Data Preprocessing

1. Importing The Required Libraries

In [1]:

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

1.2. Importing The Dataset

In [2]:

```
df=pd.read_excel("default of credit card clients.xls")
df.head(10)
```

Out[2]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5
0	20000	2	2	1	24	2	2	-1	-1	-2
1	120000	2	2	2	26	-1	2	0	0	0
2	90000	2	2	2	34	0	0	0	0	0
3	50000	2	2	1	37	0	0	0	0	0
4	50000	1	2	1	57	-1	0	-1	0	0
5	50000	1	1	2	37	0	0	0	0	0
6	500000	1	1	2	29	0	0	0	0	0
7	100000	2	2	2	23	0	-1	-1	0	0
8	140000	2	3	1	28	0	0	2	0	0
9	20000	1	3	2	35	-2	-2	-2	-2	-1

10 rows × 24 columns

In [3]:

```
1 df.tail()
```

Out[3]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY
29995	220000	1	3	1	39	0	0	0	0	
29996	150000	1	3	2	43	-1	-1	-1	-1	
29997	30000	1	2	2	37	4	3	2	-1	
29998	80000	1	3	1	41	1	-1	0	0	
29999	50000	1	2	1	46	0	0	0	0	

5 rows × 24 columns

1.3. Understanding the Dataset

1. LIMIT_BAL: Continuous:Credit limit of the person
2. SEX: Categorical: 1-male 2-female
3. EDUCATION: Categorical: 1-graduate school 2-university 3-high school 4-others 5-Unknown 6-Unknown
4. MARRIAGE: Categorical: 1-married 2-unmarried 3-others 0-Unknown
5. AGE: Continuous:age of the card holder
6. PAY_0-PAY_6: Categorical:History of the past monthly payment records from April 2005 to September 2005 "0"-No due payment "-1"-One due payment "-2"-Two due payments
7. BILL_AMT1-BILL_AMT6: Continuous:Amount of Bill Statements
8. PAY_AMT1-PAY_AMT6: Continuous:Amount of Previous Payments
9. default payment next month: Categorical:Whether person is going to default or not 1-Default 0-Not Defaul

1.4. Changing The Name of Column

In [4]:

```
1 df.rename(columns={"PAY_0":"PAY_1"},inplace=True)
```

In [5]:

```
1 df.rename(columns={"default payment next month":"Default"},inplace=True)
```

In [6]:

```
1 df.head()
```

Out[6]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5
0	20000	2	2	1	24	2	2	-1	-1	-2
1	120000	2	2	2	26	-1	2	0	0	0
2	90000	2	2	2	34	0	0	0	0	0
3	50000	2	2	1	37	0	0	0	0	0
4	50000	1	2	1	57	-1	0	-1	0	0

5 rows × 24 columns

1.5. Cleaning EDUCATION column

- The variable 'EDUCATION' has three categories with similar inform ation:
- 4: others, 5: unknown, and 6: unknown 0:unknown
- We need to group this three different categaries into one single c ategory

In [7]:

```
1 df["EDUCATION"].unique()
2
```

Out[7]:

```
array([2, 1, 3, 5, 4, 6, 0], dtype=int64)
```

In [8]:

```
1 df["EDUCATION"]=np.where(df["EDUCATION"]==0, 4, df["EDUCATION"])
2 df["EDUCATION"]=np.where(df["EDUCATION"]==5, 4, df["EDUCATION"])
3 df["EDUCATION"]=np.where(df["EDUCATION"]==6, 4, df["EDUCATION"])
```

In [9]:

```
1 df["EDUCATION"].unique()
```

Out[9]:

```
array([2, 1, 3, 4], dtype=int64)
```

1.6. Cleaning MARRIAGE Column

- Similarly, the column 'marriage' should have three categories: 1 = married, 2 = single, 3 = others but it contains a category '0' which will be joined to the category '3'.

In [10]:

```
df["MARRIAGE"]=np.where(df["MARRIAGE"]==0, 3, df["MARRIAGE"])
df["MARRIAGE"].unique()
```

Out[10]:

```
array([1, 2, 3], dtype=int64)
```

In [11]:

```
print("Number of Rows & Columns:",df.shape)
```

```
Number of Rows & Columns: (30000, 24)
```

1.7. Checking The Datatypes

In [12]:

```
df.dtypes
```

Out[12]:

```
LIMIT_BAL    int64
SEX          int64
EDUCATION    int64
MARRIAGE     int64
AGE          int64
PAY_1        int64
PAY_2        int64
PAY_3        int64
PAY_4        int64
PAY_5        int64
PAY_6        int64
BILL_AMT1    int64
BILL_AMT2    int64
BILL_AMT3    int64
BILL_AMT4    int64
BILL_AMT5    int64
BILL_AMT6    int64
PAY_AMT1     int64
PAY_AMT2     int64
PAY_AMT3     int64
PAY_AMT4     int64
PAY_AMT5     int64
PAY_AMT6     int64
Default      int64
dtype: object
```

1.8. Checking the NULL Values

In [13]:

```
1 df.isnull().sum()
```

Out[13]:

```
LIMIT_BAL    0
SEX          0
EDUCATION    0
MARRIAGE     0
AGE          0
PAY_1        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      0
dtype: int64
```

In [14]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 24 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   LIMIT_BAL  30000 non-null  int64
 1   SEX        30000 non-null  int64
 2   EDUCATION  30000 non-null  int64
 3   MARRIAGE   30000 non-null  int64
 4   AGE        30000 non-null  int64
 5   PAY_1      30000 non-null  int64
 6   PAY_2      30000 non-null  int64
 7   PAY_3      30000 non-null  int64
 8   PAY_4      30000 non-null  int64
 9   PAY_5      30000 non-null  int64
 10  PAY_6      30000 non-null  int64
 11  BILL_AMT1  30000 non-null  int64
 12  BILL_AMT2  30000 non-null  int64
 13  BILL_AMT3  30000 non-null  int64
 14  BILL_AMT4  30000 non-null  int64
 15  BILL_AMT5  30000 non-null  int64
 16  BILL_AMT6  30000 non-null  int64
 17  PAY_AMT1   30000 non-null  int64
 18  PAY_AMT2   30000 non-null  int64
 19  PAY_AMT3   30000 non-null  int64
 20  PAY_AMT4   30000 non-null  int64
 21  PAY_AMT5   30000 non-null  int64
 22  PAY_AMT6   30000 non-null  int64
 23  Default    30000 non-null  int64
dtypes: int64(24)
memory usage: 5.5 MB
```

1.9. Understanding the

Minimum,Maximum,Mean,Median,Mode,Quartiles

In [15]:

```
1 df.describe()
```

Out[15]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.0000
mean	167484.322667	1.603733	1.842267	1.557267	35.485500	-0.0167
std	129747.661567	0.489129	0.744494	0.521405	9.217904	1.1238
min	10000.000000	1.000000	1.000000	1.000000	21.000000	-2.0000
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.0000
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.0000
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.0000
max	1000000.000000	2.000000	4.000000	3.000000	79.000000	8.0000

8 rows × 24 columns

1.10. Understanding the Correlation Between The Dataset

In [16]:

```
1 df.corr()
```

Out[16]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2
LIMIT_BAL	1.000000	0.024755	-0.231088	-0.111012	0.144713	-0.271214	-0.296382
SEX	0.024755	1.000000	0.013627	-0.028522	-0.090874	-0.057643	-0.070771
EDUCATION	-0.231088	0.013627	1.000000	-0.136797	0.182434	0.112593	0.129703
MARRIAGE	-0.111012	-0.028522	-0.136797	1.000000	-0.412001	0.018557	0.023620
AGE	0.144713	-0.090874	0.182434	-0.412001	1.000000	-0.039447	-0.050148
PAY_1	-0.271214	-0.057643	0.112593	0.018557	-0.039447	1.000000	0.672164
PAY_2	-0.296382	-0.070771	0.129703	0.023620	-0.050148	0.672164	1.000000
PAY_3	-0.286123	-0.066096	0.122425	0.032399	-0.053048	0.574245	0.766552
PAY_4	-0.267460	-0.060173	0.116531	0.031831	-0.049722	0.538841	0.662067
PAY_5	-0.249411	-0.055064	0.104088	0.034377	-0.053826	0.509426	0.622780
PAY_6	-0.235195	-0.044008	0.088986	0.033168	-0.048773	0.474553	0.575501
BILL_AMT1	0.285430	-0.033642	0.016597	-0.027832	0.056239	0.187068	0.234887
BILL_AMT2	0.278314	-0.031183	0.011980	-0.025294	0.054283	0.189859	0.235257
BILL_AMT3	0.283236	-0.024563	0.006714	-0.029082	0.053710	0.179785	0.224146
BILL_AMT4	0.293988	-0.021880	-0.006131	-0.027274	0.051353	0.179125	0.222237
BILL_AMT5	0.295562	-0.017005	-0.012439	-0.029270	0.049345	0.180635	0.221348
BILL_AMT6	0.290389	-0.016733	-0.012646	-0.025066	0.047613	0.176980	0.219403
PAY_AMT1	0.195236	-0.000242	-0.041088	-0.004653	0.026147	-0.079269	-0.080701
PAY_AMT2	0.178408	-0.001391	-0.032793	-0.009513	0.021785	-0.070101	-0.058990
PAY_AMT3	0.210167	-0.008597	-0.044293	-0.004250	0.029247	-0.070561	-0.055901
PAY_AMT4	0.203242	-0.002229	-0.040949	-0.013970	0.021379	-0.064005	-0.046858
PAY_AMT5	0.217202	-0.001667	-0.045138	-0.003019	0.022850	-0.058190	-0.037093
PAY_AMT6	0.219595	-0.002766	-0.044061	-0.008383	0.019478	-0.058673	-0.036500
Default	-0.153520	-0.039961	0.033842	-0.027575	0.013890	0.324794	0.263551

24 rows × 24 columns

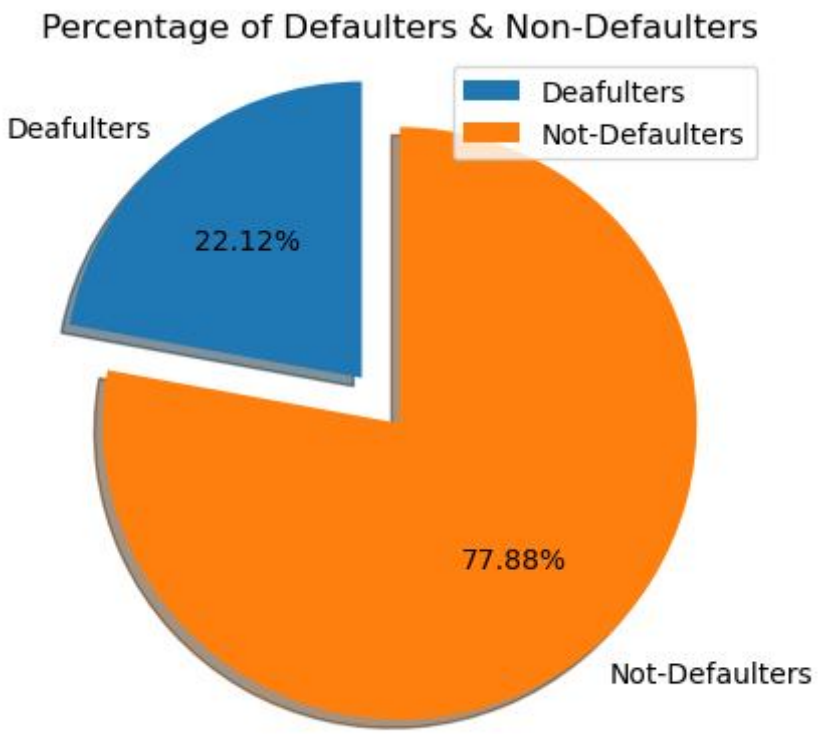
Part2-EDA(Exploratory Data Analysis)

2.1. Finding The Total Defaulters

In [17]:

```
#Count of Defaulters
defaulting=df.Default.sum()
not_defaulting=len(df)-defaulting
print("Total Number of Defaulters:",defaulting)
print("Total Number of Non-Defaulters:",not_defaulting)
print("-------------------------------------------------------")
#Percentage of Defaulters
deafulter_Perc=defaulting/len(df)*100
Nondeafulter_Perc=not_defaulting/len(df)*100
print("Total Percent of Defaulters:",deafulter_Perc,"%")
print("Total Percenr of Non-Defaulters:",Nondeafulter_Perc,"%")
print("-------------------------------------------------------")
#Plotting the Piechart
pie1=np.array([deafulter_Perc,Nondeafulter_Perc])
mylabels=["Deafulters","Not-Defaulters"]
myexplode=[0.2,0]
plt.pie(pie1,labels=mylabels,shadow=True,startangle = 90,explode=myexplode,autopct=
plt.legend()
plt.title("Percentage of Defaulters & Non-Defaulters")
plt.show()
```

```
Total Number of Defaulters: 6636
Total Number of Non-Defaulters: 23364
-------------------------------------------------------
Total Percent of Defaulters: 22.12 %
Total Percenr of Non-Defaulters: 77.88000000000001 %
-------------------------------------------------------
```



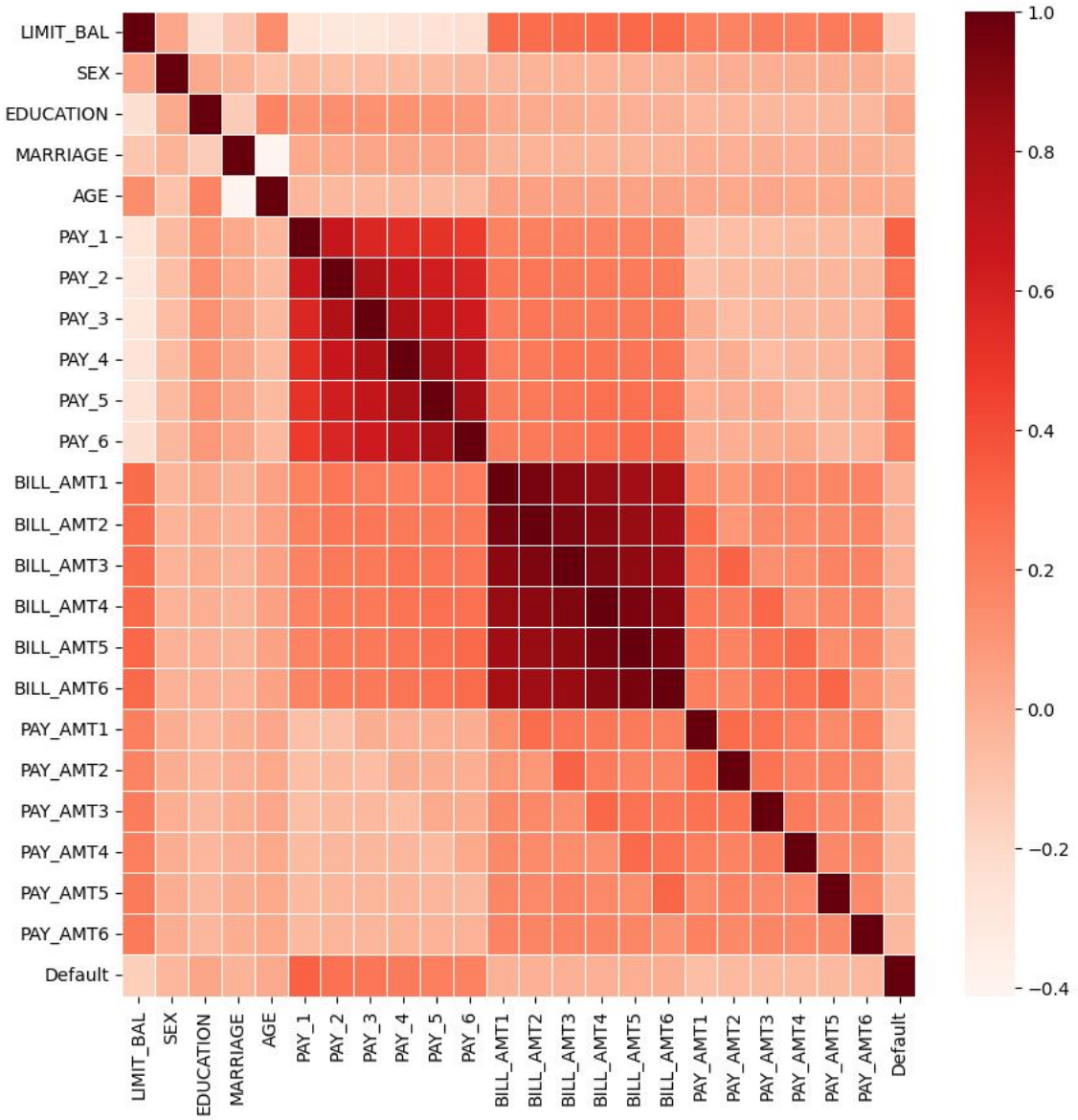
2.2. Plotting The Heatmap to Understand The Correlation Between Cloumns

In [18]:

```
plt.figure(figsize=(10,10))
corr = df.corr()
sns.heatmap(corr, cmap="Reds",linewidths=.5)
```

Out[18]:

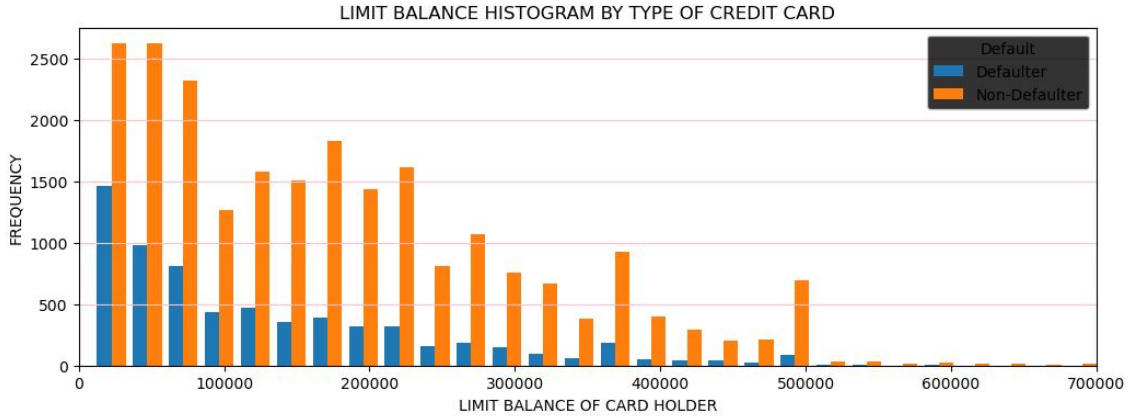
<AxesSubplot:>



2.3. Finding The Defaulters Based On Limit Balance Of the Card Holder

In [19]:

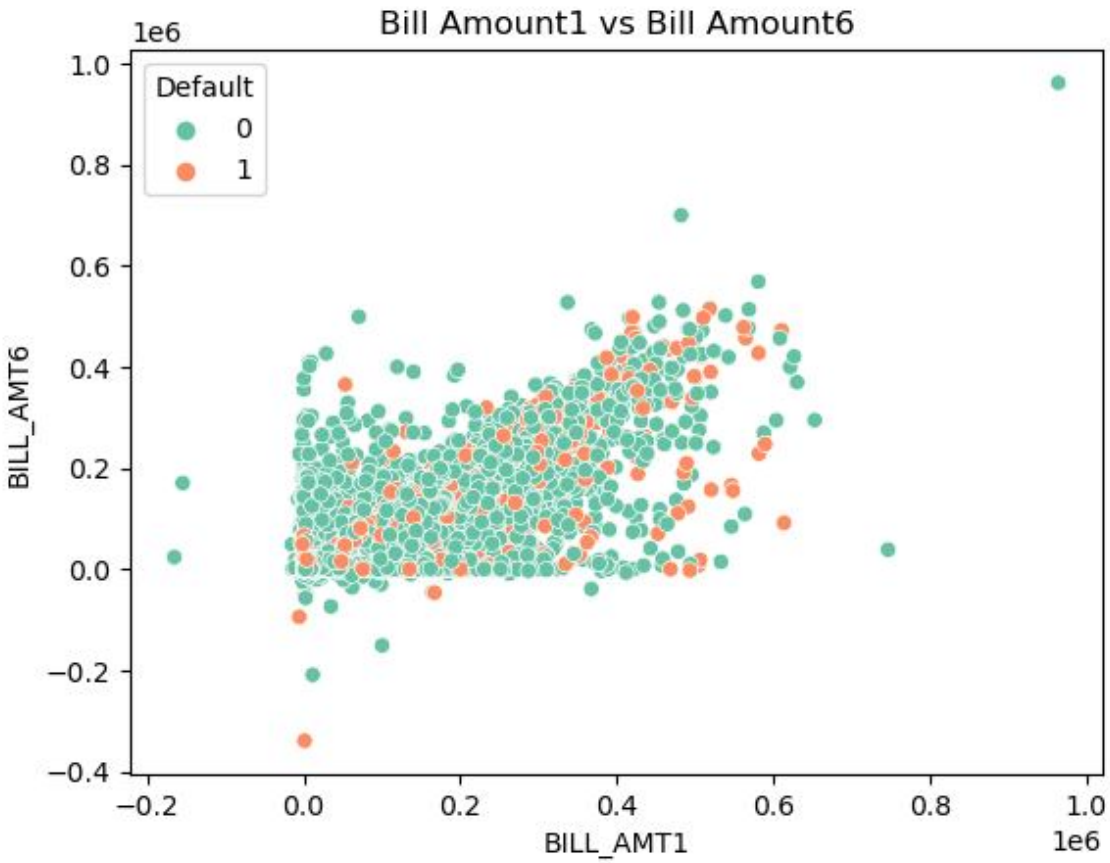
```
x1 = list(df[df['Default'] == 1]['LIMIT_BAL'])
x2 = list(df[df['Default'] == 0]['LIMIT_BAL'])
plt.figure(figsize=(12,4))
plt.hist([x1,x2],bins=40)
plt.xlim([0,700000])
plt.legend(['Defaulter', 'Non-Defaulter'], title = 'Default', loc='upper right', fa
plt.grid(color="pink",axis="y")
plt.xlabel("LIMIT BALANCE OF CARD HOLDER")
plt.ylabel("FREQUENCY")
plt.title("LIMIT BALANCE HISTOGRAM BY TYPE OF CREDIT CARD")
plt.show()
```



2.4. Finding The Linear Realtionship Between The indepedent Variables Using The ScatterPlot

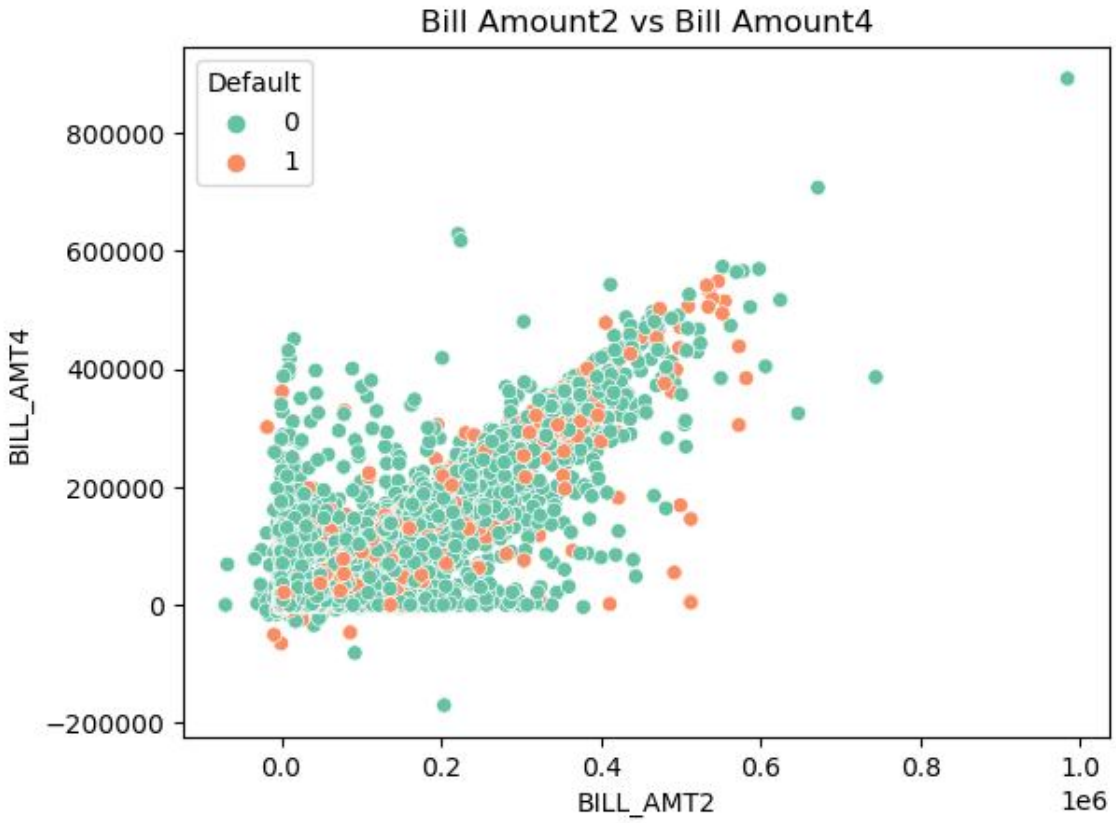
In [20]:

```
sns.scatterplot(data=df,x=df["BILL_AMT1"],y=df["BILL_AMT6"],hue="Default",
palette="Set2")
plt.title("Bill Amount1 vs Bill Amount6")
plt.show()
```



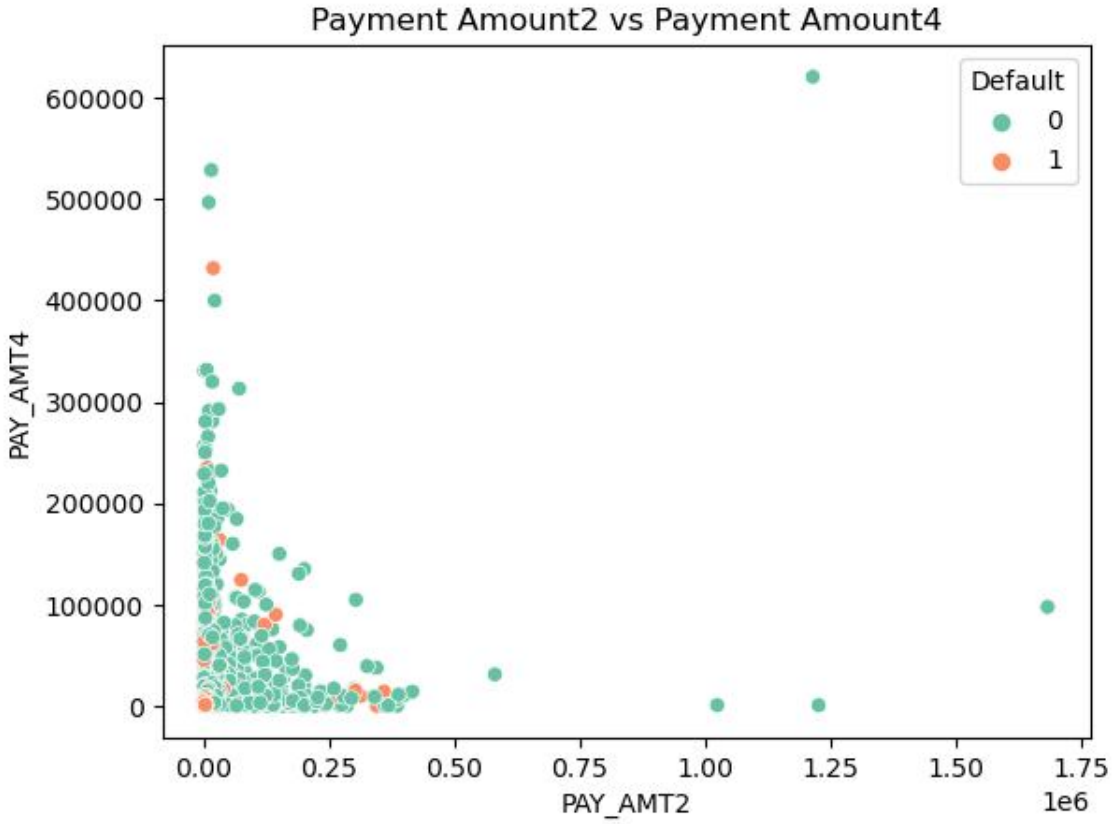
In [21]:

```
sns.scatterplot(data=df,x=df["BILL_AMT2"],y=df["BILL_AMT4"],hue="Default",
palette="Set2")
plt.title("Bill Amount2 vs Bill Amount4")
plt.show()
```



In [22]:

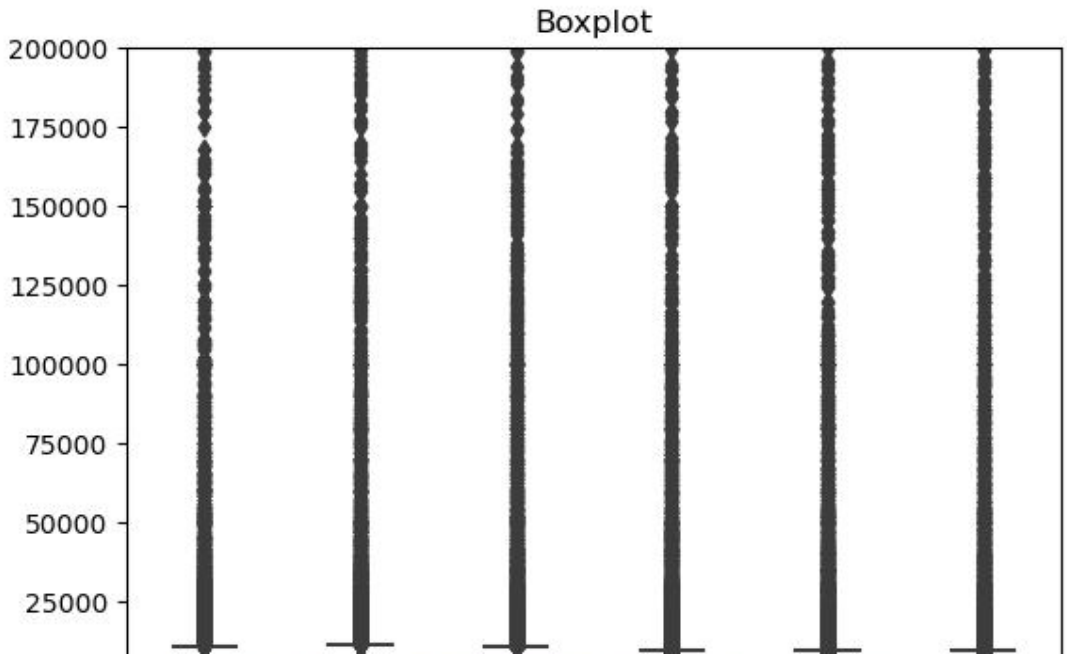
```
sns.scatterplot(data=df,x=df["PAY_AMT2"],y=df["PAY_AMT4"],hue="Default",palette="Se
plt.title("Payment Amount2 vs Payment Amount4")
plt.show()
```



2.5. Plotting The BoxPlots Using Different Variables

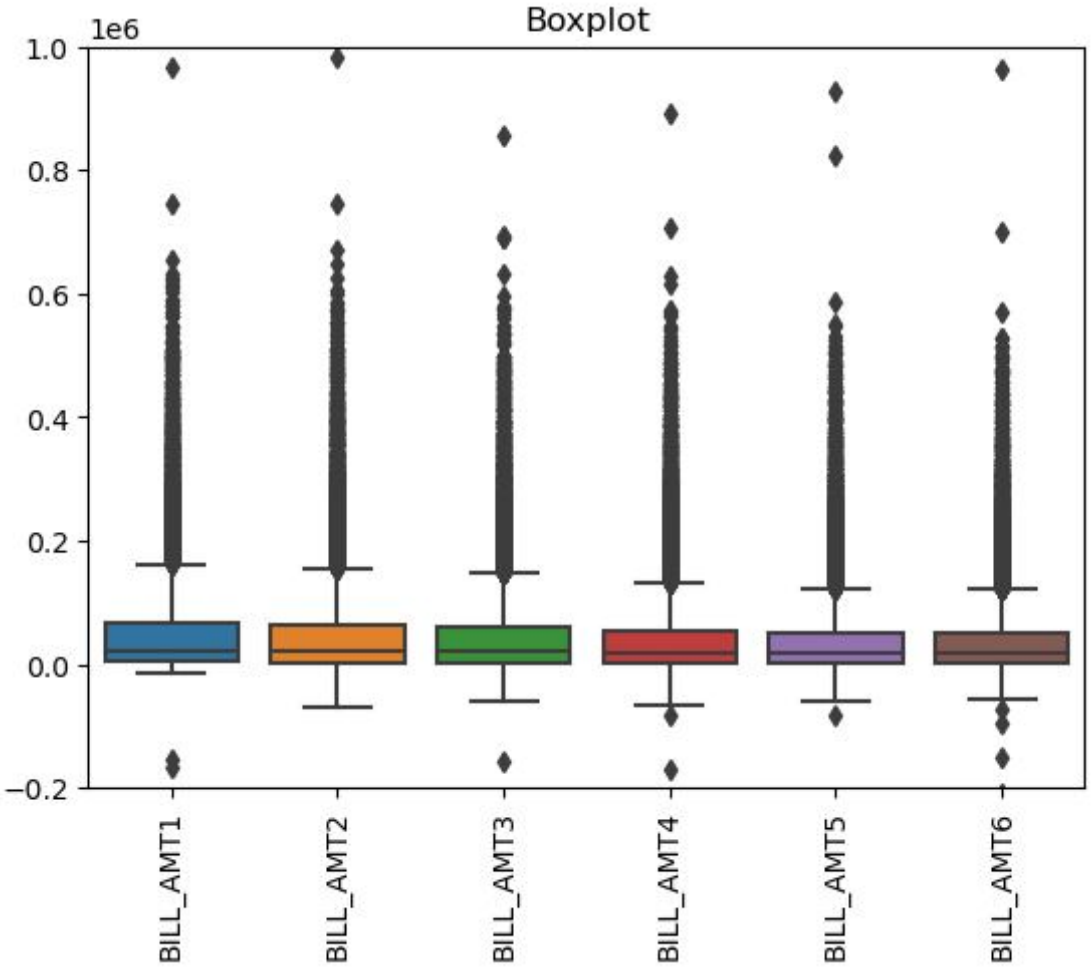
In [23]:

```
df1=df[["PAY_AMT1","PAY_AMT2","PAY_AMT3","PAY_AMT4","PAY_AMT5","PAY_AMT6"]]
sns.boxplot(data=df1)
plt.ylim([-20000,200000])
plt.title("Boxplot")
plt.xticks(rotation=90)
plt.show()
```



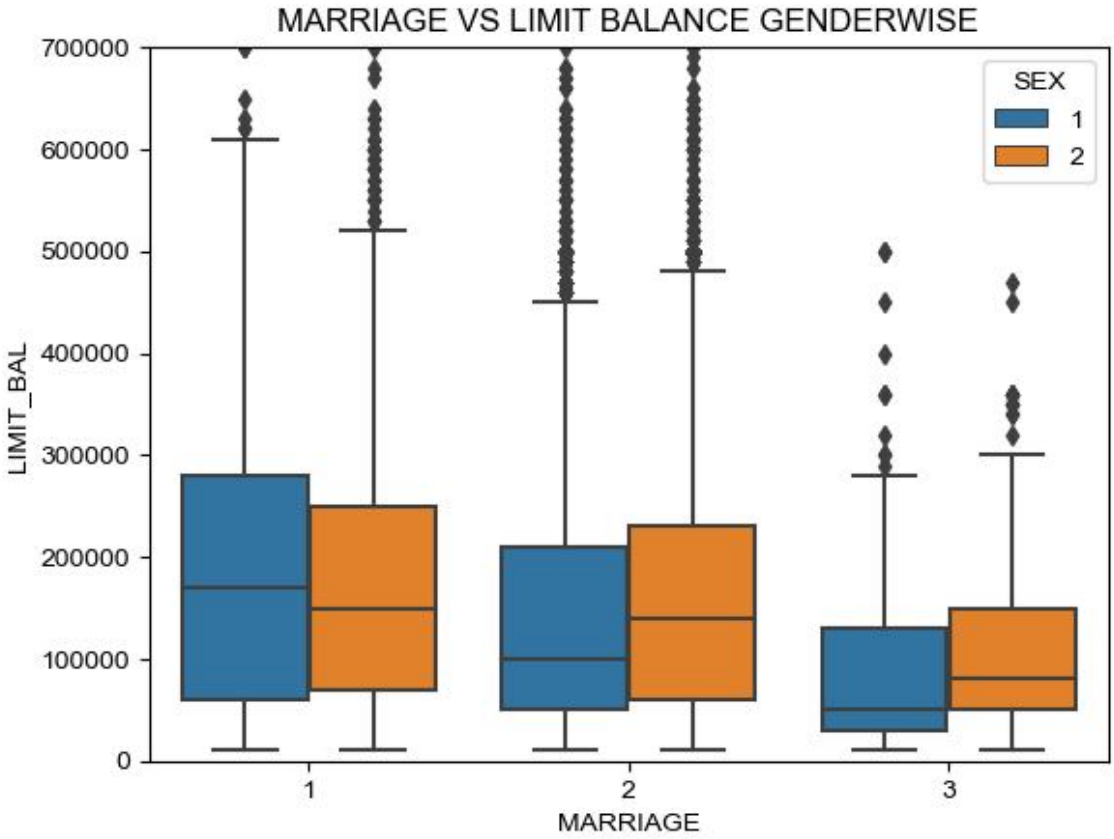
In [24]:

```
df2=df[["BILL_AMT1","BILL_AMT2","BILL_AMT3","BILL_AMT4","BILL_AMT5","BILL_AMT6"]]
sns.boxplot(data=df2)
plt.ylim([-200000,1000000])
plt.title("Boxplot")
plt.xticks(rotation=90)
plt.show()
```



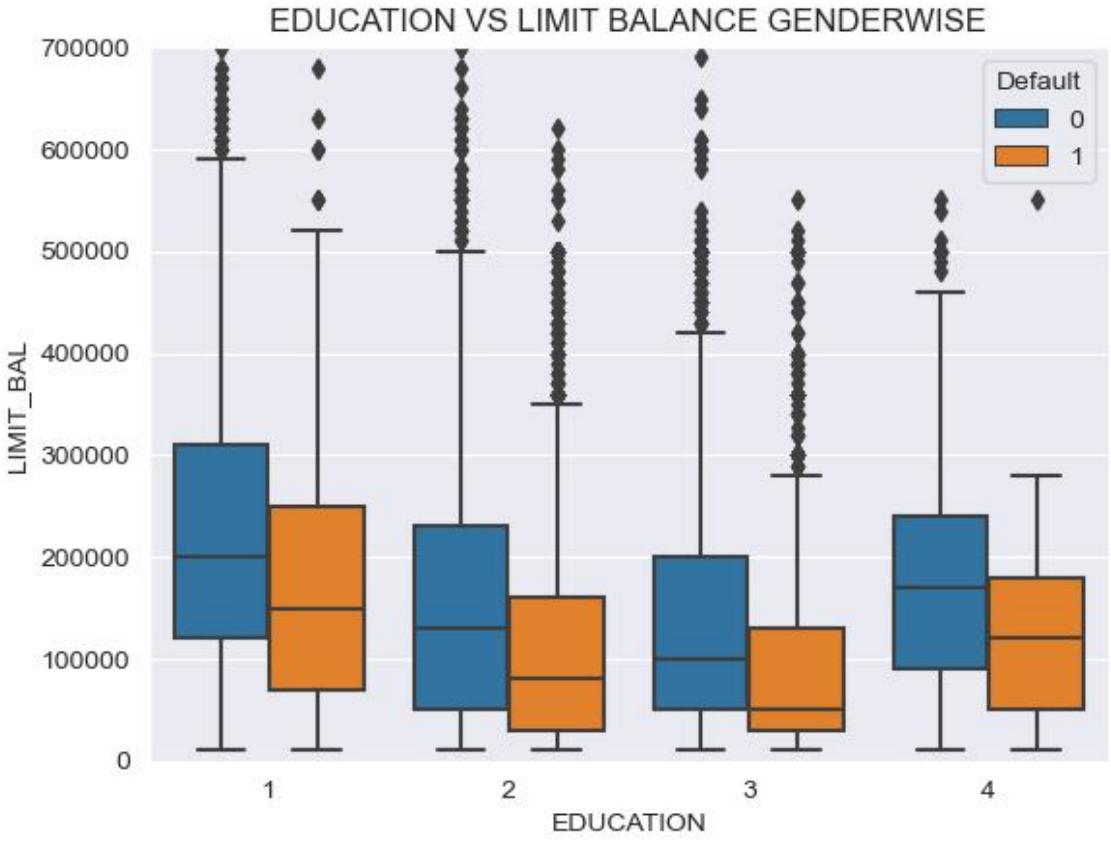
In [25]:

```
sns.boxplot(data=df,x="MARRIAGE",y="LIMIT_BAL",hue="SEX")
sns.set_style("darkgrid")
plt.ylim([0,700000])
plt.title("MARRIAGE VS LIMIT BALANCE GENDERWISE")
plt.show()
```



In [26]:

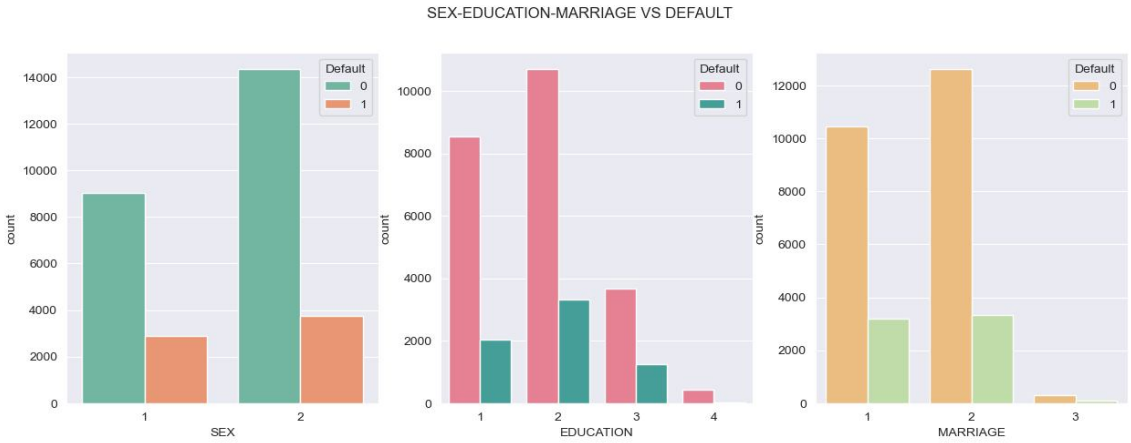
```
sns.boxplot(data=df,x="EDUCATION",y="LIMIT_BAL",hue="Default")
sns.set_style("darkgrid")
plt.ylim([0,700000])
plt.title("EDUCATION VS LIMIT BALANCE GENDERWISE")
plt.show()
```



2.6. Plotting The Bar Chart To Find Defaulters Based On Sex,Education,Marriage

In [27]:

```
f, axes = plt.subplots(1, 3, figsize=(15,5))
sns.countplot(data=df,x="SEX",hue="Default",ax=axes[0],palette="Set2")
sns.countplot(data=df,x="EDUCATION",hue="Default",ax=axes[1],palette="husl")
sns.countplot(data=df,x="MARRIAGE",hue="Default",ax=axes[2],palette="Spectral")
plt.suptitle("SEX-EDUCATION-MARRIAGE VS DEFAULT")
plt.show()
```



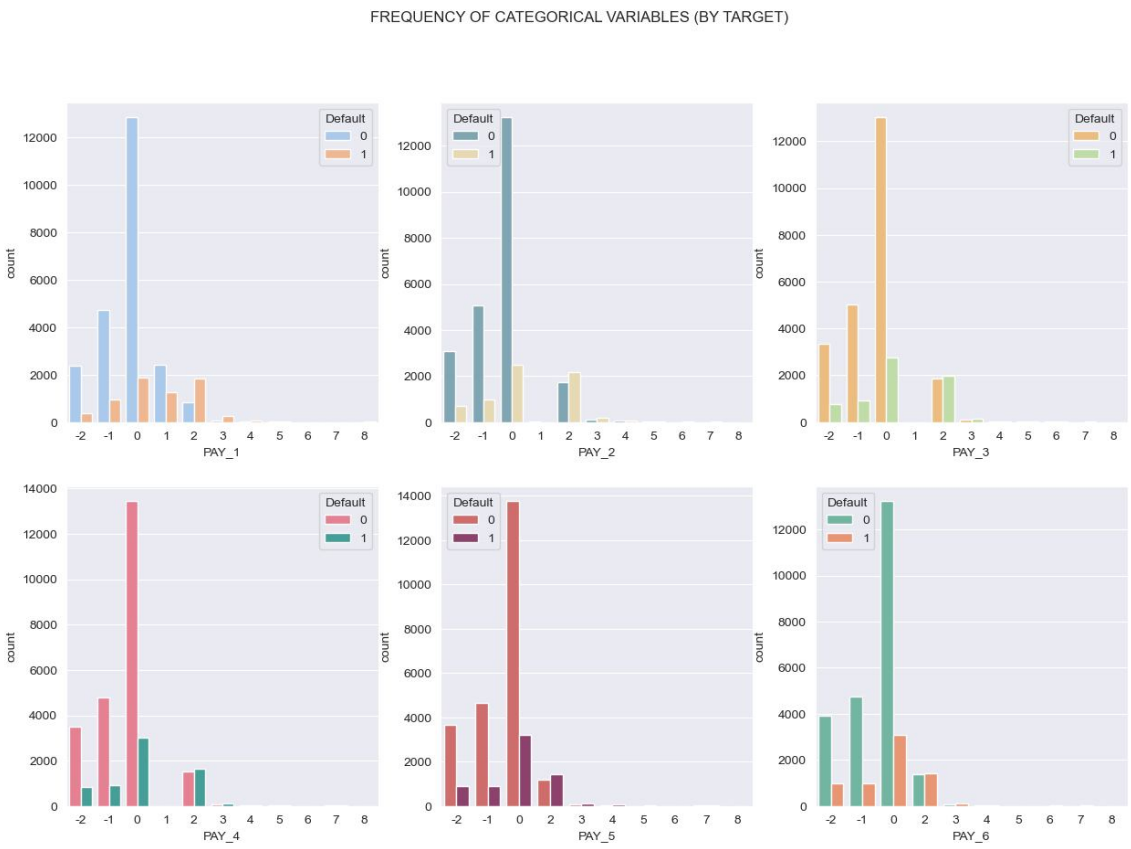
2.7. Plotting The Bar Charts To Find Out The Defaulters Based On The PAYMENT

In [28]:

```
f, axes = plt.subplots(2, 3, figsize=(15, 10))
f.suptitle('FREQUENCY OF CATEGORICAL VARIABLES (BY TARGET)')
sns.countplot(x="PAY_1", hue="Default", data=df, palette="pastel", ax=axes[0,0])
sns.countplot(x="PAY_2", hue="Default", data=df, palette="blend:#7AB,#EDA", ax=axes
sns.countplot(x="PAY_3", hue="Default", data=df, palette="Spectral", ax=axes[0,2])
sns.countplot(x="PAY_4", hue="Default", data=df, palette="husl", ax=axes[1,0])
sns.countplot(x="PAY_5", hue="Default", data=df, palette="flare", ax=axes[1,1])
sns.countplot(x="PAY_6", hue="Default", data=df, palette="Set2", ax=axes[1,2])
```

Out[28]:

```
<AxesSubplot:xlabel='PAY_6', ylabel='count'>
```



Part3-Scaling The Data Using StandardScaler

In [29]:

```
x=df.drop(labels=["Default"],axis=1)
y=df["Default"]
```

In [30]:

```
1 x.head()
```

Out[30]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5
0	20000	2	2	1	24	2	2	-1	-1	-2
1	120000	2	2	2	26	-1	2	0	0	0
2	90000	2	2	2	34	0	0	0	0	0
3	50000	2	2	1	37	0	0	0	0	0
4	50000	1	2	1	57	-1	0	-1	0	0

5 rows × 23 columns

In [31]:

```
1 y.head()
```

Out[31]:

```
0    1
1    1
2    0
3    0
4    0
Name: Default, dtype: int64
```

3.2. Splitting the Dataset into Training Dataset and Testing Dataset

In [32]:

```
1 from sklearn.model_selection import train_test_split
2 x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7,random_state=0)
```

In [33]:

```
1 x_train
```

Out[33]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY
15925	30000	2	2	2	31	-2	-2	-2	-2	
10062	200000	1	2	1	31	1	2	2	0	
19376	500000	2	4	2	38	-2	-2	-2	-2	
6384	20000	2	2	2	22	0	0	0	0	
15976	50000	2	2	1	48	1	2	0	0	
...	...	...	...	...	...	...	...	...	...	
13123	30000	1	2	2	38	0	0	0	0	
19648	210000	2	1	1	33	0	0	0	0	
9845	130000	2	3	1	43	0	0	0	0	
10799	50000	2	3	1	29	0	0	0	0	
2732	140000	2	1	2	25	0	0	0	0	

21000 rows × 23 columns

In [34]:

```
1 x_test
```

Out[34]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY
8225	20000	1	1	2	33	1	2	2	2	
10794	20000	2	2	2	35	0	0	2	0	
9163	230000	2	1	1	44	1	-1	-1	-1	
26591	100000	1	2	1	42	0	0	0	0	
6631	150000	1	1	2	29	-2	-2	-2	-2	
...	...	...	...	...	...	...	...	...	...	
21914	130000	2	1	2	23	0	0	0	0	
17453	320000	2	1	1	36	-1	2	0	0	
20344	120000	2	3	1	63	0	0	0	0	
1878	300000	2	1	1	31	-2	-2	-1	0	
6465	330000	2	1	2	34	0	0	0	0	

9000 rows × 23 columns

In [35]:

```
1 y_train
```

Out[35]:

```
15925    0
10062    0
19376    0
6384     0
15976    1
        ..
13123    0
19648    1
9845     0
10799    0
2732     0
Name: Default, Length: 21000, dtype: int64
```

In [36]:

```
1 y_test
```

Out[36]:

```
8225     0
10794    0
9163     0
26591    0
6631     0
        ..
21914    0
17453    0
20344    0
1878     0
6465     0
Name: Default, Length: 9000, dtype: int64
```

3.3. Scaling the Training Dataset Using StandardScaler

In [37]:

```
1 from sklearn.preprocessing import StandardScaler
2 train_scaler=StandardScaler()
3 test_scaler=StandardScaler()
```

In [38]:

```
scaled_train_data=train_scaler.fit_transform(x_train)
scaled_train_data=pd.DataFrame(data=scaled_train_data,columns=x_train.columns,index
scaled_train_data
```

Out[38]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	F
15925	-1.063195	0.817793	0.212872	0.848088	-0.485236	-1.756704	-1.554090	-1.5
10062	0.253364	-1.222803	0.212872	-1.071904	-0.485236	0.890707	1.756174	1.7
19376	2.576705	0.817793	2.900159	0.848088	0.276931	-1.756704	-1.554090	-1.5
6384	-1.140640	0.817793	0.212872	0.848088	-1.465165	0.008236	0.101042	0.1
15976	-0.908306	0.817793	0.212872	-1.071904	1.365741	0.890707	1.756174	0.1
...	...	...	...	...	...	...	...	
13123	-1.063195	-1.222803	0.212872	0.848088	0.276931	0.008236	0.101042	0.1
19648	0.330809	0.817793	-1.130772	-1.071904	-0.267474	0.008236	0.101042	0.1
9845	-0.288748	0.817793	1.556515	-1.071904	0.821336	0.008236	0.101042	0.1
10799	-0.908306	0.817793	1.556515	-1.071904	-0.702998	0.008236	0.101042	0.1
2732	-0.211304	0.817793	-1.130772	0.848088	-1.138522	0.008236	0.101042	0.1

21000 rows × 23 columns

3.4. Scaling the Testing Dataset StandardScaler

In [39]:

```
pd.set_option("display.max_columns",30)
scaled_test_data=test_scaler.fit_transform(x_test)
scaled_test_data=pd.DataFrame(data=scaled_test_data,columns=x_test.columns,index=x_
scaled_test_data
```

Out[39]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PA
8225	-1.127827	-1.261811	-1.132687	0.851570	-0.274667	0.93875	1.846738	1.86
10794	-1.127827	0.792512	0.209536	0.851570	-0.059495	0.03077	0.137587	1.86
9163	0.473002	0.792512	-1.132687	-1.061593	0.908779	0.93875	-0.716989	-0.69
26591	-0.517987	-1.261811	0.209536	-1.061593	0.693607	0.03077	0.137587	0.16
6631	-0.136838	-1.261811	-1.132687	0.851570	-0.705011	-1.78519	-1.571565	-1.54
...	...	...	...	...	...	...	...	
21914	-0.289298	0.792512	-1.132687	0.851570	-1.350527	0.03077	0.137587	0.16
17453	1.159071	0.792512	-1.132687	-1.061593	0.048091	-0.87721	1.846738	0.16
20344	-0.365528	0.792512	1.551759	-1.061593	2.952913	0.03077	0.137587	0.16
1878	1.006611	0.792512	-1.132687	-1.061593	-0.489839	-1.78519	-1.571565	-0.69
6465	1.235301	0.792512	-1.132687	0.851570	-0.167081	0.03077	0.137587	0.16

9000 rows × 23 columns

Part4-Feature Selection Using mutual_info_classifier

In [40]:

```
from sklearn.feature_selection import mutual_info_classif
info_gain1=mutual_info_classif(scaled_train_data,y_train)
info_gain1
```

Out[40]:

```
array([0.0161528 , 0.00749225, 0.00320401, 0.00295033, 0.00126261,
       0.07647725, 0.0533795 , 0.04053104, 0.03300315, 0.0357061 ,
       0.03248806, 0.00828689, 0.00695372, 0.00231263, 0.00523654,
       0.00729609, 0.00797595, 0.02302428, 0.01573049, 0.01875587,
       0.01398189, 0.01185536, 0.01303627])
```

In [41]:

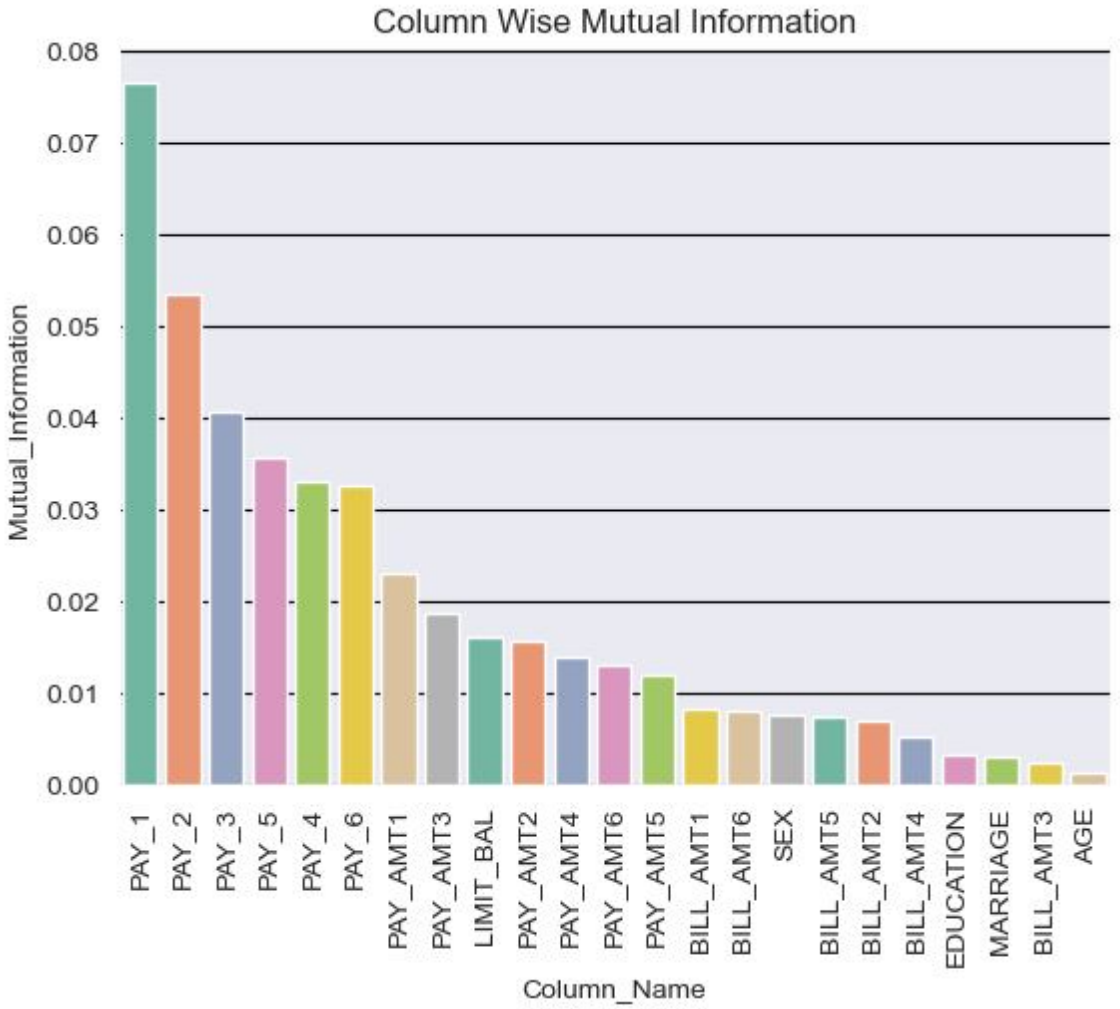
```
data={"Column_Name":x_train.columns,"Mutual_Information":info_gain1}
gain1=pd.DataFrame(data)
gain1
```

Out[41]:

	Column_Name	Mutual_Information
0	LIMIT_BAL	0.016153
1	SEX	0.007492
2	EDUCATION	0.003204
3	MARRIAGE	0.002950
4	AGE	0.001263
5	PAY_1	0.076477
6	PAY_2	0.053380
7	PAY_3	0.040531
8	PAY_4	0.033003
9	PAY_5	0.035706
10	PAY_6	0.032488
11	BILL_AMT1	0.008287
12	BILL_AMT2	0.006954
13	BILL_AMT3	0.002313
14	BILL_AMT4	0.005237
15	BILL_AMT5	0.007296
16	BILL_AMT6	0.007976
17	PAY_AMT1	0.023024
18	PAY_AMT2	0.015730
19	PAY_AMT3	0.018756
20	PAY_AMT4	0.013982
21	PAY_AMT5	0.011855
22	PAY_AMT6	0.013036

In [42]:

```
sns.barplot(data=gain1,x="Column_Name",y="Mutual_Information",palette="Set2",order=
plt.xticks(rotation=90)
plt.title("Column Wise Mutual Information")
plt.grid(color="black",axis="y")
plt.show()
```



4.2. Selecting the Top-11 Features for the Model Building

In [43]:

```
from sklearn.feature_selection import SelectKBest
eleven_col=SelectKBest(mutual_info_classif,k=11)
eleven_col.fit(scaled_train_data,y_train)
scaled_train_data.columns[eleven_col.get_support()]
```

Out[43]:

```
Index(['LIMIT_BAL', 'PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6',
       'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT5'],
      dtype='object')
```

4.3. Training Dataset

In [44]:

```
scaled_train_fs_data=eleven_col.transform(scaled_train_data)
scaled_train_fs_data=pd.DataFrame(scaled_train_fs_data,columns=scaled_train_data.co
scaled_train_fs_data

```

Out[44]:

	LIMIT_BAL	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	PAY_AM
15925	-1.063195	-1.756704	-1.554090	-1.526818	-1.523691	-1.532501	-1.487937	-0.3345
10062	0.253364	0.890707	1.756174	1.785922	0.180389	0.227164	0.244044	-0.1560
19376	2.576705	-1.756704	-1.554090	-1.526818	-1.523691	-1.532501	-1.487937	-0.2006
6384	-1.140640	0.008236	0.101042	0.129552	0.180389	0.227164	0.244044	-0.2578
15976	-0.908306	0.890707	1.756174	0.129552	0.180389	0.227164	0.244044	-0.3345
...	...	...	...	...	...	...	...	
13123	-1.063195	0.008236	0.101042	0.129552	0.180389	0.227164	0.244044	-0.2104
19648	0.330809	0.008236	0.101042	0.129552	0.180389	0.227164	0.244044	0.0792
9845	-0.288748	0.008236	0.101042	0.129552	0.180389	0.227164	1.976026	-0.0389
10799	-0.908306	0.008236	0.101042	0.129552	0.180389	0.227164	0.244044	-0.1867
2732	-0.211304	0.008236	0.101042	0.129552	0.180389	0.227164	0.244044	-0.2163

21000 rows × 11 columns

4.4. Testing Dataset

In [45]:

```
scaled_test_fs_data=eleven_col.transform(scaled_test_data)
scaled_test_fs_data=pd.DataFrame(scaled_test_fs_data,columns=scaled_test_data.colum
scaled_test_fs_data
```

Out[45]:

	LIMIT_BAL	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	PAY_AMT
8225	-1.127827	0.93875	1.846738	1.868711	1.935378	2.031109	2.031522	-0.36124
10794	-1.127827	0.03077	0.137587	1.868711	0.208561	0.253247	0.274741	-0.14471
9163	0.473002	0.93875	-0.716989	-0.692337	-0.654847	-0.635684	-1.482040	-0.30080
26591	-0.517987	0.03077	0.137587	0.161346	0.208561	0.253247	0.274741	-0.04282
6631	-0.136838	-1.78519	-1.571565	-1.546019	-1.518255	-1.524615	-1.482040	0.08384
...	...	...	...	...	...	...	...	...
21914	-0.289298	0.03077	0.137587	0.161346	0.208561	0.253247	0.274741	-0.07466
17453	1.159071	-0.87721	1.846738	0.161346	0.208561	0.253247	0.274741	-0.36124
20344	-0.365528	0.03077	0.137587	0.161346	0.208561	0.253247	0.274741	-0.17019
1878	1.006611	-1.78519	-1.571565	-0.692337	0.208561	0.253247	0.274741	-0.29055
6465	1.235301	0.03077	0.137587	0.161346	0.208561	0.253247	0.274741	0.59400

9000 rows × 11 columns

Part5-Applying Machine Learning Algorithms

5.1. Logistic Regression

In [46]:

```
import sys
from sklearn.linear_model import LogisticRegression
model_lr=LogisticRegression()
model_lr.fit(scaled_train_fs_data,y_train)
y_pred_logreg=model_lr.predict(scaled_test_fs_data)
```

In [47]:

```
info={"Actual Value":y_test,
 "Predicted Value":y_pred_logreg}
logreg=pd.DataFrame(info)
logreg.tail(18)
```

Out[47]:

	Actual Value	Predicted Value
7553	0	0
12063	0	0
11607	0	0
22038	0	0
26598	1	0
14600	0	0
20945	0	0
1310	1	1
28352	0	0
23682	0	0
21383	0	0
675	1	1
21313	0	0
21914	0	0
17453	0	0
20344	0	0
1878	0	0
6465	0	0

5.2. SVM(Support Vector Machine)

In [48]:

```
from sklearn.svm import SVC
model_svm=SVC()
model_svm.fit(scaled_train_fs_data,y_train)
y_pred_svm=model_svm.predict(scaled_test_fs_data)
```

In [49]:

```
info={"Actual Value":y_test,"Predicted Value":y_pred_svm}
svm=pd.DataFrame(info)
svm.sample(18)
```

Out[49]:

	Actual Value	Predicted Value
17896	1	0
4301	1	0
3161	0	0
8163	1	1
9512	0	0
11031	1	0
14499	0	1
21121	0	0
10053	1	0
1845	1	1
25550	0	0
15146	1	1
19438	0	0
9562	1	1
9763	0	0
23132	0	0
19581	0	0
19148	0	0

5.3. Naive Bayes

In [50]:

```
from sklearn.naive_bayes import GaussianNB
model_navbay=GaussianNB()
model_navbay.fit(scaled_train_fs_data,y_train)
y_pred_navbay=model_navbay.predict(scaled_test_fs_data)
```

In [51]:

```
info={"Actual Value":y_test,"Predicted Value":y_pred_navbay}
NB=pd.DataFrame(info)
NB.sample(18)
```

Out[51]:

	Actual Value	Predicted Value
13008	0	0
3645	0	0
8346	0	0
21737	0	0
24522	0	0
20784	0	0
10837	0	0
16848	0	0
10208	0	1
18597	0	0
27003	0	1
14910	0	0
19643	0	0
19305	0	0
17228	0	0
886	0	0
5194	0	0
24114	0	0

5.4. KNN(K-Nearest Neighbours)

In [52]:

```
from sklearn.neighbors import KNeighborsClassifier
model_knn=KNeighborsClassifier()
model_knn.fit(scaled_train_fs_data,y_train)
y_pred_KNN=model_knn.predict(scaled_test_fs_data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classificat
ion.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`,
`kurtosis`), the default behavior of `mode` typically preserves the axis
it acts along. In SciPy 1.11.0, this behavior will change: the default va
lue of `keepdims` will become False, the `axis` over which the statistic
is taken will be eliminated, and the value None will no longer be accepte
d. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [53]:

```
info={"Actual Value":y_test,"Predicted Value":y_pred_KNN}
KNN=pd.DataFrame(info)
KNN.sample(18)
```

Out[53]:

	Actual Value	Predicted Value
16749	0	0
16180	0	0
12512	0	0
12472	0	0
11442	1	1
28403	0	0
22035	0	0
14601	0	0
11123	0	0
22425	0	0
1751	1	1
21542	0	0
23994	0	0
20686	1	0
11818	0	0
14073	0	0
6718	0	0
14515	0	0

5.5. Decision Tree

In [54]:

```
from sklearn.tree import DecisionTreeClassifier
model_dt=DecisionTreeClassifier()
model_dt.fit(scaled_train_fs_data,y_train)
y_pred_DT=model_dt.predict(scaled_test_fs_data)
```

In [55]:

```
info={"Actual Value":y_test,"Predicted Value":y_pred_DT}
DT=pd.DataFrame(info)
DT.sample(18)
```

Out[55]:

	Actual Value	Predicted Value
22312	0	1
4401	0	0
9884	0	0
26683	0	0
4025	0	0
18067	0	1
880	0	0
7174	0	0
16092	1	1
29527	0	0

5.6. Random Forest

In [56]:

```
from sklearn.ensemble import RandomForestClassifier
model_rf=RandomForestClassifier(n_estimators=100)
model_rf.fit(scaled_train_fs_data,y_train)
y_pred_RF=model_rf.predict(scaled_test_fs_data)
```

In [57]:

```
# info={"Actual Value":y_test,"Predicted Value":y_pred_RF}
RF=pd.DataFrame(info)
RF.sample(18)
```

Out[57]:

	Actual Value	Predicted Value
1518	0	0
14615	0	0
5300	0	1
13679	1	0
5285	0	0
15877	1	1
8398	0	0
28648	0	0
3302	1	0
6964	0	0
16777	0	0
7568	1	0
7988	0	0
187	1	0
3002	1	1
13350	0	0
7307	1	0
1136	0	0

5.7. XG Boost(eXtreme Gradient Boosting)

In [58]:

```
from xgboost import XGBClassifier
model_xgb=XGBClassifier(n_estimators=500)
model_xgb.fit(scaled_train_fs_data,y_train)
y_pred_XGB=model_xgb.predict(scaled_test_fs_data)
```

In [59]:

```
info={"Actual Value":y_test,"Predicted Value":y_pred_XGB}
XGB=pd.DataFrame(info)
XGB.sample(18)
```

Out[59]:

	Actual Value	Predicted Value
28728	0	0
28725	0	0
15489	0	0
6658	1	0
13473	0	0
9368	1	0
6418	0	0
22988	0	0
15126	1	0
31	1	0
21441	0	0
21224	0	1
10650	0	0
28996	0	0
11650	1	0
6248	0	0
18346	0	1
20301	0	0

5.8.ANN(Artificial Neural Network) Using TensorFlow & Keras

In [60]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model_ann=Sequential()
model_ann.add(Dense(11,input_dim=11,activation="relu"))
model_ann.add(Dense(11,activation="relu"))
model_ann.add(Dense(11,activation="relu"))
model_ann.add(Dense(11,activation="relu"))
model_ann.add(Dense(1,activation="relu"))
model_ann.compile(optimizer="Adam",loss="binary_crossentropy",metrics=["accuracy"])
model_ann.fit(scaled_train_fs_data,y_train,epochs=100,verbose=1)
```

```
Epoch 1/100
657/657 [==============================] - 5s 3ms/step - loss: 0.6146
- accuracy: 0.7846
Epoch 2/100
657/657 [==============================] - 2s 3ms/step - loss: 0.4840
- accuracy: 0.8022
Epoch 3/100
657/657 [==============================] - 2s 3ms/step - loss: 0.4673
- accuracy: 0.8016
Epoch 4/100
657/657 [==============================] - 2s 3ms/step - loss: 0.4693
- accuracy: 0.8037
Epoch 5/100
657/657 [==============================] - 2s 3ms/step - loss: 0.4661
- accuracy: 0.8065
Epoch 6/100
657/657 [==============================] - 2s 3ms/step - loss: 0.4602
- accuracy: 0.8050
Epoch 7/100
```

In [61]:

```
acc=model_ann.evaluate(scaled_test_fs_data,y_test)
acc
```

```
282/282 [==============================] - 1s 2ms/step - loss: 0.4349 - a
ccuracy: 0.8198
```

Out[61]:

```
[0.4349076747894287, 0.81977778673172]
```

In [62]:

```
y_pred_ann=model_ann.predict(scaled_test_fs_data)
```

```
282/282 [==============================] - 1s 2ms/step
```

Part6-Evaluating The Performance of Algorithms

6.1. Accuracy Score

In [63]:

```
from sklearn.metrics import accuracy_score
lr_as=accuracy_score(y_test,y_pred_logreg)
svm_as=accuracy_score(y_test,y_pred_svm)
nb_as=accuracy_score(y_test,y_pred_navbay)
knn_as=accuracy_score(y_test,y_pred_KNN)
dt_as=accuracy_score(y_test,y_pred_DT)
rf_as=accuracy_score(y_test,y_pred_RF)
xgb_as=accuracy_score(y_test,y_pred_XGB)
print("Accuracy_score-Logistic Regression:",lr_as)
print("Accuracy_score-SVM:",svm_as)
print("Accuracy_score-Naive Bayes:",nb_as)
print("Accuracy_score-KNN:",knn_as)
print("Accuracy_score-Decision Tree:",dt_as)
print("Accuracy_score-Random Forest:",rf_as)
print("Accuracy_score-XGBoost:",xgb_as)
print("Accuracy_score-ANN:",acc[1])
```

```
Accuracy_score-Logistic Regression: 0.8164444444444444
Accuracy_score-SVM: 0.8246666666666667
Accuracy_score-Naive Bayes: 0.761
Accuracy_score-KNN: 0.8027777777777778
Accuracy_score-Decision Tree: 0.7184444444444444
Accuracy_score-Random Forest: 0.8191111111111111
Accuracy_score-XGBoost: 0.8036666666666666
Accuracy_score-ANN: 0.81977778673172
```

6.2. F1_Score

In [64]:

```
from sklearn.metrics import f1_score
lr_fs=f1_score(y_test,y_pred_logreg,average="weighted")
svm_fs=f1_score(y_test,y_pred_svm,average="weighted")
nb_fs=f1_score(y_test,y_pred_navbay,average="weighted")
knn_fs=f1_score(y_test,y_pred_KNN,average="weighted")
dt_fs=f1_score(y_test,y_pred_DT,average="weighted")
rf_fs=f1_score(y_test,y_pred_RF,average="weighted")
xgb_fs=f1_score(y_test,y_pred_XGB,average="weighted")
print("F1_Score-Logistic Regression:",lr_fs)
print("F1_Score-SVM:",svm_fs)
print("F1_Score-Naive Bayes:",nb_fs)
print("F1_Score-KNN:",knn_fs)
print("F1_Score-Decision Tree:",dt_fs)
print("F1_Score-Random Forest:",rf_fs)
print("F1_Score-XGBoost:",xgb_fs)
```

```
F1_Score-Logistic Regression: 0.7771630706983483
F1_Score-SVM: 0.803047447864442
F1_Score-Naive Bayes: 0.7712940224004341
F1_Score-KNN: 0.7866933208974003
F1_Score-Decision Tree: 0.7215090678216638
F1_Score-Random Forest: 0.796187042563708
F1_Score-XGBoost: 0.7785967009852762
```

6.3. Creating The Dataframe Of the Performance Metrics

In [65]:

```
data={"Accuracy Score":[lr_as,svm_as,nb_as,knn_as,dt_as,rf_as,xgb_as,acc[1]], "F1_S
table1=pd.DataFrame(data,index=["Logistic Regression","Support Vector Machine","Nai
table1
```

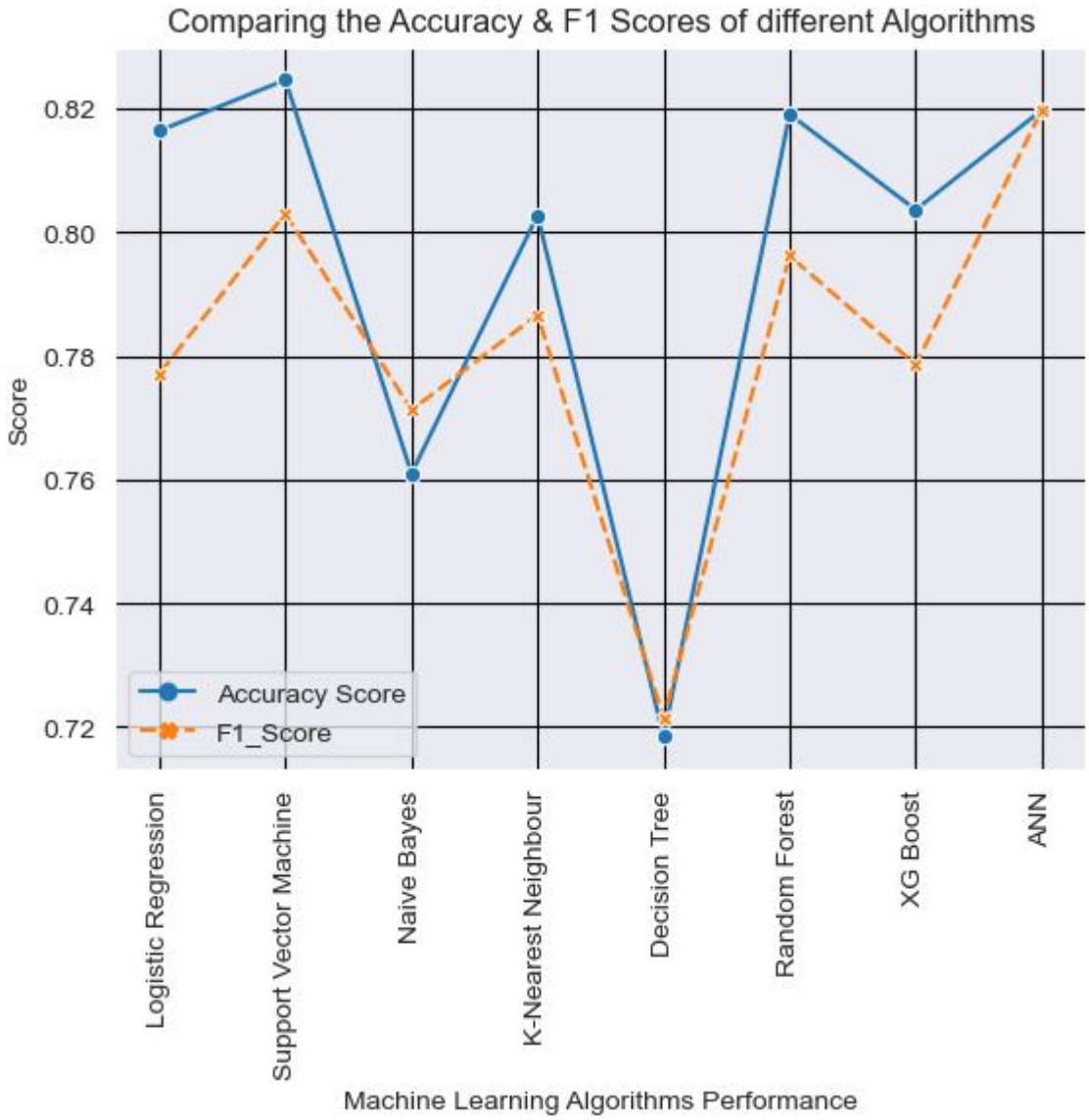
Out[65]:

	Accuracy Score	F1_Score
Logistic Regression	0.816444	0.777163
Support Vector Machine	0.824667	0.803047
Naive Bayes	0.761000	0.771294
K-Nearest Neighbour	0.802778	0.786693
Decision Tree	0.718444	0.721509
Random Forest	0.819111	0.796187
XG Boost	0.803667	0.778597
ANN	0.819778	0.819778

6.4. Plotting The Performance Of All The Algorithms Using Line Chart

In [66]:

```
sns.lineplot(data = table1, markers=True, dashes=True)
plt.xticks(rotation=90)
plt.grid(color="black")
plt.xlabel("Machine Learning Algorithms Performance")
plt.ylabel("Score")
plt.title("Comparing the Accuracy & F1 Scores of different Algorithms")
plt.show()
```



CONCLUSION

After performing 9 algorithms we can clearly conclude that out of nine algorithms following 2are the best performing algorithms

1. SVM(Support Vector Machine)
2. ANN(Artificial Neural Network)