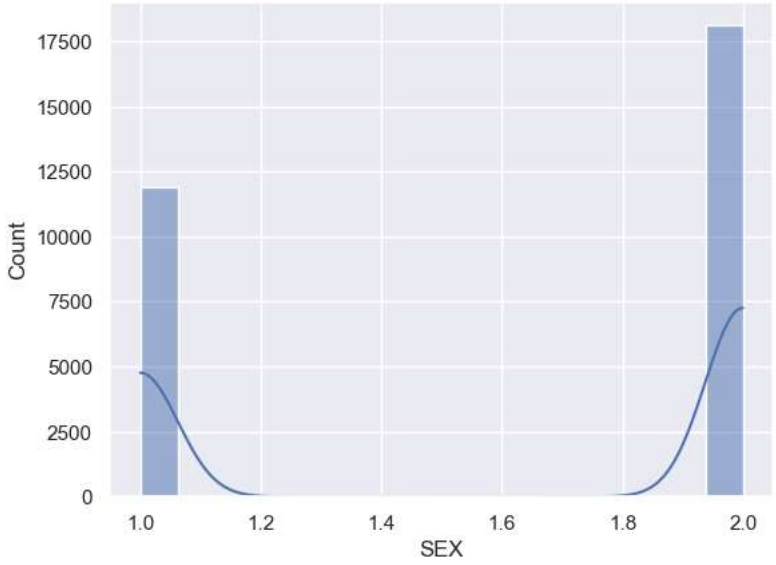

NO OF EDUCATION
COUNTS
14000
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
EDUCATION

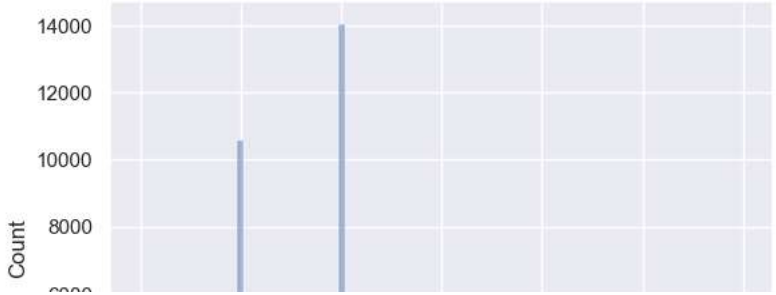

NO OF MARRIAGE
16000
14000
12000
10000

Checking Data Distribution


```
sns.set(style="darkgrid")
for i in data.columns:
    sns.histplot(x=i, data=data, kde=True)
    plt.show()
```

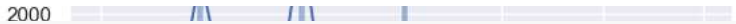

Count
LIMIT_BAL
1e6


Count
SEX

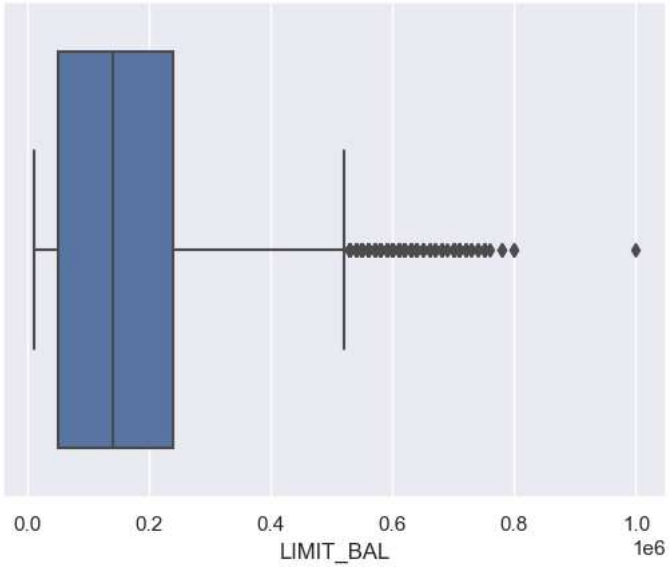

Count

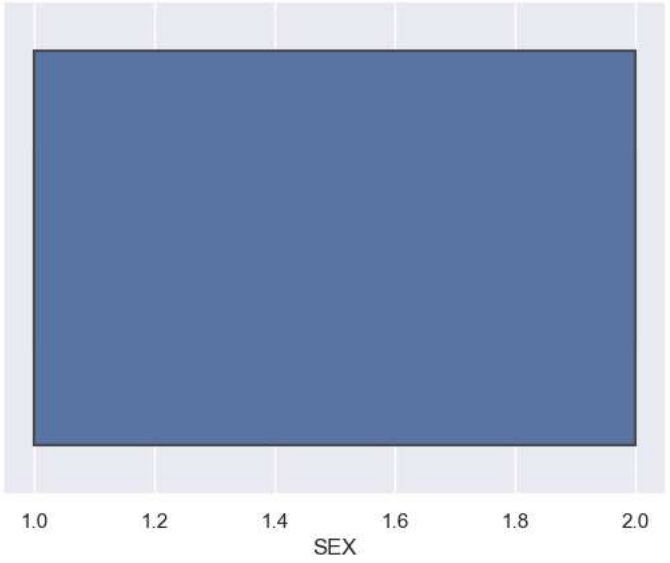
Insight

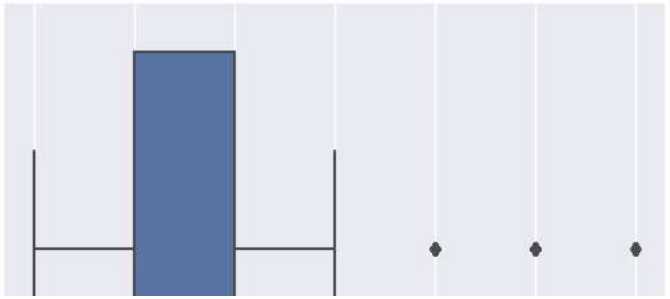
The data does not follow guassian/normal distribution.




```
sns.set(style="darkgrid")
for i in data.columns:
    sns.boxplot(x=i, data=data)
    plt.show()
```


0.0
0.2
0.4
0.6
0.8
1.0
1e6
LIMIT_BAL


1.0
1.2
1.4
1.6
1.8
2.0
SEX



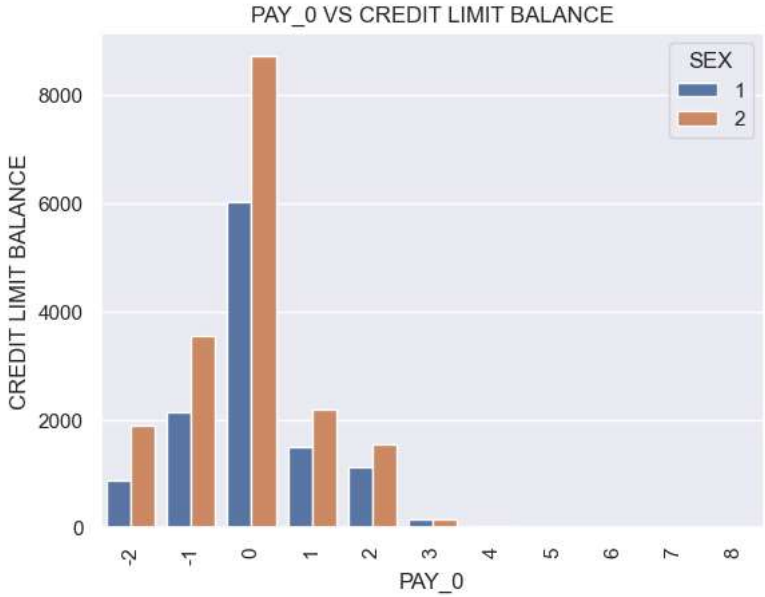
Insight

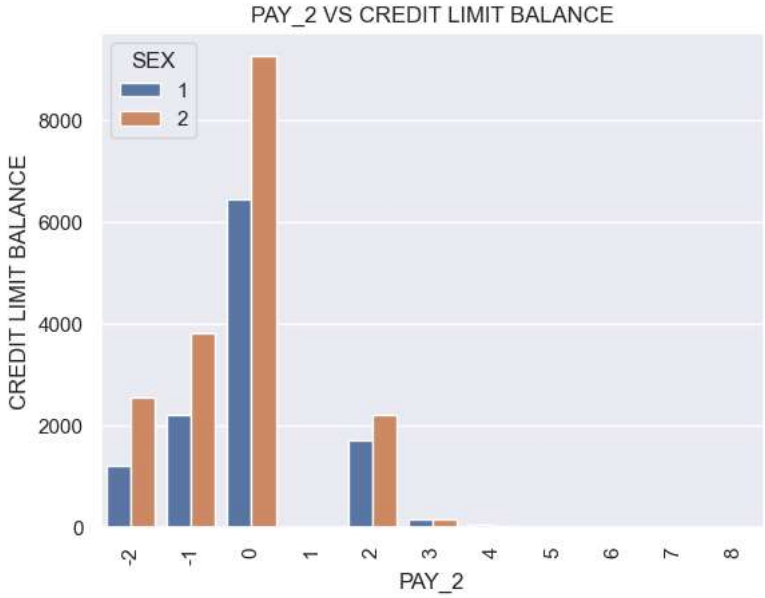
There are lots of outliers in the dataset.

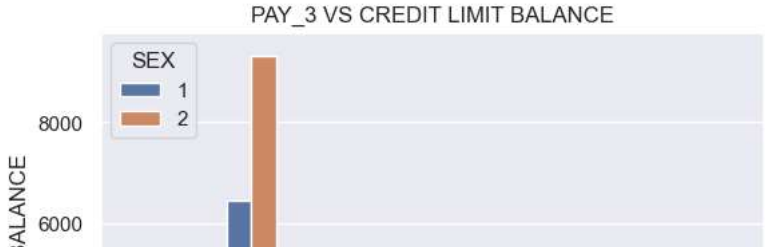
Bivariate Analysis

0 1 2 3 4 5 6

```
sns.set(style="darkgrid")
for i in data.columns[5:11]:
    sns.countplot(x=i, data=data, hue="SEX")
    plt.title(f"{i} VS CREDIT LIMIT BALANCE")
    plt.ylabel("CREDIT LIMIT BALANCE")
    plt.xticks(rotation=90)
    plt.show()
```

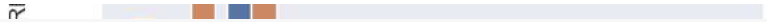

PAY_0 VS CREDIT LIMIT BALANCE
SEX
1
2
CREDIT LIMIT BALANCE
0
2000
4000
6000
8000
-2
-1
0
1
2
3
4
5
6
7
8
PAY_0


PAY_2 VS CREDIT LIMIT BALANCE
SEX
1
2
CREDIT LIMIT BALANCE
0
2000
4000
6000
8000
-2
-1
0
1
2
3
4
5
6
7
8
PAY_2

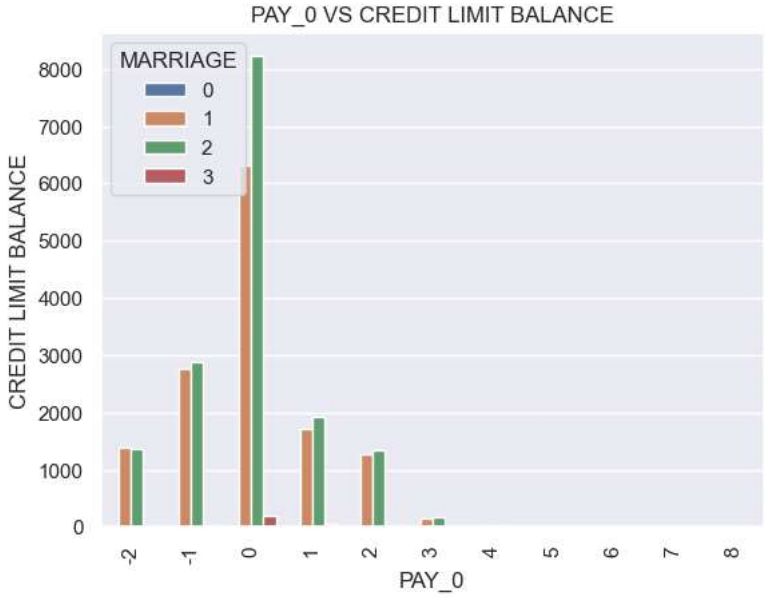

PAY_3 VS CREDIT LIMIT BALANCE
SEX
1
2
8000
6000

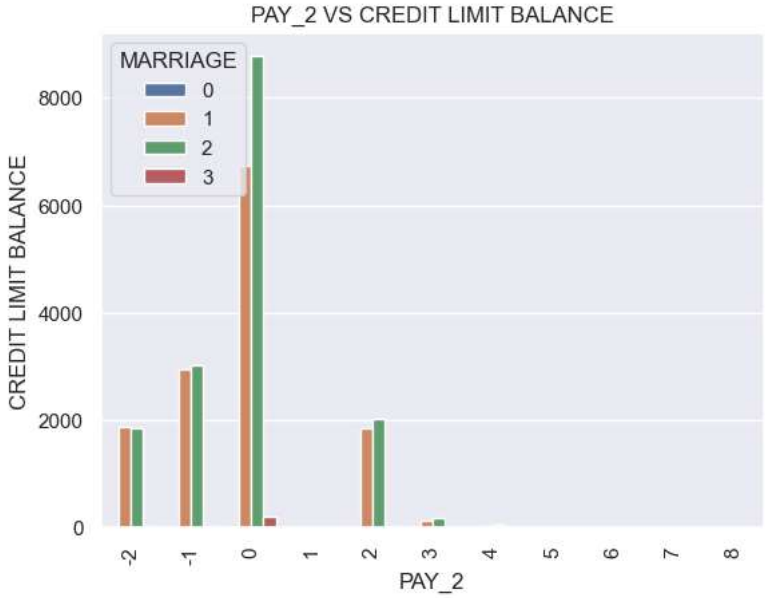
Insights

- Majority of the customer both male and female pay as at when due.



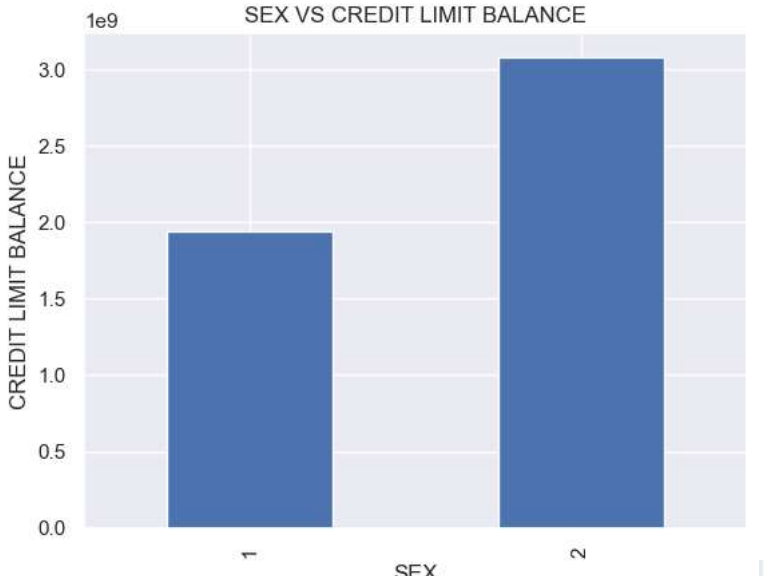
```
sns.set(style="darkgrid")
for i in data.columns[5:11]:
    sns.countplot(x=i, data=data, hue="MARRIAGE")
    plt.title(f"{i} VS CREDIT LIMIT BALANCE")
    plt.ylabel("CREDIT LIMIT BALANCE")
    plt.xticks(rotation=90)
    plt.show()
```


PAY_0 VS CREDIT LIMIT BALANCE
MARRIAGE
0
1
2
3
CREDIT LIMIT BALANCE
8000
7000
6000
5000
4000
3000
2000
1000
0
-2
-1
0
1
2
3
4
5
6
7
8
PAY_0


PAY_2 VS CREDIT LIMIT BALANCE
MARRIAGE
0
1
2
3
CREDIT LIMIT BALANCE
8000
6000
4000
2000
0
-2
-1
0
1
2
3
4
5
6
7
8
PAY_2


PAY_3 VS CREDIT LIMIT BALANCE
MARRIAGE
0
1
2
3
8000
6000

```
s asns.set(style="darkgrid")
for i in data.columns[1:4]:
    data.groupby(i)["LIMIT_BAL"].sum().plot(kind="bar")
    plt.title(f"{i} VS CREDIT LIMIT BALANCE")
    plt.ylabel("CREDIT LIMIT BALANCE")
    plt.xticks(rotation=90)
    plt.show()
```



Insights

- Bulk of the credits were advanced to female customers.
- More than 80 percent of the facility were given to graduate school and university customers.
- More than 99 percent of the credits were given to both married and single in almost equal proportion.

```
from matplotlib.pyplot import figure
figure(figsize=(15, 6), dpi=80)
data.groupby("AGE")["LIMIT_BAL"].sum().plot(kind="bar")
plt.title("AGE VS CREDIT LIMIT BALANCE")
plt.ylabel("CREDIT LIMIT BALANCE")
plt.xticks(rotation=90)
plt.show()
```

Insight

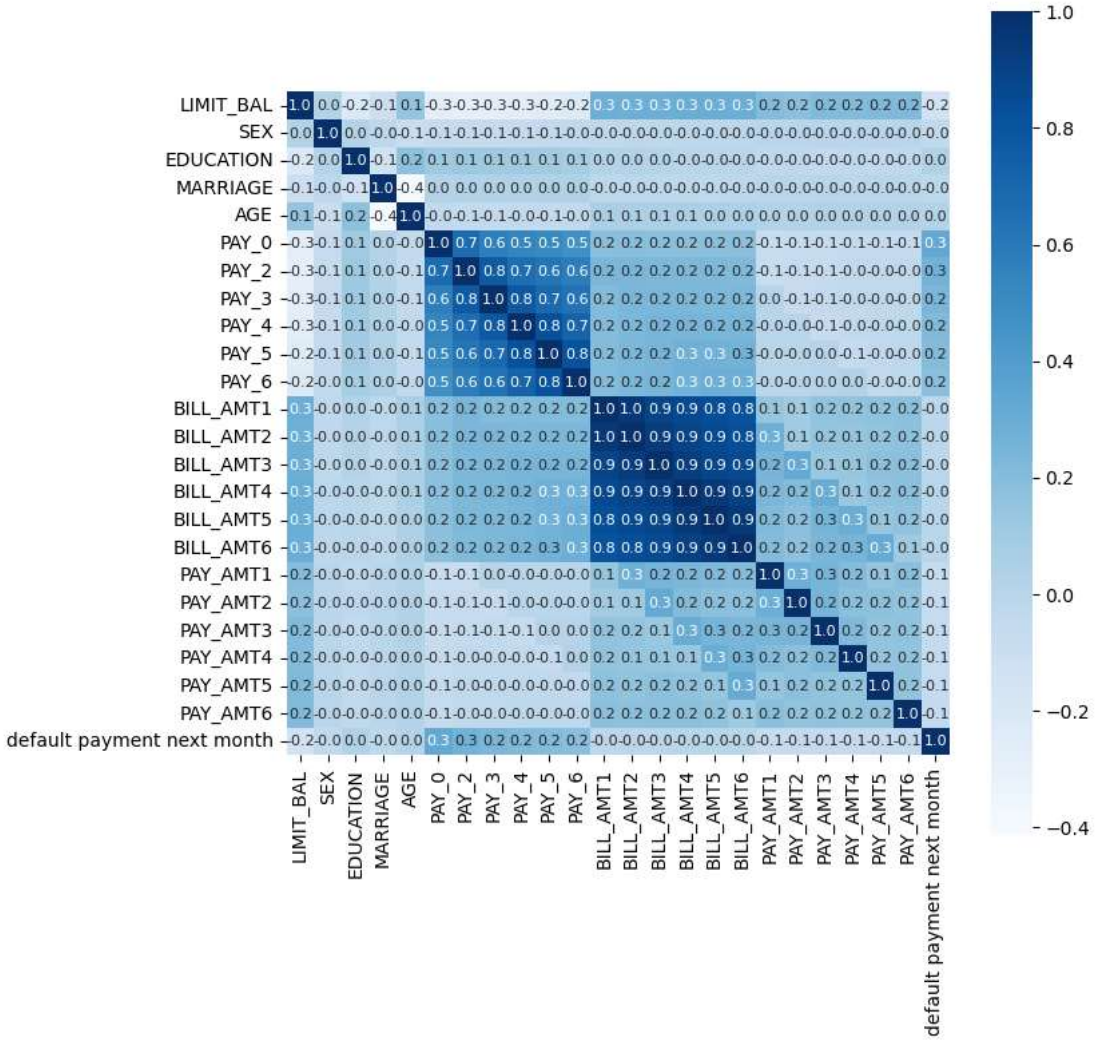
More than 60 percent of the facility were given to ages between 25 to 45

Multivariate Analysis

```
1 data.corr()
```

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	P
LIMIT_BAL	1.000000	0.024755	-0.219161	-0.108139	0.144713	-0.271214	-0.296382	-0.286123	-0.267460	-0.249411	...	0.293988	0.295562	0.290389	0.195236	0.178408	0.210167	0
SEX	0.024755	1.000000	0.014232	-0.031389	-0.090874	-0.057643	-0.070771	-0.066096	-0.060173	-0.055064	...	-0.021880	-0.017005	-0.016733	-0.000242	-0.001391	-0.008597	-0

```
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(),cbar=True, square=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap='Blues')
plt.show()
```



Feature Engineering

Feature Engineering involves handling the following

- missing values - No Missing Value
- handling outlier
- handling categorical encoding - No Categorical Variable
- feature scaling
- feature selection

```
# There are no missing values in the dataset
data.isnull().sum()
```

```
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default payment next month    0
dtype: int64
```

```
# Handling Outliers
def find_limits(df, variables):
    # Let us find the IQR (Inter Quantile Range)
    Q1 = df[variables].quantile(.25)
    Q3 = df[variables].quantile(.75)
    IQR = Q3 - Q1
    lower_limit = Q1-1.5*IQR
    upper_limit = Q3+1.5*IQR
    return lower_limit, upper_limit
```

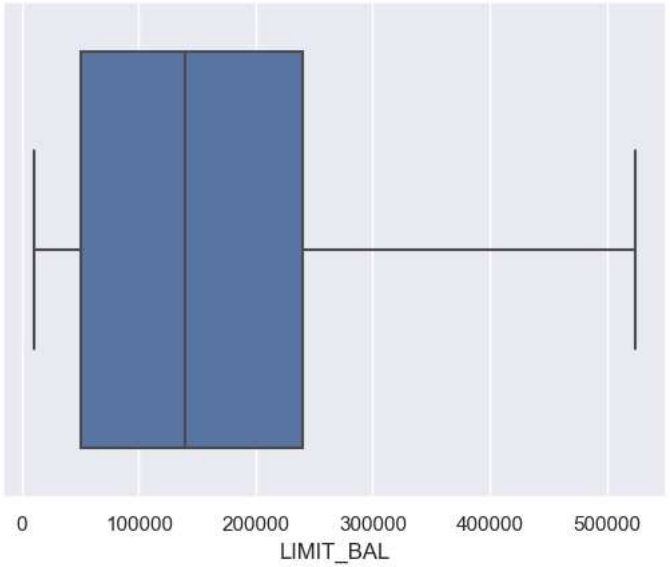
```
# Calculating Lower and Upper Limit For columns And Capping Variables At Same
for col in df.columns[0:23]:
    lower_limit, upper_limit = find_limits(df,col)
    df[col]=np.where(df[col] > upper_limit, upper_limit,
                     np.where(df[col] < lower_limit, lower_limit, df[col]))
    print(f"For {col} variable: \n")
    print(f"The Lower Limit: {lower_limit}")
    print(f"The Upper Limit: {upper_limit}")
```

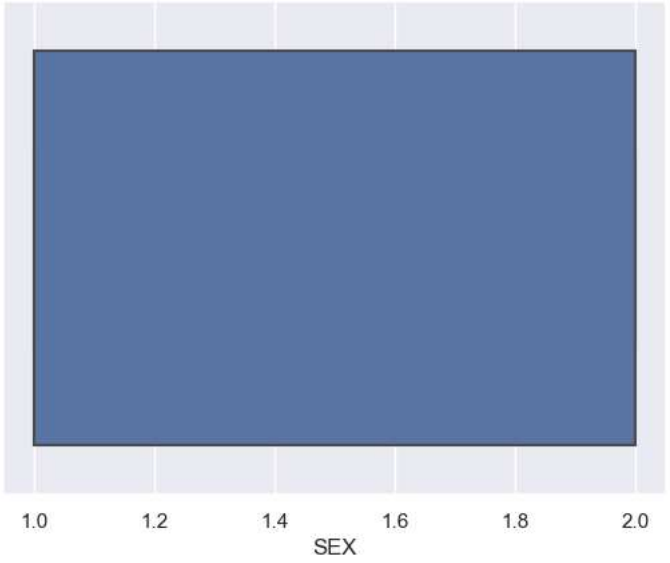
```
  print("==========================================================================================")
```

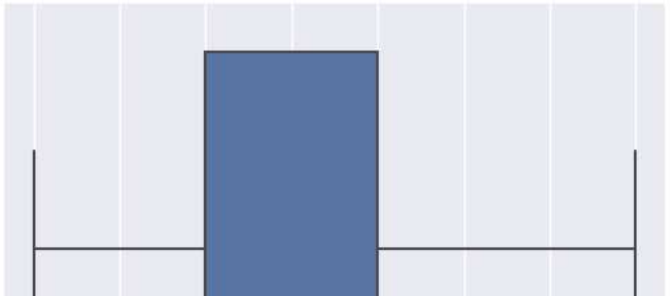
```
For LIMIT_BAL variable:

The Lower Limit: -235000.0
The Upper Limit: 525000.0
==========================================================================================
For SEX variable:

The Lower Limit: -0.5
The Upper Limit: 3.5
==========================================================================================
For EDUCATION variable:

The Lower Limit: -0.5
The Upper Limit: 3.5
==========================================================================================
For MARRIAGE variable:

The Lower Limit: -0.5
The Upper Limit: 3.5
==========================================================================================
For AGE variable:

The Lower Limit: 8.5
The Upper Limit: 60.5
==========================================================================================
For PAY_0 variable:

The Lower Limit: -2.5
The Upper Limit: 1.5
==========================================================================================
For PAY_2 variable:

The Lower Limit: -2.5
The Upper Limit: 1.5
==========================================================================================
For PAY_3 variable:

The Lower Limit: -2.5
The Upper Limit: 1.5
==========================================================================================
For PAY_4 variable:

The Lower Limit: -2.5
The Upper Limit: 1.5
==========================================================================================
For PAY_5 variable:

The Lower Limit: -2.5
The Upper Limit: 1.5
==========================================================================================
For PAY_6 variable:

The Lower Limit: -2.5
The Upper Limit: 1.5
==========================================================================================
For BILL_AMT1 variable:

The Lower Limit: -91739.625
```

```
sns.set(style="darkgrid")
for i in df.columns[0:23]:
```

```
    sns.boxplot(x=i, data=df)
    plt.show()
```


0
100000
200000
300000
400000
500000
LIMIT_BAL


1.0
1.2
1.4
1.6
1.8
2.0
SEX



Model Building, Hyper-parameter Tuning And Model Evaluation

```
# Import The Key Packages
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report,roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# Spliting Dataset Into Dependent And Independent Features
X = df.drop("default payment next month", axis =1)
y = df["default payment next month"]
```

```
# Split Dataset Into Train And Test Splits
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,random_state=42)
```

```
# Dataset Scaling And Standardisation
data_scaling = StandardScaler()
X_train =data_scaling.fit_transform(X_train)
X_test = data_scaling.transform(X_test)

```

```
X_train
```

```
array([[-0.67739393,  0.80815856,  0.22948545, ..., -0.48492216,
        -0.41463358, -0.58397358],
       [-1.06798974, -1.23738094,  0.22948545, ..., -0.57057323,
        -0.8820073 , -0.86521593],
       [ 0.10379769,  0.80815856,  2.29692198, ...,  1.35934392,
         2.25538032, -0.8069174 ],
       ...,
       [-0.91175142, -1.23738094, -1.14880556, ..., -0.88158755,
        -0.8820073 , -0.86521593],
       [-0.7555131 ,  0.80815856,  0.22948545, ..., -0.49078421,
        -0.8820073 , -0.48083004],
       [-0.05244064,  0.80815856,  0.22948545, ..., -0.34423296,
         2.25538032, -0.38473356]])
```

```
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"X_train shape: {y_train.shape}")
print(f"X_test shape: {y_test.shape}")
```

```
X_train shape: (24000, 23)
X_test shape: (6000, 23)
X_train shape: (24000,)
X_test shape: (6000,)
```

Logistic Regression Without Hyper-parameter

```
# Model Building
log_regression=LogisticRegression(verbose=False)
```

```
log_regression.fit(X_train, y_train)
y_pred = log_regression.predict(X_test)
```

```
# Model Evaluation
print("=============================================================================================")
print("Accuracy_Score")
print(accuracy_score(y_test, y_pred))
print("=============================================================================================")
print(classification_report(y_test, y_pred))
```

```
=============================================================================================
Accuracy_Score
0.8031666666666667
=============================================================================================
              precision    recall  f1-score   support

           0       0.82      0.96      0.88      4687
           1       0.64      0.23      0.34      1313

    accuracy                           0.80      6000
   macro avg       0.73      0.60      0.61      6000
weighted avg       0.78      0.80      0.76      6000
```

Logistic Regression With Hyper-parameter Tunning Using GridSearchCV

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
param_grid = {
    "penalty": ["l1", "l2","elasticnet"],
    "C" : [0.001, 0.01, 0.1, 1, 10],
    "solver" : ["lbfgs", "liblinear", "newton-cg", "sag", "saga"],
    "max_iter" : [100, 300, 500,1000,1500]
}
```

```
# Perform grid search with cross-validation
grid_search = GridSearchCV(log_regression, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:378: FitFailedWarning:
1000 fits failed out of a total of 1875.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
125 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
125 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
125 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
125 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got elasticnet penalty.

--------------------------------------------------------------------------------
125 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 64, in _check_solver
    raise ValueError(
ValueError: Only 'saga' solver supports elasticnet penalty, got solver=liblinear.

--------------------------------------------------------------------------------
125 fits failed with the following error:
Traceback (most recent call last):
```

```
Traceback (most recent call last):
```

```
# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)

```

```
Best hyperparameters:  {'C': 1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
```

```
# Train the model with the best hyperparameters
best_model = LogisticRegression(**grid_search.best_params_)
best_model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=1, solver='liblinear')
```

```
    raise ValueError(
```

```
# Evaluate the performance on the validation(test) set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy on validation set: {:.2f}".format(accuracy))
```

```
Accuracy on validation set: 0.80
```

```
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1291, in fit
```

Logistic Regression With Hyper-parameter Tunning Using RandomizedSearchCV

```
    File "c:\Users\Tunde\anaconda3\lib\site-packages\joblib\parallel.py", line 1085, in __call__
```

```
# Perform random search with cross-validation
random_search = RandomizedSearchCV(log_regression, param_grid, cv=5, scoring='accuracy', n_jobs=-1, n_iter=20, random_state=42, verbose = False)
random_search.fit(X_train, y_train)

```

```
c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:378: FitFailedWarning:
50 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
10 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
10 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got elasticnet penalty.

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1291, in fit
    fold_coefs_ = Parallel(n_jobs=self.n_jobs, verbose=self.verbose, prefer=prefer)(
  File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\utils\parallel.py", line 63, in __call__
    return super().__call__(iterable_with_config)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\joblib\parallel.py", line 1085, in __call__
    if self.dispatch_one_batch(iterator):
  File "c:\Users\Tunde\anaconda3\lib\site-packages\joblib\parallel.py", line 901, in dispatch_one_batch
    self._dispatch(tasks)
  File "c:\Users\Tunde\anaconda3\lib\site-packages\joblib\parallel.py", line 819, in _dispatch
```

```
# Print the best hyperparameters
print("Best hyperparameters: ", random_search.best_params_)
```

```
Best hyperparameters:  {'solver': 'liblinear', 'penalty': 'l1', 'max_iter': 300, 'C': 10}
```

```
# Train the model with the best hyperparameters
best_model = LogisticRegression(**random_search.best_params_)
best_model.fit(X_train, y_train)
```

LogisticRegression

LogisticRegression(C=10, max_iter=300, penalty='l1', solver='liblinear')

```
# Evaluate the performance on the validation set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy on validation set: {:.2f}".format(accuracy))

```

```
Accuracy on validation set: 0.80
```

```
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got elasticnet penalty.
```

```
1
```

```
5 fits failed with the following error:
```

```
1
```

```
  estimator.fit(X_train, y_train, **fit_params)
```

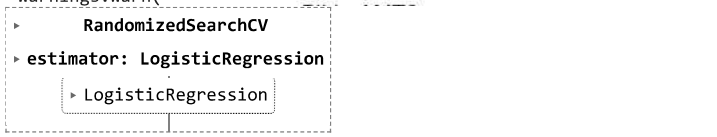
```
1
```

```
File "c:\Users\Tunde\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 64, in _check_solver
```

```
1
```

```
1
```

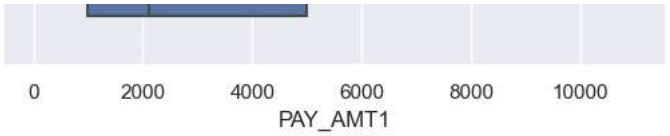
```
0.802           nan        nan 0.80504167 0.80545833 0.79854167
0.805375        nan 0.805375        nan        nan 0.80545833
0.8055          nan]
  warnings.warn(
```

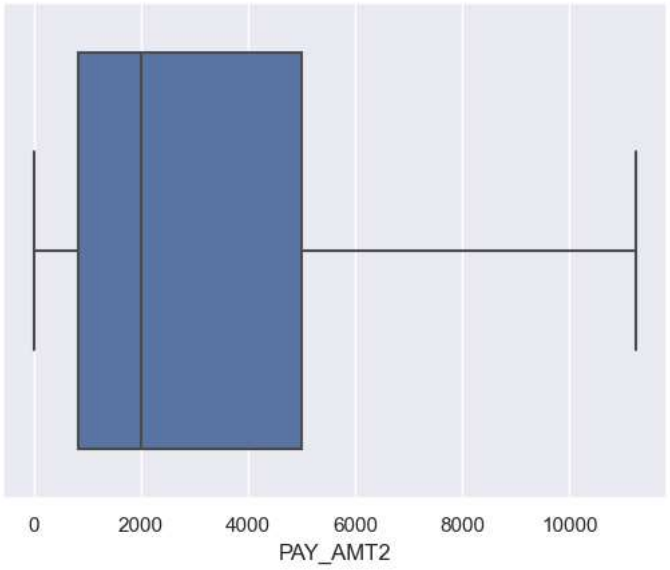


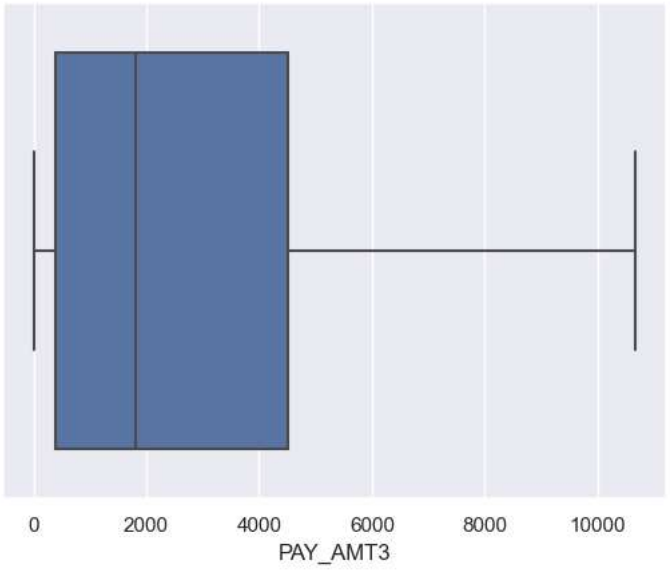
RandomizedSearchCV

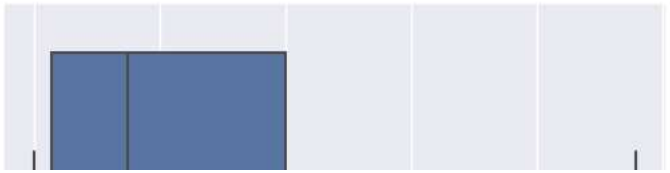
estimator: LogisticRegression

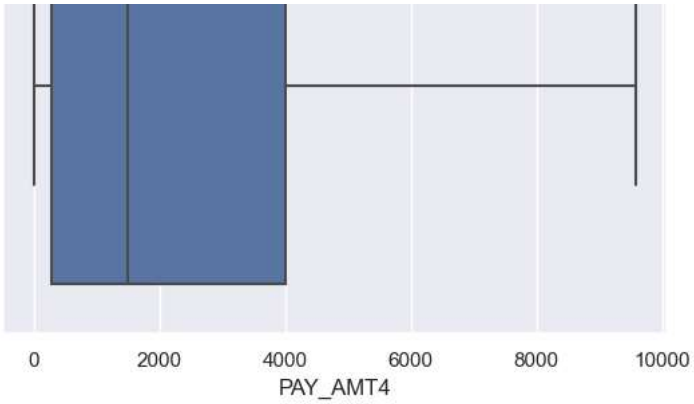
LogisticRegression

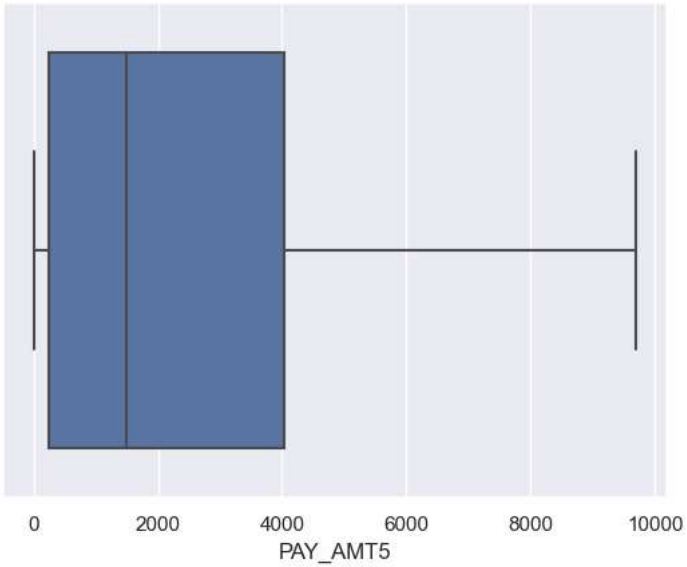

0
2000
4000
6000
8000
10000
PAY_AMT1


0
2000
4000
6000
8000
10000
PAY_AMT2


0
2000
4000
6000
8000
10000
PAY_AMT3




0
2000
4000
6000
8000
10000
PAY_AMT4


0
2000
4000
6000
8000
10000
PAY_AMT5