ds-project

May 11, 2024

#

Bank Marketing

0.0.1 Team Members :

1. Ahmed Ashraf Ahmed Alsayed (Leader)
2. Ahmed Elsayed Ahmed Hassan ( )
.3 Ahmed Elsayed Mahmoud Arafa
4. Radwa Mohamed Said Abd El- Shafi (Meme Lord)
5. Abdelrahman Abdo Hassan Ahmed
6. Nada Adel Ismael Hussainen Shahin
7. Hani Yasser ElMetwaly Sorour Ahmed

0.0.2 Table of contents :

** #

Introduction

Tabel of Contents

`Dataset`

1. Title: Bank Marketing.
2. Sources Created by: Sérgio Moro (ISCTE-IUL), Paulo Cortez (Univ. Minho) and Paulo Rita (ISCTE-IUL).
3. Past Usage: The full dataset (bank-additional-full.csv) was described and analyzed in: S. Moro, P. Cortez, and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing.
4. Relevant Information: This dataset is based on “Bank Marketing” UCI dataset (check the description at: http://archive.ics.uci.edu/ml/datasets/Bank+Marketing.
5. Number of Instances: 41188 for bank-additional-full.csv
6. Number of Attributes: 20 + output attribute.

7. Attribute information:

Attribute	Information
age	numeric
job	categorical: "admin","blue-collar" ,"entrepreneur" , "housemaid","management","retired","selfemployed", "services","student","technician","unemployed", "unknown")
marital	categorical: "divorced","married", "single","unknown")
education	categorical:"basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unknown")
default	has credit in default? (categorical: "no","yes","unknown")
housing	has housing loan? (categorical: "no","yes","unknown")
loan	has personal loan? (categorical: "no","yes" ,"unknown")
contact	categorical: "cellular","telephone")
month	last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
day_of_week	last contact day of the week (categorical: "mon" ,"tue" ,"wed","thu","fri")
duration	last contact duration, in seconds (numeric).
campaign	number of contacts performed during this campaign and for this client (numeric, includes last contact)
pdays	number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
previous	number of contacts performed before this campaign and for this client (numeric)
poutcome	outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")
emp.var.rate	employment variation rate - quarterly indicator (numeric)
cons.price.idx	consumer price index - monthly indicator (numeric)
cons.conf.idx	consumer confidence index - monthly indicator (numeric)
euribor3m	euribor 3 month rate - daily indicator (numeric)
nr.employed	number of employees - quarterly indicator (numeric)
y	has the client subscribed a term deposit? (binary: "yes","no")

8. Missing Attribute Values: There are several missing values in some categorical attributes, all coded with the "unknown" label. These missing values can be treated as a possible class label or using deletion or imputation techniques.

** #

Import Libraries

Tabel of Contents

[1]:
```
pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in /opt/conda/lib/python3.10/site-
packages (1.5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in
/opt/conda/lib/python3.10/site-packages (from yellowbrick) (3.7.5)
Requirement already satisfied: scipy>=1.0.0 in /opt/conda/lib/python3.10/site-
packages (from yellowbrick) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.0 in
/opt/conda/lib/python3.10/site-packages (from yellowbrick) (1.2.2)
Requirement already satisfied: numpy>=1.16.0 in /opt/conda/lib/python3.10/site-
packages (from yellowbrick) (1.26.4)
Requirement already satisfied: cycler>=0.10.0 in /opt/conda/lib/python3.10/site-
packages (from yellowbrick) (0.12.1)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.2.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.10/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.9.0.post0)
Requirement already satisfied: joblib>=1.1.1 in /opt/conda/lib/python3.10/site-
packages (from scikit-learn>=1.0.0->yellowbrick) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn>=1.0.0->yellowbrick)
(3.2.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick)
(1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import pandas as pd
import numpy as np
import plotly.express as px
from plotly.offline import init_notebook_mode
import plotly.graph_objs as go
```

```python
import cufflinks as cf
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
init_notebook_mode(connected=True)
cf.go_offline()
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
import warnings
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.preprocessing import MinMaxScaler,Normalizer
from sklearn.feature_selection import SelectFromModel
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import f1_score,accuracy_score,r2_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_curve,RocCurveDisplay
from sklearn.metrics import auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from sklearn.decomposition import PCA
from yellowbrick.cluster import KElbowVisualizer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
#from sklearn.preprocessing import PolynomialFeature
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
import keras
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
px.defaults.template = 'plotly_dark'
```

```
2024-05-10 13:57:08.249379: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-05-10 13:57:08.249519: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-05-10 13:57:08.398094: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

** #

Read Data

Tabel of Contents

Read in the csv file as a dataframe called data

```
The tabulated data is meticulously arranged, featuring distinct columns housing a range of dive
```

[3]:
```
data=pd.read_csv('/kaggle/input/bank-marketing/bank-additional-full.csv',sep=';
↪')
```

Check the head of data

[4]:
```
data.head()
```

[4]:
```
   age        job  marital    education  default housing loan    contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  month day_of_week  …  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  …         1    999         0  nonexistent          1.1
1   may         mon  …         1    999         0  nonexistent          1.1
2   may         mon  …         1    999         0  nonexistent          1.1
3   may         mon  …         1    999         0  nonexistent          1.1
4   may         mon  …         1    999         0  nonexistent          1.1
```

```
   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
```

Check the shape of data

[5]:
```
data.shape
```

[5]:
```
(41188, 21)
```

The dataset comprises 41,188 rows and 21 columns

Check the info of data

[6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
```

```
memory usage: 6.6+ MB
```

```
The dataset includes 5 columns of floating-point values, 5 columns of integers, and 11 columns
```

Description of data

```
If the DataFrame contains numerical data, the description contains these information for each c

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.
75% - The 75% percentile*.
max - the maximum value.
```

[7]:
```
data.describe().transpose()
```

[7]:
```
                  count         mean         std       min       25%  \
age             41188.0    40.024060   10.421250    17.000    32.000
duration        41188.0   258.285010  259.279249     0.000   102.000
campaign        41188.0     2.567593    2.770014     1.000     1.000
pdays           41188.0   962.475454  186.910907     0.000   999.000
previous        41188.0     0.172963    0.494901     0.000     0.000
emp.var.rate    41188.0     0.081886    1.570960    -3.400    -1.800
cons.price.idx  41188.0    93.575664    0.578840    92.201    93.075
cons.conf.idx   41188.0   -40.502600    4.628198   -50.800   -42.700
euribor3m       41188.0     3.621291    1.734447     0.634     1.344
nr.employed     41188.0  5167.035911   72.251528  4963.600  5099.100

                     50%       75%       max
age               38.000    47.000    98.000
duration         180.000   319.000  4918.000
campaign           2.000     3.000    56.000
pdays            999.000   999.000   999.000
previous           0.000     0.000     7.000
emp.var.rate       1.100     1.400     1.400
cons.price.idx    93.749    93.994    94.767
cons.conf.idx    -41.800   -36.400   -26.900
euribor3m          4.857     4.961     5.045
nr.employed     5191.000  5228.100  5228.100
```

```
For object data types, the describe method typically includes:

Count: The number of non-empty values.
Unique: The number of unique values.
Top: The most frequently occurring value.
Freq: The frequency of the top value.
```

[8]:
```
data.describe(include='O').transpose()
```

[8]:

	count	unique	top	freq
job	41188	12	admin.	10422
marital	41188	4	married	24928
education	41188	8	university.degree	12168
default	41188	3	no	32588
housing	41188	3	yes	21576
loan	41188	3	no	33950
contact	41188	2	cellular	26144
month	41188	10	may	13769
day_of_week	41188	5	thu	8623
poutcome	41188	3	nonexistent	35563
y	41188	2	no	36548

check for null values in the data

`Missing Attribute Values: There are several missing values in some categorical attributes, all`

[9]:
```
## Since there are no missing values, this step is not applicable in this case
data.replace("unknown",np.nan,inplace=True)
```

[10]:
```
is_null,precentage = data.isnull().sum(),(data.isnull().sum()/data.shape[0])*100
df = pd.DataFrame()
df['Count'],df['Precentage%']=is_null,precentage
df
```

[10]:

	Count	Precentage%
age	0	0.000000
job	330	0.801204
marital	80	0.194231
education	1731	4.202680
default	8597	20.872584
housing	990	2.403613
loan	990	2.403613
contact	0	0.000000
month	0	0.000000
day_of_week	0	0.000000
duration	0	0.000000
campaign	0	0.000000
pdays	0	0.000000
previous	0	0.000000
poutcome	0	0.000000
emp.var.rate	0	0.000000
cons.price.idx	0	0.000000
cons.conf.idx	0	0.000000
euribor3m	0	0.000000
nr.employed	0	0.000000

```
y                    0     0.000000
```

[11]:

```
fig = go.Figure()
distribution = df['Count']
bar_trace = go.Bar(x=distribution.index, y=distribution.values, name="Missing␣
 ↪Values",text=distribution.values, textposition='inside')
fig.add_trace(bar_trace)
fig.update_layout(
    title_text='Missing Values',
    title_x=0.5,
    title_font=dict(size=20),
    xaxis_title="Columns",
    yaxis_title='Count',
    font=dict(size=15),
    width=1000,
    height=700,
    xaxis=dict(tickangle=-90),
    template='plotly_dark'
)
fig.update_annotations(font=dict(size=20))
fig.show()
```

```
Based on the provided data frame:

 The "age" column has 0 missing values, accounting for 0% of the total.
 The "job" column has 330 missing values, making up about 0.80% of the total.
 The "marital" column has 80 missing values, representing approximately 0.19% of the total.
 The "education" column has 1731 missing values, comprising around 4.20% of the total.
 The "default" column has 8597 missing values, accounting for about 20.87% of the total.
 The "housing" and "loan" columns each have 990 missing values, accounting for about 2.40% of t
 The columns "contact", "month", "day_of_week", "duration", "campaign", "pdays", "previous", "p
```

Handle null values

```
To handle null values in your dataset, you can use various methods depending on the type of da

For Numerical Data:
1. Mean/Median Imputation: Replace missing values with the mean or median of the column.
2. Random Imputation: Replace missing values with randomly sampled values from the distributio
3. Predictive Imputation: Use a predictive model to predict missing values based on other vari

For Categorical Data:
1. Most Frequent Imputation: Replace missing values with the most frequent value in the column
2. Constant Imputation: Replace missing values with a specific constant value.
3. Predictive Imputation: You can also use a predictive model tailored for categorical data to

Best Practices:
all null columns object In this case, using Most Frequent Imputation for handling missing value
```

[12]:
```
data.mode().iloc[0]
```

[12]:
```
age                              31.0
job                            admin.
marital                       married
education           university.degree
default                            no
housing                           yes
loan                               no
contact                      cellular
month                             may
day_of_week                       thu
duration                           85
campaign                          1.0
pdays                           999.0
previous                          0.0
poutcome                  nonexistent
emp.var.rate                      1.4
cons.price.idx                 93.994
cons.conf.idx                   -36.4
euribor3m                       4.857
nr.employed                    5228.1
y                                  no
Name: 0, dtype: object
```

[13]:
```
#data.fillna(data.mode().iloc[0],inplace=True)
key = data.keys()
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data = imp.fit_transform(data)
data = pd.DataFrame(data,columns=key)
```

[14]:
```
is_null,precentage = data.isnull().sum(),(data.isnull().sum()/data.shape[0])*100
df = pd.DataFrame()
df['Count'],df['Precentage%']=is_null,precentage
df
```

[14]:

	Count	Precentage%
age	0	0.0
job	0	0.0
marital	0	0.0
education	0	0.0
default	0	0.0
housing	0	0.0
loan	0	0.0
contact	0	0.0
month	0	0.0
day_of_week	0	0.0

```
duration            0          0.0
campaign            0          0.0
pdays               0          0.0
previous            0          0.0
poutcome            0          0.0
emp.var.rate        0          0.0
cons.price.idx      0          0.0
cons.conf.idx       0          0.0
euribor3m           0          0.0
nr.employed         0          0.0
y                   0          0.0
```

check duplicate data

```
data[data.duplicated(keep=False)]
returns all rows in the DataFrame that are duplicates, including both the original rows and the
```

[15]:
```python
data[data.duplicated(keep=False)]
```

[15]:
```
       age          job   marital            education default housing loan  \
236     56  blue-collar   married             basic.4y      no      no   no
1265    39  blue-collar   married             basic.6y      no      no   no
1266    39  blue-collar   married             basic.6y      no      no   no
5664    56  blue-collar   married             basic.4y      no      no   no
12260   36      retired   married    university.degree      no      no   no
12261   36      retired   married    university.degree      no      no   no
14155   27   technician    single  professional.course      no      no   no
14234   27   technician    single  professional.course      no      no   no
16819   47   technician  divorced          high.school      no     yes   no
16956   47   technician  divorced          high.school      no     yes   no
18464   32   technician    single  professional.course      no     yes   no
18465   32   technician    single  professional.course      no     yes   no
19451   33       admin.   married    university.degree      no     yes   no
19608   33       admin.   married    university.degree      no     yes   no
20072   55     services   married          high.school      no      no   no
20216   55     services   married          high.school      no      no   no
20531   41   technician   married  professional.course      no     yes   no
20534   41   technician   married  professional.course      no     yes   no
25183   39       admin.   married    university.degree      no      no   no
25217   39       admin.   married    university.degree      no      no   no
28476   24     services    single          high.school      no     yes   no
28477   24     services    single          high.school      no     yes   no
32505   35       admin.   married    university.degree      no     yes   no
32516   35       admin.   married    university.degree      no     yes   no
36950   45       admin.   married    university.degree      no      no   no
36951   45       admin.   married    university.degree      no      no   no
38255   71      retired    single    university.degree      no      no   no
38281   71      retired    single    university.degree      no      no   no
```

	contact	month	day_of_week	…	campaign	pdays	previous	poutcome	\
236	telephone	may	mon	…	1	999	0	nonexistent	
1265	telephone	may	thu	…	1	999	0	nonexistent	
1266	telephone	may	thu	…	1	999	0	nonexistent	
5664	telephone	may	mon	…	1	999	0	nonexistent	
12260	telephone	jul	thu	…	1	999	0	nonexistent	
12261	telephone	jul	thu	…	1	999	0	nonexistent	
14155	cellular	jul	mon	…	2	999	0	nonexistent	
14234	cellular	jul	mon	…	2	999	0	nonexistent	
16819	cellular	jul	thu	…	3	999	0	nonexistent	
16956	cellular	jul	thu	…	3	999	0	nonexistent	
18464	cellular	jul	thu	…	1	999	0	nonexistent	
18465	cellular	jul	thu	…	1	999	0	nonexistent	
19451	cellular	aug	thu	…	1	999	0	nonexistent	
19608	cellular	aug	thu	…	1	999	0	nonexistent	
20072	cellular	aug	mon	…	1	999	0	nonexistent	
20216	cellular	aug	mon	…	1	999	0	nonexistent	
20531	cellular	aug	tue	…	1	999	0	nonexistent	
20534	cellular	aug	tue	…	1	999	0	nonexistent	
25183	cellular	nov	tue	…	2	999	0	nonexistent	
25217	cellular	nov	tue	…	2	999	0	nonexistent	
28476	cellular	apr	tue	…	1	999	0	nonexistent	
28477	cellular	apr	tue	…	1	999	0	nonexistent	
32505	cellular	may	fri	…	4	999	0	nonexistent	
32516	cellular	may	fri	…	4	999	0	nonexistent	
36950	cellular	jul	thu	…	1	999	0	nonexistent	
36951	cellular	jul	thu	…	1	999	0	nonexistent	
38255	telephone	oct	tue	…	1	999	0	nonexistent	
38281	telephone	oct	tue	…	1	999	0	nonexistent	

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
236	1.1	93.994	-36.4	4.857	5191.0	no
1265	1.1	93.994	-36.4	4.855	5191.0	no
1266	1.1	93.994	-36.4	4.855	5191.0	no
5664	1.1	93.994	-36.4	4.857	5191.0	no
12260	1.4	93.918	-42.7	4.966	5228.1	no
12261	1.4	93.918	-42.7	4.966	5228.1	no
14155	1.4	93.918	-42.7	4.962	5228.1	no
14234	1.4	93.918	-42.7	4.962	5228.1	no
16819	1.4	93.918	-42.7	4.962	5228.1	no
16956	1.4	93.918	-42.7	4.962	5228.1	no
18464	1.4	93.918	-42.7	4.968	5228.1	no
18465	1.4	93.918	-42.7	4.968	5228.1	no
19451	1.4	93.444	-36.1	4.968	5228.1	no
19608	1.4	93.444	-36.1	4.968	5228.1	no
20072	1.4	93.444	-36.1	4.965	5228.1	no

```
20216          1.4          93.444         -36.1      4.965       5228.1    no
20531          1.4          93.444         -36.1      4.966       5228.1    no
20534          1.4          93.444         -36.1      4.966       5228.1    no
25183         -0.1            93.2         -42.0      4.153       5195.8    no
25217         -0.1            93.2         -42.0      4.153       5195.8    no
28476         -1.8          93.075         -47.1      1.423       5099.1    no
28477         -1.8          93.075         -47.1      1.423       5099.1    no
32505         -1.8          92.893         -46.2      1.313       5099.1    no
32516         -1.8          92.893         -46.2      1.313       5099.1    no
36950         -2.9          92.469         -33.6      1.072       5076.2   yes
36951         -2.9          92.469         -33.6      1.072       5076.2   yes
38255         -3.4          92.431         -26.9      0.742       5017.5    no
38281         -3.4          92.431         -26.9      0.742       5017.5    no

[28 rows x 21 columns]
```

keep='first': When you use data.duplicated(keep='first')
it identifies and marks duplicates in the DataFrame, keeping only the first occurrence of each

[16]:
```python
data[data.duplicated(keep='first')]
```

[16]:
```
      age          job   marital            education default housing loan  \
1266   39  blue-collar   married             basic.6y      no      no   no
5664   56  blue-collar   married             basic.4y      no      no   no
12261  36      retired   married    university.degree      no      no   no
14234  27   technician    single  professional.course      no      no   no
16956  47   technician  divorced          high.school      no     yes   no
18465  32   technician    single  professional.course      no     yes   no
19608  33       admin.   married    university.degree      no     yes   no
20216  55     services   married          high.school      no      no   no
20534  41   technician   married  professional.course      no     yes   no
25217  39       admin.   married    university.degree      no      no   no
28477  24     services    single          high.school      no     yes   no
32516  35       admin.   married    university.degree      no     yes   no
36951  45       admin.   married    university.degree      no      no   no
38281  71      retired    single    university.degree      no      no   no

         contact month day_of_week  … campaign pdays previous     poutcome  \
1266   telephone   may         thu  …        1   999        0  nonexistent
5664   telephone   may         mon  …        1   999        0  nonexistent
12261  telephone   jul         thu  …        1   999        0  nonexistent
14234   cellular   jul         mon  …        2   999        0  nonexistent
16956   cellular   jul         thu  …        3   999        0  nonexistent
18465   cellular   jul         thu  …        1   999        0  nonexistent
19608   cellular   aug         thu  …        1   999        0  nonexistent
20216   cellular   aug         mon  …        1   999        0  nonexistent
20534   cellular   aug         tue  …        1   999        0  nonexistent
25217   cellular   nov         tue  …        2   999        0  nonexistent
```

```
28477   cellular   apr         tue  ...        1   999         0  nonexistent
32516   cellular   may         fri  ...        4   999         0  nonexistent
36951   cellular   jul         thu  ...        1   999         0  nonexistent
38281  telephone   oct         tue  ...        1   999         0  nonexistent

      emp.var.rate cons.price.idx cons.conf.idx euribor3m nr.employed    y
1266           1.1         93.994         -36.4     4.855      5191.0   no
5664           1.1         93.994         -36.4     4.857      5191.0   no
12261          1.4         93.918         -42.7     4.966      5228.1   no
14234          1.4         93.918         -42.7     4.962      5228.1   no
16956          1.4         93.918         -42.7     4.962      5228.1   no
18465          1.4         93.918         -42.7     4.968      5228.1   no
19608          1.4         93.444         -36.1     4.968      5228.1   no
20216          1.4         93.444         -36.1     4.965      5228.1   no
20534          1.4         93.444         -36.1     4.966      5228.1   no
25217         -0.1           93.2         -42.0     4.153      5195.8   no
28477         -1.8         93.075         -47.1     1.423      5099.1   no
32516         -1.8         92.893         -46.2     1.313      5099.1   no
36951         -2.9         92.469         -33.6     1.072      5076.2  yes
38281         -3.4         92.431         -26.9     0.742      5017.5   no

[14 rows x 21 columns]
```

keep='last': Conversely, when you use data.duplicated(keep='last')
it also identifies and marks duplicates in the DataFrame. However, it keeps only the last occu

[17]:
```python
data[data.duplicated(keep='last')]
```

[17]:
```
      age          job   marital            education default housing loan  \
236    56  blue-collar   married             basic.4y      no      no   no
1265   39  blue-collar   married             basic.6y      no      no   no
12260  36      retired   married    university.degree      no      no   no
14155  27   technician    single  professional.course      no      no   no
16819  47   technician  divorced          high.school      no     yes   no
18464  32   technician    single  professional.course      no     yes   no
19451  33       admin.   married    university.degree      no     yes   no
20072  55     services   married          high.school      no      no   no
20531  41   technician   married  professional.course      no     yes   no
25183  39       admin.   married    university.degree      no      no   no
28476  24     services    single          high.school      no     yes   no
32505  35       admin.   married    university.degree      no     yes   no
36950  45       admin.   married    university.degree      no      no   no
38255  71      retired    single    university.degree      no      no   no

         contact month day_of_week  ... campaign pdays previous     poutcome  \
236    telephone   may         mon  ...        1   999         0  nonexistent
1265   telephone   may         thu  ...        1   999         0  nonexistent
12260  telephone   jul         thu  ...        1   999         0  nonexistent
```

```
14155   cellular   jul       mon  ...     2   999        0  nonexistent
16819   cellular   jul       thu  ...     3   999        0  nonexistent
18464   cellular   jul       thu  ...     1   999        0  nonexistent
19451   cellular   aug       thu  ...     1   999        0  nonexistent
20072   cellular   aug       mon  ...     1   999        0  nonexistent
20531   cellular   aug       tue  ...     1   999        0  nonexistent
25183   cellular   nov       tue  ...     2   999        0  nonexistent
28476   cellular   apr       tue  ...     1   999        0  nonexistent
32505   cellular   may       fri  ...     4   999        0  nonexistent
36950   cellular   jul       thu  ...     1   999        0  nonexistent
38255  telephone   oct       tue  ...     1   999        0  nonexistent

      emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.employed    y
236            1.1          93.994          -36.4      4.857       5191.0   no
1265           1.1          93.994          -36.4      4.855       5191.0   no
12260          1.4          93.918          -42.7      4.966       5228.1   no
14155          1.4          93.918          -42.7      4.962       5228.1   no
16819          1.4          93.918          -42.7      4.962       5228.1   no
18464          1.4          93.918          -42.7      4.968       5228.1   no
19451          1.4          93.444          -36.1      4.968       5228.1   no
20072          1.4          93.444          -36.1      4.965       5228.1   no
20531          1.4          93.444          -36.1      4.966       5228.1   no
25183         -0.1            93.2          -42.0      4.153       5195.8   no
28476         -1.8          93.075          -47.1      1.423       5099.1   no
32505         -1.8          92.893          -46.2      1.313       5099.1   no
36950         -2.9          92.469          -33.6      1.072       5076.2  yes
38255         -3.4          92.431          -26.9      0.742       5017.5   no

[14 rows x 21 columns]
```

Remove duplicates

Remove duplicates keeping the first occurrence

[18]:
```
data.drop_duplicates(keep='first', inplace=True)
```

[19]:
```
data[data.duplicated()]
```

[19]:
```
Empty DataFrame
Columns: [age, job, marital, education, default, housing, loan, contact, month,
day_of_week, duration, campaign, pdays, previous, poutcome, emp.var.rate,
cons.price.idx, cons.conf.idx, euribor3m, nr.employed, y]
Index: []

[0 rows x 21 columns]
```

** #

EDA

Tabel of Contents

Exploratory Data Analysis (EDA) is a crucial step in data analysis where you explore and summa

Helper Functions

[20]:
```
def hist_hue(feature,hue,title_f,title_h,title):
    num_bins=20
    total_hist, _ = np.histogram(data[feature], bins=num_bins)
    fig = make_subplots(rows=1, cols=2, subplot_titles=(title_f, f"{title_f} VS␣
↪{title_h}"))
    histogram_trace_total = go.Bar(x=np.arange(num_bins), y=total_hist,␣
↪name=title_f, text=total_hist, textposition='inside')
    fig.add_trace(histogram_trace_total, row=1, col=1)
    for category in data[hue].unique():
        category_data = data[data[hue] == category][feature]
        category_hist, _ = np.histogram(category_data, bins=num_bins)
        histogram_trace_by_hue = go.Bar(x=np.arange(num_bins), y=category_hist,␣
↪name=f'{title_f} VS {title_h} ({category})', text=category_hist,␣
↪textposition='inside')
        fig.add_trace(histogram_trace_by_hue, row=1, col=2)
    fig.update_layout(
       title_text=title,
       title_x=0.5,
       title_font=dict(size=20),
       font=dict(size=15),
       width=1000,
       height=700,
       barmode='stack',
       template='plotly_dark',
        xaxis_title=title_f,
        yaxis_title='Count',
        xaxis2_title=title_f,
    )
    fig.update_annotations(font=dict(size=20))
    fig.show()
```

[21]:
```
def Bar_hue(feature,hue,title_f,title_h,title):
    fig = make_subplots(rows=1, cols=2, subplot_titles=(title_f, f"{title_f} VS␣
↪{title_h}"))
    distribution = data[feature].value_counts()
    bar_trace = go.Bar(x=distribution.index, y=distribution.values,␣
↪name=title_f,text=distribution.values, textposition='inside')
    fig.add_trace(bar_trace, row=1, col=1)
    for category in data[hue].unique():
        category_data = data[data[hue] == category][feature].value_counts()
```

```
        bar_trace_by_hue = go.Bar(x=category_data.index, y=category_data.
↪values, name=f'{title_f} VS {title_h} ({category})', text=category_data.
↪values, textposition='inside')
        fig.add_trace(bar_trace_by_hue, row=1, col=2)
    fig.update_layout(
        title_text=title,
        title_x=0.5,
        title_font=dict(size=20),
        xaxis_title=title_f,
        yaxis_title='Count',
        xaxis2_title=title_f,
        font=dict(size=15),
        barmode='stack',
        width=1000,
        height=700,
        xaxis=dict(tickangle=-90),
        xaxis2=dict(tickangle=-90),
        template='plotly_dark'
    )
    fig.update_annotations(font=dict(size=20))
    fig.show()
```

[22]:

```
def␣
↪Bar_2hue(feature,hue1,title_f,title_h1,title='',make_subplot=True,hue2='',title_h2=''):
↪
    if make_subplot:
        fig = make_subplots(rows=1, cols=2, subplot_titles=(f"{title_f} VS␣
↪{title_h1}",f"{title_f} VS {title_h2}"))
        for category in data[hue1].unique():
            category_data = data[data[hue1] == category][feature].value_counts()
            bar_trace_by_hue = go.Bar(x=category_data.index, y=category_data.
↪values, name=f'{title_f} VS {title_h1} ({category})',text=category_data.
↪values, textposition='inside')
            fig.add_trace(bar_trace_by_hue, row=1, col=1)
        for category in data[hue2].unique():
            category_data = data[data[hue2] == category][feature].value_counts()
            bar_trace_by_hue = go.Bar(x=category_data.index, y=category_data.
↪values, name=f'{title_f} VS {title_h2} ({category})',text=category_data.
↪values, textposition='inside')
            fig.add_trace(bar_trace_by_hue, row=1, col=2)
        fig.update_layout(
            title_text=title,
            title_x=0.5,
            title_font=dict(size=20),
            xaxis_title=title_f,
            yaxis_title='Count',
            xaxis2_title=title_f,
```

```
            font=dict(
                size=15,
            ),
            barmode='stack',
            width=1000,
            height=700,
            xaxis=dict(tickangle=-90),
            xaxis1=dict(tickangle=-90),
            xaxis2=dict(tickangle=-90),
            template='plotly_dark'
        )
        fig.update_annotations(font=dict(size=20))
        fig.show()
    else:
        fig = go.Figure()
        for category in data[hue1].unique():
            category_data = data[data[hue1] == category][feature].value_counts()
            bar_trace_by_hue = go.Bar(x=category_data.index, y=category_data.
↪values, name=f'{title_f} VS {title_h1} ({category})',text=category_data.
↪values, textposition='inside')
            fig.add_trace(bar_trace_by_hue)
        fig.update_layout(
            title_text=f'{title_f} VS {title_h1}',
            title_x=0.5,
            title_font=dict(size=20),
            xaxis_title=title_f,
            yaxis_title='Count',
            font=dict(
                size=15,
            ),
            barmode="stack",
            width=800,
            height=700,
            xaxis=dict(tickangle=-90),
            template='plotly_dark'
        )
        fig.update_annotations(font=dict(size=20))
        fig.show()
```

[23]:
```
def Boxplot_outlier(feature,title):
    fig = make_subplots(rows=1, cols=2, subplot_titles=("Box Plot", "Violin␣
↪Plot"))
    fig.add_trace(
        go.Box(y=data[feature], name='BoxPlot'),
        row=1, col=1
    )
    fig.add_trace(
```

```
        go.Violin(y=data[feature], name='ViolinPlot'),
        row=1, col=2
    )
    fig.update_layout(
        title_text=title,
        title_x=0.5,
        title_font=dict(size=20),
        font=dict(size=15),
        width=1000,
        height=500,
        template='plotly_dark'
    )
    fig.update_annotations(font=dict(size=20))
    fig.show()
```

[24]:
```
def Pie(feature,title_f,title):
    distribution = data[feature].value_counts()
    pie_trace = go.Pie(labels=distribution.index, values=distribution.values,␣
 ↪name=title_f)
    pie_layout = go.Layout(
        title=title,
        title_font=dict(size=20),
        width=600,
        height=500,
        title_x=0.5,
        template='plotly_dark'
    )
    fig_pie = go.Figure(data=[pie_trace], layout=pie_layout)
    fig_pie.show()
```

[25]:
```
def Heatmap(pivot1, title, feature, feature_h1, make_subplot=True,␣
 ↪feature_h2='', pivot2='', color='inferno'):
    fig_heatmap = None
    if make_subplot:
        fig = make_subplots(rows=1, cols=2, subplot_titles=(f"{feature} VS␣
 ↪{feature_h1}", f"{feature} VS {feature_h2}"))
        heat1 = go.Heatmap(
            z=pivot1.values,
            x=pivot1.columns,
            y=pivot1.index,
            colorscale=color,
            colorbar=dict(title='Count'),
            colorbar_x=0.45,
            colorbar_len=0.8
        )
        fig.add_trace(heat1, row=1, col=1)
        heat2 = go.Heatmap(
```

```
            z=pivot2.values,
            x=pivot2.columns,
            y=pivot2.index,
            colorscale=color,
            colorbar=dict(title='Count'),
            colorbar_x=1,
            colorbar_len=0.8
        )
        fig.add_trace(heat2, row=1, col=2)
        fig.update_layout(
            title=title,
            title_x=0.5,
            title_font=dict(size=20),
            width=1100,
            height=500,
            xaxis=dict(title=feature_h1, tickangle=-90),
            xaxis2=dict(title=feature_h2, tickangle=-90),
            yaxis=dict(title=feature, tickangle=-90),
            yaxis2=dict(tickangle=-90),
            font=dict(size=15),
            template='plotly_dark'
        )
        fig_heatmap = fig
    else:
        fig_heatmap = go.Figure(data=go.Heatmap(
            z=pivot1.values,
            x=pivot1.columns,
            y=pivot1.index,
            colorscale=color,
            colorbar=dict(title='Count')
        ))
        fig_heatmap.update_layout(
            title=title,
            title_x=0.5,
            title_font=dict(size=20),
            xaxis=dict(title=feature_h1),
            yaxis=dict(title=feature),
            font=dict(size=15),
            width=800,
            height=500,
            template='plotly_dark'
        )
    fig_heatmap.update_annotations(font=dict(size=20))
    fig_heatmap.show()
```

[26]:
```
def mean_plot(pivot_table,feature,hue,feature_t,hue_t):
    fig = go.Figure()
```

```
    for i in data[hue].unique():
        cate = pivot_table[pivot_table.index==i]
        bar_trace = go.Bar(x=cate.index, y=cate[feature],␣
 ↪text=round(cate[feature],2), textposition='inside', name=i)
        fig.add_trace(bar_trace)
    fig.update_layout(
        title_text=f'Average {feature_t}',
        title_x=0.5,
        title_font=dict(size=20),
        xaxis_title=hue_t,
        yaxis_title='Average',
        font=dict(size=15),
        barmode='stack',
        width=800,
        height=700,
        xaxis=dict(tickangle=-90),
        template='plotly_dark'
    )

    fig.update_annotations(font=dict(size=20))
    fig.show()
```

[27]:
```
def pivot(values_f,index_f,mean=True):
    if mean:
        return pd.pivot_table(data, values=values_f, index=index_f,␣
 ↪aggfunc='mean')
    else:
        return pd.pivot_table(data, index=values_f, columns=index_f,␣
 ↪aggfunc='size', fill_value=0)
def cross_t(index,columns):
    return pd.crosstab(index=data[index], columns=data[columns])
```

What is age distribution?

Find the minimum age

[28]:
```
data.age.min()
```

[28]: 17

Find the maximum age

[29]:
```
data.age.max()
```

[29]: 98

Find the top 5 most frequent ages

[30]:
```
data['age'].value_counts().head(5)
```

[30]:
```
age
31    1947
32    1845
33    1832
36    1779
35    1758
Name: count, dtype: int64
```

Based on the output, it seems that the age group 31 to 36 has the highest counts of observatio

Age 31: 1947 observations
Age 32: 1845 observations
Age 33: 1832 observations
Age 35: 1758 observations
Age 36: 1779 observations

calculate the mean age for each category in the y column

[31]:
```python
pivot_table = pivot('age','y')
pivot_table
```

[31]:
```
           age
y
no   39.910743
yes  40.912266
```

Observation: The pivot table reveals that the mean age of individuals who subscribed to the se

Visualization

[32]:
```python
hist_hue('age','y','Age','Y','Age Distribution')
```

[33]:
```python
Boxplot_outlier('age','Age Distribution')
```

Observation: Based on the figure, it appears that the age column contains some outliers.

What is Job distribution?

calculate the value counts for the job column

[34]:
```python
data.job.value_counts().to_frame()
```

[34]:
```
               count
job
admin.         10748
blue-collar     9252
technician      6739
services        3967
management      2924
retired         1718
entrepreneur    1456
```

```
self-employed   1421
housemaid       1060
unemployed      1014
student          875
```

Observation: The value counts for the "job" column indicate the frequency of each job category

The most common job category is "admin." with 10,748 occurrences.
Following "admin.", the next most frequent categories are "blue-collar" (9,252 occurrences) and
Some job categories have relatively fewer occurrences, such as "student" (875 occurrences) and

count the occurrences of each combination of job and y

[35]:
```python
pivot_table = pivot('job','y',False)
pivot_table
```

[35]:
```
y                no   yes
job
admin.         9360  1388
blue-collar    8614   638
entrepreneur   1332   124
housemaid       954   106
management     2596   328
retired        1284   434
self-employed  1272   149
services       3644   323
student         600   275
technician     6009   730
unemployed      870   144
```

Visualization

[36]:
```python
Pie('job','Job','Job Distribution')
```

[37]:
```python
Bar_hue('job','y','Job','Y','Job Distribution')
```

[38]:
```python
Heatmap(pivot_table,'Job Vs Y Categories','Job','Y',make_subplot=False)
```

observation based on figure: show frequency between Job and Y

What is marital distribution?

calculate the value counts for the "marital" column

[39]:
```python
data.marital.value_counts().to_frame()
```

[39]:
```
          count
marital
married   24999
single    11564
```

```
divorced   4611
```

shows the count of each category in the marital status data. For example, there are 24999 marri

count the occurrences of each combination of marital and y

[40]:
```python
pivot_table =pivot('marital','y',False)
pivot_table
```

[40]:
```
y             no   yes
marital
divorced    4135   476
married    22456  2543
single      9944  1620
```

count the occurrences of each combination of marital and job

[41]:
```python
pivot_table1 = pivot('marital','job',False)
pivot_table1
```

[41]:
```
job       admin.  blue-collar  entrepreneur  housemaid  management  retired  \
marital
divorced    1293          728           179        161         331      348
married     5506         6699          1074        780        2092     1278
single      3949         1825           203        119         501       92

job       self-employed  services  student  technician  unemployed
marital
divorced            133       532        9         773         124
married             909      2299       42        3681         639
single              379      1136      824        2285         251
```

count the occurrences of each combination of marital , job and y

[42]:
```python
data.groupby(['y','job','marital'])['marital'].count().to_frame()
```

[42]:
```
                          marital
y   job         marital
no  admin.      divorced     1158
                married      4834
                single       3368
    blue-collar divorced      675
                married      6275
…                           …
yes technician  married       386
                single        279
    unemployed  divorced       10
                married        86
                single         48
```

[66 rows x 1 columns]

Visalization

[43]:
```
Pie('marital','Marital','Marital Distribution')
```

[44]:
```
Bar_hue('marital','y','Marital','Y','Marital Distribution')
```

[45]:
```
Bar_2hue('marital','job','Marital','Job',make_subplot=False)
```

[46]:
```
Heatmap(pivot_table,'Marital␣
↪Distribution','Marital','Y',make_subplot=True,feature_h2='Job',pivot2=pivot_table1)
```

What is education distribution?

calculate the value counts for the education column

[47]:
```
data.education.value_counts().to_frame()
```

[47]:

	count
education	
university.degree	13893
high.school	9512
basic.9y	6045
professional.course	5240
basic.4y	4175
basic.6y	2291
illiterate	18

The observation for the education data shows the count of individuals in each category. For ins

count the occurrences of each combination of education and y

[48]:
```
pivot_table = pivot('education','y',False)
pivot_table
```

[48]:

y	no	yes
education		
basic.4y	3747	428
basic.6y	2103	188
basic.9y	5572	473
high.school	8481	1031
illiterate	14	4
professional.course	4645	595
university.degree	11973	1920

count the occurrences of each combination of education and job

[49]:
```
pivot_table1 = pivot('education','job',False)
pivot_table1
```

[49]:
```
job                 admin.  blue-collar  entrepreneur  housemaid  management  \
education
basic.4y               129         2317           137        474         100
basic.6y               173         1425            71         77          85
basic.9y               530         3623           210         94         166
high.school           3366          878           234        174         298
illiterate               1            8             2          1           0
professional.course    375          453           135         59          89
university.degree     6174          548           667        181        2186

job                 retired  self-employed  services  student  technician  \
education
basic.4y                597             93       132       26          58
basic.6y                 75             25       226       13          87
basic.9y                145            220       388       99         384
high.school             276            118      2680      357         872
illiterate                3              3         0        0           0
professional.course     241            168       218       43        3317
university.degree       381            794       323      337        2021

job                 unemployed
education
basic.4y                   112
basic.6y                    34
basic.9y                   186
high.school                259
illiterate                   0
professional.course        142
university.degree          281
```

count the occurrences of each combination of education and marital

[50]:
```
pivot_table2 = pivot('education','marital',False)
pivot_table2
```

[50]:
```
marital             divorced  married  single
education
basic.4y                 489     3233     453
basic.6y                 182     1772     337
basic.9y                 565     4164    1316
high.school             1192     5171    3149
illiterate                 2       15       1
professional.course      657     3161    1422
university.degree       1524     7483    4886
```

count the occurrences of each combination of education , job , marital and y

[51]:
```
data.groupby(['y','job','marital','education'])['education'].count().to_frame()
```

[51]:

```
                                                education
y   job        marital  education
no  admin.     divorced basic.4y                    5
                        basic.6y                   16
                        basic.9y                   72
                        high.school               400
                        professional.course        43
…                                                  …
yes unemployed single   basic.4y                    3
                        basic.9y                    6
                        high.school                12
                        professional.course         2
                        university.degree          25

[370 rows x 1 columns]
```

Visualization

[52]:

```python
Pie('education','Education','Education Distribution')
```

[53]:

```python
Bar_hue('education','y','Education','Y','Education Distribution')
```

[54]:

```python
Bar_2hue('education','job','Education','Job',title='Education␣
↪Distribution',hue2='marital',title_h2='Marital')
```

[55]:

```python
Heatmap(pivot_table,'Education Vs Y␣
↪Categories','Education','Y',make_subplot=False)
```

[56]:

```python
Heatmap(pivot_table1,'Education Vs Job␣
↪Categories','Education','Job',make_subplot=False)
```

[57]:

```python
Heatmap(pivot_table2,'Education Vs Marital␣
↪Categories','Education','Marital',make_subplot=False)
```

What is default distribution

calculate the value counts for the default column

[58]:

```python
data.default.value_counts().to_frame()
```

[58]:

```
         count
default
no       41171
yes          3
```

based on statistic most people don't have credit
no : 41171/41174 = 99.99271384854521 %
yes : 3/41174 = 0.007286151454801573 %

count the occurrences of each combination of default and y

[59]:
```
cross = cross_t('default','y')
cross
```

[59]:
```
y            no   yes
default
no        36532  4639
yes           3     0
```

There are 36,532 observations where 'default' is 'no' and 'y' is 'no'.
There are 4,639 observations where 'default' is 'no' and 'y' is 'yes'.
There are 3 observations where 'default' is 'yes' and 'y' is 'no'.
There are 0 observations where 'default' is 'yes' and 'y' is 'yes'.

count the occurrences of each combination of default and job

[60]:
```
cross1 = cross_t('default','job')
cross1
```

[60]:
```
job      admin.  blue-collar  entrepreneur  housemaid  management  retired  \
default
no        10748         9252          1456       1060        2924     1718
yes           0            0             0          0           0        0

job      self-employed  services  student  technician  unemployed
default
no                1421      3967      875        6737        1013
yes                  0         0        0           2           1
```

There are 10,748 observations where 'default' is 'no' and the job is 'admin.'.
There are 9,252 observations where 'default' is 'no' and the job is 'blue-collar'.
There are 1,456 observations where 'default' is 'no' and the job is 'entrepreneur'.
And so on...

count the occurrences of each combination of default and marital

[61]:
```
cross2 = cross_t('default','marital')
cross2
```

[61]:
```
marital  divorced  married  single
default
no           4611    24996   11564
yes             0        3       0
```

There are 4,611 observations where 'default' is 'no' and the marital status is 'divorced'.
There are 24,996 observations where 'default' is 'no' and the marital status is 'married'.
There are 11,564 observations where 'default' is 'no' and the marital status is 'single'.

count the occurrences of each combination of default and education

[62]:
```
cross3 = cross_t('default','education')
cross3
```

[62]:
```
education  basic.4y  basic.6y  basic.9y  high.school  illiterate  \
default
no             4175      2291      6045         9511          18
yes               0         0         0            1           0

education  professional.course  university.degree
default
no                        5238              13893
yes                          2                  0
```

```
There are 4,175 observations where 'default' is 'no' and the education level is 'basic.4y'.
There are 2,291 observations where 'default' is 'no' and the education level is 'basic.6y'.
There are 6,045 observations where 'default' is 'no' and the education level is 'basic.9y'.
There are 9,511 observations where 'default' is 'no' and the education level is 'high.school'.
There is 1 observation where 'default' is 'yes' and the education level is 'high.school'.
There are 18 observations where 'default' is 'no' and the education level is 'illiterate'.
There are 5,238 observations where 'default' is 'no' and the education level is 'professional.c
There are 13,893 observations where 'default' is 'no' and the education level is 'university.de
There are 2 observations where 'default' is 'yes' and the education level is 'professional.cou
```

count the occurrences of each combination of default , education , job , marital and y

[63]:
```python
data.groupby(['y','job','marital','education','default'])['default'].count().
↪to_frame()
```

[63]:
```
                                                      default
y   job        marital  education           default
no  admin.     divorced basic.4y            no              5
                        basic.6y            no             16
                        basic.9y            no             72
                        high.school         no            400
                        professional.course no             43
…                                                         …
yes unemployed single   basic.4y            no              3
                        basic.9y            no              6
                        high.school         no             12
                        professional.course no              2
                        university.degree   no             25

[372 rows x 1 columns]
```

Visualization

[64]:
```python
Pie('default','Default','Default Distribution')
```

[65]:
```python
Bar_hue('default','y','Default','Y','Default Distribution')
```

[66]:
```python
Bar_2hue('default','job','Default','Job',title='Default␣
↪Distribution',hue2='marital',title_h2='Marital')
```

[67]:
```
Bar_2hue('default','education','Default','Education',make_subplot=False)
```

[68]:
```
Heatmap(cross,'Default␣
↪Distribution','Default','Y',make_subplot=True,feature_h2='Job',pivot2=cross1)
```

[69]:
```
Heatmap(cross2,'Default␣
↪Distribution','Default','Marital',make_subplot=True,feature_h2='Education',pivot2=cross3)
```

What is housing distribution

calculate the value counts for the housing column

[70]:
```
data.housing.value_counts().to_frame()
```

[70]:

	count
housing	
yes	22560
no	18614

There are 22,560 observations where 'housing' is 'yes'.
There are 18,614 observations where 'housing' is 'no'.

count the occurrences of each combination of housing and y

[71]:
```
cross = cross_t('housing','y')
cross
```

[71]:

y	no	yes
housing		
no	16589	2025
yes	19946	2614

There are 16,589 observations where 'housing' is 'no' and 'y' is 'no'.
There are 2,025 observations where 'housing' is 'no' and 'y' is 'yes'.
There are 19,946 observations where 'housing' is 'yes' and 'y' is 'no'.
There are 2,614 observations where 'housing' is 'yes' and 'y' is 'yes'.

count the occurrences of each combination of housing and job

[72]:
```
cross1 = cross_t('housing','job')
cross1
```

[72]:

job	admin.	blue-collar	entrepreneur	housemaid	management	retired	\
housing							
no	4787	4302	641	491	1363	782	
yes	5961	4950	815	569	1561	936	

job	self-employed	services	student	technician	unemployed
housing					
no	641	1817	381	2979	430
yes	780	2150	494	3760	584

There are 4,787 observations where 'housing' is 'no' and the job is 'admin.'.
There are 4,302 observations where 'housing' is 'no' and the job is 'blue-collar'.
There are 641 observations where 'housing' is 'no' and the job is 'entrepreneur'.
There are 491 observations where 'housing' is 'no' and the job is 'housemaid'.
There are 1,363 observations where 'housing' is 'no' and the job is 'management'.
And so on...

count the occurrences of each combination of housing and marital

[73]:
```python
cross2 = cross_t('housing','marital')
cross2
```

[73]:
```
marital  divorced  married  single
housing
no           2092    11427    5095
yes          2519    13572    6469
```

There are 2,092 observations where 'housing' is 'no' and the marital status is 'divorced'.
There are 11,427 observations where 'housing' is 'no' and the marital status is 'married'.
There are 5,095 observations where 'housing' is 'no' and the marital status is 'single'.
There are 2,519 observations where 'housing' is 'yes' and the marital status is 'divorced'.
There are 13,572 observations where 'housing' is 'yes' and the marital status is 'married'.
There are 6,469 observations where 'housing' is 'yes' and the marital status is 'single'.

count the occurrences of each combination of housing and education

[74]:
```python
cross3 = cross_t('housing','education')
cross3
```

[74]:
```
education  basic.4y  basic.6y  basic.9y  high.school  illiterate  \
housing
no             1954      1069      2743         4362           8
yes            2221      1222      3302         5150          10

education  professional.course  university.degree
housing
no                        2279               6199
yes                       2961               7694
```

There are 1,954 observations where 'housing' is 'no' and the education level is 'basic.4y'.
There are 1,069 observations where 'housing' is 'no' and the education level is 'basic.6y'.
There are 2,743 observations where 'housing' is 'no' and the education level is 'basic.9y'.
There are 4,362 observations where 'housing' is 'no' and the education level is 'high.school'.
There are 8 observations where 'housing' is 'no' and the education level is 'illiterate'.
There are 2,279 observations where 'housing' is 'no' and the education level is 'professional.
There are 6,199 observations where 'housing' is 'no' and the education level is 'university.deg
There are similar counts for each education level when 'housing' is 'yes'.

count the occurrences of each combination of housing and default

[75]:
```
cross4 = cross_t('housing','default')
cross4
```

[75]:
```
default     no  yes
housing
no       18612    2
yes      22559    1
```

There are 18,612 observations where 'housing' is 'no' and 'default' is 'no'.
There are 2 observations where 'housing' is 'no' and 'default' is 'yes'.
There are 22,559 observations where 'housing' is 'yes' and 'default' is 'no'.
There is 1 observation where 'housing' is 'yes' and 'default' is 'yes'.

count the occurrences of each combination of housing , default , education , job , marital and

[76]:
```
data.groupby(['y','job','marital','education','default','housing'])['housing'].
    ↪count().to_frame()
```

[76]:
```
                                                             housing
y   job        marital  education           default housing
no  admin.     divorced basic.4y            no      no             3
                                                    yes            2
                        basic.6y            no      no            11
                                                    yes            5
                        basic.9y            no      no            29
…                                                                …
yes unemployed single   high.school         no      yes            9
                        professional.course no      no             1
                                                    yes            1
                        university.degree   no      no             5
                                                    yes           20

[696 rows x 1 columns]
```

Visualization

[77]:
```
Pie('housing','Housing','Housing Distribution')
```

[78]:
```
Bar_hue('housing','y','Housing','Y','Housing Distribution')
```

[79]:
```
Bar_2hue('housing','job','Housing','Job',title='Housing␣
    ↪Distribution',hue2='marital',title_h2='Marital')
```

[80]:
```
Bar_2hue('housing','education','Housing','Education',title='Housing␣
    ↪Distribution',hue2='default',title_h2='Default')
```

[81]:
```
Heatmap(cross,'Housing␣
    ↪Distribution','Housing','Y',make_subplot=True,feature_h2='Job',pivot2=cross1)
```

[82]:
```
Heatmap(cross2,'Housing␣
↪Distribution','Housing','Marital',make_subplot=True,feature_h2='Education',pivot2=cross3)
```

[83]:
```
Heatmap(cross4,'Housing VS  Default␣
↪Categories','Housing','Default',make_subplot=False)
```

What is loan distribution

calculate the value counts for the loan column

[84]:
```
data.loan.value_counts().to_frame()
```

[84]:
```
      count
loan
no    34926
yes    6248
```

There are 34,926 observations where 'loan' is 'no'.
There are 6,248 observations where 'loan' is 'yes'.

count the occurrences of each combination of loan and y

[85]:
```
cross = cross_t('loan','y')
cross
```

[85]:
```
y        no   yes
loan
no    30970  3956
yes    5565   683
```

There are 30,970 observations where 'loan' is 'no' and 'y' is 'no'.
There are 3,956 observations where 'loan' is 'no' and 'y' is 'yes'.
There are 5,565 observations where 'loan' is 'yes' and 'y' is 'no'.
There are 683 observations where 'loan' is 'yes' and 'y' is 'yes'.

count the occurrences of each combination of loan and job

[86]:
```
cross1 = cross_t('loan','job')
cross1
```

[86]:
```
job   admin.  blue-collar  entrepreneur  housemaid  management  retired  \
loan
no      8981         7886          1250        906        2485     1478
yes     1767         1366           206        154         439      240

job   self-employed  services  student  technician  unemployed
loan
no             1226      3366      733        5750         865
yes             195       601      142         989         149
```

There are 8,981 observations where 'loan' is 'no' and the job is 'admin.'.

```
There are 7,886 observations where 'loan' is 'no' and the job is 'blue-collar'.
There are 1,250 observations where 'loan' is 'no' and the job is 'entrepreneur'.
There are 906 observations where 'loan' is 'no' and the job is 'housemaid'.
There are 2,485 observations where 'loan' is 'no' and the job is 'management'.
And so on...
```

count the occurrences of each combination of loan and marital

[87]:
```python
cross2 = cross_t('loan','marital')
cross2
```

[87]:

marital	divorced	married	single
loan			
no	3936	21214	9776
yes	675	3785	1788

```
There are 3,936 observations where 'loan' is 'no' and the marital status is 'divorced'.
There are 21,214 observations where 'loan' is 'no' and the marital status is 'married'.
There are 9,776 observations where 'loan' is 'no' and the marital status is 'single'.
There are 675 observations where 'loan' is 'yes' and the marital status is 'divorced'.
There are 3,785 observations where 'loan' is 'yes' and the marital status is 'married'.
There are 1,788 observations where 'loan' is 'yes' and the marital status is 'single'.
```

count the occurrences of each combination of loan and education

[88]:
```python
cross3 = cross_t('loan','education')
cross3
```

[88]:

education	basic.4y	basic.6y	basic.9y	high.school	illiterate	\
loan						
no	3551	1961	5162	8069	15	
yes	624	330	883	1443	3	

education	professional.course	university.degree
loan		
no	4447	11721
yes	793	2172

```
There are 3,551 observations where 'loan' is 'no' and the education level is 'basic.4y'.
There are 1,961 observations where 'loan' is 'no' and the education level is 'basic.6y'.
There are 5,162 observations where 'loan' is 'no' and the education level is 'basic.9y'.
There are 8,069 observations where 'loan' is 'no' and the education level is 'high.school'.
There are 15 observations where 'loan' is 'no' and the education level is 'illiterate'.
There are 4,447 observations where 'loan' is 'no' and the education level is 'professional.cou
There are 11,721 observations where 'loan' is 'no' and the education level is 'university.degre
There are similar counts for each education level when 'loan' is 'yes'.
```

count the occurrences of each combination of loan and default

[89]:
```python
cross4 = cross_t('loan','default')
cross4
```

[89]:
```
default     no  yes
loan
no       34923    3
yes       6248    0
```

There are 34,923 observations where 'loan' is 'no' and 'default' is 'no'.
There are 3 observations where 'loan' is 'no' and 'default' is 'yes'.
There are 6,248 observations where 'loan' is 'yes' and 'default' is 'no'.
There are 0 observations where 'loan' is 'yes' and 'default' is 'yes'.

count the occurrences of each combination of loan and housing

[90]:
```python
cross5 = cross_t('loan','housing')
cross5
```

[90]:
```
housing     no    yes
loan
no       16057  18869
yes       2557   3691
```

There are 16,057 observations where 'loan' is 'no' and 'housing' is 'no'.
There are 18,869 observations where 'loan' is 'no' and 'housing' is 'yes'.
There are 2,557 observations where 'loan' is 'yes' and 'housing' is 'no'.
There are 3,691 observations where 'loan' is 'yes' and 'housing' is 'yes'.

count the occurrences of each combination of loan , housing , default , education , job , mar

[91]:
```python
data.
 ↪groupby(['y','job','marital','education','default','housing','loan'])['loan'].
 ↪count().to_frame()
```

[91]:
```
                                                                 loan
y   job        marital  education           default housing loan
no  admin.     divorced basic.4y            no      no      no      3
                                                    yes     no      2
                        basic.6y            no      no      no      7
                                                            yes     4
                                                    yes     no      5
…                                                                 …
yes unemployed single   professional.course no      no      no      1
                                                    yes     yes     1
                        university.degree   no      no      no      5
                                                    yes     no     18
                                                            yes     2

[1170 rows x 1 columns]
```

Visualization

[92]:
```
Pie('loan','Loan','Loan Distribution')
```

[93]:
```
Bar_hue('loan','y','Loan','Y','Loan Distribution')
```

[94]:
```
Bar_2hue('loan','job','Loan','Job',title='Loan␣
 ↪Distribution',hue2='marital',title_h2='Marital')
```

[95]:
```
Bar_2hue('loan','education','Loan','Education',title='Loan␣
 ↪Distribution',hue2='default',title_h2='Default')
```

[96]:
```
Bar_2hue('loan','housing','Loan','Housing',make_subplot=False)
```

[97]:
```
Heatmap(cross,'Loan␣
 ↪Distribution','Loan','Y',make_subplot=True,feature_h2='Job',pivot2=cross1)
```

[98]:
```
Heatmap(cross2,'Loan␣
 ↪Distribution','Loan','Marital',make_subplot=True,feature_h2='Education',pivot2=cross3)
```

[99]:
```
Heatmap(cross4,'Loan␣
 ↪Distribution','Loan','Default',make_subplot=True,feature_h2='Housing',pivot2=cross5)
```

What is contact distribution

calculate the value counts for the contact column

[100]:
```
data.contact.value_counts().to_frame()
```

[100]:

	count
contact	
cellular	26134
telephone	15040

There are 26,134 observations where the contact method is 'cellular'.
There are 15,040 observations where the contact method is 'telephone'.

count the occurrences of each combination of contact and y

[101]:
```
cross = cross_t('contact','y')
cross
```

[101]:

y	no	yes
contact		
cellular	22282	3852
telephone	14253	787

There are 22,282 observations where the contact method is 'cellular' and the outcome 'y' is 'no
There are 3,852 observations where the contact method is 'cellular' and the outcome 'y' is 'yes
There are 14,253 observations where the contact method is 'telephone' and the outcome 'y' is 'n
There are 787 observations where the contact method is 'telephone' and the outcome 'y' is 'yes

count the occurrences of each combination of contact and job

[102]:
```
cross1 = cross_t('contact','job')
cross1
```

[102]:
```
job        admin.  blue-collar  entrepreneur  housemaid  management  retired  \
contact
cellular     7290         5090           855        640        1902     1231
telephone    3458         4162           601        420        1022      487

job        self-employed  services  student  technician  unemployed
contact
cellular             893      2309      671        4633         620
telephone            528      1658      204        2106         394
```

There are 7,290 observations where the contact method is 'cellular' and the job is 'admin.'.
There are 5,090 observations where the contact method is 'cellular' and the job is 'blue-colla
There are 855 observations where the contact method is 'cellular' and the job is 'entrepreneur
There are 640 observations where the contact method is 'cellular' and the job is 'housemaid'.
There are 1,902 observations where the contact method is 'cellular' and the job is 'management
There are similar counts for each occupation when the contact method is 'cellular', and simila

count the occurrences of each combination of contact and marital

[103]:
```
cross2 = cross_t('contact','marital')
cross2
```

[103]:
```
marital    divorced  married  single
contact
cellular       2907    15253    7974
telephone      1704     9746    3590
```

There are 2,907 observations where the contact method is 'cellular' and the marital status is
There are 15,253 observations where the contact method is 'cellular' and the marital status is
There are 7,974 observations where the contact method is 'cellular' and the marital status is
There are 1,704 observations where the contact method is 'telephone' and the marital status is
There are 9,746 observations where the contact method is 'telephone' and the marital status is
There are 3,590 observations where the contact method is 'telephone' and the marital status is

count the occurrences of each combination of contact and education

[104]:
```
cross3 = cross_t('contact','education')
cross3
```

[104]:
```
education  basic.4y  basic.6y  basic.9y  high.school  illiterate  \
contact
cellular       2350      1247      3452         5925          15
telephone      1825      1044      2593         3587           3

education  professional.course  university.degree
```

```
contact
cellular                     3475                9670
telephone                    1765                4223
```

```
There are 2,350 observations where the contact method is 'cellular' and the education level is
There are 1,247 observations where the contact method is 'cellular' and the education level is
There are 3,452 observations where the contact method is 'cellular' and the education level is
There are 5,925 observations where the contact method is 'cellular' and the education level is
There are 15 observations where the contact method is 'cellular' and the education level is 'il
There are 3,475 observations where the contact method is 'cellular' and the education level is
There are 9,670 observations where the contact method is 'cellular' and the education level is
```

count the occurrences of each combination of contact and default

[105]:
```
cross4 = cross_t('contact','default')
cross4
```

[105]:
```
default       no  yes
contact
cellular   26131    3
telephone  15040    0
```

```
There are 26,131 observations where 'contact' is 'cellular' and 'default' is 'no'.
There are 3 observations where 'contact' is 'cellular' and 'default' is 'yes'.
There are 15,040 observations where 'contact' is 'telephone' and 'default' is 'no'.
There are 0 observations where 'contact' is 'telephone' and 'default' is 'yes'.
```

count the occurrences of each combination of contact and housing

[106]:
```
cross5 = cross_t('contact','housing')
cross5
```

[106]:
```
housing       no    yes
contact
cellular   11047  15087
telephone   7567   7473
```

```
There are 11,047 observations where 'housing' is 'no' and 'contact' is 'cellular'.
There are 15,087 observations where 'housing' is 'yes' and 'contact' is 'cellular'.
There are 7,567 observations where 'housing' is 'no' and 'contact' is 'telephone'.
There are 7,473 observations where 'housing' is 'yes' and 'contact' is 'telephone'.
```

count the occurrences of each combination of contact and loan

[107]:
```
cross6 = cross_t('contact','loan')
cross6
```

[107]:
```
loan          no   yes
contact
cellular   22073  4061
```

```
telephone  12853  2187
```

```
There are 22,073 observations where 'loan' is 'no' and 'contact' is 'cellular'.
There are 4,061 observations where 'loan' is 'yes' and 'contact' is 'cellular'.
There are 12,853 observations where 'loan' is 'no' and 'contact' is 'telephone'.
There are 2,187 observations where 'loan' is 'yes' and 'contact' is 'telephone'.
```

count the occurrences of each combination of contact , loan , housing , default , education , j

[108]:
```
data.
↪groupby(['y','job','marital','education','default','housing','loan','contact'])['contact'].
↪count().to_frame()
```

[108]:
```
contact
y   job        marital  education           default housing loan contact
no  admin.     divorced basic.4y            no      no      no   cellular
2
                                                                 telephone

1
                                                    yes     no   cellular

2
                        basic.6y            no      no      no   cellular
5
                                                                 telephone

2
…
…
yes unemployed single   professional.course no      yes     yes  cellular
1
                        university.degree   no      no      no   cellular
5
                                                    yes     no   cellular

16
                                                                 telephone

2
                                                            yes  cellular

2

[1937 rows x 1 columns]
```

Visualization

[109]:
```
Pie('contact','Contact','Contact Distribution')
```

[110]:
```
Bar_hue('contact','y','Contact','Y','Contact Distribution')
```

[111]:
```
Bar_2hue('contact','job','Contact','Job',title='Contact␣
↪Distribution',hue2='marital',title_h2='Marital')
```

[112]:
```
Bar_2hue('contact','education','Contact','Education',title='Contact␣
↪Distribution',hue2='default',title_h2='Default')
```

[113]:
```
Bar_2hue('contact','housing','Contact','Housing',title='Contact␣
↪Distribution',hue2='loan',title_h2='Loan')
```

[114]:
```
Heatmap(cross,'Contcat␣
↪Distribution','Contcat','Y',make_subplot=True,feature_h2='Job',pivot2=cross1)
```

[115]:
```
Heatmap(cross2,'Contcat␣
↪Distribution','Contcat','Marital',make_subplot=True,feature_h2='Education',pivot2=cross3)
```

[116]:
```
Heatmap(cross4,'Contcat␣
↪Distribution','Contcat','Default',make_subplot=True,feature_h2='Housing',pivot2=cross5)
```

[117]:
```
Heatmap(cross6,'Contact VS  Loan␣
↪Categories','Contact','Loan',make_subplot=False)
```

What is month distribution

```
calculate the value counts for the month column
```

[118]:
```
data.month.value_counts().to_frame()
```

[118]:
```
       count
month
may    13766
jul     7169
aug     6175
jun     5318
nov     4100
apr     2631
oct      717
sep      570
mar      546
dec      182
```

```
There are 13,766 observations in the month of May.
There are 7,169 observations in the month of July.
There are 6,175 observations in the month of August.
There are 5,318 observations in the month of June.
There are 4,100 observations in the month of November.
There are 2,631 observations in the month of April.
There are 717 observations in the month of October.
There are 570 observations in the month of September.
There are 546 observations in the month of March.
There are 182 observations in the month of December.
```

```
count the occurrences of each combination of month and y
```

[119]:
```
cross = cross_t('month','y')
cross
```

[119]:
```
y         no  yes
month
apr     2092  539
aug     5520  655
dec       93   89
jul     6521  648
jun     4759  559
mar      270  276
may    12880  886
nov     3684  416
oct      402  315
sep      314  256
```

In April, there are 2,092 observations where the outcome 'y' is 'no', and 539 observations whe
In August, there are 5,520 observations where the outcome 'y' is 'no', and 655 observations wh
In December, there are 93 observations where the outcome 'y' is 'no', and 89 observations wher
Similar counts are provided for each month and each outcome category.

count the occurrences of each combination of month and contact

[120]:
```
cross1 = cross_t('month','contact')
cross1
```

[120]:
```
contact  cellular  telephone
month
apr          2444        187
aug          5906        269
dec           149         33
jul          6092       1077
jun           820       4498
mar           486         60
may          5517       8249
nov          3675        425
oct           563        154
sep           482         88
```

In April, there are 2,444 observations where the contact method is 'cellular', and 187 observa
In August, there are 5,906 observations where the contact method is 'cellular', and 269 observa
In December, there are 149 observations where the contact method is 'cellular', and 33 observat
Similar counts are provided for each month and each contact method.

Visualization

[121]:
```
Pie('month','Month','Month Distribution')
```

[122]:
```
Bar_hue('month','y','Month','Y','Month Distribution')
```

[123]:
```
Bar_2hue('month','contact','Month','Contact',make_subplot=False)
```

[124]:
```
Heatmap(cross,'Month VS Y Categories','Month','Y',make_subplot=False)
```

[125]:
```
Heatmap(cross1,'Month VS Contact␣
↪Categories','Month','Contact',make_subplot=False)
```

[126]:
```
monthly_duration_by_contact = data.groupby(['month', 'contact'])['duration'].
↪sum().reset_index()
custom_colors = {
    'cellular': 'rgb(255, 127, 14)',
    'telephone': 'rgb(255, 0, 0)'
}
fig = px.area(monthly_duration_by_contact, x='month', y='duration',␣
↪color='contact',
              color_discrete_map=custom_colors)
fig.update_xaxes(title='Month')
fig.update_yaxes(title='Total Duration')
fig.update_layout(title_text="Monthly Duration by Contact Type ", title_x=0.5,
                  title_font=dict(size=20),template='plotly_dark')

fig.show()
```

[127]:
```
fig = go.Figure()
for contact_type in monthly_duration_by_contact['contact'].unique():
    data_subset =␣
↪monthly_duration_by_contact[monthly_duration_by_contact['contact'] ==␣
↪contact_type]
    fig.add_trace(go.Scatter(x=data_subset['month'], y=data_subset['duration'],
                             mode='lines',
                             name=contact_type,
                             stackgroup='one',
                             line=dict(color=custom_colors[contact_type])))
fig.update_layout(title='Monthly Duration by Contact Type',title_x=.
↪5,title_font=dict(size=20),
                  xaxis_title='Month',
                  yaxis_title='Total Duration',
                  template='plotly_dark')

fig.show()
```

What is day_of_week distribution

calculate the value counts for the day_of_week column

[128]:
```
data.day_of_week.value_counts().to_frame()
```

```
[128]:              count
       day_of_week
       thu           8617
       mon           8511
       wed           8134
       tue           8086
       fri           7826
```

There are 8,617 observations that occurred on Thursday ('thu').
There are 8,511 observations that occurred on Monday ('mon').
There are 8,134 observations that occurred on Wednesday ('wed').
There are 8,086 observations that occurred on Tuesday ('tue').
There are 7,826 observations that occurred on Friday ('fri').

count the occurrences of each combination of day_of_week and y

```
[129]: cross = cross_t('day_of_week','y')
       cross
```

```
[129]: y              no   yes
       day_of_week
       fri          6980   846
       mon          7664   847
       thu          7573  1044
       tue          7133   953
       wed          7185   949
```

On Fridays, there are 6,980 observations where the outcome 'y' is 'no', and 846 observations wh
On Mondays, there are 7,664 observations where the outcome 'y' is 'no', and 847 observations wh
On Thursdays, there are 7,573 observations where the outcome 'y' is 'no', and 1,044 observation
Similar counts are provided for each day of the week and each outcome category.

count the occurrences of each combination of day_of_week and month

```
[130]: cross1 = cross_t('day_of_week','month')
       cross1
```

```
[130]: month        apr   aug  dec   jul   jun  mar   may  nov  oct  sep
       day_of_week
       fri          610  1070   24  1012  1147   94  2857  755  142  115
       mon          702  1221   53  1515  1251  143  2641  766  129   90
       thu          768  1346   45  1668   967   99  2536  903  163  122
       tue          251  1295   25  1517   970  140  2809  813  148  118
       wed          300  1243   35  1457   983   70  2923  863  135  125
```

There are 610 observations that occurred on Fridays in April ('apr').
There are 1,070 observations that occurred on Fridays in August ('aug').
There are 24 observations that occurred on Fridays in December ('dec').
There are similar counts for each combination of day of the week and month.

count the occurrences of each combination of day_of_week and contact

```
[131]: cross2 = cross_t('day_of_week','contact')
       cross2
```

```
[131]: contact      cellular  telephone
       day_of_week
       fri              4644       3182
       mon              5533       2978
       thu              5801       2816
       tue              5104       2982
       wed              5052       3082
```

On Fridays, there are 4,644 observations where the contact method is 'cellular', and 3,182 obse
On Mondays, there are 5,533 observations where the contact method is 'cellular', and 2,978 obse
On Thursdays, there are 5,801 observations where the contact method is 'cellular', and 2,816 o
Similar counts are provided for each day of the week and each contact method.

Visualization

```
[132]: Pie('day_of_week','Day','Day Distribution')
```

```
[133]: Bar_hue('day_of_week','y','Day','Y','Day Distribution')
```

```
[134]: Bar_2hue('day_of_week','month','Day','Month',title='Default␣
        ↪Distribution',hue2='contact',title_h2='Contact')
```

```
[135]: Heatmap(cross,'Day␣
        ↪Distribution','Day','Y',make_subplot=True,feature_h2='Month',pivot2=cross1)
```

```
[136]: Heatmap(cross2,'Day VS  Contact Categories','Day','Contact',make_subplot=False)
```

```
[137]: day_duration_by_contact = data.groupby(['day_of_week', 'contact'])['duration'].
        ↪sum().reset_index()
       fig = px.area(day_duration_by_contact, x='day_of_week', y='duration',␣
        ↪color='contact',
                     color_discrete_map=custom_colors)
       fig.update_xaxes(title='Day')
       fig.update_yaxes(title='Total Duration')
       fig.update_layout(title_text="Days Duration by Contact Type ", title_x=0.5,
                         title_font=dict(size=20),template='plotly_dark')

       fig.show()
```

```
[138]: fig = go.Figure()
       for contact_type in day_duration_by_contact['contact'].unique():
           data_subset = day_duration_by_contact[day_duration_by_contact['contact'] ==␣
        ↪contact_type]
           fig.add_trace(go.Scatter(x=data_subset['day_of_week'],␣
        ↪y=data_subset['duration'],
```

```
                                    mode='lines',
                                    name=contact_type,
                                    stackgroup='one',
                                    line=dict(color=custom_colors[contact_type]))
fig.update_layout(title='Days Duration by Contact Type',title_x=.
 ↪5,title_font=dict(size=20),
                      xaxis_title='Day',
                      yaxis_title='Total Duration',
                      template='plotly_dark')

fig.show()
```

What is duration distribution?

Find the minimum duration

[139]:
```
data.duration.min()
```

[139]: 0

Find the maximum duration

[140]:
```
data.duration.max()
```

[140]: 4918

Find the top 5 most frequent duration

[141]:
```
data.duration.value_counts().to_frame().head()
```

[141]:

duration	count
90	170
85	170
136	167
73	167
124	163

calculate the mean duration for each category in the contact column

[142]:
```
pivot_table = pivot('duration','contact')
pivot_table
```

[142]:

contact	duration
cellular	263.569067
telephone	249.208976

Visualization

[143]:
```
Boxplot_outlier('duration','Duration Distribution')
```

Observation: Based on the figure, it appears that the duration column contains some outliers.

[144]:
```
mean_plot(pivot_table,'duration','contact','Duration','Contact')
```

What is campaign distribution?

Find the minimum campaign

[145]:
```
data.campaign.min()
```

[145]: 1

Find the maximum campaign

[146]:
```
data.campaign.max()
```

[146]: 56

Find the top 5 most frequent duration

[147]:
```
data.campaign.value_counts().to_frame().head()
```

[147]:

campaign	count
1	17632
2	10568
3	5340
4	2650
5	1599

count the occurrences of each combination of campaign and contact

[148]:
```
cross = cross_t('campaign','contact')
cross
```

[148]:

contact campaign	cellular	telephone
1	11753	5879
2	6675	3893
3	3303	2037
4	1583	1067
5	998	601
6	577	402
7	359	270
8	219	181
9	148	135
10	116	109
11	101	76
12	54	71
13	49	43
14	31	38

```
15              22          29
16              17          34
17              30          28
18              11          22
19              11          15
20              16          14
21               6          18
22               7          10
23               6          10
24               8           7
25               3           5
26               2           6
27               5           6
28               2           6
29               5           5
30               5           2
31               2           5
32               0           4
33               4           0
34               2           1
35               2           3
37               0           1
39               0           1
40               1           1
41               0           1
42               0           2
43               1           1
56               0           1
```

Average between campaign and duration

[149]:
```
pivot_table = pivot('duration','campaign')
pivot_table
```

[149]:
```
            duration
campaign
1         256.804049
2         279.706945
3         270.044569
4          251.46566
5         227.759225
6         225.955056
7         223.330684
8            189.525
9         211.526502
10        208.706667
11        207.723164
12           185.288
```

```
13         175.282609
14         134.594203
15              152.0
16         117.352941
17         199.258621
18          85.424242
19         164.692308
20          62.233333
21          82.583333
22         113.529412
23           129.1875
24         111.466667
25             45.875
26            305.625
27         100.909091
28             118.25
29              118.0
30               69.0
31          33.571429
32              30.25
33               37.5
34               37.0
35               49.6
37               17.0
39               44.0
40               15.5
41               25.0
42              135.5
43               40.5
56              261.0
```

Visualization

[150]:
```
Bar_hue('campaign','contact','Campaign','Contact','Cmpaign Distribution')
```

[151]:
```
Boxplot_outlier('campaign','Campaign Distribution')
```

[152]:
```
fig = go.Figure()
scatter_trace = go.Scatter(
    x=pivot_table.index,
    y=pivot_table['duration'],
    mode='lines+markers',
    marker=dict(
        size=10,
        color='blue',
        symbol='circle',
        opacity=0.8
    ),
```

```
    line=dict(
        color='red',
        width=2
    )
)
fig.add_trace(scatter_trace)
fig.update_layout(
    title_text='Average between Campaign and Duration',
    title_x=0.5,
    title_font=dict(size=20),
    xaxis_title='Campaign',
    yaxis_title='Average Duration',
    font=dict(size=15),
    width=800,
    height=700,
    xaxis=dict(tickangle=-90),
    template='plotly_dark'
)
fig.update_annotations(font=dict(size=20))
fig.show()
```

[153]:
```
if not pd.api.types.is_numeric_dtype(data['duration']):
    data['duration'] = pd.to_numeric(data['duration'], errors='coerce')

grouped_data = data.groupby(['y', 'campaign'])['duration'].mean().reset_index()
fig = go.Figure()
for category, group in grouped_data.groupby('y'):
    fig.add_trace(go.Scatter(
        x=group['campaign'],
        y=group['duration'],
        mode='markers',
        marker=dict(
            size=group['duration'],
            sizemode='area',
            sizeref=2. * max(group['duration']) / (40. ** 2),
            color=group['campaign'],
            opacity=0.7,
            line=dict(width=0.5, color='DarkSlateGrey')
        ),
        name=category
    ))
fig.update_layout(
    title='Mean Duration VS. Campaign',
    title_x=.5,
    title_font=dict(size=20),
    xaxis_title='Campaign',
    yaxis_title='Mean Duration',
```

```
    template='plotly_dark'
)
fig.show()
```

[154]:
```
grouped_data = data.groupby(['contact', 'campaign'])['duration'].mean().
 ↪reset_index()
fig = go.Figure()
for category, group in grouped_data.groupby('contact'):
    fig.add_trace(go.Scatter(
        x=group['campaign'],
        y=group['duration'],
        mode='markers',
        marker=dict(size=group['duration'], sizemode='area', sizeref=2.
 ↪*max(group['duration'])/(40.**2),
                    color=group['campaign'],
                    opacity=0.7,
                    line=dict(width=0.5, color='DarkSlateGrey')),
        name=category
    ))
fig.update_layout(title='Mean Duration VS. Campaign',title_x=.
 ↪5,title_font=dict(size=20),
                  xaxis_title='Campaign',
                  yaxis_title='Mean Duration',
                 template='plotly_dark')
fig.show()
```

Observation: Based on the figure, it appears that the campaign column contains some outliers.

What is pdays distribution?

Find the minimum pdays

[155]:
```
data.pdays.min()
```

[155]: 0

Find the maximum pdays

[156]:
```
data.pdays.max()
```

[156]: 999

Find the top 5 most frequent pdays

[157]:
```
data.pdays.value_counts().to_frame().head()
```

[157]:

	count
pdays	
999	39659
3	439

```
6        412
4        118
9         64
```

calculate the mean pdays for each category in the contact column

[158]:
```
pivot_table = pivot('pdays','contact')
pivot_table
```

[158]:
```
                pdays
contact
cellular   945.728859
telephone  991.540891
```

Visualization

[159]:
```
mean_plot(pivot_table,'pdays','contact','Pdays','Contact')
```

What is previous distribution?

Find the minimum previous

[160]:
```
data.previous.min()
```

[160]: 0

Find the maximum previous

[161]:
```
data.previous.max()
```

[161]: 7

Find the top 5 most frequent previous

[162]:
```
data.previous.value_counts().to_frame().head()
```

[162]:
```
          count
previous
0         35549
1          4561
2           754
3           216
4            70
```

count the occurrences of each combination of previous and contact

[163]:
```
cross = cross_t('previous','contact')
cross
```

[163]:
```
contact   cellular  telephone
previous
```

```
0          20912   14637
1           4240     321
2            691      63
3            205      11
4             63       7
5             17       1
6              5       0
7              1       0
```

Visualization

[164]:
```
Bar_hue('previous','contact','Previous','Contact','Previous Distribution')
```

[165]:
```
Heatmap(cross,'Previous VS Contact␣
 ↪Categories','Previous','Contact',make_subplot=False)
```

What is poutcome distribution?

calculate the value counts for the poutcome column

[166]:
```
data.poutcome.value_counts().to_frame()
```

[166]:

poutcome	count
nonexistent	35549
failure	4252
success	1373

count the occurrences of each combination of poutcome and y

[167]:
```
cross = cross_t('poutcome','y')
cross
```

[167]:

poutcome \ y	no	yes
failure	3647	605
nonexistent	32409	3140
success	479	894

count the occurrences of each combination of poutcome and contact

[168]:
```
cross1 = cross_t('poutcome','contact')
cross1
```

[168]:

poutcome \ contact	cellular	telephone
failure	3952	300
nonexistent	20912	14637
success	1270	103

Visualization

[169]:
```
Pie('poutcome','Poutcome','Poutcome Distribution')
```

[170]:
```
Bar_hue('poutcome','y','Poutcome','Y','Poutcome Distribution')
```

[171]:
```
Bar_2hue('poutcome','contact','Poutcome','Contact',make_subplot=False)
```

[172]:
```
Heatmap(cross,'Poutcome␣
↪Distribution','Poutcome','Y',make_subplot=True,feature_h2='Contact',pivot2=cross1)
```

What is emp.var.rate distribution?

Find the minimum emp.var.rate

[173]:
```
data['emp.var.rate'].min()
```

[173]:
```
-3.4
```

Find the maximum emp.var.rate

[174]:
```
data['emp.var.rate'].max()
```

[174]:
```
1.4
```

Visualization

[175]:
```
Boxplot_outlier('emp.var.rate','Emp.Var.Rate Distribution')
```

What is cons.price.idx distribution?

Find the minimum cons.price.idx

[176]:
```
data['cons.price.idx'].min()
```

[176]:
```
92.201
```

Find the maximum cons.price.idx

[177]:
```
data['cons.price.idx'].max()
```

[177]:
```
94.767
```

Visualization

[178]:
```
Boxplot_outlier('cons.price.idx','Cons.Price.Idx Distribution')
```

What is cons.conf.idx distribution?

Find the minimum cons.conf.idx

[179]:
```
data['cons.conf.idx'].min()
```

[179]:
```
-50.8
```

Find the maximum cons.conf.idx

[180]:
```python
data['cons.conf.idx'].max()
```

[180]: -26.9

Visualization

[181]:
```python
Boxplot_outlier('cons.conf.idx','Cons.Conf.Idx Distribution')
```

What is euribor3m distribution?

Find the minimum euribor3m

[182]:
```python
data.euribor3m.min()
```

[182]: 0.634

Find the maximum euribor3m

[183]:
```python
data.euribor3m.max()
```

[183]: 5.045

Visualization

[184]:
```python
Boxplot_outlier('euribor3m','Euribor3m Distribution')
```

What is nr.employed distribution?

Find the minimum nr.employed

[185]:
```python
data['nr.employed'].min()
```

[185]: 4963.6

Find the maximum nr.employed

[186]:
```python
data['nr.employed'].max()
```

[186]: 5228.1

Visualization

[187]:
```python
Boxplot_outlier('nr.employed','Nr.Employed Distribution')
```

What is y distribution?

calculate the value counts for the y column

[188]:
```python
data.y.value_counts().to_frame()
```

[188]:
```
     count
y
no   36535
yes   4639
```

Visualization

[189]:
```
Pie('y','Traget','Traget Distribution')
```

Observation based on figure the dataset is imbanlanced

Remove Outliers

applying outlier removal techniques using the interquartile range (IQR) method to the specified columns ('age', 'duration', 'campaign', 'cons.conf.idx')

[190]:
```
cols = ['age', 'duration', 'campaign', 'cons.conf.idx']

for col in cols:
    q1 = data[col].quantile(0.25)
    q3 = data[col].quantile(0.75)
    iqr = q3 - q1
    upper = q3 + (1.5 * iqr)
    lower = q1 - (1.5 * iqr)

    data.loc[data[col] > upper, col] = upper
    data.loc[data[col] < lower, col] = lower

    print(f'For {col} :\n', q1, q3, iqr, upper, lower)
```

```
For age :
 32.0 47.0 15.0 69.5 9.5
For duration :
 102.0 319.0 217.0 644.5 -223.5
For campaign :
 1.0 3.0 2.0 6.0 -2.0
For cons.conf.idx :
 -42.7 -36.4 6.300000000000004 -26.949999999999992 -52.150000000000006
```

Observations:

- For the 'age' column, the first quartile (Q1) is approximately 32.0, the third quartile (Q3) is approximately 47.0, and the interquartile range (IQR) is 15.0. The upper bound for outlier detection is 69.5, and the lower bound is 9.5.
- For the 'duration' column, Q1 is approximately 102.0, Q3 is approximately 319.0, and the IQR is 217.0. The upper bound for outlier detection is 644.5, and the lower bound is -223.5.
- For the 'campaign' column, Q1 is 1.0, Q3 is 3.0, and the IQR is 2.0. The upper bound for outlier detection is 6.0, and the lower bound is -2.0.
- For the 'cons.conf.idx' column, Q1 is approximately -42.7, Q3 is approximately -36.4, and the IQR is approximately 6.3. The upper bound for outlier detection is approximately -26.95, and the lower bound is approximately -52.15.

[191]:
```
fig = make_subplots(rows=2, cols=2, subplot_titles=cols)
for i, col in enumerate(cols, start=1):
    q1 = data[col].quantile(0.25)
```

```
    q3 = data[col].quantile(0.75)
    iqr = q3 - q1
    upper = q3 + (1.5 * iqr)
    lower = q1 - (1.5 * iqr)
    data[col][data[col]>upper] = upper
    data[col][data[col]<lower] = lower
    trace = go.Box(y=data[col], name=col)
    fig.add_trace(trace, row=(i - 1) // 2 + 1, col=(i - 1) % 2 + 1)
fig.update_layout(title_text='Box Plot of Columns without Outliers',title_x=0.
 ↪5, title_y=0.95,
                  height=800, width=1000, template='plotly_dark')
fig.show()
```

** #

PreProcessing

Tabel of Contents

[192]:
```
#create new features or transform existing features to improve the performance␣
 ↪of your data science model
#data['duration']=data['duration']/60
```

[193]:
```
ct = ColumnTransformer(transformers=[('encoder',␣
 ↪OneHotEncoder(),['education'])])
data_ = ct.fit_transform(data[['education']])
```

[194]:
```
pd.DataFrame(data_.toarray(),columns=data['education'].unique())
```

[194]:
```
       basic.4y  high.school  basic.6y  basic.9y  professional.course  \
0           1.0          0.0       0.0       0.0                  0.0
1           0.0          0.0       0.0       1.0                  0.0
2           0.0          0.0       0.0       1.0                  0.0
3           0.0          1.0       0.0       0.0                  0.0
4           0.0          0.0       0.0       1.0                  0.0
…          …           …         …         …                    …
41169       0.0          0.0       0.0       0.0                  0.0
41170       0.0          0.0       0.0       0.0                  0.0
41171       0.0          0.0       0.0       0.0                  0.0
41172       0.0          0.0       0.0       0.0                  0.0
41173       0.0          0.0       0.0       0.0                  0.0

       university.degree  illiterate
0                    0.0         0.0
1                    0.0         0.0
2                    0.0         0.0
3                    0.0         0.0
4                    0.0         0.0
```

```
…                      …         …
41169                  1.0       0.0
41170                  1.0       0.0
41171                  0.0       1.0
41172                  1.0       0.0
41173                  1.0       0.0

[41174 rows x 7 columns]
```

Transform Object Columns

[195]:
```python
data2=data.copy()
object=data2.select_dtypes(include='object').columns
label=LabelEncoder()
for col in object:
    data2[col] = label.fit_transform(data2[col])
data2.head()
```

[195]:
```
   age  job  marital  education  default  housing  loan  contact  month  \
0   39    3        1          0        0        0     0        1      6
1   40    7        1          3        0        0     0        1      6
2   20    7        1          3        0        1     0        1      6
3   23    0        1          1        0        0     0        1      6
4   39    7        1          3        0        0     1        1      6

   day_of_week  …  campaign  pdays  previous  poutcome  emp.var.rate  \
0            1  …         0     26         0         1             8
1            1  …         0     26         0         1             8
2            1  …         0     26         0         1             8
3            1  …         0     26         0         1             8
4            1  …         0     26         0         1             8

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
0              18             16        287            8  0
1              18             16        287            8  0
2              18             16        287            8  0
3              18             16        287            8  0
4              18             16        287            8  0

[5 rows x 21 columns]
```

Show Correlation

[196]:
```python
data2.corr()
```

[196]:
```
                     age       job   marital  education   default   housing  \
age             1.000000 -0.014713 -0.397301  -0.124721  0.002010 -0.002133
job            -0.014713  1.000000  0.025377   0.131910  0.013701  0.007435
```

```
marital        -0.397301  0.025377  1.000000   0.111375 -0.002388  0.011345
education      -0.124721  0.131910  0.111375   1.000000  0.002577  0.016452
default         0.002010  0.013701 -0.002388   0.002577  1.000000 -0.003680
housing        -0.002133  0.007435  0.011345   0.016452 -0.003680  1.000000
loan           -0.007670 -0.011802  0.006495   0.009289 -0.003610  0.036398
contact         0.011662 -0.031847 -0.054634  -0.110425 -0.006476 -0.077803
month          -0.027123 -0.033017 -0.008822  -0.084502 -0.004530 -0.016868
day_of_week    -0.019192 -0.004149  0.002440  -0.016863  0.006079  0.003329
duration        0.002060 -0.002335  0.007733  -0.019020 -0.006338 -0.010737
campaign        0.003200 -0.007181 -0.011543   0.002299 -0.005187 -0.011019
pdays          -0.029975 -0.024770 -0.035479  -0.045991  0.001638 -0.011442
previous        0.016304  0.022185  0.037718   0.037724  0.002765  0.021653
poutcome        0.018395  0.006647  0.002458   0.016768 -0.006195 -0.012577
emp.var.rate    0.013960 -0.007612 -0.081283  -0.028145  0.005324 -0.055101
cons.price.idx -0.000150 -0.022616 -0.055310  -0.085634 -0.002861 -0.075450
cons.conf.idx   0.124852  0.048777 -0.028161   0.084596  0.004757 -0.026991
euribor3m      -0.032588 -0.027161 -0.078541  -0.056048  0.004853 -0.040914
nr.employed    -0.011279 -0.022999 -0.079942  -0.034934  0.006332 -0.036220
y               0.021529  0.025596  0.045892   0.057237 -0.003042  0.011144

                    loan   contact     month  day_of_week  …  campaign  \
age            -0.007670  0.011662 -0.027123    -0.019192  …  0.003200
job            -0.011802 -0.031847 -0.033017    -0.004149  … -0.007181
marital         0.006495 -0.054634 -0.008822     0.002440  … -0.011543
education       0.009289 -0.110425 -0.084502    -0.016863  …  0.002299
default        -0.003610 -0.006476 -0.004530     0.006079  … -0.005187
housing         0.036398 -0.077803 -0.016868     0.003329  … -0.011019
loan            1.000000 -0.013393 -0.007111    -0.009492  …  0.012112
contact        -0.013393  1.000000  0.276465    -0.009591  …  0.071659
month          -0.007111  0.276465  1.000000     0.027697  … -0.063819
day_of_week    -0.009492 -0.009591  0.027697     1.000000  … -0.051029
duration       -0.006608 -0.036197  0.008218     0.031255  … -0.080191
campaign        0.012112  0.071659 -0.063819    -0.051029  …  1.000000
pdays          -0.001016  0.116138 -0.047412    -0.010465  …  0.059798
previous       -0.002194 -0.212905  0.103149    -0.004109  … -0.083856
poutcome       -0.000209  0.118773 -0.065009     0.018737  …  0.030048
emp.var.rate    0.000827  0.350374 -0.188202     0.035965  …  0.142389
cons.price.idx -0.005576  0.584651 -0.006331     0.002217  …  0.112596
cons.conf.idx  -0.013157  0.243189 -0.018811     0.035204  … -0.024704
euribor3m       0.005097  0.274110 -0.197034     0.023543  …  0.134282
nr.employed     0.006289  0.176080 -0.266913     0.023306  …  0.142881
y              -0.004486 -0.144774 -0.006057     0.015964  … -0.069413

                   pdays  previous  poutcome  emp.var.rate  cons.price.idx  \
age            -0.029975  0.016304  0.018395      0.013960       -0.000150
job            -0.024770  0.022185  0.006647     -0.007612       -0.022616
marital        -0.035479  0.037718  0.002458     -0.081283       -0.055310
```

```
education       -0.045991  0.037724  0.016768     -0.028145        -0.085634
default          0.001638  0.002765 -0.006195      0.005324        -0.002861
housing         -0.011442  0.021653 -0.012577     -0.055101        -0.075450
loan            -0.001016 -0.002194 -0.000209      0.000827        -0.005576
contact          0.116138 -0.212905  0.118773      0.350374         0.584651
month           -0.047412  0.103149 -0.065009     -0.188202        -0.006331
day_of_week     -0.010465 -0.004109  0.018737      0.035965         0.002217
duration        -0.062397  0.037364  0.038345     -0.048517        -0.000610
campaign         0.059798 -0.083856  0.030048      0.142389         0.112596
pdays            1.000000 -0.579460 -0.486940      0.257257         0.090841
previous        -0.579460  1.000000 -0.313096     -0.405913        -0.197490
poutcome        -0.486940 -0.313096  1.000000      0.192381         0.198958
emp.var.rate     0.257257 -0.405913  0.192381      1.000000         0.750857
cons.price.idx   0.090841 -0.197490  0.198958      0.750857         1.000000
cons.conf.idx   -0.108991 -0.020104  0.166272      0.122006        -0.024101
euribor3m        0.384726 -0.489973  0.089883      0.868708         0.546774
nr.employed      0.375595 -0.499543  0.087034      0.845379         0.409424
y               -0.320975  0.230197  0.129814     -0.286795        -0.140511

                 cons.conf.idx  euribor3m  nr.employed         y
age                   0.124852  -0.032588    -0.011279  0.021529
job                   0.048777  -0.027161    -0.022999  0.025596
marital              -0.028161  -0.078541    -0.079942  0.045892
education             0.084596  -0.056048    -0.034934  0.057237
default               0.004757   0.004853     0.006332 -0.003042
housing              -0.026991  -0.040914    -0.036220  0.011144
loan                 -0.013157   0.005097     0.006289 -0.004486
contact               0.243189   0.274110     0.176080 -0.144774
month                -0.018811  -0.197034    -0.266913 -0.006057
day_of_week           0.035204   0.023543     0.023306  0.015964
duration             -0.004162  -0.062160    -0.074227  0.401301
campaign             -0.024704   0.134282     0.142881 -0.069413
pdays                -0.108991   0.384726     0.375595 -0.320975
previous             -0.020104  -0.489973    -0.499543  0.230197
poutcome              0.166272   0.089883     0.087034  0.129814
emp.var.rate          0.122006   0.868708     0.845379 -0.286795
cons.price.idx       -0.024101   0.546774     0.409424 -0.140511
cons.conf.idx         1.000000  -0.123080    -0.064467  0.069911
euribor3m            -0.123080   1.000000     0.912388 -0.368182
nr.employed          -0.064467   0.912388     1.000000 -0.355120
y                     0.069911  -0.368182    -0.355120  1.000000

[21 rows x 21 columns]
```

[197]:

```
corr = data2.corr()
corr=corr.round(2)
fig = ff.create_annotated_heatmap(z=corr.values,
```

```python
                                    x=corr.columns.tolist(),
                                    y=corr.columns.tolist(),
                                    colorscale='RdBu',
                                    hoverinfo='none',
                                    showscale=True,
                                    ygap=1,
                                    xgap=1
                                   )
fig.update_xaxes(side='bottom')
fig.update_layout(
    title_text='Heatmap',
    title_x=0.5,
    width=1000,
    height=1000,
    xaxis=dict(showgrid=True),
    yaxis=dict(showgrid=True, autorange='reversed'),
    template='plotly_dark'
)
fig.show()
```

[198]:
```python
mask = np.triu(np.ones_like(corr, dtype=bool))
df_mask = corr.mask(mask)
df_mask_rounded = df_mask.round(2)
fig = ff.create_annotated_heatmap(z=df_mask_rounded.values,
                                  x=df_mask_rounded.columns.tolist(),
                                  y=df_mask_rounded.columns.tolist(),
                                  colorscale='RdBu',
                                  hoverinfo='none',
                                  showscale=True,
                                  ygap=1,
                                  xgap=1
                                 )
fig.update_xaxes(side='bottom')
fig.update_layout(
    title_text='Heatmap',
    title_x=0.5,
    width=1000,
    height=1000,
    xaxis=dict(showgrid=True),
    yaxis=dict(showgrid=True, autorange='reversed'),
    template='plotly_dark'
)
for annotation in fig.layout.annotations:
    if annotation.text == 'nan':
        annotation.text = ""

fig.show()
```

Classification

[199]:
```
X_classification = data2.iloc[:,:-1]
y_classification = data2.iloc[:,-1]
key = X_classification.keys()
X_classification.head()
```

[199]:
```
   age  job  marital  education  default  housing  loan  contact  month  \
0   39    3        1          0        0        0     0        1      6
1   40    7        1          3        0        0     0        1      6
2   20    7        1          3        0        1     0        1      6
3   23    0        1          1        0        0     0        1      6
4   39    7        1          3        0        0     1        1      6

   day_of_week  duration  campaign  pdays  previous  poutcome  emp.var.rate  \
0            1     261.0         0     26         0         1             8
1            1     149.0         0     26         0         1             8
2            1     226.0         0     26         0         1             8
3            1     151.0         0     26         0         1             8
4            1     307.0         0     26         0         1             8

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed
0              18             16        287            8
1              18             16        287            8
2              18             16        287            8
3              18             16        287            8
4              18             16        287            8
```

[200]:
```
y_classification.head()
```

[200]:
```
0    0
1    0
2    0
3    0
4    0
Name: y, dtype: int64
```

Clustering

[201]:
```
X_cluster = data2.copy()
X_cluster.head()
```

[201]:
```
   age  job  marital  education  default  housing  loan  contact  month  \
0   39    3        1          0        0        0     0        1      6
1   40    7        1          3        0        0     0        1      6
2   20    7        1          3        0        1     0        1      6
3   23    0        1          1        0        0     0        1      6
4   39    7        1          3        0        0     1        1      6
```

```
   day_of_week  …  campaign  pdays  previous  poutcome  emp.var.rate  \
0            1  …         0     26         0         1             8
1            1  …         0     26         0         1             8
2            1  …         0     26         0         1             8
3            1  …         0     26         0         1             8
4            1  …         0     26         0         1             8

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
0              18             16        287            8  0
1              18             16        287            8  0
2              18             16        287            8  0
3              18             16        287            8  0
4              18             16        287            8  0

[5 rows x 21 columns]
```

Regression

[202]:
```
X_regression = data2.drop('duration',axis=1)
y_regression = data2['duration']
key = X_regression.keys()
X_regression.head()
```

[202]:
```
   age  job  marital  education  default  housing  loan  contact  month  \
0   39    3        1          0        0        0     0        1      6
1   40    7        1          3        0        0     0        1      6
2   20    7        1          3        0        1     0        1      6
3   23    0        1          1        0        0     0        1      6
4   39    7        1          3        0        0     1        1      6

   day_of_week  campaign  pdays  previous  poutcome  emp.var.rate  \
0            1         0     26         0         1             8
1            1         0     26         0         1             8
2            1         0     26         0         1             8
3            1         0     26         0         1             8
4            1         0     26         0         1             8

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
0              18             16        287            8  0
1              18             16        287            8  0
2              18             16        287            8  0
3              18             16        287            8  0
4              18             16        287            8  0
```

[203]:
```
y_regression=y_regression/y_regression.max()
y_regression.head()
```

[203]:
```
0    0.404965
1    0.231187
2    0.350659
3    0.234290
4    0.476338
Name: duration, dtype: float64
```

Banlanced Data

[204]:
```
over = RandomOverSampler(sampling_strategy='minority')
X_classification_over,y_classification_over=over.
 ↪fit_resample(X_classification,y_classification)
```

[205]:
```
under = RandomUnderSampler()
X_classification_under,y_classification_under=under.
 ↪fit_resample(X_classification,y_classification)
```

** #

ML Models

Tabel of Contents

Classification Models

`RandomForestClassifier`

[206]:
```
def Split(X, y='', classification=1):
    if classification == 1:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪1, random_state=44, shuffle=True, stratify=y)
    elif classification == 2:
        X_train, X_test = train_test_split(X, test_size=0.1, random_state=44,␣
 ↪shuffle=True)
        print('X_train shape is ', X_train.shape)
        print('X_test shape is ', X_test.shape)
        return X_train, X_test
    else:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪1, random_state=44, shuffle=True)
    print('X_train shape is ', X_train.shape)
    print('X_test shape is ', X_test.shape)
    print('y_train shape is ', y_train.shape)
    print('y_test shape is ', y_test.shape)
    return X_train, y_train, X_test, y_test

def SelectFeature(model, X_train, y_train):
    FeatureSelection = SelectFromModel(estimator=model)
    FeatureSelection.fit(X_train, y_train)
    return X_train.iloc[:, FeatureSelection.get_support()].columns
```

```python
def Search(model, parameters, X_train, y_train):
    GridSearchModel = GridSearchCV(model, parameters, cv=5,␣
 ↪return_train_score=True)
    GridSearchModel.fit(X_train, y_train)
    return GridSearchModel.best_estimator_

def cross_validation(model, X_train, y_train):
    CrossValidateValues1 = cross_validate(model, X_train, y_train, cv=5,␣
 ↪return_train_score=True)
    print('Train Score Value : ', CrossValidateValues1['train_score'], "\t␣
 ↪Mean", CrossValidateValues1['train_score'].mean())
    print('Test Score Value : ', CrossValidateValues1['test_score'], "\t Mean",␣
 ↪CrossValidateValues1['test_score'].mean())

def PipeLine(model, X_train, y_train, flage=0):
    if flage == 0:
        steps = [('model', model)]
    elif flage == 1:
        steps = [('scaling', MinMaxScaler()), ('model', model)]
    elif flage == 2:
        steps = [('scaling', Normalizer()), ('model', model)]
    elif flage == 3:
        steps = [('pca', PCA()), ('model', model)]
    elif flage == 4:
        steps = [('scaling', MinMaxScaler()), ('pca', PCA()), ('model', model)]
    else:
        steps = [('scaling', Normalizer()), ('pca', PCA()), ('model', model)]
    return Pipeline(steps).fit(X_train, y_train)

def Area(fprValue2, tprValue2, AUCValue):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=fprValue2, y=tprValue2,
                    mode='lines',
                    name='ROC curve (AUC = {:.2f})'.format(AUCValue),␣
 ↪line=dict(color='red')))
    fig.add_shape(type='line',
        x0=0, y0=0, x1=1, y1=1,
        line=dict(color='orange', width=2, dash='dash'),
        name='Random Guessing')
    fig.update_layout(
                    title='Receiver Operating Characteristic (ROC) Curve',
                    title_x=.5,
                    xaxis_title='False Positive Rate',
                    yaxis_title='True Positive Rate',
                    xaxis=dict(range=[0, 1], constrain='domain'),
                    yaxis=dict(range=[0, 1]),
```

```python
                        legend=dict(x=0.01, y=0.99),
                        showlegend=True,
                        template='plotly_dark'
                       )
    fig.update_annotations(font=dict(size=20))
    fig.show()

def Check(model='', X_train='', y_train='', X_test='', y_test='',␣
↪cluster=0,y_train2='',y_train_pred='',y_test2='',y_pred=''):
    if cluster:
        train = accuracy_score(y_train2, y_train_pred)
        test = accuracy_score(y_test2, y_pred)
        y_pred = y_pred
        y_test = y_test2

    else:
        y_pred = model.predict(X_test)
        train = accuracy_score(y_train, model.predict(X_train))
        test = accuracy_score(y_test, y_pred)
    print('Model Train Score is : ', train)
    print('Model Test Score is : ', test)
    F1Score = f1_score(y_test, y_pred)
    print('F1 Score is : ', F1Score)
    RecallScore = recall_score(y_test, y_pred)
    print('Recall Score is : ', RecallScore)
    PrecisionScore = precision_score(y_test, y_pred)
    print('Precision Score is : ', PrecisionScore)
    fprValue2, tprValue2, thresholdsValue2 = roc_curve(y_test, y_pred)
    AUCValue = auc(fprValue2, tprValue2)
    print('AUC Value  : ', AUCValue)
    Area(fprValue2, tprValue2, AUCValue)
    ClassificationReport = classification_report(y_test, y_pred)
    print('Classification Report is : ', ClassificationReport)
    CM = confusion_matrix(y_test, y_pred)
    print('Confusion Matrix is : \n', CM)
    disp = ConfusionMatrixDisplay(confusion_matrix=CM, display_labels=[0, 1])
    disp.plot(cmap='Blues')
    values = [train, test, F1Score, RecallScore, PrecisionScore, AUCValue]
    return values

def Models(models, X_train, y_train, X_test, y_test):
    print('Apply Model With Normal Data : \n')
    model = PipeLine(models, X_train, y_train)
    value1 =␣
↪Check(model=model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test)
    print("\n\n Apply Model With Feature Selection :\n")
    try:
```

```python
        feature = SelectFeature(model, X_train, y_train)
    except:
        feature = SelectFeature(RandomForestClassifier(max_depth=20), X_train,␣
↪y_train)
    X_train1 = X_train.loc[:, feature]
    X_test1 = X_test.loc[:, feature]
    model = PipeLine(models, X_train1, y_train, flage=1)
    value2 =␣
↪Check(model=model,X_train=X_train1,y_train=y_train,X_test=X_test1,y_test=y_test)
    print("\n\n Apply Model With Normal Data With Scaling :\n")
    model = PipeLine(models, X_train, y_train, flage=1)
    value3 =␣
↪Check(model=model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test)
    print("\n\n Apply Model With Normal Data With Normalize :\n")
    model = PipeLine(models, X_train, y_train, flage=2)
    value4 =␣
↪Check(model=model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test)
    print("\n\n Apply Model With Normal Data With PCA :\n")
    model = PipeLine(models, X_train, y_train, flage=3)
    value5 =␣
↪Check(model=model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test)
    print("\n\n Apply Model With Normal Data With PCA and Scaling :\n")
    model = PipeLine(models, X_train, y_train, flage=4)
    value6 =␣
↪Check(model=model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test)
    print("\n\n Apply Model With Normal Data With PCA and Normalize :\n")
    model = PipeLine(models, X_train, y_train, flage=5)
    value7 =␣
↪Check(model=model,X_train=X_train,y_train=y_train,X_test=X_test,y_test=y_test)
    return [value1, value2, value3, value4, value5, value6, value7]
def models_draw(df):
    figure = go.Figure()
    for column in df.columns:
        trace = go.Bar(
            x=df.index,
            y=df[column],
            name=column,
            text=df[column].values.round(2),
            textposition='inside'
        )
        figure.add_trace(trace)
    figure.update_layout(
        barmode='group',
        title='Performance Metrics Comparison',
        title_x=.5,
        xaxis=dict(title='Models'),
```

```
        yaxis=dict(title='Score'),
        template='plotly_dark',
        width=1100,
        height=700
    )
    figure.show()
```

[207]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[208]:
```
Search(RandomForestClassifier(max_depth=20),{'max_depth':
↪[5,10,15,20,25,30,35,40]},X_train,y_train)
```

[208]:
```
RandomForestClassifier(max_depth=10)
```

[209]:
```
cross_validation(RandomForestClassifier(max_depth=10),X_train,y_train)
```

```
Train Score Value :  [0.93877344 0.94025974 0.93735875 0.93867431 0.93860685]
Mean 0.9387346181375941
Test Score Value :  [0.91351862 0.91256241 0.91188773 0.91013359 0.91822966]
Mean 0.913266400792917
```

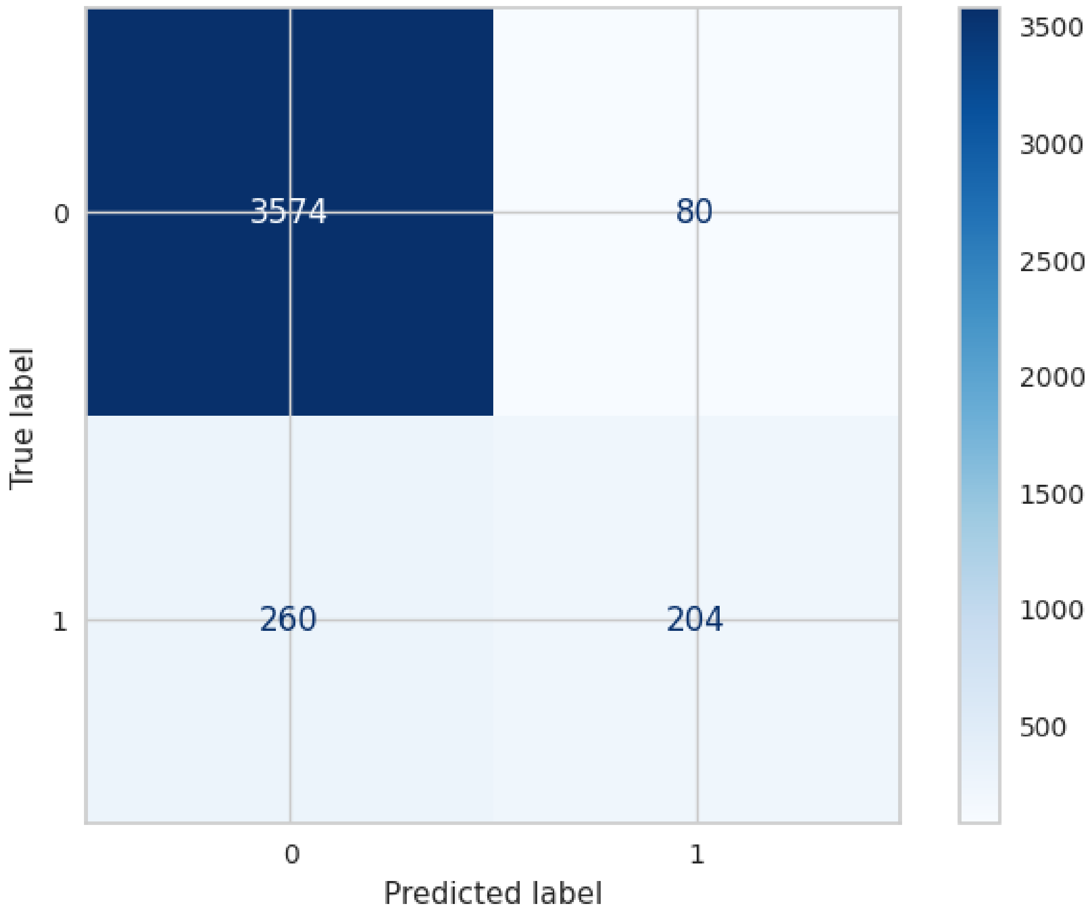
[210]:
```
Values =␣
↪Models(RandomForestClassifier(max_depth=10),X_train,y_train,X_test,y_test)
```

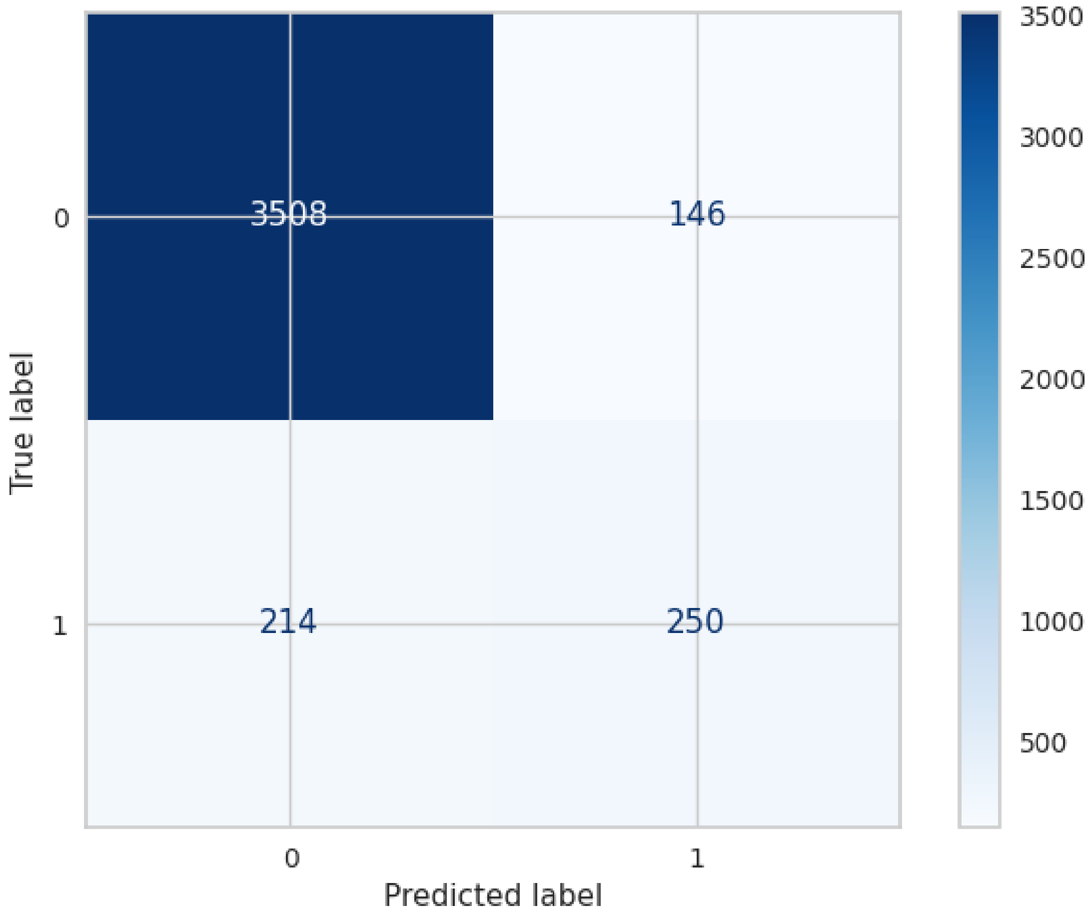
```
Apply Model With Normal Data :

Model Train Score is :  0.9355839810017271
Model Test Score is :  0.9174356483729966
F1 Score is :  0.5454545454545455
Recall Score is :  0.4396551724137931
Precision Score is :  0.7183098591549296
AUC Value  :  0.708880678708265

Classification Report is :                 precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.72      0.44      0.55       464

    accuracy                           0.92      4118
   macro avg       0.83      0.71      0.75      4118
weighted avg       0.91      0.92      0.91      4118


Confusion Matrix is :
```

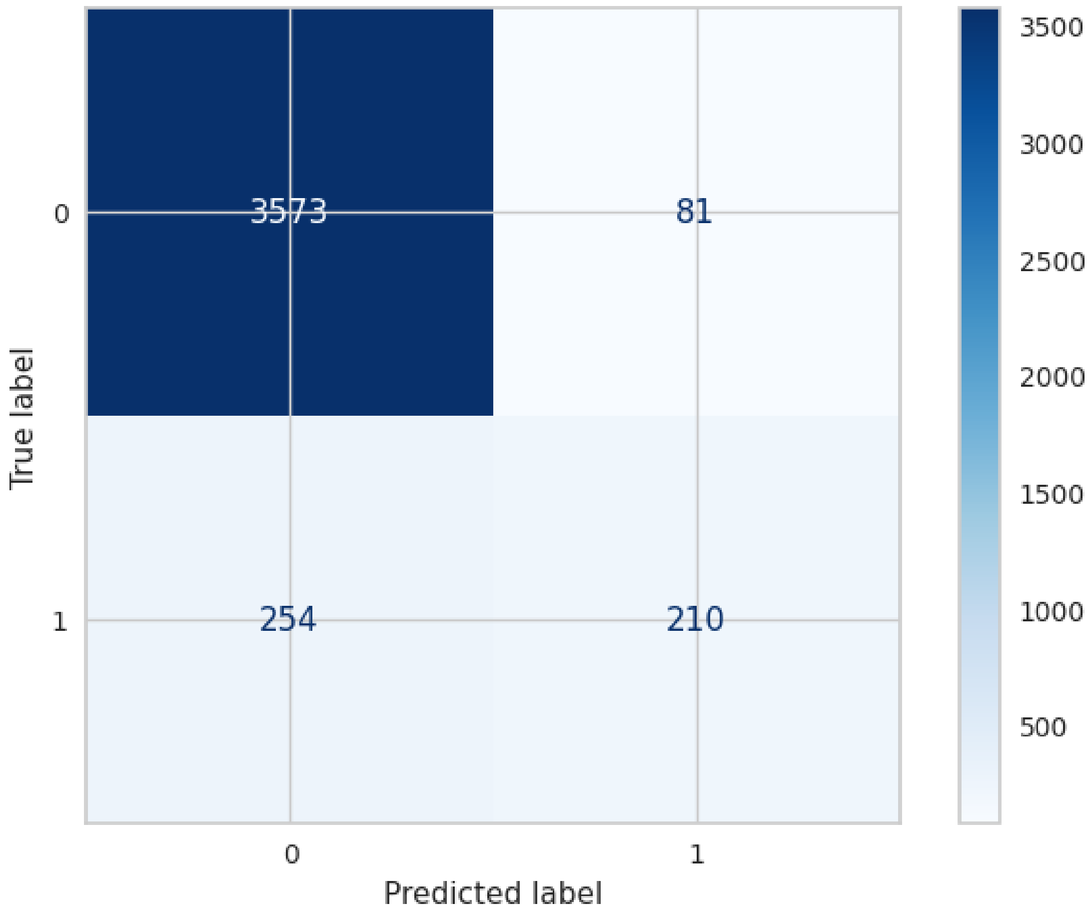
```
 [[3574   80]
 [ 260  204]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9357458981001727
Model Test Score is :  0.9125789218067023
F1 Score is :  0.5813953488372092
Recall Score is :  0.5387931034482759
Precision Score is :  0.6313131313131313
AUC Value  :  0.7494184455391352
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.96      0.95      3654
           1       0.63      0.54      0.58       464

    accuracy                           0.91      4118
   macro avg       0.79      0.75      0.77      4118
weighted avg       0.91      0.91      0.91      4118

Confusion Matrix is :
 [[3508  146]
 [ 214  250]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9361776770293609
Model Test Score is :  0.9186498300145702
F1 Score is :  0.5562913907284768
Recall Score is :  0.4525862068965517
Precision Score is :  0.7216494845360825
AUC Value  :  0.7152093596059113
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.96      3654
           1       0.72      0.45      0.56       464

    accuracy                           0.92      4118
   macro avg       0.83      0.72      0.76      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3573   81]
```

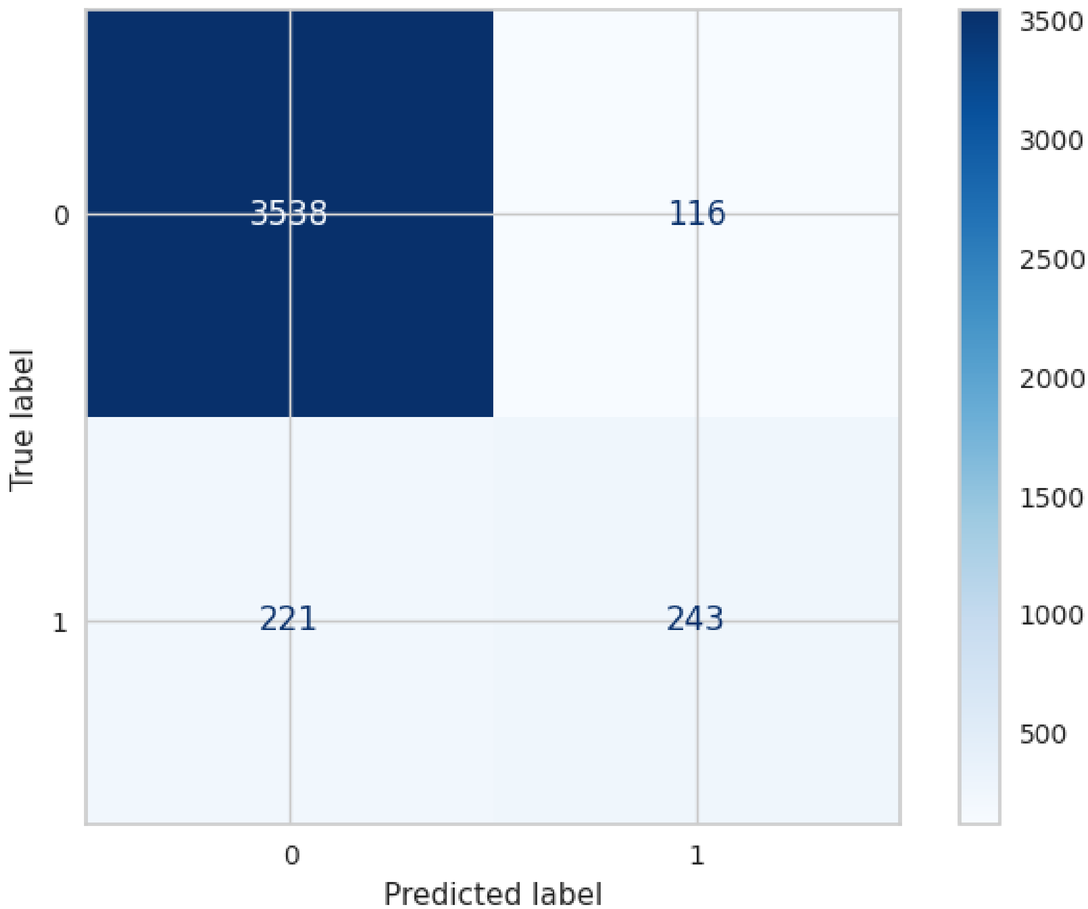
```
 [ 254  210]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9417368307426598
Model Test Score is :  0.9181641573579408
F1 Score is :  0.5905224787363303
Recall Score is :  0.5237068965517241
Precision Score is :  0.6768802228412256
AUC Value  :  0.7459804324028462
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.97      0.95      3654
           1       0.68      0.52      0.59       464

    accuracy                           0.92      4118
   macro avg       0.81      0.75      0.77      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3538  116]
 [ 221  243]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9455958549222798
Model Test Score is :  0.9140359397765906
F1 Score is :  0.5267379679144385
Recall Score is :  0.4245689655172414
Precision Score is :  0.6936619718309859
AUC Value  :  0.7003797208538588
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.69      0.42      0.53       464

    accuracy                           0.91      4118
   macro avg       0.81      0.70      0.74      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3567   87]
 [ 267  197]]
```

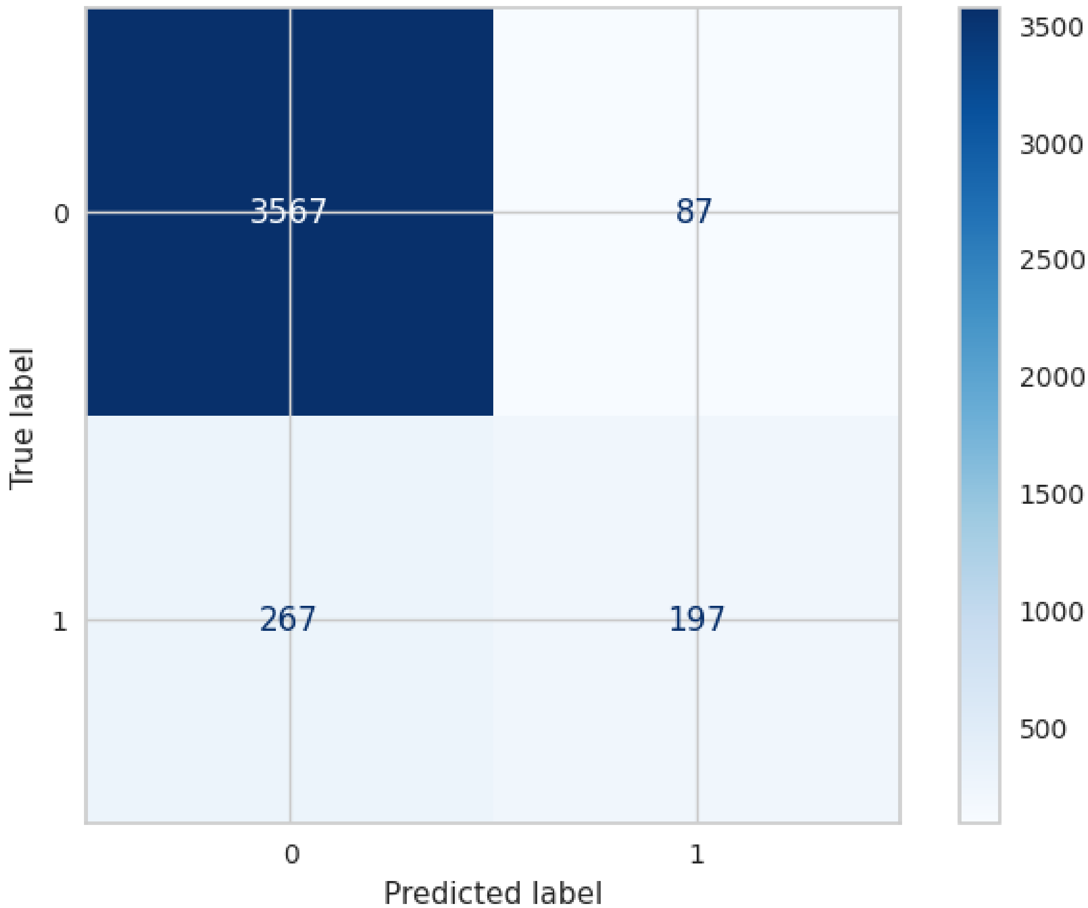
```
 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9403875215889465
Model Test Score is :  0.9169499757163672
F1 Score is :  0.5196629213483146
Recall Score is :  0.39870689655172414
Precision Score is :  0.7459677419354839
AUC Value  :  0.6907327586206897

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.75      0.40      0.52       464

    accuracy                           0.92      4118
   macro avg       0.84      0.69      0.74      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3591   63]
 [ 279  185]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9353411053540587
Model Test Score is :  0.9133074307916464
F1 Score is :  0.5233644859813085
Recall Score is :  0.4224137931034483
Precision Score is :  0.6877192982456141
AUC Value  :  0.6990284619594965

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.69      0.42      0.52       464

    accuracy                           0.91      4118
   macro avg       0.81      0.70      0.74      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3565   89]
 [ 268  196]]
```

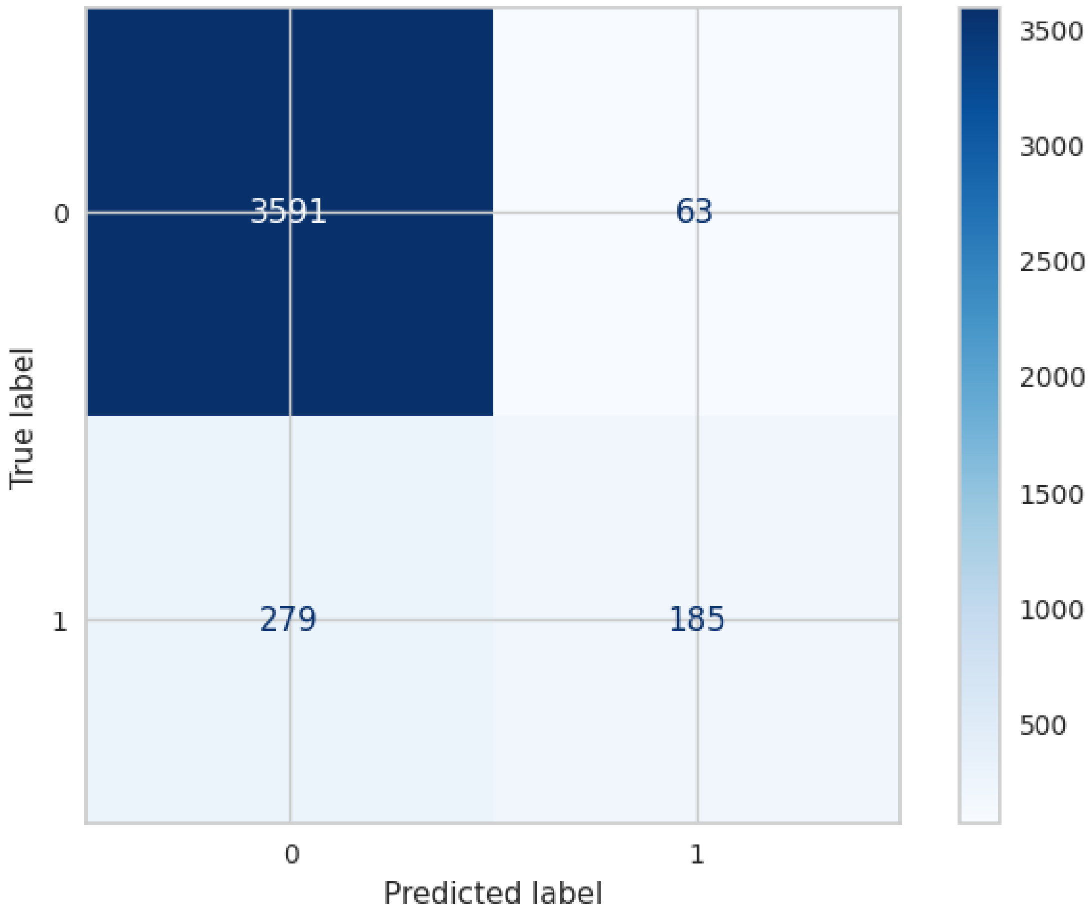

3574
80
260
204
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500

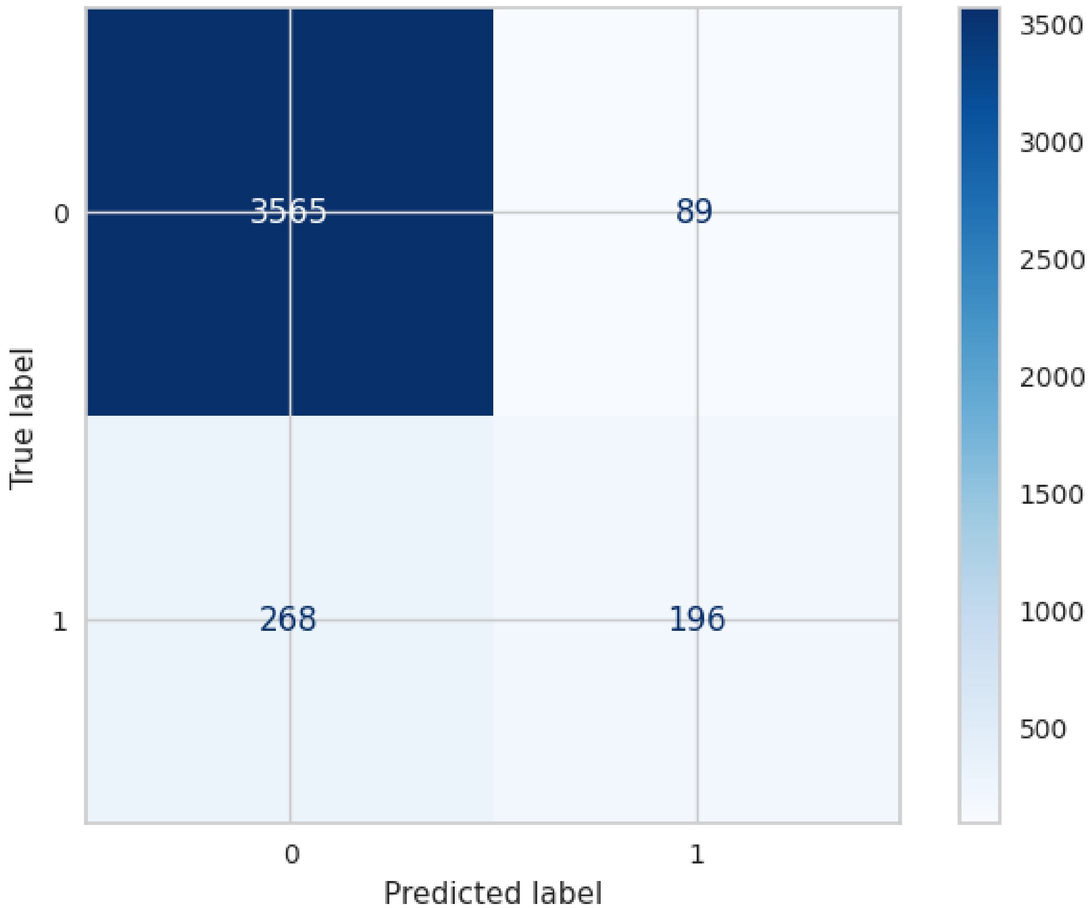

3508
146
214
250
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3573
81
254
210
0
1
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500


3538
116
221
243
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3567
87
267
197
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3591
63
279
185
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[211]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Forest','Forest With Feature','Forest Scaling','Forest With␣
↪Normalize','Forest With PCA','Forest With PCA and Scaling',
                 'Forest With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[211]:
```
                               Train Accuracy  Test Accuracy   Test F1  \
Models
Forest                               0.935584       0.917436  0.545455
Forest With Feature                  0.935746       0.912579  0.581395
Forest Scaling                       0.936178       0.918650  0.556291
Forest With Normalize                0.941737       0.918164  0.590522
Forest With PCA                      0.945596       0.914036  0.526738
Forest With PCA and Scaling          0.940388       0.916950  0.519663
Forest With PCA and Normalize        0.935341       0.913307  0.523364
```

```
                              Test Recall  Test Precision       AUC
Models
Forest                           0.439655        0.718310  0.708881
Forest With Feature              0.538793        0.631313  0.749418
Forest Scaling                   0.452586        0.721649  0.715209
Forest With Normalize            0.523707        0.676880  0.745980
Forest With PCA                  0.424569        0.693662  0.700380
Forest With PCA and Scaling      0.398707        0.745968  0.690733
Forest With PCA and Normalize    0.422414        0.687719  0.699028
```

[212]:
```python
models_draw(df)
```

RandomOverSampler

[213]:
```python
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[214]:
```python
Search(RandomForestClassifier(max_depth=20),{'max_depth':
  ↪[20,25,30,35,40]},X_train,y_train)
```

[214]: RandomForestClassifier(max_depth=30)

[215]:
```python
cross_validation(RandomForestClassifier(max_depth=40),X_train,y_train)
```

```
Train Score Value :  [0.99992397 0.99990496 0.99992397 0.99990496 0.99996199]
Mean 0.9999239693330242
Test Score Value :  [0.962366   0.96358245 0.9628982  0.96616484 0.96175487]
Mean 0.9633532717966704
```

[216]:
```python
Values =␣
  ↪Models(RandomForestClassifier(max_depth=40),X_train,y_train,X_test,y_test)
```

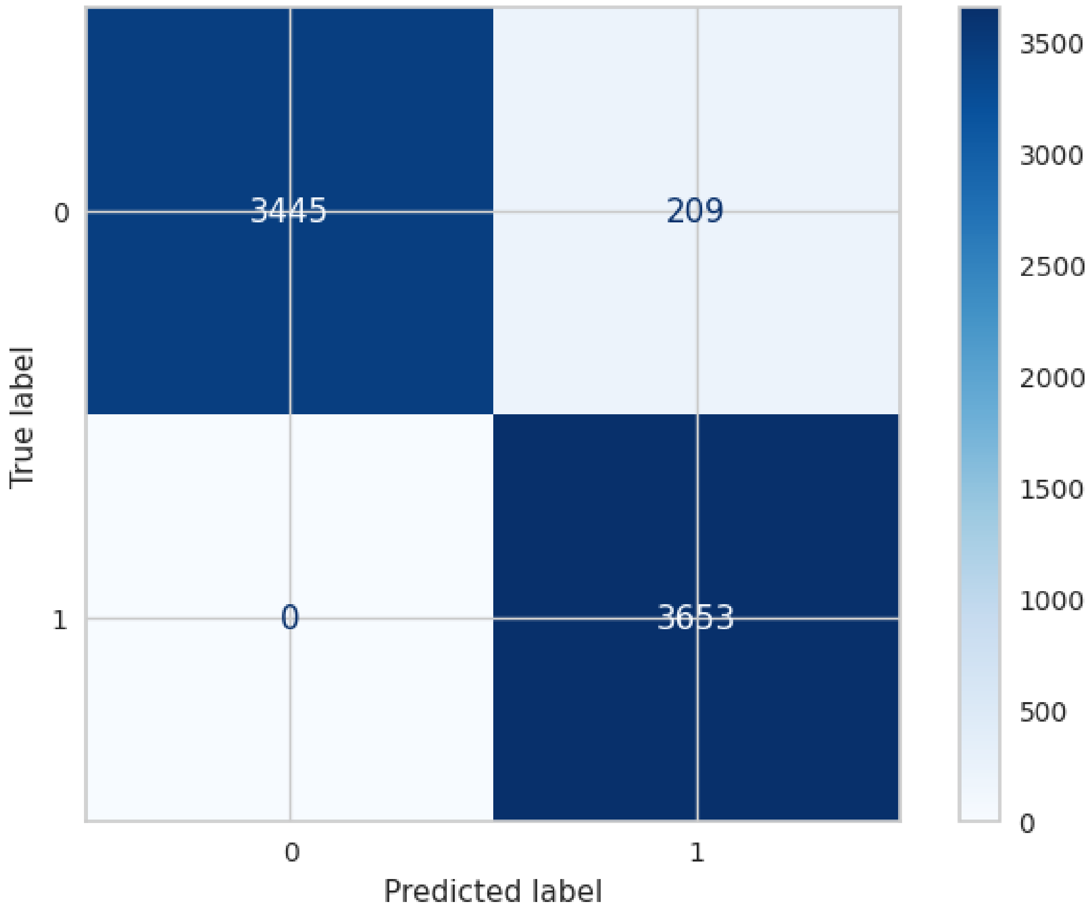
```
Apply Model With Normal Data :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9713972902696045
F1 Score is :  0.9721889554224884
Recall Score is :  1.0
Precision Score is :  0.9458829621957535
AUC Value  :  0.9714012041598248

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.95      1.00      0.97      3653
```

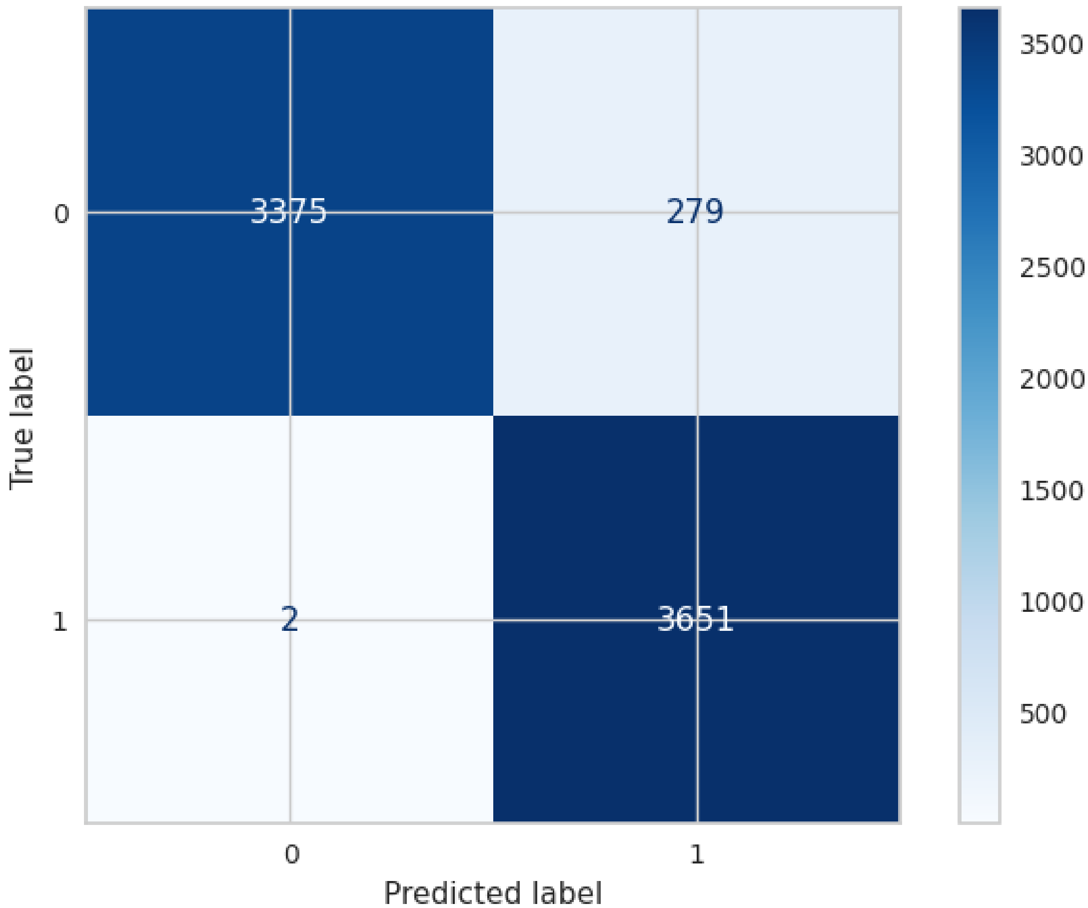
```
    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3445  209]
 [   0 3653]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9883977312470538
Model Test Score is :  0.9615437251950185
F1 Score is :  0.9629434260846631
Recall Score is :  0.9994525047905831
Precision Score is :  0.9290076335877863
AUC Value  :  0.9615489124938138
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.92      0.96      3654
           1       0.93      1.00      0.96      3653

    accuracy                           0.96      7307
   macro avg       0.96      0.96      0.96      7307
weighted avg       0.96      0.96      0.96      7307

Confusion Matrix is :
 [[3375  279]
 [   2 3651]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9707130149172026
F1 Score is :  0.9715425531914893
Recall Score is :  1.0
Precision Score is :  0.9446599431083528
AUC Value  :  0.9707170224411603
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653
```

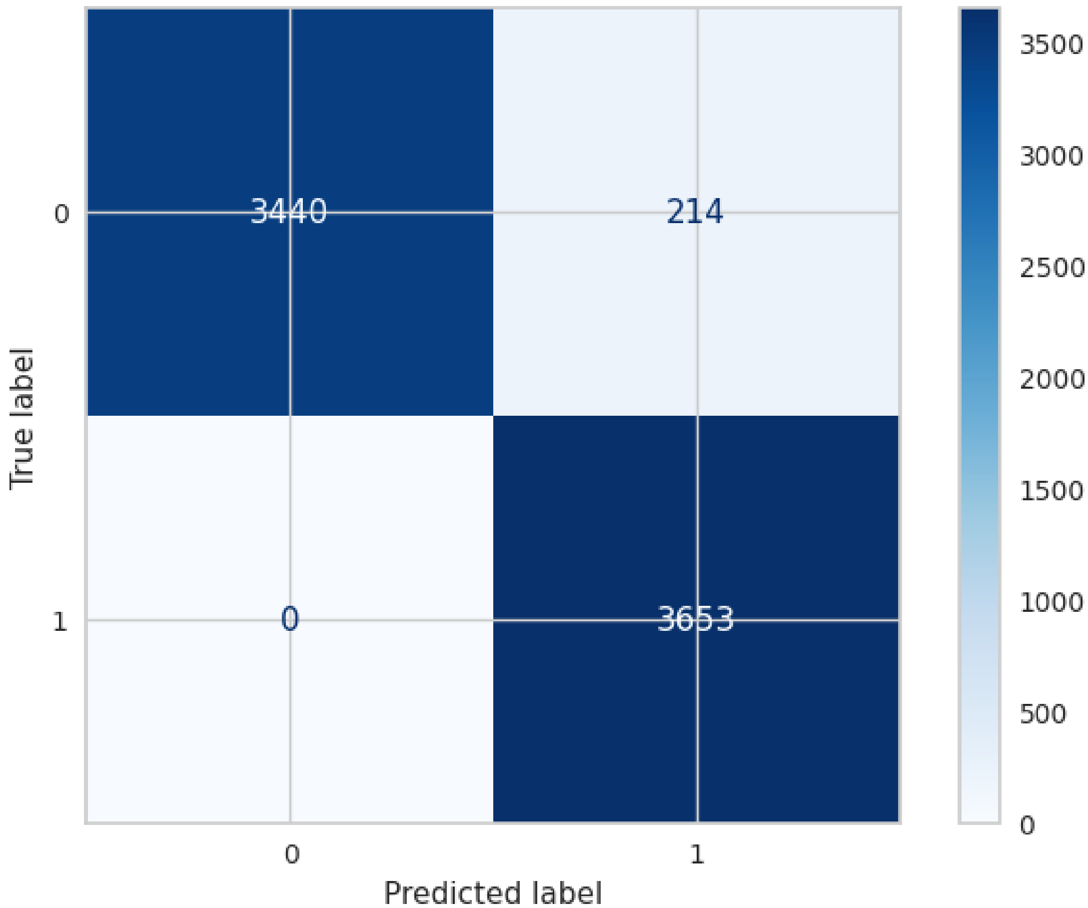
```
    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3440  214]
 [   0 3653]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9705761598467223
F1 Score is :  0.971413375880867
Recall Score is :  1.0
Precision Score is :  0.9444157187176836
AUC Value  :  0.9705801860974275

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653

    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3439  215]
 [   0 3653]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9727658409744081
F1 Score is :  0.9734843437708194
Recall Score is :  1.0
Precision Score is :  0.9483385254413291
AUC Value  :  0.9727695675971538

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.95      0.97      3654
           1       0.95      1.00      0.97      3653

    accuracy                           0.97      7307
```

```
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3455  199]
 [   0 3653]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9745449568906528
F1 Score is :  0.9751735184196476
Recall Score is :  1.0
Precision Score is :  0.9515498827819745
AUC Value  :  0.9745484400656814
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.95      0.97      3654
           1       0.95      1.00      0.98      3653

    accuracy                           0.97      7307
   macro avg       0.98      0.97      0.97      7307
weighted avg       0.98      0.97      0.97      7307

Confusion Matrix is :
 [[3468  186]
 [   0 3653]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9999087632863465
Model Test Score is :  0.9708498699876831
F1 Score is :  0.9716717648623487
Recall Score is :  1.0
Precision Score is :  0.9449042938437662
AUC Value  :  0.9708538587848933
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653

    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
```

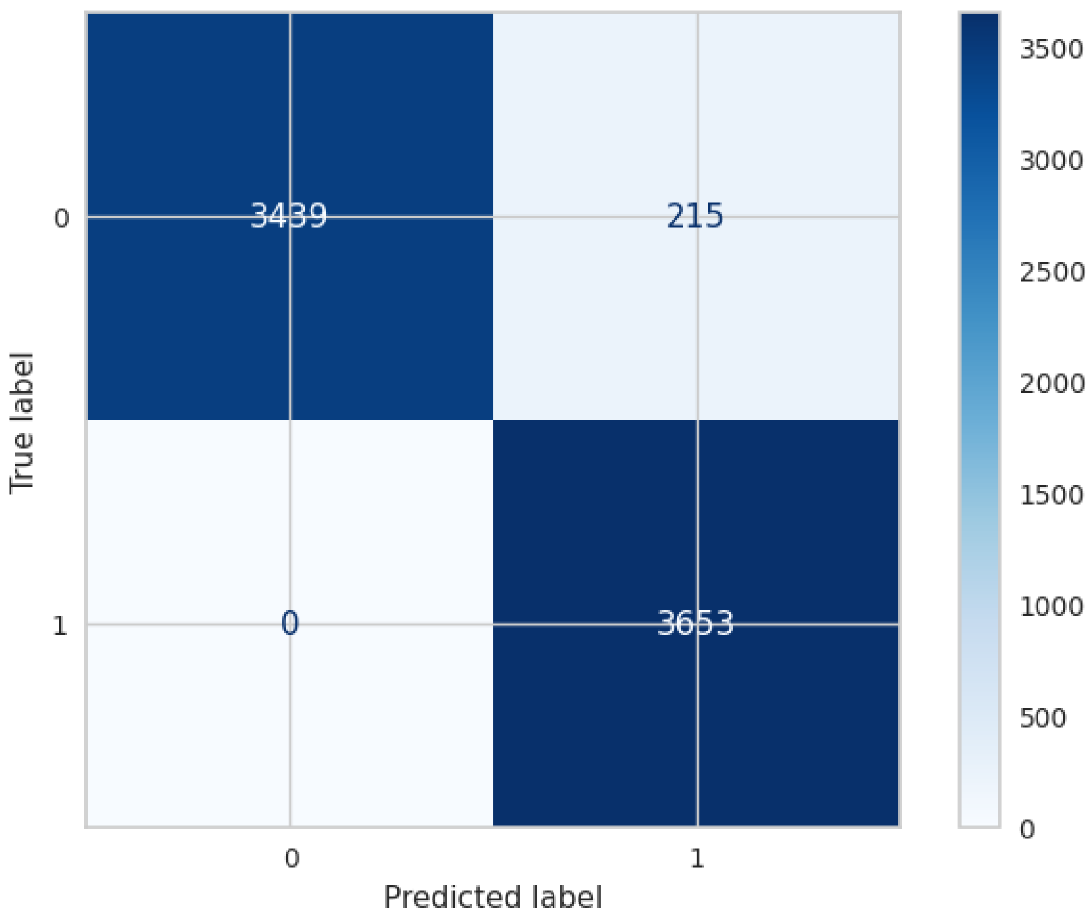
```
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3441  213]
 [   0 3653]]
```

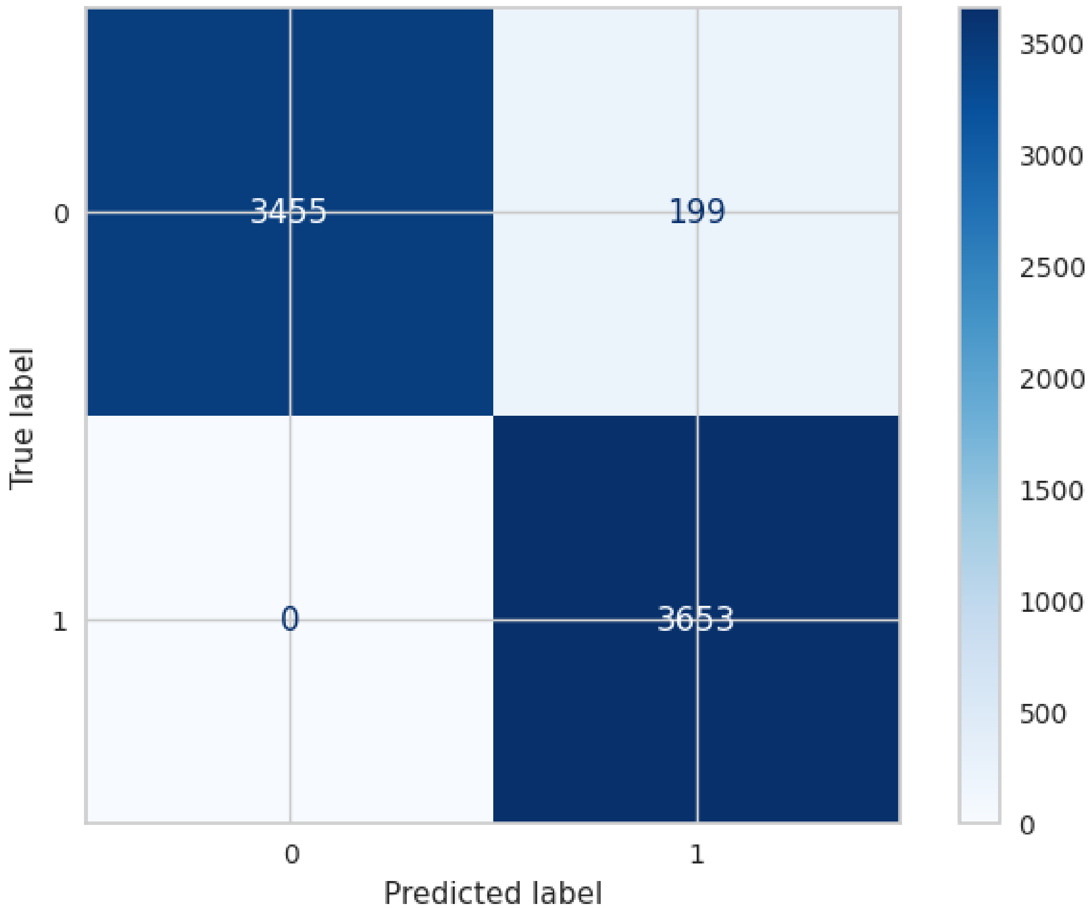



3375
279
2
3651
True label
0
1
Predicted label
0
1
3500
3000
2500
2000
1500
1000
500

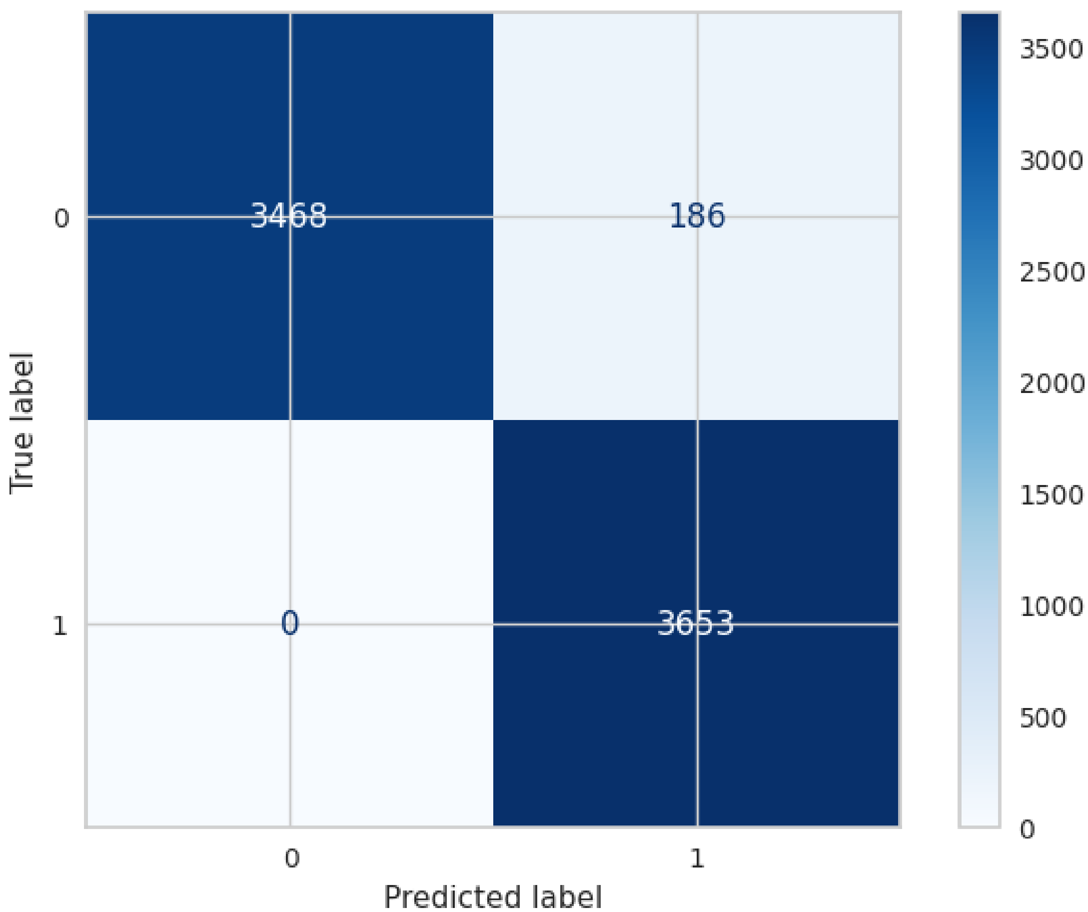


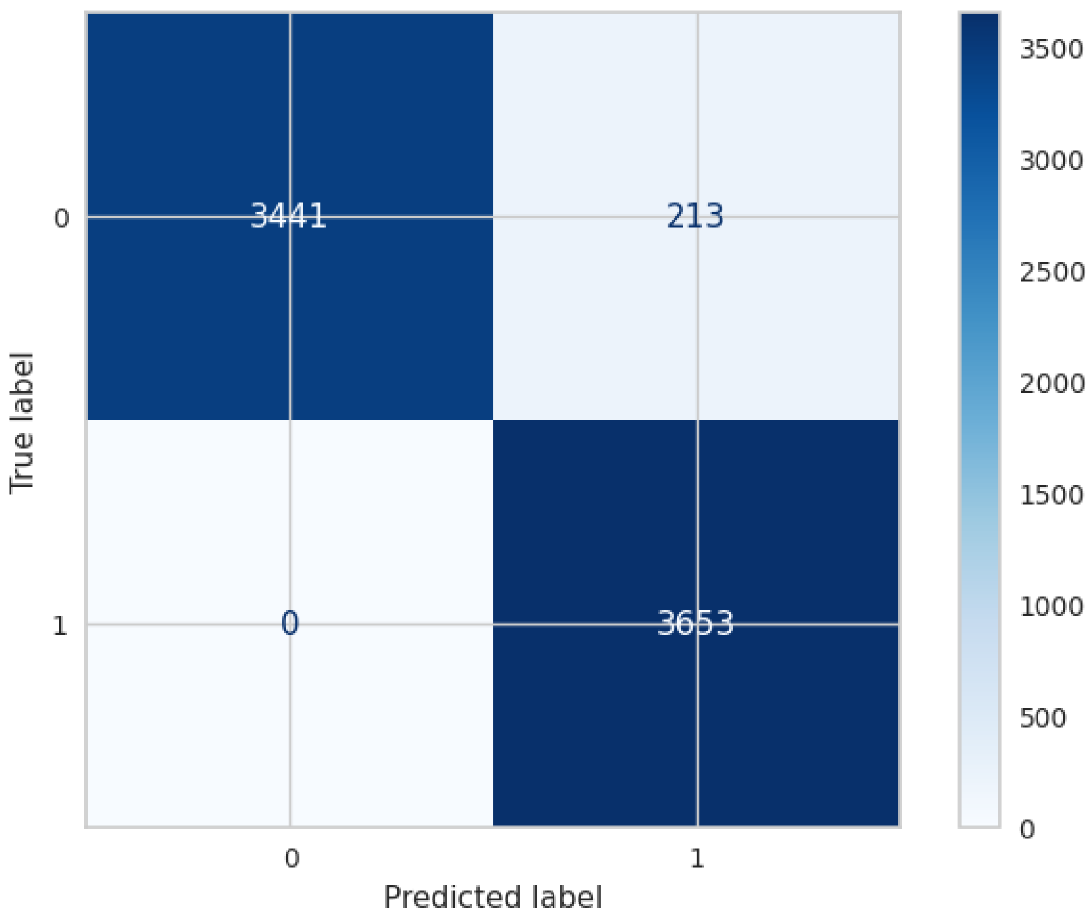
3440
214
0
3653
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500
0


3439
215
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0



3455
199
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0


3468
186
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0



[217]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Forest Over','Forest Over With Feature','Forest Over␣
↪Scaling','Foresr Over With Normalize','Forest Over With PCA'
                ,'Forest Over With PCA and Scaling',
                'Forest Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[217]:
```
                                    Train Accuracy  Test Accuracy   Test F1  \
Models
Forest Over                               0.999924       0.971397  0.972189
Forest Over With Feature                  0.988398       0.961544  0.962943
Forest Over Scaling                       0.999924       0.970713  0.971543
Foresr Over With Normalize                0.999924       0.970576  0.971413
Forest Over With PCA                      0.999924       0.972766  0.973484
Forest Over With PCA and Scaling          0.999924       0.974545  0.975174
Forest Over With PCA and Normalize        0.999909       0.970850  0.971672
```

```
                                     Test Recall  Test Precision       AUC
Models
Forest Over                             1.000000        0.945883  0.971401
Forest Over With Feature                0.999453        0.929008  0.961549
Forest Over Scaling                     1.000000        0.944660  0.970717
Foresr Over With Normalize              1.000000        0.944416  0.970580
Forest Over With PCA                    1.000000        0.948339  0.972770
Forest Over With PCA and Scaling        1.000000        0.951550  0.974548
Forest Over With PCA and Normalize      1.000000        0.944904  0.970854
```

[218]:
```
models_draw(df)
```

RandomUnderSampler

[219]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[220]:
```
Search(RandomForestClassifier(max_depth=20),{'max_depth':
 ↪[20,25,30,35,40]},X_train,y_train)
```

[220]: `RandomForestClassifier(max_depth=25)`

[221]:
```
cross_validation(RandomForestClassifier(max_depth=35),X_train,y_train)
```

```
Train Score Value :  [0.9998503 0.9998503 1.        1.        0.9998503]
Mean 0.9999101796407185
Test Score Value :  [0.88383234 0.88802395 0.87724551 0.89161677 0.88443114]
Mean 0.8850299401197604
```

[222]:
```
Values =␣
 ↪Models(RandomForestClassifier(max_depth=35),X_train,y_train,X_test,y_test)
```

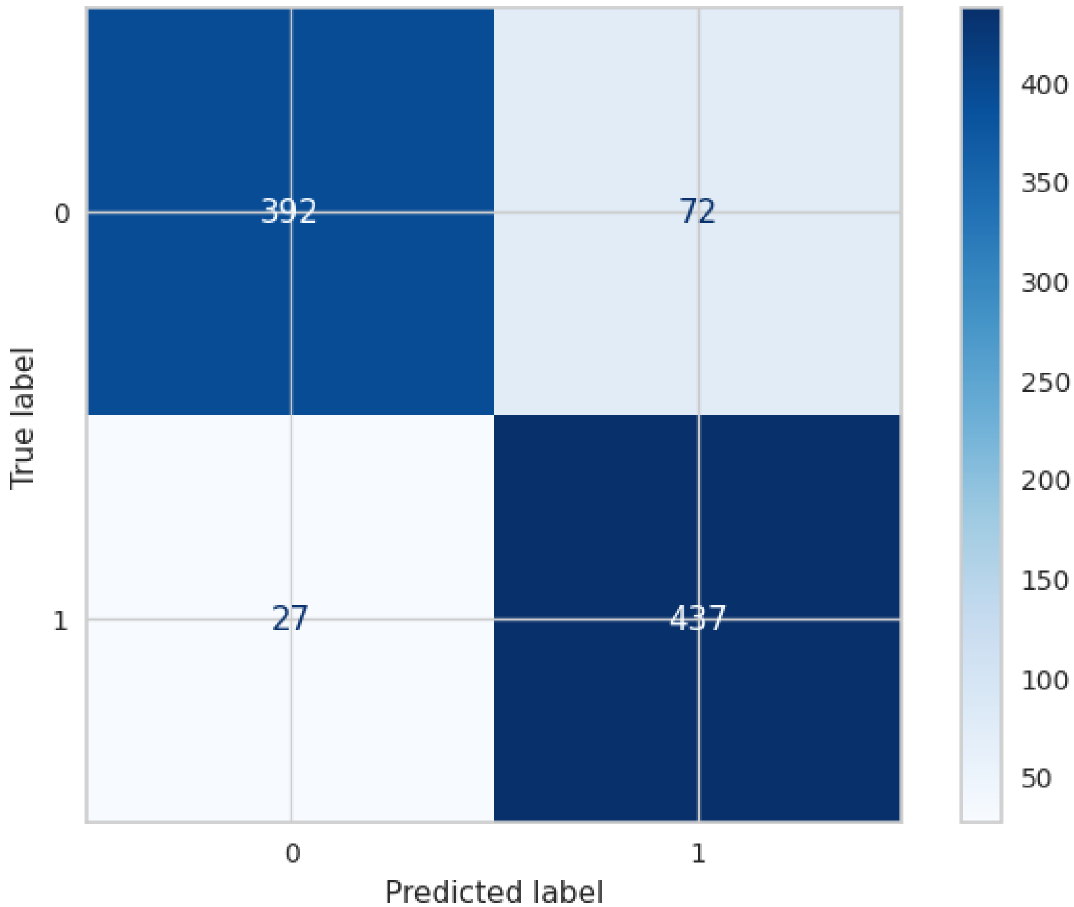
```
Apply Model With Normal Data :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8933189655172413
F1 Score is :  0.8982528263103802
Recall Score is :  0.9418103448275862
Precision Score is :  0.8585461689587426
AUC Value  :  0.8933189655172414

Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.84      0.89       464
```

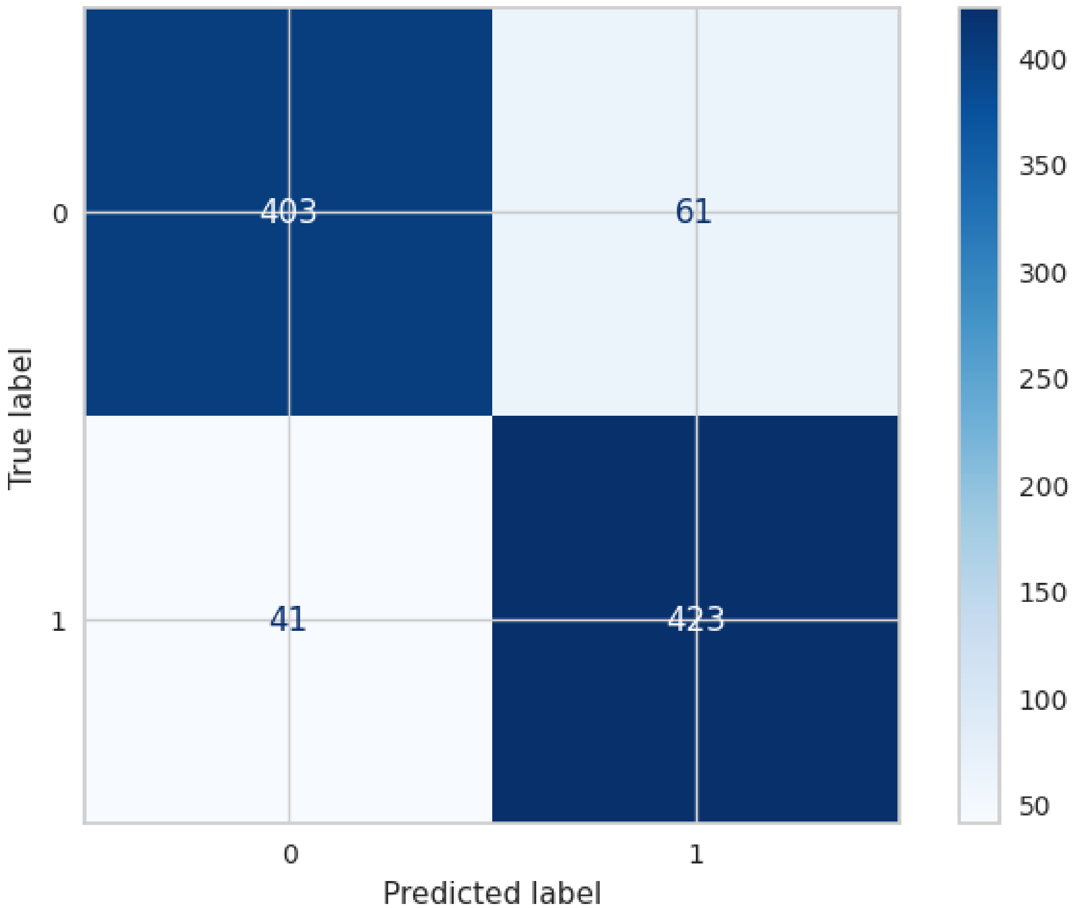
```
           1       0.86      0.94      0.90       464

    accuracy                           0.89       928
   macro avg       0.90      0.89      0.89       928
weighted avg       0.90      0.89      0.89       928

Confusion Matrix is :
 [[392  72]
 [ 27 437]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9906586826347306
Model Test Score is :  0.8900862068965517
F1 Score is :  0.8924050632911392
Recall Score is :  0.9116379310344828
Precision Score is :  0.8739669421487604
AUC Value  :  0.8900862068965517
Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.87      0.89       464
           1       0.87      0.91      0.89       464

    accuracy                           0.89       928
   macro avg       0.89      0.89      0.89       928
weighted avg       0.89      0.89      0.89       928

Confusion Matrix is :
 [[403  61]
 [ 41 423]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8879310344827587
F1 Score is :  0.8925619834710743
Recall Score is :  0.9310344827586207
Precision Score is :  0.8571428571428571
AUC Value  :  0.8879310344827586
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.84      0.88       464
           1       0.86      0.93      0.89       464
```

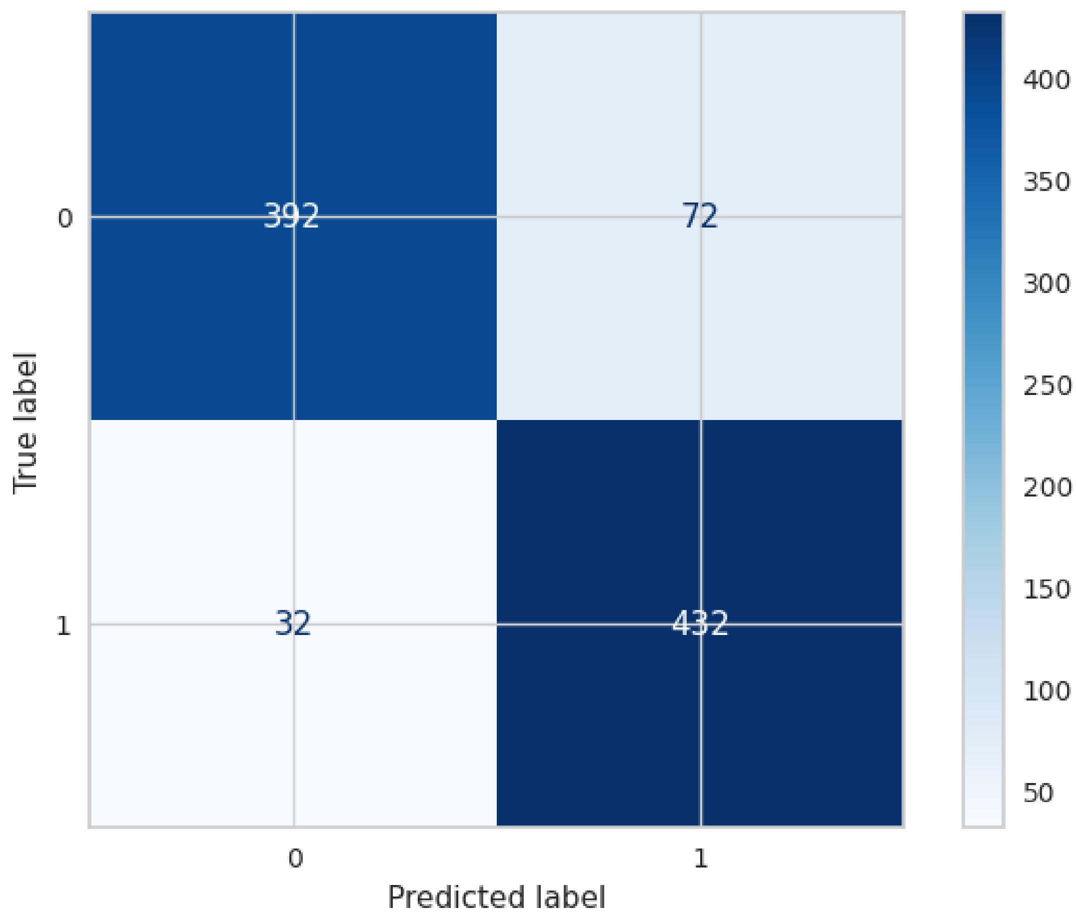
```
    accuracy                           0.89       928
   macro avg       0.89      0.89      0.89       928
weighted avg       0.89      0.89      0.89       928

Confusion Matrix is :
 [[392  72]
 [ 32 432]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8879310344827587
F1 Score is :  0.892116182572614
Recall Score is :  0.9267241379310345
Precision Score is :  0.86
AUC Value  :  0.8879310344827586
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.85      0.88       464
           1       0.86      0.93      0.89       464

    accuracy                           0.89       928
   macro avg       0.89      0.89      0.89       928
weighted avg       0.89      0.89      0.89       928

Confusion Matrix is :
 [[394  70]
 [ 34 430]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8933189655172413
F1 Score is :  0.8984615384615384
Recall Score is :  0.9439655172413793
Precision Score is :  0.8571428571428571
AUC Value  :  0.8933189655172414
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.84      0.89       464
           1       0.86      0.94      0.90       464
```

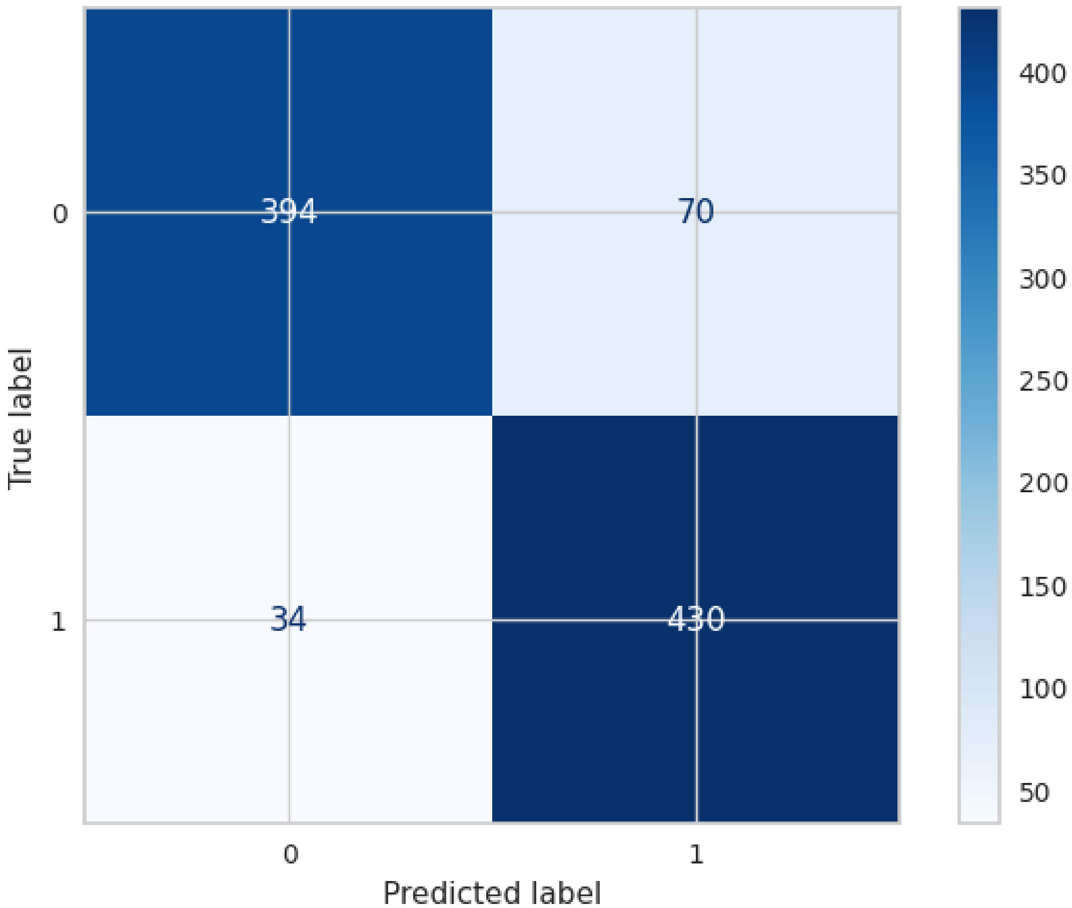
```
    accuracy                           0.89       928
   macro avg       0.90      0.89      0.89       928
weighted avg       0.90      0.89      0.89       928

Confusion Matrix is :
 [[391  73]
 [ 26 438]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9997604790419161
Model Test Score is :  0.8685344827586207
F1 Score is :  0.8737060041407868
Recall Score is :  0.9094827586206896
Precision Score is :  0.8406374501992032
AUC Value  :  0.8685344827586207

Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.83      0.86       464
           1       0.84      0.91      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[384  80]
 [ 42 422]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.884698275862069
F1 Score is :  0.8886576482830385
Recall Score is :  0.9202586206896551
Precision Score is :  0.8591549295774648
AUC Value  :  0.8846982758620691

Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.85      0.88       464
           1       0.86      0.92      0.89       464

    accuracy                           0.88       928
```

```
   macro avg       0.89      0.88      0.88       928
weighted avg       0.89      0.88      0.88       928

Confusion Matrix is :
 [[394  70]
 [ 37 427]]
```

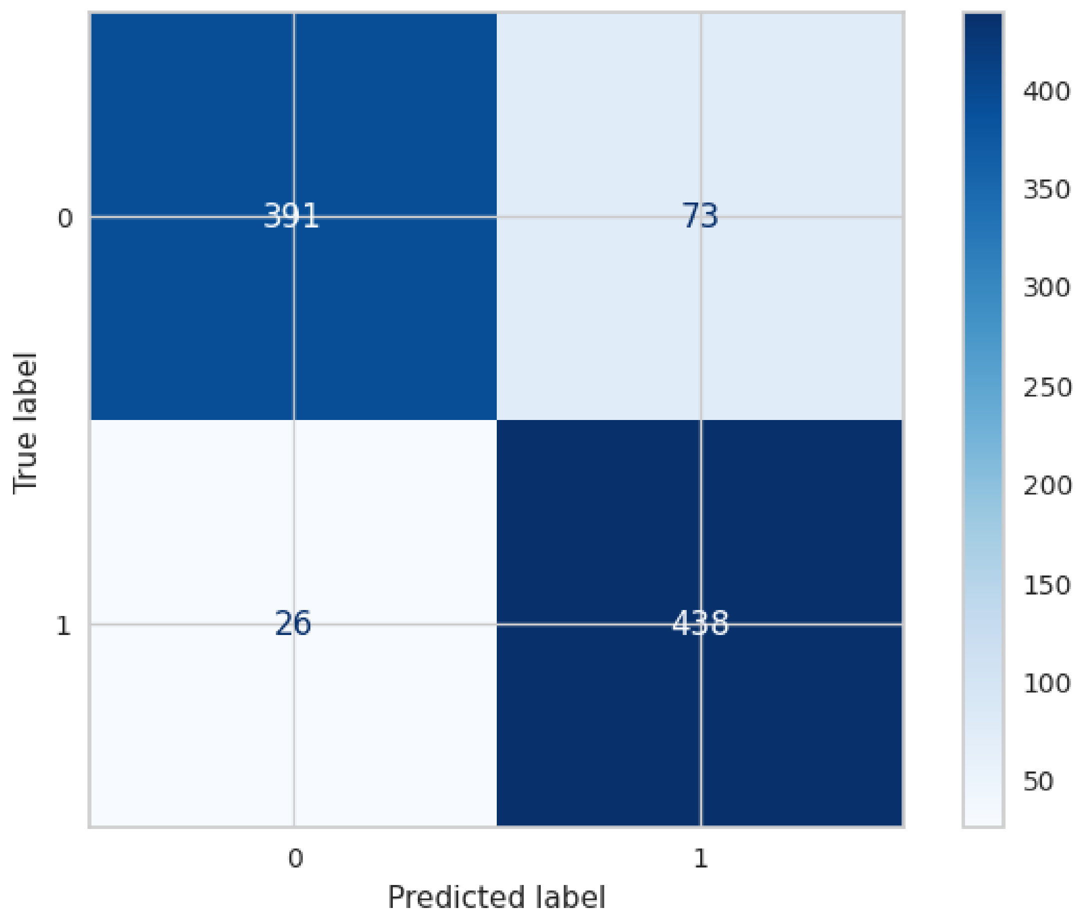


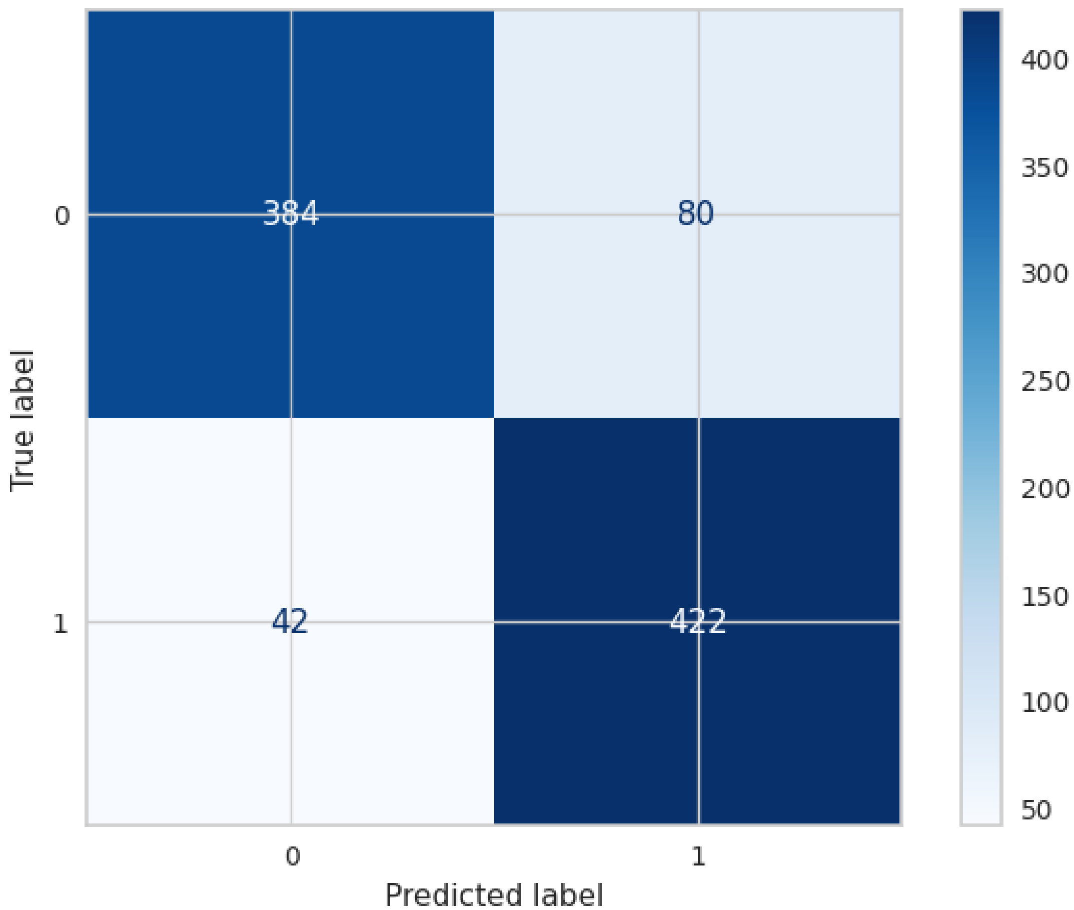

403
61
41
423
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50

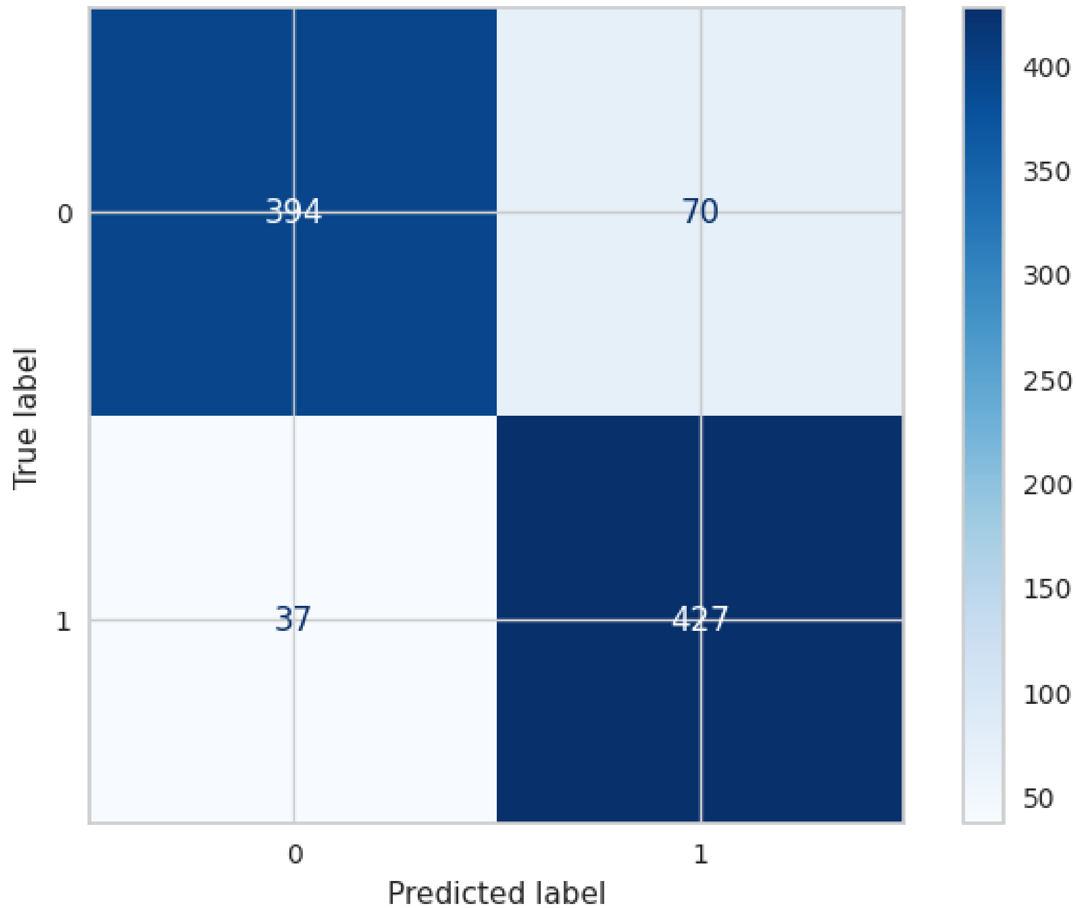

True label
Predicted label
0
1
0
1
392
72
32
432
400
350
300
250
200
150
100
50



394
70
34
430
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


391
73
26
438
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


384
80
42
422
0
1
0
1
True label
Predicted label
400
350
300
250
200
150
100
50



[223]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Forest Under','Forest Under With Feature','Forest Under␣
↪Scaling','Foresr Under With Normalize','Forest Under With PCA'
                ,'Forest Under With PCA and Scaling',
                'Forest Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[223]:
```
                                    Train Accuracy  Test Accuracy   Test F1  \
Models
Forest Under                              0.999880       0.893319  0.898253
Forest Under With Feature                 0.990659       0.890086  0.892405
Forest Under Scaling                      0.999880       0.887931  0.892562
Foresr Under With Normalize               0.999880       0.887931  0.892116
Forest Under With PCA                     0.999880       0.893319  0.898462
Forest Under With PCA and Scaling         0.999760       0.868534  0.873706
Forest Under With PCA and Normalize       0.999880       0.884698  0.888658
```

```
                                      Test Recall  Test Precision       AUC
Models
Forest Under                             0.941810        0.858546  0.893319
Forest Under With Feature                0.911638        0.873967  0.890086
Forest Under Scaling                     0.931034        0.857143  0.887931
Foresr Under With Normalize              0.926724        0.860000  0.887931
Forest Under With PCA                    0.943966        0.857143  0.893319
Forest Under With PCA and Scaling        0.909483        0.840637  0.868534
Forest Under With PCA and Normalize      0.920259        0.859155  0.884698
```

[224]:
```python
models_draw(df)
```

DecisionTreeClassifier

[225]:
```python
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[226]:
```python
Search(DecisionTreeClassifier(max_depth=20),{'max_depth':
 ↪[5,10,15,20,25,30,35,40]},X_train,y_train)
```

[226]: DecisionTreeClassifier(max_depth=5)

[227]:
```python
cross_validation(DecisionTreeClassifier(max_depth=5),X_train,y_train)
```

```
Train Score Value :  [0.91387802 0.91506156 0.9149941  0.91361106 0.91121606]
Mean 0.9137521597437622
Test Score Value :  [0.90879655 0.91256241 0.90500607 0.91188773 0.91593577]
Mean 0.9108377062057444
```

[228]:
```python
Values =␣
 ↪Models(DecisionTreeClassifier(max_depth=5),X_train,y_train,X_test,y_test)
```

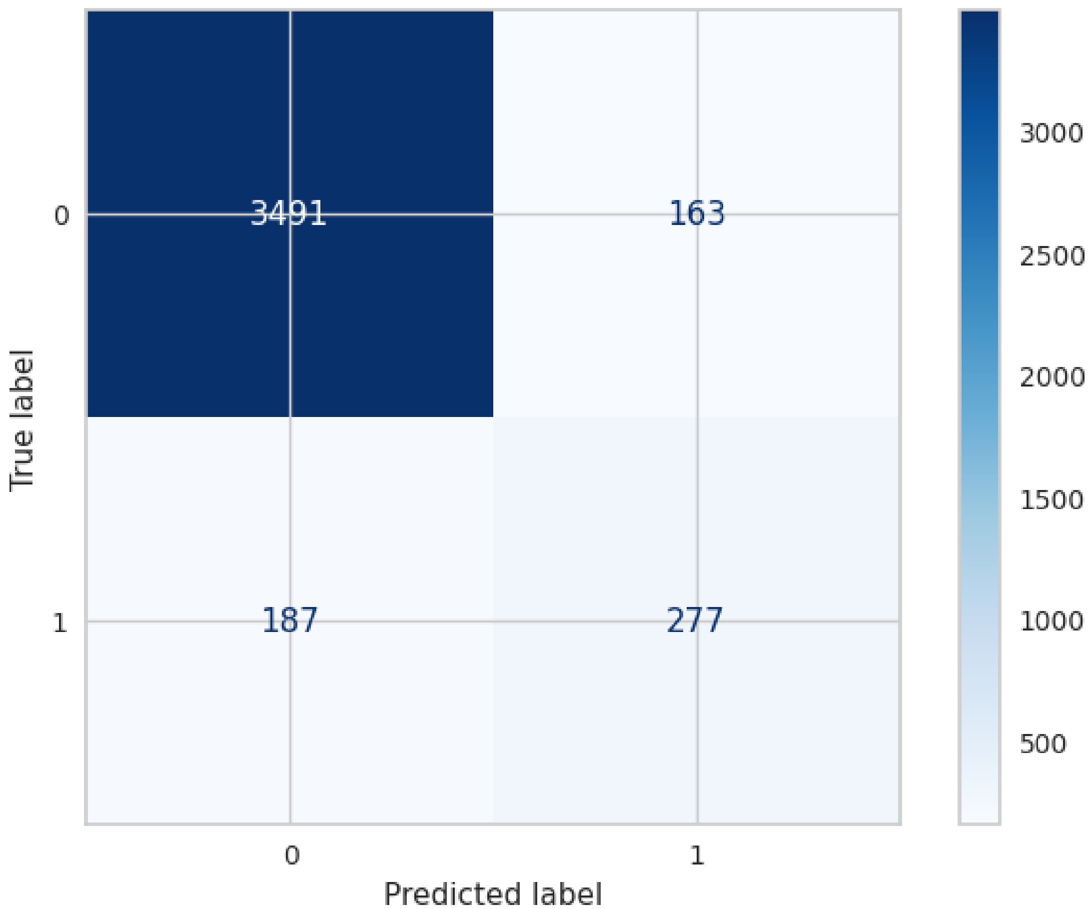
```
Apply Model With Normal Data :

Model Train Score is :  0.9132933937823834
Model Test Score is :  0.9150072850898494
F1 Score is :  0.6128318584070797
Recall Score is :  0.5969827586206896
Precision Score is :  0.6295454545454545
AUC Value  :  0.7761870552818828

Classification Report is :                precision    recall  f1-score
support

           0       0.95      0.96      0.95      3654
```

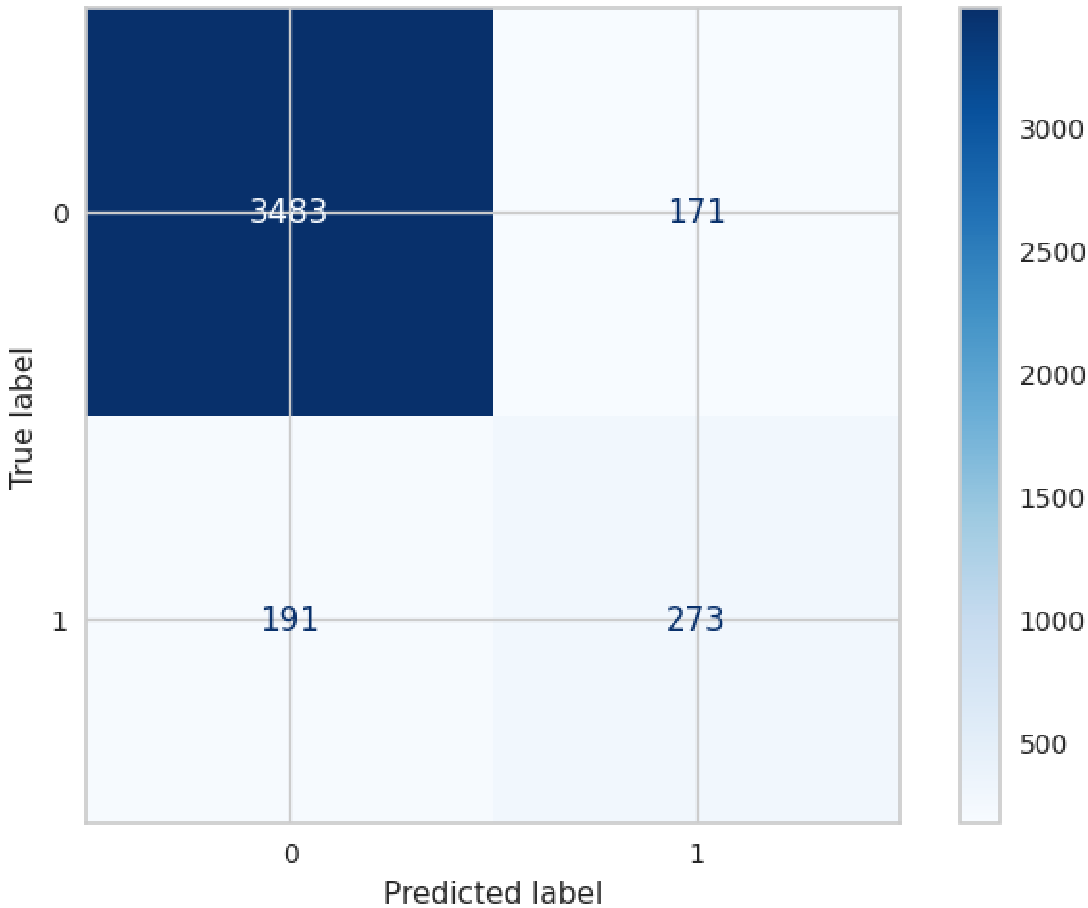
```
           1       0.63      0.60      0.61       464

    accuracy                           0.92      4118
   macro avg       0.79      0.78      0.78      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3491  163]
 [ 187  277]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9093534110535406
Model Test Score is :  0.9120932491500728
F1 Score is :  0.6013215859030837
Recall Score is :  0.5883620689655172
Precision Score is :  0.6148648648648649
AUC Value  :  0.7707820197044335
Classification Report is :                precision    recall  f1-score
support

           0       0.95      0.95      0.95      3654
           1       0.61      0.59      0.60       464

    accuracy                           0.91      4118
   macro avg       0.78      0.77      0.78      4118
weighted avg       0.91      0.91      0.91      4118

Confusion Matrix is :
 [[3483  171]
 [ 191  273]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9132933937823834
Model Test Score is :  0.9150072850898494
F1 Score is :  0.6128318584070797
Recall Score is :  0.5969827586206896
Precision Score is :  0.6295454545454545
AUC Value  :  0.7761870552818828
Classification Report is :                precision    recall  f1-score
support

           0       0.95      0.96      0.95      3654
           1       0.63      0.60      0.61       464
```

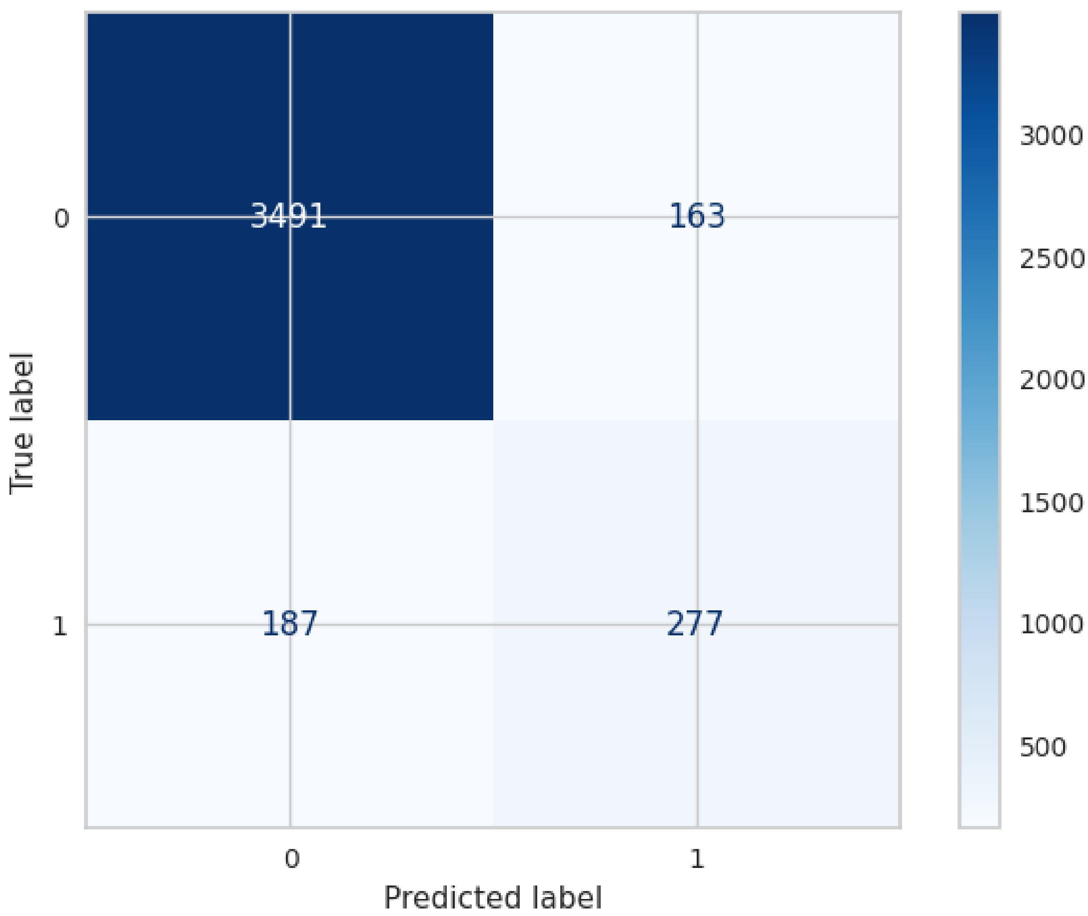
```
    accuracy                           0.92      4118
   macro avg       0.79      0.78      0.78      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3491  163]
 [ 187  277]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9134283246977547
Model Test Score is :  0.912821758135017
F1 Score is :  0.5282522996057819
Recall Score is :  0.4331896551724138
Precision Score is :  0.6767676767676768
AUC Value  :  0.7034585385878489

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.97      0.95      3654
           1       0.68      0.43      0.53       464

    accuracy                           0.91      4118
   macro avg       0.80      0.70      0.74      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3558   96]
 [ 263  201]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9149395509499136
Model Test Score is :  0.9133074307916464
F1 Score is :  0.5993265993265994
Recall Score is :  0.5754310344827587
Precision Score is :  0.6252927400468384
AUC Value  :  0.765821702244116

Classification Report is :                precision    recall  f1-score
support

           0       0.95      0.96      0.95      3654
           1       0.63      0.58      0.60       464
```

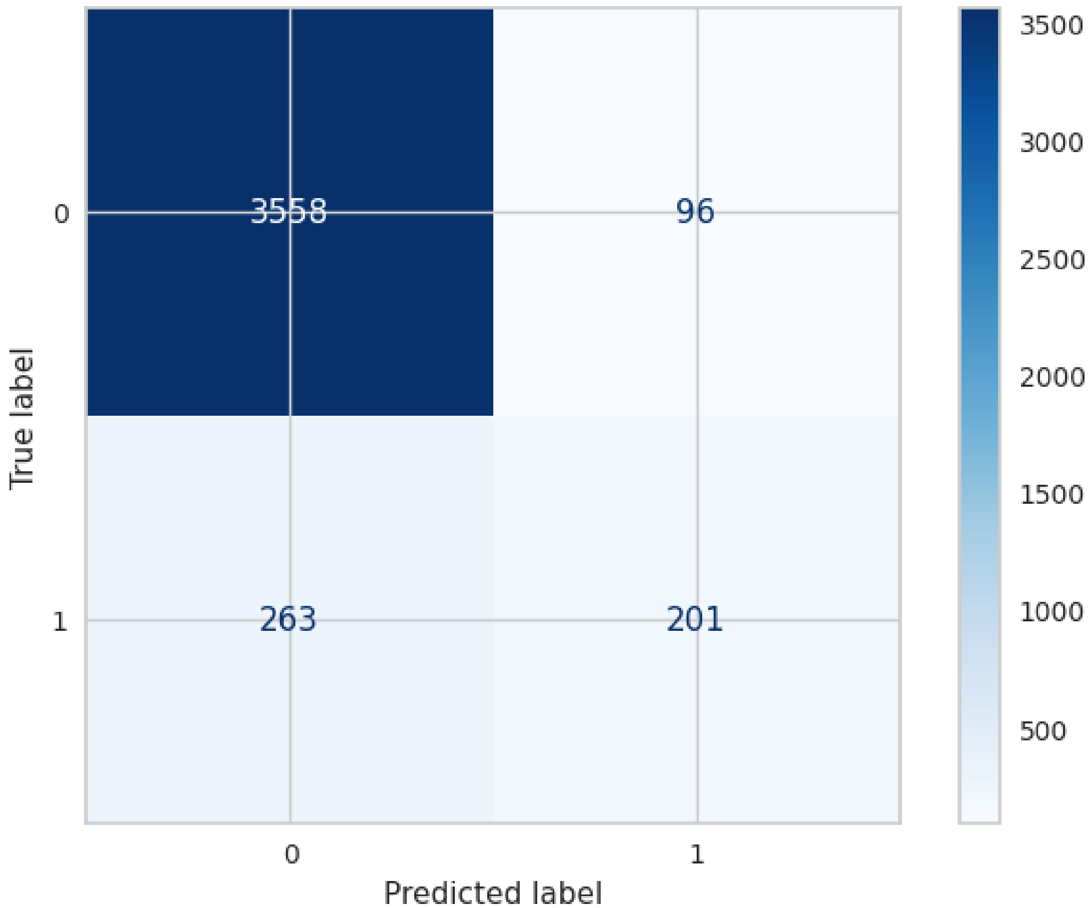
```
    accuracy                           0.91      4118
   macro avg       0.79      0.77      0.78      4118
weighted avg       0.91      0.91      0.91      4118

Confusion Matrix is :
 [[3494  160]
 [ 197  267]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9082199913644214
Model Test Score is :  0.9084507042253521
F1 Score is :  0.4800000000000001
Recall Score is :  0.375
Precision Score is :  0.6666666666666666
AUC Value  :  0.675595238095238

Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
           1       0.67      0.38      0.48       464

    accuracy                           0.91      4118
   macro avg       0.80      0.68      0.71      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3567   87]
 [ 290  174]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9111614853195165
Model Test Score is :  0.9113647401651287
F1 Score is :  0.5326504481434058
Recall Score is :  0.4482758620689655
Precision Score is :  0.6561514195583596
AUC Value  :  0.7092227695675971

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.97      0.95      3654
           1       0.66      0.45      0.53       464

    accuracy                           0.91      4118
```

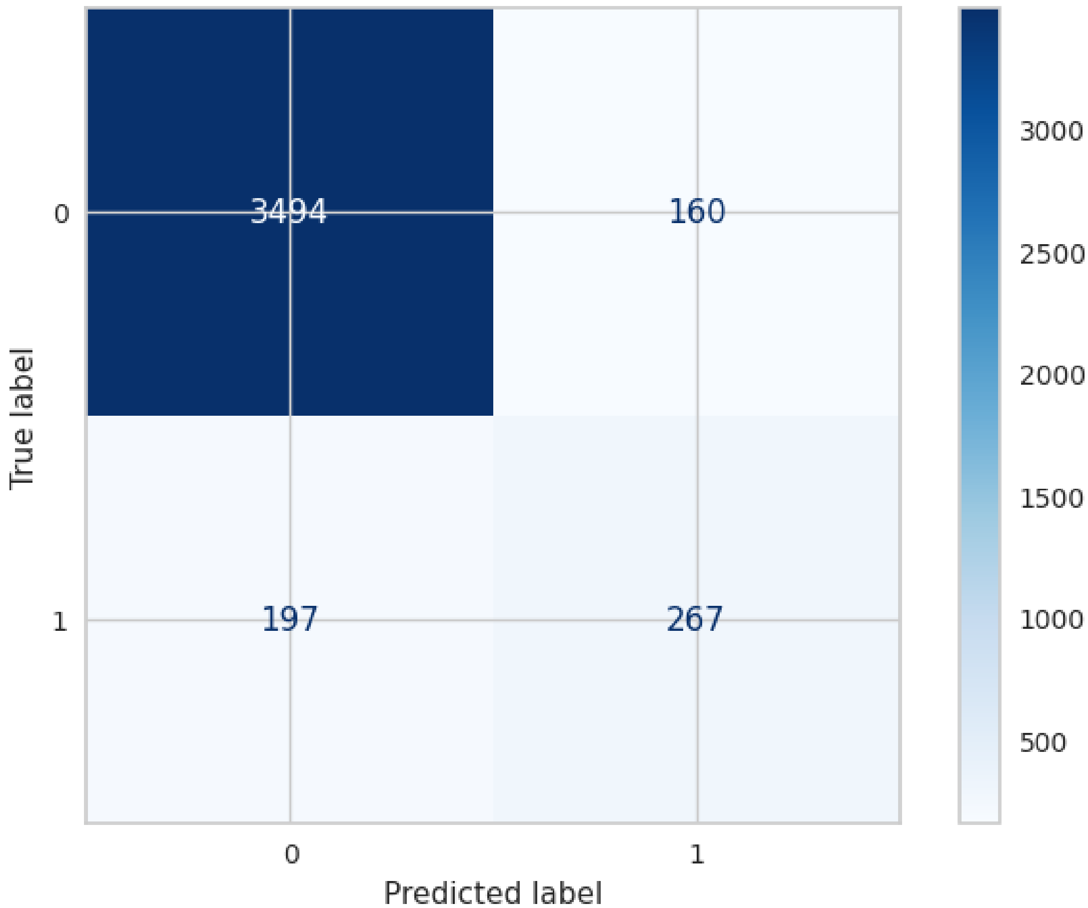
```
   macro avg       0.79      0.71      0.74      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3545  109]
 [ 256  208]]
```



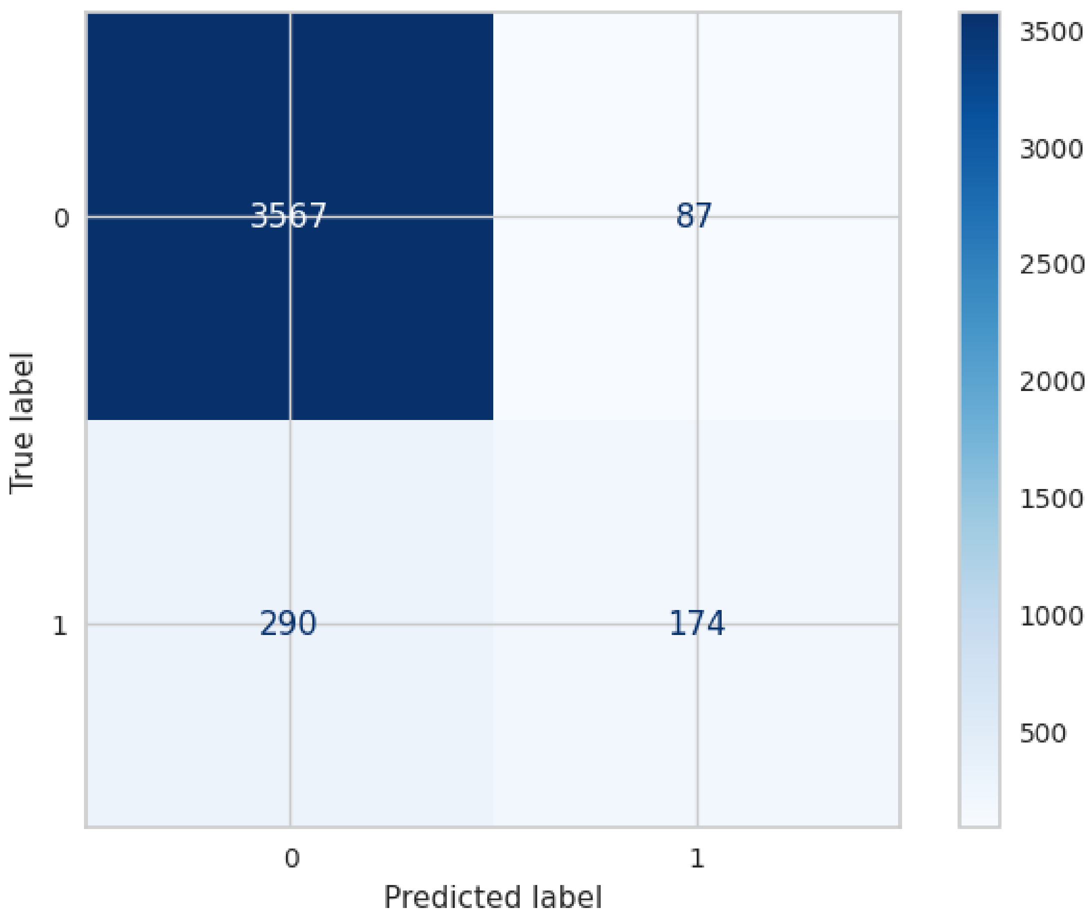

3483
171
191
273
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500

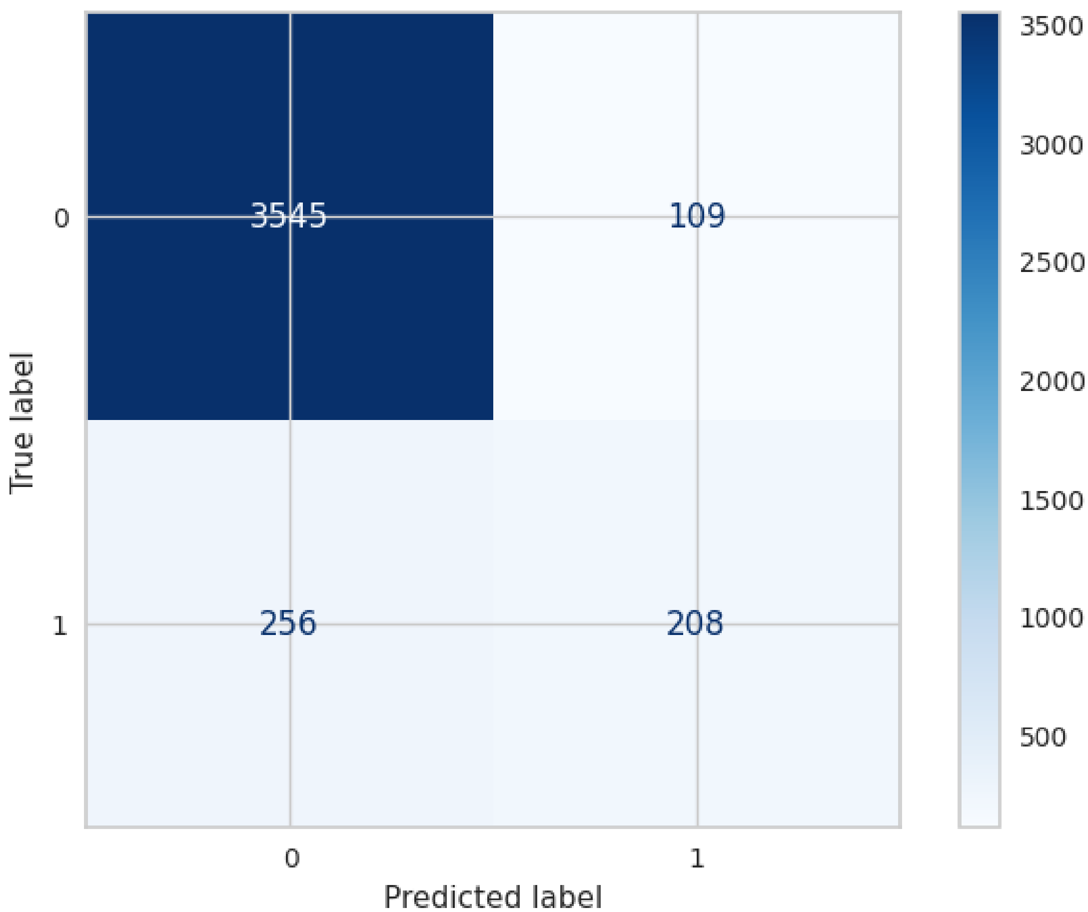

3491
163
187
277
0
1
0
1
True label
Predicted label
3000
2500
2000
1500
1000
500


3558
96
263
201
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500



3494
160
197
267
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


3567
87
290
174
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[229]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Decision','Decision With Feature','Decision Scaling','Decision␣
↪With Normalize','Decision With PCA','Decision With PCA and Scaling',
                 'Decision With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[229]:
```
                                 Train Accuracy  Test Accuracy   Test F1  \
Models
Decision                               0.913293       0.915007  0.612832
Decision With Feature                  0.909353       0.912093  0.601322
Decision Scaling                       0.913293       0.915007  0.612832
Decision With Normalize                0.913428       0.912822  0.528252
Decision With PCA                      0.914940       0.913307  0.599327
Decision With PCA and Scaling          0.908220       0.908451  0.480000
Decision With PCA and Normalize        0.911161       0.911365  0.532650
```

```
                                  Test Recall  Test Precision       AUC
Models
Decision                             0.596983        0.629545  0.776187
Decision With Feature                0.588362        0.614865  0.770782
Decision Scaling                     0.596983        0.629545  0.776187
Decision With Normalize              0.433190        0.676768  0.703459
Decision With PCA                    0.575431        0.625293  0.765822
Decision With PCA and Scaling        0.375000        0.666667  0.675595
Decision With PCA and Normalize      0.448276        0.656151  0.709223
```

[230]:
```python
models_draw(df)
```

RandomOverSampler

[231]:
```python
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[232]:
```python
Search(DecisionTreeClassifier(max_depth=20),{'max_depth':
 ↪[20,25,30,35,40]},X_train,y_train)
```

[232]:
```
DecisionTreeClassifier(max_depth=35)
```

[233]:
```python
cross_validation(DecisionTreeClassifier(max_depth=35),X_train,y_train)
```

```
Train Score Value :  [0.99992397 0.99990496 0.99992397 0.99990496 0.99996199]
Mean 0.9999239693330242
Test Score Value :  [0.96396259 0.96228997 0.96434274 0.96441606 0.96411192]
Mean 0.9638246563974839
```

[234]:
```python
Values =␣
 ↪Models(DecisionTreeClassifier(max_depth=35),X_train,y_train,X_test,y_test)
```

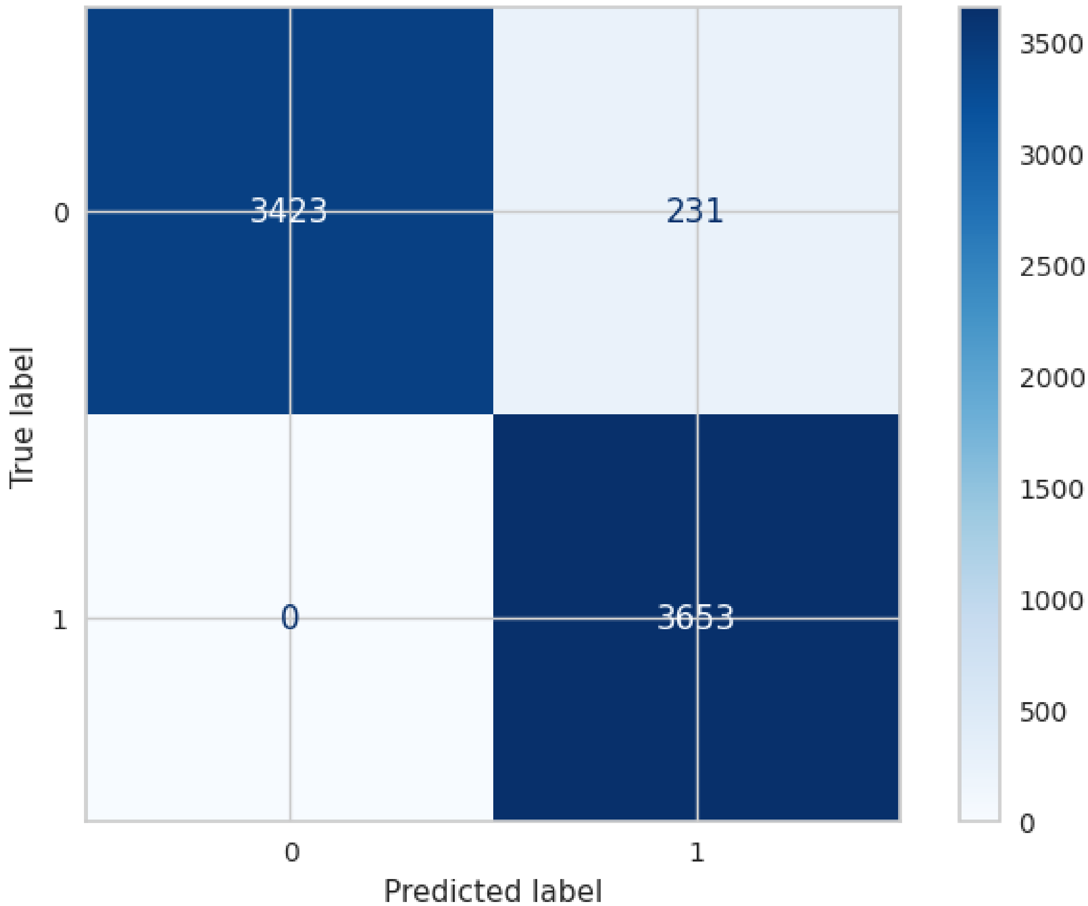
```
Apply Model With Normal Data :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9683864787190365
F1 Score is :  0.9693512007430012
Recall Score is :  1.0
Precision Score is :  0.9405252317198765
AUC Value  :  0.9683908045977012

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653
```

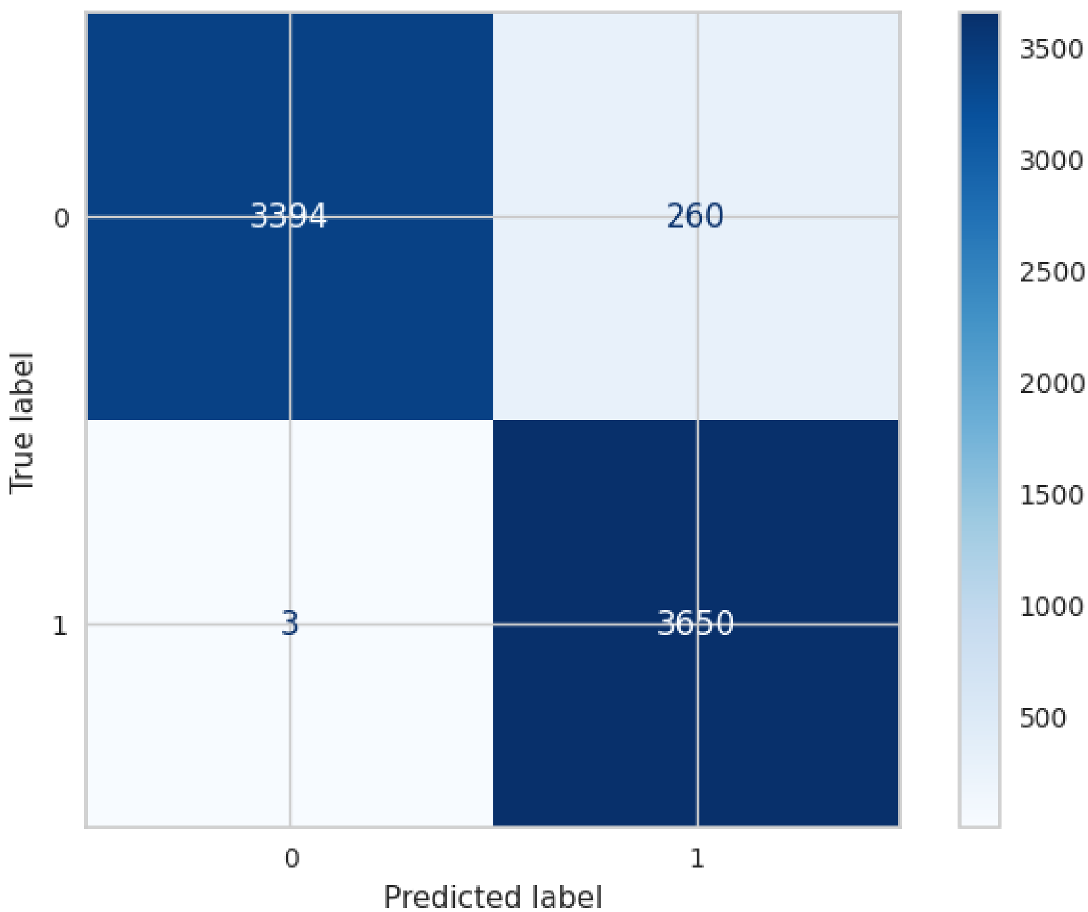
```
    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3423  231]
 [   0 3653]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9883977312470538
Model Test Score is :  0.964007116463665
F1 Score is :  0.9652254396403543
Recall Score is :  0.9991787571858747
Precision Score is :  0.9335038363171355
AUC Value  :  0.9640119292223844
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.93      0.96      3654
           1       0.93      1.00      0.97      3653

    accuracy                           0.96      7307
   macro avg       0.97      0.96      0.96      7307
weighted avg       0.97      0.96      0.96      7307

Confusion Matrix is :
 [[3394  260]
 [   3 3650]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9681127685780758
F1 Score is :  0.969094044302958
Recall Score is :  1.0
Precision Score is :  0.9400411734431292
AUC Value  :  0.9681171319102353
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653
```

```
    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3421  233]
 [   0 3653]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9949363623922266
Model Test Score is :  0.967428493225674
F1 Score is :  0.9684517497348887
Recall Score is :  1.0
Precision Score is :  0.9388332048316628
AUC Value  :  0.9674329501915708
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.93      0.97      3654
           1       0.94      1.00      0.97      3653

    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3416  238]
 [   0 3653]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9909067408725271
Model Test Score is :  0.9660599425208704
F1 Score is :  0.9671697114111728
Recall Score is :  1.0
Precision Score is :  0.9364265572930018
AUC Value  :  0.966064586754242
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.93      0.96      3654
           1       0.94      1.00      0.97      3653

    accuracy                           0.97      7307
```

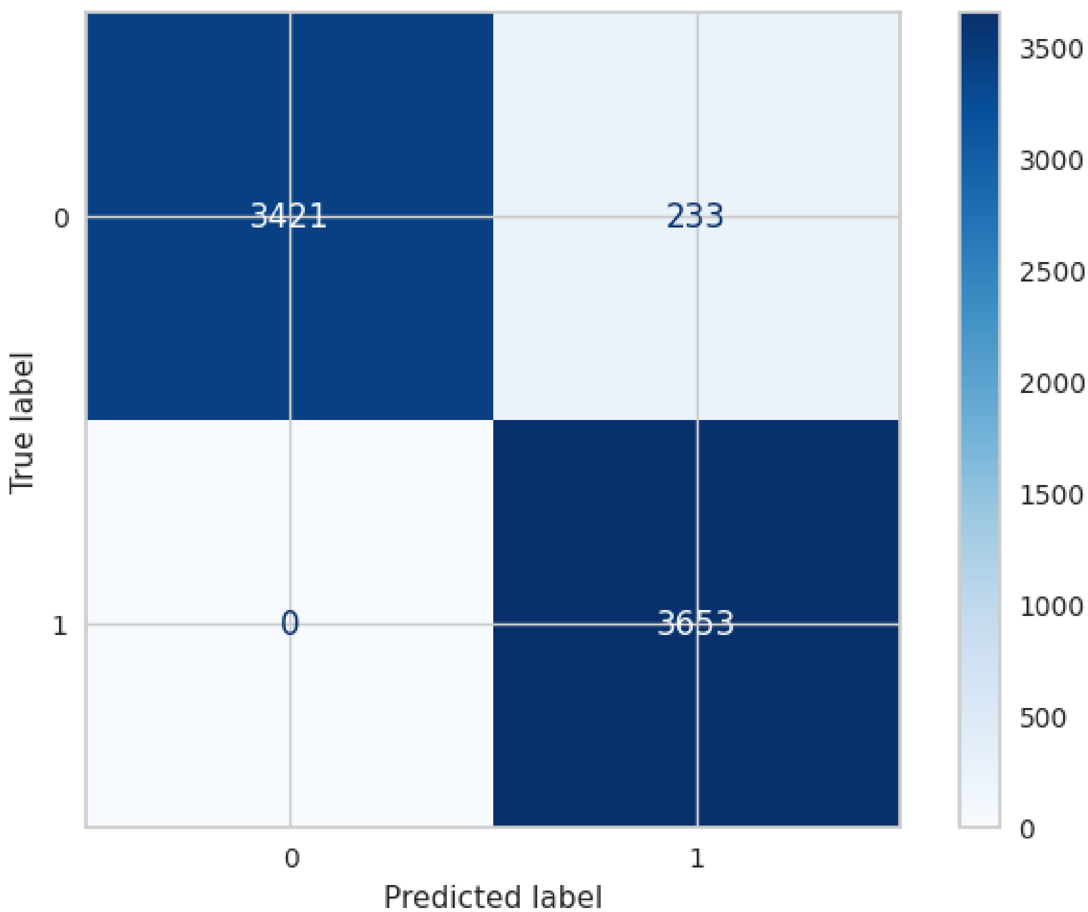
```
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3406  248]
 [   0 3653]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9970956312820279
Model Test Score is :  0.9637334063227042
F1 Score is :  0.9649980187557786
Recall Score is :  1.0
Precision Score is :  0.9323634507401736
AUC Value  :  0.9637383689107827
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.93      0.96      3654
           1       0.93      1.00      0.96      3653

    accuracy                           0.96      7307
   macro avg       0.97      0.96      0.96      7307
weighted avg       0.97      0.96      0.96      7307

Confusion Matrix is :
 [[3389  265]
 [   0 3653]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9921384365068504
Model Test Score is :  0.9644176816751061
F1 Score is :  0.9656357388316151
Recall Score is :  1.0
Precision Score is :  0.9335548172757475
AUC Value  :  0.9644225506294472
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.93      0.96      3654
           1       0.93      1.00      0.97      3653

    accuracy                           0.96      7307
   macro avg       0.97      0.96      0.96      7307
```

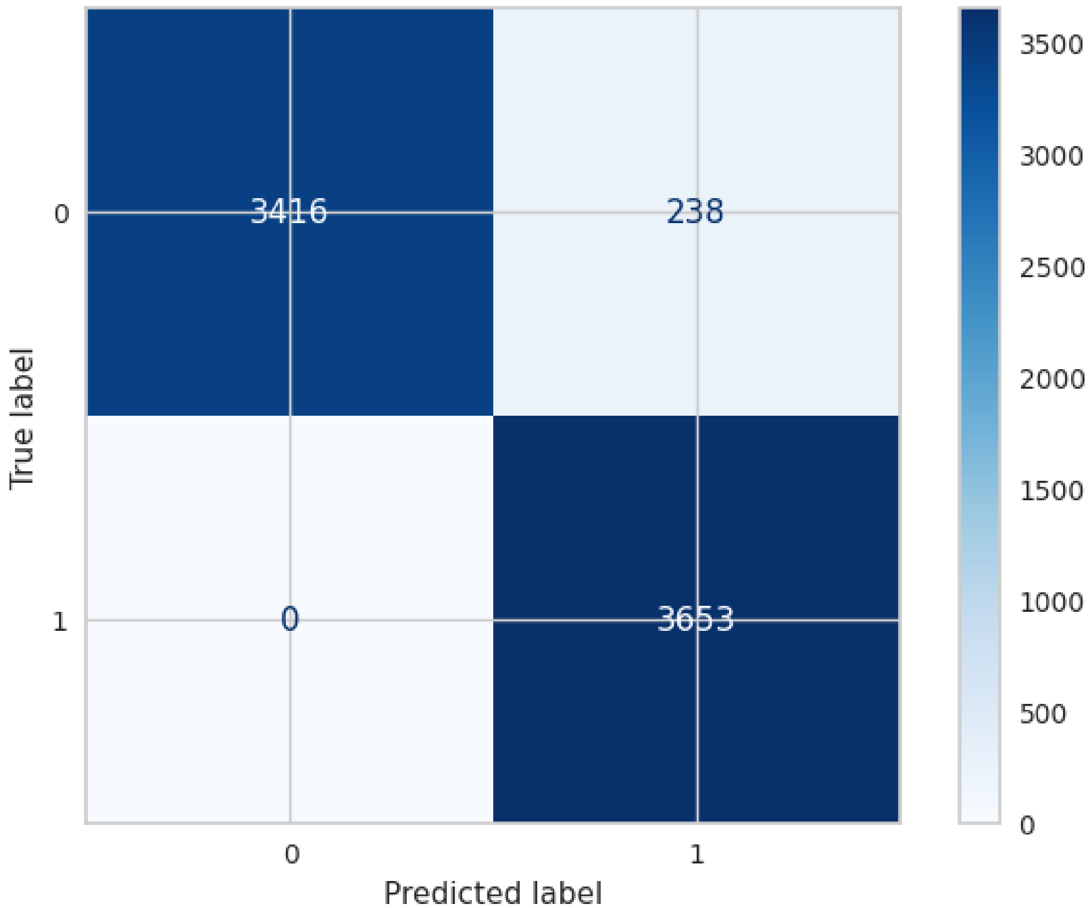
```
weighted avg       0.97      0.96      0.96      7307

Confusion Matrix is :
 [[3394  260]
 [   0 3653]]
```



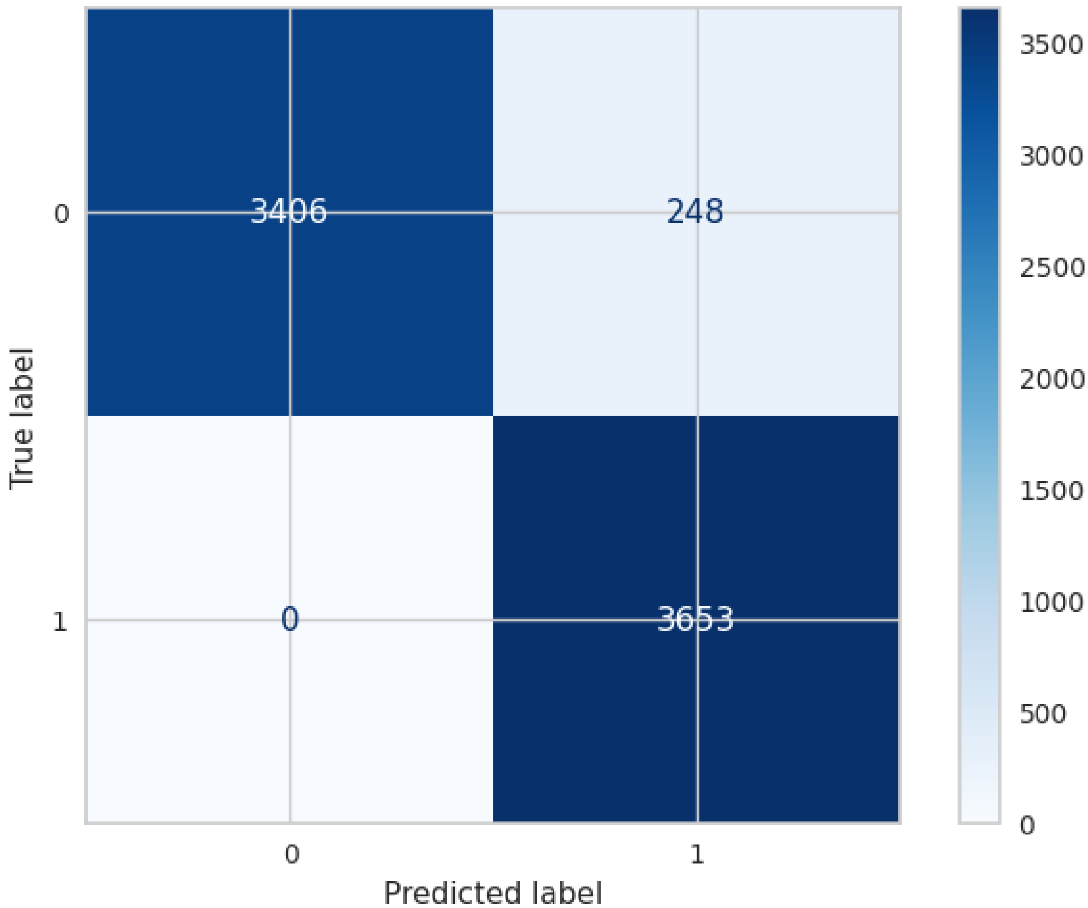


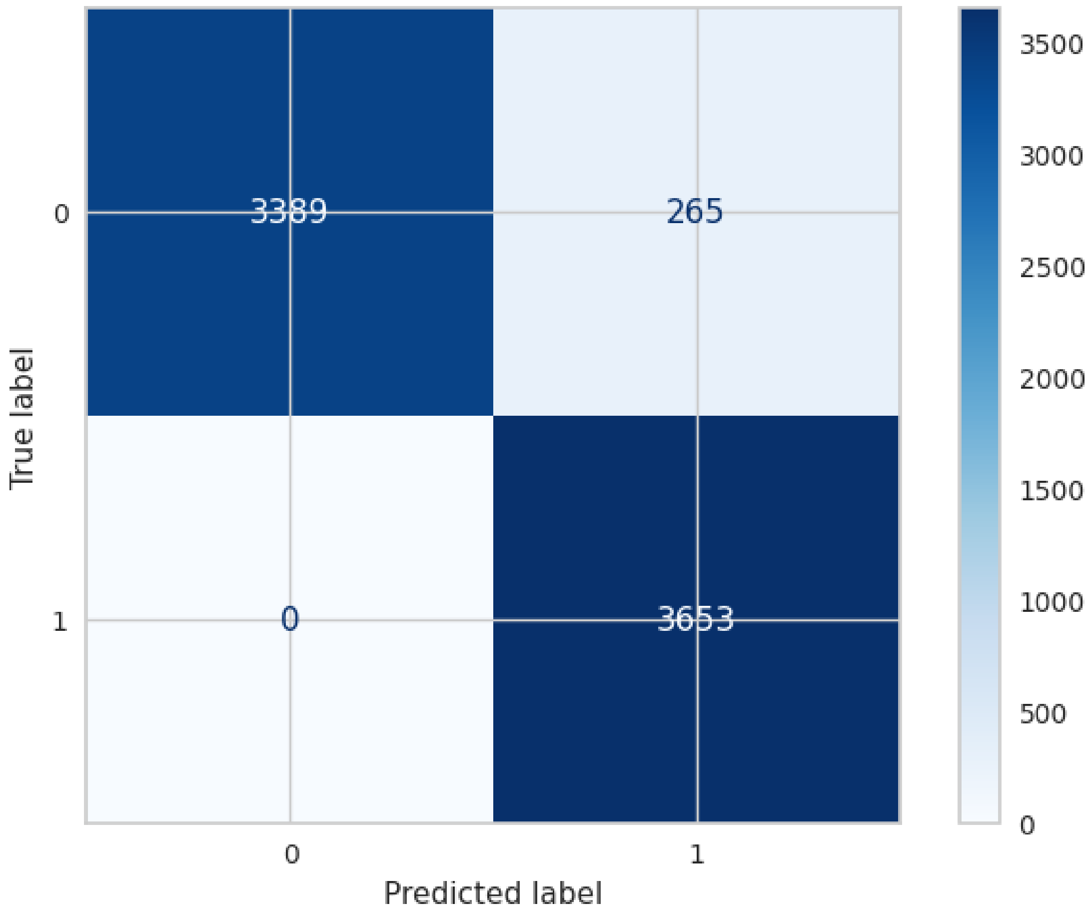
3394
260
3
3650
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500

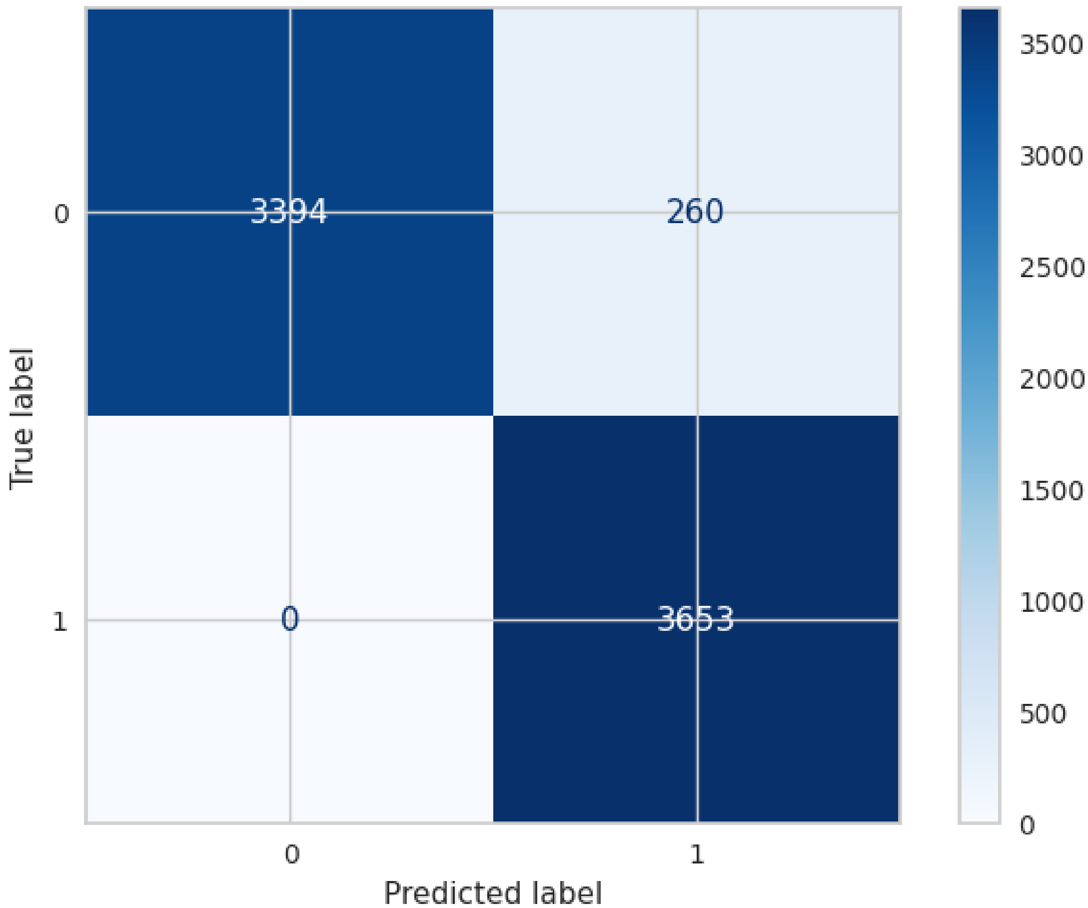

3421
233
0
3653
0
1
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500
0



3416
238
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0


3406
248
0
3653
0
1
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500
0


3389
265
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0



[235]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Decision Over','Decision Over With Feature','Decision Over␣
↪Scaling','Decision Over With Normalize','Decision Over With PCA'
                ,'Decision Over With PCA and Scaling',
                'Decision Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[235]:
```
                                      Train Accuracy  Test Accuracy   Test F1  \
Models
Decision Over                               0.999924       0.968386  0.969351
Decision Over With Feature                  0.988398       0.964007  0.965225
Decision Over Scaling                       0.999924       0.968113  0.969094
Decision Over With Normalize                0.994936       0.967428  0.968452
Decision Over With PCA                      0.990907       0.966060  0.967170
Decision Over With PCA and Scaling          0.997096       0.963733  0.964998
Decision Over With PCA and Normalize        0.992138       0.964418  0.965636
```

```
                                         Test Recall  Test Precision       AUC
Models
Decision Over                               1.000000        0.940525  0.968391
Decision Over With Feature                  0.999179        0.933504  0.964012
Decision Over Scaling                       1.000000        0.940041  0.968117
Decision Over With Normalize                1.000000        0.938833  0.967433
Decision Over With PCA                      1.000000        0.936427  0.966065
Decision Over With PCA and Scaling          1.000000        0.932363  0.963738
Decision Over With PCA and Normalize        1.000000        0.933555  0.964423
```

[236]:
```
models_draw(df)
```

RandomUnderSampler

[237]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[238]:
```
Search(DecisionTreeClassifier(max_depth=20),{'max_depth':
  ↪[20,25,30,35,40]},X_train,y_train)
```

[238]:
```
DecisionTreeClassifier(max_depth=20)
```

[239]:
```
cross_validation(DecisionTreeClassifier(max_depth=35),X_train,y_train)
```

```
Train Score Value :  [0.9998503 0.9998503 1.        1.        0.9998503]
Mean 0.9999101796407185
Test Score Value :  [0.82934132 0.82994012 0.83413174 0.82994012 0.83892216]
Mean 0.8324550898203592
```

[240]:
```
Values =␣
  ↪Models(DecisionTreeClassifier(max_depth=35),X_train,y_train,X_test,y_test)
```

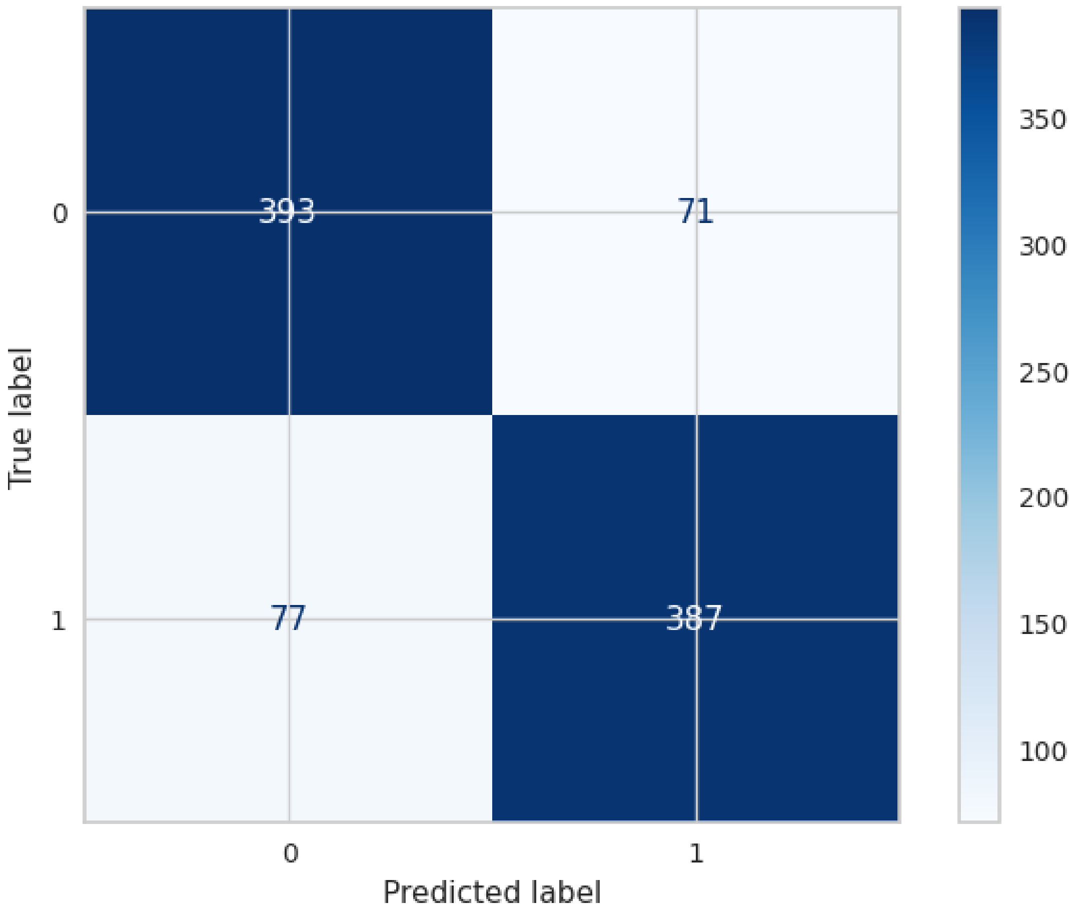
```
Apply Model With Normal Data :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8405172413793104
F1 Score is :  0.8394793926247288
Recall Score is :  0.834051724137931
Precision Score is :  0.8449781659388647
AUC Value  :  0.8405172413793104

Classification Report is :               precision    recall  f1-score
support

           0       0.84      0.85      0.84       464
```

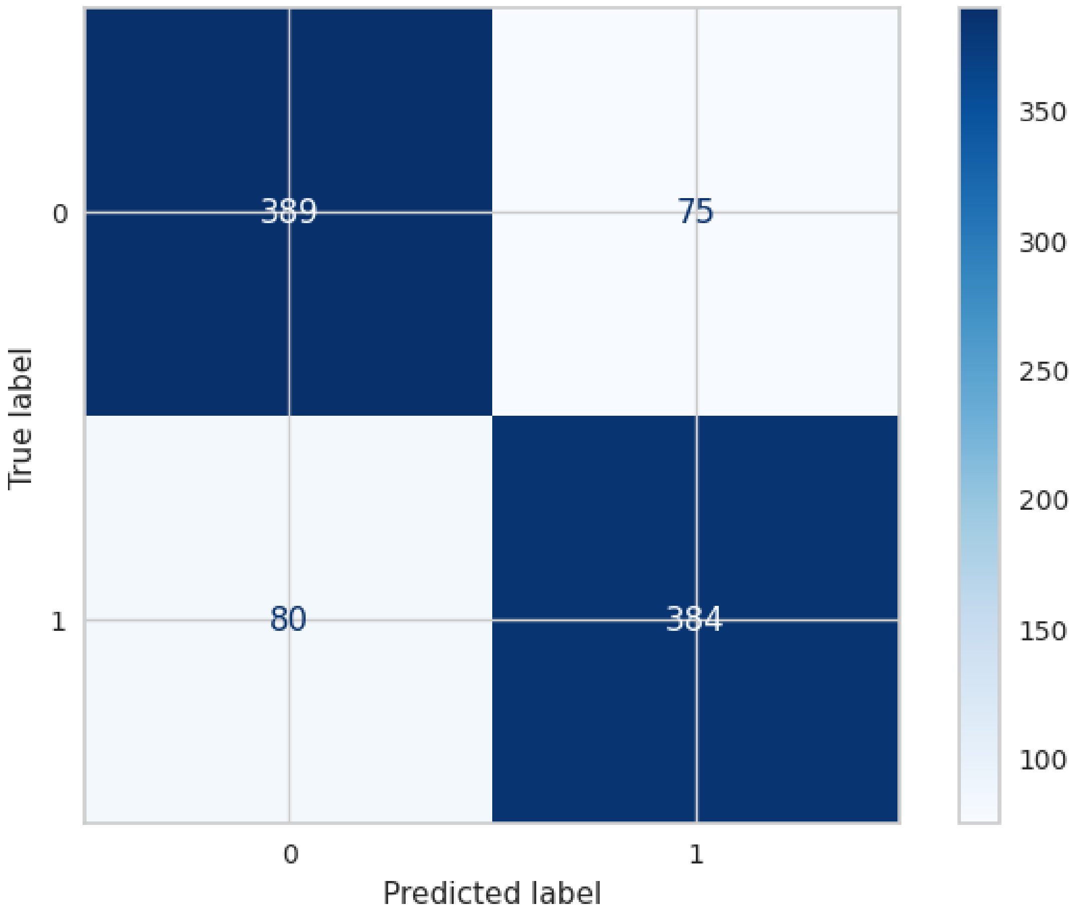
```
           1       0.84      0.83      0.84       464

    accuracy                           0.84       928
   macro avg       0.84      0.84      0.84       928
weighted avg       0.84      0.84      0.84       928

Confusion Matrix is :
 [[393  71]
 [ 77 387]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9906586826347306
Model Test Score is :  0.8329741379310345
F1 Score is :  0.8320693391115926
Recall Score is :  0.8275862068965517
Precision Score is :  0.8366013071895425
AUC Value  :  0.8329741379310344

Classification Report is :                precision    recall  f1-score
support

           0       0.83      0.84      0.83       464
           1       0.84      0.83      0.83       464

    accuracy                           0.83       928
   macro avg       0.83      0.83      0.83       928
weighted avg       0.83      0.83      0.83       928

Confusion Matrix is :
 [[389  75]
 [ 80 384]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8318965517241379
F1 Score is :  0.8300653594771242
Recall Score is :  0.8211206896551724
Precision Score is :  0.8392070484581498
AUC Value  :  0.8318965517241379

Classification Report is :                precision    recall  f1-score
support

           0       0.82      0.84      0.83       464
           1       0.84      0.82      0.83       464
```

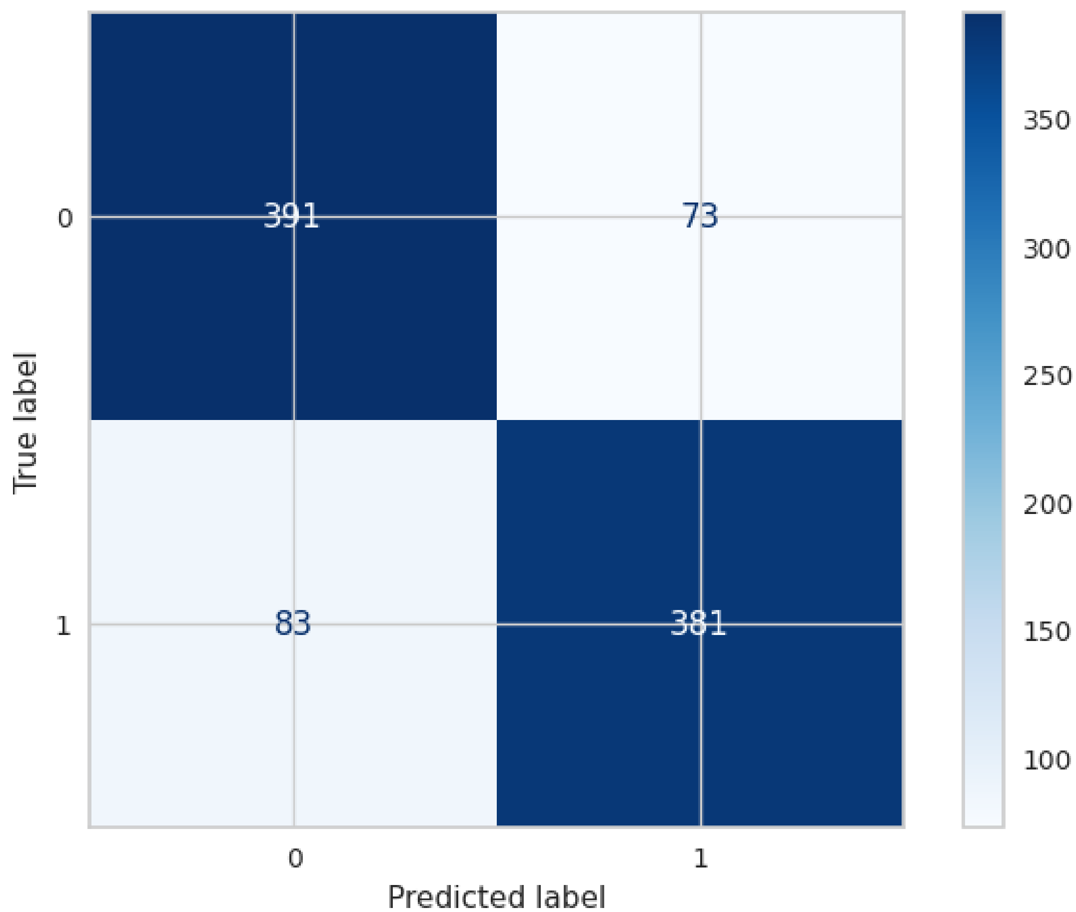
```
    accuracy                           0.83       928
   macro avg       0.83      0.83      0.83       928
weighted avg       0.83      0.83      0.83       928

Confusion Matrix is :
 [[391  73]
 [ 83 381]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8157327586206896
F1 Score is :  0.815135135135135
Recall Score is :  0.8125
Precision Score is :  0.8177874186550976
AUC Value  :  0.8157327586206896
Classification Report is :                precision    recall  f1-score
support

           0       0.81      0.82      0.82       464
           1       0.82      0.81      0.82       464

    accuracy                           0.82       928
   macro avg       0.82      0.82      0.82       928
weighted avg       0.82      0.82      0.82       928

Confusion Matrix is :
 [[380  84]
 [ 87 377]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.8351293103448276
F1 Score is :  0.8363636363636364
Recall Score is :  0.8426724137931034
Precision Score is :  0.8301486199575372
AUC Value  :  0.8351293103448276
Classification Report is :                precision    recall  f1-score
support

           0       0.84      0.83      0.83       464
           1       0.83      0.84      0.84       464
```

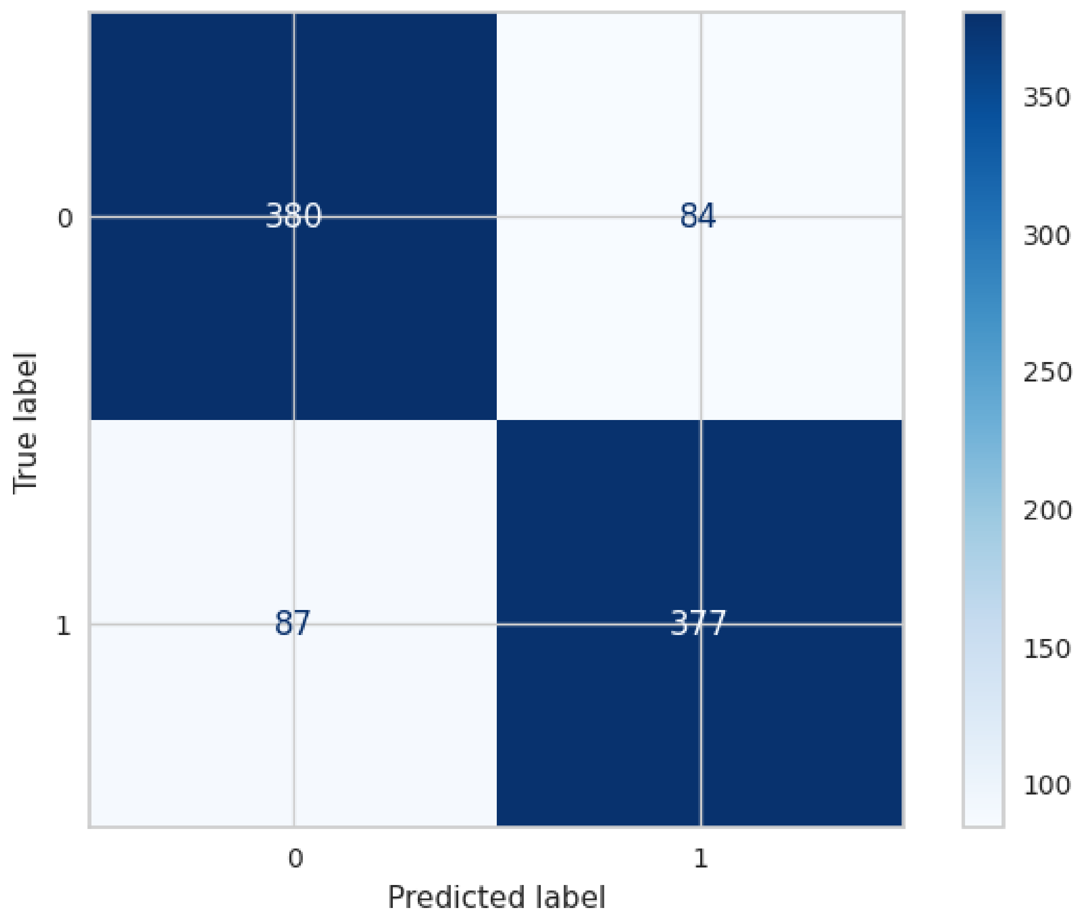
```
    accuracy                           0.84       928
   macro avg       0.84      0.84      0.84       928
weighted avg       0.84      0.84      0.84       928

Confusion Matrix is :
 [[384  80]
 [ 73 391]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.7780172413793104
F1 Score is :  0.7726269315673289
Recall Score is :  0.7543103448275862
Precision Score is :  0.7918552036199095
AUC Value  :  0.7780172413793104
Classification Report is :                precision    recall  f1-score
support

           0       0.77      0.80      0.78       464
           1       0.79      0.75      0.77       464

    accuracy                           0.78       928
   macro avg       0.78      0.78      0.78       928
weighted avg       0.78      0.78      0.78       928

Confusion Matrix is :
 [[372  92]
 [114 350]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9998802395209581
Model Test Score is :  0.802801724137931
F1 Score is :  0.7995618838992332
Recall Score is :  0.7866379310344828
Precision Score is :  0.8129175946547884
AUC Value  :  0.802801724137931
Classification Report is :                precision    recall  f1-score
support

           0       0.79      0.82      0.81       464
           1       0.81      0.79      0.80       464

    accuracy                           0.80       928
```

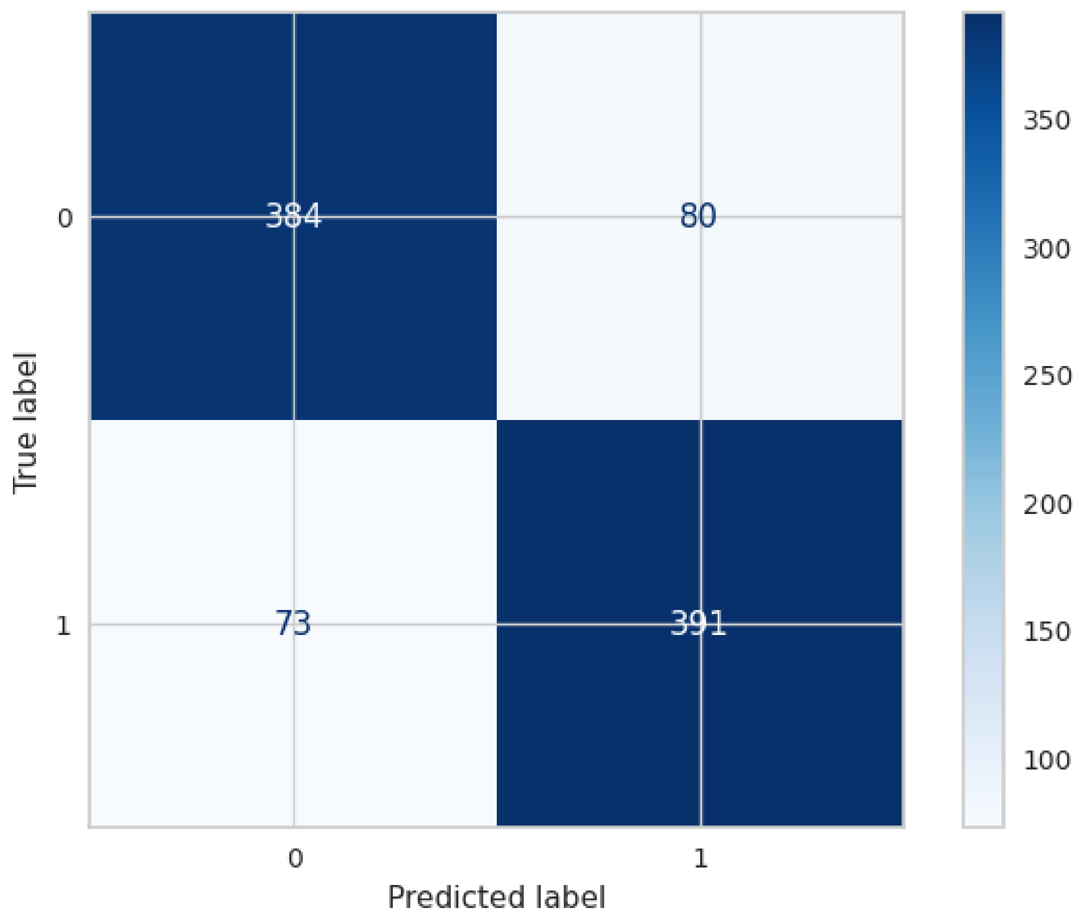
```
   macro avg       0.80      0.80      0.80       928
weighted avg       0.80      0.80      0.80       928

Confusion Matrix is :
 [[380  84]
 [ 99 365]]
```

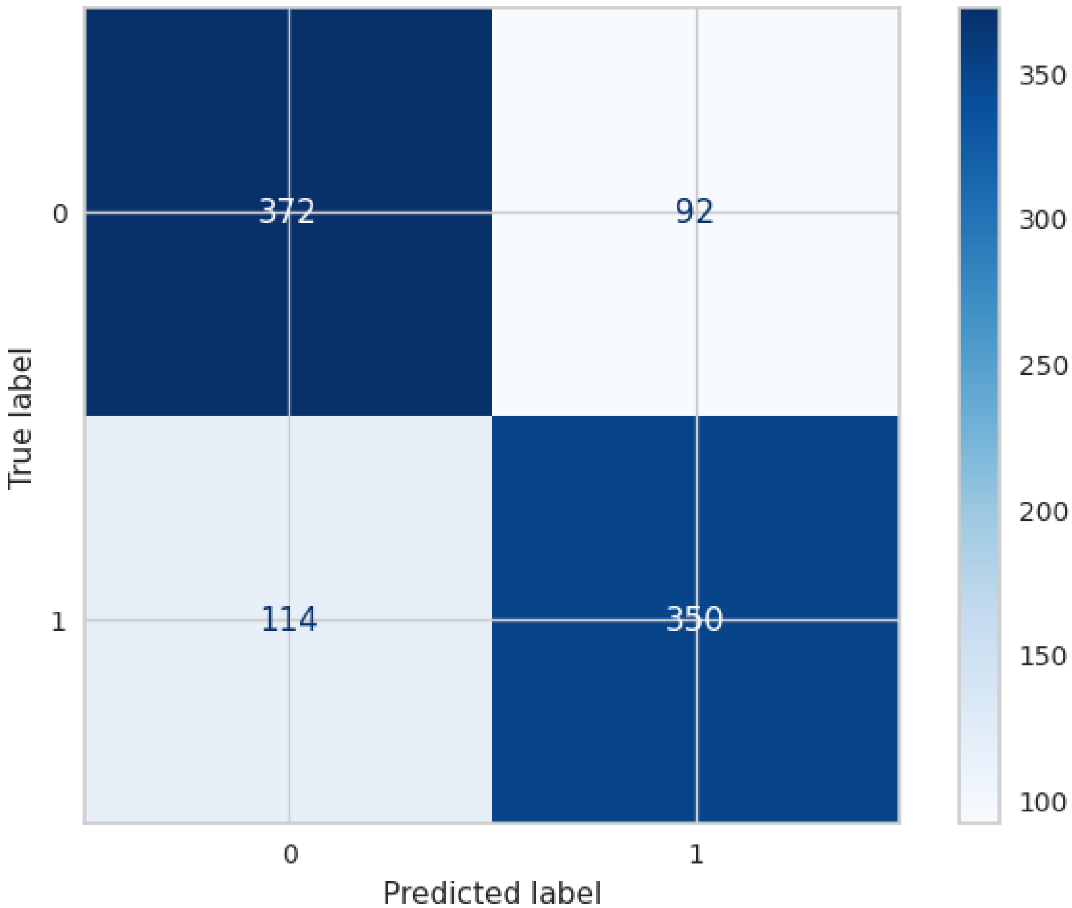


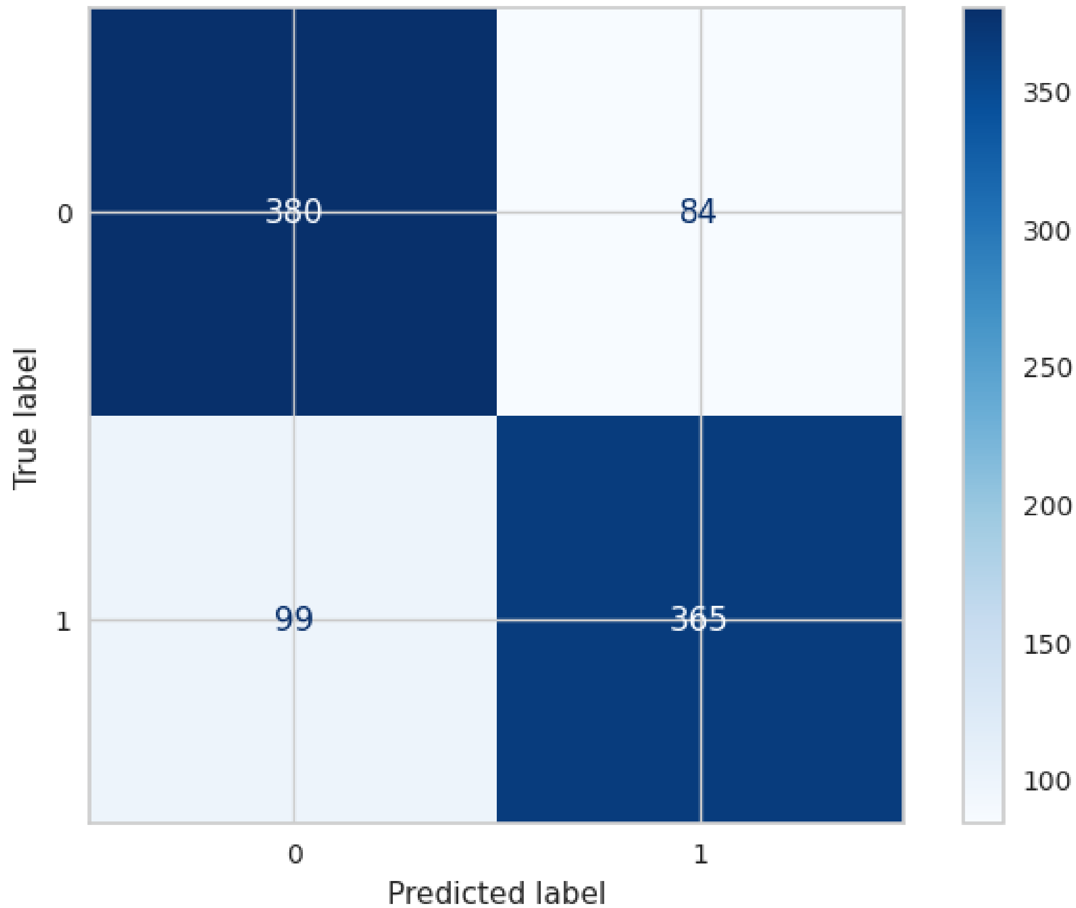

389
75
80
384
0
1
0
1
True label
Predicted label
350
300
250
200
150
100


391
73
83
381
0
1
True label
0
1
Predicted label
350
300
250
200
150
100


380
84
87
377
0
1
0
1
True label
Predicted label
350
300
250
200
150
100


384
80
73
391
0
1
True label
0
1
Predicted label
350
300
250
200
150
100


372
92
114
350
0
1
True label
0
1
Predicted label
350
300
250
200
150
100



[241]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Decision Under','Decision Under With Feature','Decision Under␣
↪Scaling','Decision Under With Normalize','Decision Under With PCA'
                ,'Decision Under With PCA and Scaling',
                'Decision Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[241]:
```
                                       Train Accuracy  Test Accuracy  \
Models
Decision Under                               0.999880       0.840517
Decision Under With Feature                  0.990659       0.832974
Decision Under Scaling                       0.999880       0.831897
Decision Under With Normalize                0.999880       0.815733
Decision Under With PCA                      0.999880       0.835129
Decision Under With PCA and Scaling          0.999880       0.778017
Decision Under With PCA and Normalize        0.999880       0.802802
```

```
                                         Test F1  Test Recall  Test Precision  \
Models
Decision Under                          0.839479     0.834052        0.844978
Decision Under With Feature             0.832069     0.827586        0.836601
Decision Under Scaling                  0.830065     0.821121        0.839207
Decision Under With Normalize           0.815135     0.812500        0.817787
Decision Under With PCA                 0.836364     0.842672        0.830149
Decision Under With PCA and Scaling     0.772627     0.754310        0.791855
Decision Under With PCA and Normalize   0.799562     0.786638        0.812918

                                             AUC
Models
Decision Under                          0.840517
Decision Under With Feature             0.832974
Decision Under Scaling                  0.831897
Decision Under With Normalize           0.815733
Decision Under With PCA                 0.835129
Decision Under With PCA and Scaling     0.778017
Decision Under With PCA and Normalize   0.802802
```

[242]:
```
models_draw(df)
```

KNeighborsClassifier

[243]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[244]:
```
Search(KNeighborsClassifier(n_neighbors=3),{'n_neighbors':
 ↪[3,5,7,9,11]},X_train,y_train)
```

[244]:
```
KNeighborsClassifier(n_neighbors=11)
```

[245]:
```
cross_validation(KNeighborsClassifier(n_neighbors=11),X_train,y_train)
```

```
Train Score Value :  [0.92018621 0.91840108 0.91998651 0.9200877  0.91860347]
Mean 0.9194529950157511
Test Score Value :  [0.90326498 0.90622048 0.90352179 0.90500607 0.90999865]
Mean 0.905602394684234
```

[246]:
```
Values =␣
 ↪Models(KNeighborsClassifier(n_neighbors=11),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

Model Train Score is :  0.9197970639032815
```

```
Model Test Score is :  0.9108790675084992
F1 Score is :  0.5625744934445768
Recall Score is :  0.5086206896551724
Precision Score is :  0.6293333333333333
AUC Value  :  0.7352900930487138
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.96      0.95      3654
           1       0.63      0.51      0.56       464

    accuracy                           0.91      4118
   macro avg       0.78      0.74      0.76      4118
weighted avg       0.90      0.91      0.91      4118

Confusion Matrix is :
 [[3515  139]
 [ 228  236]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9168015975820379
Model Test Score is :  0.9065080135988344
F1 Score is :  0.5299145299145299
Recall Score is :  0.4676724137931034
Precision Score is :  0.6112676056338028
AUC Value  :  0.7149527914614121
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.96      0.95      3654
           1       0.61      0.47      0.53       464

    accuracy                           0.91      4118
   macro avg       0.77      0.71      0.74      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3516  138]
 [ 247  217]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9145347582037997
Model Test Score is :  0.9023797960174842
```

```
F1 Score is :  0.41739130434782606
Recall Score is :  0.3103448275862069
Precision Score is :  0.6371681415929203
AUC Value  :  0.643951833607006
```

```
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
           1       0.64      0.31      0.42       464

    accuracy                           0.90      4118
   macro avg       0.78      0.64      0.68      4118
weighted avg       0.89      0.90      0.89      4118
```

```
Confusion Matrix is :
 [[3572   82]
 [ 320  144]]
```

```
 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.918825561312608
Model Test Score is :  0.9052938319572608
F1 Score is :  0.524390243902439
Recall Score is :  0.46336206896551724
Precision Score is :  0.6039325842696629
AUC Value  :  0.7123871100164204
```

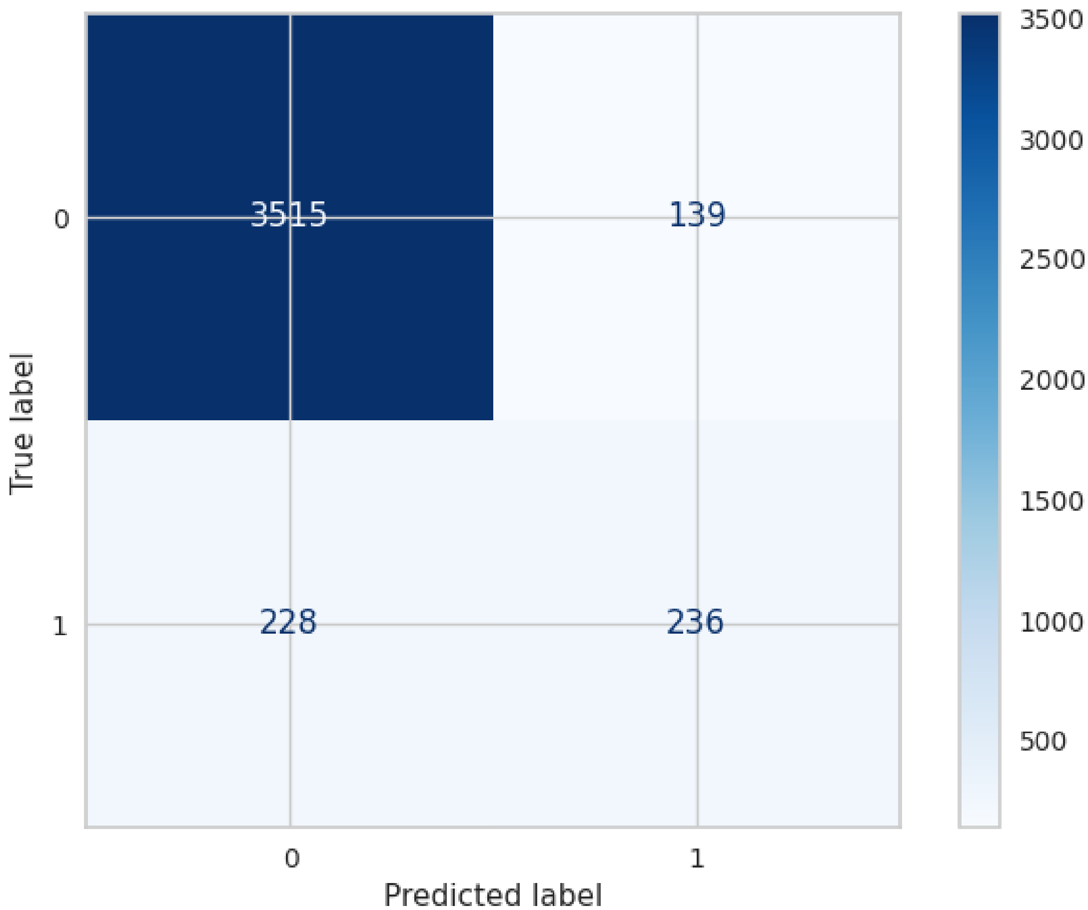
```
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.96      0.95      3654
           1       0.60      0.46      0.52       464

    accuracy                           0.91      4118
   macro avg       0.77      0.71      0.74      4118
weighted avg       0.90      0.91      0.90      4118
```

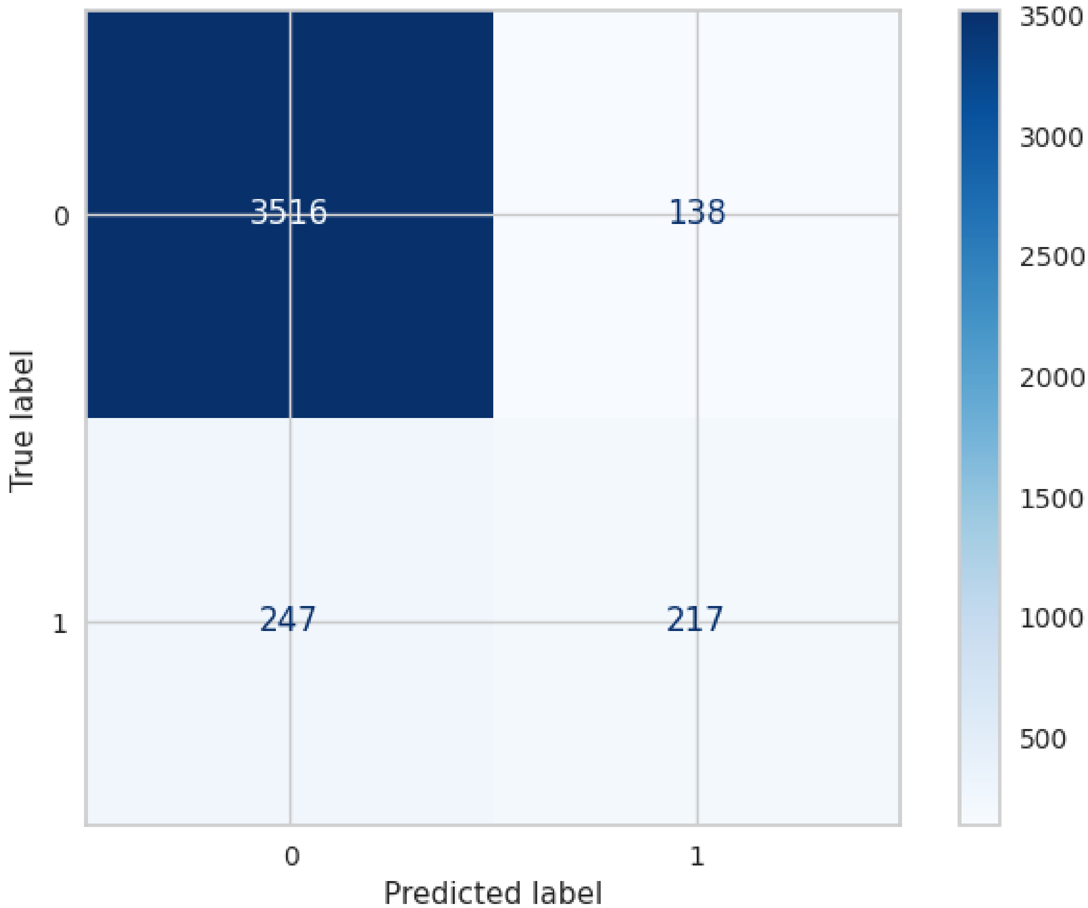
```
Confusion Matrix is :
 [[3513  141]
 [ 249  215]]
```

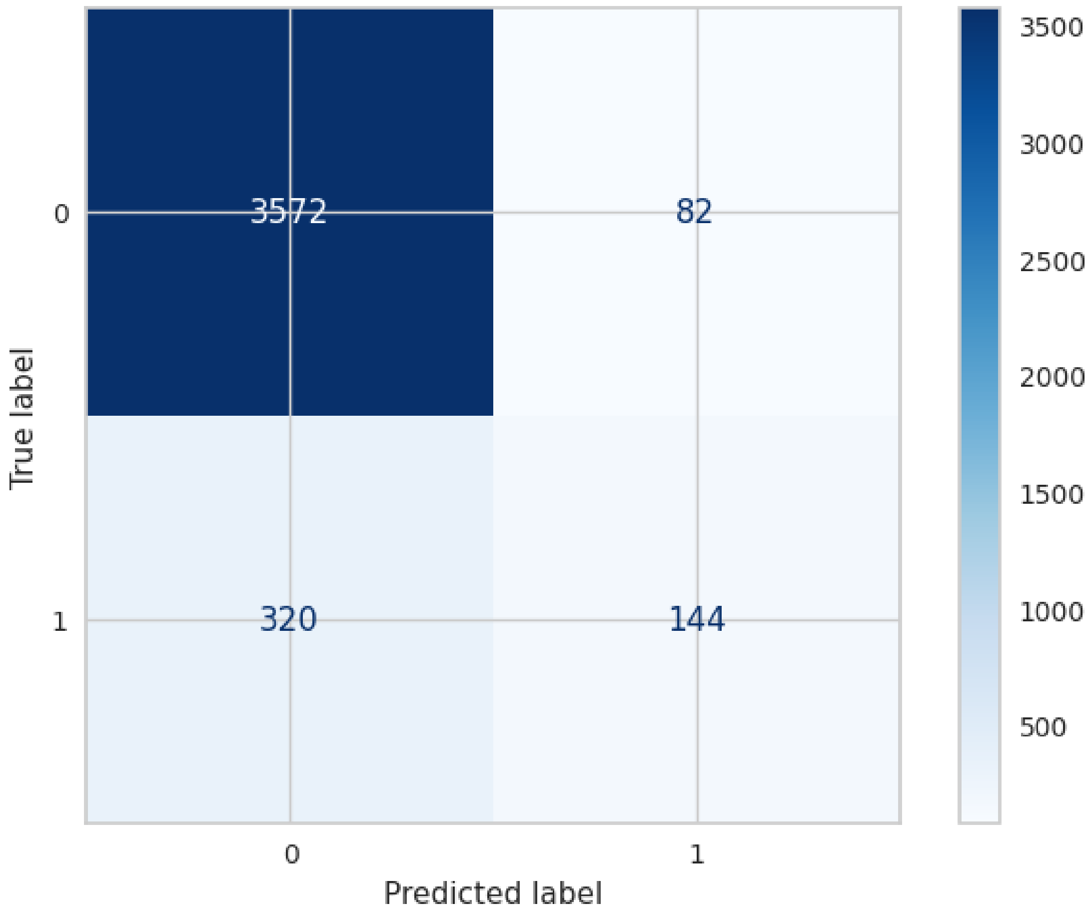
```
 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9199319948186528
Model Test Score is :  0.9103933948518699
F1 Score is :  0.5591397849462365
```

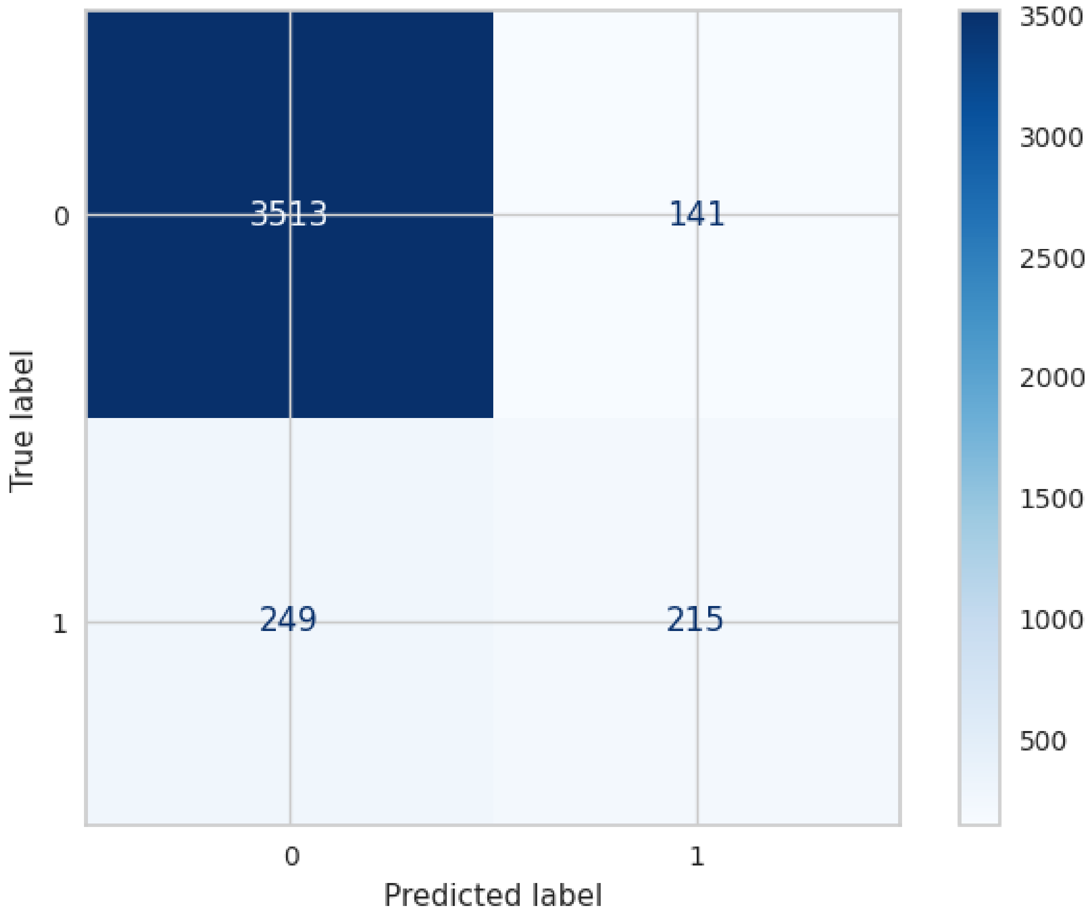
```
Recall Score is :  0.5043103448275862
Precision Score is :  0.6273458445040214
AUC Value  :  0.7331349206349207

Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.96      0.95      3654
           1       0.63      0.50      0.56       464

    accuracy                           0.91      4118
   macro avg       0.78      0.73      0.75      4118
weighted avg       0.90      0.91      0.91      4118

Confusion Matrix is :
 [[3515  139]
 [ 230  234]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9145347582037997
Model Test Score is :  0.9023797960174842
F1 Score is :  0.41739130434782606
Recall Score is :  0.3103448275862069
Precision Score is :  0.6371681415929203
AUC Value  :  0.643951833607006

Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
           1       0.64      0.31      0.42       464

    accuracy                           0.90      4118
   macro avg       0.78      0.64      0.68      4118
weighted avg       0.89      0.90      0.89      4118

Confusion Matrix is :
 [[3572   82]
 [ 320  144]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.918825561312608
Model Test Score is :  0.9052938319572608
F1 Score is :  0.524390243902439
Recall Score is :  0.46336206896551724
```

```
Precision Score is :  0.6039325842696629
AUC Value  :  0.7123871100164204
Classification Report is :                 precision    recall  f1-score
support

           0       0.93      0.96      0.95      3654
           1       0.60      0.46      0.52       464

    accuracy                           0.91      4118
   macro avg       0.77      0.71      0.74      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3513  141]
 [ 249  215]]
```

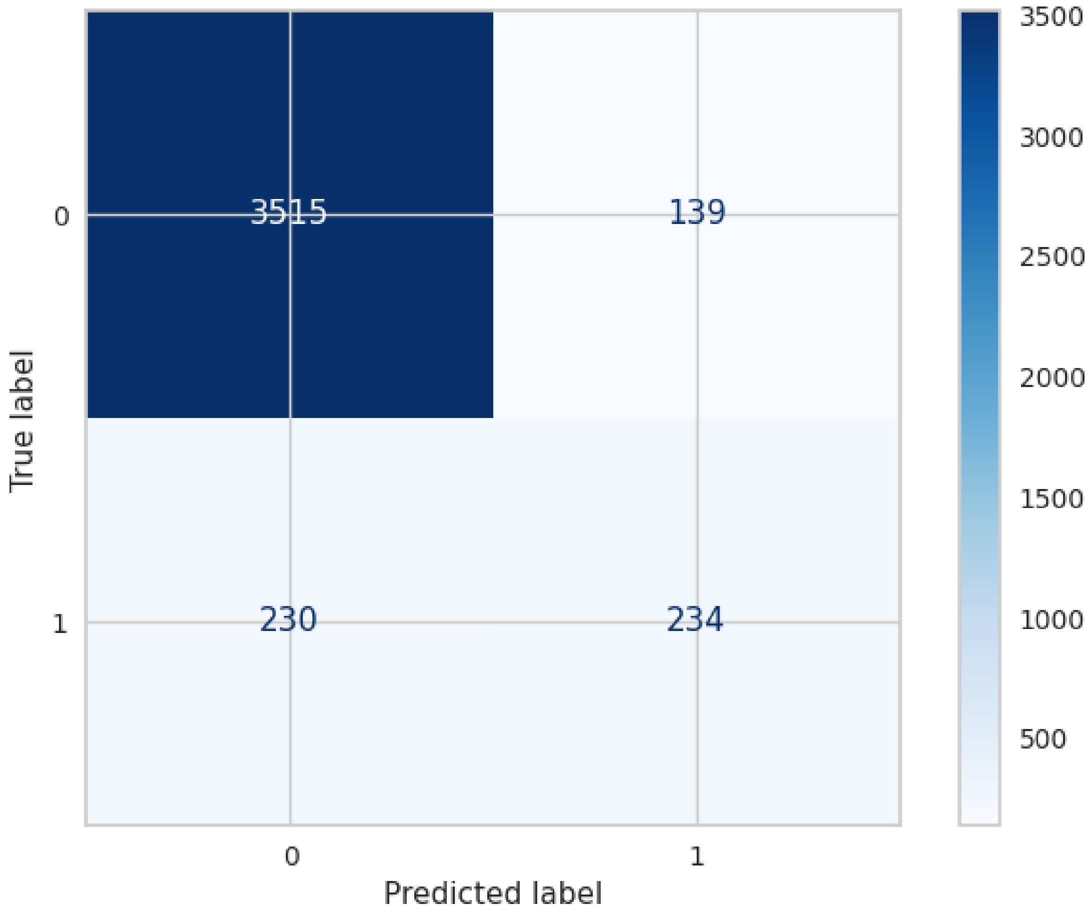


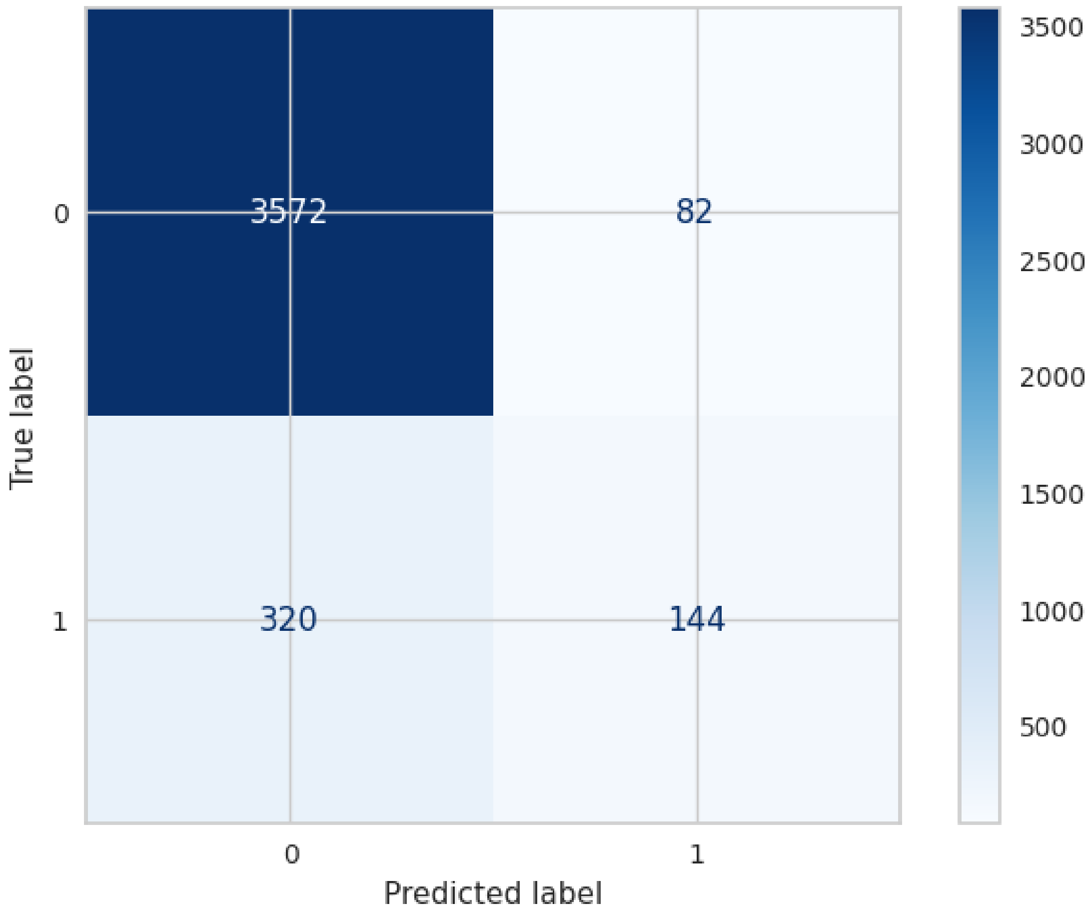

3516
138
247
217
True label
0
1
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3572
82
320
144
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500

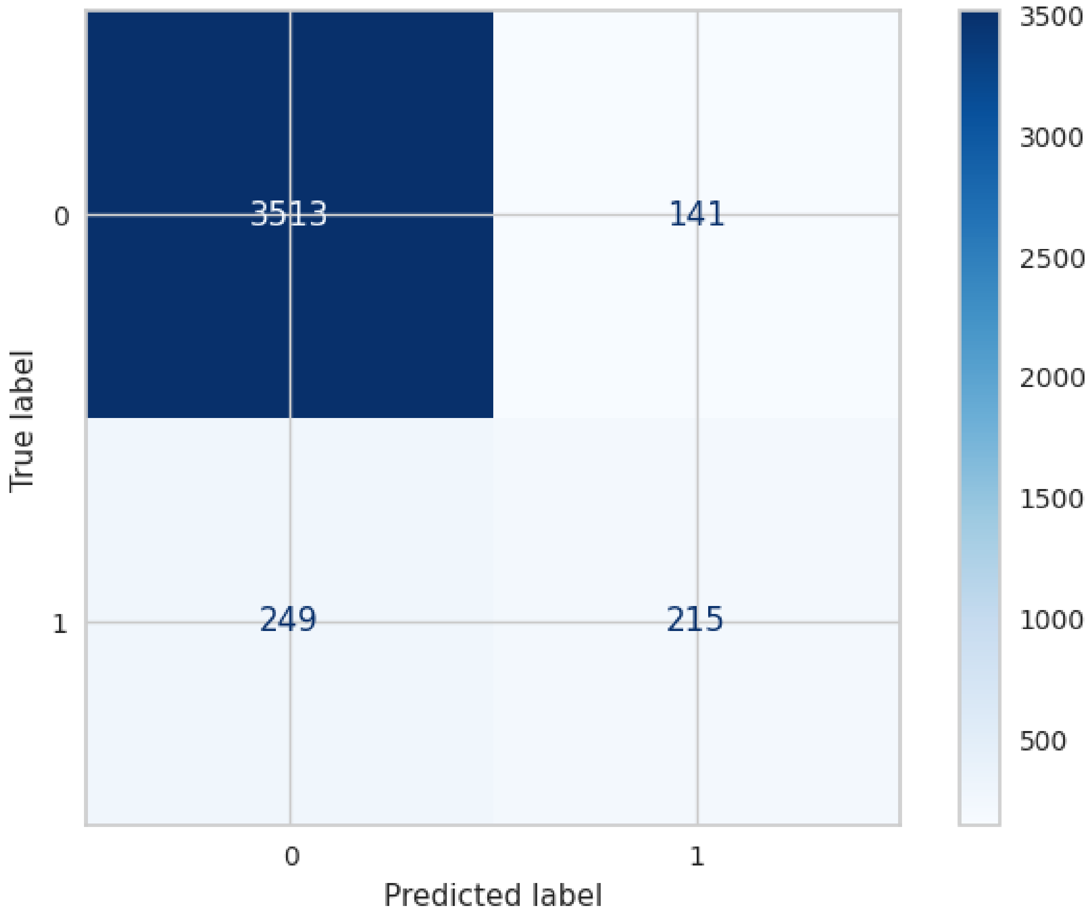


3513
141
249
215
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3515
139
230
234
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500



3572
82
320
144
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[247]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['KNN','KNN With Feature','KNN Scaling','KNN With␣
↪Normalize','KNN With PCA'
                 ,'KNN With PCA and Scaling',
                 'KNN With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[247]:
```
                            Train Accuracy  Test Accuracy   Test F1  \
Models
KNN                               0.919797       0.910879  0.562574
KNN With Feature                  0.916802       0.906508  0.529915
KNN Scaling                       0.914535       0.902380  0.417391
KNN With Normalize                0.918826       0.905294  0.524390
KNN With PCA                      0.919932       0.910393  0.559140
KNN With PCA and Scaling          0.914535       0.902380  0.417391
KNN With PCA and Normalize        0.918826       0.905294  0.524390
```

```
                            Test Recall  Test Precision       AUC
Models
KNN                            0.508621        0.629333  0.735290
KNN With Feature               0.467672        0.611268  0.714953
KNN Scaling                    0.310345        0.637168  0.643952
KNN With Normalize             0.463362        0.603933  0.712387
KNN With PCA                   0.504310        0.627346  0.733135
KNN With PCA and Scaling       0.310345        0.637168  0.643952
KNN With PCA and Normalize     0.463362        0.603933  0.712387
```

[248]:
```
models_draw(df)
```

RandomOverSampler

[249]:
```
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[250]:
```
Search(KNeighborsClassifier(n_neighbors=3),{'n_neighbors':
 ↪[3,5,7,9,11]},X_train,y_train)
```

[250]:
```
KNeighborsClassifier(n_neighbors=3)
```

[251]:
```
cross_validation(KNeighborsClassifier(n_neighbors=3),X_train,y_train)
```

```
Train Score Value :  [0.96306786 0.96342901 0.96350504 0.96392389 0.96242231]
Mean 0.9632696204288106
Test Score Value :  [0.94176234 0.93894929 0.94062191 0.94023723 0.93993309]
Mean 0.9403007704754245
```

[252]:
```
Values =␣
 ↪Models(KNeighborsClassifier(n_neighbors=3),X_train,y_train,X_test,y_test)
```

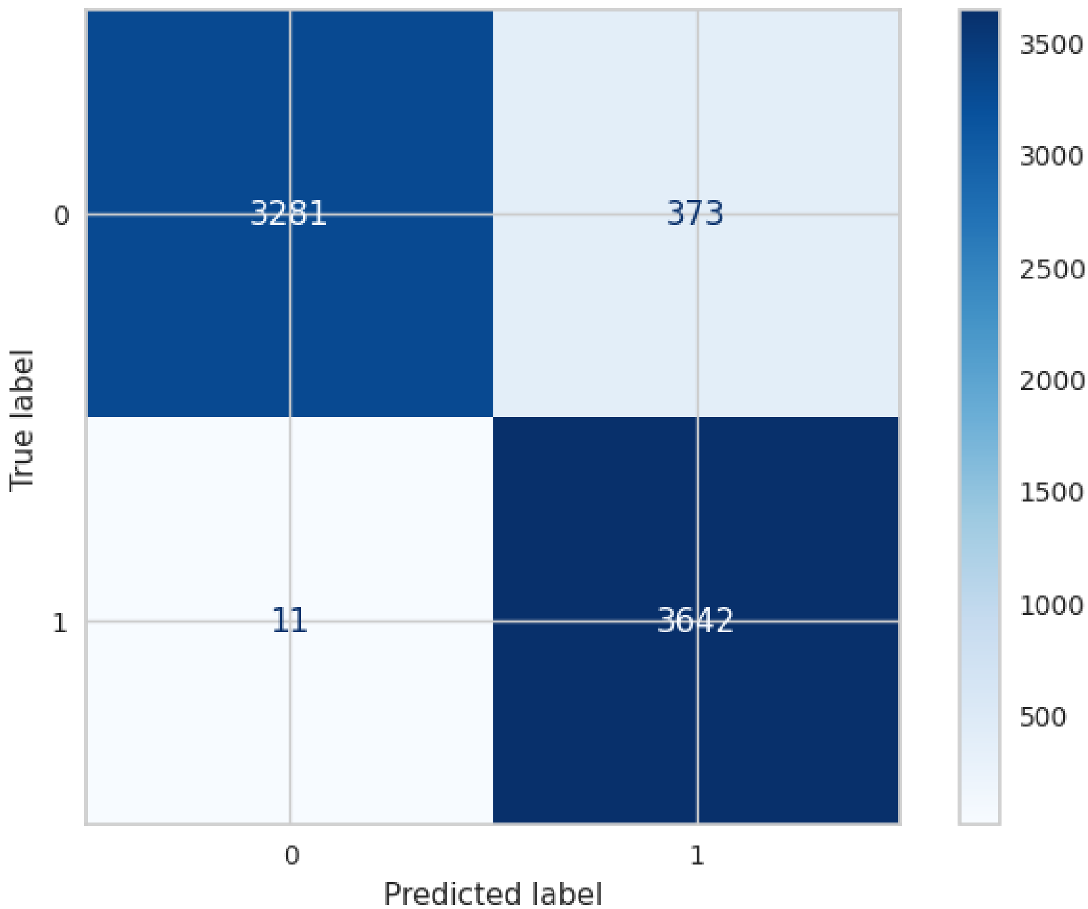
```
Apply Model With Normal Data :

Model Train Score is :  0.9675805544150966
Model Test Score is :  0.9474476529355412
F1 Score is :  0.9499217527386542
Recall Score is :  0.9969887763482069
Precision Score is :  0.9070983810709838
AUC Value  :  0.9474544319617335

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.90      0.94      3654
```

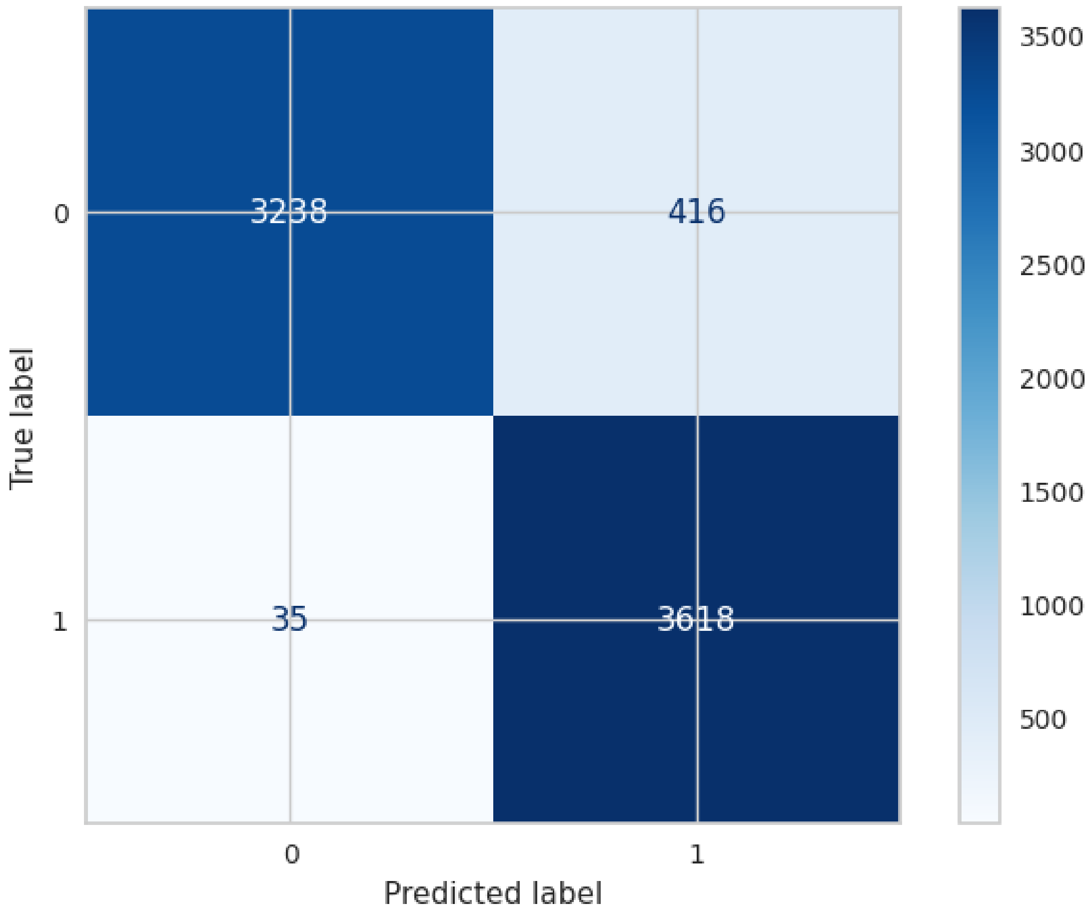
```
           1       0.91      1.00      0.95      3653

    accuracy                           0.95      7307
   macro avg       0.95      0.95      0.95      7307
weighted avg       0.95      0.95      0.95      7307

Confusion Matrix is :
 [[3281  373]
 [  11 3642]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9594604868999286
Model Test Score is :  0.938278363213357
F1 Score is :  0.9413295173669831
Recall Score is :  0.9904188338352039
Precision Score is :  0.8968765493306892
AUC Value  :  0.9382854979247175

Classification Report is :                precision    recall  f1-score
support

           0       0.99      0.89      0.93      3654
           1       0.90      0.99      0.94      3653

    accuracy                           0.94      7307
   macro avg       0.94      0.94      0.94      7307
weighted avg       0.94      0.94      0.94      7307

Confusion Matrix is :
 [[3238  416]
 [  35 3618]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9662728281860621
Model Test Score is :  0.9469002326536198
F1 Score is :  0.9494264859228363
Recall Score is :  0.9969887763482069
Precision Score is :  0.9061955710375715
AUC Value  :  0.946907086586802

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.90      0.94      3654
           1       0.91      1.00      0.95      3653
```

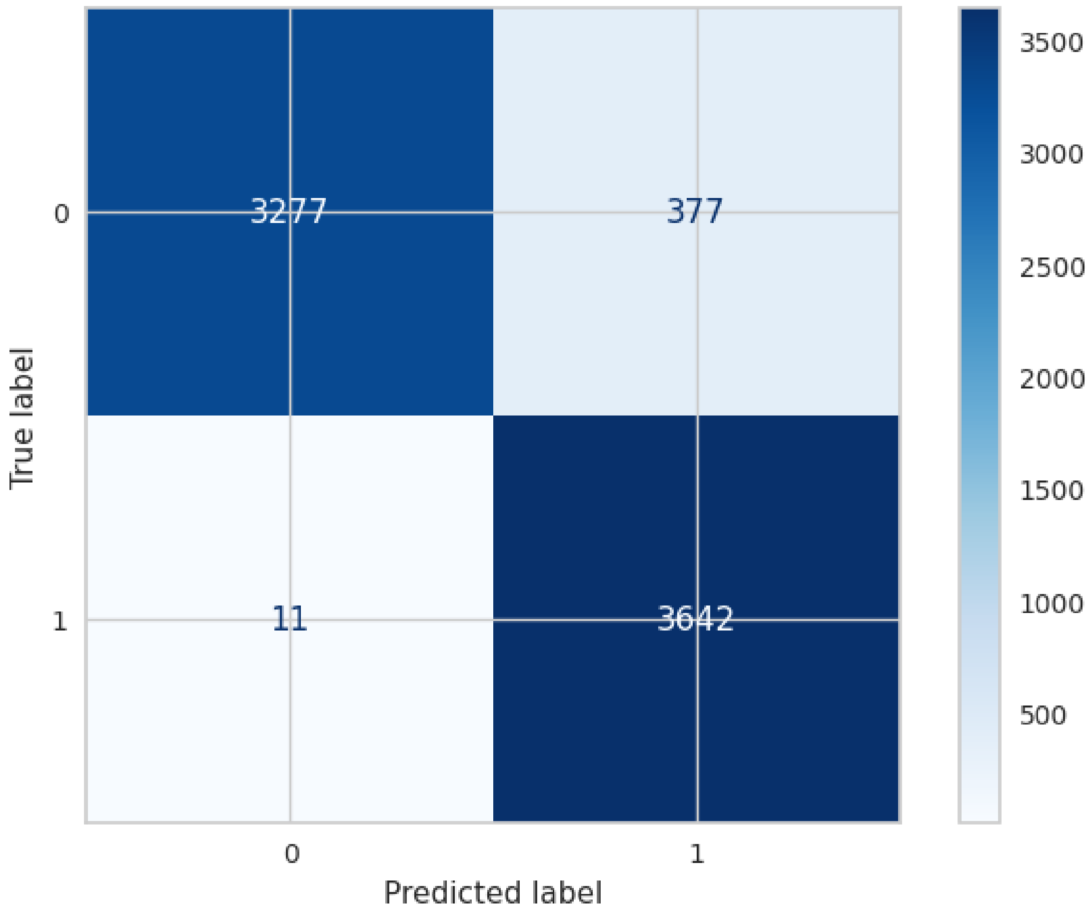
```
    accuracy                           0.95      7307
   macro avg       0.95      0.95      0.95      7307
weighted avg       0.95      0.95      0.95      7307

Confusion Matrix is :
 [[3277  377]
 [  11 3642]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9663640648997156
Model Test Score is :  0.9473107978650609
F1 Score is :  0.9498240583865502
Recall Score is :  0.9975362715576239
Precision Score is :  0.9064676616915422
AUC Value  :  0.9473176705352434
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.90      0.94      3654
           1       0.91      1.00      0.95      3653

    accuracy                           0.95      7307
   macro avg       0.95      0.95      0.95      7307
weighted avg       0.95      0.95      0.95      7307

Confusion Matrix is :
 [[3278  376]
 [   9 3644]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9680215318644222
Model Test Score is :  0.9482687833584235
F1 Score is :  0.9506656225528581
Recall Score is :  0.9969887763482069
Precision Score is :  0.9084559740583686
AUC Value  :  0.9482754500241309
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.90      0.95      3654
           1       0.91      1.00      0.95      3653
```

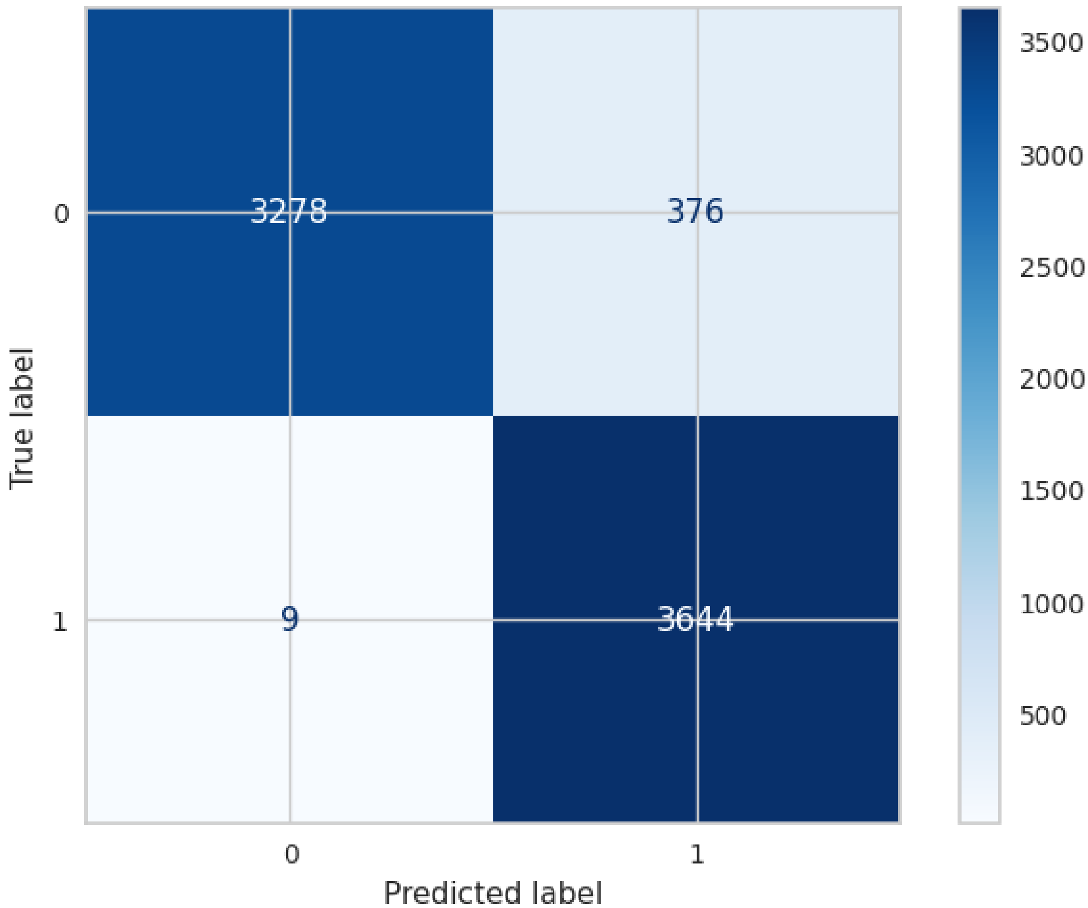
```
    accuracy                           0.95      7307
   macro avg       0.95      0.95      0.95      7307
weighted avg       0.95      0.95      0.95      7307

Confusion Matrix is :
 [[3287  367]
 [  11 3642]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9662728281860621
Model Test Score is :  0.9470370877241002
F1 Score is :  0.9495502542041455
Recall Score is :  0.9969887763482069
Precision Score is :  0.9064211050273768
AUC Value  :  0.9470439229305349

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.90      0.94      3654
           1       0.91      1.00      0.95      3653

    accuracy                           0.95      7307
   macro avg       0.95      0.95      0.95      7307
weighted avg       0.95      0.95      0.95      7307

Confusion Matrix is :
 [[3278  376]
 [  11 3642]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9663488587807734
Model Test Score is :  0.9474476529355412
F1 Score is :  0.9499478623566215
Recall Score is :  0.9975362715576239
Precision Score is :  0.906693207265489
AUC Value  :  0.9474545068789761

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.90      0.94      3654
           1       0.91      1.00      0.95      3653

    accuracy                           0.95      7307
```

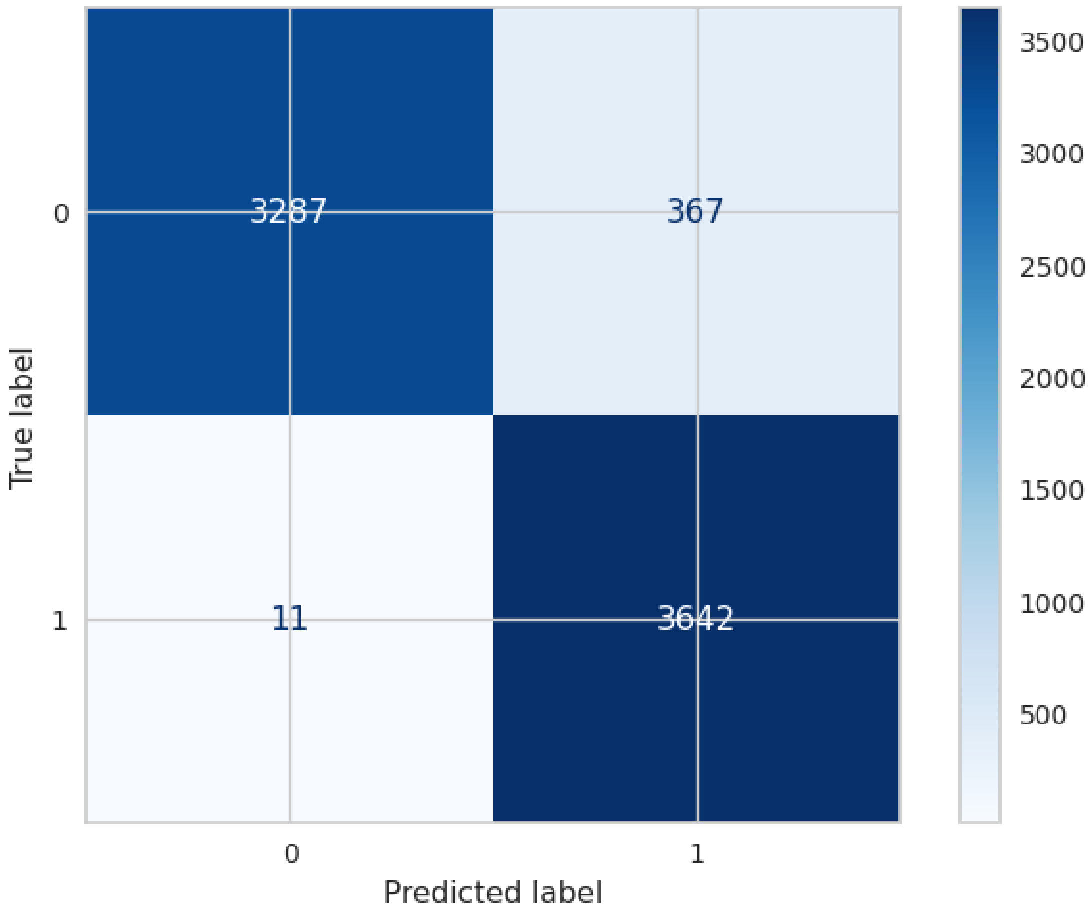
```
   macro avg       0.95      0.95      0.95      7307
weighted avg       0.95      0.95      0.95      7307

Confusion Matrix is :
 [[3279  375]
 [   9 3644]]
```

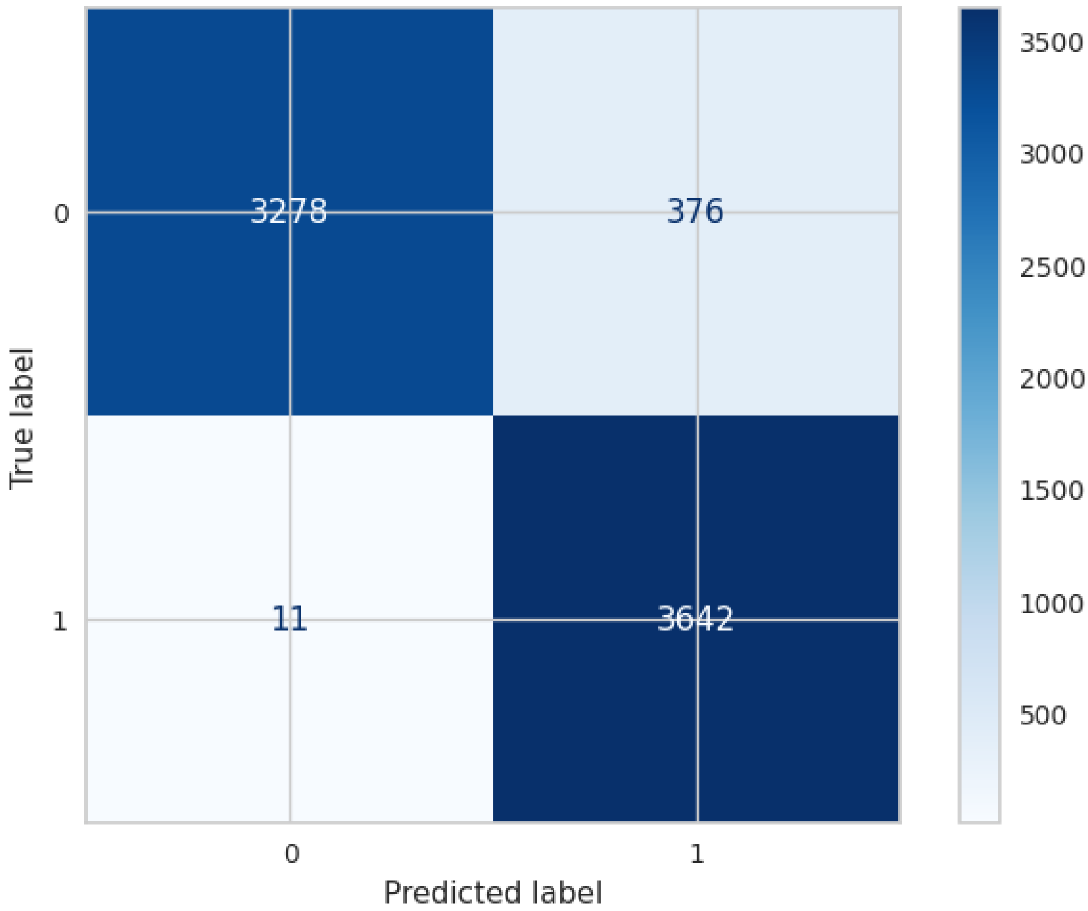



3238
416
35
3618
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500

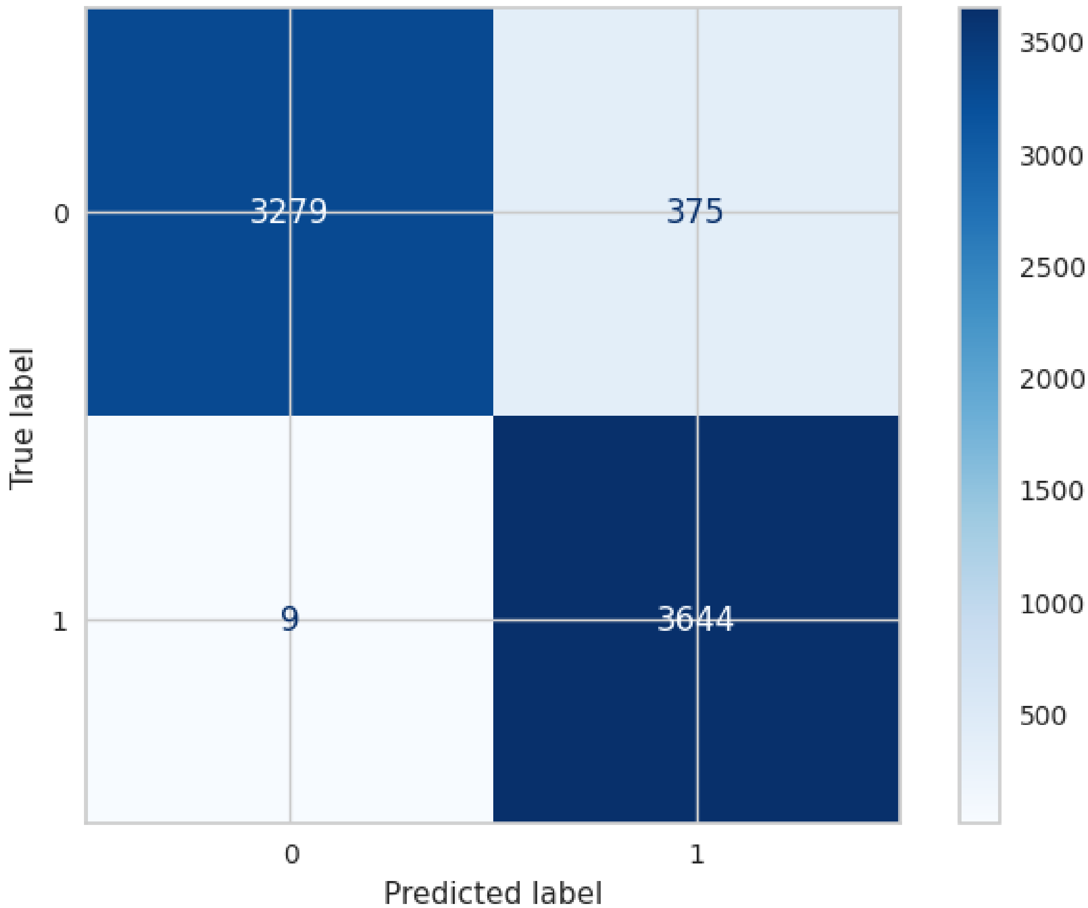

3277
377
11
3642
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3278
376
9
3644
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3287
367
11
3642
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



3278
376
11
3642
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[253]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['KNN Over','KNN Over With Feature','KNN Over Scaling','KNN Over␣
↪With Normalize','KNN Over With PCA'
                ,'KNN Over With PCA and Scaling',
                'KNN Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[253]:
```
                                 Train Accuracy  Test Accuracy   Test F1  \
Models
KNN Over                               0.967581       0.947448  0.949922
KNN Over With Feature                  0.959460       0.938278  0.941330
KNN Over Scaling                       0.966273       0.946900  0.949426
KNN Over With Normalize                0.966364       0.947311  0.949824
KNN Over With PCA                      0.968022       0.948269  0.950666
KNN Over With PCA and Scaling          0.966273       0.947037  0.949550
KNN Over With PCA and Normalize        0.966349       0.947448  0.949948
```

```
                                   Test Recall  Test Precision       AUC
Models
KNN Over                              0.996989        0.907098  0.947454
KNN Over With Feature                 0.990419        0.896877  0.938285
KNN Over Scaling                      0.996989        0.906196  0.946907
KNN Over With Normalize               0.997536        0.906468  0.947318
KNN Over With PCA                     0.996989        0.908456  0.948275
KNN Over With PCA and Scaling         0.996989        0.906421  0.947044
KNN Over With PCA and Normalize       0.997536        0.906693  0.947455
```

[254]:
```
models_draw(df)
```

RandomUnderSampler

[255]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[256]:
```
Search(KNeighborsClassifier(n_neighbors=3),{'n_neighbors':
 ↪[3,5,7,9,11]},X_train,y_train)
```

[256]:
```
KNeighborsClassifier(n_neighbors=9)
```

[257]:
```
cross_validation(KNeighborsClassifier(n_neighbors=11),X_train,y_train)
```

```
Train Score Value :  [0.87709581 0.87799401 0.88488024 0.88023952 0.87739521]
Mean 0.8795209580838323
Test Score Value :  [0.86467066 0.86107784 0.83952096 0.8742515  0.86826347]
Mean 0.8615568862275449
```

[258]:
```
Values =␣
 ↪Models(KNeighborsClassifier(n_neighbors=11),X_train,y_train,X_test,y_test)
```

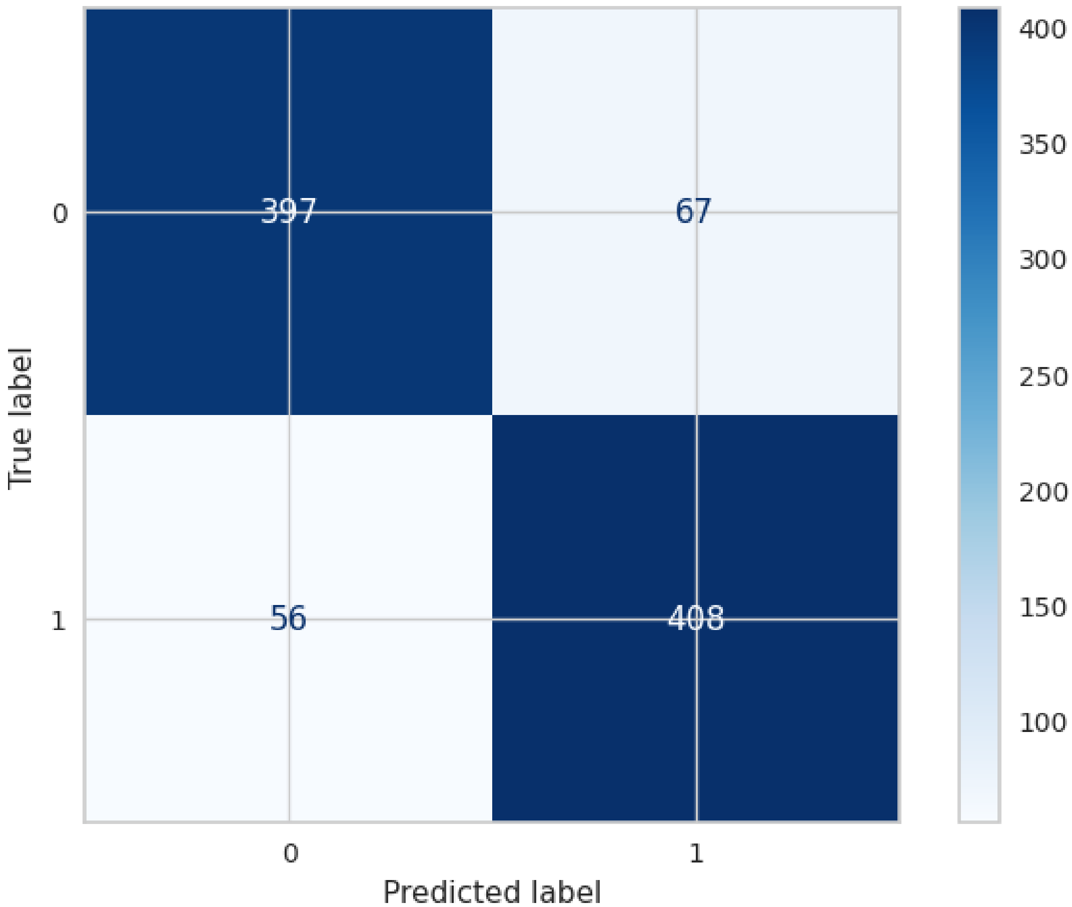
```
Apply Model With Normal Data :

Model Train Score is :  0.8831137724550898
Model Test Score is :  0.8674568965517241
F1 Score is :  0.8690095846645368
Recall Score is :  0.8793103448275862
Precision Score is :  0.8589473684210527
AUC Value  :  0.8674568965517242

Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.86      0.87       464
```

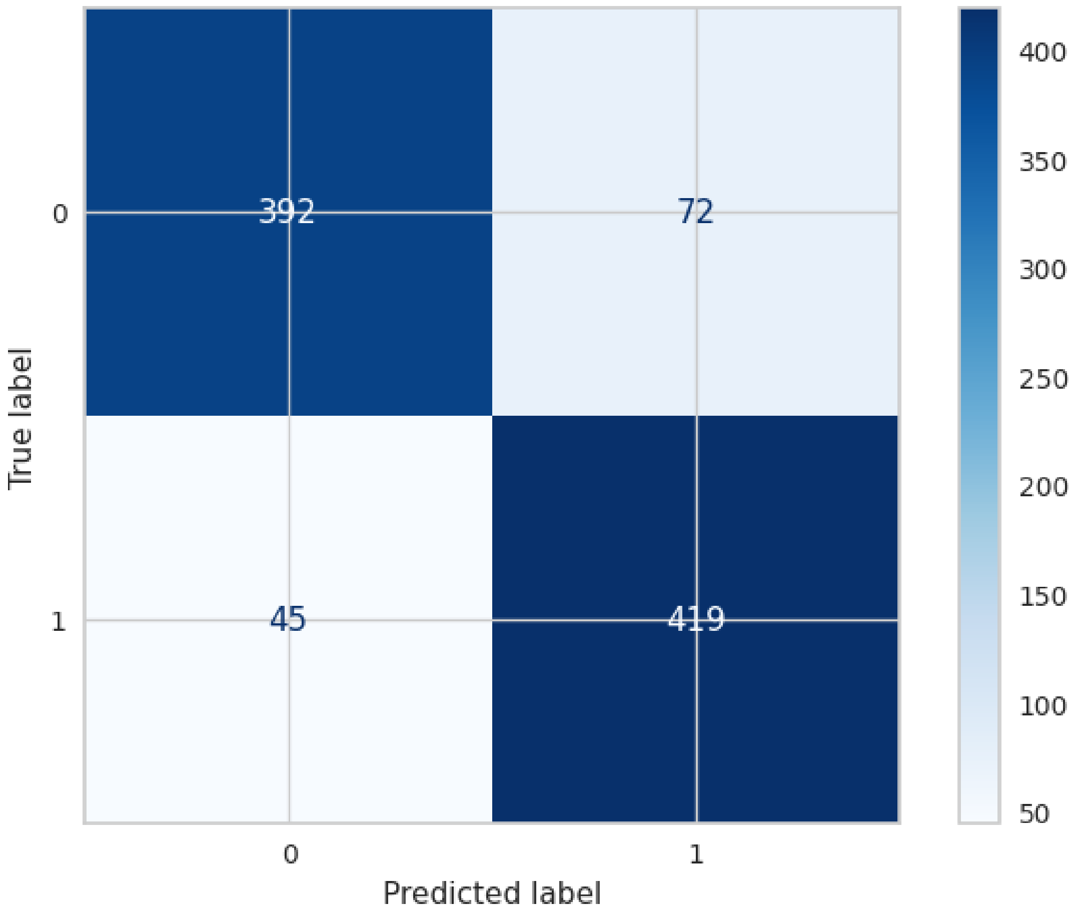
```
           1       0.86      0.88      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[397  67]
 [ 56 408]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8815568862275449
Model Test Score is :  0.8739224137931034
F1 Score is :  0.8774869109947644
Recall Score is :  0.9030172413793104
Precision Score is :  0.8533604887983707
AUC Value  :  0.8739224137931035

Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.84      0.87       464
           1       0.85      0.90      0.88       464

    accuracy                           0.87       928
   macro avg       0.88      0.87      0.87       928
weighted avg       0.88      0.87      0.87       928

Confusion Matrix is :
 [[392  72]
 [ 45 419]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8677844311377245
Model Test Score is :  0.8556034482758621
F1 Score is :  0.858050847457627
Recall Score is :  0.8728448275862069
Precision Score is :  0.84375
AUC Value  :  0.855603448275862

Classification Report is :                precision    recall  f1-score
support

           0       0.87      0.84      0.85       464
           1       0.84      0.87      0.86       464
```

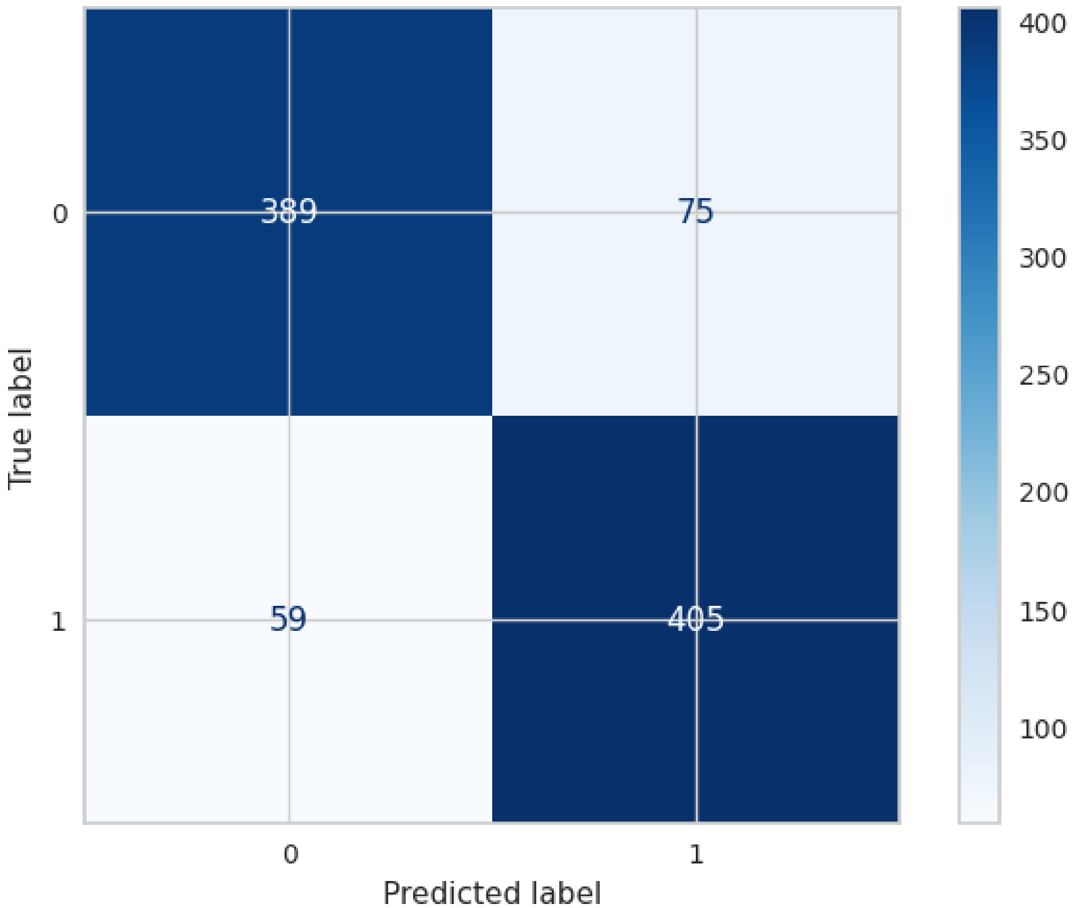
```
    accuracy                           0.86       928
   macro avg       0.86      0.86      0.86       928
weighted avg       0.86      0.86      0.86       928

Confusion Matrix is :
 [[389  75]
 [ 59 405]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8774850299401198
Model Test Score is :  0.8620689655172413
F1 Score is :  0.8632478632478633
Recall Score is :  0.8706896551724138
Precision Score is :  0.8559322033898306
AUC Value  :  0.8620689655172413

Classification Report is :                precision    recall  f1-score
support

           0       0.87      0.85      0.86       464
           1       0.86      0.87      0.86       464

    accuracy                           0.86       928
   macro avg       0.86      0.86      0.86       928
weighted avg       0.86      0.86      0.86       928

Confusion Matrix is :
 [[396  68]
 [ 60 404]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8828742514970059
Model Test Score is :  0.8685344827586207
F1 Score is :  0.8699360341151386
Recall Score is :  0.8793103448275862
Precision Score is :  0.8607594936708861
AUC Value  :  0.8685344827586208

Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.86      0.87       464
           1       0.86      0.88      0.87       464
```

```
    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[398  66]
 [ 56 408]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8677844311377245
Model Test Score is :  0.8556034482758621
F1 Score is :  0.858050847457627
Recall Score is :  0.8728448275862069
Precision Score is :  0.84375
AUC Value  :  0.855603448275862

Classification Report is :                precision    recall  f1-score
support

           0       0.87      0.84      0.85       464
           1       0.84      0.87      0.86       464

    accuracy                           0.86       928
   macro avg       0.86      0.86      0.86       928
weighted avg       0.86      0.86      0.86       928

Confusion Matrix is :
 [[389  75]
 [ 59 405]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8774850299401198
Model Test Score is :  0.8620689655172413
F1 Score is :  0.8632478632478633
Recall Score is :  0.8706896551724138
Precision Score is :  0.8559322033898306
AUC Value  :  0.8620689655172413

Classification Report is :                precision    recall  f1-score
support

           0       0.87      0.85      0.86       464
           1       0.86      0.87      0.86       464

    accuracy                           0.86       928
```

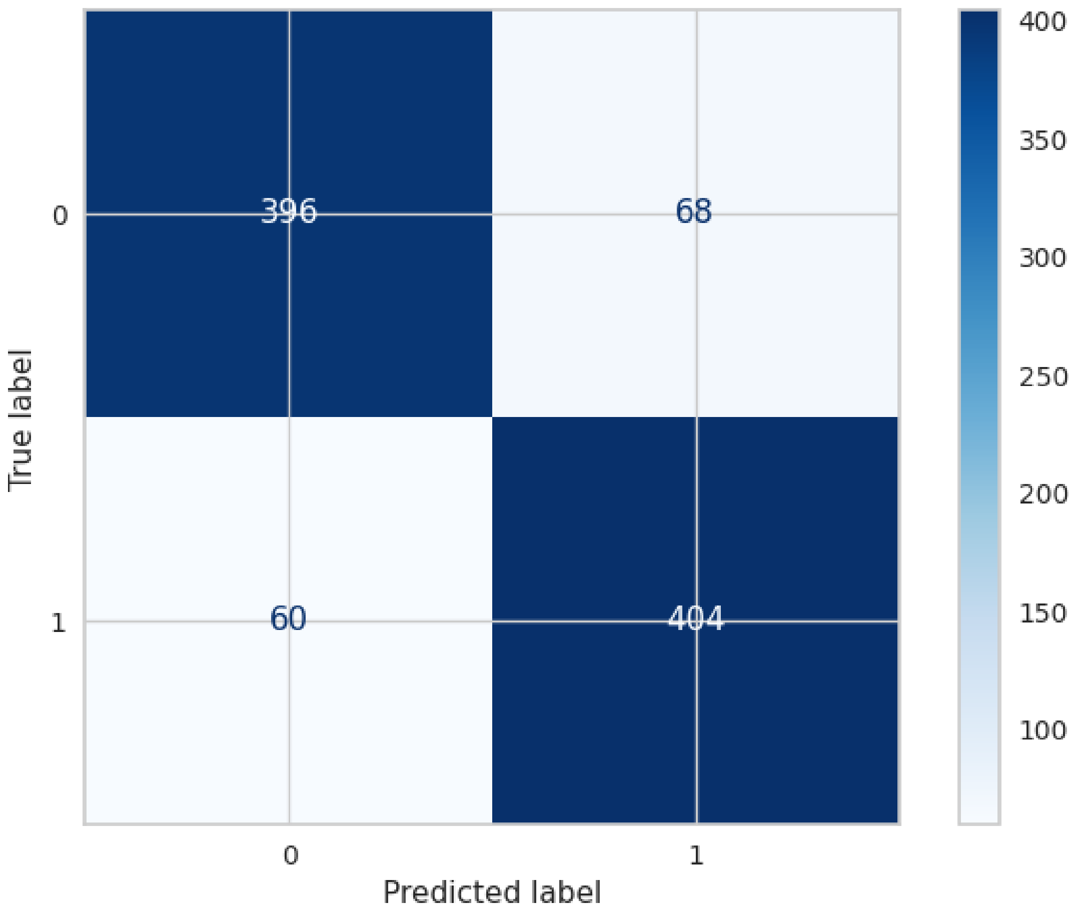
```
   macro avg       0.86      0.86      0.86       928
weighted avg       0.86      0.86      0.86       928

Confusion Matrix is :
 [[396  68]
 [ 60 404]]
```

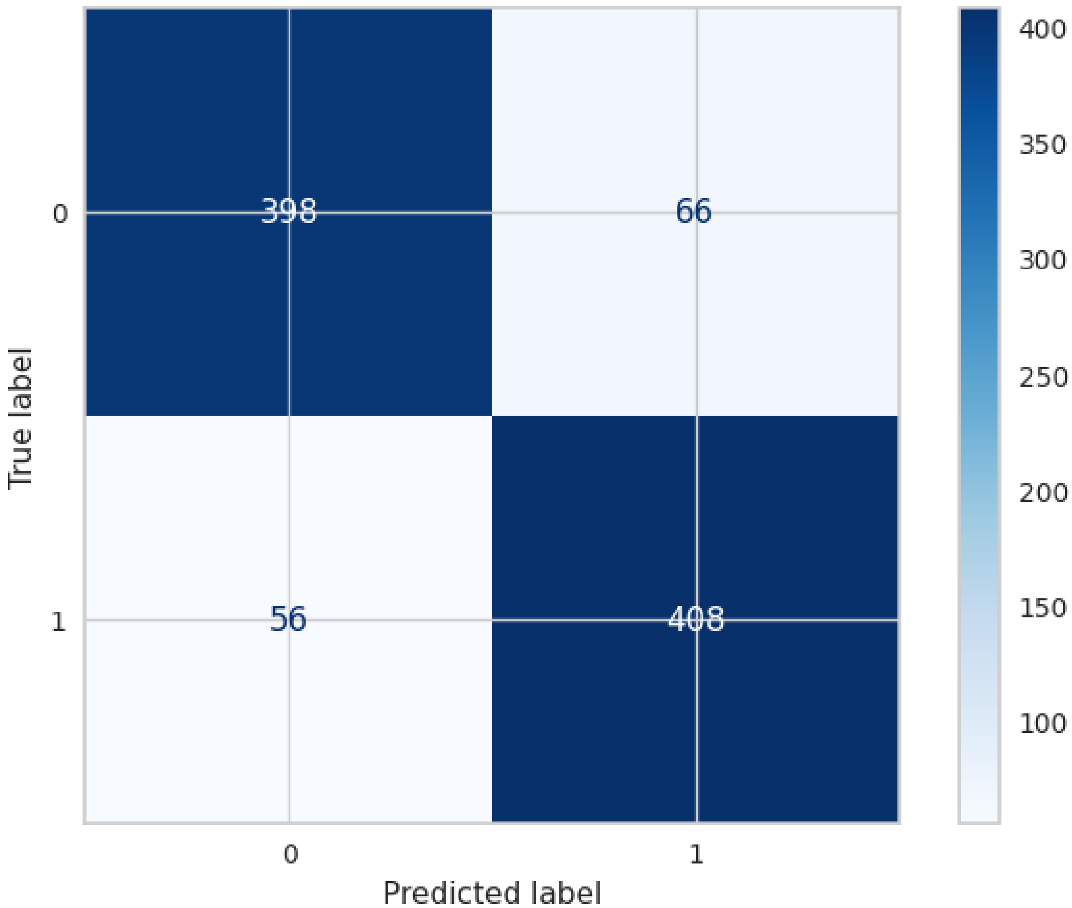


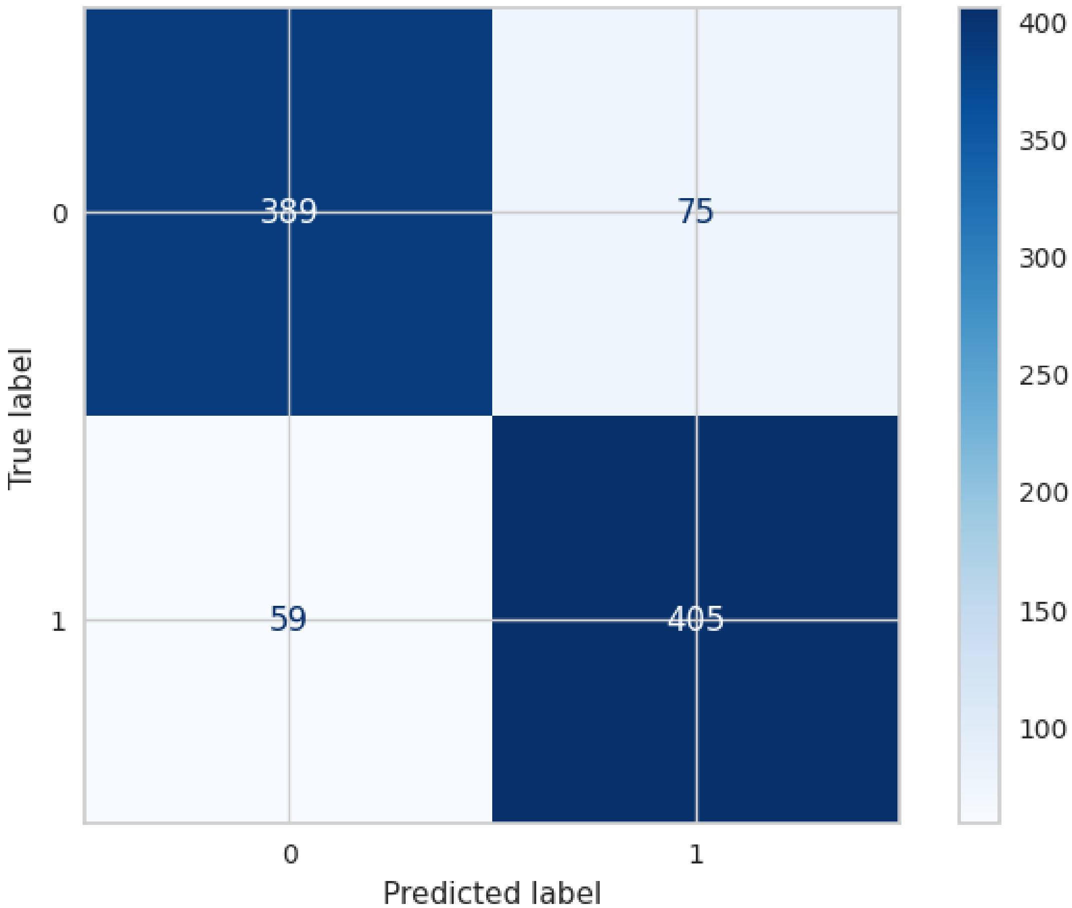

392
72
45
419
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50

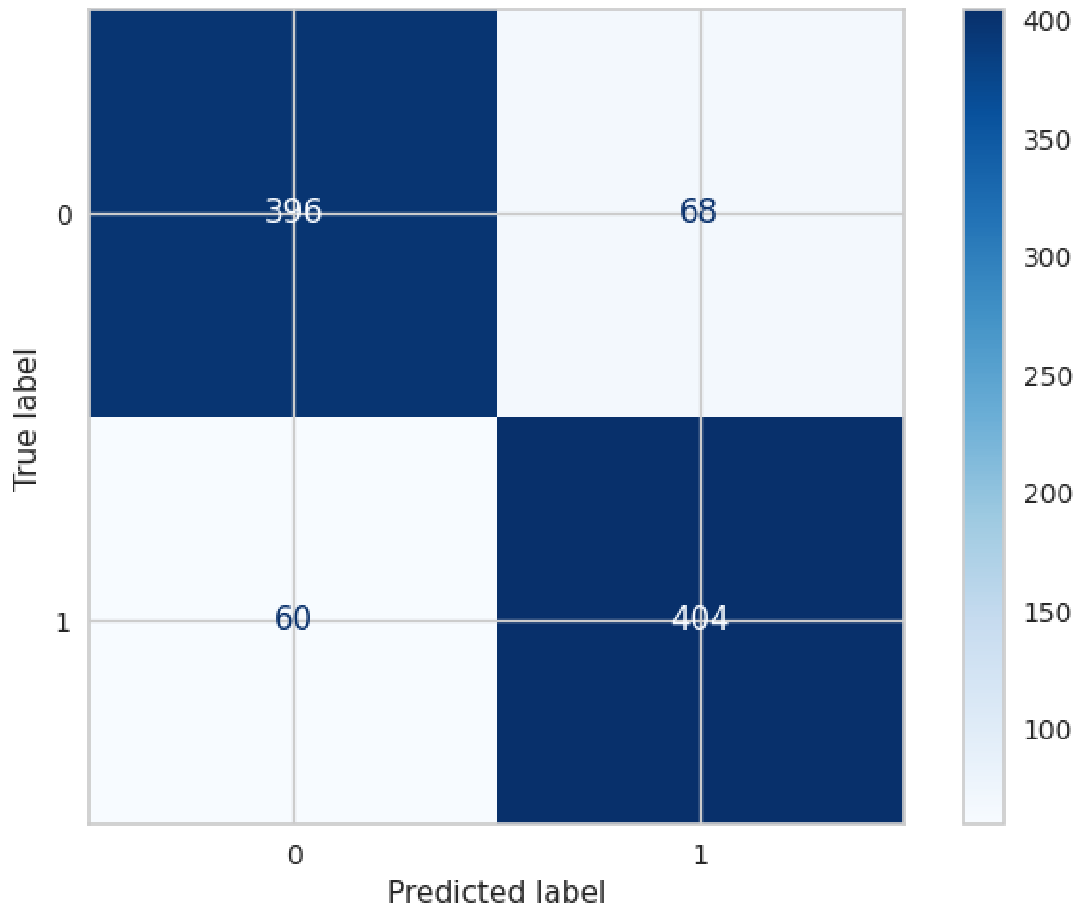


0
1
True label
389
75
59
405
0
1
Predicted label
400
350
300
250
200
150
100


396
68
60
404
True label
Predicted label
0
1
0
1
400
350
300
250
200
150
100


398
66
56
408
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100


389
75
59
405
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100



[259]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['KNN Under','KNN Under With Feature','KNN Under Scaling','KNN␣
↪Under With Normalize','KNN Under With PCA'
                ,'KNN Under With PCA and Scaling',
                'KNN Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[259]:
```
                                  Train Accuracy  Test Accuracy   Test F1  \
Models
KNN Under                               0.883114       0.867457  0.869010
KNN Under With Feature                  0.881557       0.873922  0.877487
KNN Under Scaling                       0.867784       0.855603  0.858051
KNN Under With Normalize                0.877485       0.862069  0.863248
KNN Under With PCA                      0.882874       0.868534  0.869936
KNN Under With PCA and Scaling          0.867784       0.855603  0.858051
KNN Under With PCA and Normalize        0.877485       0.862069  0.863248
```

```
                                    Test Recall  Test Precision       AUC
Models
KNN Under                              0.879310        0.858947  0.867457
KNN Under With Feature                 0.903017        0.853360  0.873922
KNN Under Scaling                      0.872845        0.843750  0.855603
KNN Under With Normalize               0.870690        0.855932  0.862069
KNN Under With PCA                     0.879310        0.860759  0.868534
KNN Under With PCA and Scaling         0.872845        0.843750  0.855603
KNN Under With PCA and Normalize       0.870690        0.855932  0.862069
```

[260]:
```
models_draw(df)
```

SVC

[261]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[262]:
```
Search(SVC(kernel= 'rbf',max_iter=1000,C=1.0),{'C':[1,.
↪5,2,3,5,10]},X_train,y_train)
```

[262]:
```
SVC(C=0.5, max_iter=1000)
```

[263]:
```
cross_validation(SVC(kernel= 'rbf',max_iter=1000,C=.5),X_train,y_train)
```

```
Train Score Value :  [0.79624882 0.31151965 0.88257716 0.85093608 0.85710912]
Mean 0.7396781666305908
Test Score Value :  [0.79803022 0.30576171 0.87424099 0.85062745 0.86614492]
Mean 0.7389610570713463
```

[264]:
```
Values = Models(SVC(kernel= 'rbf',max_iter=1000,C=.
↪5),X_train,y_train,X_test,y_test)
```

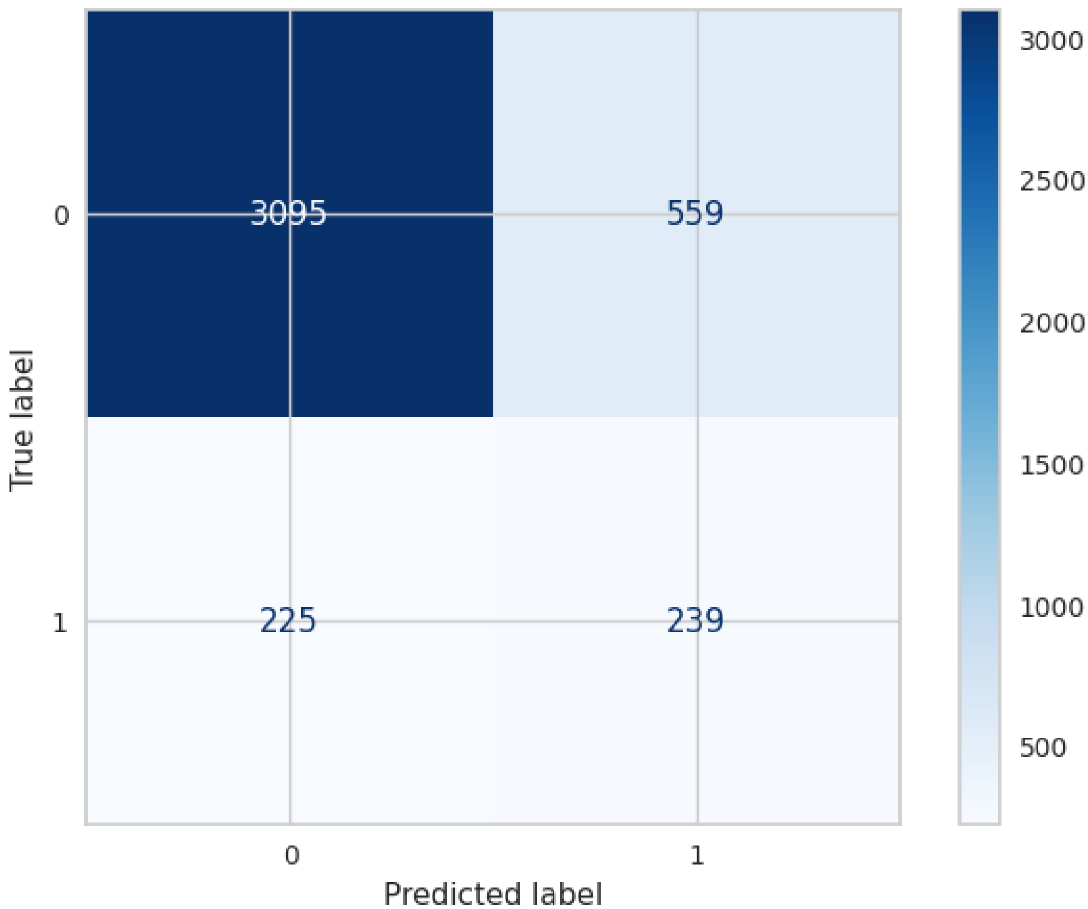
```
Apply Model With Normal Data :

Model Train Score is :  0.8246707685664939
Model Test Score is :  0.8096163186012627
F1 Score is :  0.3787638668779715
Recall Score is :  0.5150862068965517
Precision Score is :  0.29949874686716793
AUC Value  :  0.6810515873015872

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.85      0.89      3654
```

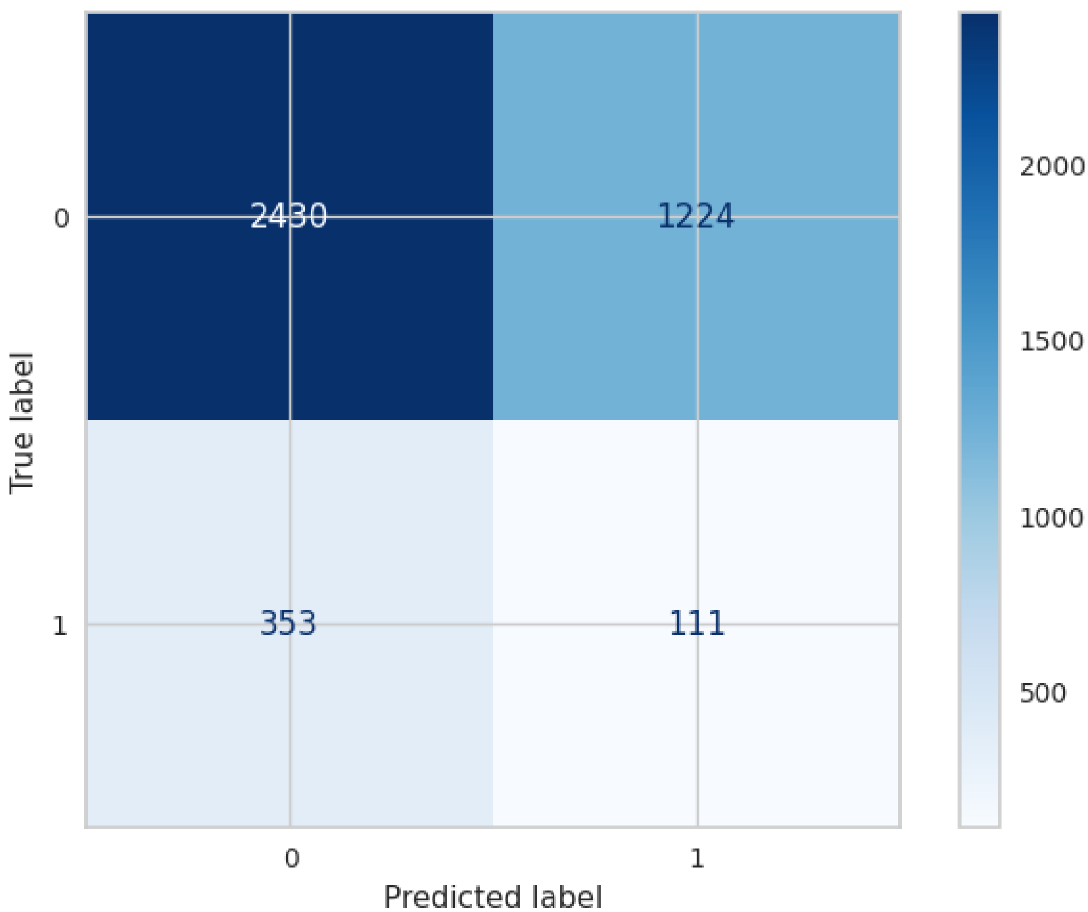
```
           1       0.30      0.52      0.38       464

    accuracy                           0.81      4118
   macro avg       0.62      0.68      0.63      4118
weighted avg       0.86      0.81      0.83      4118

Confusion Matrix is :
 [[3095  559]
 [ 225  239]]


 Apply Model With Feature Selection :

Model Train Score is :  0.6215997409326425
Model Test Score is :  0.6170471102476931
F1 Score is :  0.12340188993885493
Recall Score is :  0.23922413793103448
Precision Score is :  0.08314606741573034
AUC Value  :  0.4521243842364532
Classification Report is :                precision    recall  f1-score
support

           0       0.87      0.67      0.76      3654
           1       0.08      0.24      0.12       464

    accuracy                           0.62      4118
   macro avg       0.48      0.45      0.44      4118
weighted avg       0.78      0.62      0.68      4118

Confusion Matrix is :
 [[2430 1224]
 [ 353  111]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8245628238341969
Model Test Score is :  0.8300145701796989
F1 Score is :  0.38488576449912126
Recall Score is :  0.47198275862068967
Precision Score is :  0.3249258160237389
AUC Value  :  0.6737308429118776
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.88      0.90      3654
           1       0.32      0.47      0.38       464
```

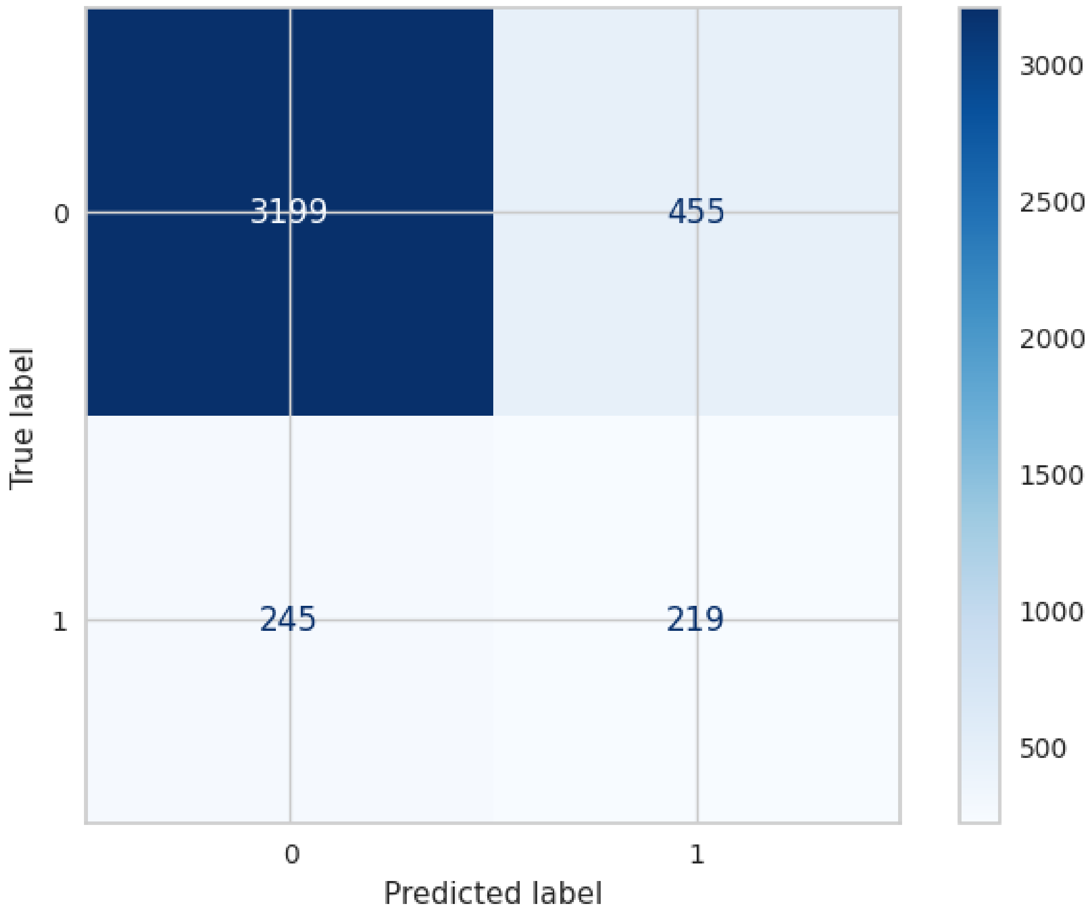
```
   accuracy                           0.83      4118
  macro avg       0.63      0.67      0.64      4118
weighted avg       0.86      0.83      0.84      4118

Confusion Matrix is :
 [[3199  455]
 [ 245  219]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.6014950345423143
Model Test Score is :  0.59834871296746
F1 Score is :  0.3508634222919937
Recall Score is :  0.9633620689655172
Precision Score is :  0.21449136276391556
AUC Value  :  0.7576799397920089
Classification Report is :                precision    recall  f1-score
support

           0       0.99      0.55      0.71      3654
           1       0.21      0.96      0.35       464

    accuracy                           0.60      4118
   macro avg       0.60      0.76      0.53      4118
weighted avg       0.90      0.60      0.67      4118

Confusion Matrix is :
 [[2017 1637]
 [  17  447]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.2692681347150259
Model Test Score is :  0.2799902865468674
F1 Score is :  0.15164520743919882
Recall Score is :  0.5711206896551724
Precision Score is :  0.08742989112504124
AUC Value  :  0.40707101806239737
Classification Report is :                precision    recall  f1-score
support

           0       0.82      0.24      0.37      3654
           1       0.09      0.57      0.15       464
```

```
    accuracy                           0.28      4118
   macro avg       0.45      0.41      0.26      4118
weighted avg       0.73      0.28      0.35      4118

Confusion Matrix is :
 [[ 888 2766]
 [ 199  265]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8845531088082902
Model Test Score is :  0.8841670713938805
F1 Score is :  0.49201277955271566
Recall Score is :  0.4978448275862069
Precision Score is :  0.4863157894736842
AUC Value  :  0.7155343459222769

Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.93      0.93      3654
           1       0.49      0.50      0.49       464

    accuracy                           0.88      4118
   macro avg       0.71      0.72      0.71      4118
weighted avg       0.89      0.88      0.88      4118

Confusion Matrix is :
 [[3410  244]
 [ 233  231]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.7518890328151986
Model Test Score is :  0.7467217095677513
F1 Score is :  0.43834141087775985
Recall Score is :  0.8771551724137931
Precision Score is :  0.2921751615218952
AUC Value  :  0.8036569512862617

Classification Report is :                precision    recall  f1-score
support

           0       0.98      0.73      0.84      3654
           1       0.29      0.88      0.44       464

    accuracy                           0.75      4118
```

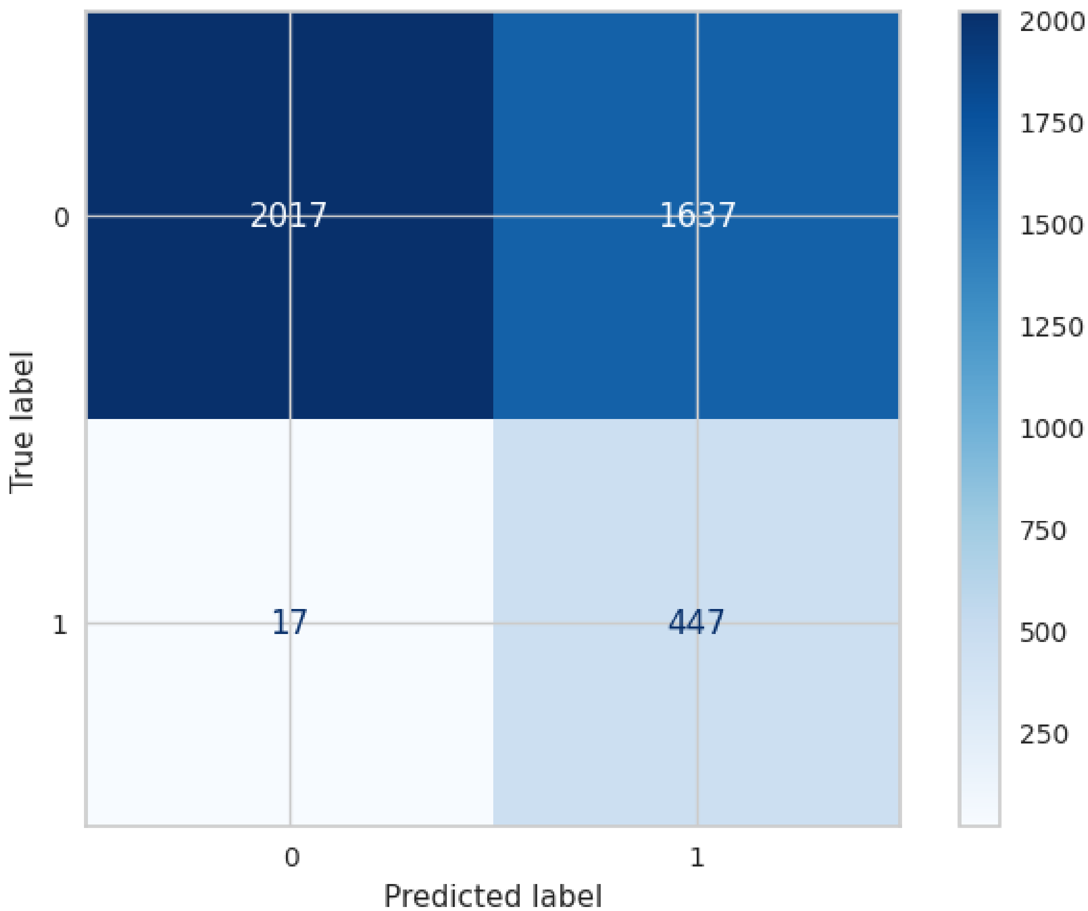
```
   macro avg       0.64      0.80      0.64      4118
weighted avg       0.90      0.75      0.79      4118

Confusion Matrix is :
 [[2668  986]
 [  57  407]]
```



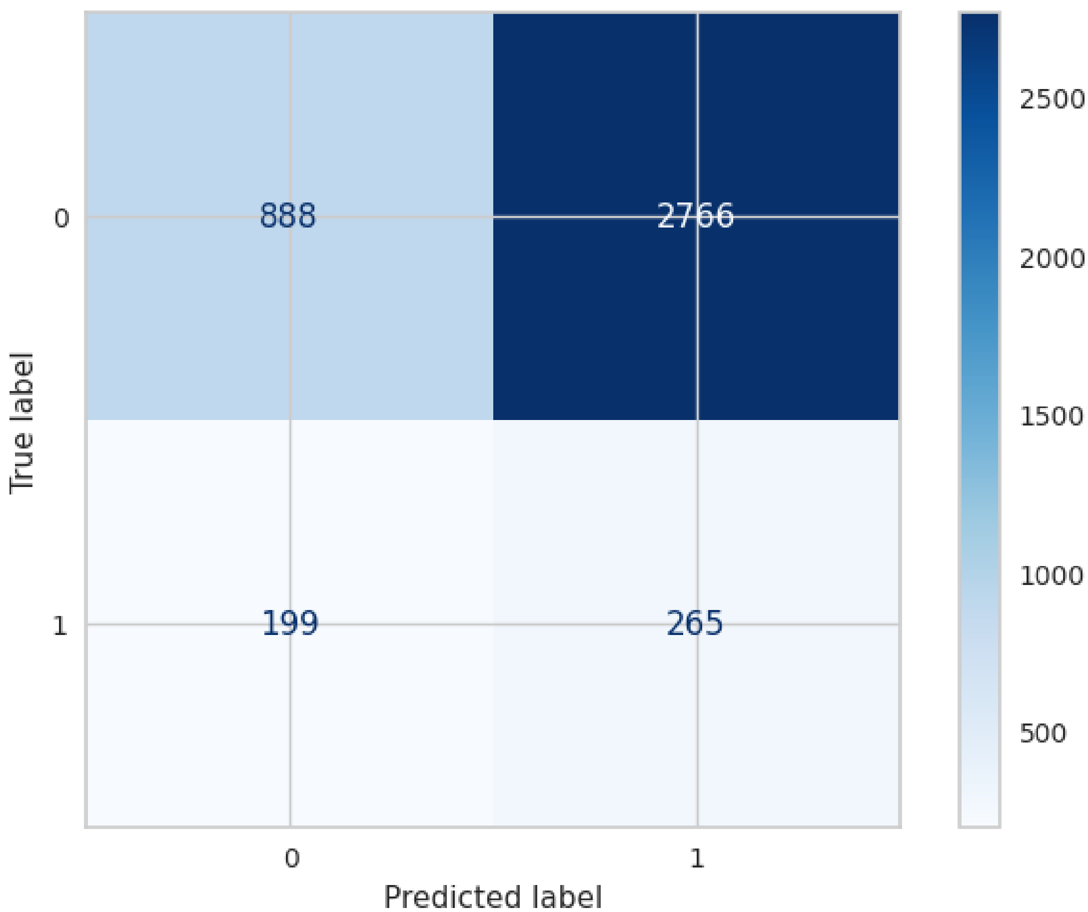

2430
1224
353
111
0
1
True label
0
1
Predicted label
2000
1500
1000
500

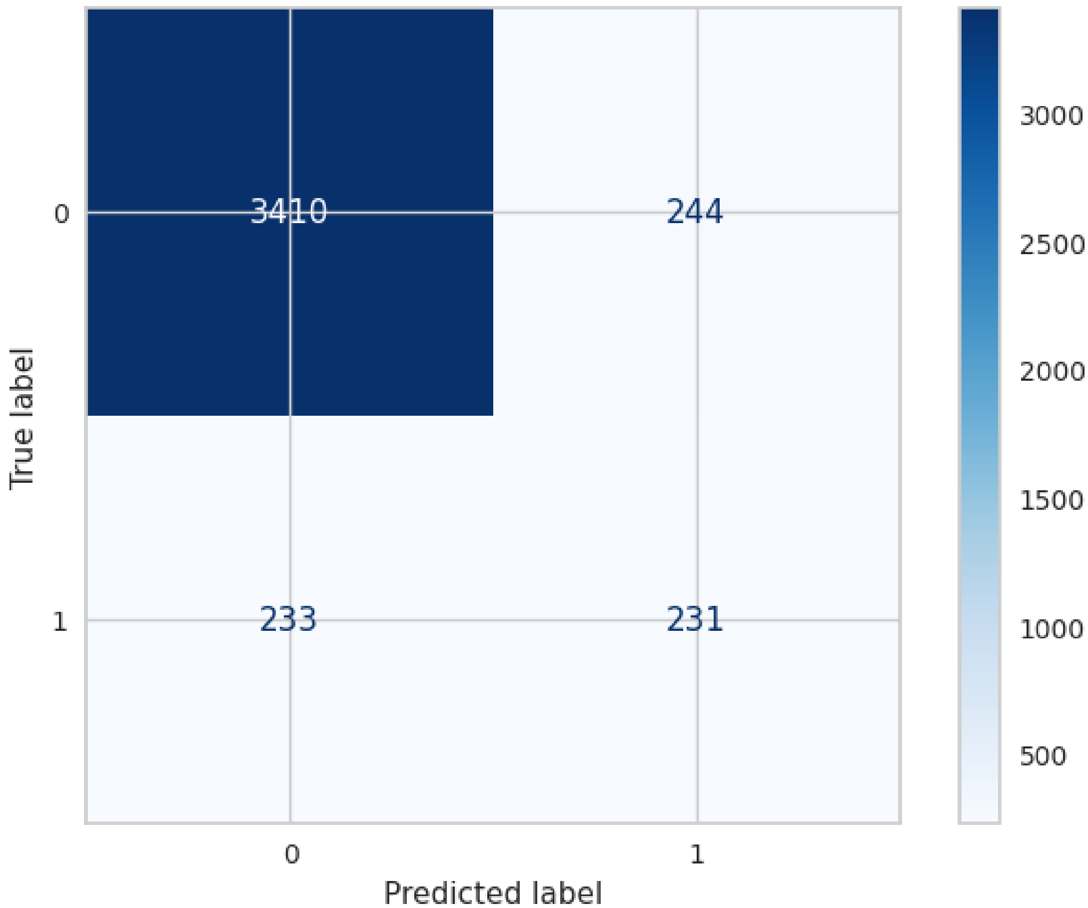


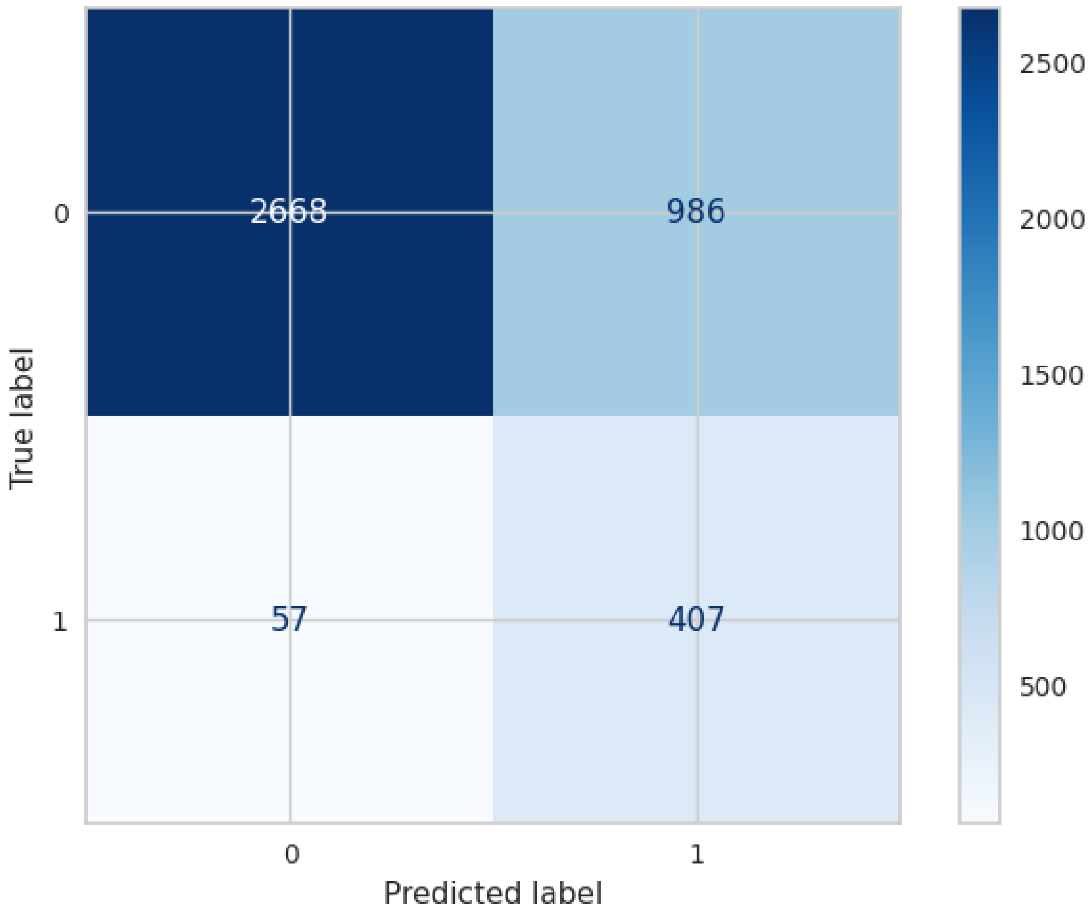
3199
455
245
219
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500



0
1
True label
2017
1637
17
447
0
1
Predicted label
2000
1750
1500
1250
1000
750
500
250


888
2766
199
265
True label
0
1
0
1
Predicted label
2500
2000
1500
1000
500


3410
244
233
231
True label
Predicted label
0
1
0
1
3000
2500
2000
1500
1000
500



[265]:
```python
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
 ↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['SVC','SVC With Feature','SVC Scaling','SVC With␣
 ↪Normalize','SVC With PCA'
                ,'SVC With PCA and Scaling',
                'SVC With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[265]:

Models	Train Accuracy	Test Accuracy	Test F1 \
SVC	0.824671	0.809616	0.378764
SVC With Feature	0.621600	0.617047	0.123402
SVC Scaling	0.824563	0.830015	0.384886
SVC With Normalize	0.601495	0.598349	0.350863
SVC With PCA	0.269268	0.279990	0.151645
SVC With PCA and Scaling	0.884553	0.884167	0.492013
SVC With PCA and Normalize	0.751889	0.746722	0.438341

```
                              Test Recall  Test Precision       AUC
Models
SVC                              0.515086        0.299499  0.681052
SVC With Feature                 0.239224        0.083146  0.452124
SVC Scaling                      0.471983        0.324926  0.673731
SVC With Normalize               0.963362        0.214491  0.757680
SVC With PCA                     0.571121        0.087430  0.407071
SVC With PCA and Scaling         0.497845        0.486316  0.715534
SVC With PCA and Normalize       0.877155        0.292175  0.803657
```

[266]:
```
models_draw(df)
```

RandomOverSampler

[267]:
```
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[268]:
```
Search(SVC(kernel= 'rbf',max_iter=1000,C=1.0),{'C':[1,.
↪5,2,3,5,10]},X_train,y_train)
```

[268]:
```
SVC(C=2, max_iter=1000)
```

[269]:
```
cross_validation(SVC(kernel= 'rbf',max_iter=1000,C=.5),X_train,y_train)
```

```
Train Score Value :  [0.62311348 0.59977191 0.58137236 0.53536333 0.71095398]
Mean 0.6101150111487927
Test Score Value :  [0.62198738 0.60244811 0.58055197 0.53223844 0.71243917]
Mean 0.609933014181032
```

[270]:
```
Values = Models(SVC(kernel= 'rbf',max_iter=1000,C=.
↪5),X_train,y_train,X_test,y_test)
```

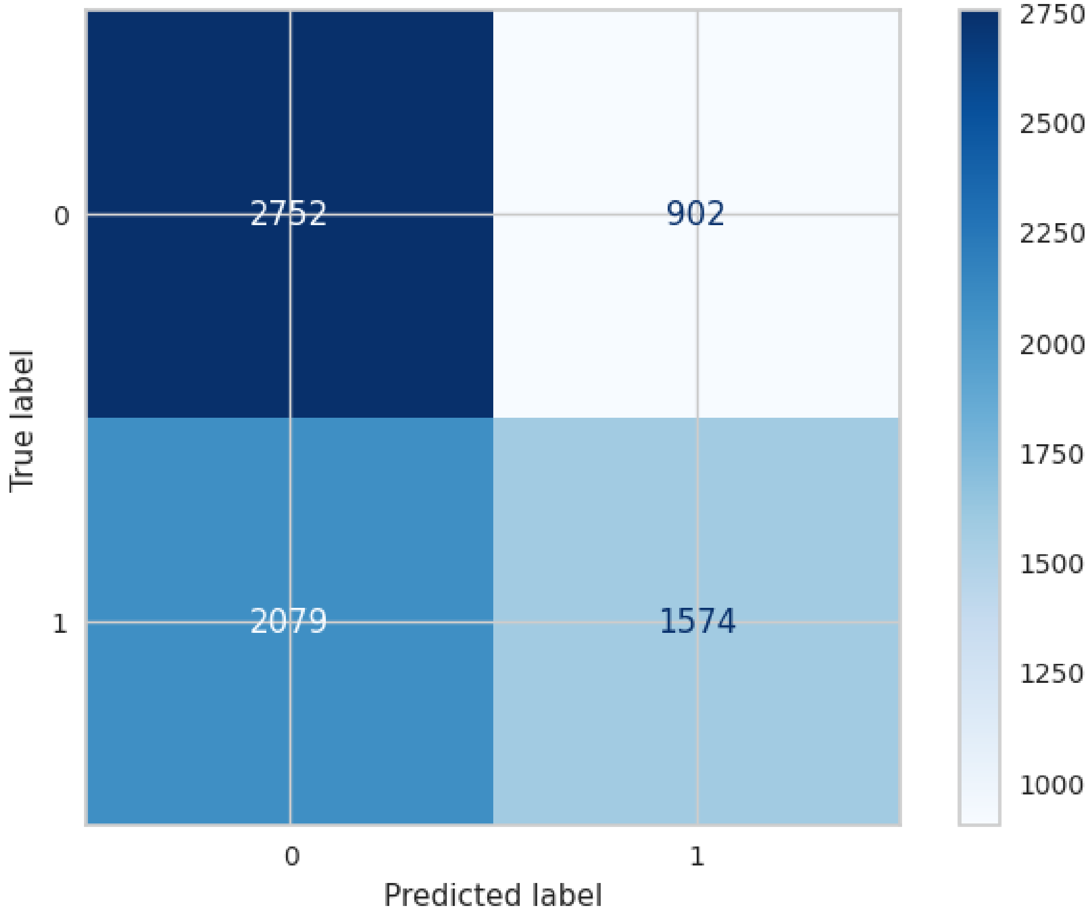
```
Apply Model With Normal Data :

Model Train Score is :  0.589602055867281
Model Test Score is :  0.592035034898043
F1 Score is :  0.5136237559145048
Recall Score is :  0.43087872981111414
Precision Score is :  0.635702746365105
AUC Value  :  0.5920129828584854

Classification Report is :                precision    recall  f1-score
support

           0       0.57      0.75      0.65      3654
```

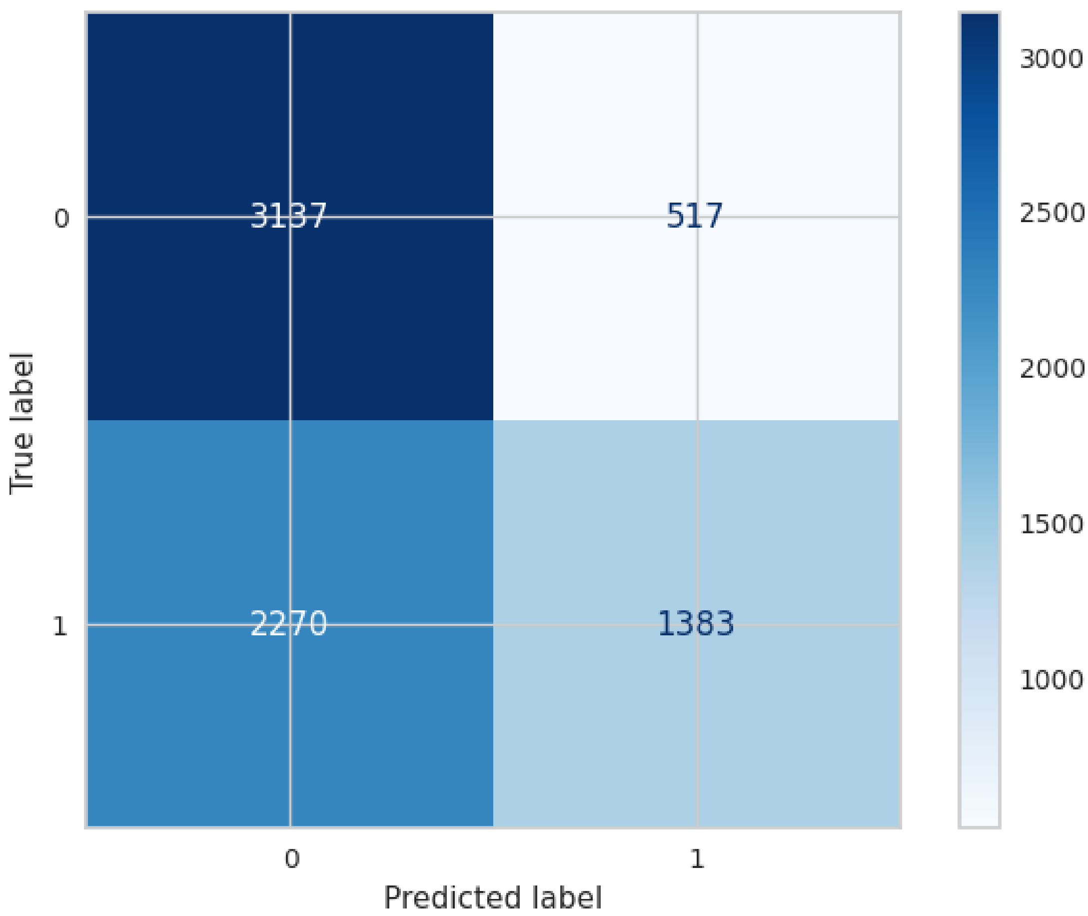
```
           1       0.64      0.43      0.51      3653

    accuracy                           0.59      7307
   macro avg       0.60      0.59      0.58      7307
weighted avg       0.60      0.59      0.58      7307

Confusion Matrix is :
 [[2752  902]
 [2079 1574]]


 Apply Model With Feature Selection :

Model Train Score is :  0.624028709152563
Model Test Score is :  0.6185849185712331
F1 Score is :  0.49810913019989184
Recall Score is :  0.3785929373117985
Precision Score is :  0.7278947368421053
AUC Value  :  0.6185520789459923
Classification Report is :                precision    recall  f1-score
support

           0       0.58      0.86      0.69      3654
           1       0.73      0.38      0.50      3653

    accuracy                           0.62      7307
   macro avg       0.65      0.62      0.60      7307
weighted avg       0.65      0.62      0.60      7307

Confusion Matrix is :
 [[3137  517]
 [2270 1383]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.7322658637835865
Model Test Score is :  0.7331326125632954
F1 Score is :  0.7325102880658436
Recall Score is :  0.730906104571585
Precision Score is :  0.7341215287324718
AUC Value  :  0.7331323078960827
Classification Report is :                precision    recall  f1-score
support

           0       0.73      0.74      0.73      3654
           1       0.73      0.73      0.73      3653
```

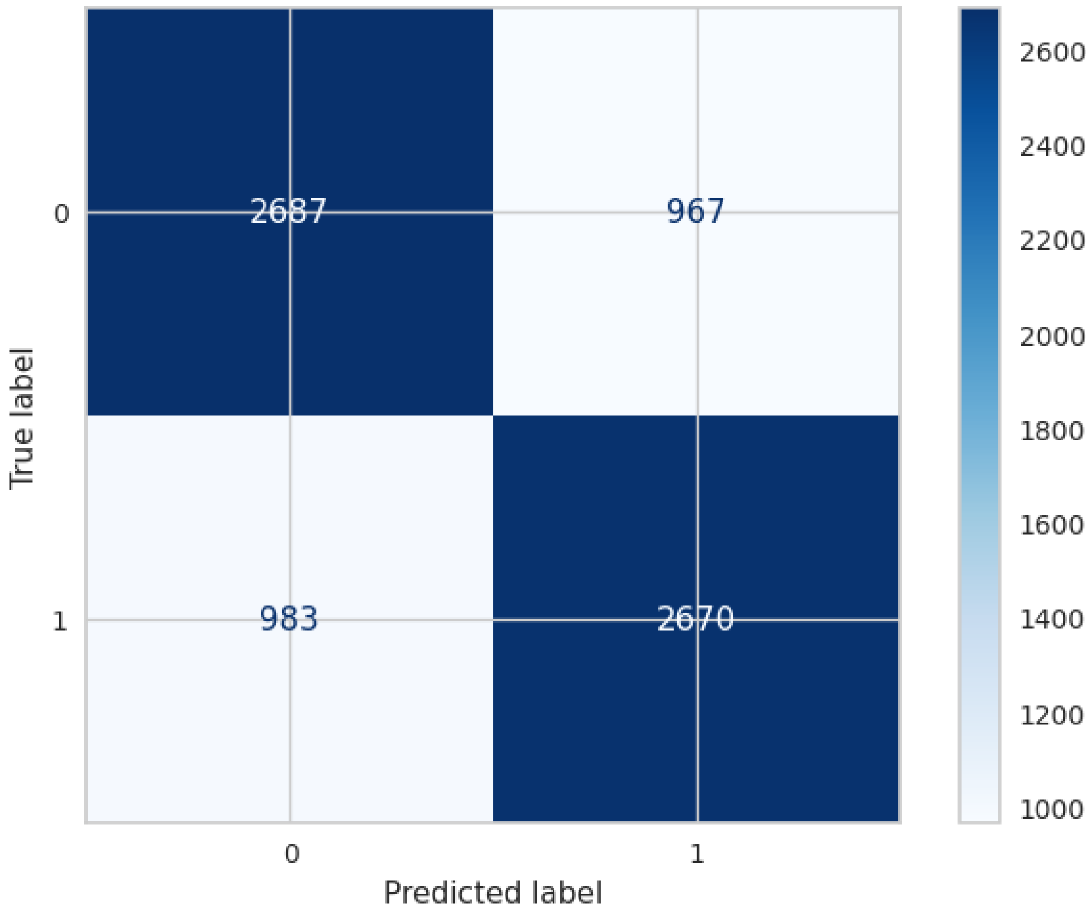
```
    accuracy                           0.73      7307
   macro avg       0.73      0.73      0.73      7307
weighted avg       0.73      0.73      0.73      7307

Confusion Matrix is :
 [[2687  967]
 [ 983 2670]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8163252892964129
Model Test Score is :  0.8099083071027782
F1 Score is :  0.8160508541914977
Recall Score is :  0.8434163701067615
Precision Score is :  0.7904053360697794
AUC Value  :  0.8099128922236052
Classification Report is :                precision    recall  f1-score
support

           0       0.83      0.78      0.80      3654
           1       0.79      0.84      0.82      3653

    accuracy                           0.81      7307
   macro avg       0.81      0.81      0.81      7307
weighted avg       0.81      0.81      0.81      7307

Confusion Matrix is :
 [[2837  817]
 [ 572 3081]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.3730517160105226
Model Test Score is :  0.3786779800191597
F1 Score is :  0.3530920490168139
Recall Score is :  0.33917328223378046
Precision Score is :  0.3682020802377415
AUC Value  :  0.3786725743407545
Classification Report is :                precision    recall  f1-score
support

           0       0.39      0.42      0.40      3654
           1       0.37      0.34      0.35      3653
```

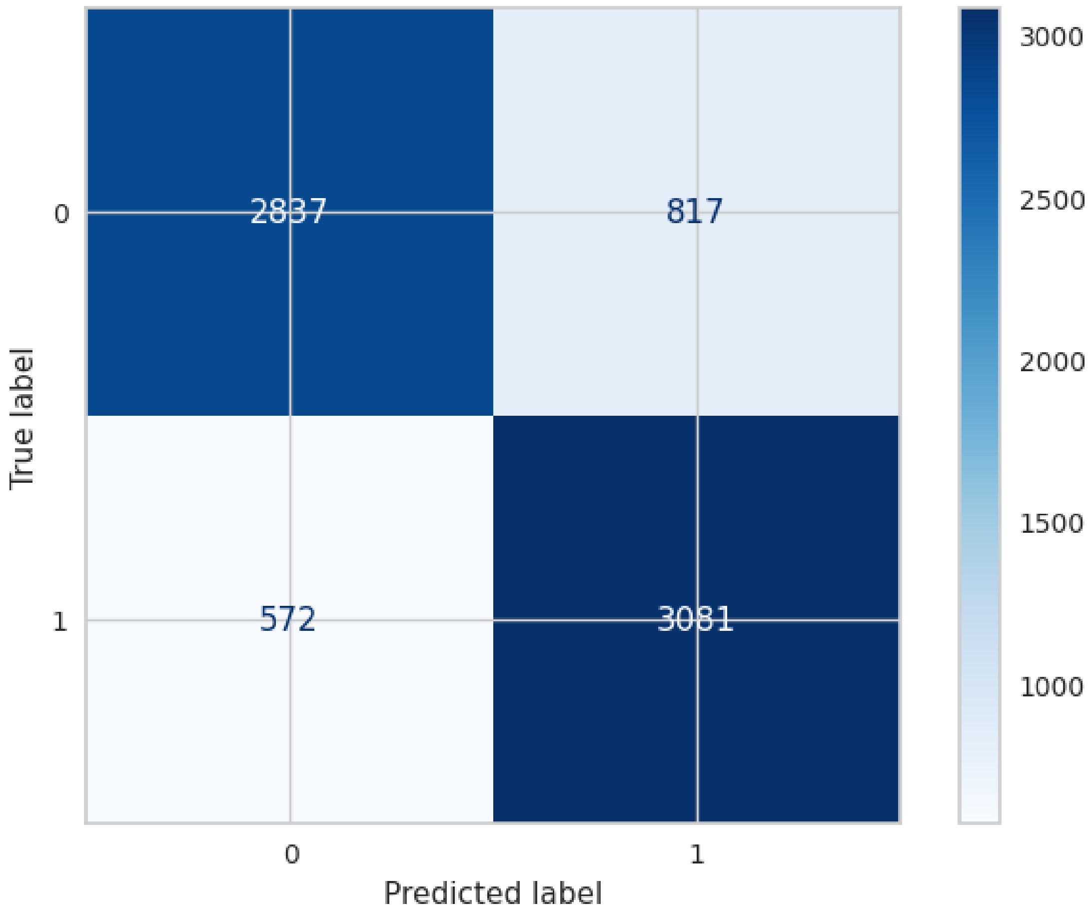
```
    accuracy                           0.38      7307
   macro avg       0.38      0.38      0.38      7307
weighted avg       0.38      0.38      0.38      7307

Confusion Matrix is :
 [[1528 2126]
 [2414 1239]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.7399145416115445
Model Test Score is :  0.7502394963733406
F1 Score is :  0.7867741558593293
Recall Score is :  0.9217081850533808
Precision Score is :  0.6863024867509172
AUC Value  :  0.7502629595217642
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.58      0.70      3654
           1       0.69      0.92      0.79      3653

    accuracy                           0.75      7307
   macro avg       0.78      0.75      0.74      7307
weighted avg       0.78      0.75      0.74      7307

Confusion Matrix is :
 [[2115 1539]
 [ 286 3367]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8064869303407691
Model Test Score is :  0.8062132201998085
F1 Score is :  0.7884672841350462
Recall Score is :  0.7224199288256228
Precision Score is :  0.8678066425517922
AUC Value  :  0.8062017542321874
Classification Report is :                precision    recall  f1-score
support

           0       0.76      0.89      0.82      3654
           1       0.87      0.72      0.79      3653

    accuracy                           0.81      7307
```

```
   macro avg       0.82      0.81      0.80      7307
weighted avg       0.82      0.81      0.80      7307

Confusion Matrix is :
 [[3252  402]
 [1014 2639]]
```

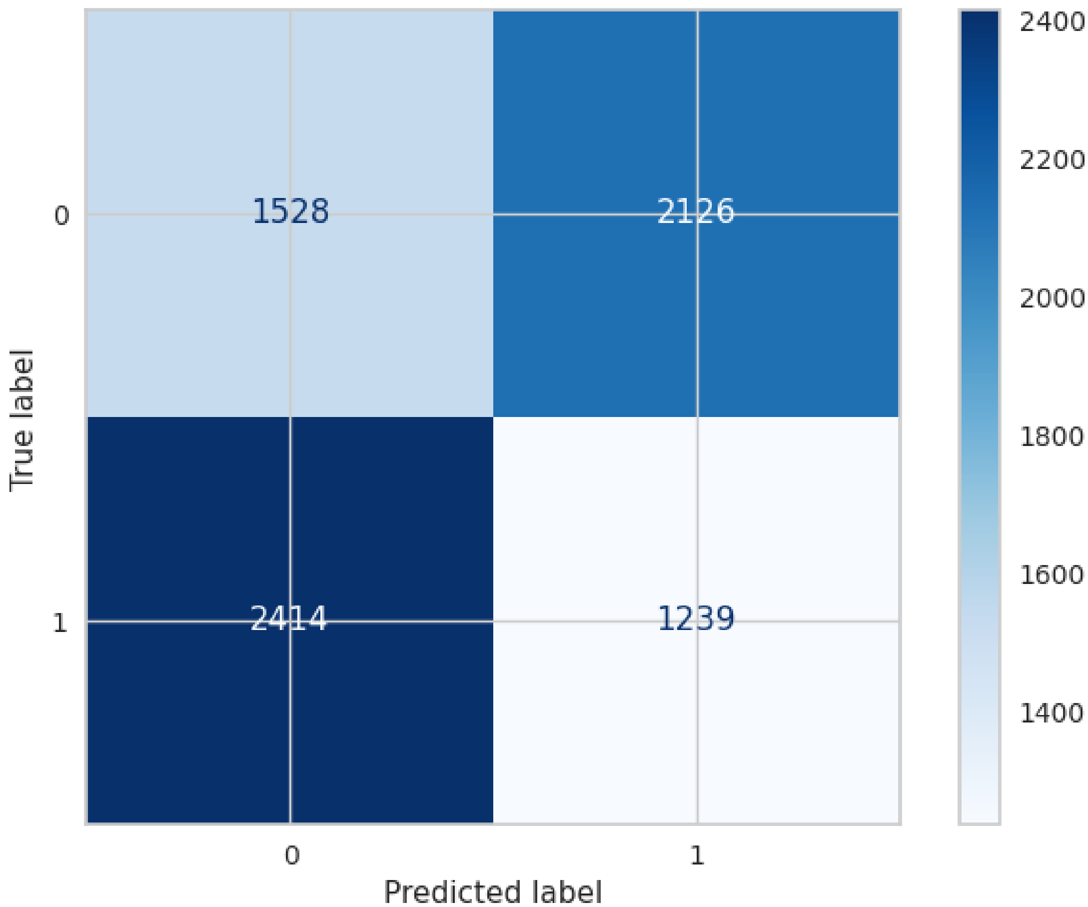


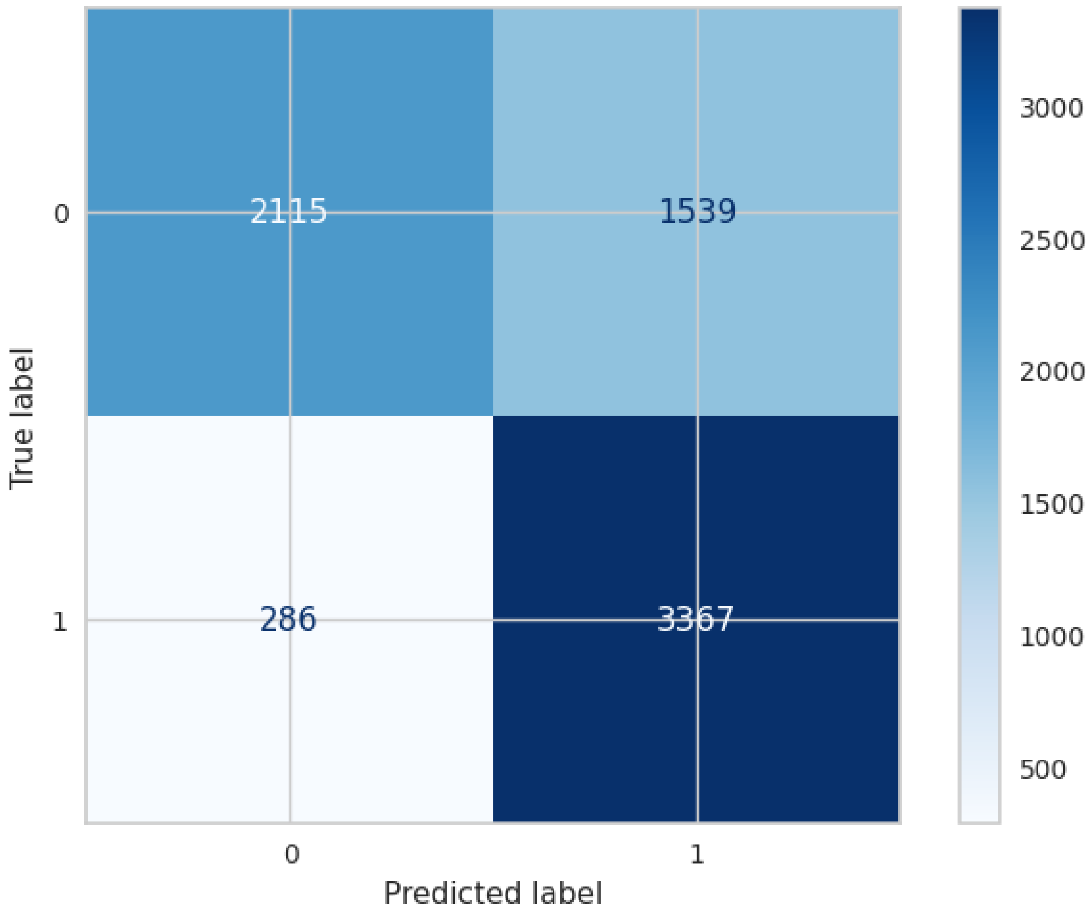

3137
517
2270
1383
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000

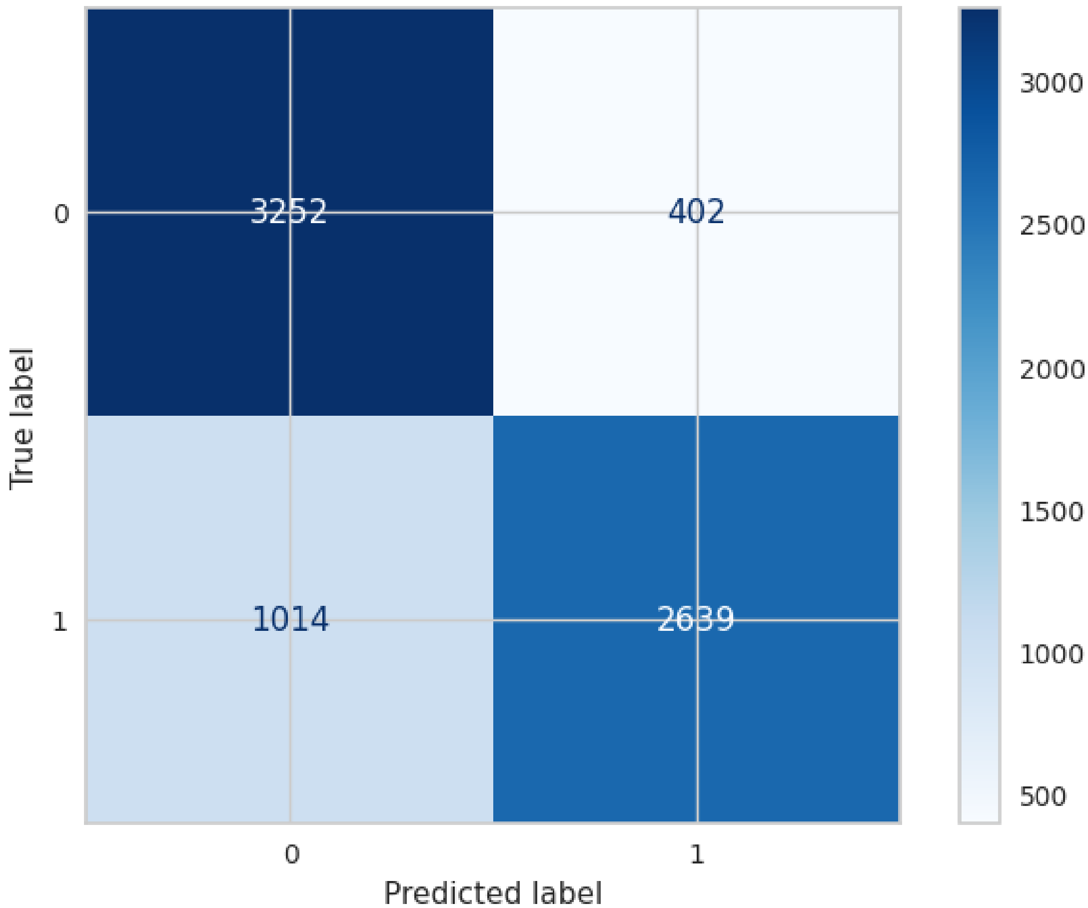

2687
967
983
2670
0
1
True label
0
1
Predicted label
2600
2400
2200
2000
1800
1600
1400
1200
1000



2837
817
572
3081
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000


1528
2126
2414
1239
0
1
True label
0
1
Predicted label
2400
2200
2000
1800
1600
1400


2115
1539
286
3367
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500



[271]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['SVC Under','SVC Under With Feature','SVC Under Scaling','SVC␣
↪Under With Normalize','SVC Under With PCA'
                ,'SVC Under With PCA and Scaling',
                'SVC Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[271]:
```
                                   Train Accuracy  Test Accuracy   Test F1  \
Models
SVC Under                                0.589602       0.592035  0.513624
SVC Under With Feature                   0.624029       0.618585  0.498109
SVC Under Scaling                        0.732266       0.733133  0.732510
SVC Under With Normalize                 0.816325       0.809908  0.816051
SVC Under With PCA                       0.373052       0.378678  0.353092
SVC Under With PCA and Scaling           0.739915       0.750239  0.786774
SVC Under With PCA and Normalize         0.806487       0.806213  0.788467
```

```
                                      Test Recall  Test Precision       AUC
Models
SVC Under                                0.430879        0.635703  0.592013
SVC Under With Feature                   0.378593        0.727895  0.618552
SVC Under Scaling                        0.730906        0.734122  0.733132
SVC Under With Normalize                 0.843416        0.790405  0.809913
SVC Under With PCA                       0.339173        0.368202  0.378673
SVC Under With PCA and Scaling           0.921708        0.686302  0.750263
SVC Under With PCA and Normalize         0.722420        0.867807  0.806202
```

[272]:
```
models_draw(df)
```

RandomUnderSampler

[273]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[274]:
```
Search(SVC(kernel= 'rbf',max_iter=1000,C=1.0),{'C':[1,.
↪5,2,3,5,10]},X_train,y_train)
```

[274]:
```
SVC(C=2, max_iter=1000)
```

[275]:
```
cross_validation(SVC(kernel= 'rbf',max_iter=1000,C=1),X_train,y_train)
```

```
Train Score Value :  [0.38742515 0.425      0.73053892 0.36616766 0.30928144]
Mean 0.44368263473053887
Test Score Value :  [0.37964072 0.4245509  0.71497006 0.37305389 0.30778443]
Mean 0.44000000000000006
```

[276]:
```
Values = Models(SVC(kernel=␣
↪'rbf',max_iter=1000,C=1),X_train,y_train,X_test,y_test)
```

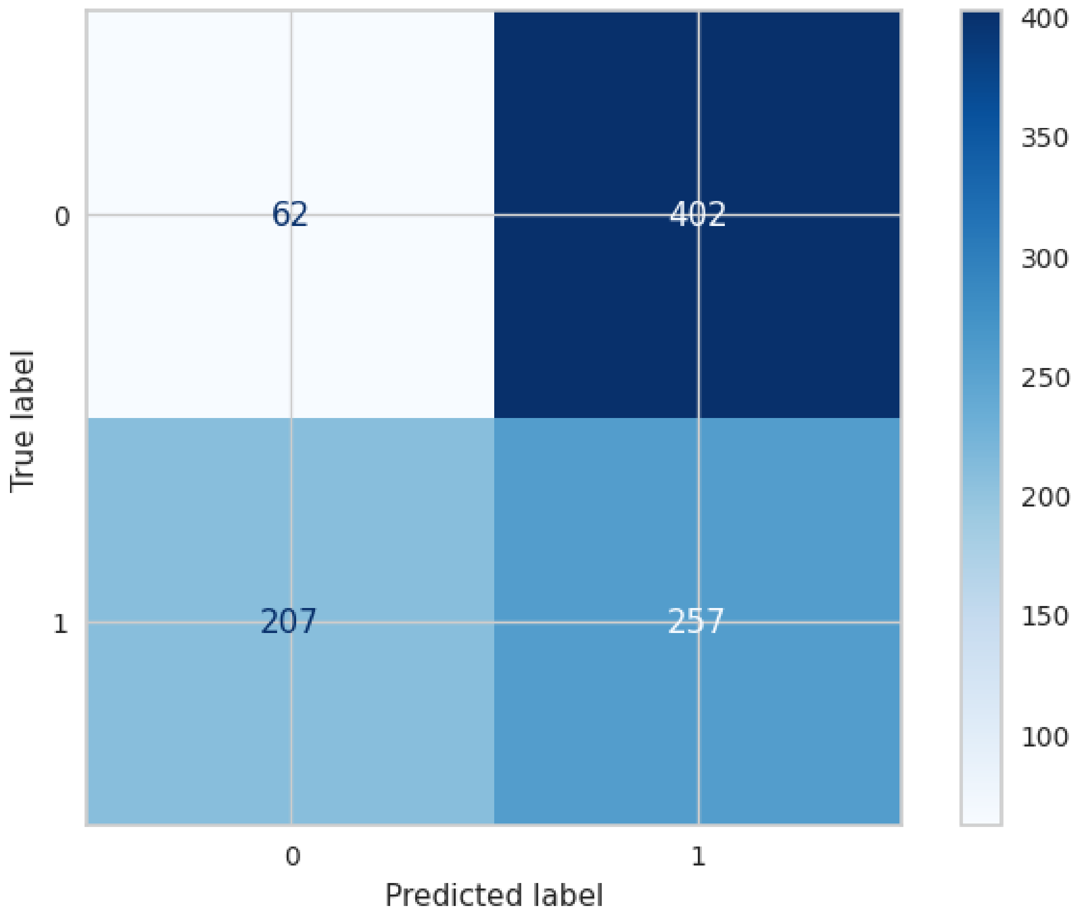
```
Apply Model With Normal Data :

Model Train Score is :  0.3317365269461078
Model Test Score is :  0.34375
F1 Score is :  0.4577025823686554
Recall Score is :  0.5538793103448276
Precision Score is :  0.3899848254931715
AUC Value  :  0.34375

Classification Report is :                precision    recall  f1-score
support

           0       0.23      0.13      0.17       464
```

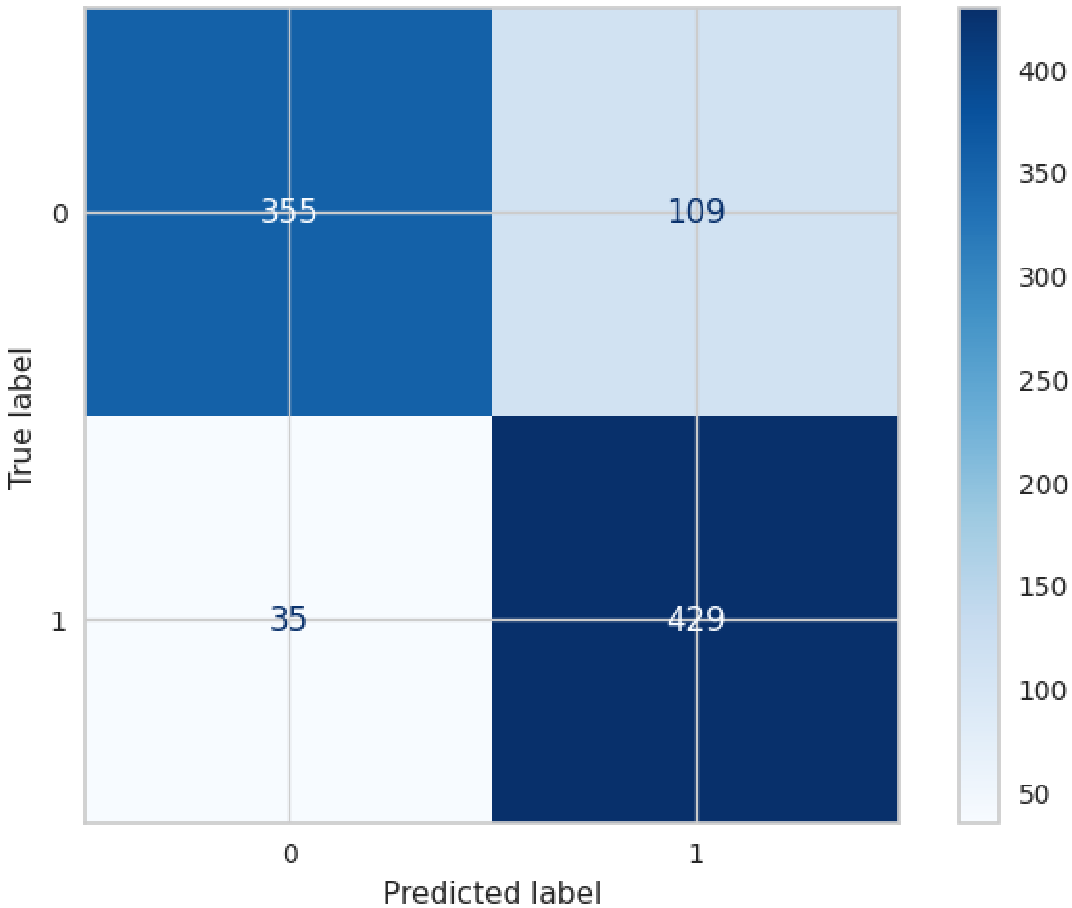
```
           1       0.39      0.55      0.46       464

    accuracy                           0.34       928
   macro avg       0.31      0.34      0.31       928
weighted avg       0.31      0.34      0.31       928

Confusion Matrix is :
 [[ 62 402]
 [207 257]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8344910179640719
Model Test Score is :  0.8448275862068966
F1 Score is :  0.8562874251497006
Recall Score is :  0.9245689655172413
Precision Score is :  0.7973977695167286
AUC Value  :  0.8448275862068966
Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.77      0.83       464
           1       0.80      0.92      0.86       464

    accuracy                           0.84       928
   macro avg       0.85      0.84      0.84       928
weighted avg       0.85      0.84      0.84       928

Confusion Matrix is :
 [[355 109]
 [ 35 429]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8390419161676647
Model Test Score is :  0.8415948275862069
F1 Score is :  0.8595988538681948
Recall Score is :  0.9698275862068966
Precision Score is :  0.7718696397941681
AUC Value  :  0.841594827586207
Classification Report is :                precision    recall  f1-score
support

           0       0.96      0.71      0.82       464
           1       0.77      0.97      0.86       464
```

```
    accuracy                           0.84       928
   macro avg       0.87      0.84      0.84       928
weighted avg       0.87      0.84      0.84       928

Confusion Matrix is :
 [[331 133]
 [ 14 450]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.3379640718562874
Model Test Score is :  0.33728448275862066
F1 Score is :  0.49047224523612265
Recall Score is :  0.6379310344827587
Precision Score is :  0.3983849259757739
AUC Value  :  0.3372844827586207
Classification Report is :                precision    recall  f1-score
support

           0       0.09      0.04      0.05       464
           1       0.40      0.64      0.49       464

    accuracy                           0.34       928
   macro avg       0.25      0.34      0.27       928
weighted avg       0.25      0.34      0.27       928

Confusion Matrix is :
 [[ 17 447]
 [168 296]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8217964071856287
Model Test Score is :  0.8308189655172413
F1 Score is :  0.8459273797841022
Recall Score is :  0.9288793103448276
Precision Score is :  0.7765765765765765
AUC Value  :  0.8308189655172414
Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.73      0.81       464
           1       0.78      0.93      0.85       464
```

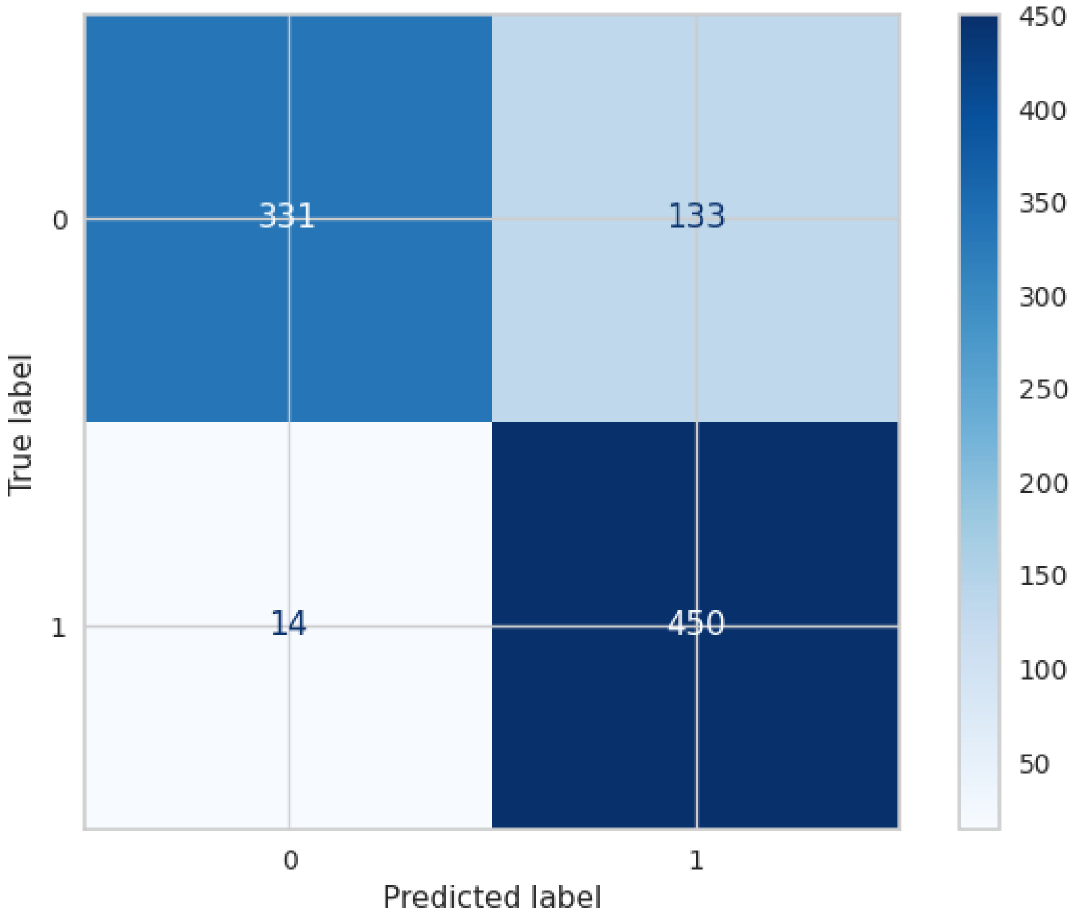
```
    accuracy                           0.83       928
   macro avg       0.84      0.83      0.83       928
weighted avg       0.84      0.83      0.83       928

Confusion Matrix is :
 [[340 124]
 [ 33 431]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8662275449101796
Model Test Score is :  0.8556034482758621
F1 Score is :  0.8651911468812877
Recall Score is :  0.9267241379310345
Precision Score is :  0.8113207547169812
AUC Value  :  0.8556034482758621

Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.78      0.84       464
           1       0.81      0.93      0.87       464

    accuracy                           0.86       928
   macro avg       0.86      0.86      0.85       928
weighted avg       0.86      0.86      0.85       928

Confusion Matrix is :
 [[364 100]
 [ 34 430]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.49353293413173654
Model Test Score is :  0.49461206896551724
F1 Score is :  0.6584122359796066
Recall Score is :  0.9741379310344828
Precision Score is :  0.49724972497249725
AUC Value  :  0.49461206896551724

Classification Report is :                precision    recall  f1-score
support

           0       0.37      0.02      0.03       464
           1       0.50      0.97      0.66       464

    accuracy                           0.49       928
```

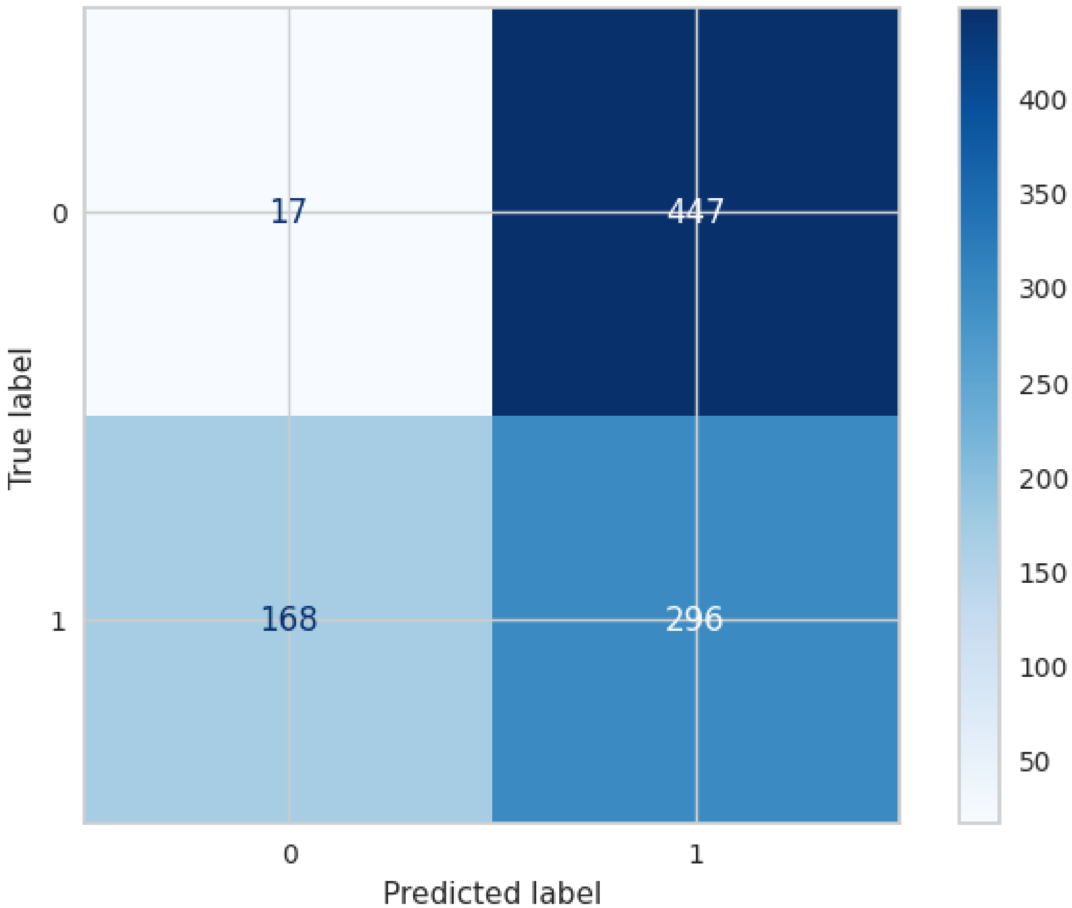
```
   macro avg       0.43      0.49      0.34       928
weighted avg       0.43      0.49      0.34       928

Confusion Matrix is :
 [[  7 457]
 [ 12 452]]
```

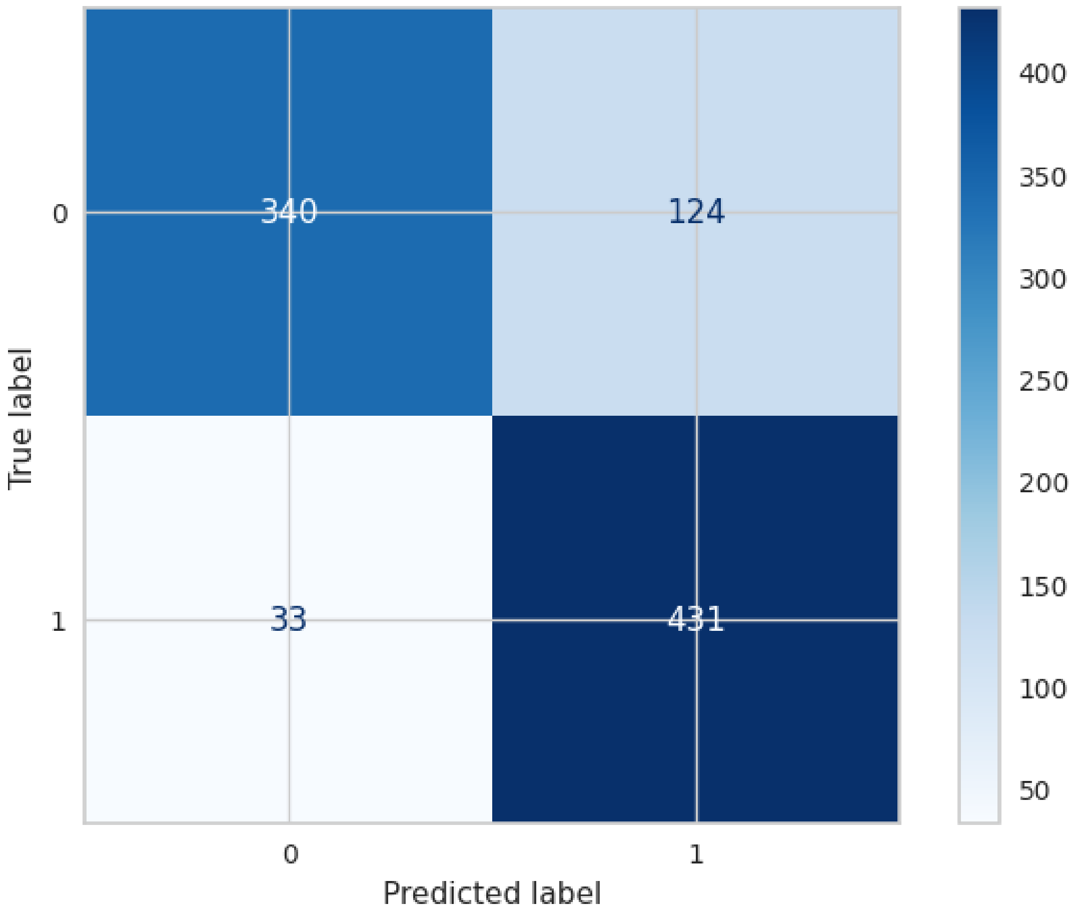


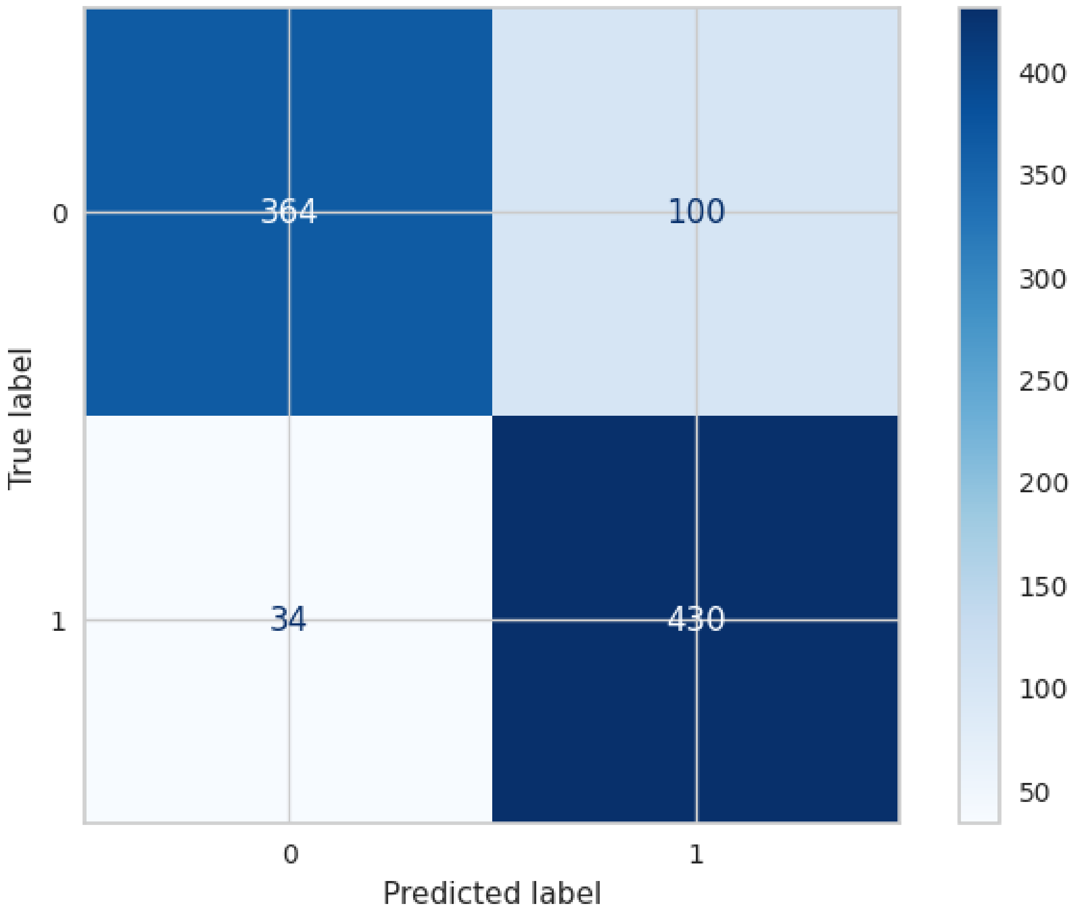


355
109
35
429
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50

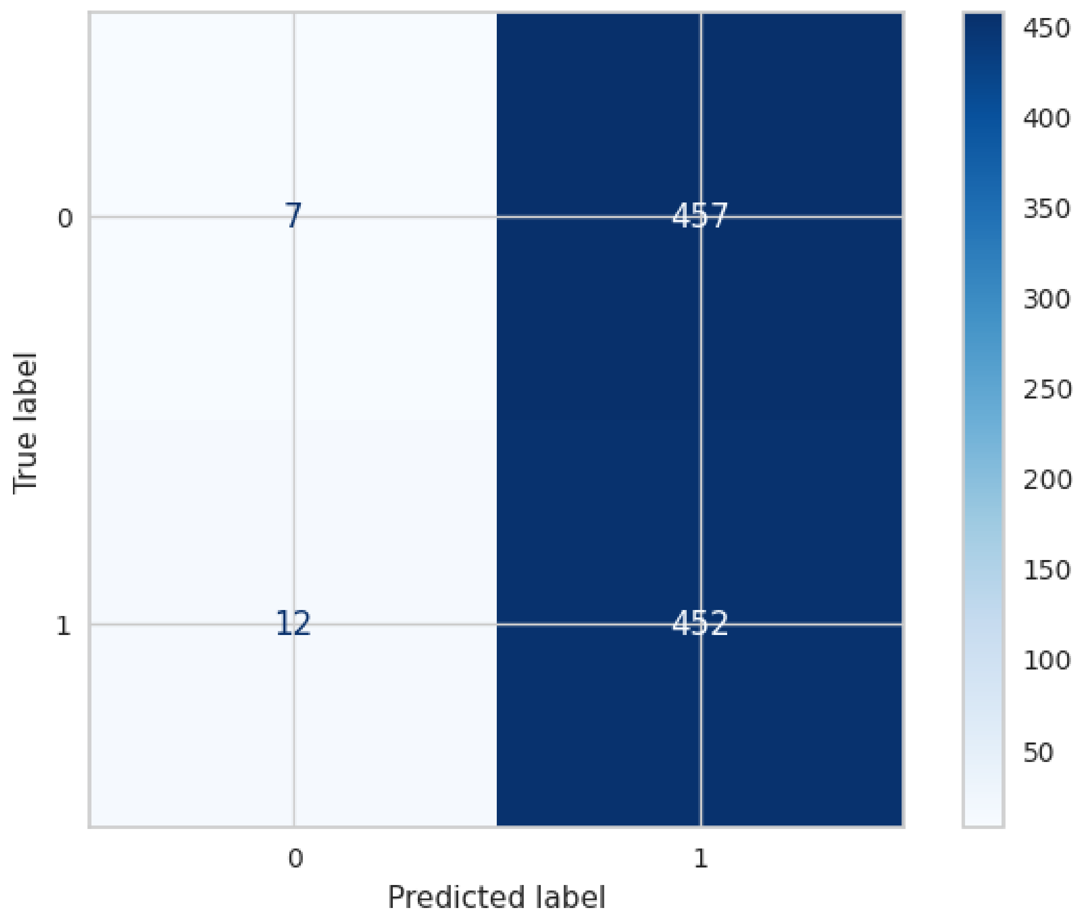

331
133
14
450
0
1
True label
0
1
Predicted label
450
400
350
300
250
200
150
100
50


17
447
168
296
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


340
124
33
431
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50



364
100
34
430
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50




```
[277]: df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
        ↪F1','Test Recall','Test Precision','AUC'])
       df['Models'] = ['SVC Under','SVC Under With Feature','SVC Under Scaling','SVC␣
        ↪Under With Normalize','SVC Under With PCA'
                       ,'SVC Under With PCA and Scaling',
                       'SVC Under With PCA and Normalize']
       df.set_index('Models', inplace=True)
       df
```

```
[277]:                                 Train Accuracy  Test Accuracy   Test F1  \
       Models
       SVC Under                             0.331737       0.343750  0.457703
       SVC Under With Feature                0.834491       0.844828  0.856287
       SVC Under Scaling                     0.839042       0.841595  0.859599
       SVC Under With Normalize              0.337964       0.337284  0.490472
       SVC Under With PCA                    0.821796       0.830819  0.845927
       SVC Under With PCA and Scaling        0.866228       0.855603  0.865191
       SVC Under With PCA and Normalize      0.493533       0.494612  0.658412
```

```
                                    Test Recall  Test Precision       AUC
Models
SVC Under                              0.553879        0.389985  0.343750
SVC Under With Feature                 0.924569        0.797398  0.844828
SVC Under Scaling                      0.969828        0.771870  0.841595
SVC Under With Normalize               0.637931        0.398385  0.337284
SVC Under With PCA                     0.928879        0.776577  0.830819
SVC Under With PCA and Scaling         0.926724        0.811321  0.855603
SVC Under With PCA and Normalize       0.974138        0.497250  0.494612
```

[278]:
```
models_draw(df)
```

LogisticRegression

[279]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[280]:
```
Search(LogisticRegression(penalty='l2',solver='sag',C=1.0),{'C':[1,.
↪5,2,3,5,10]},X_train,y_train)
```

[280]:
```
LogisticRegression(C=0.5, solver='sag')
```

[281]:
```
cross_validation(LogisticRegression(penalty='l2',solver='sag',C=3),X_train,y_train)
```

```
Train Score Value :  [0.90750236 0.90922584 0.90838253 0.90892225 0.90747175]
Mean 0.9083009445259662
Test Score Value :  [0.90974096 0.90500607 0.90703009 0.90703009 0.91377682]
Mean 0.9085168063429874
```

[282]:
```
Values =␣
↪Models(LogisticRegression(penalty='l2',solver='sag',C=3),X_train,y_train,X_test,y_test)
```

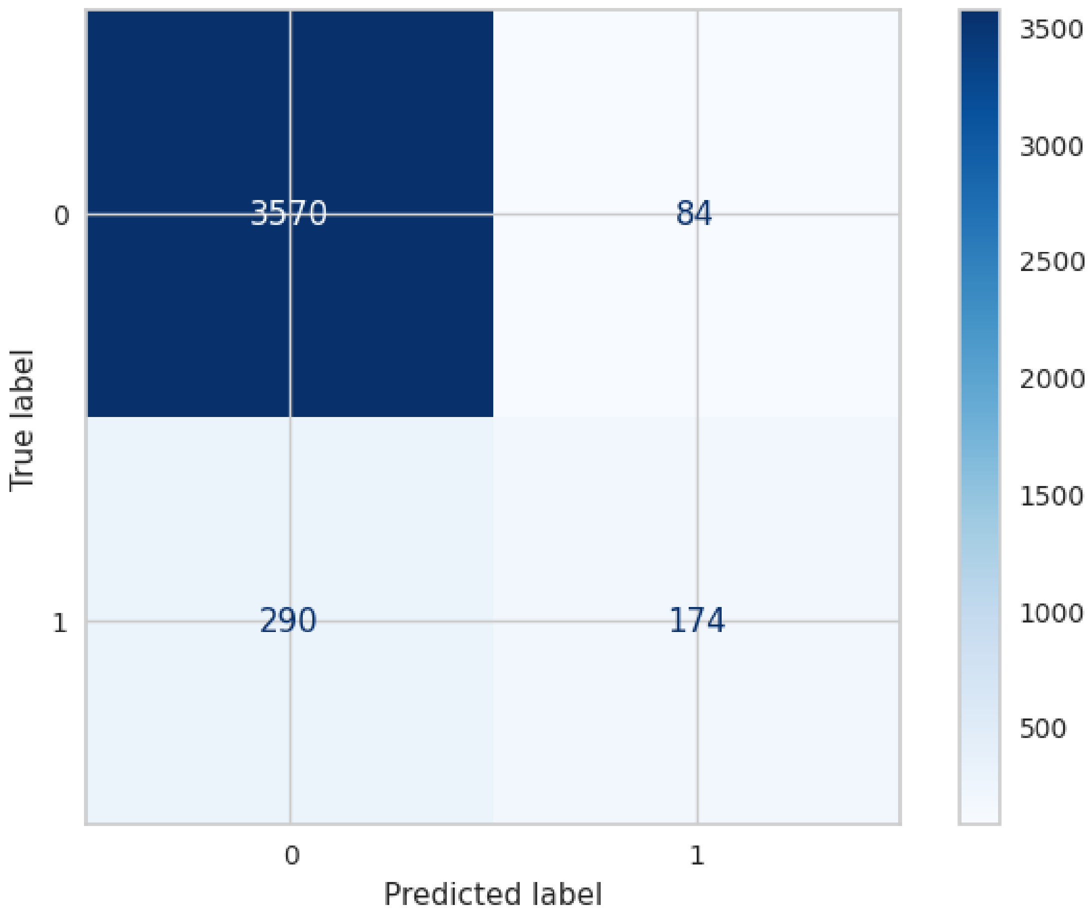
```
Apply Model With Normal Data :

Model Train Score is :  0.9085708117443869
Model Test Score is :  0.9091792132102963
F1 Score is :  0.4819944598337951
Recall Score is :  0.375
Precision Score is :  0.6744186046511628
AUC Value  :  0.6760057471264368

Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
```

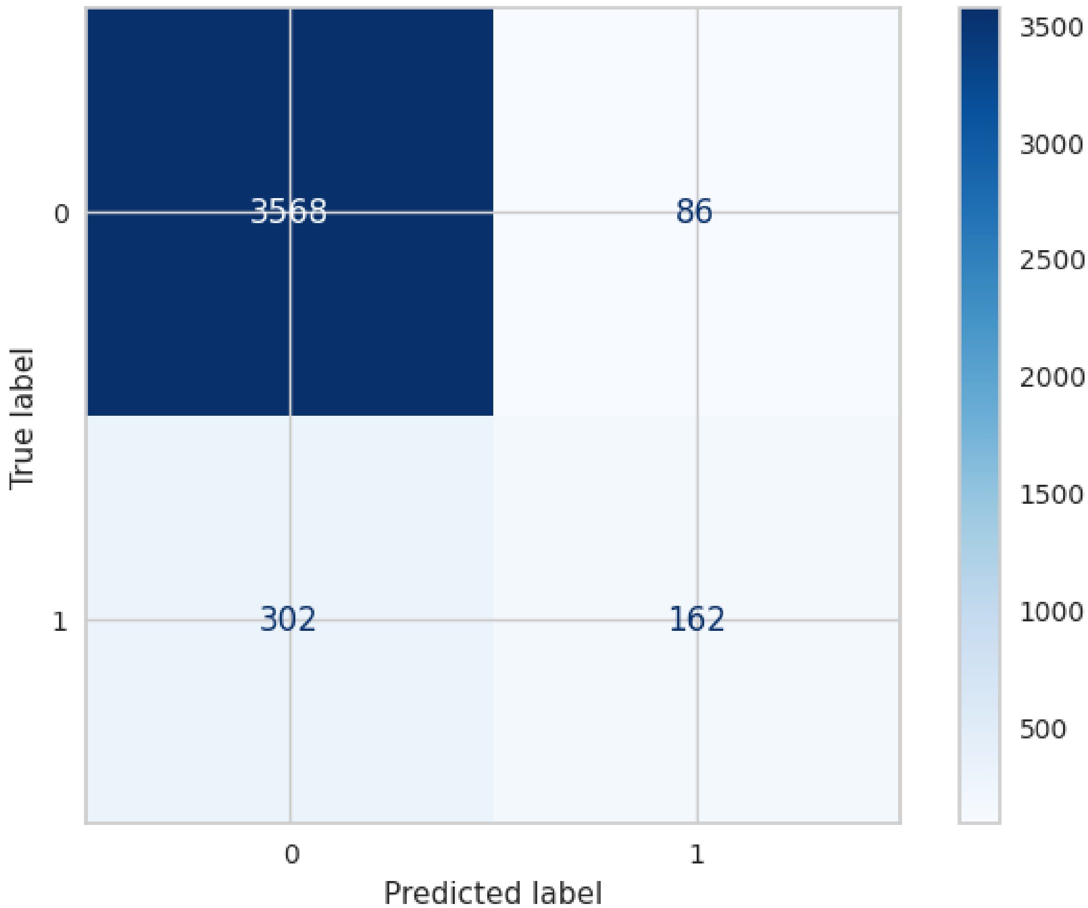
```
           1       0.67      0.38      0.48       464

    accuracy                           0.91      4118
   macro avg       0.80      0.68      0.72      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3570   84]
 [ 290  174]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9031735751295337
Model Test Score is :  0.9057795046138902
F1 Score is :  0.4550561797752809
Recall Score is :  0.34913793103448276
Precision Score is :  0.6532258064516129
AUC Value  :  0.6628010399562123
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
           1       0.65      0.35      0.46       464

    accuracy                           0.91      4118
   macro avg       0.79      0.66      0.70      4118
weighted avg       0.89      0.91      0.89      4118

Confusion Matrix is :
 [[3568   86]
 [ 302  162]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9086247841105354
Model Test Score is :  0.9091792132102963
F1 Score is :  0.48907103825136605
Recall Score is :  0.3857758620689655
Precision Score is :  0.667910447761194
AUC Value  :  0.6807094964422551
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.67      0.39      0.49       464
```

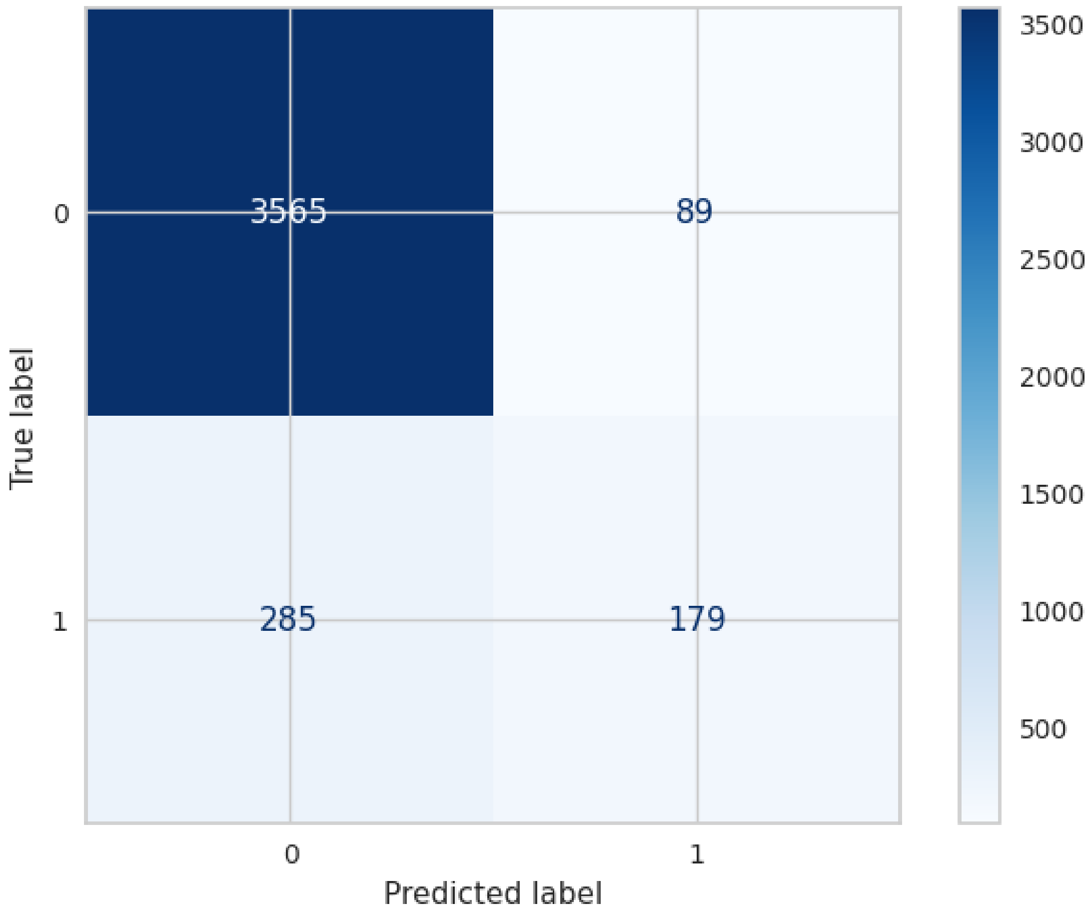
```
   accuracy                           0.91      4118
  macro avg       0.80      0.68      0.72      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3565   89]
 [ 285  179]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.904630829015544
Model Test Score is :  0.9084507042253521
F1 Score is :  0.45283018867924524
Recall Score is :  0.33620689655172414
Precision Score is :  0.6933333333333334
AUC Value  :  0.6586617405582923
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
           1       0.69      0.34      0.45       464

   accuracy                           0.91      4118
  macro avg       0.81      0.66      0.70      4118
weighted avg       0.90      0.91      0.89      4118

Confusion Matrix is :
 [[3585   69]
 [ 308  156]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9085168393782384
Model Test Score is :  0.9082078678970374
F1 Score is :  0.48076923076923067
Recall Score is :  0.3771551724137931
Precision Score is :  0.6628787878787878
AUC Value  :  0.6763991516146689
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.66      0.38      0.48       464
```

```
    accuracy                           0.91      4118
   macro avg       0.79      0.68      0.72      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3565   89]
 [ 289  175]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9086247841105354
Model Test Score is :  0.9091792132102963
F1 Score is :  0.48907103825136605
Recall Score is :  0.3857758620689655
Precision Score is :  0.667910447761194
AUC Value  :  0.6807094964422551

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.98      0.95      3654
           1       0.67      0.39      0.49       464

    accuracy                           0.91      4118
   macro avg       0.80      0.68      0.72      4118
weighted avg       0.90      0.91      0.90      4118

Confusion Matrix is :
 [[3565   89]
 [ 285  179]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9043879533678757
Model Test Score is :  0.9084507042253521
F1 Score is :  0.45283018867924524
Recall Score is :  0.33620689655172414
Precision Score is :  0.6933333333333334
AUC Value  :  0.6586617405582923

Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.98      0.95      3654
           1       0.69      0.34      0.45       464

    accuracy                           0.91      4118
```

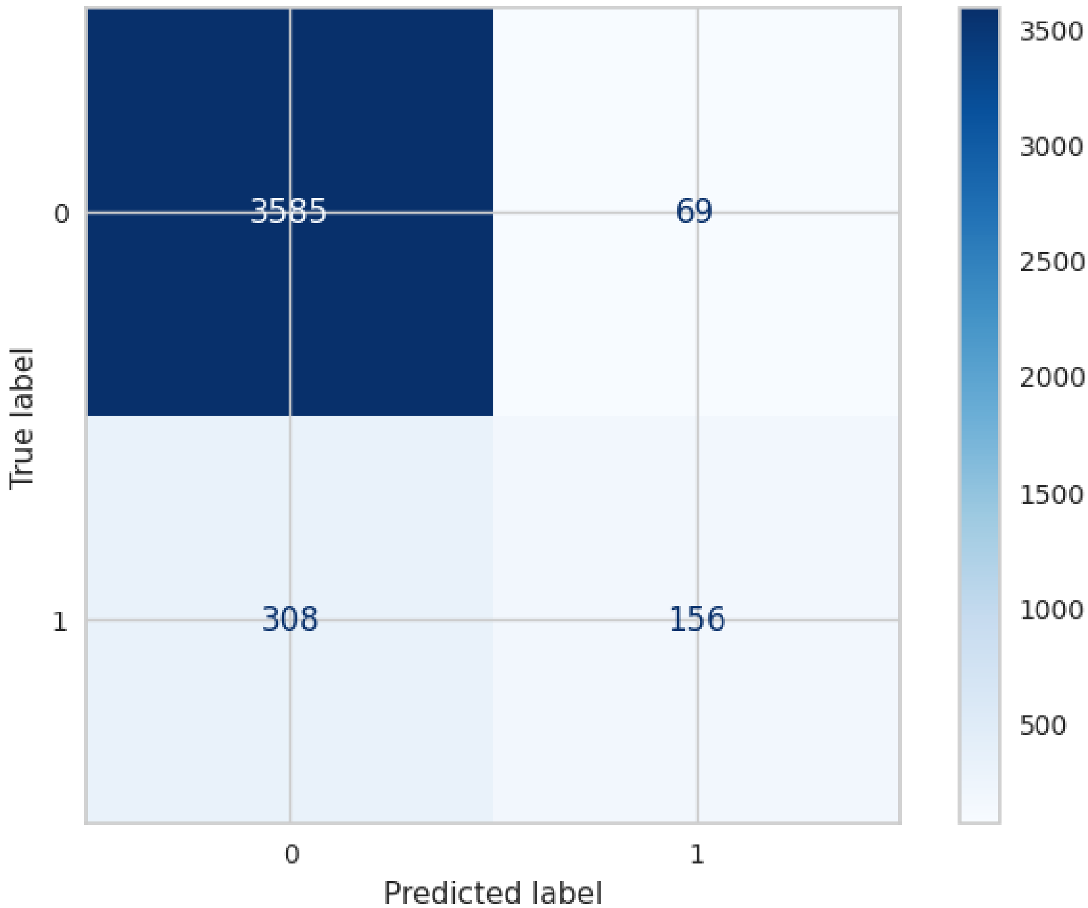
```
   macro avg       0.81      0.66      0.70      4118
weighted avg       0.90      0.91      0.89      4118

Confusion Matrix is :
 [[3585   69]
 [ 308  156]]
```

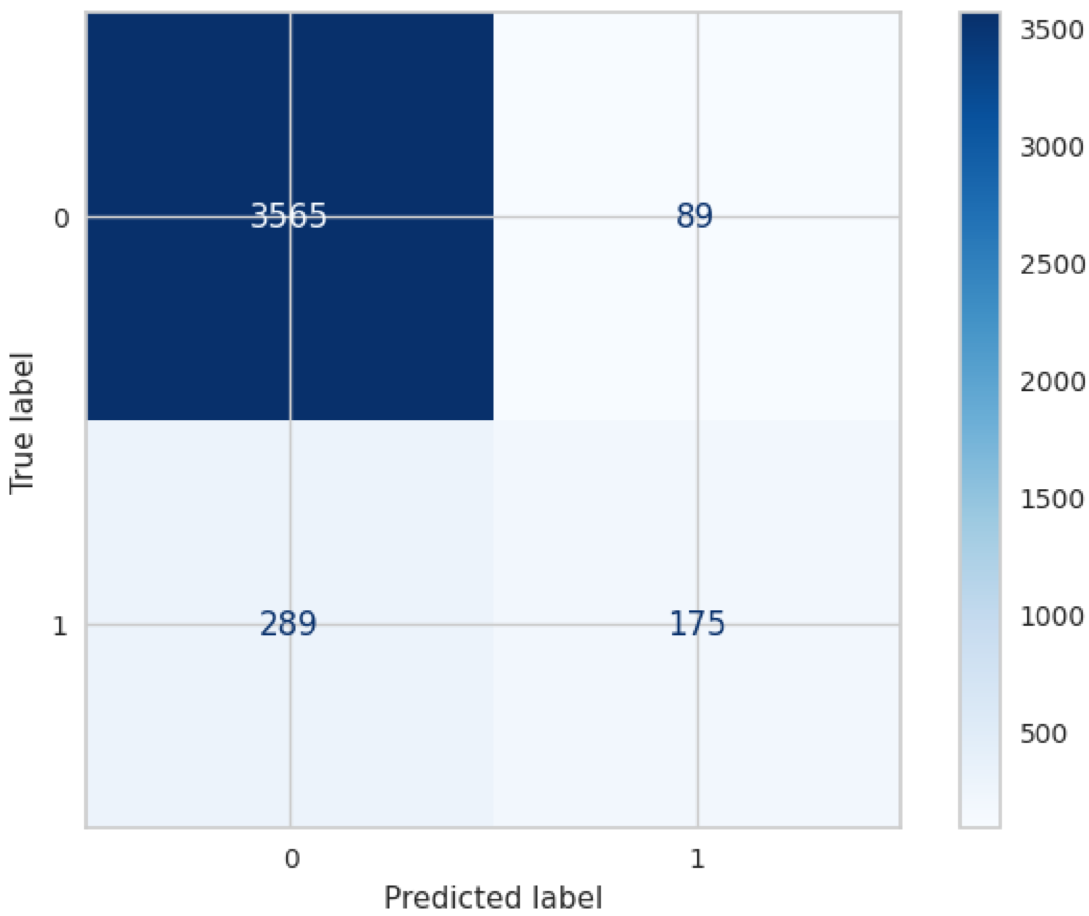


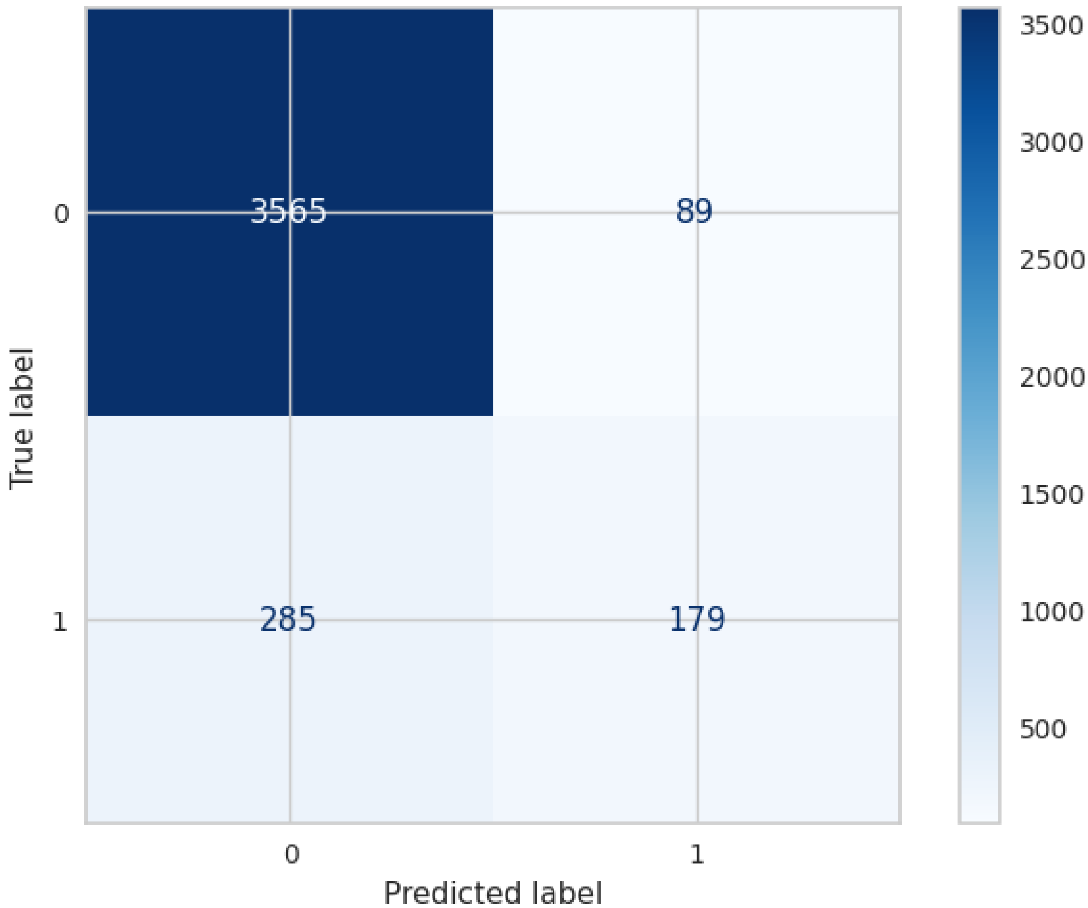

3568
86
302
162
0
1
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500

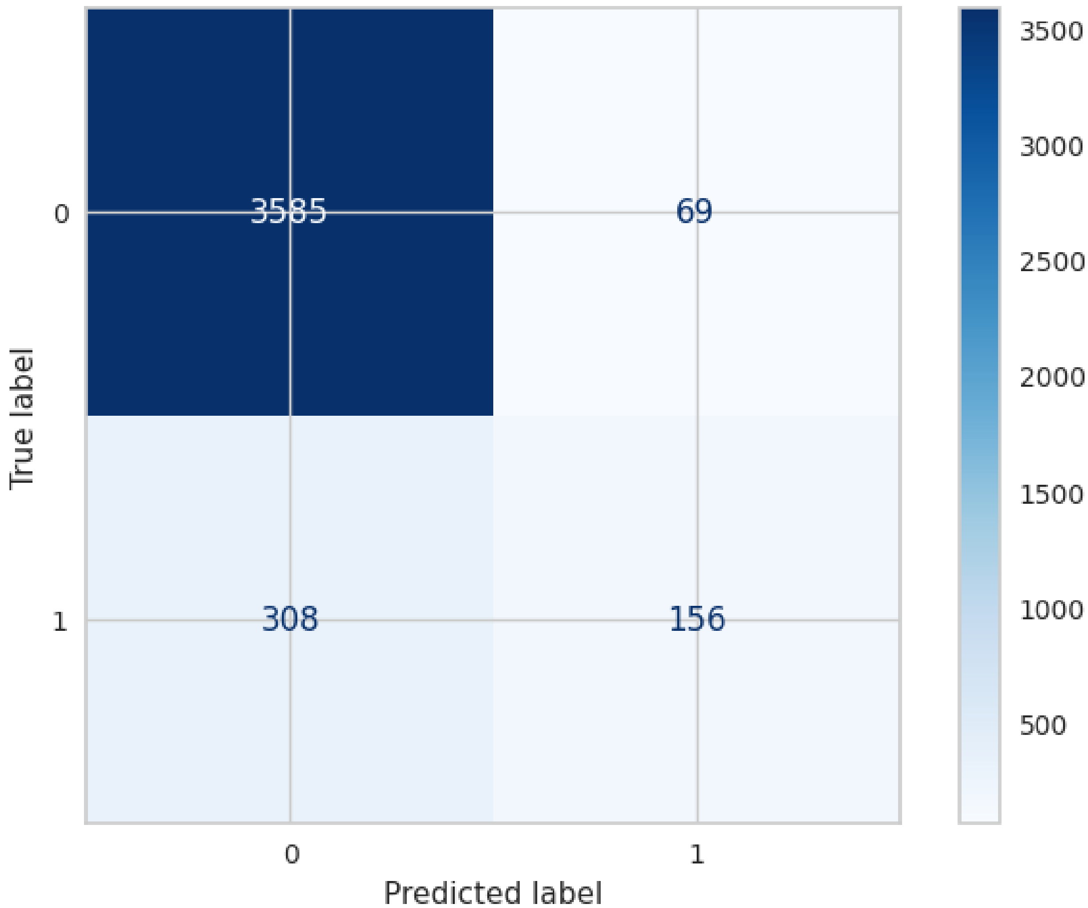


3500
3000
2500
2000
1500
1000
500
0
3565
89
1
285
179
True label
0
1
Predicted label


3585
69
308
156
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500


3565
89
289
175
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3565
89
285
179
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[283]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Logistic','Logistic With Feature','Logistic Scaling','Logistic␣
↪With Normalize','Logistic With PCA'
                ,'Logistic With PCA and Scaling',
                'Logistic With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[283]:
```
                                 Train Accuracy  Test Accuracy   Test F1  \
Models
Logistic                               0.908571       0.909179  0.481994
Logistic With Feature                  0.903174       0.905780  0.455056
Logistic Scaling                       0.908625       0.909179  0.489071
Logistic With Normalize                0.904631       0.908451  0.452830
Logistic With PCA                      0.908517       0.908208  0.480769
Logistic With PCA and Scaling          0.908625       0.909179  0.489071
Logistic With PCA and Normalize        0.904388       0.908451  0.452830
```

```
                                       Test Recall  Test Precision       AUC
Models
Logistic                                  0.375000        0.674419  0.676006
Logistic With Feature                     0.349138        0.653226  0.662801
Logistic Scaling                          0.385776        0.667910  0.680709
Logistic With Normalize                   0.336207        0.693333  0.658662
Logistic With PCA                         0.377155        0.662879  0.676399
Logistic With PCA and Scaling             0.385776        0.667910  0.680709
Logistic With PCA and Normalize           0.336207        0.693333  0.658662
```

[284]:
```
models_draw(df)
```

RandomOverSampler

[285]:
```
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[286]:
```
Search(LogisticRegression(penalty='l2',solver='sag',C=1.0),{'C':[1,.
 ↪5,2,3,5,10]},X_train,y_train)
```

[286]:
```
LogisticRegression(C=10, solver='sag')
```

[287]:
```
cross_validation(LogisticRegression(penalty='l2',solver='sag',C=1),X_train,y_train)
```

```
Train Score Value :  [0.86310587 0.86373313 0.86314389 0.86390679 0.8639448 ]
Mean 0.8635668965915606
Test Score Value :  [0.86489774 0.86124838 0.86664639 0.86306265 0.8613899 ]
Mean 0.8634490147123051
```

[288]:
```
Values =␣
 ↪Models(LogisticRegression(penalty='l2',solver='sag',C=1),X_train,y_train,X_test,y_test)
```

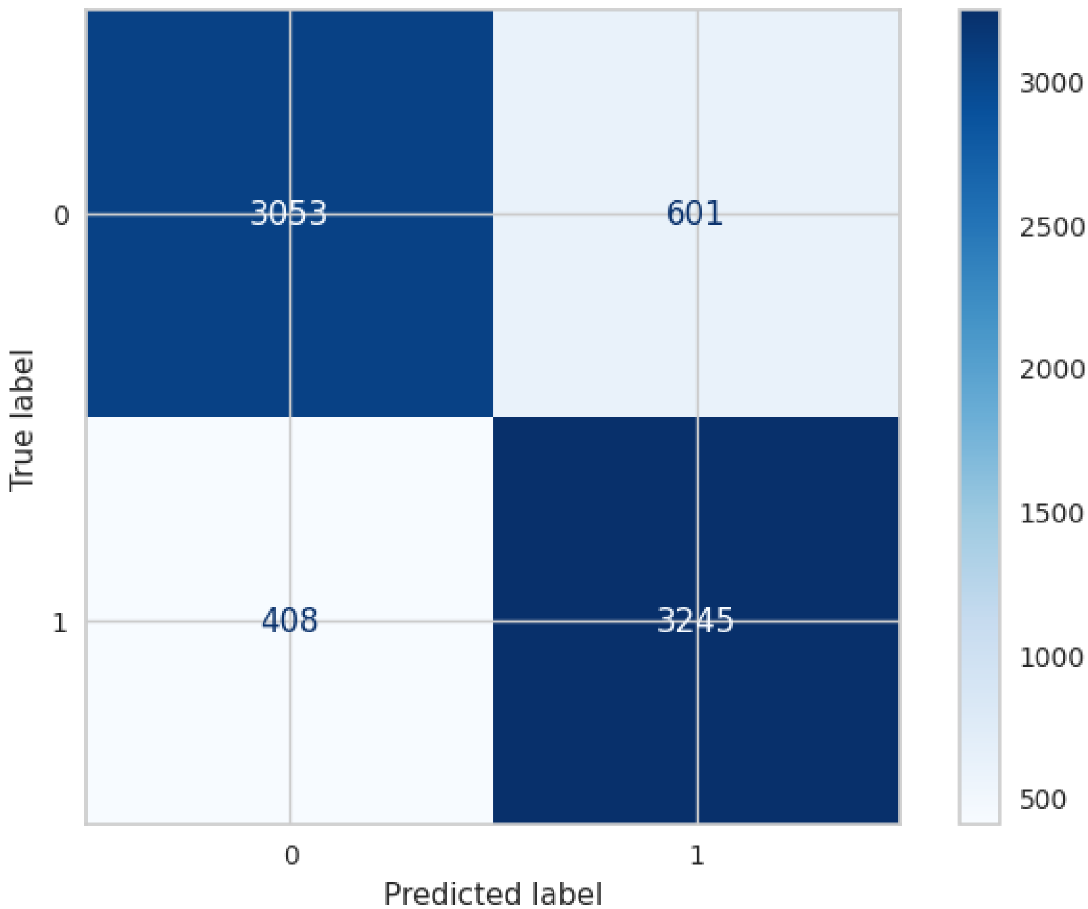
```
Apply Model With Normal Data :

Model Train Score is :  0.8636011130879065
Model Test Score is :  0.8619132338853155
F1 Score is :  0.8654487264968662
Recall Score is :  0.8883109772789488
Precision Score is :  0.843733749349974
AUC Value  :  0.8619168460560042

Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.84      0.86      3654
```

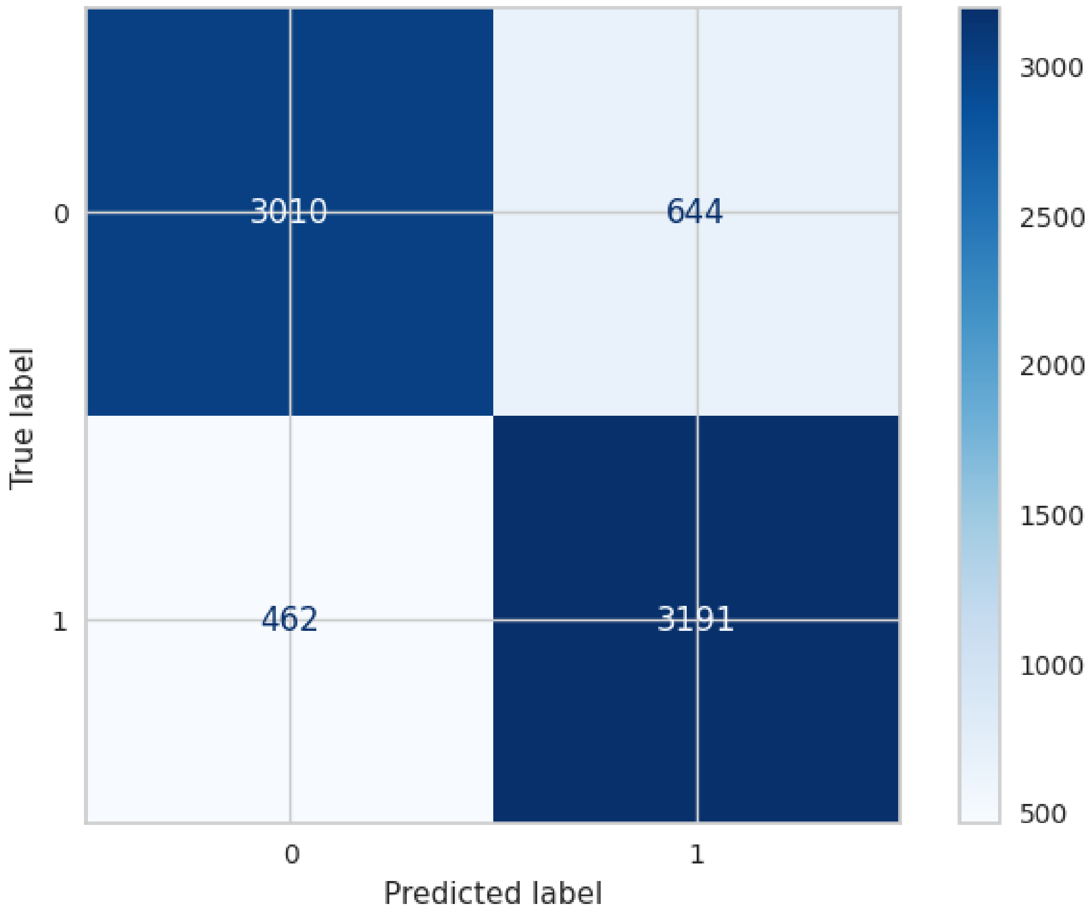
```
          1       0.84      0.89      0.87      3653

   accuracy                           0.86      7307
  macro avg       0.86      0.86      0.86      7307
weighted avg       0.86      0.86      0.86      7307

Confusion Matrix is :
 [[3053  601]
 [ 408 3245]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8501589039429467
Model Test Score is :  0.8486382920487204
F1 Score is :  0.8522970085470085
Recall Score is :  0.873528606624692
Precision Score is :  0.8320730117340287
AUC Value  :  0.8486416979483614
Classification Report is :                precision    recall  f1-score
support

          0       0.87      0.82      0.84      3654
          1       0.83      0.87      0.85      3653

   accuracy                           0.85      7307
  macro avg       0.85      0.85      0.85      7307
weighted avg       0.85      0.85      0.85      7307

Confusion Matrix is :
 [[3010  644]
 [ 462 3191]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8631753417575232
Model Test Score is :  0.861776378814835
F1 Score is :  0.8654768247202984
Recall Score is :  0.8894059676977827
Precision Score is :  0.8428015564202335
AUC Value  :  0.8617801595467567
Classification Report is :                precision    recall  f1-score
support

          0       0.88      0.83      0.86      3654
          1       0.84      0.89      0.87      3653
```

```
    accuracy                           0.86      7307
   macro avg       0.86      0.86      0.86      7307
weighted avg       0.86      0.86      0.86      7307

Confusion Matrix is :
 [[3048  606]
 [ 404 3249]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8430424402779678
Model Test Score is :  0.8349527850006843
F1 Score is :  0.8413157894736841
Recall Score is :  0.8751710922529428
Precision Score is :  0.809982265011401
AUC Value  :  0.8349582883267997
Classification Report is :                precision    recall  f1-score
support

           0       0.86      0.79      0.83      3654
           1       0.81      0.88      0.84      3653

    accuracy                           0.83      7307
   macro avg       0.84      0.83      0.83      7307
weighted avg       0.84      0.83      0.83      7307

Confusion Matrix is :
 [[2904  750]
 [ 456 3197]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8630536928059851
Model Test Score is :  0.8625975092377173
F1 Score is :  0.8661690215942415
Recall Score is :  0.8894059676977827
Precision Score is :  0.8441153546375681
AUC Value  :  0.862601177609154
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.84      0.86      3654
           1       0.84      0.89      0.87      3653
```

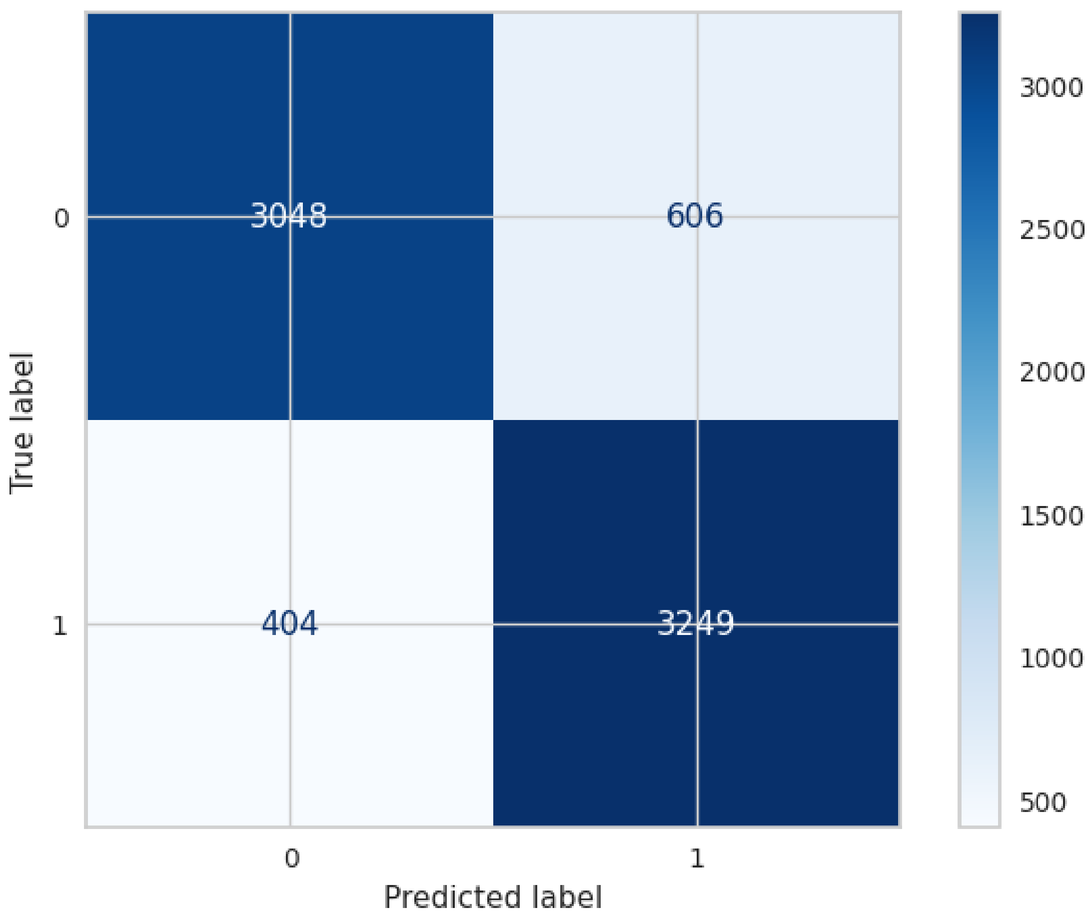
```
    accuracy                           0.86      7307
   macro avg       0.86      0.86      0.86      7307
weighted avg       0.86      0.86      0.86      7307

Confusion Matrix is :
 [[3054  600]
 [ 404 3249]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8631297234006965
Model Test Score is :  0.861776378814835
F1 Score is :  0.8654768247202984
Recall Score is :  0.8894059676977827
Precision Score is :  0.8428015564202335
AUC Value  :  0.8617801595467567
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.83      0.86      3654
           1       0.84      0.89      0.87      3653

    accuracy                           0.86      7307
   macro avg       0.86      0.86      0.86      7307
weighted avg       0.86      0.86      0.86      7307

Confusion Matrix is :
 [[3048  606]
 [ 404 3249]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8430424402779678
Model Test Score is :  0.835226495141645
F1 Score is :  0.8415372466438537
Recall Score is :  0.8751710922529428
Precision Score is :  0.8103929024081116
AUC Value  :  0.8352319610142656
Classification Report is :                precision    recall  f1-score
support

           0       0.86      0.80      0.83      3654
           1       0.81      0.88      0.84      3653

    accuracy                           0.84      7307
```

```
   macro avg       0.84      0.84      0.83      7307
weighted avg       0.84      0.84      0.83      7307

Confusion Matrix is :
 [[2906  748]
 [ 456 3197]]
```

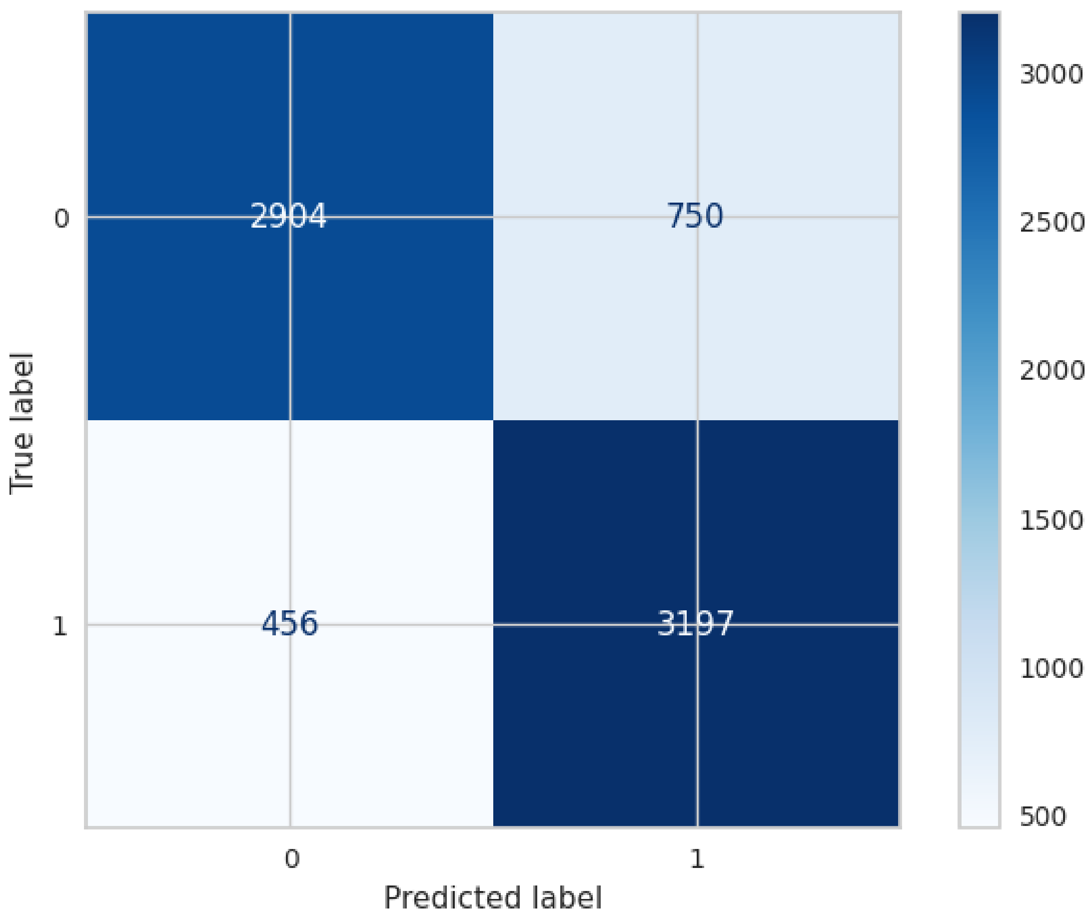


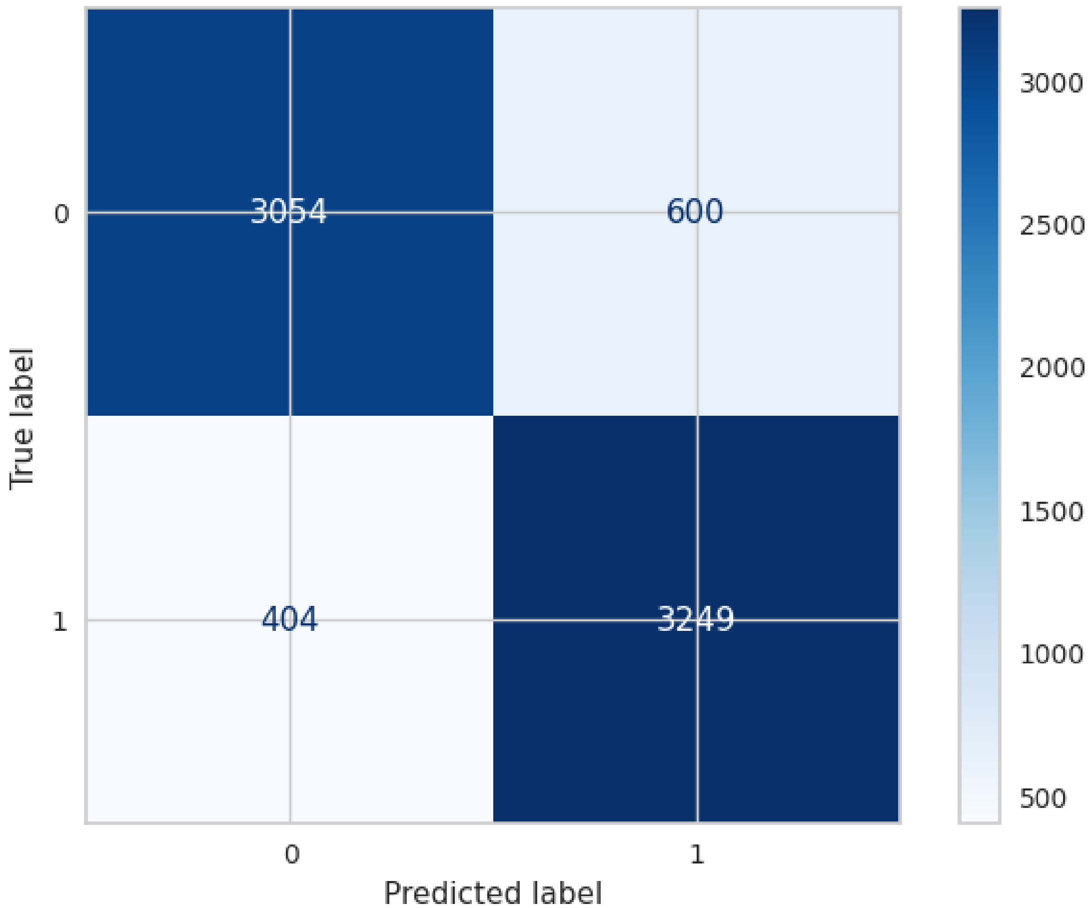


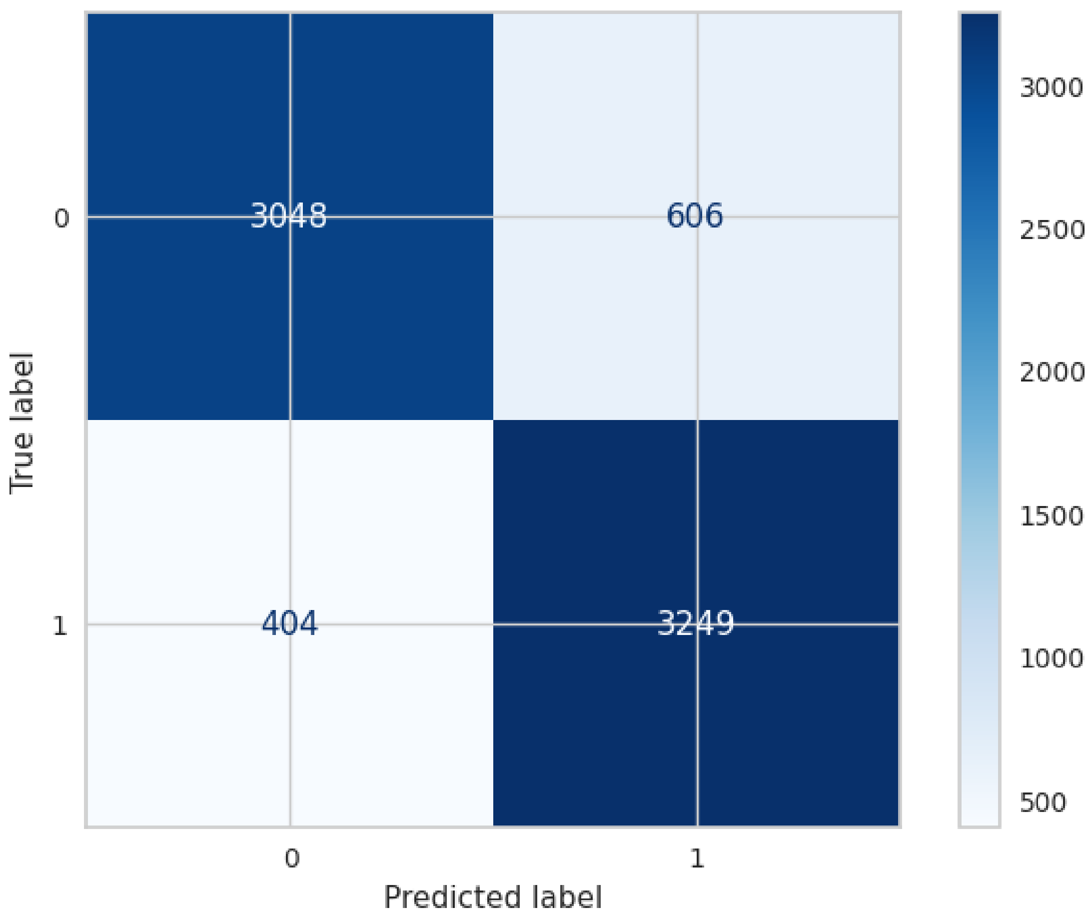
0
1
True label
3010
644
462
3191
0
1
Predicted label
3000
2500
2000
1500
1000
500

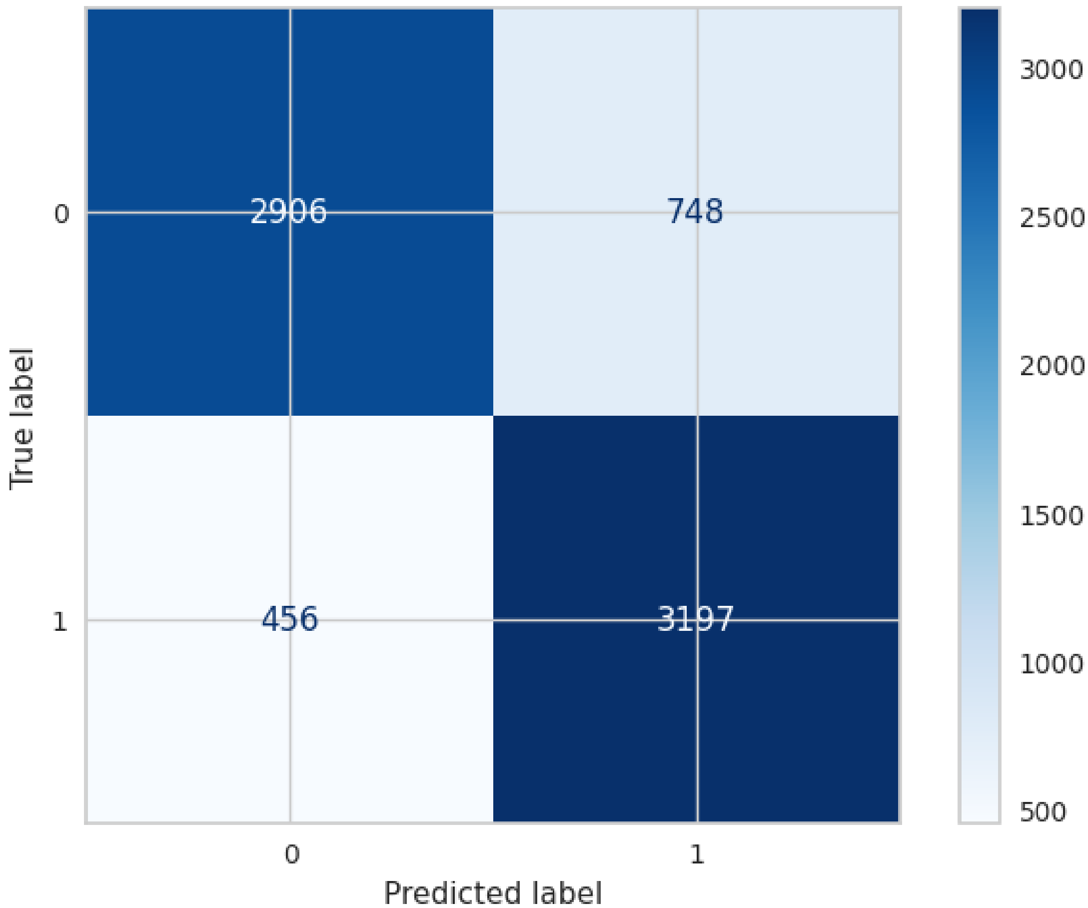


3048
606
404
3249
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


2904
750
456
3197
True label
Predicted label
0
1
0
1
3000
2500
2000
1500
1000
500


3054
600
404
3249
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


3048
606
404
3249
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500



[289]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Logistic Over','Logistic Over With Feature','Logistic Over␣
↪Scaling','Logistic Over With Normalize','Logistic Over With PCA'
                ,'Logistic Over With PCA and Scaling',
                'Logistic Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[289]:

```
                                    Train Accuracy  Test Accuracy   Test F1  \
Models
Logistic Over                             0.863601       0.861913  0.865449
Logistic Over With Feature                0.850159       0.848638  0.852297
Logistic Over Scaling                     0.863175       0.861776  0.865477
Logistic Over With Normalize              0.843042       0.834953  0.841316
Logistic Over With PCA                    0.863054       0.862598  0.866169
Logistic Over With PCA and Scaling        0.863130       0.861776  0.865477
Logistic Over With PCA and Normalize      0.843042       0.835226  0.841537
```

```
                                       Test Recall  Test Precision       AUC
Models
Logistic Over                             0.888311        0.843734  0.861917
Logistic Over With Feature                0.873529        0.832073  0.848642
Logistic Over Scaling                     0.889406        0.842802  0.861780
Logistic Over With Normalize              0.875171        0.809982  0.834958
Logistic Over With PCA                    0.889406        0.844115  0.862601
Logistic Over With PCA and Scaling        0.889406        0.842802  0.861780
Logistic Over With PCA and Normalize      0.875171        0.810393  0.835232
```

[290]:
```
models_draw(df)
```

RandomUnderSampler

[291]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[292]:
```
Search(LogisticRegression(penalty='l2',solver='sag',C=1.0),{'C':[1,.
 ↪5,2,3,5,10]},X_train,y_train)
```

[292]:
```
LogisticRegression(C=1, solver='sag')
```

[293]:
```
cross_validation(LogisticRegression(penalty='l2',solver='sag',C=10),X_train,y_train)
```

```
Train Score Value :  [0.85703593 0.85269461 0.85643713 0.85688623 0.85479042]
Mean 0.8555688622754491
Test Score Value :  [0.85149701 0.86107784 0.84730539 0.85748503 0.85628743]
Mean 0.8547305389221556
```

[294]:
```
Values =␣
 ↪Models(LogisticRegression(penalty='l2',solver='sag',C=10),X_train,y_train,X_test,y_test)
```

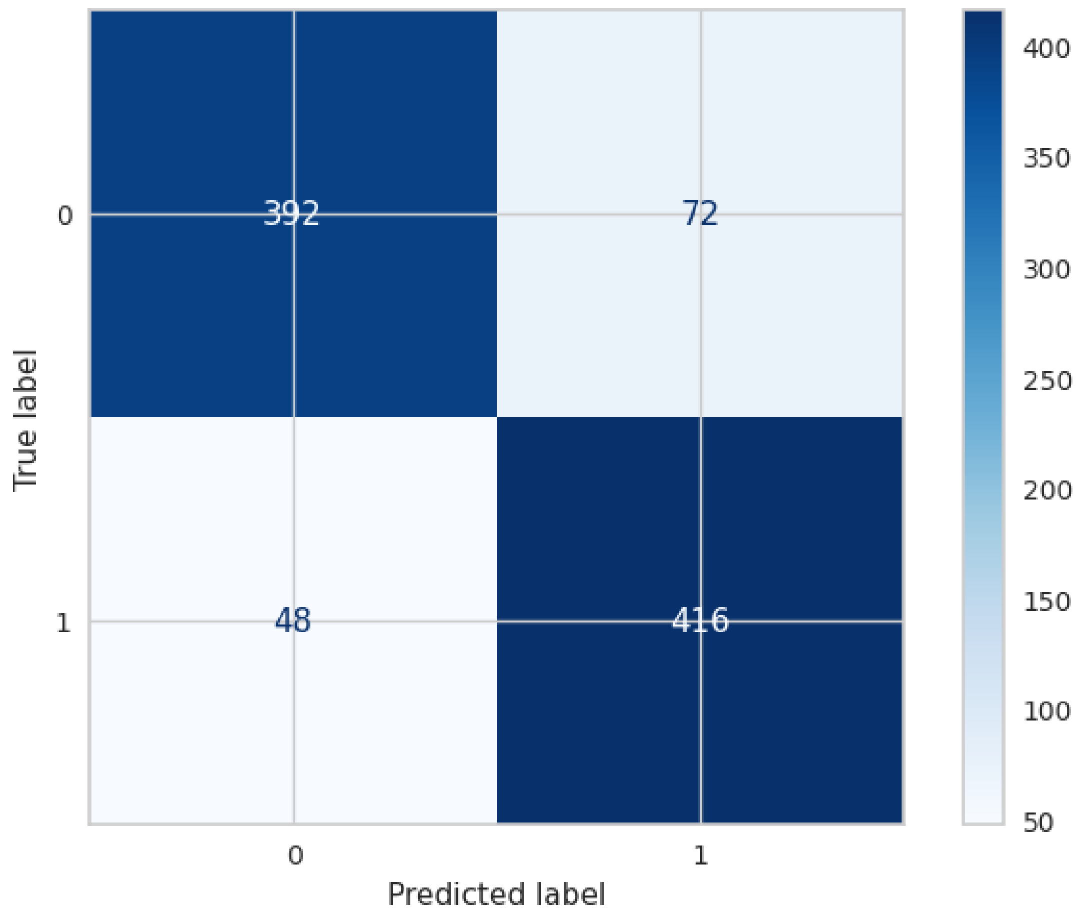
```
Apply Model With Normal Data :

Model Train Score is :  0.8562874251497006
Model Test Score is :  0.8706896551724138
F1 Score is :  0.8739495798319327
Recall Score is :  0.896551724137931
Precision Score is :  0.8524590163934426
AUC Value  :  0.8706896551724139

Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.84      0.87       464
```

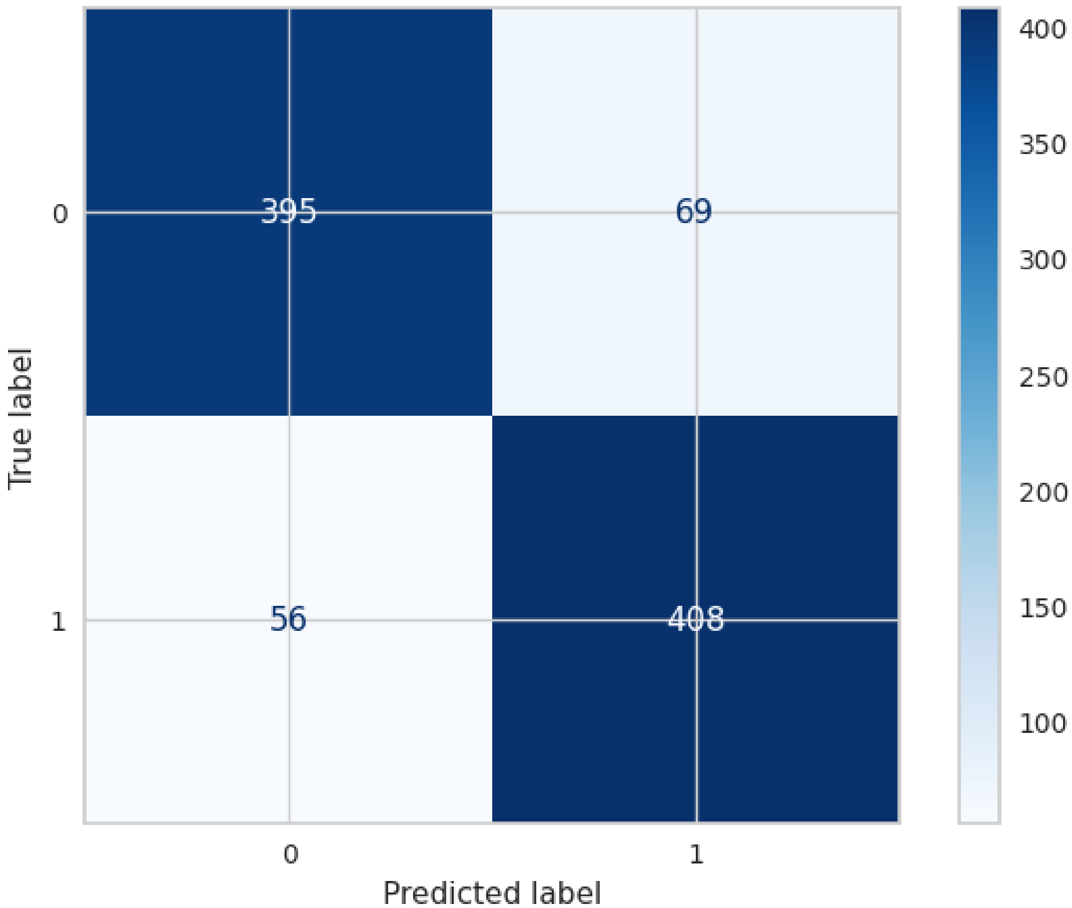
```
           1       0.85      0.90      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[392  72]
 [ 48 416]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8476646706586827
Model Test Score is :  0.865301724137931
F1 Score is :  0.8671625929861849
Recall Score is :  0.8793103448275862
Precision Score is :  0.8553459119496856
AUC Value  :  0.865301724137931
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.85      0.86       464
           1       0.86      0.88      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[395  69]
 [ 56 408]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8602395209580839
Model Test Score is :  0.8760775862068966
F1 Score is :  0.8783068783068783
Recall Score is :  0.8943965517241379
Precision Score is :  0.8627858627858628
AUC Value  :  0.8760775862068966
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.86      0.87       464
           1       0.86      0.89      0.88       464
```

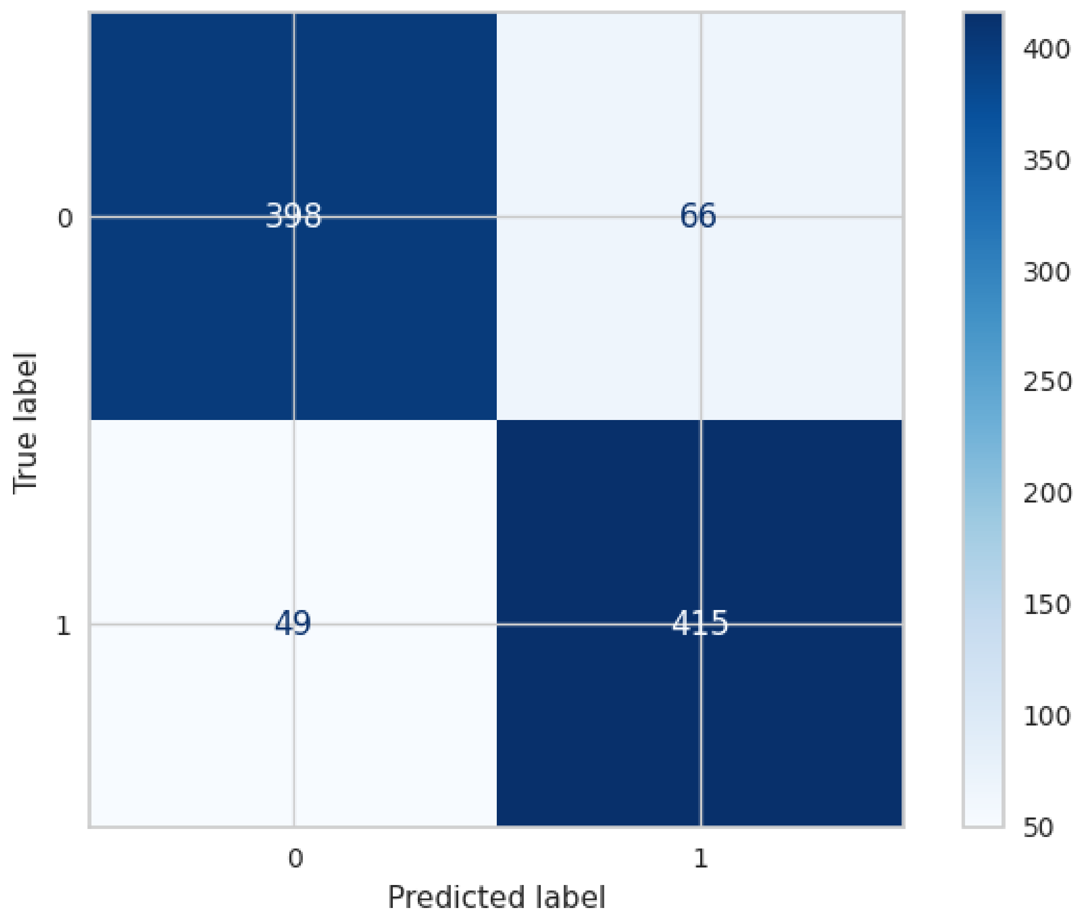
```
    accuracy                           0.88       928
   macro avg       0.88      0.88      0.88       928
weighted avg       0.88      0.88      0.88       928

Confusion Matrix is :
 [[398  66]
 [ 49 415]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8402395209580839
Model Test Score is :  0.853448275862069
F1 Score is :  0.8586278586278586
Recall Score is :  0.8900862068965517
Precision Score is :  0.8293172690763052
AUC Value  :  0.853448275862069
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.82      0.85       464
           1       0.83      0.89      0.86       464

    accuracy                           0.85       928
   macro avg       0.86      0.85      0.85       928
weighted avg       0.86      0.85      0.85       928

Confusion Matrix is :
 [[379  85]
 [ 51 413]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8583233532934131
Model Test Score is :  0.875
F1 Score is :  0.8778947368421053
Recall Score is :  0.8987068965517241
Precision Score is :  0.8580246913580247
AUC Value  :  0.875
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.85      0.87       464
           1       0.86      0.90      0.88       464
```

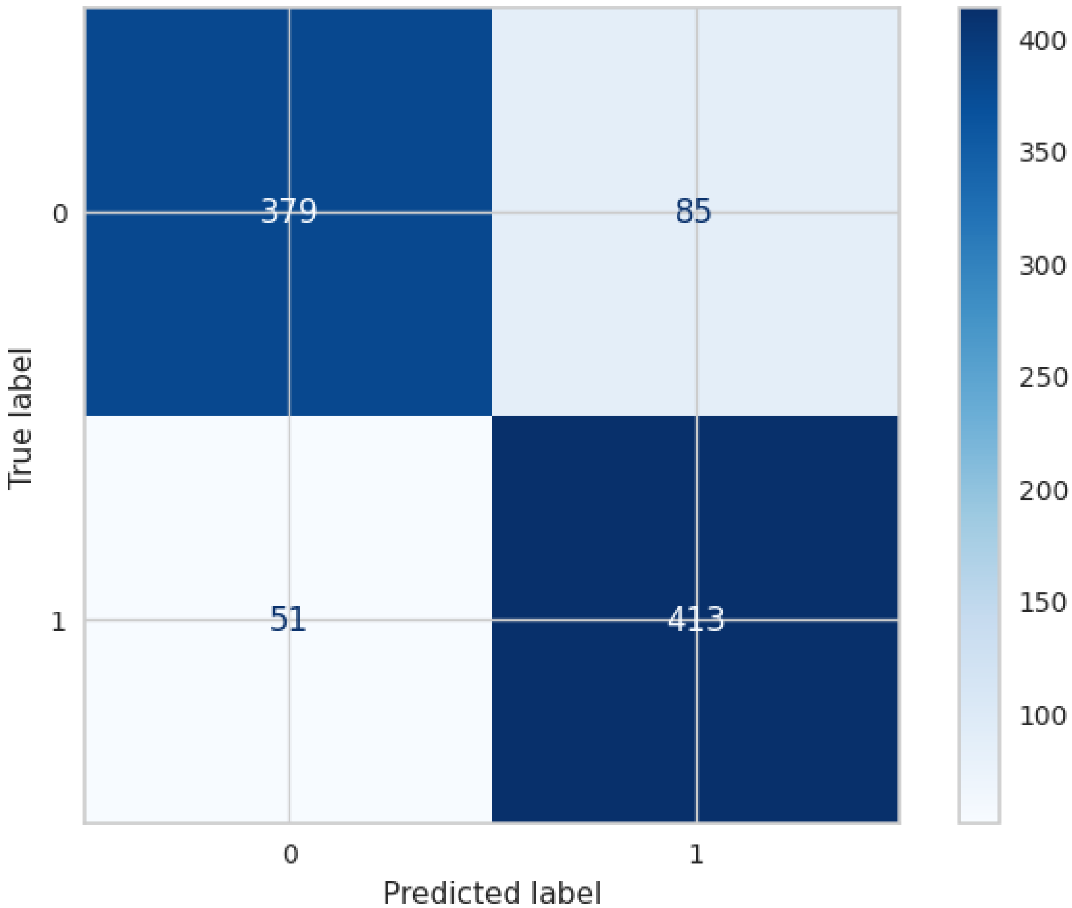
```
    accuracy                           0.88       928
   macro avg       0.88      0.88      0.87       928
weighted avg       0.88      0.88      0.87       928

Confusion Matrix is :
 [[395  69]
 [ 47 417]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8602395209580839
Model Test Score is :  0.8760775862068966
F1 Score is :  0.8783068783068783
Recall Score is :  0.8943965517241379
Precision Score is :  0.8627858627858628
AUC Value  :  0.8760775862068966
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.86      0.87       464
           1       0.86      0.89      0.88       464

    accuracy                           0.88       928
   macro avg       0.88      0.88      0.88       928
weighted avg       0.88      0.88      0.88       928

Confusion Matrix is :
 [[398  66]
 [ 49 415]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8402395209580839
Model Test Score is :  0.853448275862069
F1 Score is :  0.8586278586278586
Recall Score is :  0.8900862068965517
Precision Score is :  0.8293172690763052
AUC Value  :  0.853448275862069
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.82      0.85       464
           1       0.83      0.89      0.86       464

    accuracy                           0.85       928
```

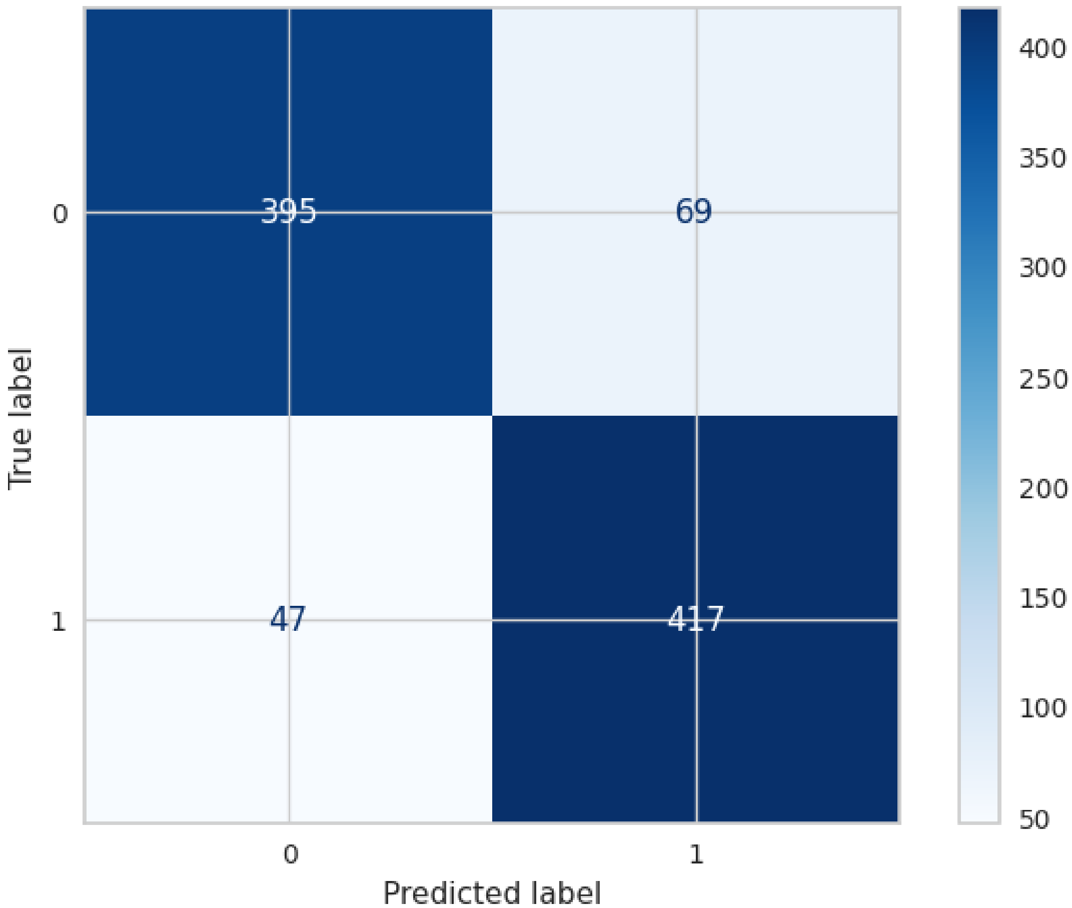
```
   macro avg       0.86      0.85      0.85       928
weighted avg       0.86      0.85      0.85       928

Confusion Matrix is :
 [[379  85]
 [ 51 413]]
```

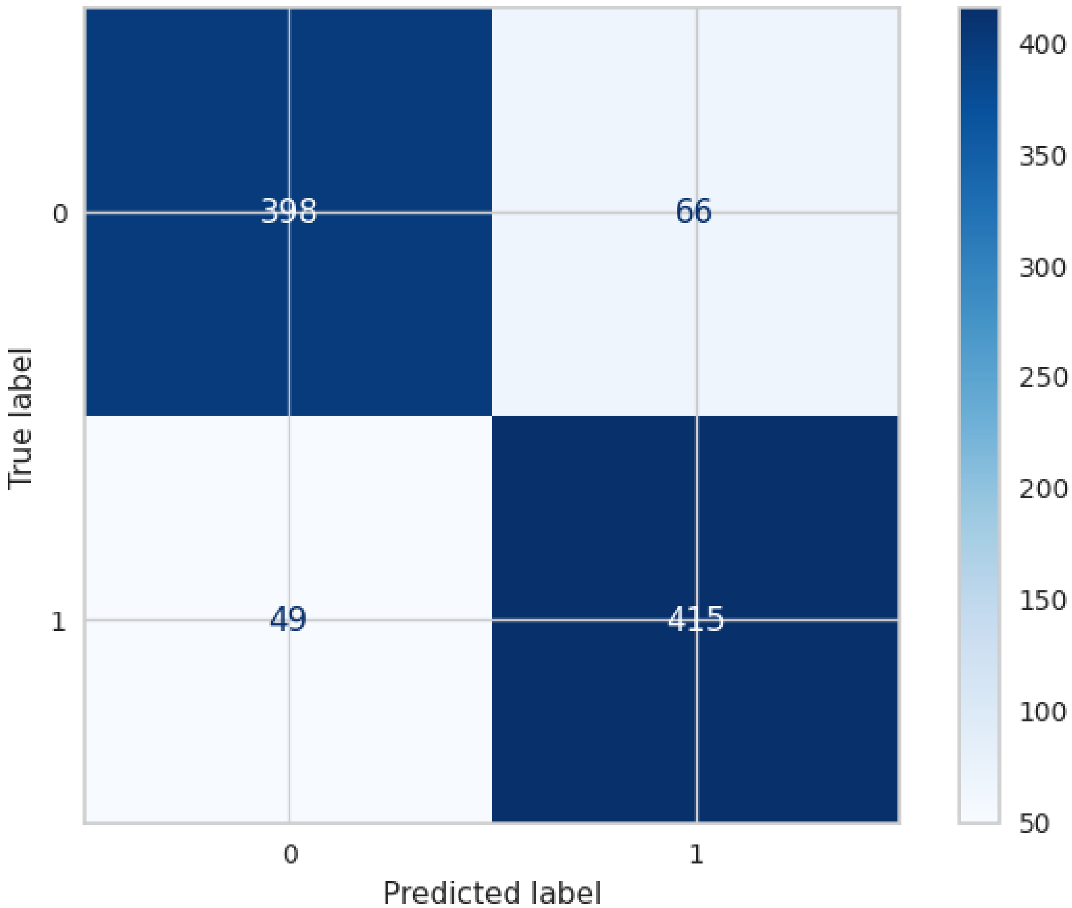



395
69
56
408
0
1
0
1
True label
Predicted label
400
350
300
250
200
150
100

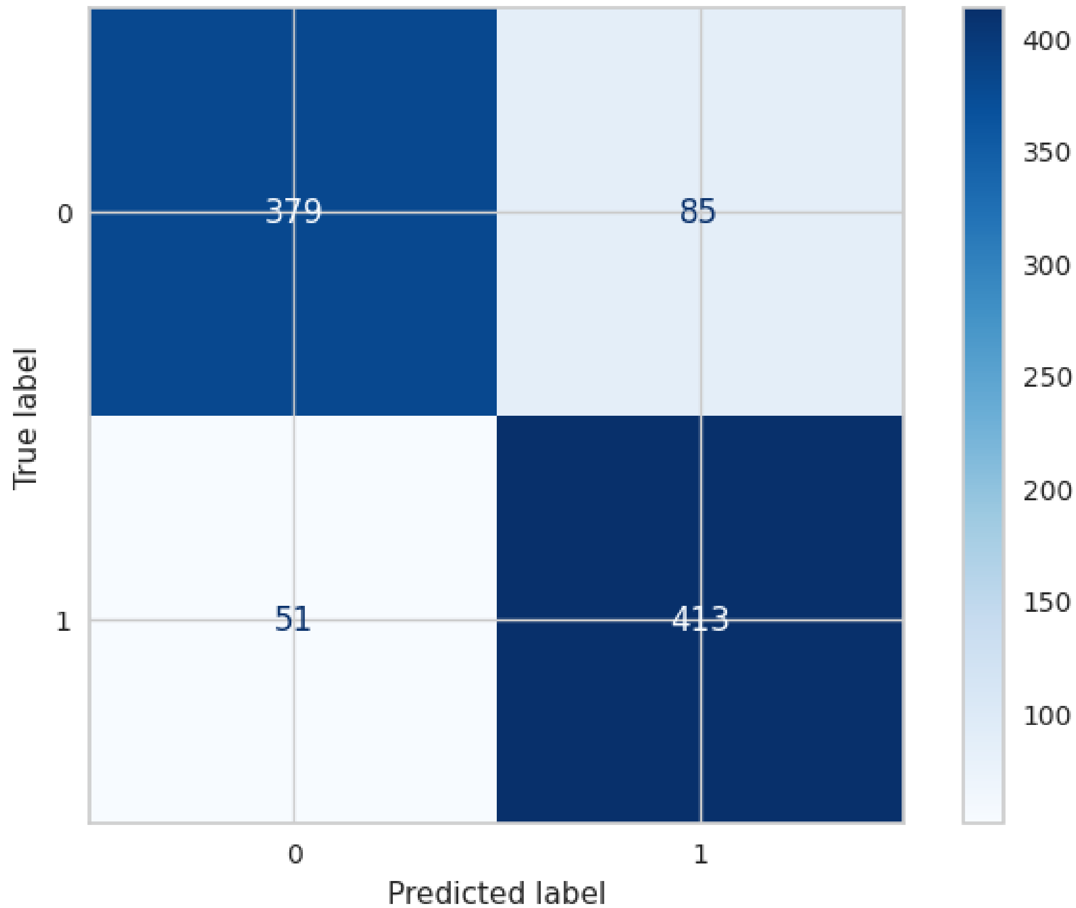

398
66
49
415
True label
Predicted label
0
1
0
1
400
350
300
250
200
150
100
50


379
85
51
413
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100


395
69
47
417
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50



398
66
49
415
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50



[295]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Logistic Under','Logistic Under With Feature','Logistic Under␣
↪Scaling','Logistic Under With Normalize','Logistic Under With PCA'
                ,'Logistic Under With PCA and Scaling',
                'Logistic Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[295]:

Models	Train Accuracy	Test Accuracy \
Logistic Under	0.856287	0.870690
Logistic Under With Feature	0.847665	0.865302
Logistic Under Scaling	0.860240	0.876078
Logistic Under With Normalize	0.840240	0.853448
Logistic Under With PCA	0.858323	0.875000
Logistic Under With PCA and Scaling	0.860240	0.876078
Logistic Under With PCA and Normalize	0.840240	0.853448

```
                                     Test F1  Test Recall  Test Precision  \
Models
Logistic Under                       0.873950     0.896552        0.852459
Logistic Under With Feature          0.867163     0.879310        0.855346
Logistic Under Scaling               0.878307     0.894397        0.862786
Logistic Under With Normalize        0.858628     0.890086        0.829317
Logistic Under With PCA              0.877895     0.898707        0.858025
Logistic Under With PCA and Scaling  0.878307     0.894397        0.862786
Logistic Under With PCA and Normalize 0.858628    0.890086        0.829317

                                          AUC
Models
Logistic Under                       0.870690
Logistic Under With Feature          0.865302
Logistic Under Scaling               0.876078
Logistic Under With Normalize        0.853448
Logistic Under With PCA              0.875000
Logistic Under With PCA and Scaling  0.876078
Logistic Under With PCA and Normalize 0.853448
```

[296]:
```
models_draw(df)
```

GaussianNB

[297]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[298]:
```
cross_validation(GaussianNB(),X_train,y_train)
```

```
Train Score Value :  [0.83946161 0.84091752 0.84358239 0.85208298 0.83973689]
Mean 0.8431562790461198
Test Score Value :  [0.83904479 0.83969775 0.84347591 0.84995277 0.84172176]
Mean 0.8427785981704972
```

[299]:
```
Values = Models(GaussianNB(),X_train,y_train,X_test,y_test)
```

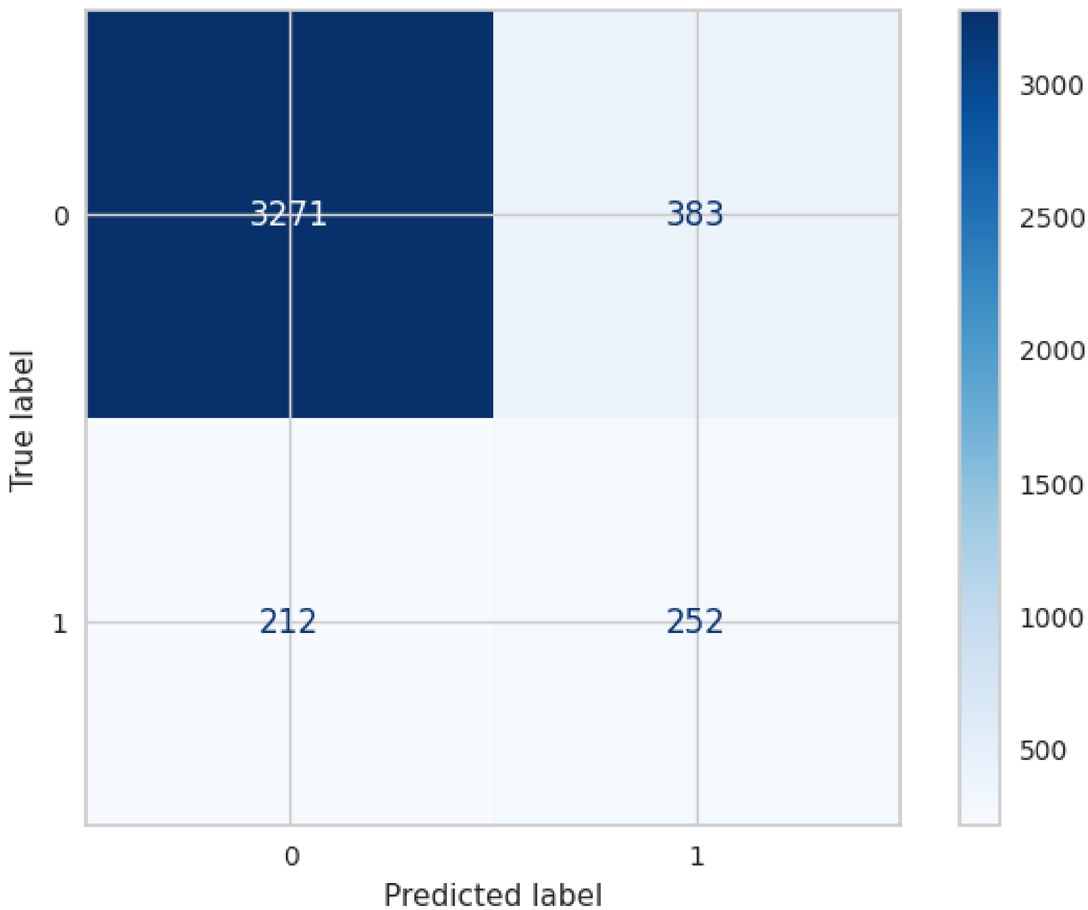
```
Apply Model With Normal Data :

Model Train Score is :  0.8424006908462867
Model Test Score is :  0.855512384652744
F1 Score is :  0.4585987261146497
Recall Score is :  0.5431034482758621
Precision Score is :  0.3968503937007874
AUC Value  :  0.7191434044882321
```

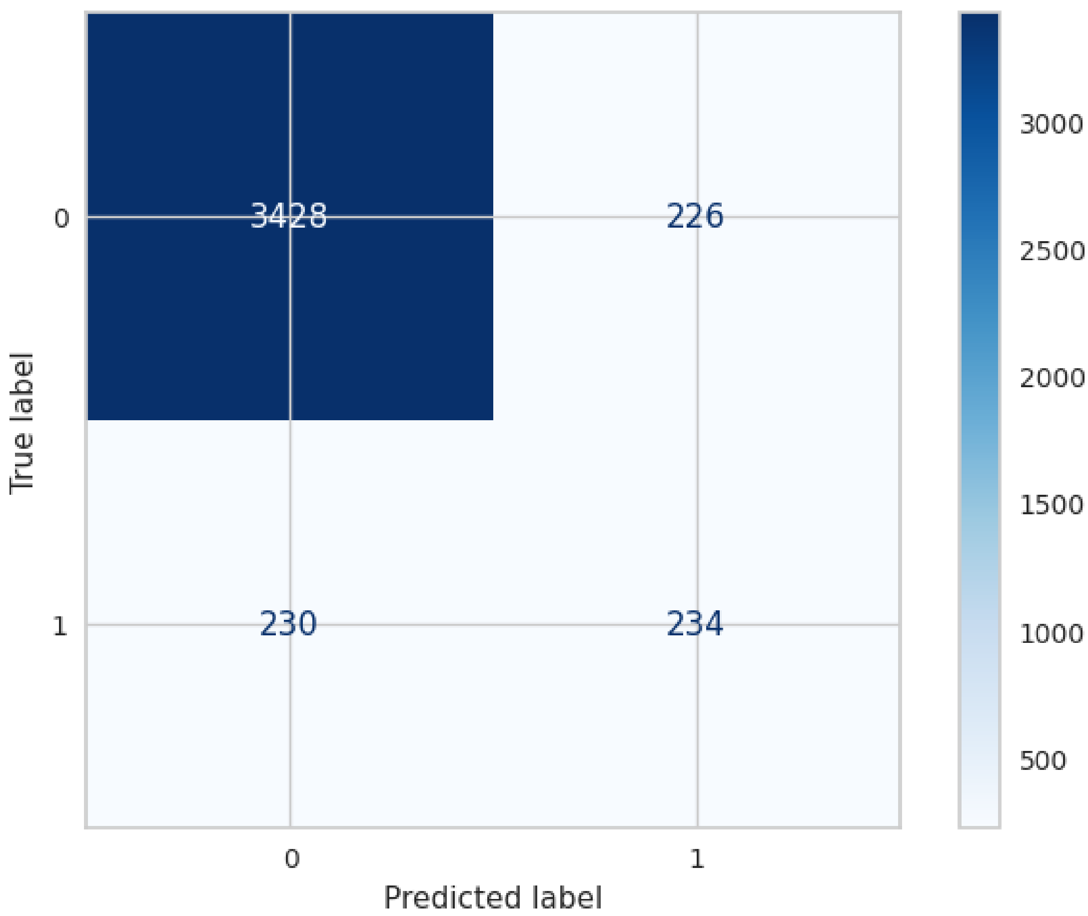
```
Classification Report is :               precision    recall  f1-score
support

           0       0.94      0.90      0.92      3654
           1       0.40      0.54      0.46       464

    accuracy                           0.86      4118
   macro avg       0.67      0.72      0.69      4118
weighted avg       0.88      0.86      0.87      4118

Confusion Matrix is :
 [[3271  383]
 [ 212  252]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8853626943005182
Model Test Score is :  0.8892666342884895
F1 Score is :  0.5064935064935064
Recall Score is :  0.5043103448275862
Precision Score is :  0.508695652173913
AUC Value  :  0.7212301587301588
Classification Report is :               precision    recall  f1-score
support

           0       0.94      0.94      0.94      3654
           1       0.51      0.50      0.51       464

    accuracy                           0.89      4118
   macro avg       0.72      0.72      0.72      4118
weighted avg       0.89      0.89      0.89      4118

Confusion Matrix is :
 [[3428  226]
 [ 230  234]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.7029900690846287
Model Test Score is :  0.7102962603205439
F1 Score is :  0.406172224987556
Recall Score is :  0.8793103448275862
Precision Score is :  0.26407766990291265
AUC Value  :  0.784072249589491
Classification Report is :               precision    recall  f1-score
```

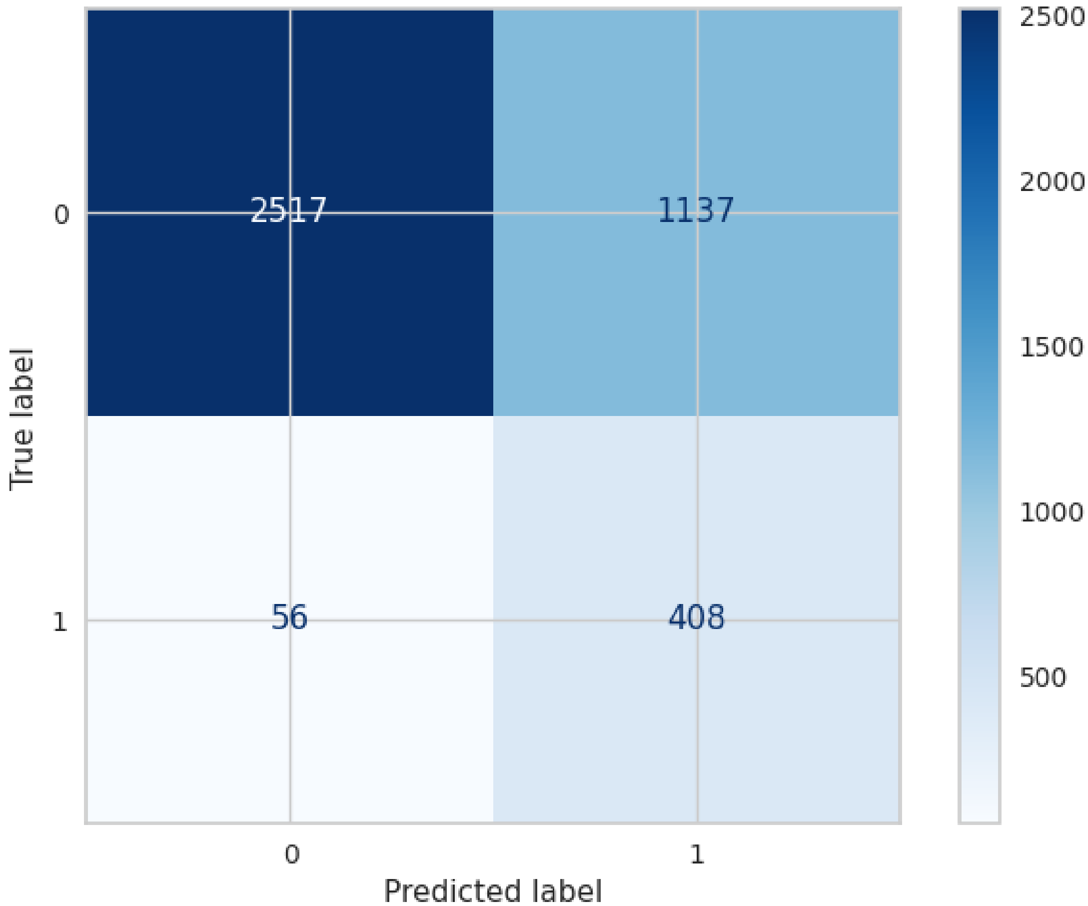
```
support

           0       0.98      0.69      0.81      3654
           1       0.26      0.88      0.41       464

    accuracy                           0.71      4118
   macro avg       0.62      0.78      0.61      4118
weighted avg       0.90      0.71      0.76      4118

Confusion Matrix is :
 [[2517 1137]
 [  56  408]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8132286269430051
Model Test Score is :  0.8142302088392424
F1 Score is :  0.5215759849906191
Recall Score is :  0.8987068965517241
Precision Score is :  0.3674008810572687
AUC Value  :  0.8511049534756432

Classification Report is :                precision    recall  f1-score
support

           0       0.98      0.80      0.88      3654
           1       0.37      0.90      0.52       464

    accuracy                           0.81      4118
   macro avg       0.68      0.85      0.70      4118
weighted avg       0.91      0.81      0.84      4118

Confusion Matrix is :
 [[2936  718]
 [  47  417]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8778875215889465
Model Test Score is :  0.8827100534239922
F1 Score is :  0.4627363737486096
Recall Score is :  0.4482758620689655
Precision Score is :  0.4781609195402299
AUC Value  :  0.6930760810071155

Classification Report is :                precision    recall  f1-score
support
```

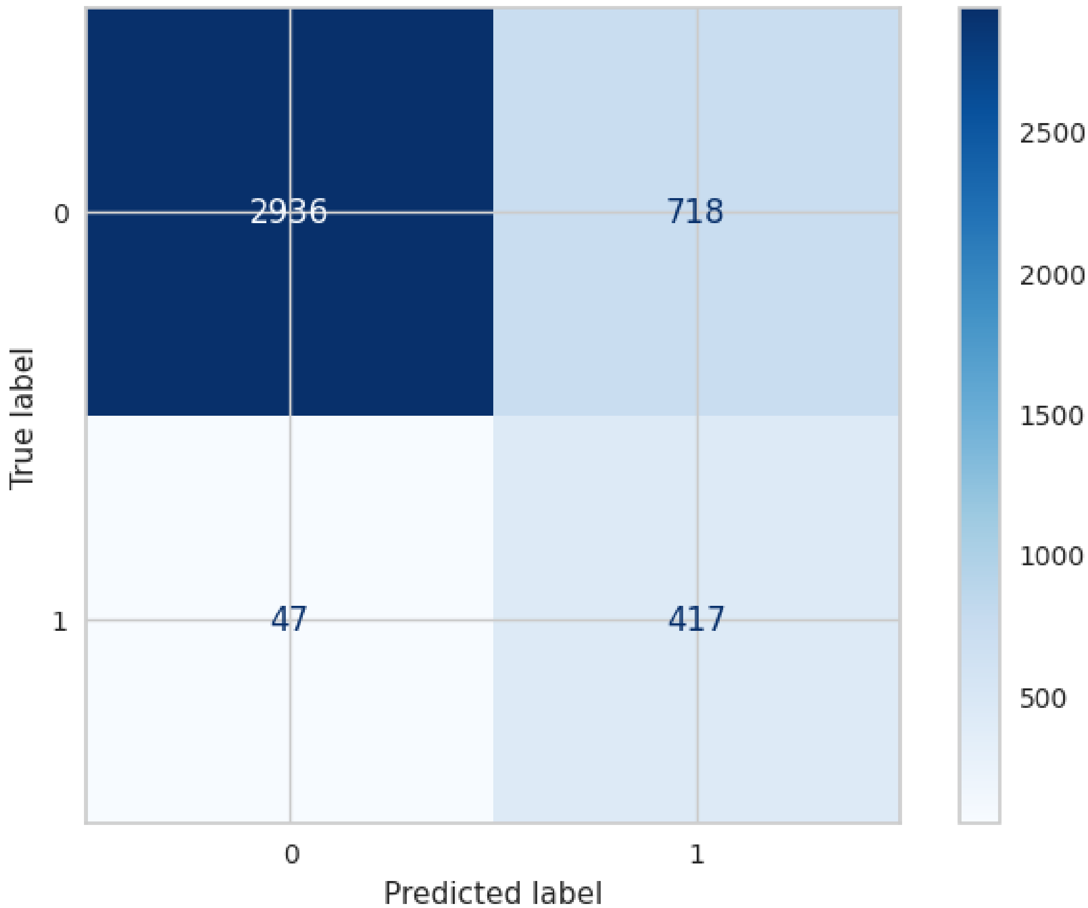
```
           0       0.93      0.94      0.93      3654
           1       0.48      0.45      0.46       464

    accuracy                           0.88      4118
   macro avg       0.70      0.69      0.70      4118
weighted avg       0.88      0.88      0.88      4118

Confusion Matrix is :
 [[3427  227]
 [ 256  208]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8537618739205527
Model Test Score is :  0.861583292860612
F1 Score is :  0.5086206896551724
Recall Score is :  0.6357758620689655
Precision Score is :  0.4238505747126437
AUC Value  :  0.7630165571975918
Classification Report is :               precision    recall  f1-score
support

           0       0.95      0.89      0.92      3654
           1       0.42      0.64      0.51       464

    accuracy                           0.86      4118
   macro avg       0.69      0.76      0.71      4118
weighted avg       0.89      0.86      0.87      4118

Confusion Matrix is :
 [[3253  401]
 [ 169  295]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8816655872193437
Model Test Score is :  0.8858669256920836
F1 Score is :  0.5164609053497943
Recall Score is :  0.540948275862069
Precision Score is :  0.4940944881889764
AUC Value  :  0.7353071975916803
Classification Report is :               precision    recall  f1-score
support
```

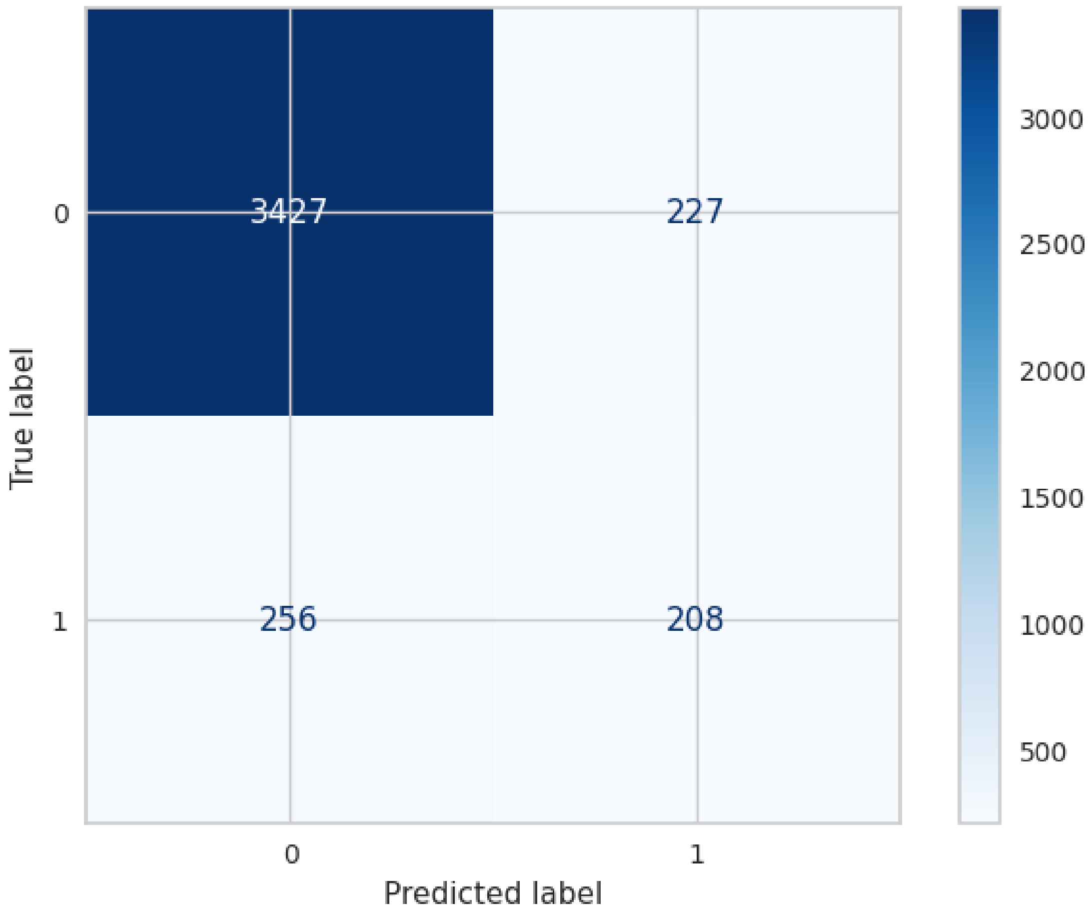
```
           0       0.94      0.93      0.94      3654
           1       0.49      0.54      0.52       464

    accuracy                           0.89      4118
   macro avg       0.72      0.74      0.73      4118
weighted avg       0.89      0.89      0.89      4118

Confusion Matrix is :
 [[3397  257]
 [ 213  251]]
```

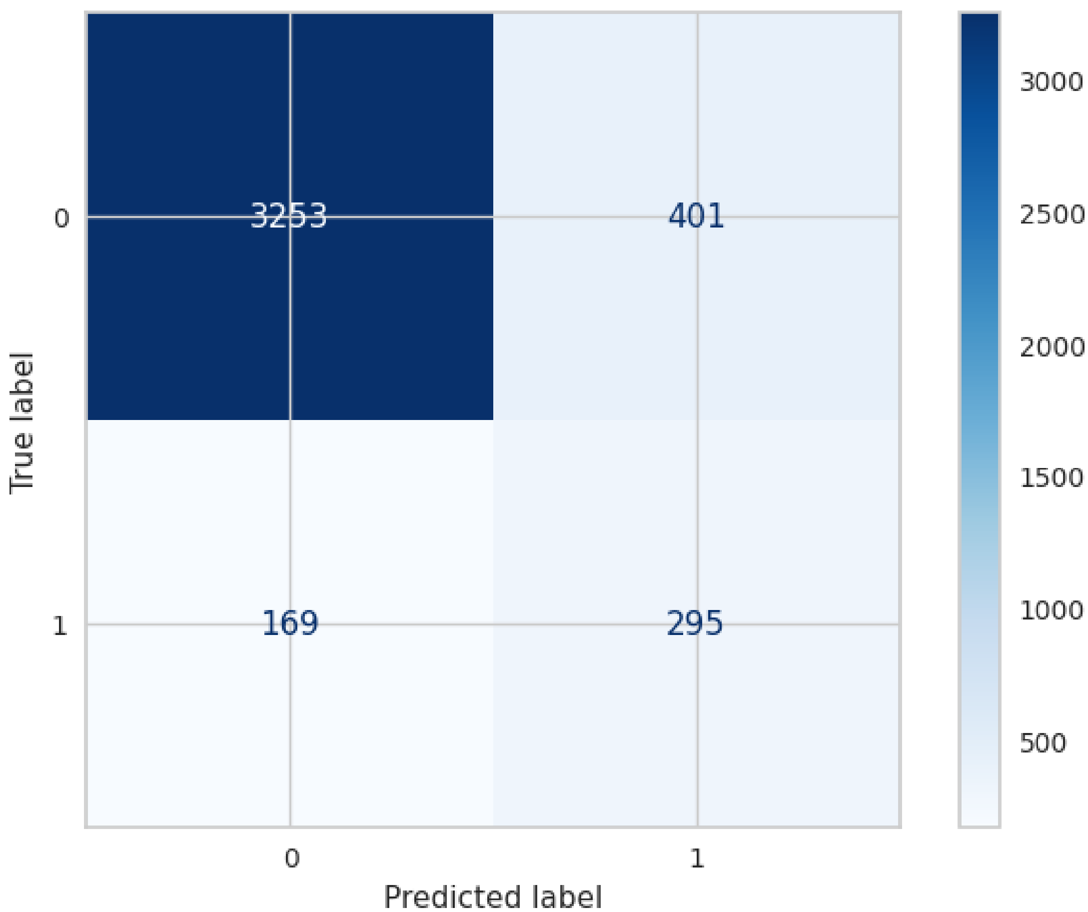


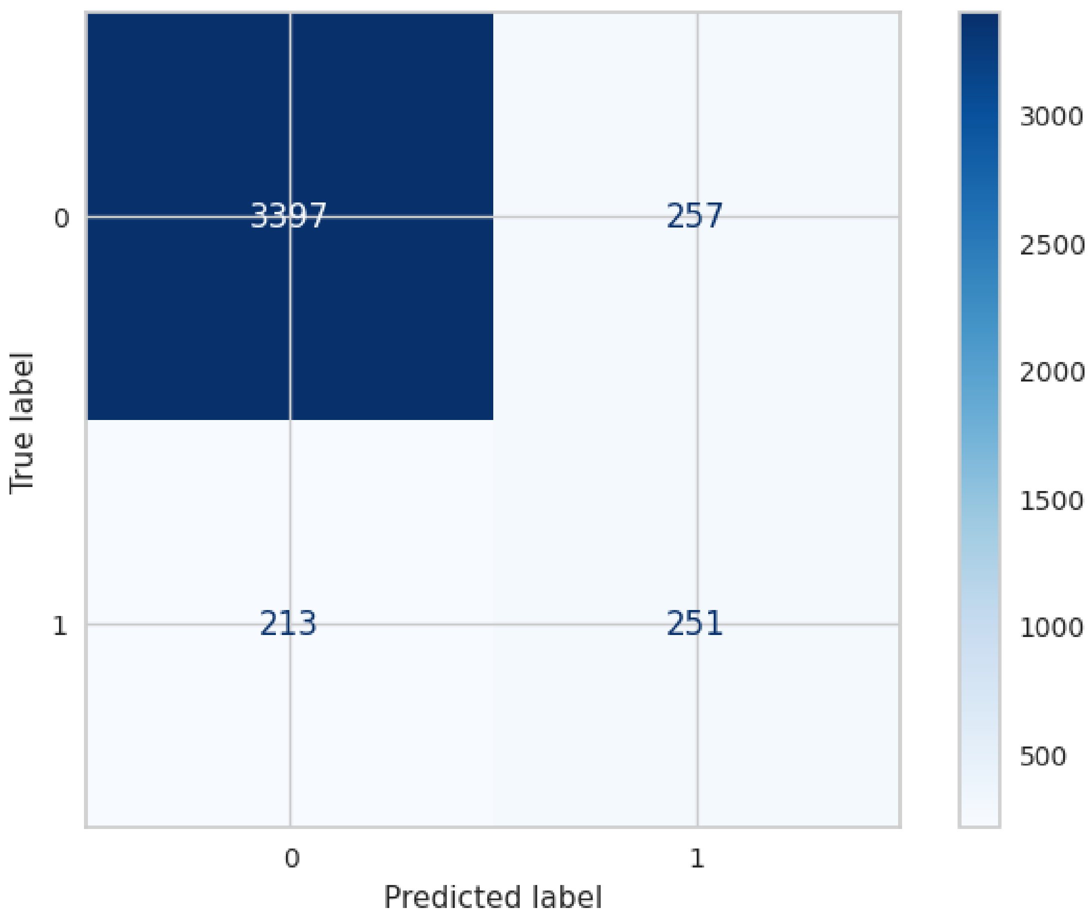

3428
226
230
234
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


2517
1137
56
408
0
1
True label
0
1
Predicted label
2500
2000
1500
1000
500


2936
718
47
417
0
1
True label
0
1
Predicted label
2500
2000
1500
1000
500


3427
227
256
208
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


3253
401
169
295
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500



[300]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['NB','NB With Feature','NB Scaling','NB With Normalize','NB␣
↪With PCA'
                ,'NB With PCA and Scaling',
                'NB With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[300]:

Models	Train Accuracy	Test Accuracy	Test F1 \
NB	0.842401	0.855512	0.458599
NB With Feature	0.885363	0.889267	0.506494
NB Scaling	0.702990	0.710296	0.406172
NB With Normalize	0.813229	0.814230	0.521576
NB With PCA	0.877888	0.882710	0.462736
NB With PCA and Scaling	0.853762	0.861583	0.508621
NB With PCA and Normalize	0.881666	0.885867	0.516461

```
                          Test Recall  Test Precision       AUC
Models
NB                           0.543103        0.396850  0.719143
NB With Feature              0.504310        0.508696  0.721230
NB Scaling                   0.879310        0.264078  0.784072
NB With Normalize            0.898707        0.367401  0.851105
NB With PCA                  0.448276        0.478161  0.693076
NB With PCA and Scaling      0.635776        0.423851  0.763017
NB With PCA and Normalize    0.540948        0.494094  0.735307
```

[301]:
```
models_draw(df)
```

RandomOverSampler

[302]:
```
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[303]:
```
cross_validation(GaussianNB(),X_train,y_train)
```

```
Train Score Value :  [0.73259837 0.73402395 0.73497434 0.73319268 0.73401   ]
Mean 0.733759866174962
Test Score Value :  [0.73511746 0.73777845 0.73329278 0.73289234 0.73000304]
Mean 0.7338168158652342
```

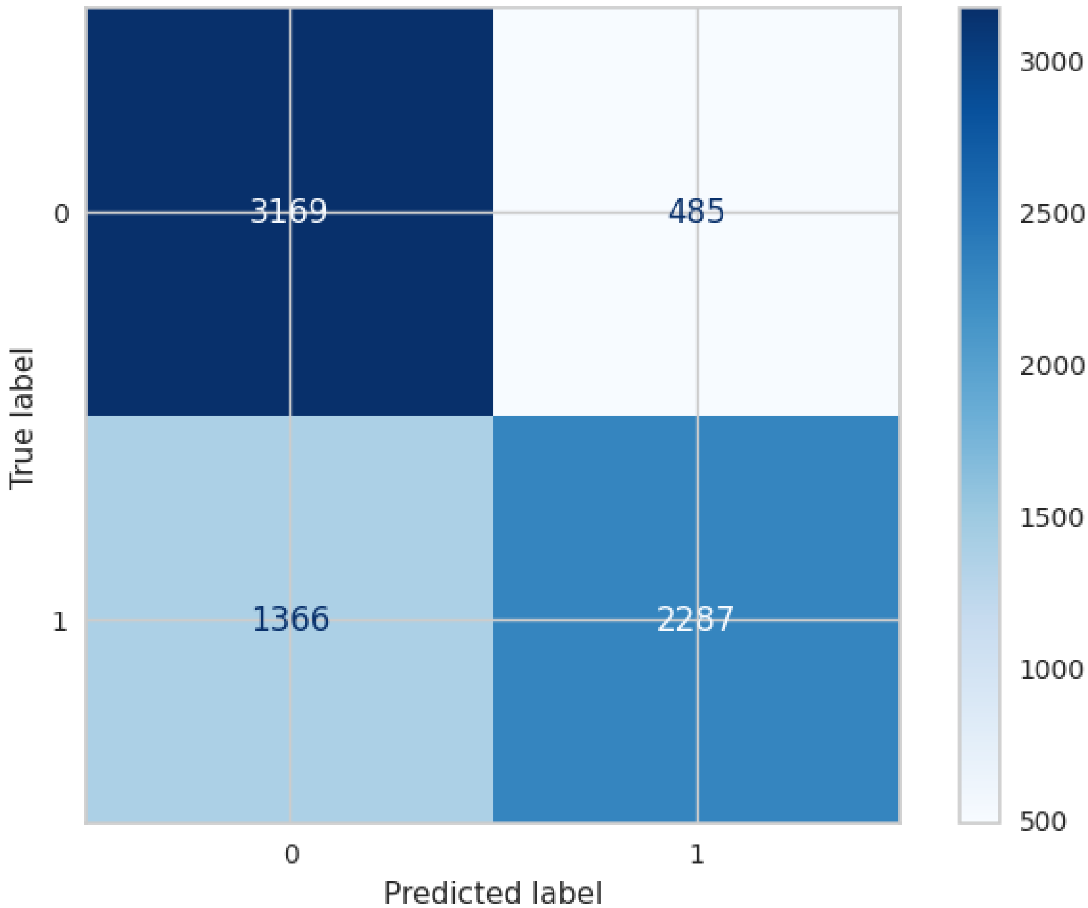
[304]:
```
Values = Models(GaussianNB(),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

Model Train Score is :  0.7336648267262746
Model Test Score is :  0.7466812645408513
F1 Score is :  0.7119066147859923
Recall Score is :  0.6260607719682453
Precision Score is :  0.825036075036075
AUC Value  :  0.7466647592736683

Classification Report is :                precision    recall  f1-score
support

           0       0.70      0.87      0.77      3654
           1       0.83      0.63      0.71      3653

    accuracy                           0.75      7307
   macro avg       0.76      0.75      0.74      7307
weighted avg       0.76      0.75      0.74      7307
```

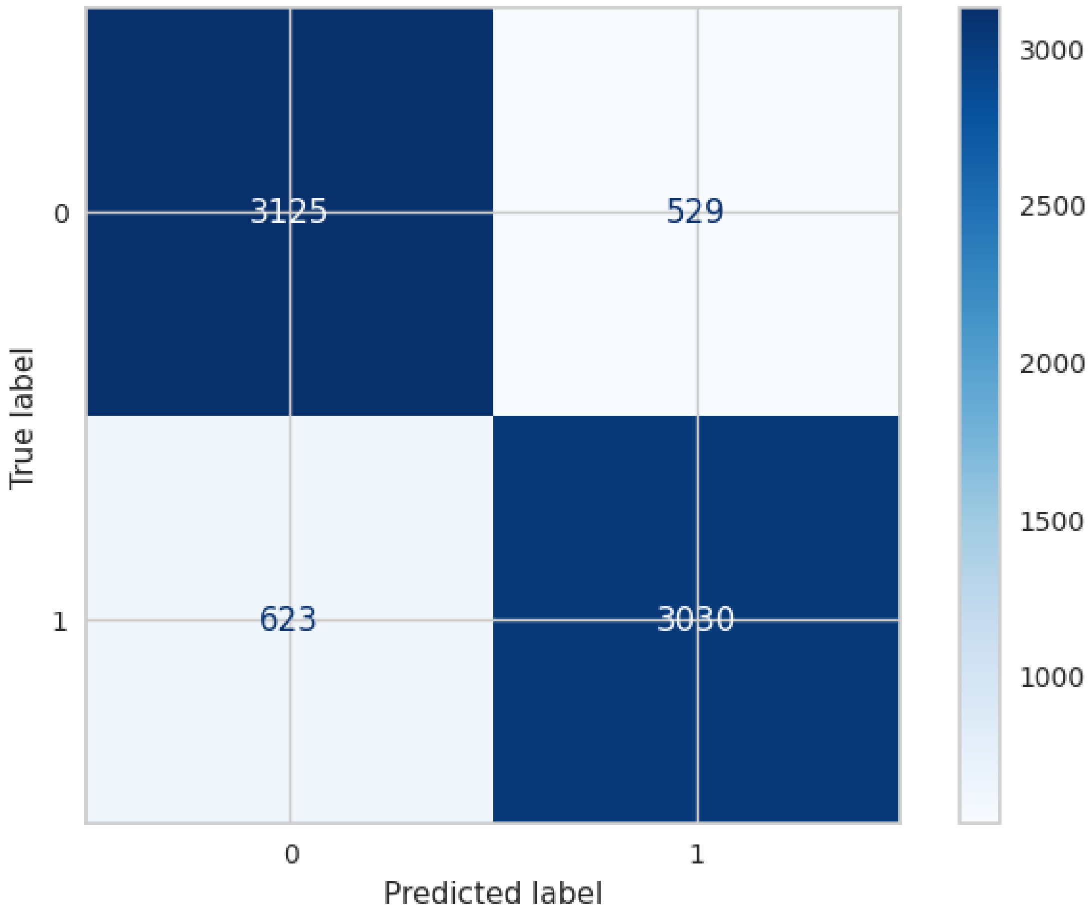
```
Confusion Matrix is :
 [[3169  485]
 [1366 2287]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8431032647537369
Model Test Score is :  0.8423429588066238
F1 Score is :  0.8402662229617305
Recall Score is :  0.8294552422666301
Precision Score is :  0.8513627423433549
AUC Value  :  0.8423411952986134
Classification Report is :                precision    recall  f1-score
support

           0       0.83      0.86      0.84      3654
           1       0.85      0.83      0.84      3653

    accuracy                           0.84      7307
   macro avg       0.84      0.84      0.84      7307
weighted avg       0.84      0.84      0.84      7307

Confusion Matrix is :
 [[3125  529]
 [ 623 3030]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.7440658120827821
Model Test Score is :  0.7495552210209389
F1 Score is :  0.7950268817204301
Recall Score is :  0.9715302491103203
Precision Score is :  0.6727962085308057
AUC Value  :  0.7495855952721826
Classification Report is :                precision    recall  f1-score
support

           0       0.95      0.53      0.68      3654
           1       0.67      0.97      0.80      3653

    accuracy                           0.75      7307
   macro avg       0.81      0.75      0.74      7307
weighted avg       0.81      0.75      0.74      7307

Confusion Matrix is :
```

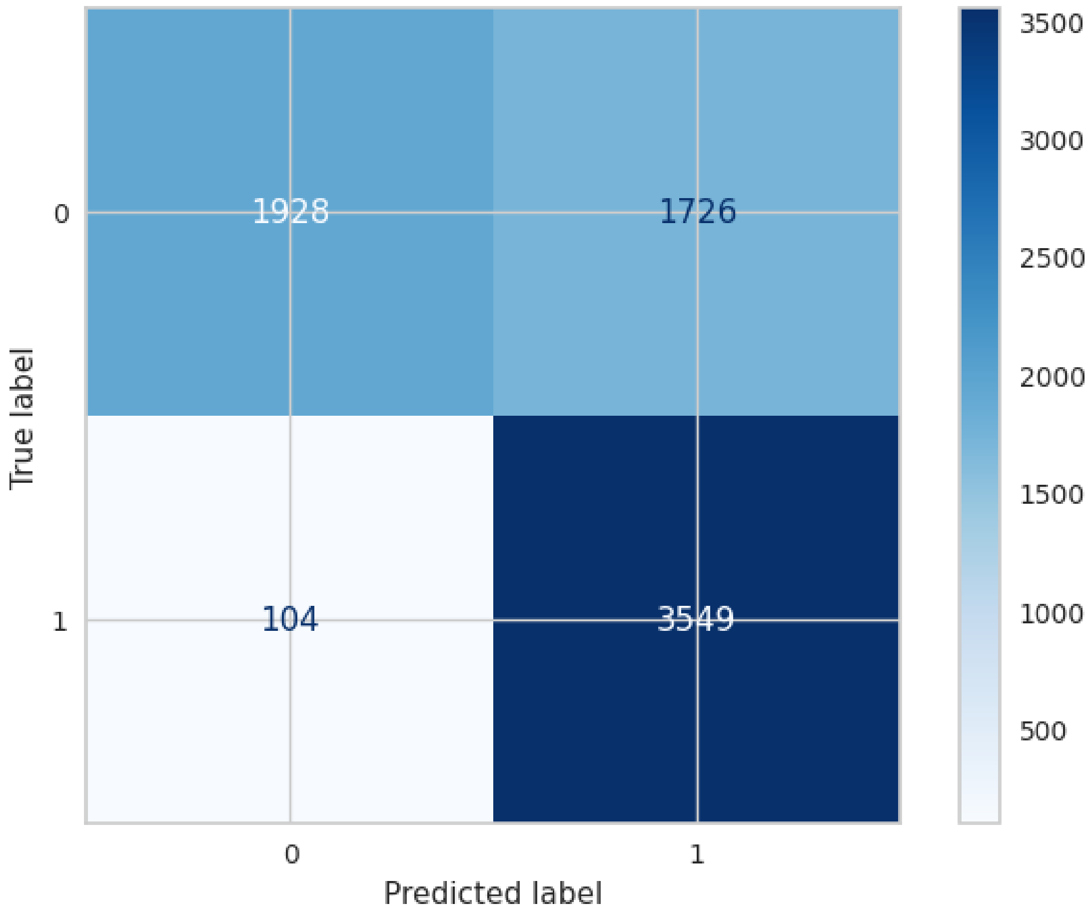
```
 [[1928 1726]
 [ 104 3549]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8395602390401897
Model Test Score is :  0.8378267414807719
F1 Score is :  0.8492558198702456
Recall Score is :  0.9137695045168355
Precision Score is :  0.7932509505703422
AUC Value  :  0.8378371332107987
Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.76      0.82      3654
           1       0.79      0.91      0.85      3653

    accuracy                           0.84      7307
   macro avg       0.85      0.84      0.84      7307
weighted avg       0.85      0.84      0.84      7307

Confusion Matrix is :
 [[2784  870]
 [ 315 3338]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.7979867098520445
Model Test Score is :  0.8067606404817298
F1 Score is :  0.7963069821119446
Recall Score is :  0.7555433889953463
Precision Score is :  0.8417200365965233
AUC Value  :  0.8067536321003004
Classification Report is :                precision    recall  f1-score
support

           0       0.78      0.86      0.82      3654
           1       0.84      0.76      0.80      3653

    accuracy                           0.81      7307
   macro avg       0.81      0.81      0.81      7307
weighted avg       0.81      0.81      0.81      7307

Confusion Matrix is :
 [[3135  519]
```

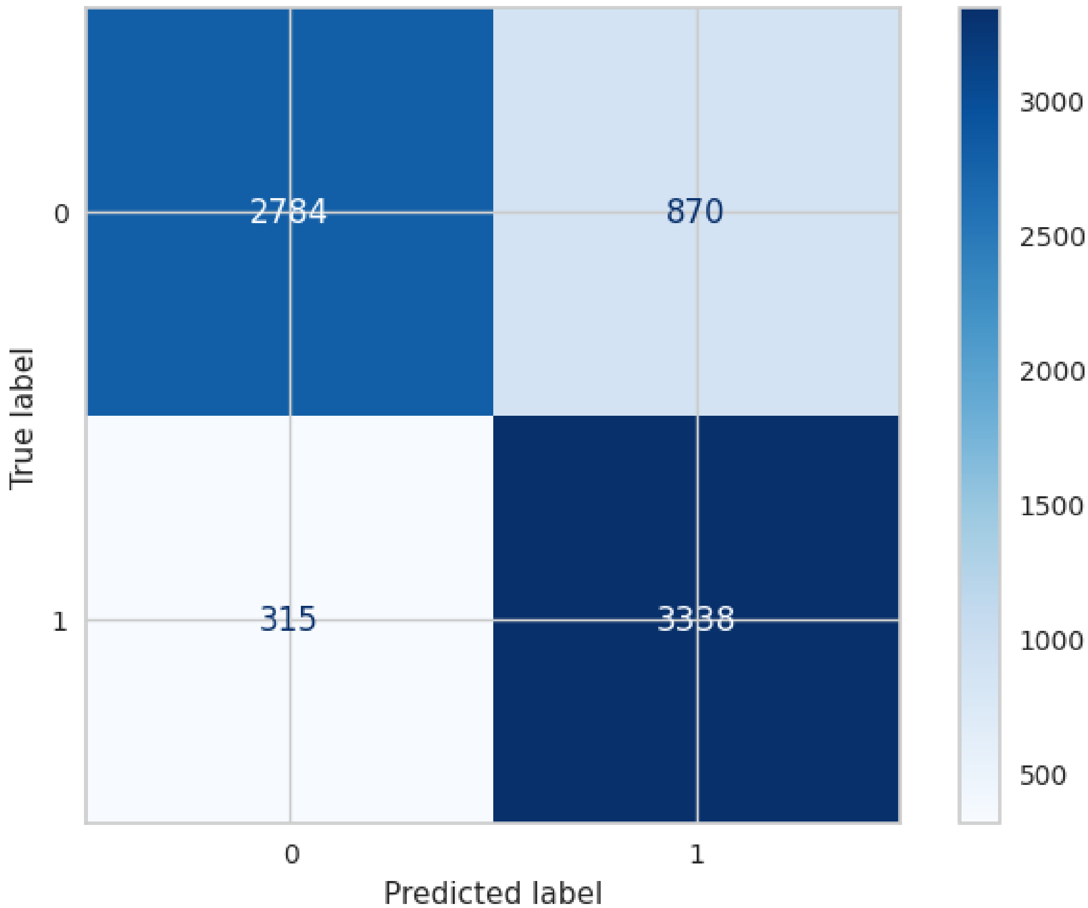
```
 [ 893 2760]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.690449036692365
Model Test Score is :  0.6986451348022444
F1 Score is :  0.7662420382165606
Recall Score is :  0.9879551053928278
Precision Score is :  0.6258019767643489
AUC Value  :  0.6986847229208255

Classification Report is :                precision    recall  f1-score
support

           0       0.97      0.41      0.58      3654
           1       0.63      0.99      0.77      3653

    accuracy                           0.70      7307
   macro avg       0.80      0.70      0.67      7307
weighted avg       0.80      0.70      0.67      7307

Confusion Matrix is :
 [[1496 2158]
 [  44 3609]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8274105500053222
Model Test Score is :  0.8261940604899412
F1 Score is :  0.841961174713788
Recall Score is :  0.9260881467287161
Precision Score is :  0.7718457677389916
AUC Value  :  0.8262077296314627

Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.73      0.81      3654
           1       0.77      0.93      0.84      3653

    accuracy                           0.83      7307
   macro avg       0.84      0.83      0.82      7307
weighted avg       0.84      0.83      0.82      7307

Confusion Matrix is :
 [[2654 1000]
 [ 270 3383]]
```

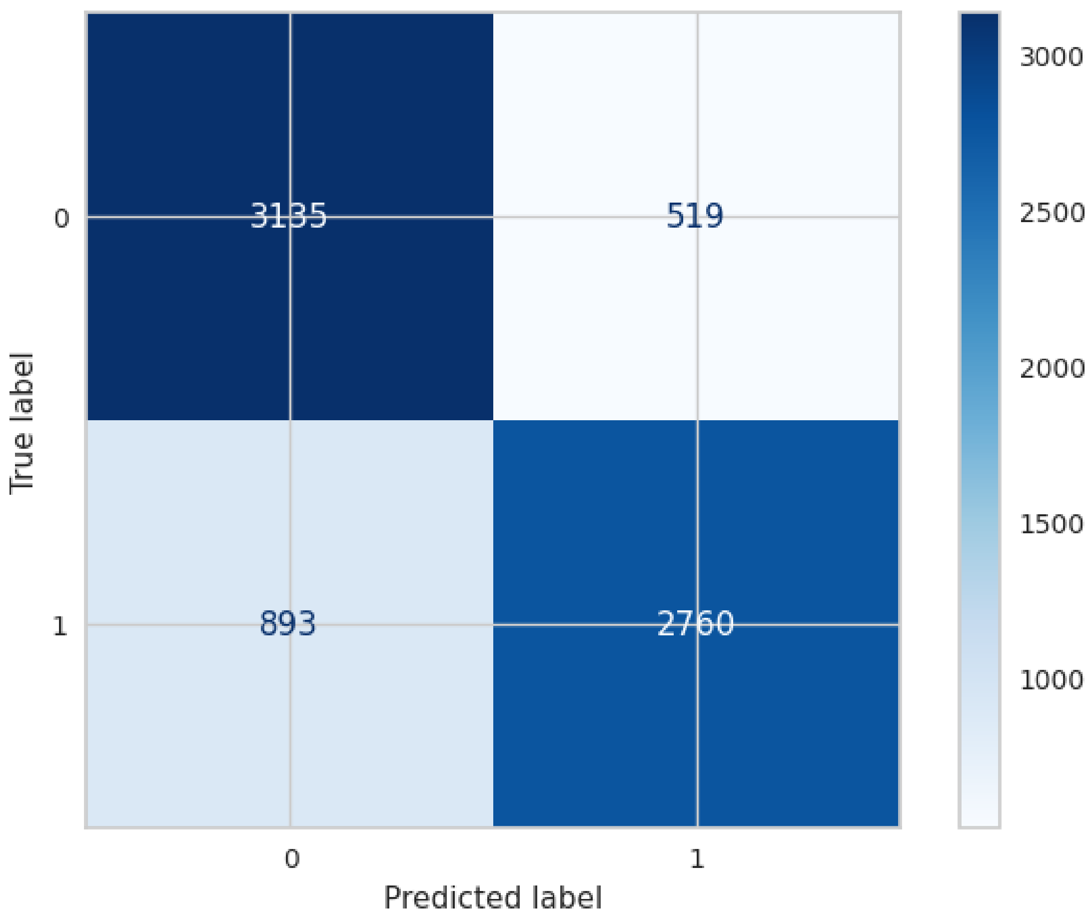


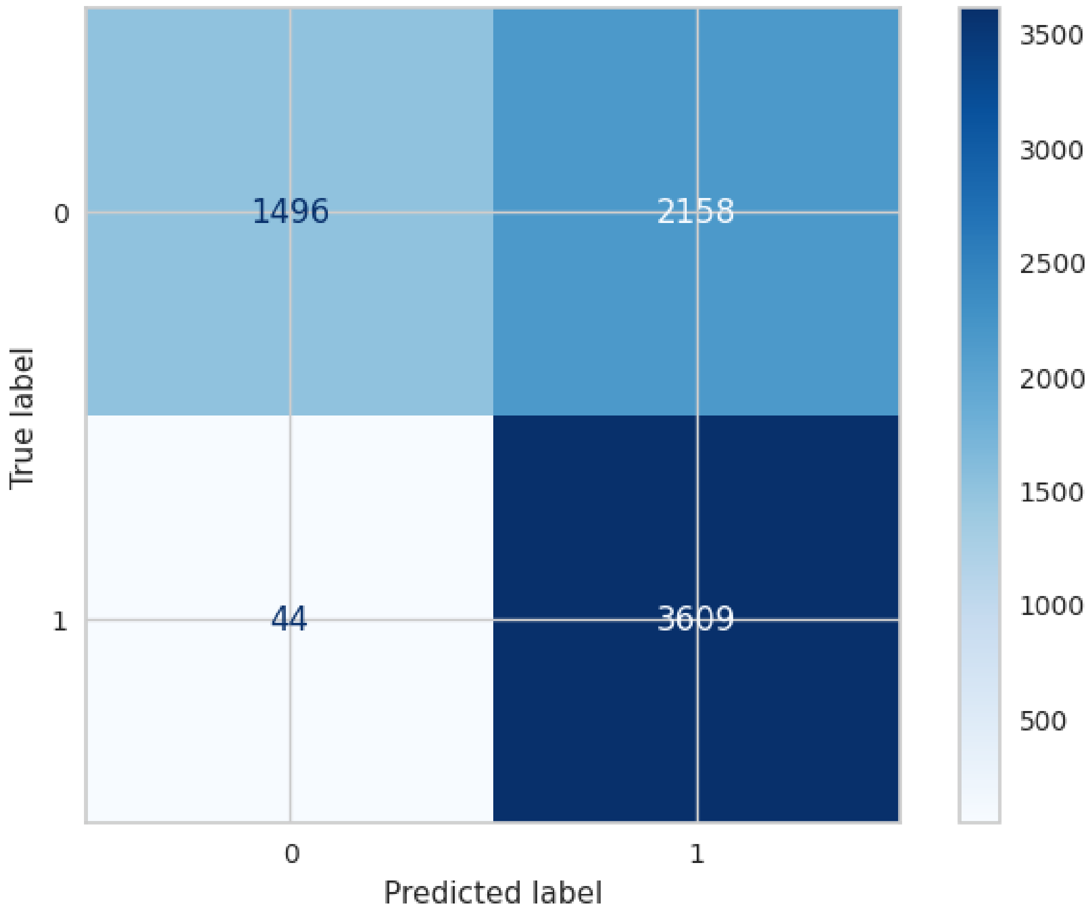
3169
485
1366
2287
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


3125
529
623
3030
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000

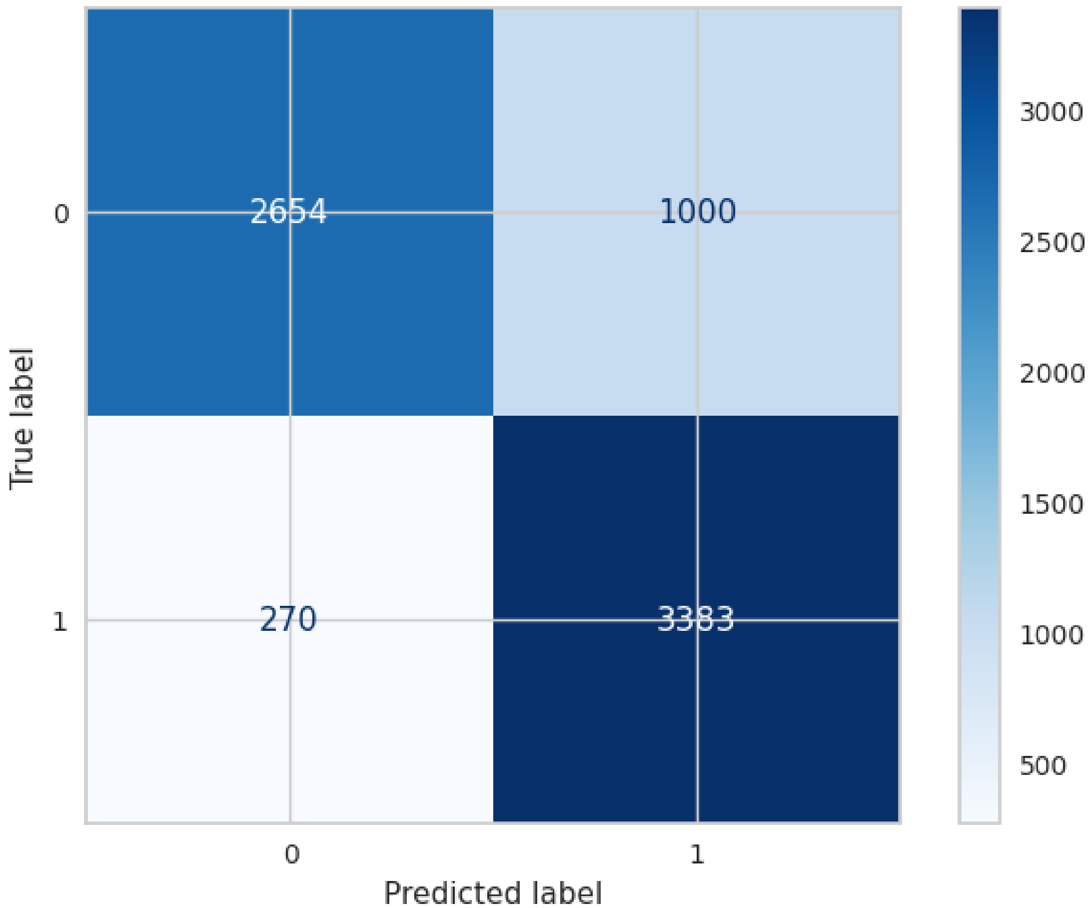

1928
1726
104
3549
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


2784
870
315
3338
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


3135
519
893
2760
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000



1496
2158
44
3609
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[305]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['NB Over','NB Over With Feature','NB Over Scaling','NB Over␣
↪With Normalize','NB Over With PCA'
                ,'NB Over With PCA and Scaling',
                'NB Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[305]:

Models	Train Accuracy	Test Accuracy	Test F1 \
NB Over	0.733665	0.746681	0.711907
NB Over With Feature	0.843103	0.842343	0.840266
NB Over Scaling	0.744066	0.749555	0.795027
NB Over With Normalize	0.839560	0.837827	0.849256
NB Over With PCA	0.797987	0.806761	0.796307
NB Over With PCA and Scaling	0.690449	0.698645	0.766242
NB Over With PCA and Normalize	0.827411	0.826194	0.841961

```
                                 Test Recall  Test Precision       AUC
Models
NB Over                             0.626061        0.825036  0.746665
NB Over With Feature                0.829455        0.851363  0.842341
NB Over Scaling                     0.971530        0.672796  0.749586
NB Over With Normalize              0.913770        0.793251  0.837837
NB Over With PCA                    0.755543        0.841720  0.806754
NB Over With PCA and Scaling        0.987955        0.625802  0.698685
NB Over With PCA and Normalize      0.926088        0.771846  0.826208
```

[306]:
```
models_draw(df)
```

RandomUnderSampler

[307]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[308]:
```
cross_validation(GaussianNB(),X_train,y_train)
```

```
Train Score Value :  [0.72649701 0.72709581 0.73038922 0.7254491  0.72679641]
Mean 0.727245508982036
Test Score Value :  [0.72275449 0.72155689 0.71976048 0.73532934 0.73413174]
Mean 0.7267065868263474
```

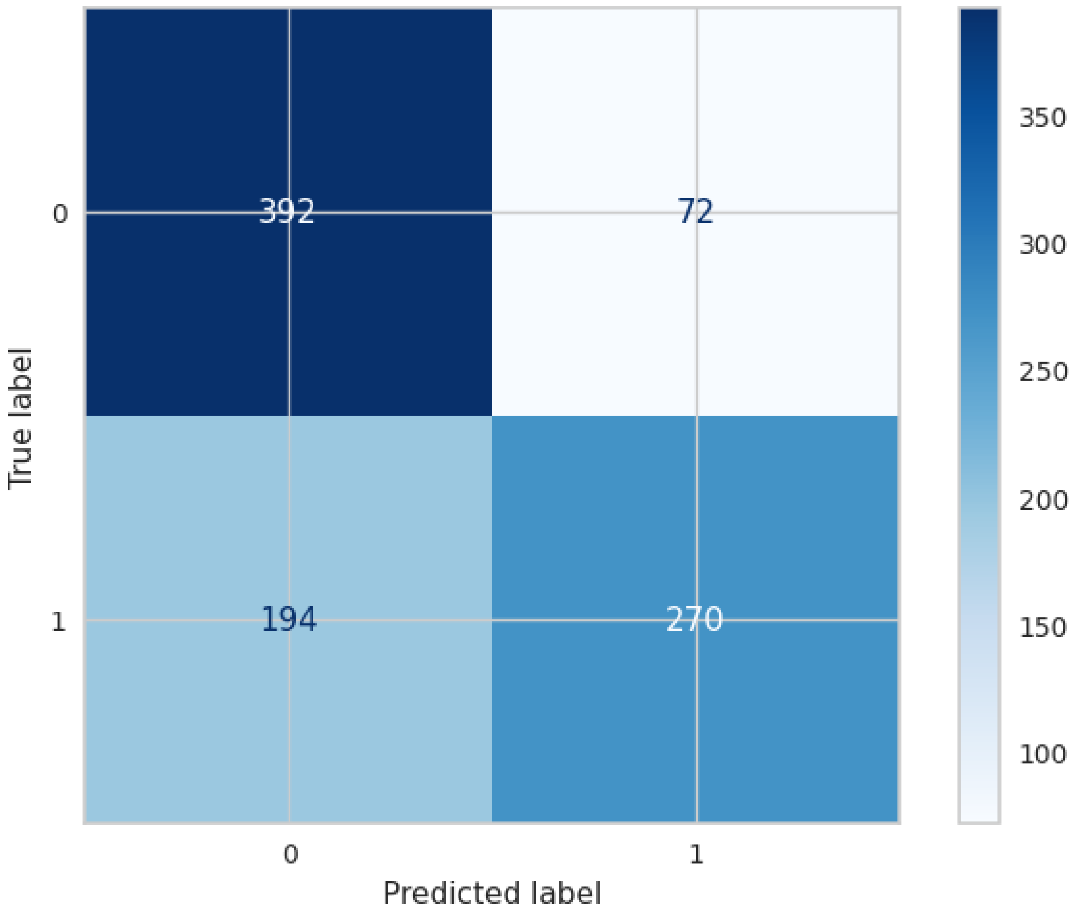
[309]:
```
Values = Models(GaussianNB(),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

Model Train Score is :  0.7275449101796407
Model Test Score is :  0.7133620689655172
F1 Score is :  0.6699751861042184
Recall Score is :  0.5818965517241379
Precision Score is :  0.7894736842105263
AUC Value  :  0.7133620689655172

Classification Report is :                precision    recall  f1-score
support

           0       0.67      0.84      0.75       464
           1       0.79      0.58      0.67       464

    accuracy                           0.71       928
   macro avg       0.73      0.71      0.71       928
weighted avg       0.73      0.71      0.71       928
```

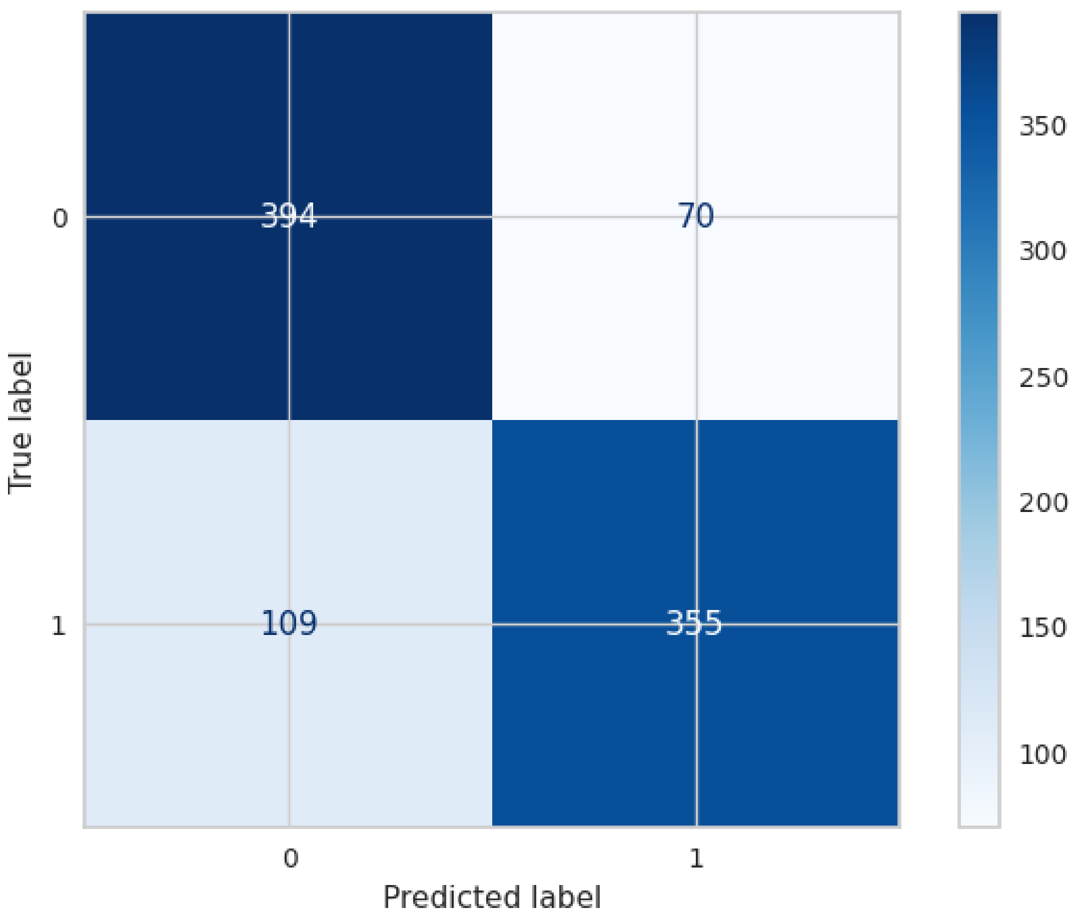
```
Confusion Matrix is :
 [[392  72]
 [194 270]]


 Apply Model With Feature Selection :

Model Train Score is :  0.7902994011976048
Model Test Score is :  0.8071120689655172
F1 Score is :  0.7986501687289089
Recall Score is :  0.7650862068965517
Precision Score is :  0.8352941176470589
AUC Value  :  0.8071120689655171
Classification Report is :                precision    recall  f1-score
support

           0       0.78      0.85      0.81       464
           1       0.84      0.77      0.80       464

    accuracy                           0.81       928
   macro avg       0.81      0.81      0.81       928
weighted avg       0.81      0.81      0.81       928

Confusion Matrix is :
 [[394  70]
 [109 355]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.7275449101796407
Model Test Score is :  0.7133620689655172
F1 Score is :  0.6699751861042184
Recall Score is :  0.5818965517241379
Precision Score is :  0.7894736842105263
AUC Value  :  0.7133620689655172
Classification Report is :                precision    recall  f1-score
support

           0       0.67      0.84      0.75       464
           1       0.79      0.58      0.67       464

    accuracy                           0.71       928
   macro avg       0.73      0.71      0.71       928
weighted avg       0.73      0.71      0.71       928

Confusion Matrix is :
```

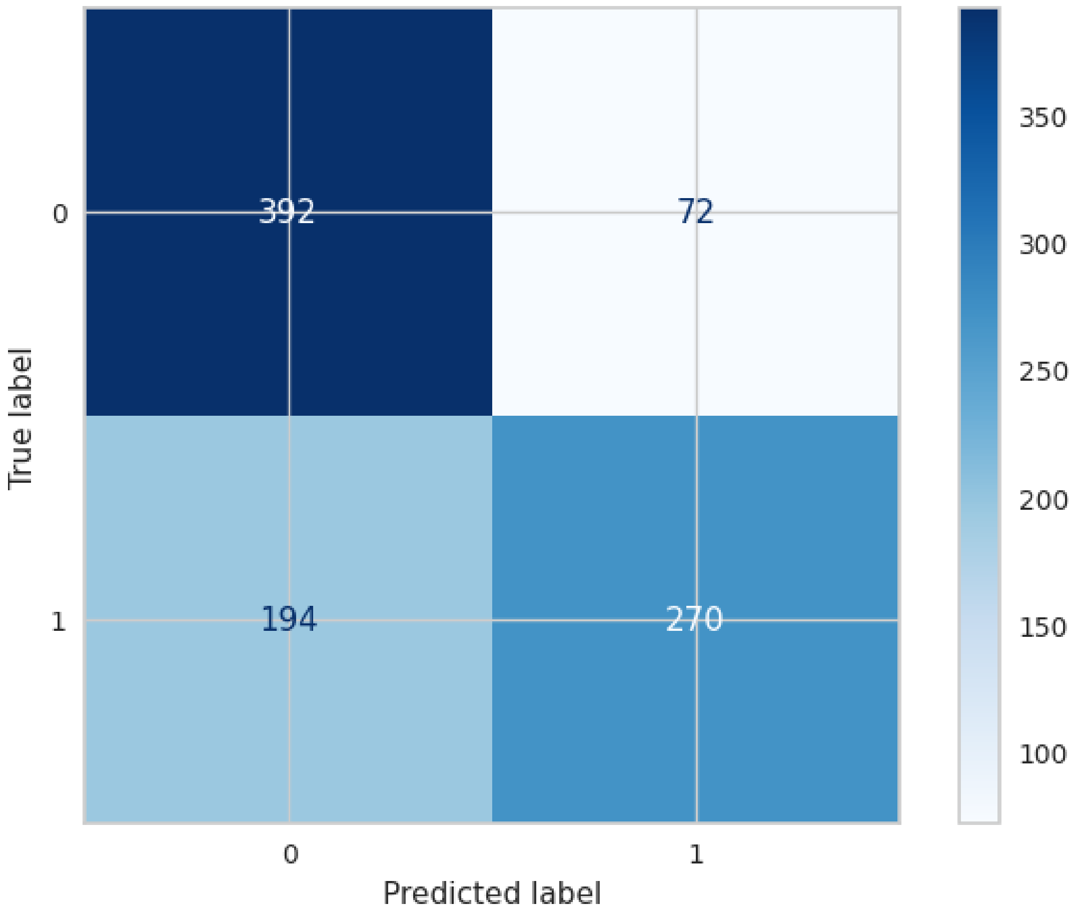
```
 [[392  72]
 [194 270]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8396407185628743
Model Test Score is :  0.853448275862069
F1 Score is :  0.8612244897959183
Recall Score is :  0.9094827586206896
Precision Score is :  0.8178294573643411
AUC Value  :  0.853448275862069
Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.80      0.84       464
           1       0.82      0.91      0.86       464

    accuracy                           0.85       928
   macro avg       0.86      0.85      0.85       928
weighted avg       0.86      0.85      0.85       928

Confusion Matrix is :
 [[370  94]
 [ 42 422]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.7662275449101796
Model Test Score is :  0.7672413793103449
F1 Score is :  0.7333333333333334
Recall Score is :  0.6400862068965517
Precision Score is :  0.8583815028901735
AUC Value  :  0.7672413793103448
Classification Report is :                precision    recall  f1-score
support

           0       0.71      0.89      0.79       464
           1       0.86      0.64      0.73       464

    accuracy                           0.77       928
   macro avg       0.79      0.77      0.76       928
weighted avg       0.79      0.77      0.76       928

Confusion Matrix is :
 [[415  49]
```

```
 [167 297]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.7553293413173653
Model Test Score is :  0.7640086206896551
F1 Score is :  0.7326007326007326
Recall Score is :  0.646551724137931
Precision Score is :  0.8450704225352113
AUC Value  :  0.7640086206896552
Classification Report is :                precision    recall  f1-score
support

           0       0.71      0.88      0.79       464
           1       0.85      0.65      0.73       464

    accuracy                           0.76       928
   macro avg       0.78      0.76      0.76       928
weighted avg       0.78      0.76      0.76       928

Confusion Matrix is :
 [[409  55]
 [164 300]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8405988023952096
Model Test Score is :  0.8556034482758621
F1 Score is :  0.8624229979466119
Recall Score is :  0.9051724137931034
Precision Score is :  0.8235294117647058
AUC Value  :  0.8556034482758621
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.81      0.85       464
           1       0.82      0.91      0.86       464

    accuracy                           0.86       928
   macro avg       0.86      0.86      0.86       928
weighted avg       0.86      0.86      0.86       928

Confusion Matrix is :
 [[374  90]
 [ 44 420]]
```

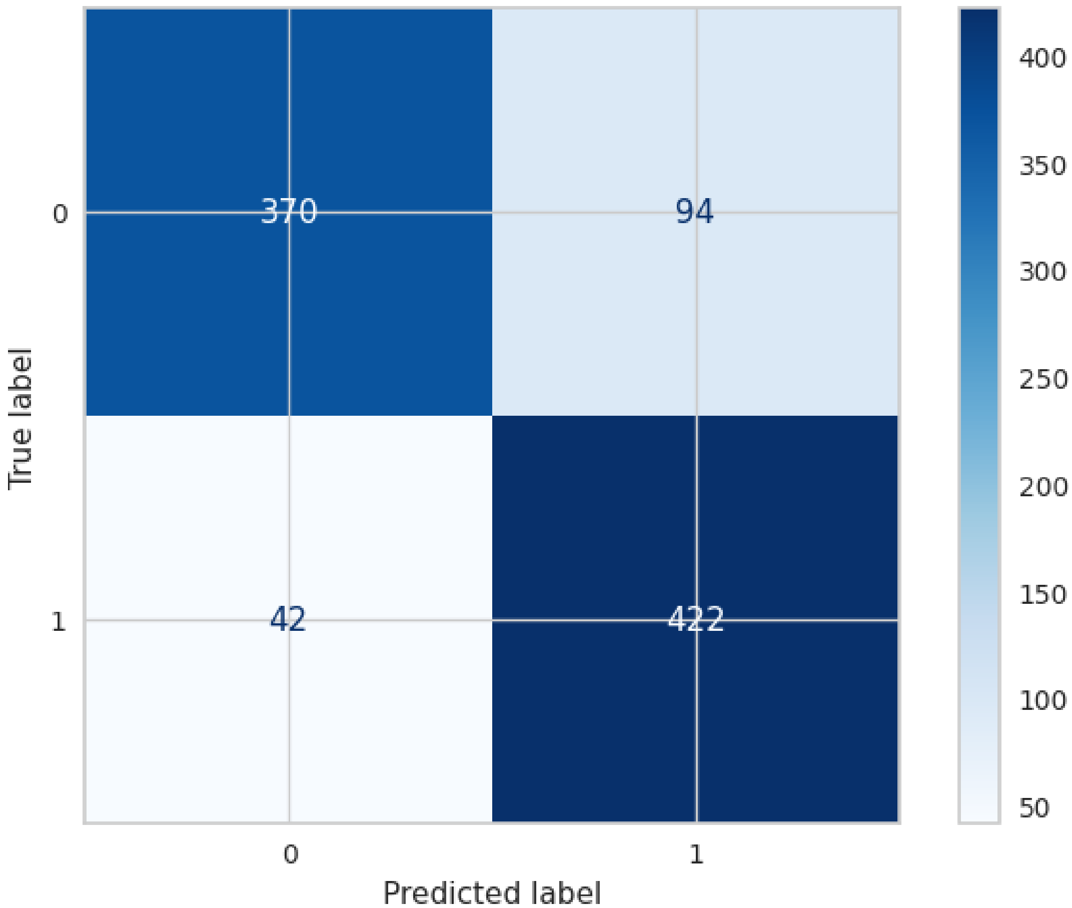


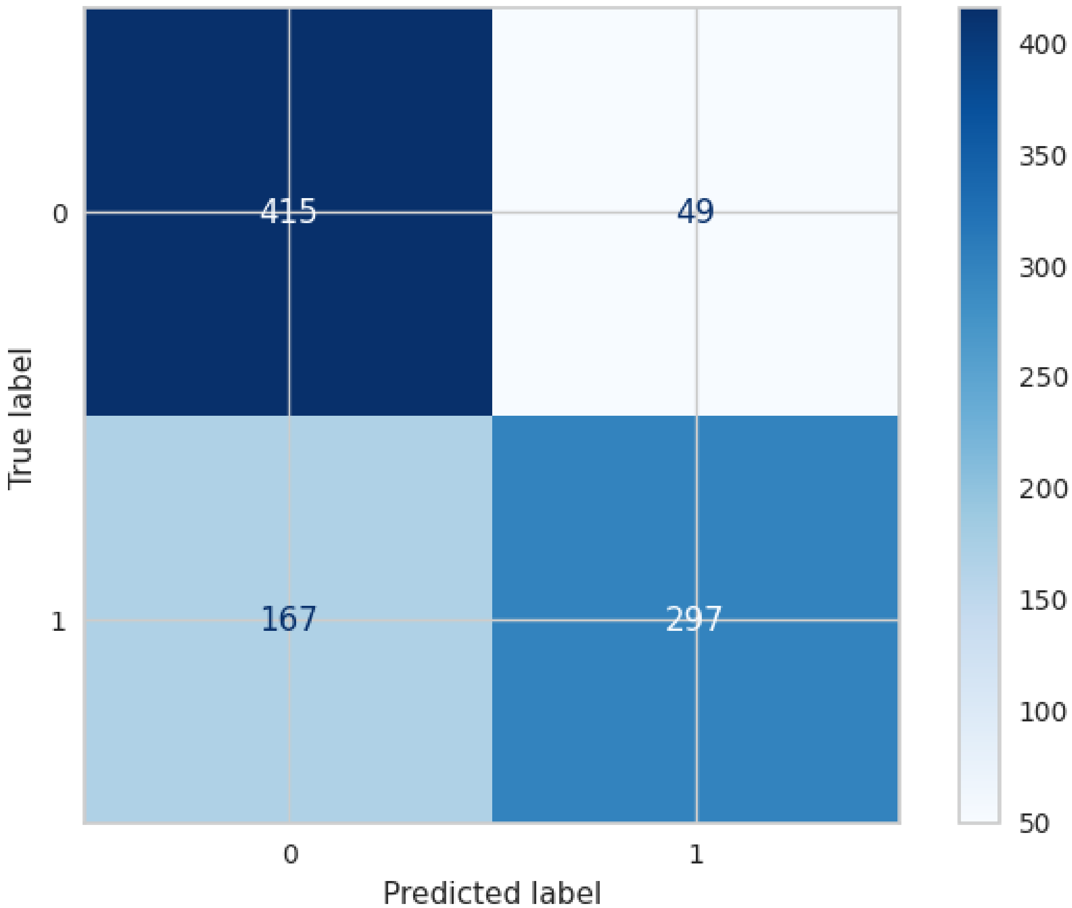
392
72
194
270
0
1
True label
0
1
Predicted label
350
300
250
200
150
100

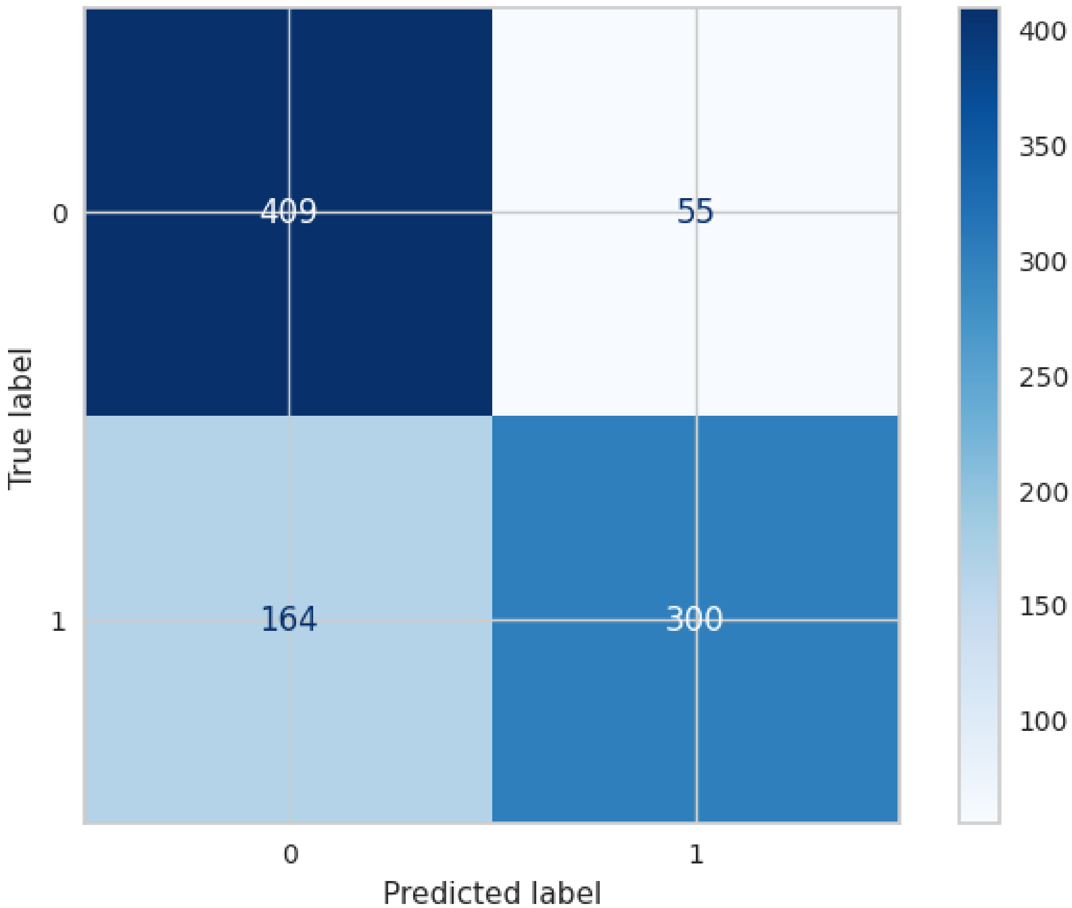

394
70
109
355
0
1
True label
0
1
Predicted label
350
300
250
200
150
100



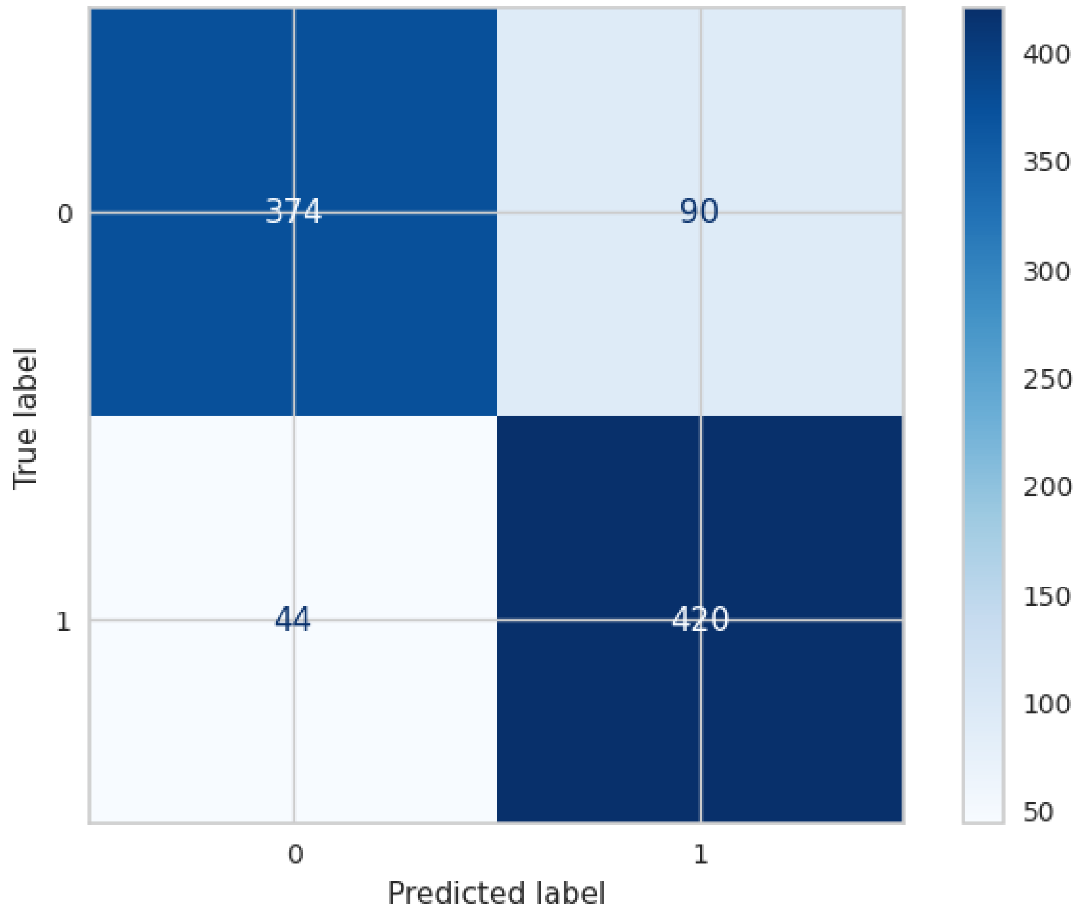
392
72
194
270
0
1
True label
0
1
Predicted label
350
300
250
200
150
100


370
94
42
422
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


415
49
167
297
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50



409
55
164
300
0
1
0
1
True label
Predicted label
400
350
300
250
200
150
100



[310]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['NB Under','NB Under With Feature','NB Under Scaling','NB Under␣
↪With Normalize','NB Under With PCA'
                ,'NB Under With PCA and Scaling',
                'NB Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[310]:
```
                                 Train Accuracy  Test Accuracy   Test F1  \
Models
NB Under                               0.727545       0.713362  0.669975
NB Under With Feature                  0.790299       0.807112  0.798650
NB Under Scaling                       0.727545       0.713362  0.669975
NB Under With Normalize                0.839641       0.853448  0.861224
NB Under With PCA                      0.766228       0.767241  0.733333
NB Under With PCA and Scaling          0.755329       0.764009  0.732601
NB Under With PCA and Normalize        0.840599       0.855603  0.862423
```

```
                                 Test Recall  Test Precision       AUC
Models
NB Under                            0.581897        0.789474  0.713362
NB Under With Feature               0.765086        0.835294  0.807112
NB Under Scaling                    0.581897        0.789474  0.713362
NB Under With Normalize             0.909483        0.817829  0.853448
NB Under With PCA                   0.640086        0.858382  0.767241
NB Under With PCA and Scaling       0.646552        0.845070  0.764009
NB Under With PCA and Normalize     0.905172        0.823529  0.855603
```

[311]:
```
models_draw(df)
```

GradientBoostingClassifier

[312]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[313]:
```
Search(GradientBoostingClassifier(n_estimators=100,max_depth=3) ,{'max_depth':
↪[5,10,20,25,30,40]},X_train,y_train)
```

[313]:
```
GradientBoostingClassifier(max_depth=5)
```

[314]:
```
cross_validation(GradientBoostingClassifier(n_estimators=100,max_depth=5),X_train,y_train)
```

```
Train Score Value :  [0.93529888 0.93452522 0.93631304 0.93688649 0.93354697]
Mean 0.9353141190681737
Test Score Value :  [0.91203454 0.91634057 0.91242747 0.90878424 0.9187694 ]
Mean 0.9136712445138689
```

[315]:
```
Values =␣
↪Models(GradientBoostingClassifier(n_estimators=100,max_depth=5),X_train,y_train,X_test,y_te
```

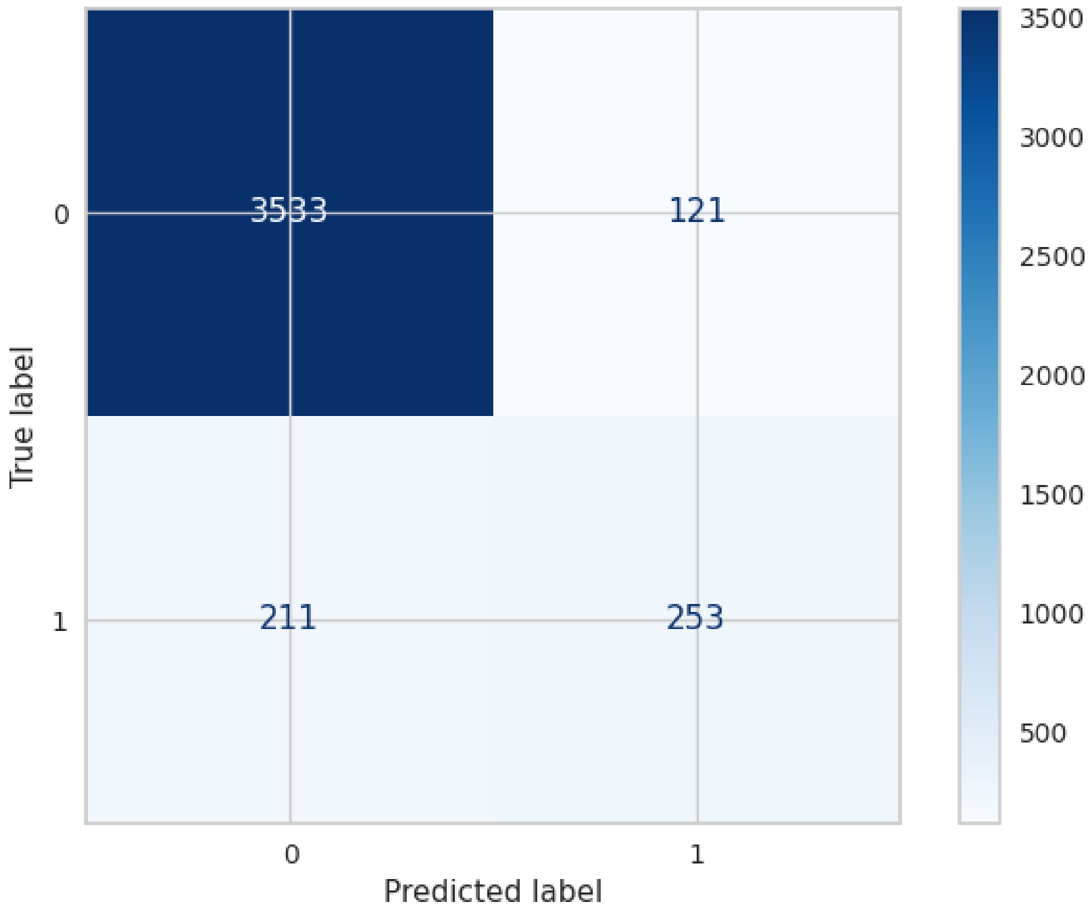
```
Apply Model With Normal Data :

Model Train Score is :  0.9332361830742659
Model Test Score is :  0.9193783389995144
F1 Score is :  0.6038186157517901
Recall Score is :  0.5452586206896551
Precision Score is :  0.6764705882352942
AUC Value  :  0.7560721127531472

Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.97      0.96      3654
```

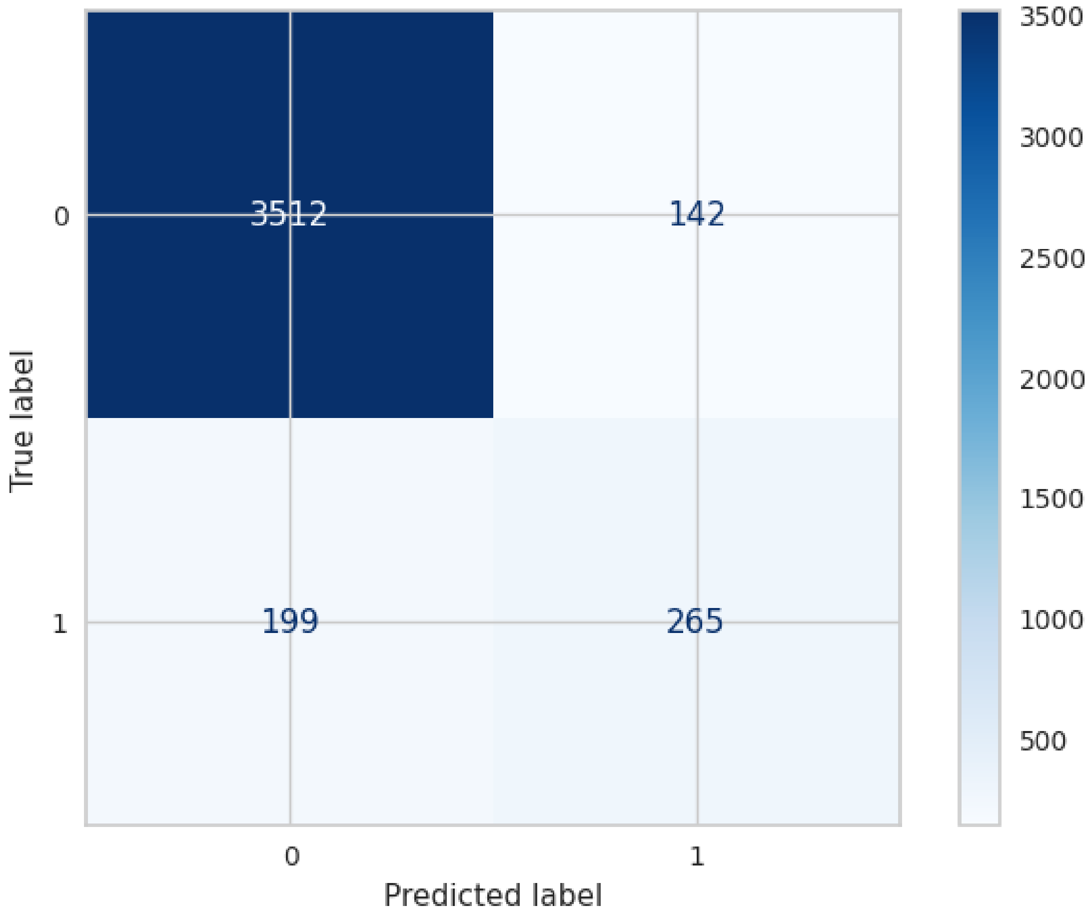
```
           1       0.68      0.55      0.60       464

    accuracy                           0.92      4118
   macro avg       0.81      0.76      0.78      4118
weighted avg       0.91      0.92      0.92      4118

Confusion Matrix is :
 [[3533  121]
 [ 211  253]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9240878670120898
Model Test Score is :  0.9171928120446818
F1 Score is :  0.6084959816303099
Recall Score is :  0.5711206896551724
Precision Score is :  0.6511056511056511
AUC Value  :  0.766129584017515
Classification Report is :                precision    recall  f1-score
support

           0       0.95      0.96      0.95      3654
           1       0.65      0.57      0.61       464

    accuracy                           0.92      4118
   macro avg       0.80      0.77      0.78      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3512  142]
 [ 199  265]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9332361830742659
Model Test Score is :  0.9193783389995144
F1 Score is :  0.6038186157517901
Recall Score is :  0.5452586206896551
Precision Score is :  0.6764705882352942
AUC Value  :  0.7560721127531472
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.97      0.96      3654
           1       0.68      0.55      0.60       464
```

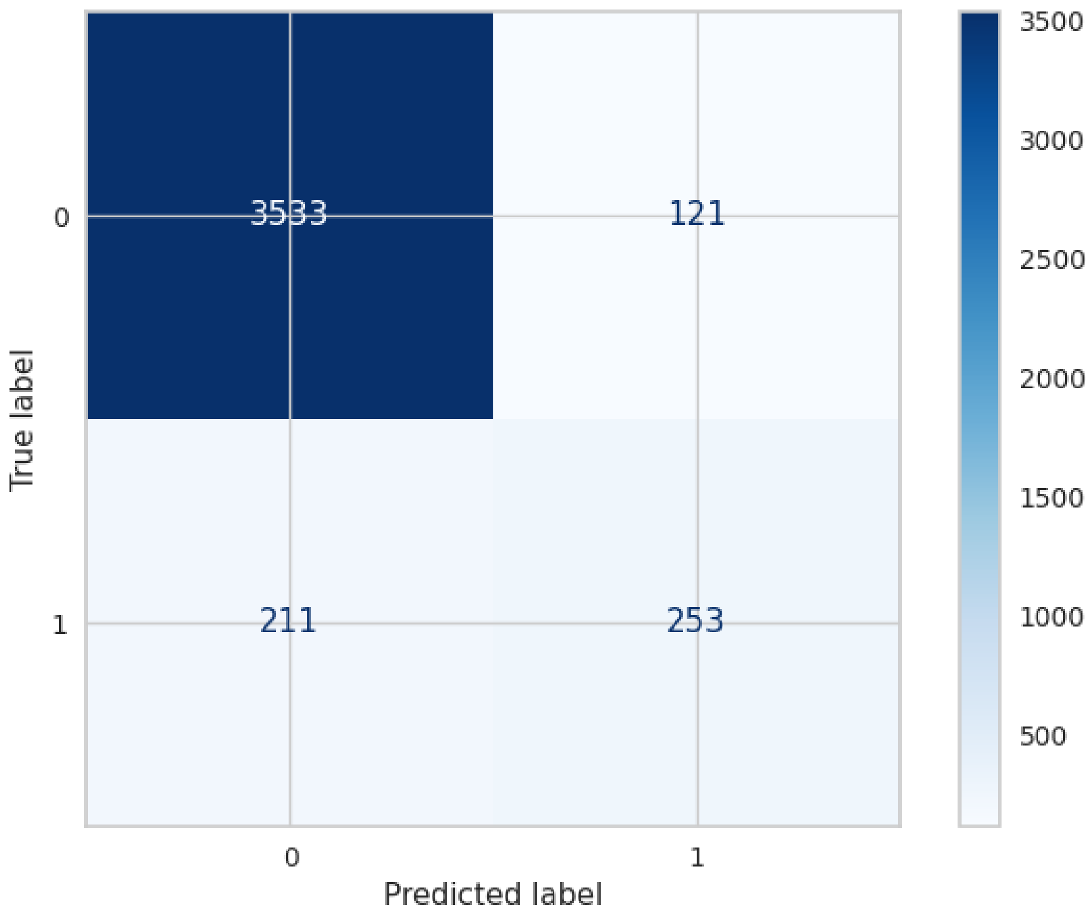
```
    accuracy                           0.92      4118
   macro avg       0.81      0.76      0.78      4118
weighted avg       0.91      0.92      0.92      4118

Confusion Matrix is :
 [[3533  121]
 [ 211  253]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9356649395509499
Model Test Score is :  0.9133074307916464
F1 Score is :  0.5641025641025641
Recall Score is :  0.4978448275862069
Precision Score is :  0.6507042253521127
AUC Value  :  0.7319547071702244
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.97      0.95      3654
           1       0.65      0.50      0.56       464

    accuracy                           0.91      4118
   macro avg       0.79      0.73      0.76      4118
weighted avg       0.91      0.91      0.91      4118

Confusion Matrix is :
 [[3530  124]
 [ 233  231]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9418987478411054
Model Test Score is :  0.917921321029626
F1 Score is :  0.5947242206235012
Recall Score is :  0.5344827586206896
Precision Score is :  0.6702702702702703
AUC Value  :  0.7505473453749315
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.97      0.95      3654
           1       0.67      0.53      0.59       464
```

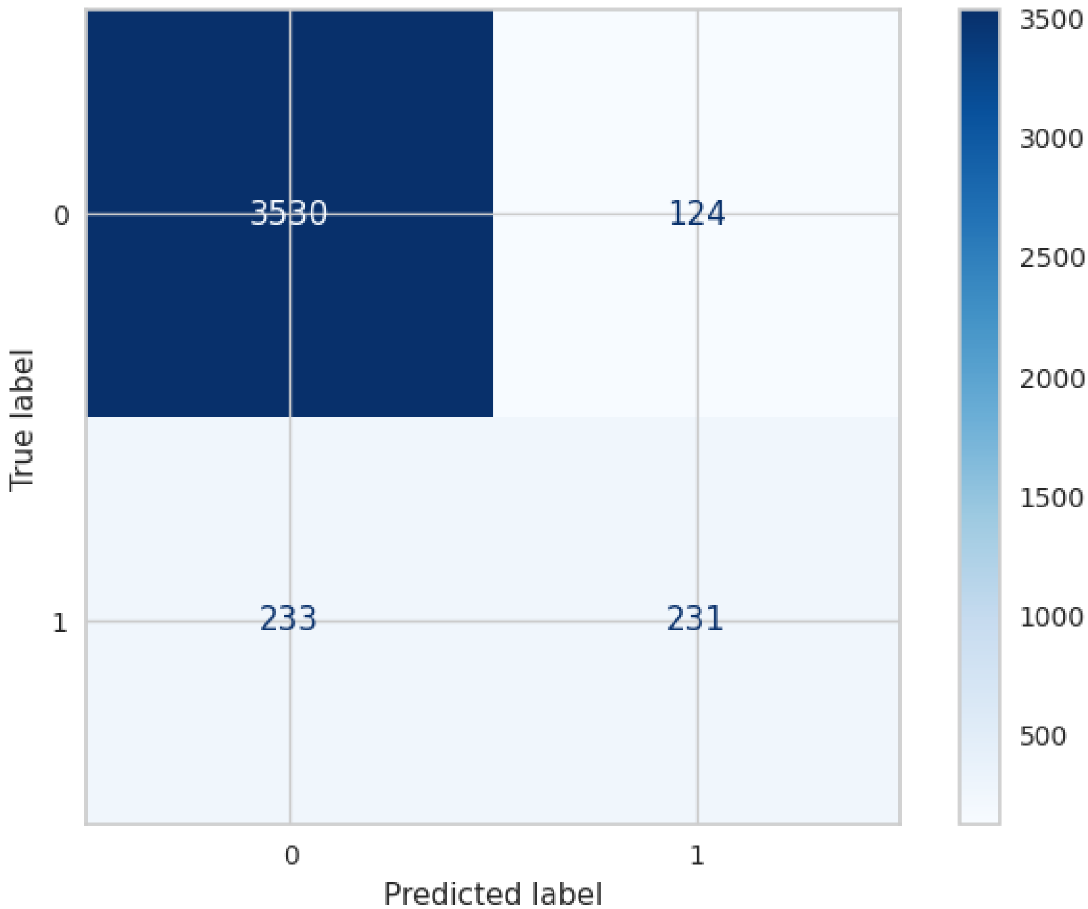
```
    accuracy                           0.92      4118
   macro avg       0.81      0.75      0.77      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3532  122]
 [ 216  248]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9396588946459413
Model Test Score is :  0.9140359397765906
F1 Score is :  0.549618320610687
Recall Score is :  0.46551724137931033
Precision Score is :  0.6708074534161491
AUC Value  :  0.7182539682539683
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.97      0.95      3654
           1       0.67      0.47      0.55       464

    accuracy                           0.91      4118
   macro avg       0.80      0.72      0.75      4118
weighted avg       0.90      0.91      0.91      4118

Confusion Matrix is :
 [[3548  106]
 [ 248  216]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9359078151986183
Model Test Score is :  0.9157357940747936
F1 Score is :  0.5678704856787049
Recall Score is :  0.49137931034482757
Precision Score is :  0.672566371681416
AUC Value  :  0.7305008210180625
Classification Report is :                precision    recall  f1-score
support

           0       0.94      0.97      0.95      3654
           1       0.67      0.49      0.57       464

    accuracy                           0.92      4118
```

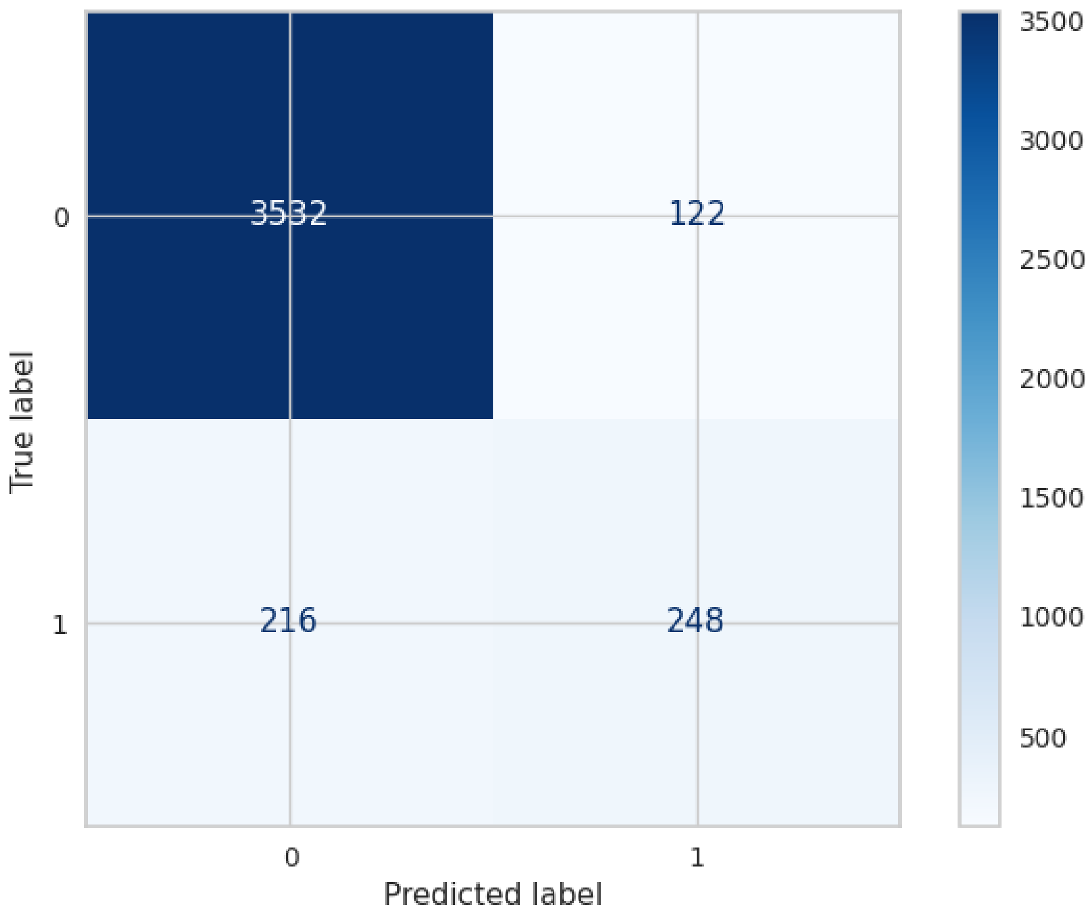
```
   macro avg       0.81      0.73      0.76      4118
weighted avg       0.91      0.92      0.91      4118

Confusion Matrix is :
 [[3543  111]
 [ 236  228]]
```



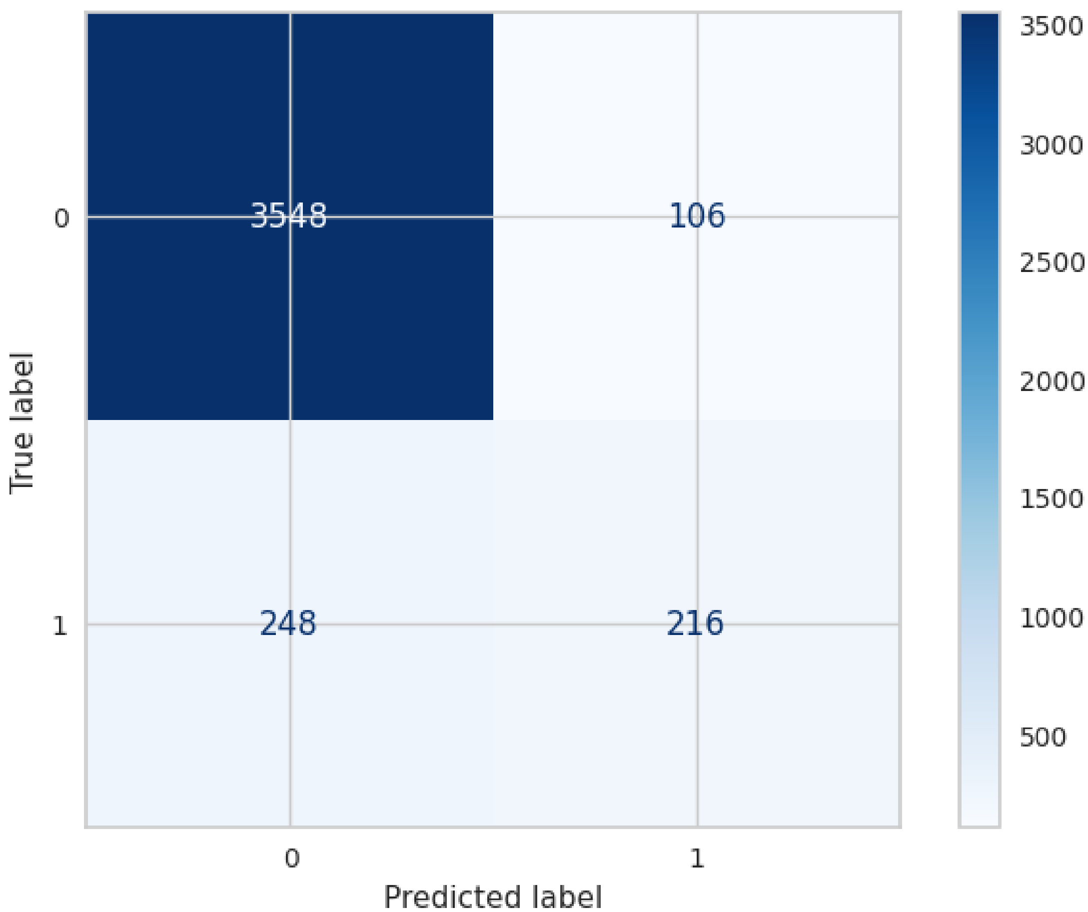

3512
142
199
265
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500

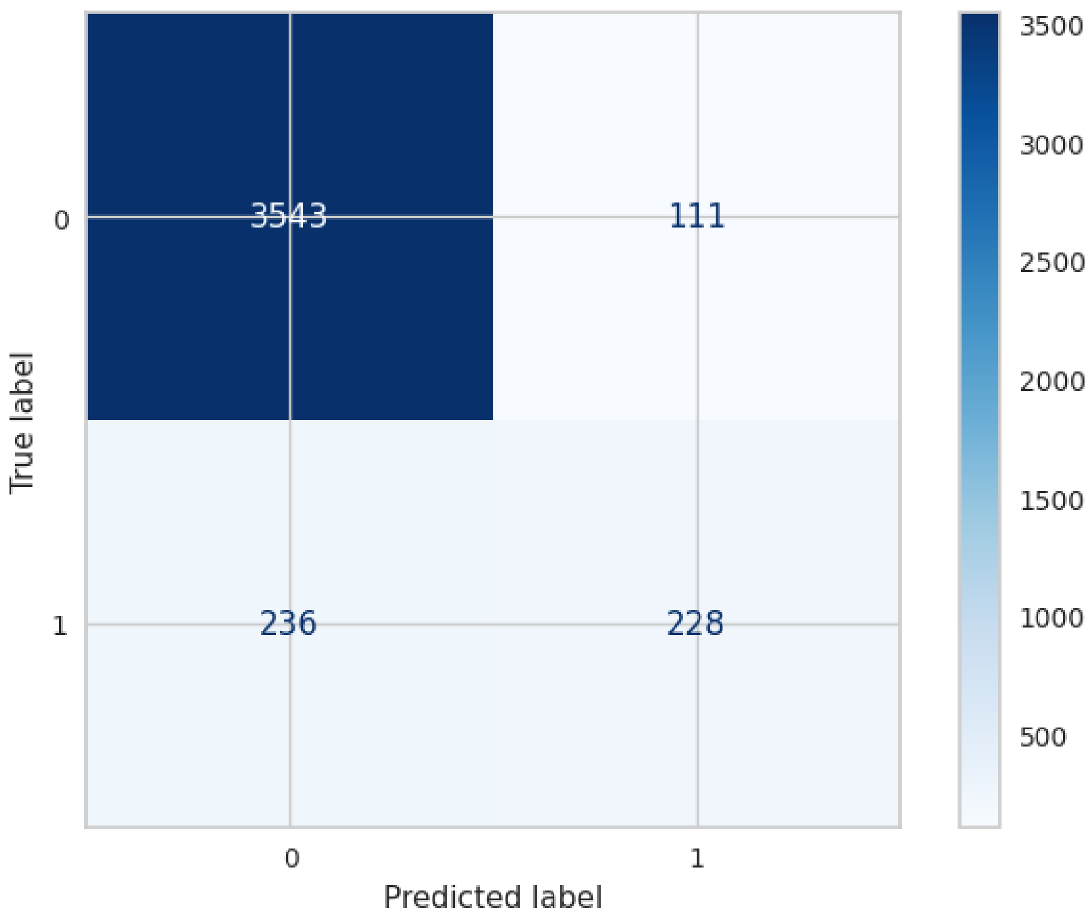

3533
121
211
253
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3530
124
233
231
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500



3532
122
216
248
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3548
106
248
216
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[316]:

```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Gradient','Gradient With Feature','Gradient Scaling','Gradient␣
↪With Normalize','Gradient With PCA'
                ,'Gradient With PCA and Scaling',
                'Gradient With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[316]:

```
                                 Train Accuracy  Test Accuracy   Test F1  \
Models
Gradient                               0.933236       0.919378  0.603819
Gradient With Feature                  0.924088       0.917193  0.608496
Gradient Scaling                       0.933236       0.919378  0.603819
Gradient With Normalize                0.935665       0.913307  0.564103
Gradient With PCA                      0.941899       0.917921  0.594724
Gradient With PCA and Scaling          0.939659       0.914036  0.549618
Gradient With PCA and Normalize        0.935908       0.915736  0.567870
```

```
                                   Test Recall  Test Precision       AUC
Models
Gradient                              0.545259        0.676471  0.756072
Gradient With Feature                 0.571121        0.651106  0.766130
Gradient Scaling                      0.545259        0.676471  0.756072
Gradient With Normalize               0.497845        0.650704  0.731955
Gradient With PCA                     0.534483        0.670270  0.750547
Gradient With PCA and Scaling         0.465517        0.670807  0.718254
Gradient With PCA and Normalize       0.491379        0.672566  0.730501
```

[317]:
```
models_draw(df)
```

RandomOverSampler

[318]:
```
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[319]:
```
Search(GradientBoostingClassifier(n_estimators=100,max_depth=3) ,{'max_depth':
 ↪[5,10,20,25,30,40]},X_train,y_train)
```

[319]:
```
GradientBoostingClassifier(max_depth=20)
```

[320]:
```
cross_validation(GradientBoostingClassifier(n_estimators=100,max_depth=20),X_train,y_train)
```

```
Train Score Value :  [0.99992397 0.99990496 0.99992397 0.99990496 0.99996199]
Mean 0.9999239693330242
Test Score Value :  [0.96487493 0.96609139 0.96609139 0.9654045  0.96487226]
Mean 0.9654668938913403
```

[321]:
```
Values =␣
 ↪Models(GradientBoostingClassifier(n_estimators=100,max_depth=20),X_train,y_train,X_test,y_t
```

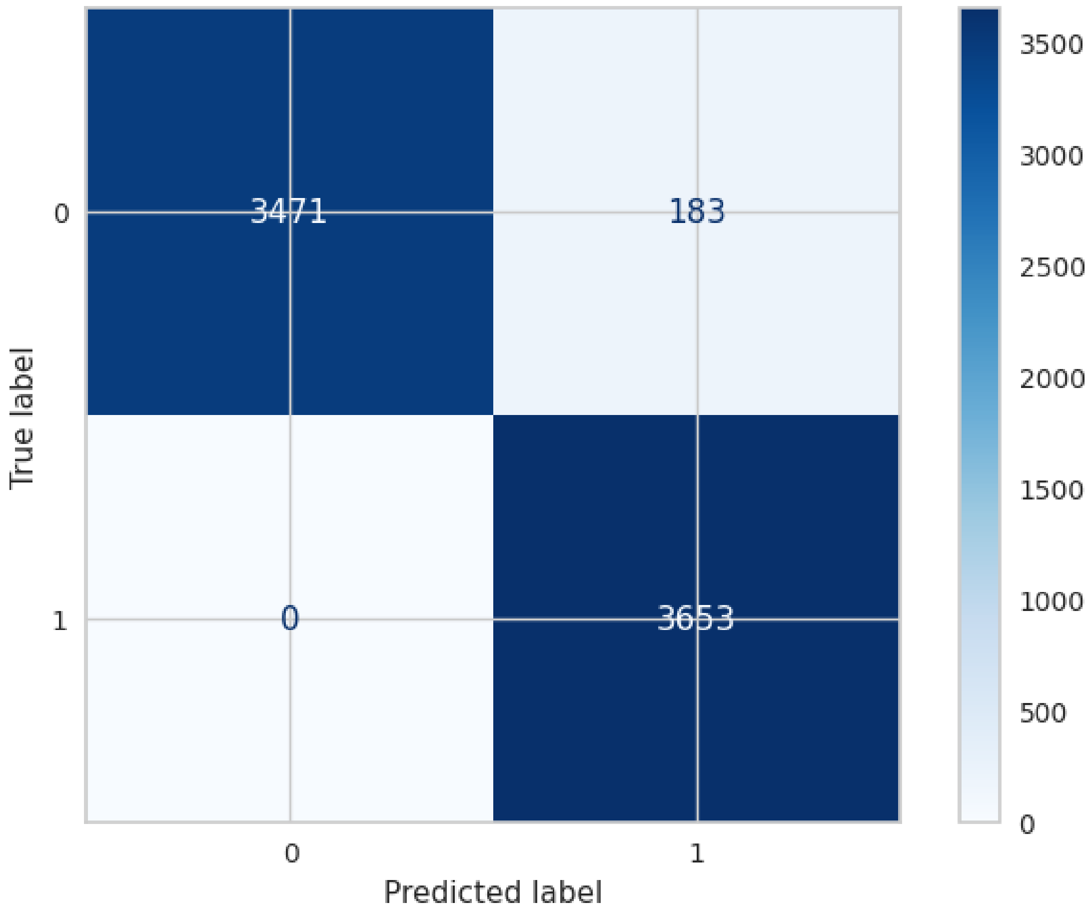
```
Apply Model With Normal Data :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9749555221020939
F1 Score is :  0.975564160769128
Recall Score is :  1.0
Precision Score is :  0.9522940563086548
AUC Value  :  0.9749589490968802

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.95      0.97      3654
```

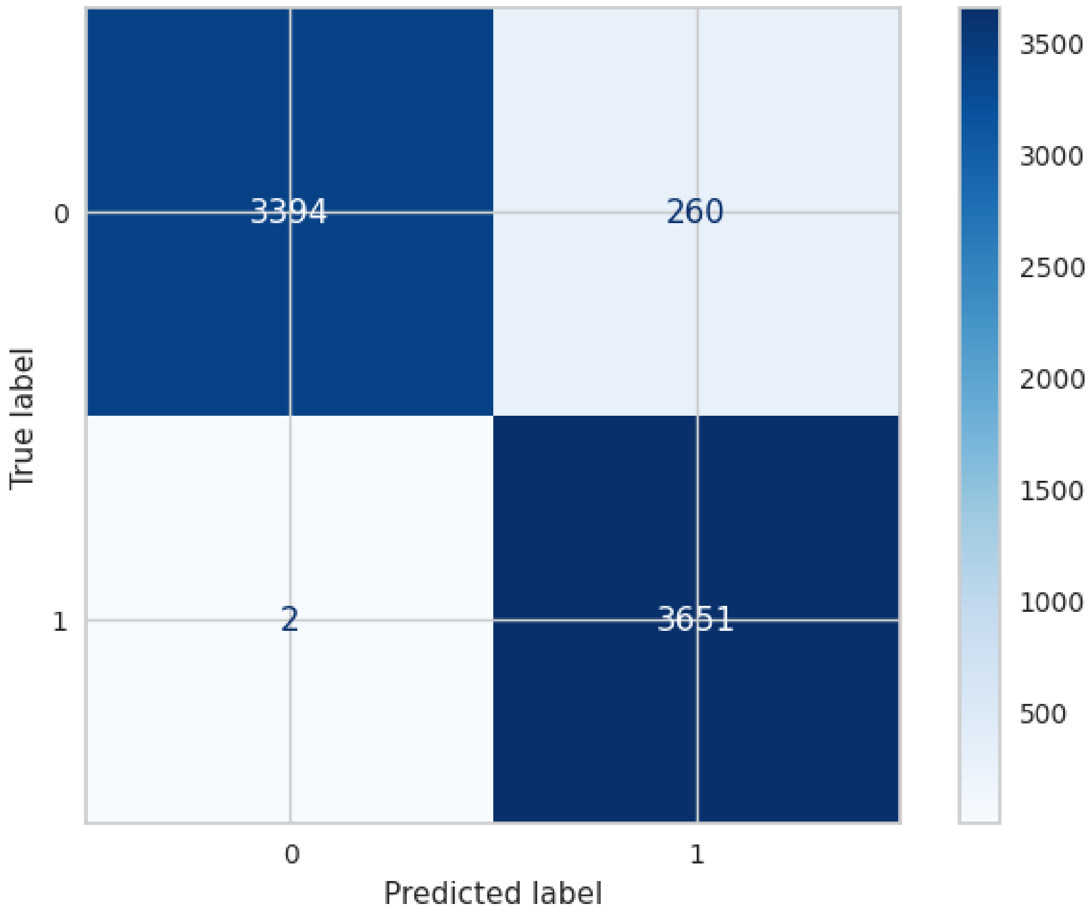
```
           1       0.95      1.00      0.98      3653

    accuracy                           0.97      7307
   macro avg       0.98      0.97      0.97      7307
weighted avg       0.98      0.97      0.97      7307

Confusion Matrix is :
 [[3471  183]
 [   0 3653]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9883977312470538
Model Test Score is :  0.9641439715341453
F1 Score is :  0.9653622421998942
Recall Score is :  0.9994525047905831
Precision Score is :  0.9335208386601892
AUC Value  :  0.9641488030247387
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.93      0.96      3654
           1       0.93      1.00      0.97      3653

    accuracy                           0.96      7307
   macro avg       0.97      0.96      0.96      7307
weighted avg       0.97      0.96      0.96      7307

Confusion Matrix is :
 [[3394  260]
 [   2 3651]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9742712467496921
F1 Score is :  0.974913263944489
Recall Score is :  1.0
Precision Score is :  0.9510544129133038
AUC Value  :  0.9742747673782157
Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.95      0.97      3654
           1       0.95      1.00      0.97      3653
```

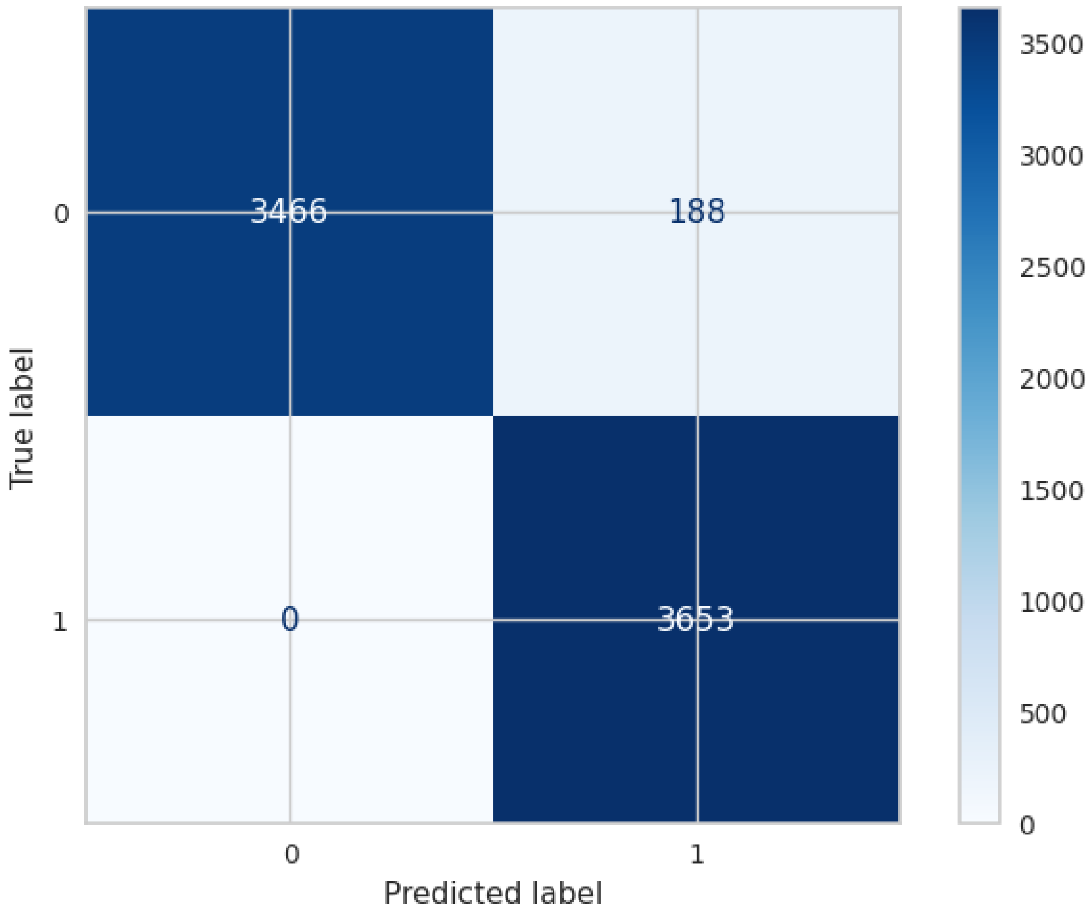
```
    accuracy                           0.97      7307
   macro avg       0.98      0.97      0.97      7307
weighted avg       0.98      0.97      0.97      7307

Confusion Matrix is :
 [[3466  188]
 [   0 3653]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9686601888599973
F1 Score is :  0.9696084936960849
Recall Score is :  1.0
Precision Score is :  0.9410097887686759
AUC Value  :  0.9686644772851669

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653

    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3425  229]
 [   0 3653]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9718078554810455
F1 Score is :  0.972577209797657
Recall Score is :  1.0
Precision Score is :  0.9466182948950506
AUC Value  :  0.9718117131910236

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.95      1.00      0.97      3653
```

```
    accuracy                           0.97      7307
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3448  206]
 [   0 3653]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.973860681538251
F1 Score is :  0.9745231425903694
Recall Score is :  1.0
Precision Score is :  0.950312174817898
AUC Value  :  0.973864258347017

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.95      0.97      3654
           1       0.95      1.00      0.97      3653

    accuracy                           0.97      7307
   macro avg       0.98      0.97      0.97      7307
weighted avg       0.98      0.97      0.97      7307

Confusion Matrix is :
 [[3463  191]
 [   0 3653]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9999239694052887
Model Test Score is :  0.9687970439304776
F1 Score is :  0.9697371913989913
Recall Score is :  1.0
Precision Score is :  0.9412522545735635
AUC Value  :  0.9688013136288998

Classification Report is :                precision    recall  f1-score
support

           0       1.00      0.94      0.97      3654
           1       0.94      1.00      0.97      3653

    accuracy                           0.97      7307
```

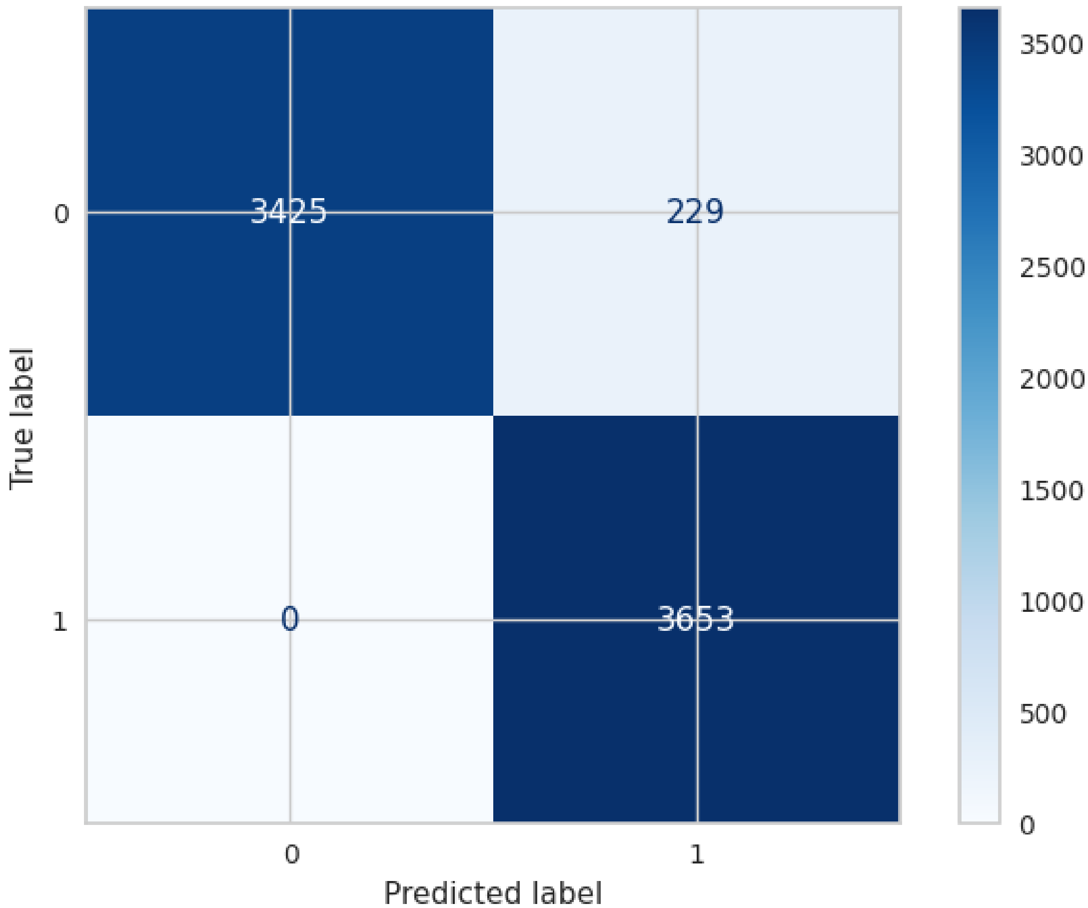
```
   macro avg       0.97      0.97      0.97      7307
weighted avg       0.97      0.97      0.97      7307

Confusion Matrix is :
 [[3426  228]
 [   0 3653]]
```



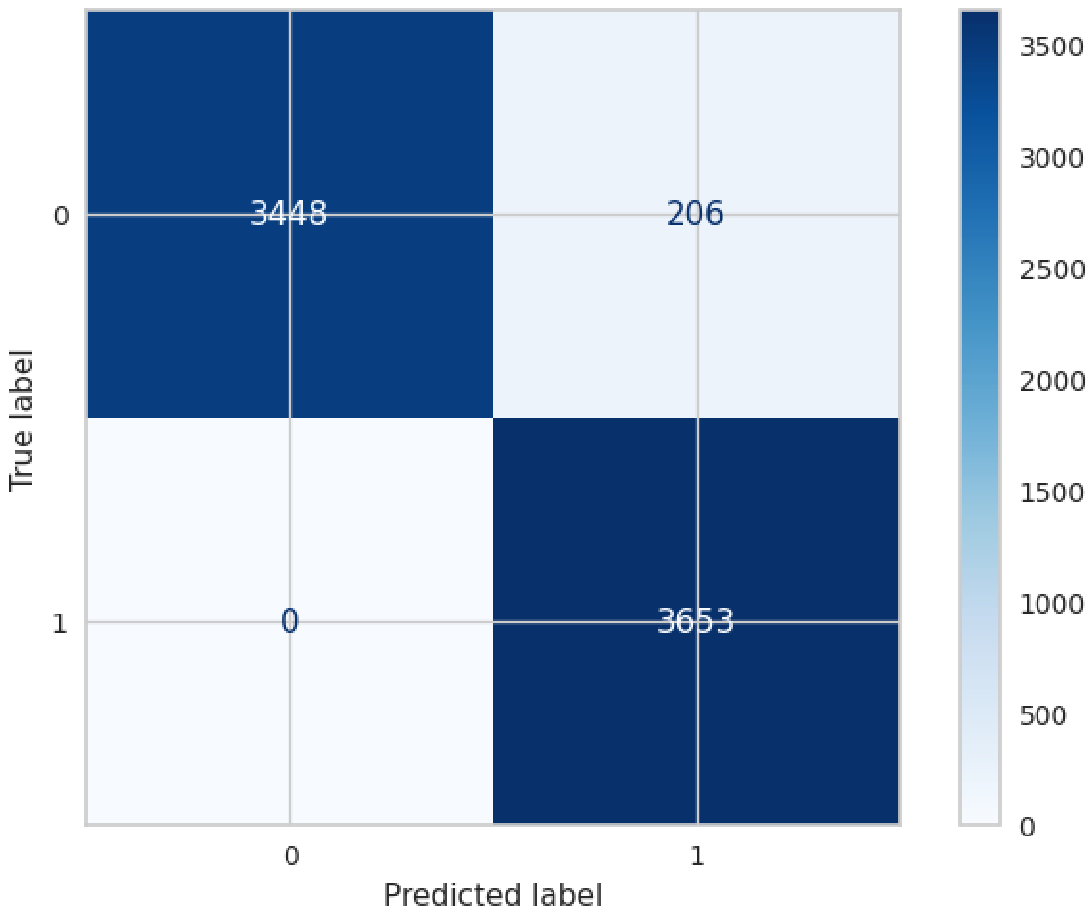

3394
260
2
3651
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500

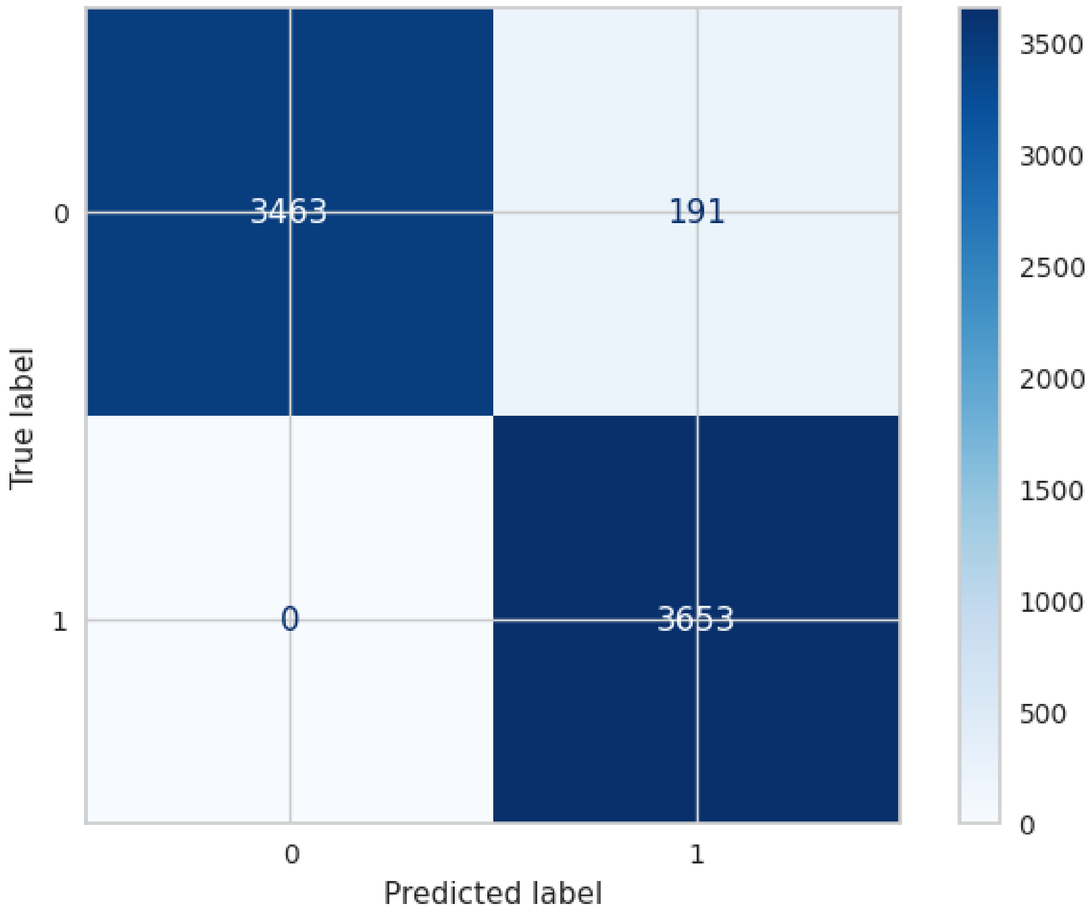


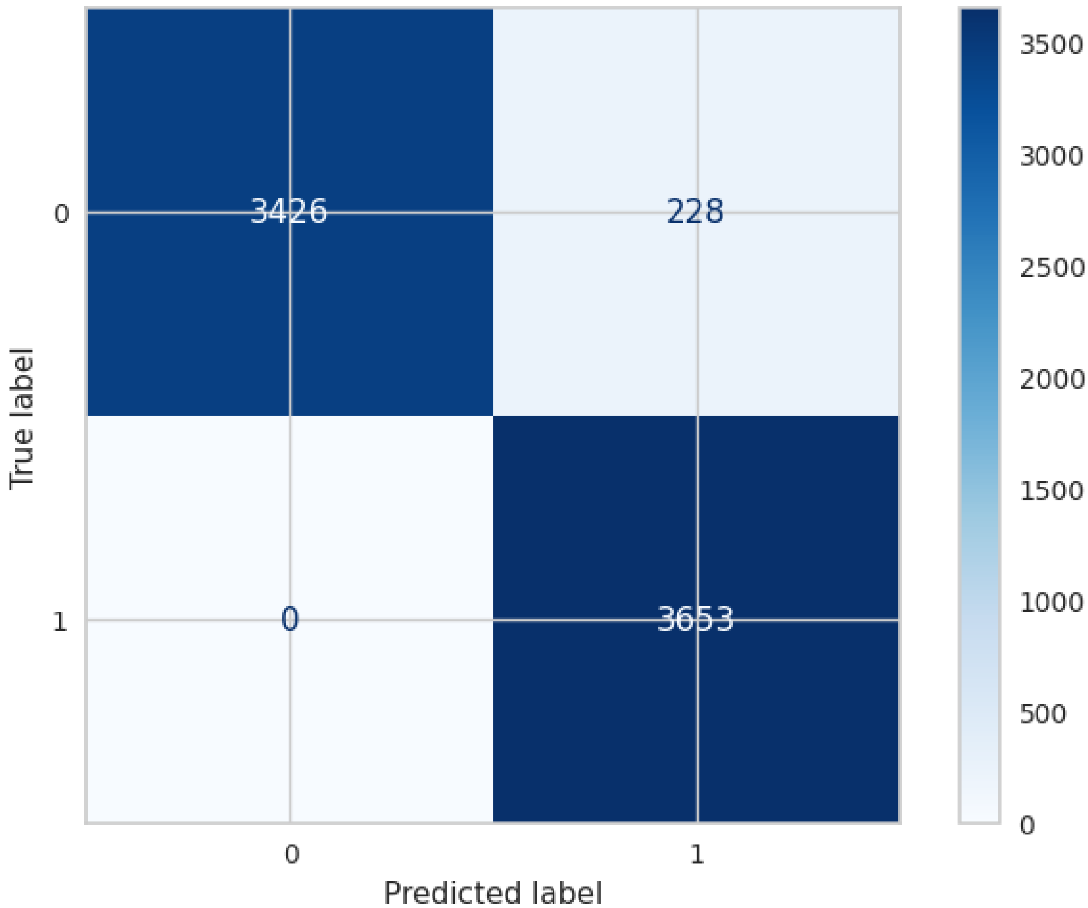
3466
188
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0



3425
229
0
3653
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500
0


3448
206
0
3653
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500
0


3463
191
0
3653
True label
Predicted label
0
1
0
1
3500
3000
2500
2000
1500
1000
500
0



[322]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Gradient Over','Gradient Over With Feature','Gradient Over␣
↪Scaling','Gradient Over With Normalize','Gradient Over With PCA'
                ,'Gradient Over With PCA and Scaling',
                'Gradient Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[322]:
```
                                      Train Accuracy  Test Accuracy   Test F1  \
Models
Gradient Over                               0.999924       0.974956  0.975564
Gradient Over With Feature                  0.988398       0.964144  0.965362
Gradient Over Scaling                       0.999924       0.974271  0.974913
Gradient Over With Normalize                0.999924       0.968660  0.969608
Gradient Over With PCA                      0.999924       0.971808  0.972577
Gradient Over With PCA and Scaling          0.999924       0.973861  0.974523
Gradient Over With PCA and Normalize        0.999924       0.968797  0.969737
```

```
                                      Test Recall  Test Precision       AUC
Models
Gradient Over                            1.000000        0.952294  0.974959
Gradient Over With Feature               0.999453        0.933521  0.964149
Gradient Over Scaling                    1.000000        0.951054  0.974275
Gradient Over With Normalize             1.000000        0.941010  0.968664
Gradient Over With PCA                   1.000000        0.946618  0.971812
Gradient Over With PCA and Scaling       1.000000        0.950312  0.973864
Gradient Over With PCA and Normalize     1.000000        0.941252  0.968801
```

[323]:
```
models_draw(df)
```

RandomUnderSampler

[324]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[325]:
```
Search(GradientBoostingClassifier(n_estimators=100,max_depth=3) ,{'max_depth':
↪[5,10,20,25,30,40]},X_train,y_train)
```

[325]:
```
GradientBoostingClassifier(max_depth=5)
```

[326]:
```
cross_validation(GradientBoostingClassifier(n_estimators=100,max_depth=5),X_train,y_train)
```

```
Train Score Value :  [0.93323353 0.93023952 0.93173653 0.92889222 0.93218563]
Mean 0.93125748502994
Test Score Value :  [0.88383234 0.88742515 0.88023952 0.88982036 0.88742515]
Mean 0.885748502994012
```

[327]:
```
Values =␣
↪Models(GradientBoostingClassifier(n_estimators=100,max_depth=5),X_train,y_train,X_test,y_te
```

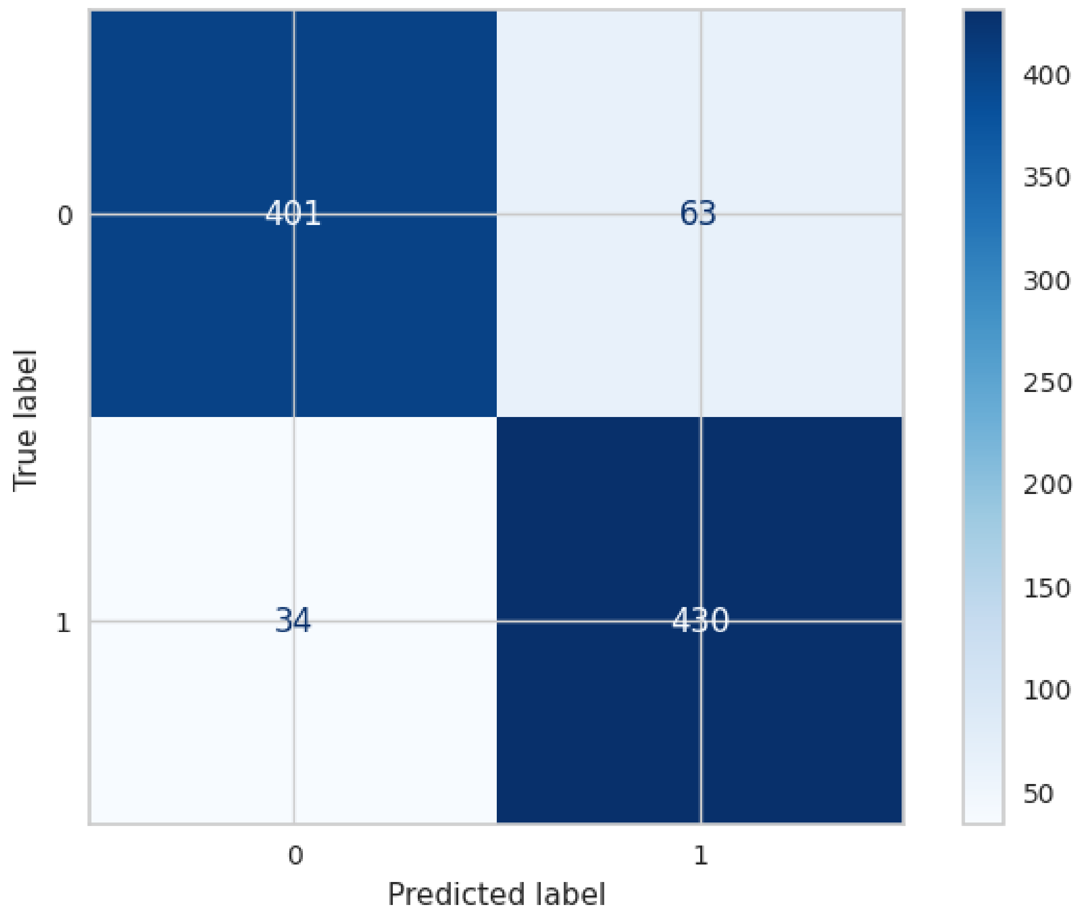
```
Apply Model With Normal Data :

Model Train Score is :  0.9279041916167665
Model Test Score is :  0.8954741379310345
F1 Score is :  0.8986415882967608
Recall Score is :  0.9267241379310345
Precision Score is :  0.8722109533468559
AUC Value  :  0.8954741379310345

Classification Report is :               precision    recall  f1-score
support

           0       0.92      0.86      0.89       464
```

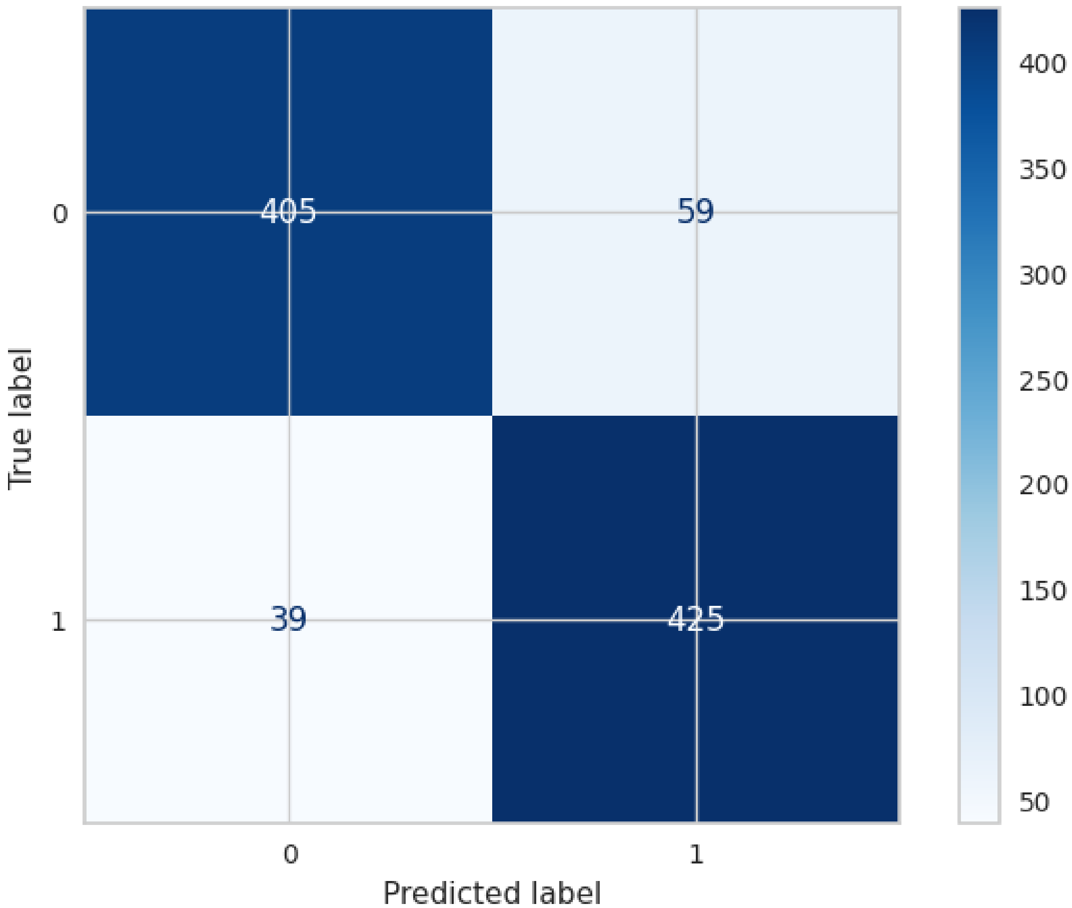
```
           1       0.87      0.93      0.90       464

    accuracy                           0.90       928
   macro avg       0.90      0.90      0.90       928
weighted avg       0.90      0.90      0.90       928

Confusion Matrix is :
 [[401  63]
 [ 34 430]]


 Apply Model With Feature Selection :

Model Train Score is :  0.9155688622754491
Model Test Score is :  0.8943965517241379
F1 Score is :  0.8966244725738397
Recall Score is :  0.915948275862069
Precision Score is :  0.878099173553719
AUC Value  :  0.8943965517241379
Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.87      0.89       464
           1       0.88      0.92      0.90       464

    accuracy                           0.89       928
   macro avg       0.90      0.89      0.89       928
weighted avg       0.90      0.89      0.89       928

Confusion Matrix is :
 [[405  59]
 [ 39 425]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9279041916167665
Model Test Score is :  0.8954741379310345
F1 Score is :  0.8986415882967608
Recall Score is :  0.9267241379310345
Precision Score is :  0.8722109533468559
AUC Value  :  0.8954741379310345
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.86      0.89       464
           1       0.87      0.93      0.90       464
```

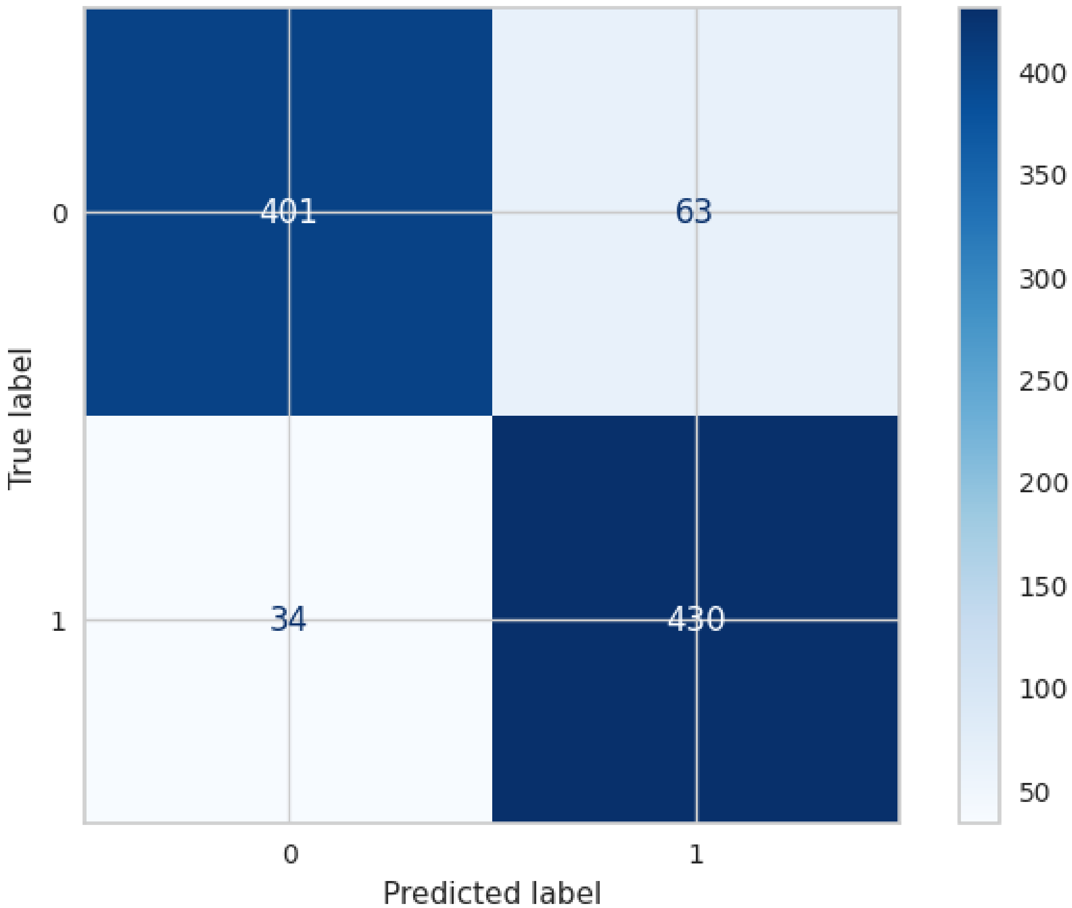
```
    accuracy                           0.90       928
   macro avg       0.90      0.90      0.90       928
weighted avg       0.90      0.90      0.90       928

Confusion Matrix is :
 [[401  63]
 [ 34 430]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.9221556886227545
Model Test Score is :  0.8954741379310345
F1 Score is :  0.8984293193717278
Recall Score is :  0.9245689655172413
Precision Score is :  0.8737270875763747
AUC Value  :  0.8954741379310345
Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.87      0.89       464
           1       0.87      0.92      0.90       464

    accuracy                           0.90       928
   macro avg       0.90      0.90      0.90       928
weighted avg       0.90      0.90      0.90       928

Confusion Matrix is :
 [[402  62]
 [ 35 429]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9295808383233533
Model Test Score is :  0.896551724137931
F1 Score is :  0.9002079002079002
Recall Score is :  0.9331896551724138
Precision Score is :  0.8694779116465864
AUC Value  :  0.896551724137931
Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.86      0.89       464
           1       0.87      0.93      0.90       464
```

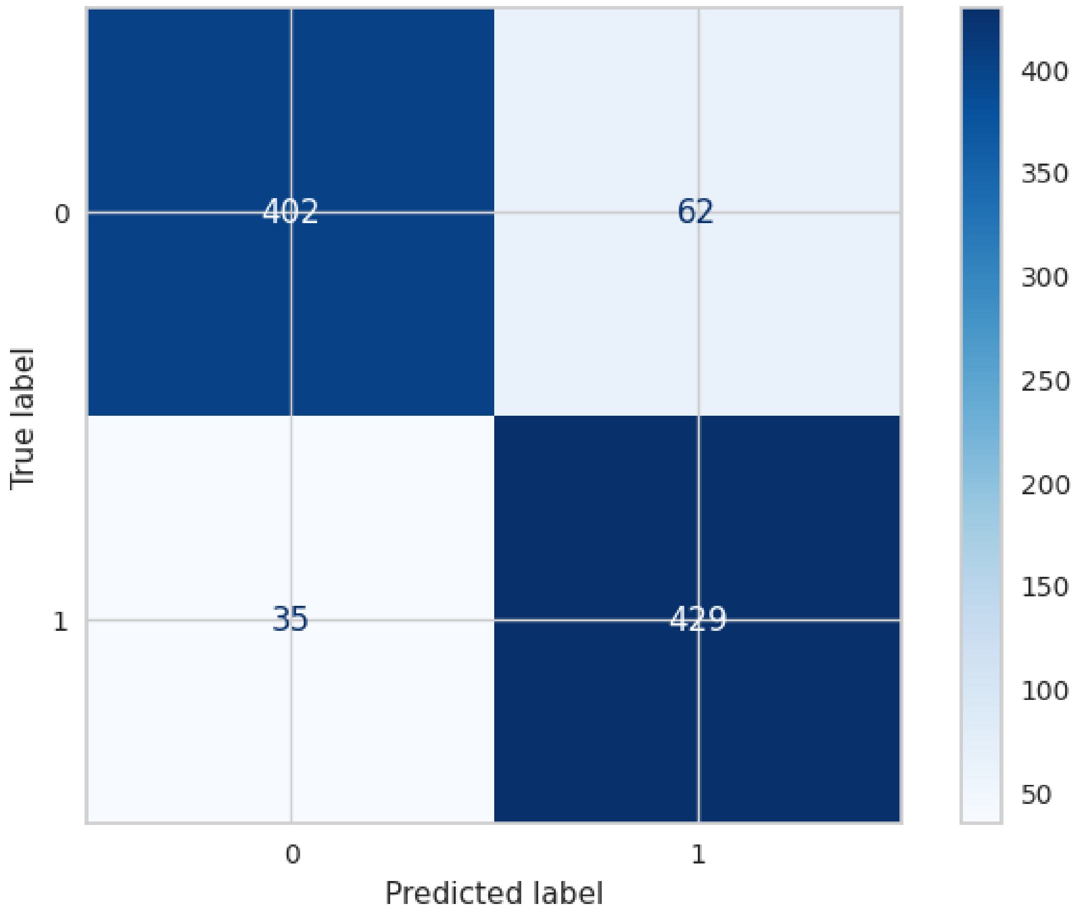
```
    accuracy                           0.90       928
   macro avg       0.90      0.90      0.90       928
weighted avg       0.90      0.90      0.90       928

Confusion Matrix is :
 [[399  65]
 [ 31 433]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9291017964071856
Model Test Score is :  0.8674568965517241
F1 Score is :  0.8714733542319748
Recall Score is :  0.8987068965517241
Precision Score is :  0.845841784989858
AUC Value  :  0.8674568965517242

Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.84      0.86       464
           1       0.85      0.90      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[388  76]
 [ 47 417]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.9305389221556887
Model Test Score is :  0.8900862068965517
F1 Score is :  0.893970893970894
Recall Score is :  0.9267241379310345
Precision Score is :  0.8634538152610441
AUC Value  :  0.8900862068965517

Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.85      0.89       464
           1       0.86      0.93      0.89       464

    accuracy                           0.89       928
```

```
   macro avg       0.89      0.89      0.89       928
weighted avg       0.89      0.89      0.89       928

Confusion Matrix is :
 [[396  68]
 [ 34 430]]
```

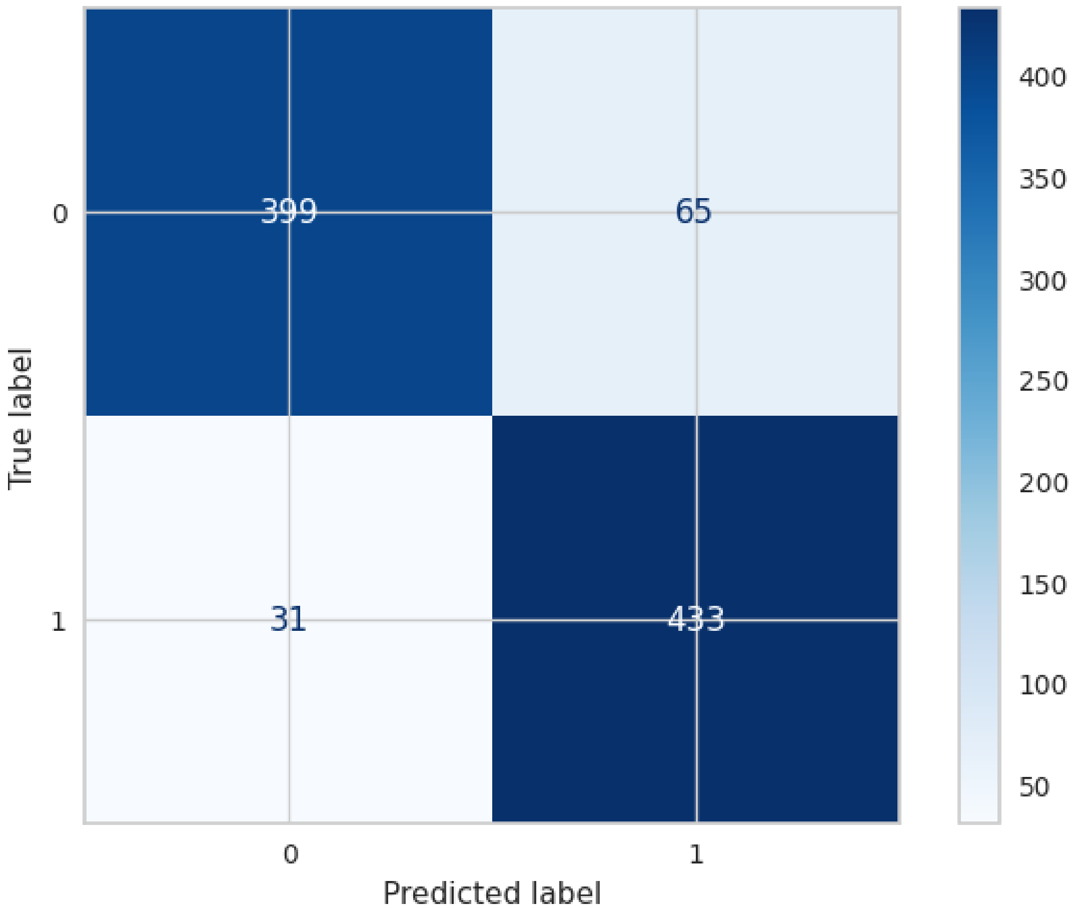



405
59
39
425
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50

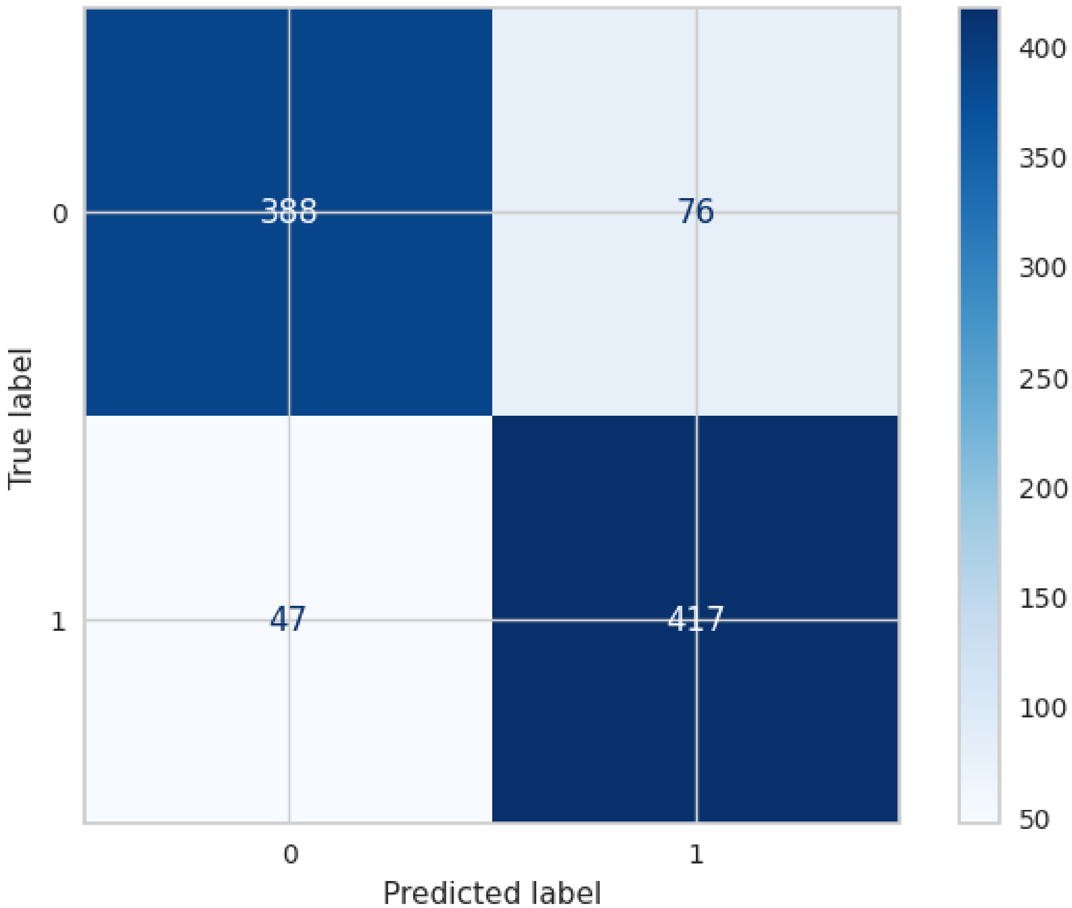

401
63
34
430
0
1
0
1
True label
Predicted label
400
350
300
250
200
150
100
50

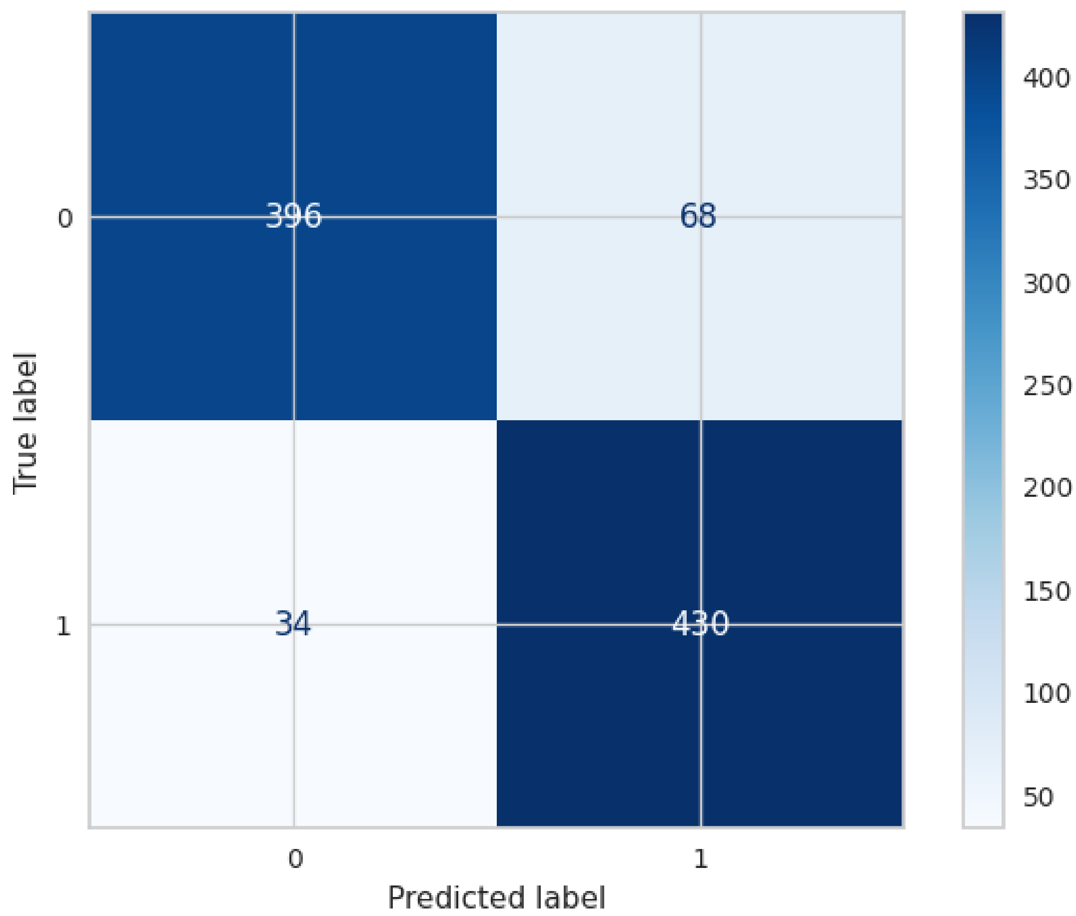


402
62
35
429
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50



True label
Predicted label
0
1
0
1
399
65
31
433
400
350
300
250
200
150
100
50


388
76
47
417
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50



[328]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Gradient Under','Gradient Under With Feature','Gradient Under␣
↪Scaling','Gradient Under With Normalize','Gradient Under With PCA'
                ,'Gradient Under With PCA and Scaling',
                'Gradient Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[328]:
```
                                       Train Accuracy  Test Accuracy  \
Models
Gradient Under                               0.927904       0.895474
Gradient Under With Feature                  0.915569       0.894397
Gradient Under Scaling                       0.927904       0.895474
Gradient Under With Normalize                0.922156       0.895474
Gradient Under With PCA                      0.929581       0.896552
Gradient Under With PCA and Scaling          0.929102       0.867457
Gradient Under With PCA and Normalize        0.930539       0.890086
```

```
                                      Test F1  Test Recall  Test Precision  \
Models
Gradient Under                       0.898642     0.926724        0.872211
Gradient Under With Feature          0.896624     0.915948        0.878099
Gradient Under Scaling               0.898642     0.926724        0.872211
Gradient Under With Normalize        0.898429     0.924569        0.873727
Gradient Under With PCA              0.900208     0.933190        0.869478
Gradient Under With PCA and Scaling  0.871473     0.898707        0.845842
Gradient Under With PCA and Normalize 0.893971    0.926724        0.863454

                                          AUC
Models
Gradient Under                       0.895474
Gradient Under With Feature          0.894397
Gradient Under Scaling               0.895474
Gradient Under With Normalize        0.895474
Gradient Under With PCA              0.896552
Gradient Under With PCA and Scaling  0.867457
Gradient Under With PCA and Normalize 0.890086
```

[329]:
```
models_draw(df)
```

SGDClassifier

[330]:
```
X_train,y_train,X_test,y_test=Split(X_classification,y_classification)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[331]:
```
cross_validation(SGDClassifier(penalty='l2'),X_train,y_train)
```

```
Train Score Value :  [0.88756578 0.85724405 0.90386237 0.79777365 0.90612245]
Mean 0.870513662106801
Test Score Value :  [0.88761468 0.85656457 0.90325192 0.79355013 0.9117528 ]
Mean 0.8705468191963595
```

[332]:
```
Values = Models(SGDClassifier(penalty='l2'),X_train,y_train,X_test,y_test)
```

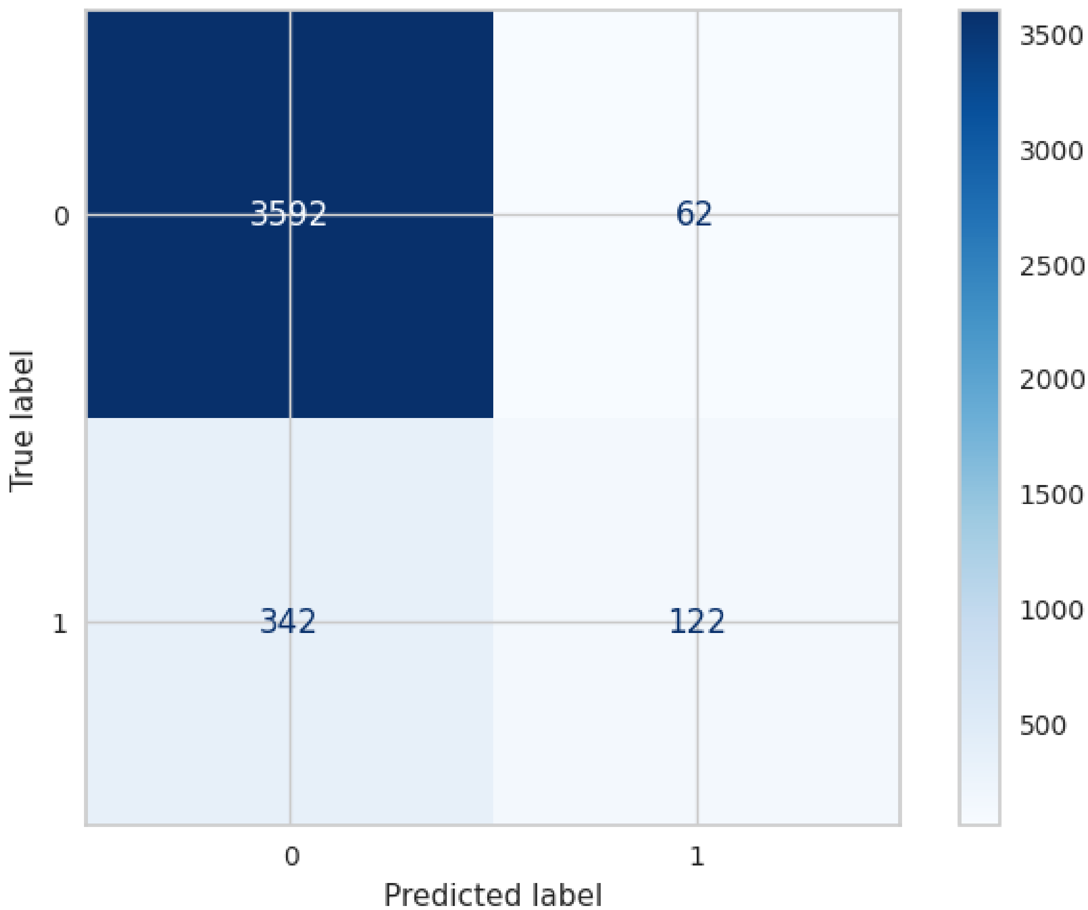
```
Apply Model With Normal Data :

Model Train Score is :  0.9047927461139896
Model Test Score is :  0.9018941233608548
F1 Score is :  0.3765432098765432
Recall Score is :  0.2629310344827586
Precision Score is :  0.6630434782608695
AUC Value  :  0.6229816639299398
```

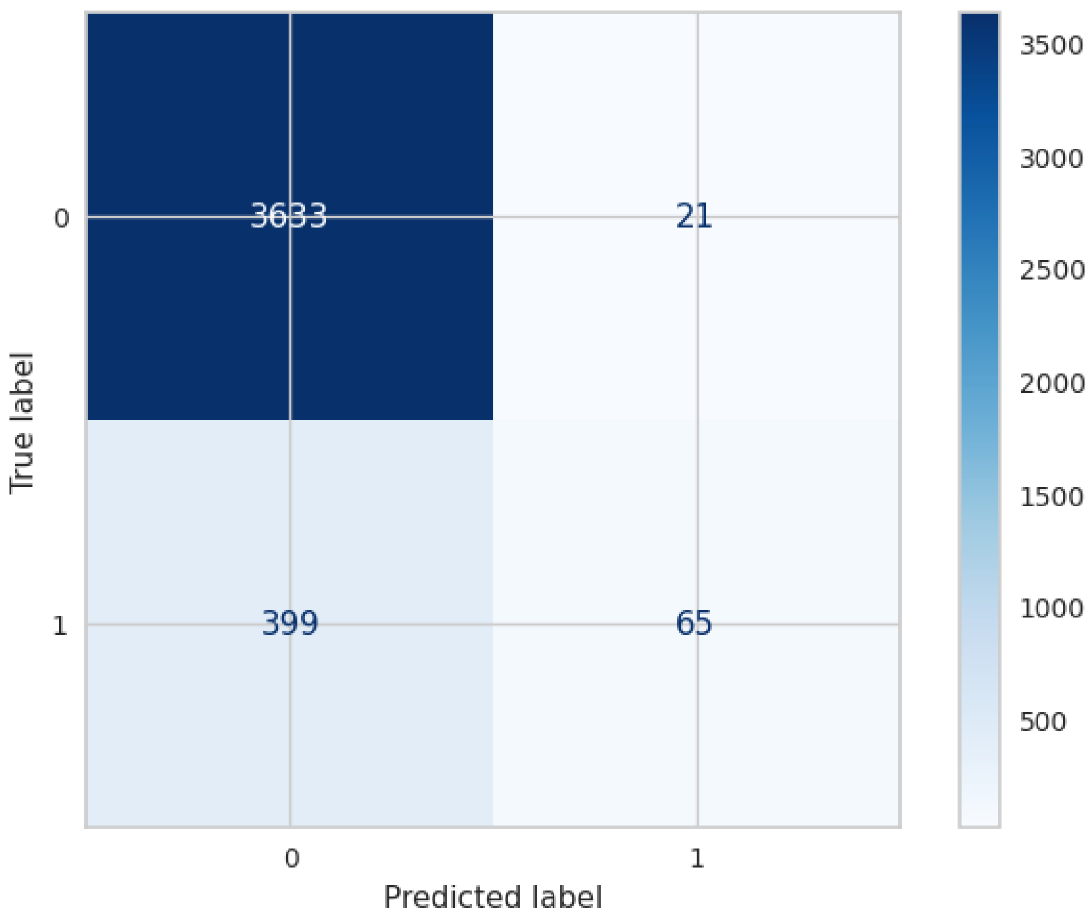
```
Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.98      0.95      3654
           1       0.66      0.26      0.38       464

    accuracy                           0.90      4118
   macro avg       0.79      0.62      0.66      4118
weighted avg       0.88      0.90      0.88      4118

Confusion Matrix is :
 [[3592   62]
 [ 342  122]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8966429188255614
Model Test Score is :  0.8980087421078193
F1 Score is :  0.23636363636363636
Recall Score is :  0.1400862068965517
Precision Score is :  0.7558139534883721
AUC Value  :  0.567169540229885
Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.99      0.95      3654
           1       0.76      0.14      0.24       464

    accuracy                           0.90      4118
   macro avg       0.83      0.57      0.59      4118
weighted avg       0.88      0.90      0.87      4118

Confusion Matrix is :
 [[3633   21]
 [ 399   65]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.9043339810017271
Model Test Score is :  0.9028654686741137
F1 Score is :  0.34853420195439744
Recall Score is :  0.23060344827586207
Precision Score is :  0.7133333333333334
AUC Value  :  0.6094177613574165
Classification Report is :                precision    recall  f1-score
```

```
support

           0       0.91      0.99      0.95      3654
           1       0.71      0.23      0.35       464

    accuracy                           0.90      4118
   macro avg       0.81      0.61      0.65      4118
weighted avg       0.89      0.90      0.88      4118

Confusion Matrix is :
 [[3611   43]
 [ 357  107]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.897020725388601
Model Test Score is :  0.9006799417192812
F1 Score is :  0.2911611785095321
Recall Score is :  0.1810344827586207
Precision Score is :  0.7433628318584071
AUC Value  :  0.5865489874110564
Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.99      0.95      3654
           1       0.74      0.18      0.29       464

    accuracy                           0.90      4118
   macro avg       0.82      0.59      0.62      4118
weighted avg       0.89      0.90      0.87      4118

Confusion Matrix is :
 [[3625   29]
 [ 380   84]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.9051975388601037
Model Test Score is :  0.9021369596891695
F1 Score is :  0.49813200498132004
Recall Score is :  0.43103448275862066
Precision Score is :  0.5899705014749262
AUC Value  :  0.6964969896004378
Classification Report is :                precision    recall  f1-score
support
```

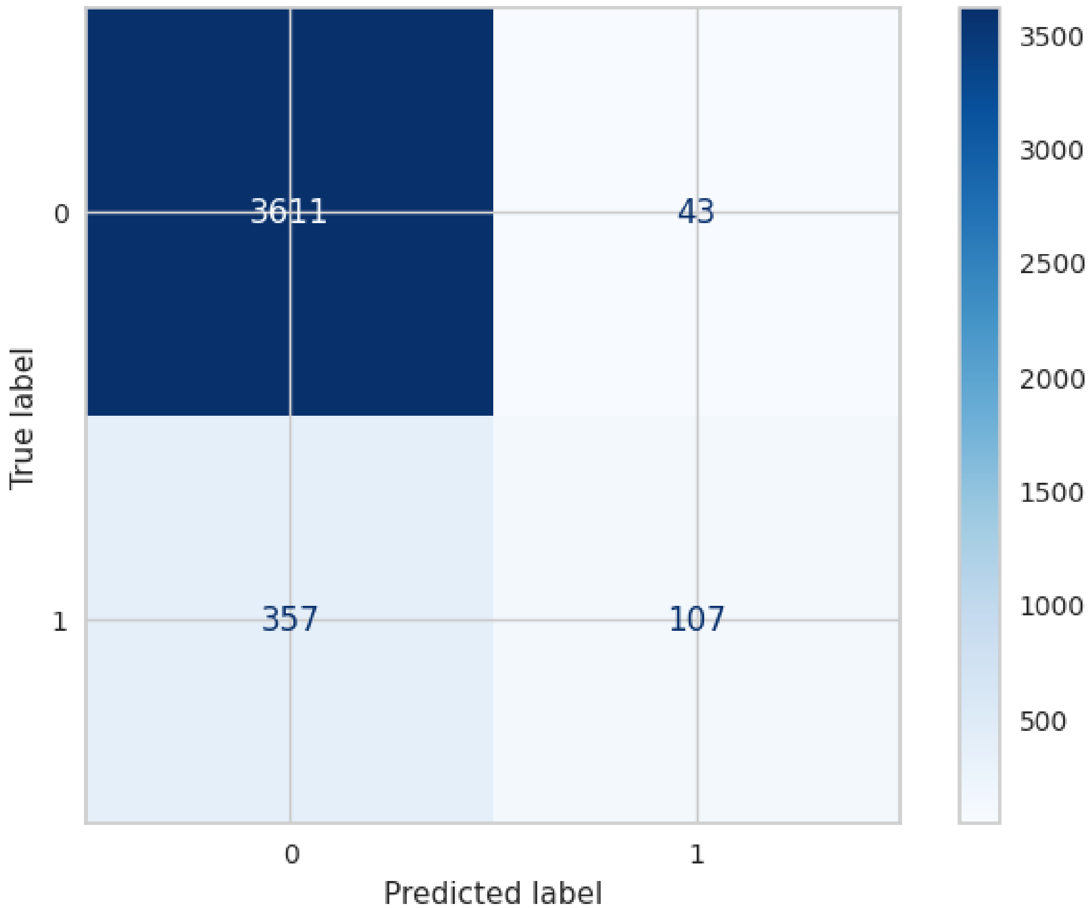
```
          0       0.93      0.96      0.95      3654
          1       0.59      0.43      0.50       464

   accuracy                           0.90      4118
  macro avg       0.76      0.70      0.72      4118
weighted avg      0.89      0.90      0.90      4118

Confusion Matrix is :
 [[3515  139]
 [ 264  200]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.9054674006908463
Model Test Score is :  0.9050509956289461
F1 Score is :  0.384251968503937
Recall Score is :  0.2629310344827586
Precision Score is :  0.7134502923976608
AUC Value  :  0.6247605363984675

Classification Report is :                precision    recall  f1-score
support

          0       0.91      0.99      0.95      3654
          1       0.71      0.26      0.38       464

   accuracy                           0.91      4118
  macro avg       0.81      0.62      0.67      4118
weighted avg      0.89      0.91      0.88      4118

Confusion Matrix is :
 [[3605   49]
 [ 342  122]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8955634715025906
Model Test Score is :  0.8977659057795047
F1 Score is :  0.24144144144144145
Recall Score is :  0.14439655172413793
Precision Score is :  0.7362637362637363
AUC Value  :  0.5689142036124795

Classification Report is :                precision    recall  f1-score
support
```

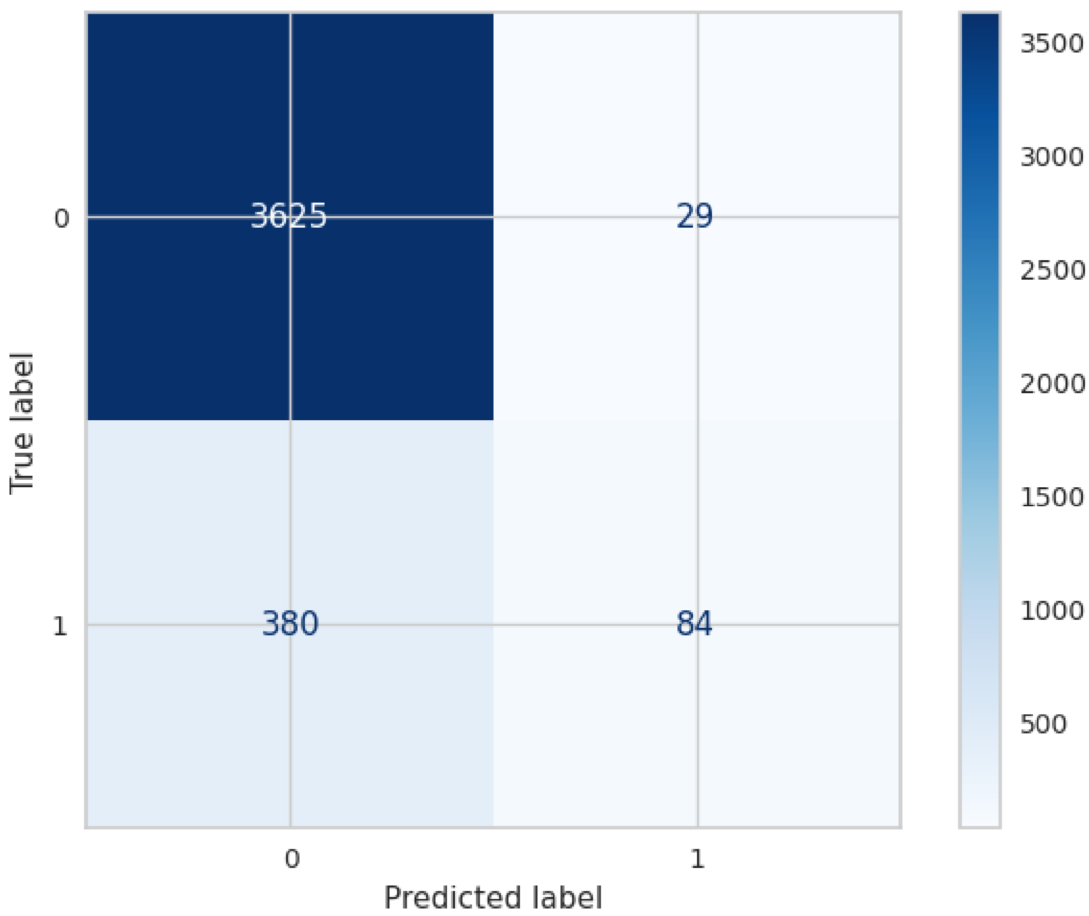
```
           0       0.90      0.99      0.95      3654
           1       0.74      0.14      0.24       464

    accuracy                           0.90      4118
   macro avg       0.82      0.57      0.59      4118
weighted avg       0.88      0.90      0.87      4118

Confusion Matrix is :
 [[3630   24]
 [ 397   67]]
```

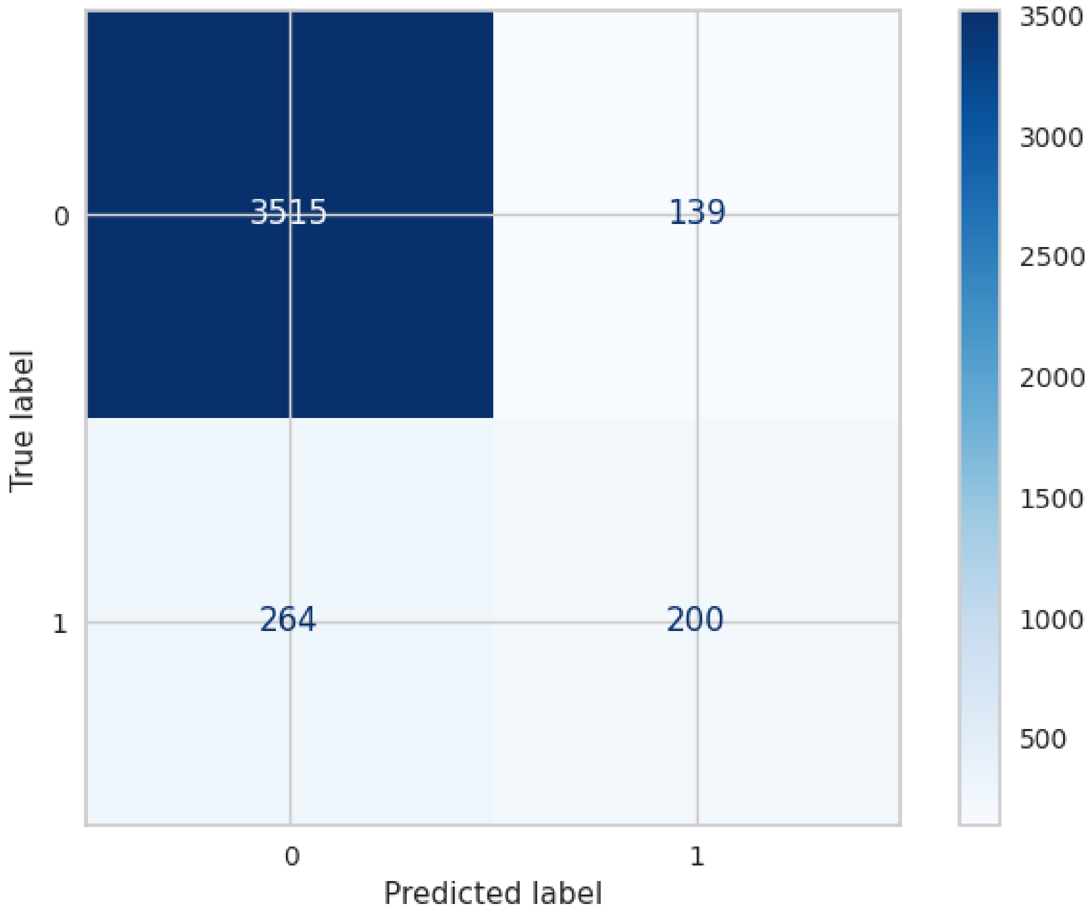


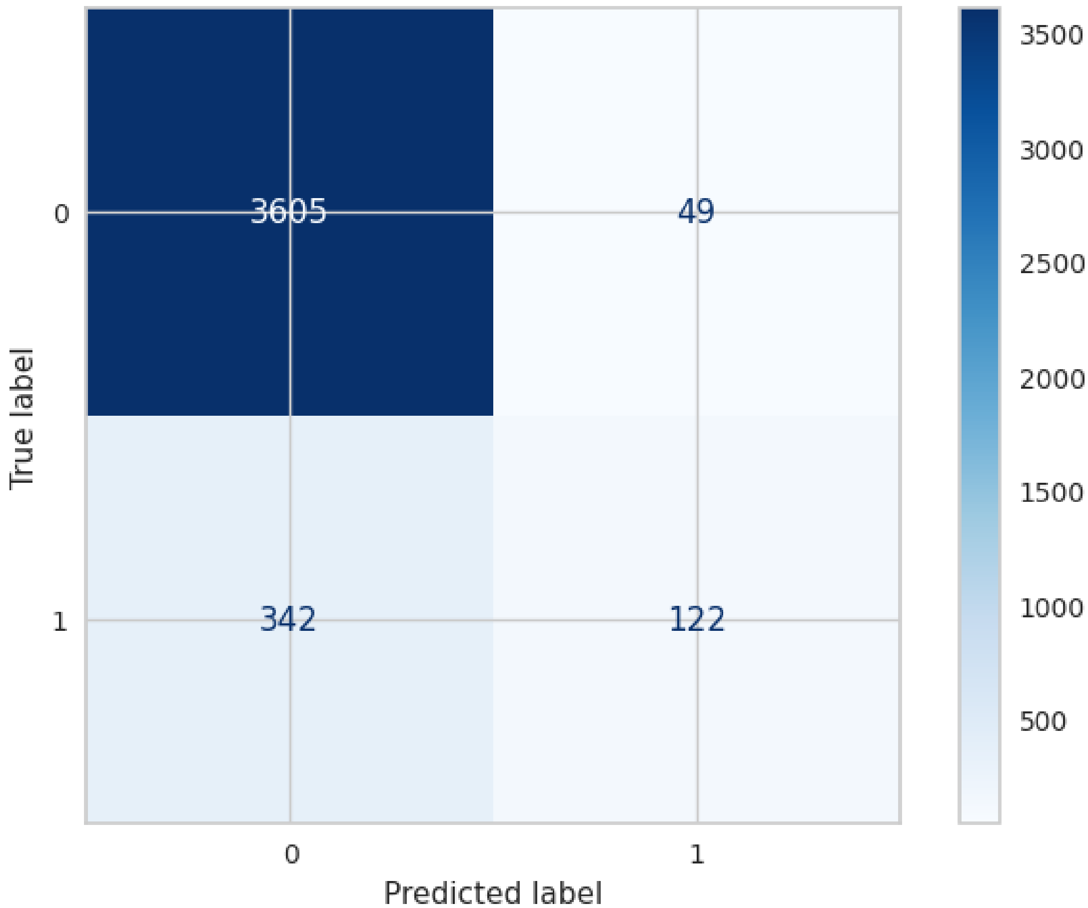


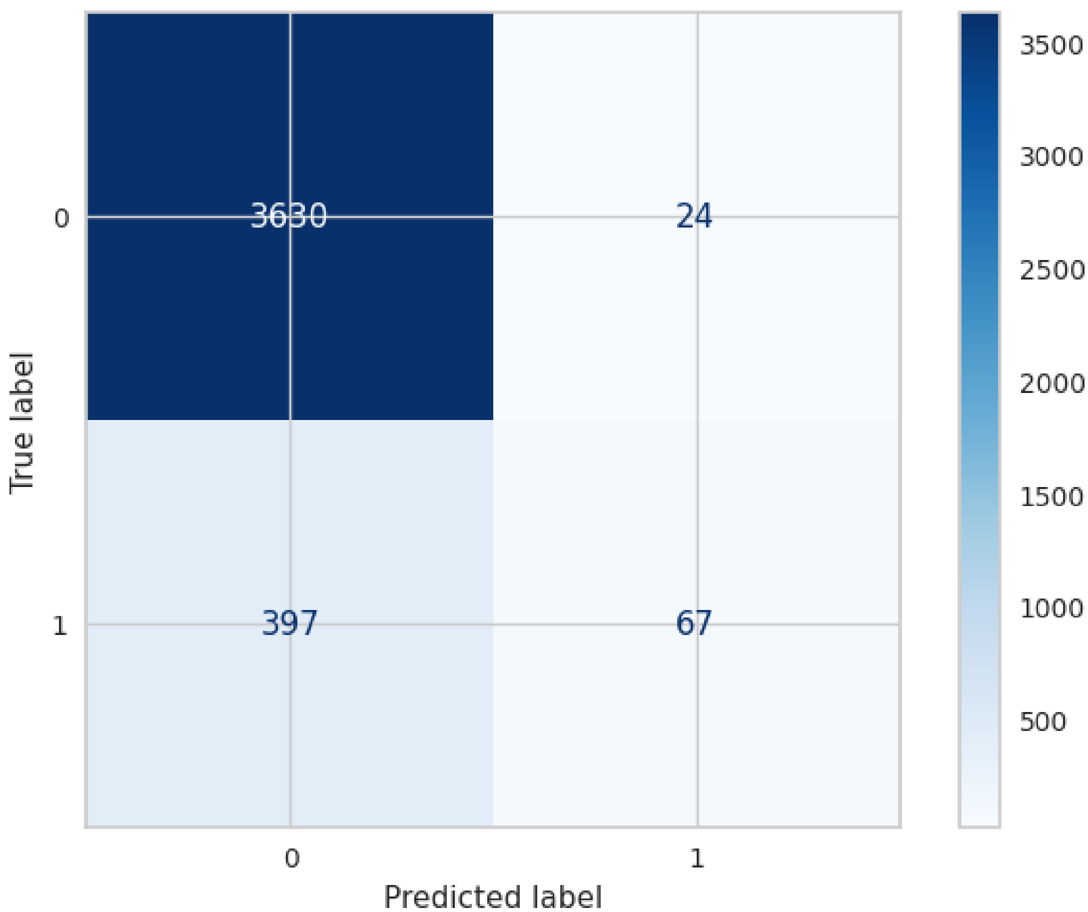
3633
21
399
65
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3611
43
357
107
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3625
29
380
84
0
1
True label
Predicted label
3500
3000
2500
2000
1500
1000
500


3515
139
264
200
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500


3605
49
342
122
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500



[333]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['SGD','SGD With Feature','SGD Scaling','SGD With␣
↪Normalize','SGD With PCA'
                ,'SGD With PCA and Scaling',
                'SGD With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[333]:
```
                            Train Accuracy  Test Accuracy   Test F1  \
Models
SGD                               0.904793       0.901894  0.376543
SGD With Feature                  0.896643       0.898009  0.236364
SGD Scaling                       0.904334       0.902865  0.348534
SGD With Normalize                0.897021       0.900680  0.291161
SGD With PCA                      0.905198       0.902137  0.498132
SGD With PCA and Scaling          0.905467       0.905051  0.384252
SGD With PCA and Normalize        0.895563       0.897766  0.241441
```

```
                              Test Recall  Test Precision       AUC
Models
SGD                              0.262931        0.663043  0.622982
SGD With Feature                 0.140086        0.755814  0.567170
SGD Scaling                      0.230603        0.713333  0.609418
SGD With Normalize               0.181034        0.743363  0.586549
SGD With PCA                     0.431034        0.589971  0.696497
SGD With PCA and Scaling         0.262931        0.713450  0.624761
SGD With PCA and Normalize       0.144397        0.736264  0.568914
```

[334]:
```
models_draw(df)
```

RandomOverSampler

[335]:
```
X_train,y_train,X_test,y_test=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[336]:
```
cross_validation(SGDClassifier(penalty='l2'),X_train,y_train)
```

```
Train Score Value :  [0.71668884 0.75460939 0.84295761 0.76181787 0.80361521]
Mean 0.7759377858464764
Test Score Value :  [0.71899947 0.75830609 0.84452216 0.76125304 0.8053528 ]
Mean 0.7776867118655731
```

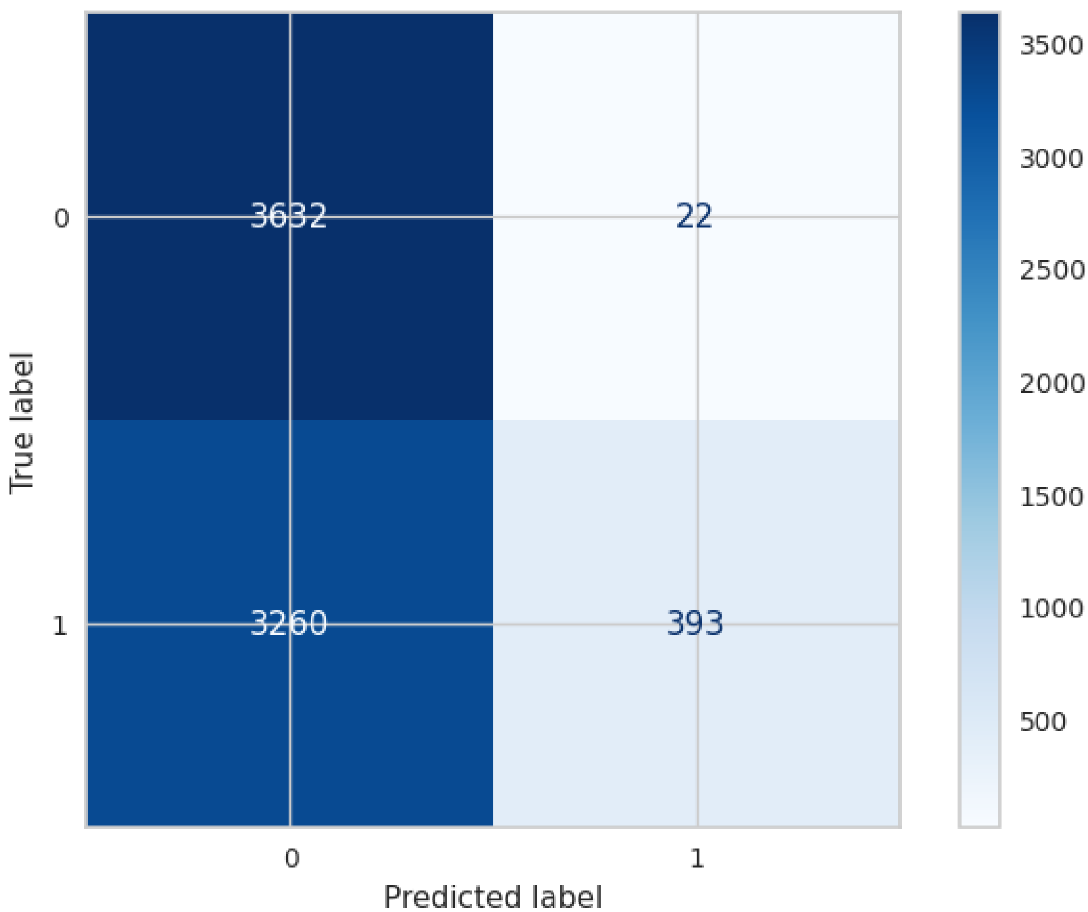
[337]:
```
Values = Models(SGDClassifier(penalty='l2'),X_train,y_train,X_test,y_test)
```

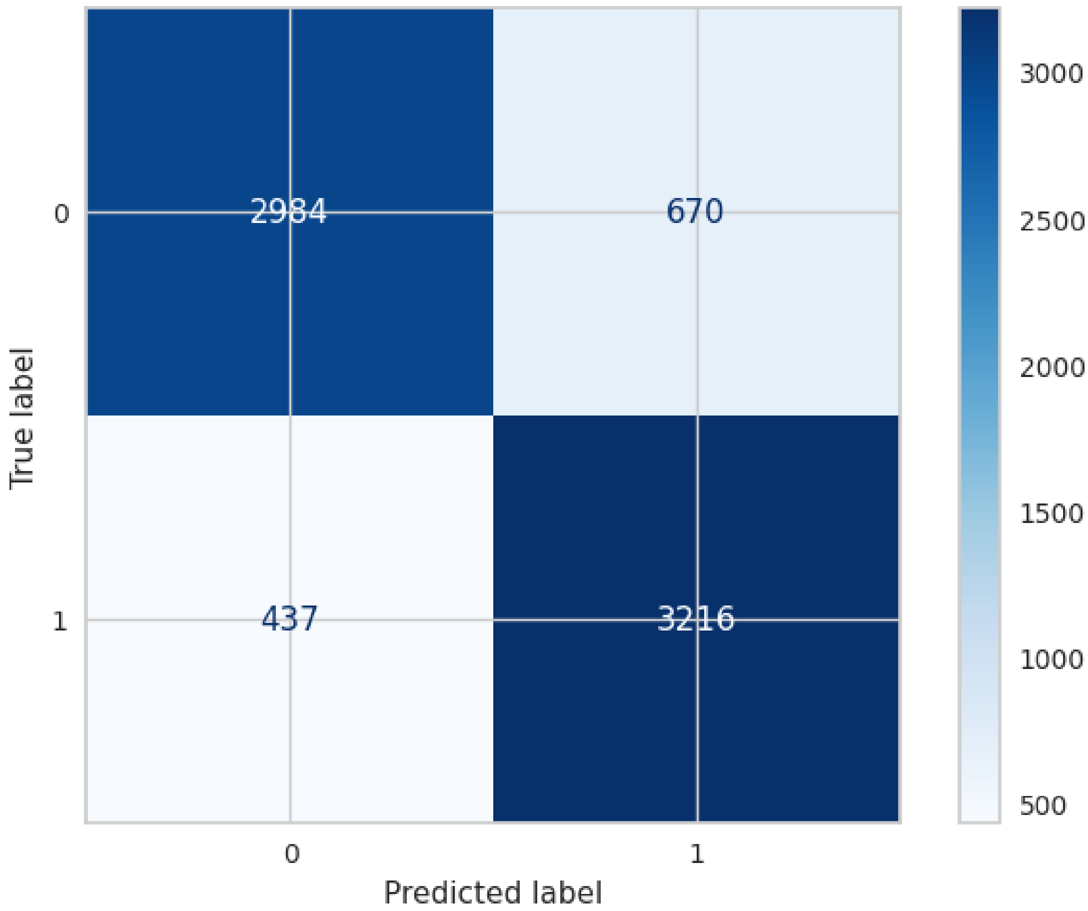
```
Apply Model With Normal Data :

Model Train Score is :  0.54839347353375
Model Test Score is :  0.5508416586834542
F1 Score is :  0.19321533923303835
Recall Score is :  0.10758280865042431
Precision Score is :  0.946987951807229
AUC Value  :  0.5507810047630884

Classification Report is :                precision    recall  f1-score
support

           0       0.53      0.99      0.69      3654
           1       0.95      0.11      0.19      3653

    accuracy                           0.55      7307
   macro avg       0.74      0.55      0.44      7307
weighted avg       0.74      0.55      0.44      7307
```

```
Confusion Matrix is :
 [[3632   22]
 [3260  393]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8501893161808312
Model Test Score is :  0.8485014369782401
F1 Score is :  0.8531635495423796
Recall Score is :  0.8803722967424035
Precision Score is :  0.8275862068965517
AUC Value  :  0.8485057980701618

Classification Report is :                precision    recall  f1-score
support

           0       0.87      0.82      0.84      3654
           1       0.83      0.88      0.85      3653

    accuracy                           0.85      7307
   macro avg       0.85      0.85      0.85      7307
weighted avg       0.85      0.85      0.85      7307

Confusion Matrix is :
 [[2984  670]
 [ 437 3216]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8633274029469459
Model Test Score is :  0.8631449295196387
F1 Score is :  0.8703319502074688
Recall Score is :  0.9186969614015877
Precision Score is :  0.8268046316826805
AUC Value  :  0.8631525310565683

Classification Report is :                precision    recall  f1-score
support

           0       0.91      0.81      0.86      3654
           1       0.83      0.92      0.87      3653

    accuracy                           0.86      7307
   macro avg       0.87      0.86      0.86      7307
weighted avg       0.87      0.86      0.86      7307

Confusion Matrix is :
```

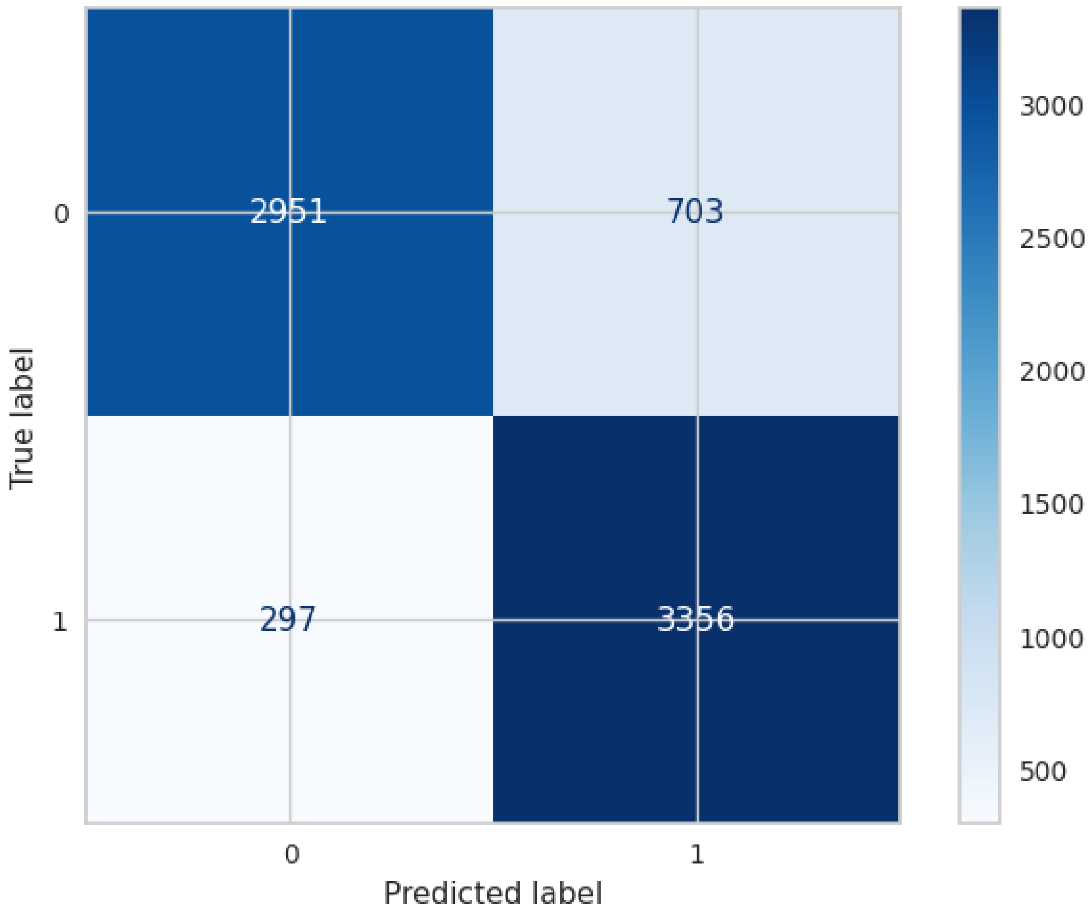
```
 [[2951  703]
 [ 297 3356]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8389215820446148
Model Test Score is :  0.8304365676748323
F1 Score is :  0.8360026472534745
Recall Score is :  0.8644949356693129
Precision Score is :  0.8093285494618144
AUC Value  :  0.8304412280973822
Classification Report is :                precision    recall  f1-score
support

           0       0.85      0.80      0.82      3654
           1       0.81      0.86      0.84      3653

    accuracy                           0.83      7307
   macro avg       0.83      0.83      0.83      7307
weighted avg       0.83      0.83      0.83      7307

Confusion Matrix is :
 [[2910  744]
 [ 495 3158]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8332040813223242
Model Test Score is :  0.8355002052826057
F1 Score is :  0.835116598079561
Recall Score is :  0.8332877087325485
Precision Score is :  0.836953533131702
AUC Value  :  0.8354999025326673
Classification Report is :                precision    recall  f1-score
support

           0       0.83      0.84      0.84      3654
           1       0.84      0.83      0.84      3653

    accuracy                           0.84      7307
   macro avg       0.84      0.84      0.84      7307
weighted avg       0.84      0.84      0.84      7307

Confusion Matrix is :
 [[3061  593]
```

```
 [ 609 3044]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8616851421011815
Model Test Score is :  0.8587655672642671
F1 Score is :  0.8640674394099052
Recall Score is :  0.8978921434437449
Precision Score is :  0.8326986544808327
AUC Value  :  0.8587709212018945
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.82      0.85      3654
           1       0.83      0.90      0.86      3653

    accuracy                           0.86      7307
   macro avg       0.86      0.86      0.86      7307
weighted avg       0.86      0.86      0.86      7307

Confusion Matrix is :
 [[2995  659]
 [ 373 3280]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8362148928728921
Model Test Score is :  0.8282468865471466
F1 Score is :  0.8360976883897089
Recall Score is :  0.8762660826717766
Precision Score is :  0.7994505494505495
AUC Value  :  0.8282534573183732
Classification Report is :                precision    recall  f1-score
support

           0       0.86      0.78      0.82      3654
           1       0.80      0.88      0.84      3653

    accuracy                           0.83      7307
   macro avg       0.83      0.83      0.83      7307
weighted avg       0.83      0.83      0.83      7307

Confusion Matrix is :
 [[2851  803]
 [ 452 3201]]
```

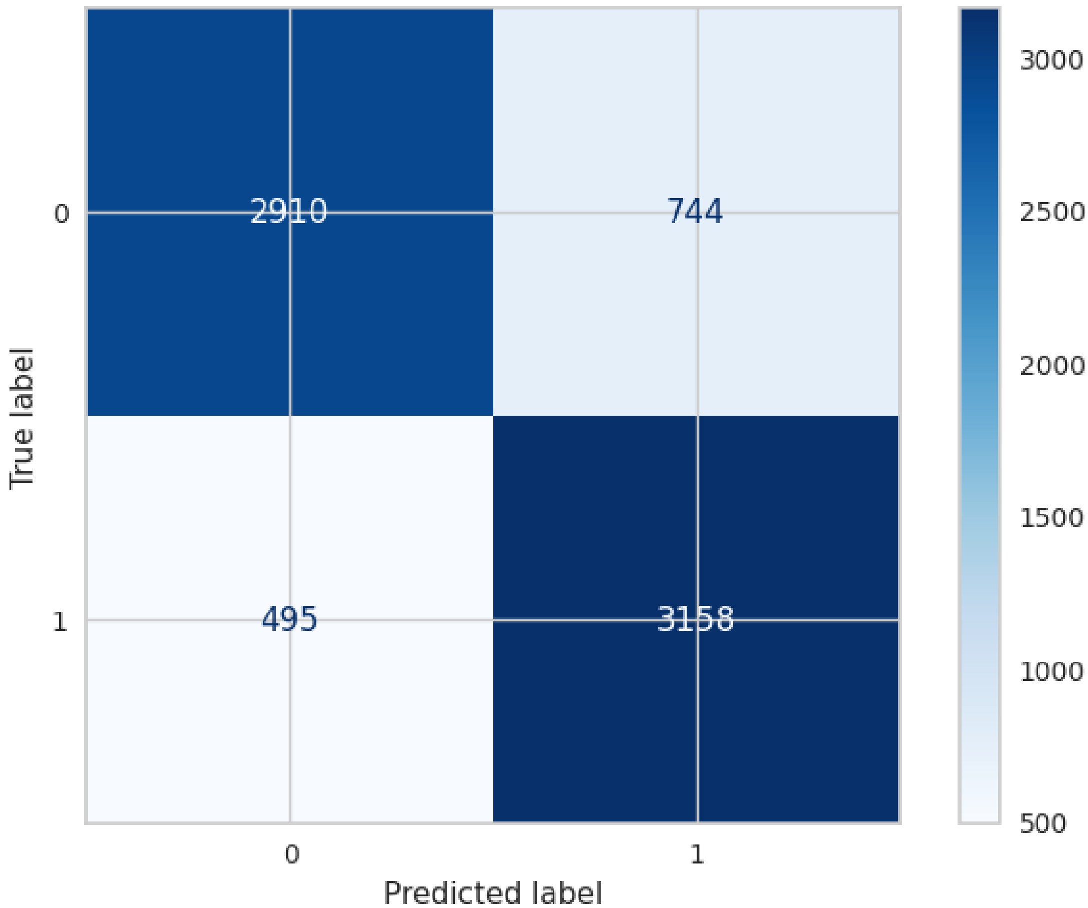

3632
22
3260
393
0
1
True label
0
1
Predicted label
3500
3000
2500
2000
1500
1000
500

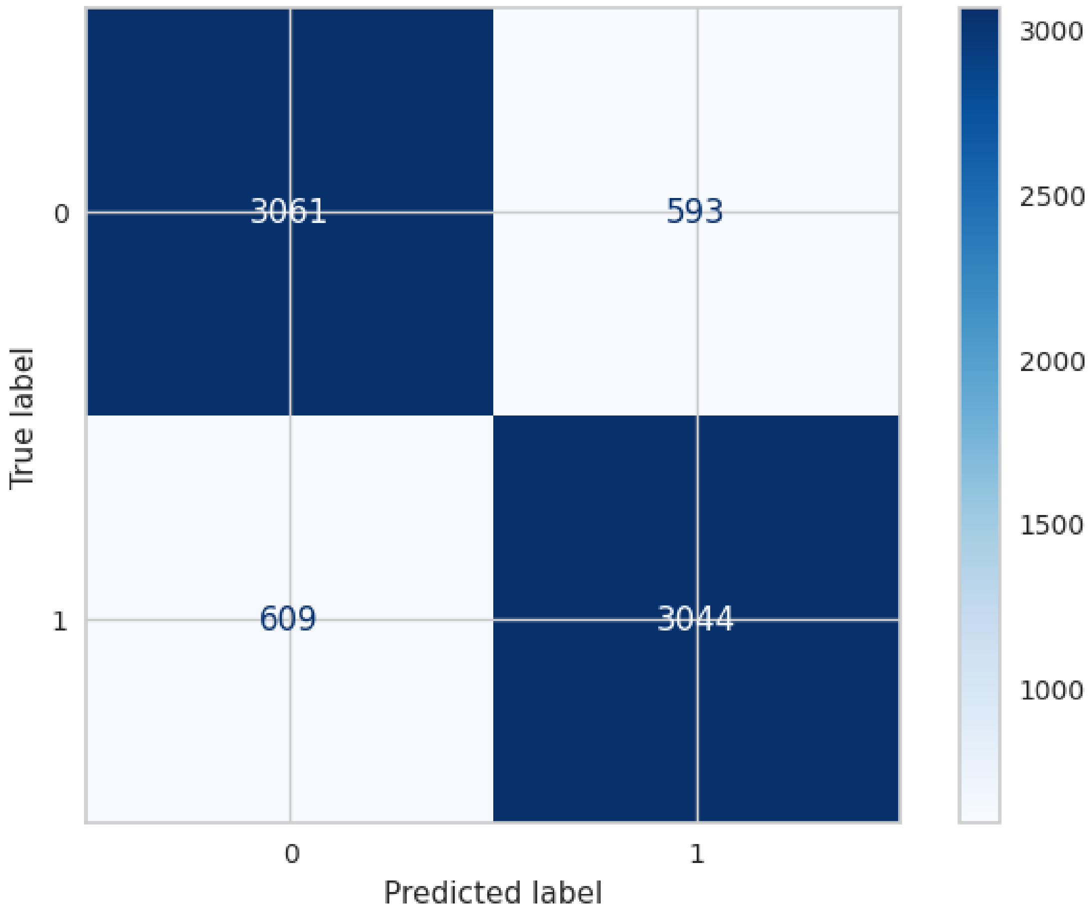


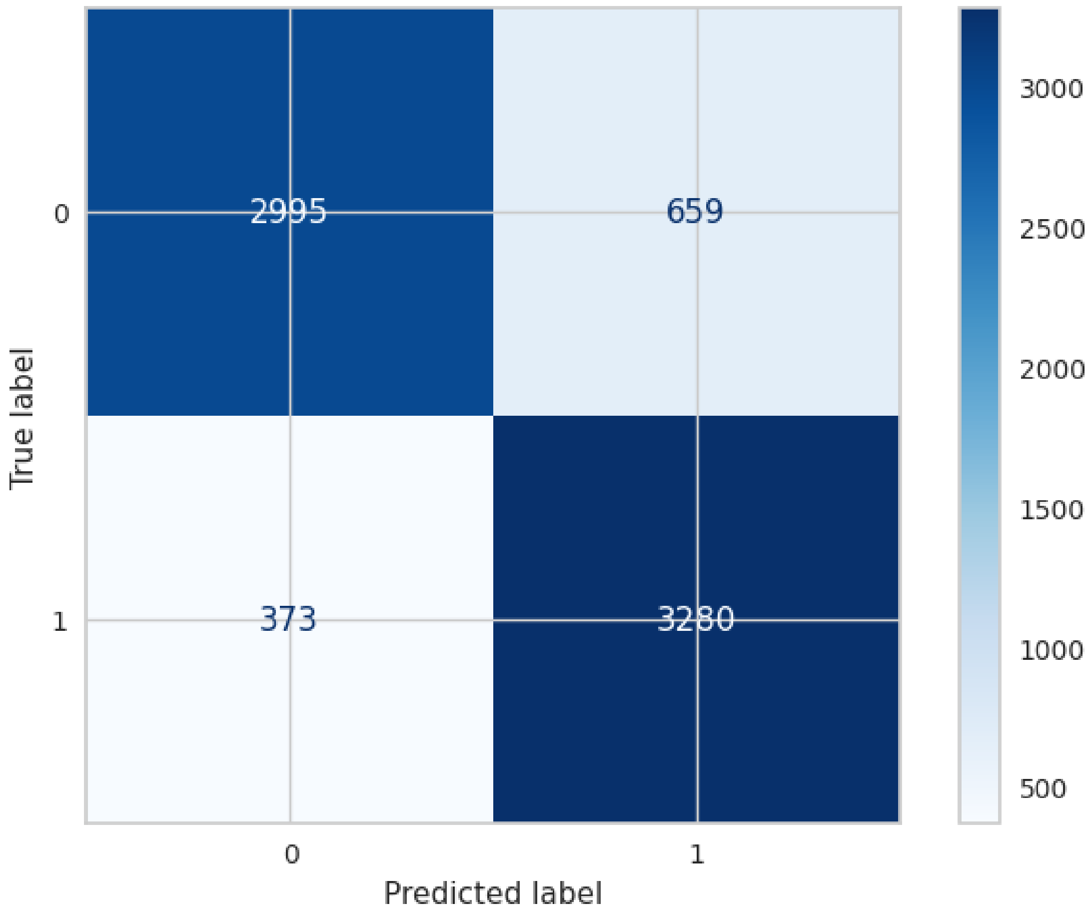
0
1
True label
2984
670
437
3216
0
1
Predicted label
3000
2500
2000
1500
1000
500



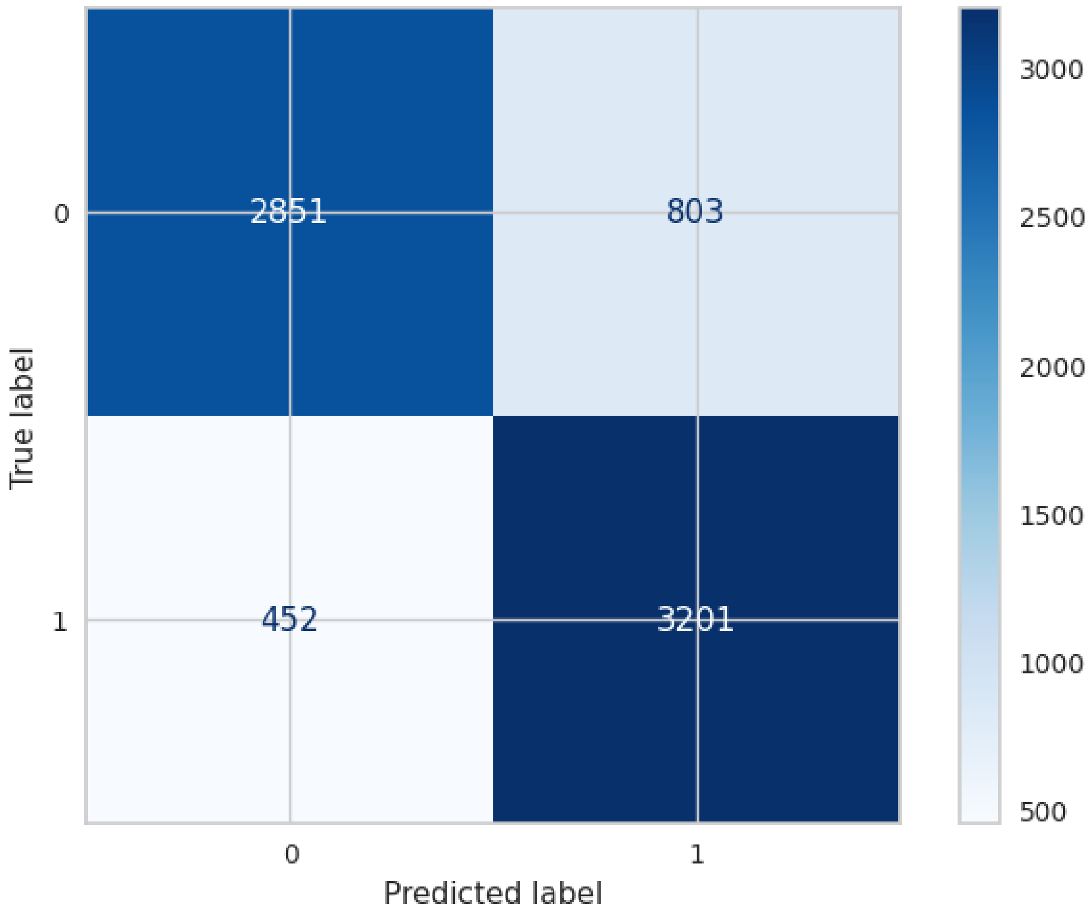
2951
703
297
3356
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


2910
744
495
3158
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500


3061
593
609
3044
0
1
True label
Predicted label
3000
2500
2000
1500
1000



0
1
True label
2995
659
373
3280
0
1
Predicted label
3000
2500
2000
1500
1000
500



[338]:
```python
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['SGD Over','SGD Over With Feature','SGD Over Scaling','SGD Over␣
↪With Normalize','SGD Over With PCA'
                ,'SGD Over With PCA and Scaling',
                'SGD Over With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[338]:
```
                                 Train Accuracy  Test Accuracy   Test F1  \
Models
SGD Over                               0.548393       0.550842  0.193215
SGD Over With Feature                  0.850189       0.848501  0.853164
SGD Over Scaling                       0.863327       0.863145  0.870332
SGD Over With Normalize                0.838922       0.830437  0.836003
SGD Over With PCA                      0.833204       0.835500  0.835117
SGD Over With PCA and Scaling          0.861685       0.858766  0.864067
SGD Over With PCA and Normalize        0.836215       0.828247  0.836098
```

```
                                Test Recall  Test Precision       AUC
Models
SGD Over                           0.107583        0.946988  0.550781
SGD Over With Feature              0.880372        0.827586  0.848506
SGD Over Scaling                   0.918697        0.826805  0.863153
SGD Over With Normalize            0.864495        0.809329  0.830441
SGD Over With PCA                  0.833288        0.836954  0.835500
SGD Over With PCA and Scaling      0.897892        0.832699  0.858771
SGD Over With PCA and Normalize    0.876266        0.799451  0.828253
```

[339]:
```
models_draw(df)
```

RandomUnderSampler

[340]:
```
X_train,y_train,X_test,y_test=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[341]:
```
cross_validation(SGDClassifier(penalty='l2'),X_train,y_train)
```

```
Train Score Value :  [0.66691617 0.72979042 0.85389222 0.85359281 0.70344311]
Mean 0.7615269461077844
Test Score Value :  [0.67664671 0.71616766 0.84371257 0.86227545 0.7005988 ]
Mean 0.7598802395209582
```

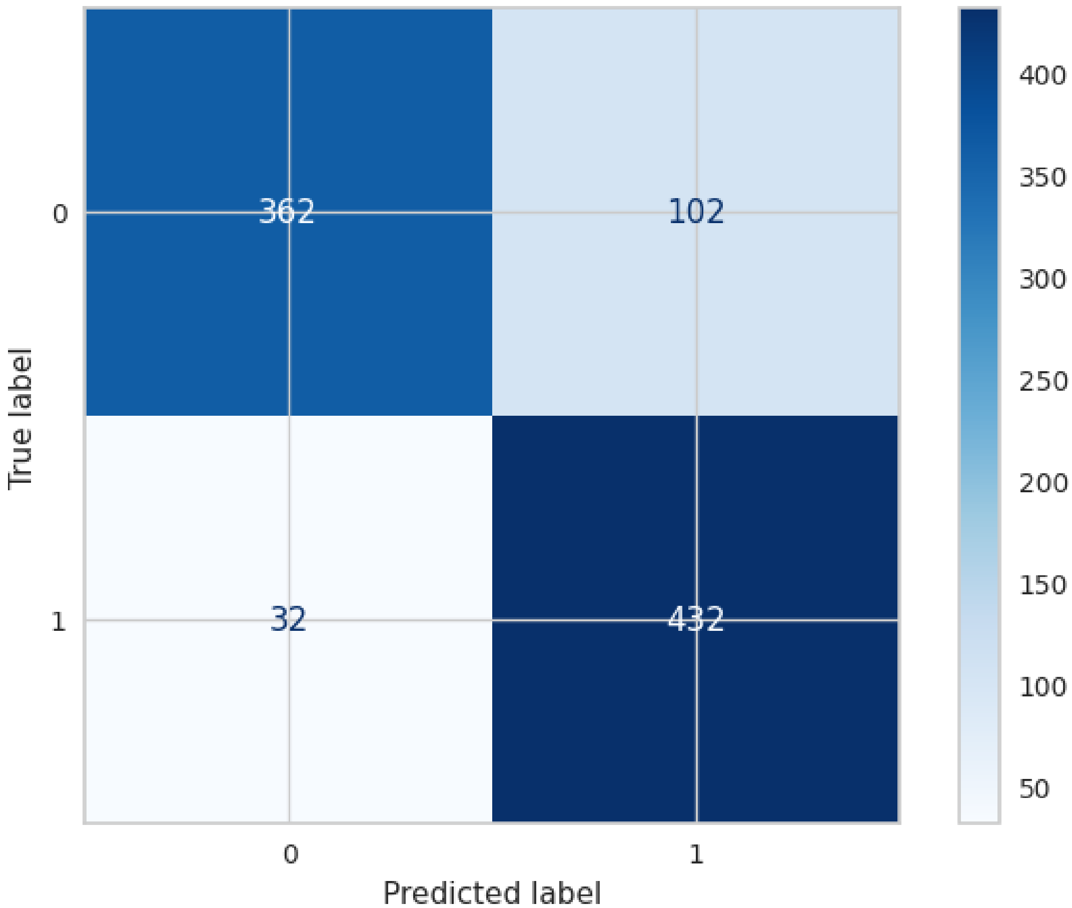
[342]:
```
Values = Models(SGDClassifier(penalty='l2'),X_train,y_train,X_test,y_test)
```

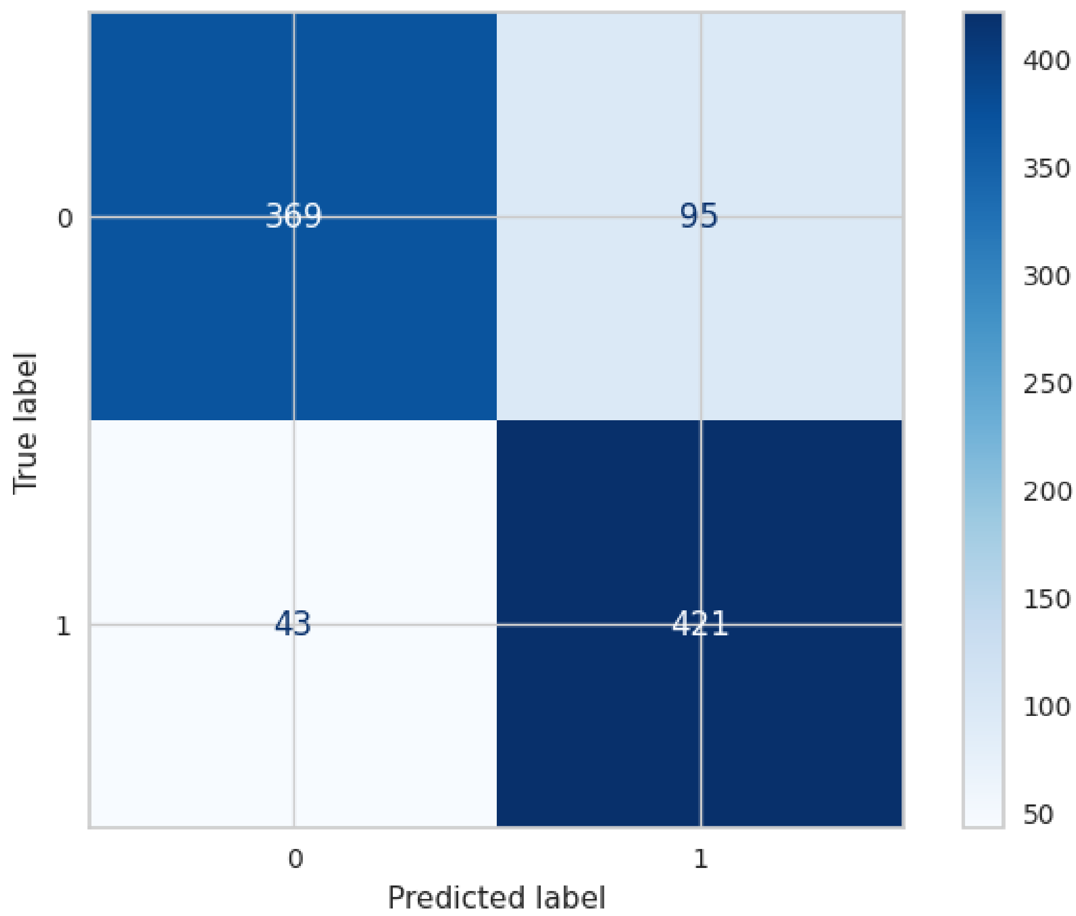
```
Apply Model With Normal Data :

Model Train Score is :  0.8469461077844311
Model Test Score is :  0.8556034482758621
F1 Score is :  0.8657314629258516
Recall Score is :  0.9310344827586207
Precision Score is :  0.8089887640449438
AUC Value  :  0.855603448275862

Classification Report is :                precision    recall  f1-score
support

           0       0.92      0.78      0.84       464
           1       0.81      0.93      0.87       464

    accuracy                           0.86       928
   macro avg       0.86      0.86      0.85       928
weighted avg       0.86      0.86      0.85       928
```

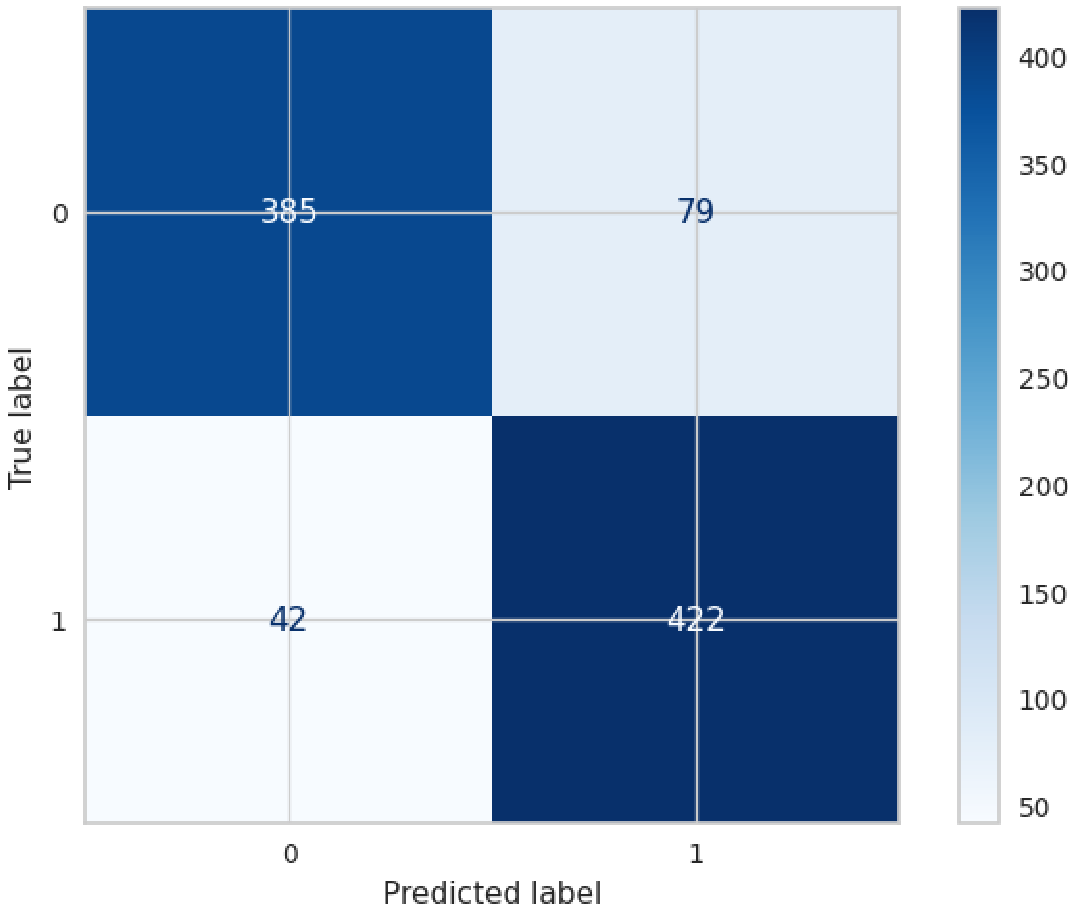
```
Confusion Matrix is :
 [[362 102]
 [ 32 432]]


 Apply Model With Feature Selection :

Model Train Score is :  0.8367664670658682
Model Test Score is :  0.8512931034482759
F1 Score is :  0.8591836734693877
Recall Score is :  0.9073275862068966
Precision Score is :  0.8158914728682171
AUC Value  :  0.8512931034482758
Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.80      0.84       464
           1       0.82      0.91      0.86       464

    accuracy                           0.85       928
   macro avg       0.86      0.85      0.85       928
weighted avg       0.86      0.85      0.85       928

Confusion Matrix is :
 [[369  95]
 [ 43 421]]


 Apply Model With Normal Data With Scaling :

Model Train Score is :  0.8567664670658682
Model Test Score is :  0.8696120689655172
F1 Score is :  0.8746113989637305
Recall Score is :  0.9094827586206896
Precision Score is :  0.8423153692614771
AUC Value  :  0.8696120689655172
Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.83      0.86       464
           1       0.84      0.91      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
```

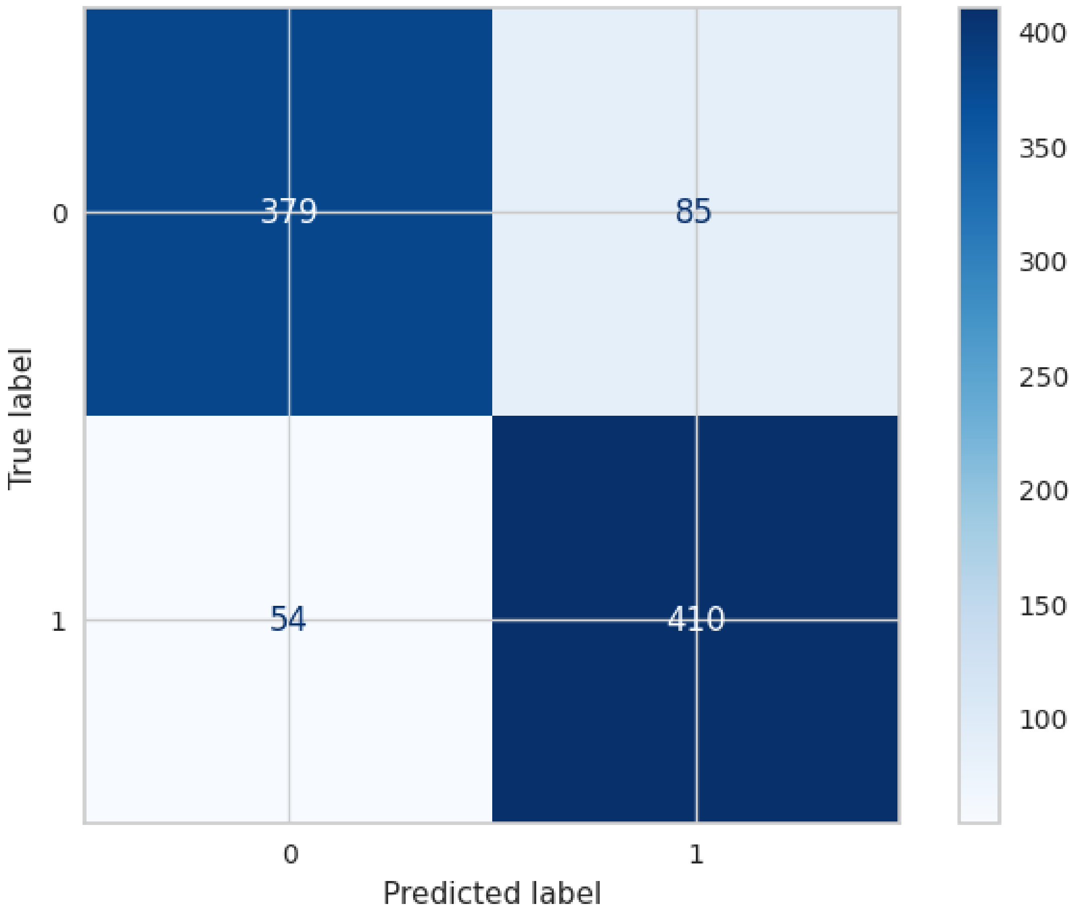
```
 [[385  79]
 [ 42 422]]


 Apply Model With Normal Data With Normalize :

Model Train Score is :  0.8318562874251497
Model Test Score is :  0.8502155172413793
F1 Score is :  0.8550573514077163
Recall Score is :  0.8836206896551724
Precision Score is :  0.8282828282828283
AUC Value  :  0.8502155172413793
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.82      0.85       464
           1       0.83      0.88      0.86       464

    accuracy                           0.85       928
   macro avg       0.85      0.85      0.85       928
weighted avg       0.85      0.85      0.85       928

Confusion Matrix is :
 [[379  85]
 [ 54 410]]


 Apply Model With Normal Data With PCA :

Model Train Score is :  0.8504191616766467
Model Test Score is :  0.8674568965517241
F1 Score is :  0.8712041884816754
Recall Score is :  0.896551724137931
Precision Score is :  0.8472505091649695
AUC Value  :  0.8674568965517242
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.84      0.86       464
           1       0.85      0.90      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[389  75]
```

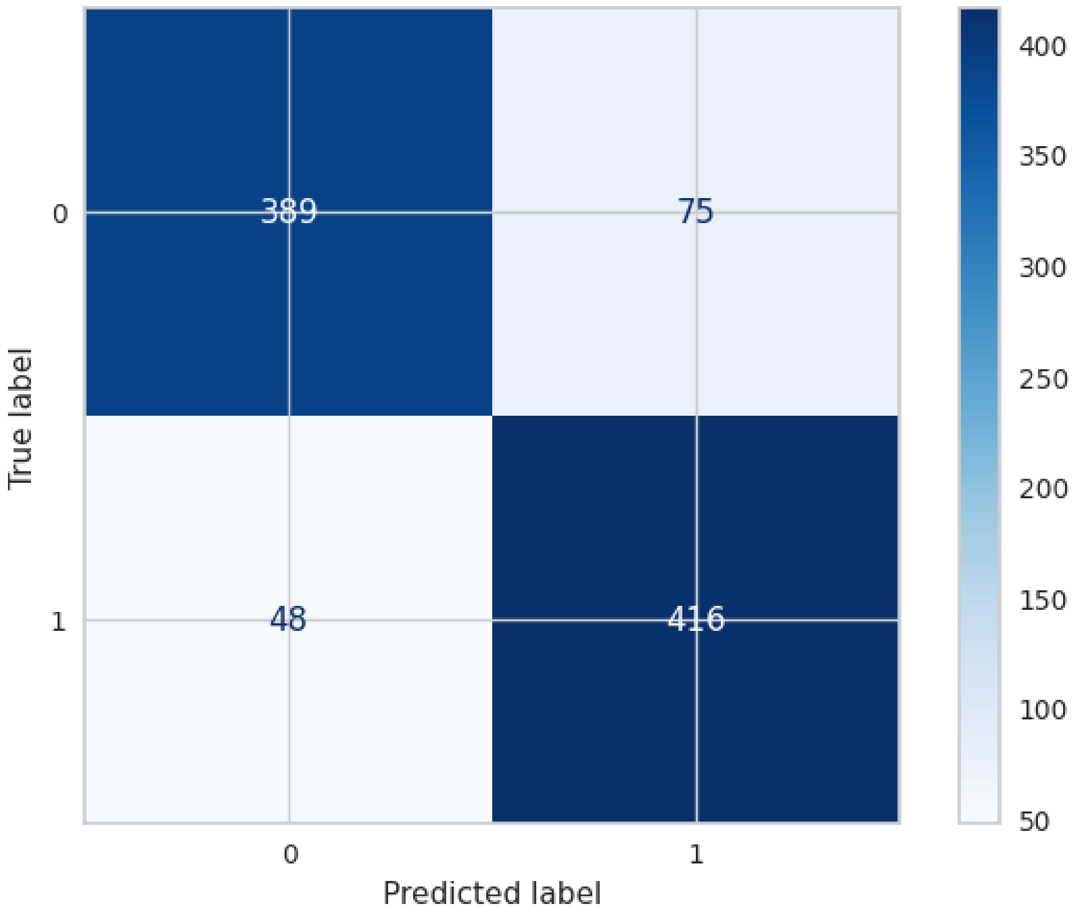
```
 [ 48 416]]


 Apply Model With Normal Data With PCA and Scaling :

Model Train Score is :  0.8570059880239521
Model Test Score is :  0.8685344827586207
F1 Score is :  0.8721174004192872
Recall Score is :  0.896551724137931
Precision Score is :  0.8489795918367347
AUC Value  :  0.8685344827586206
Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.84      0.86       464
           1       0.85      0.90      0.87       464

    accuracy                           0.87       928
   macro avg       0.87      0.87      0.87       928
weighted avg       0.87      0.87      0.87       928

Confusion Matrix is :
 [[390  74]
 [ 48 416]]


 Apply Model With Normal Data With PCA and Normalize :

Model Train Score is :  0.8235928143712575
Model Test Score is :  0.8405172413793104
F1 Score is :  0.8508064516129032
Recall Score is :  0.9094827586206896
Precision Score is :  0.7992424242424242
AUC Value  :  0.8405172413793103
Classification Report is :                precision    recall  f1-score
support

           0       0.90      0.77      0.83       464
           1       0.80      0.91      0.85       464

    accuracy                           0.84       928
   macro avg       0.85      0.84      0.84       928
weighted avg       0.85      0.84      0.84       928

Confusion Matrix is :
 [[358 106]
 [ 42 422]]
```

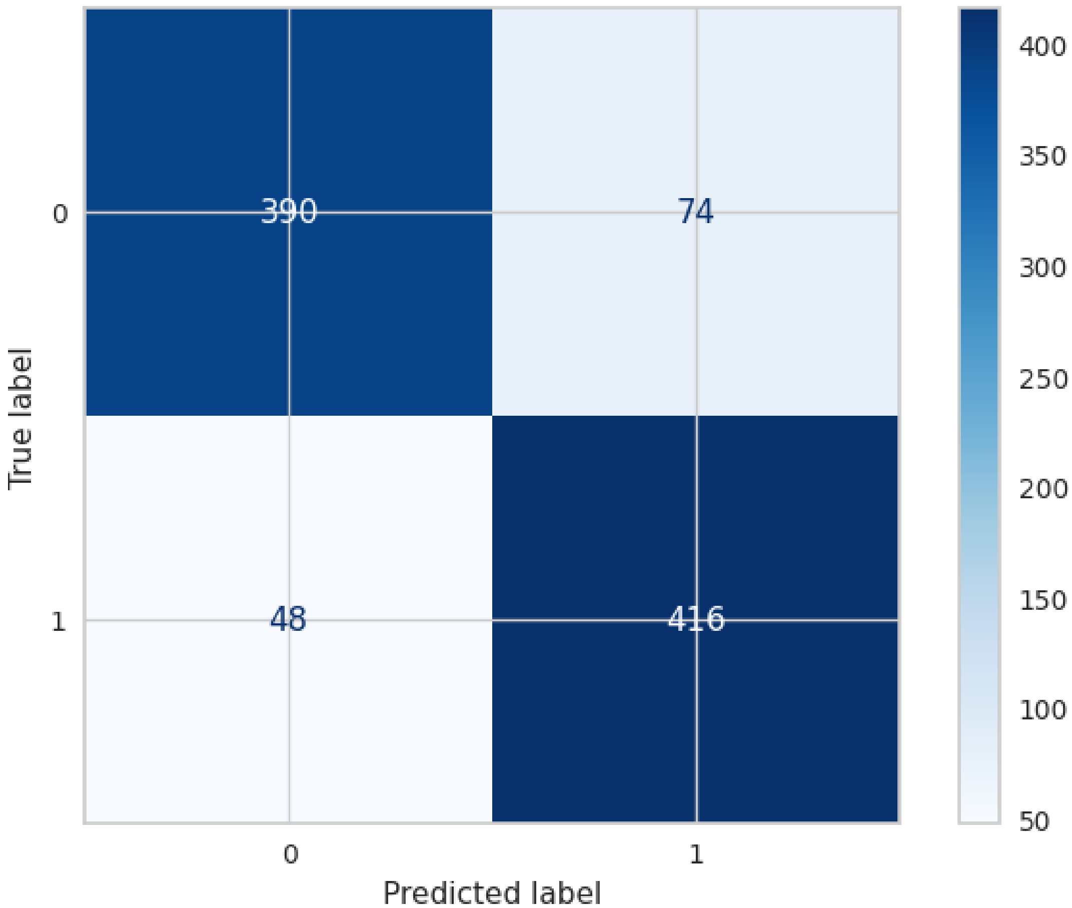

362
102
32
432
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50

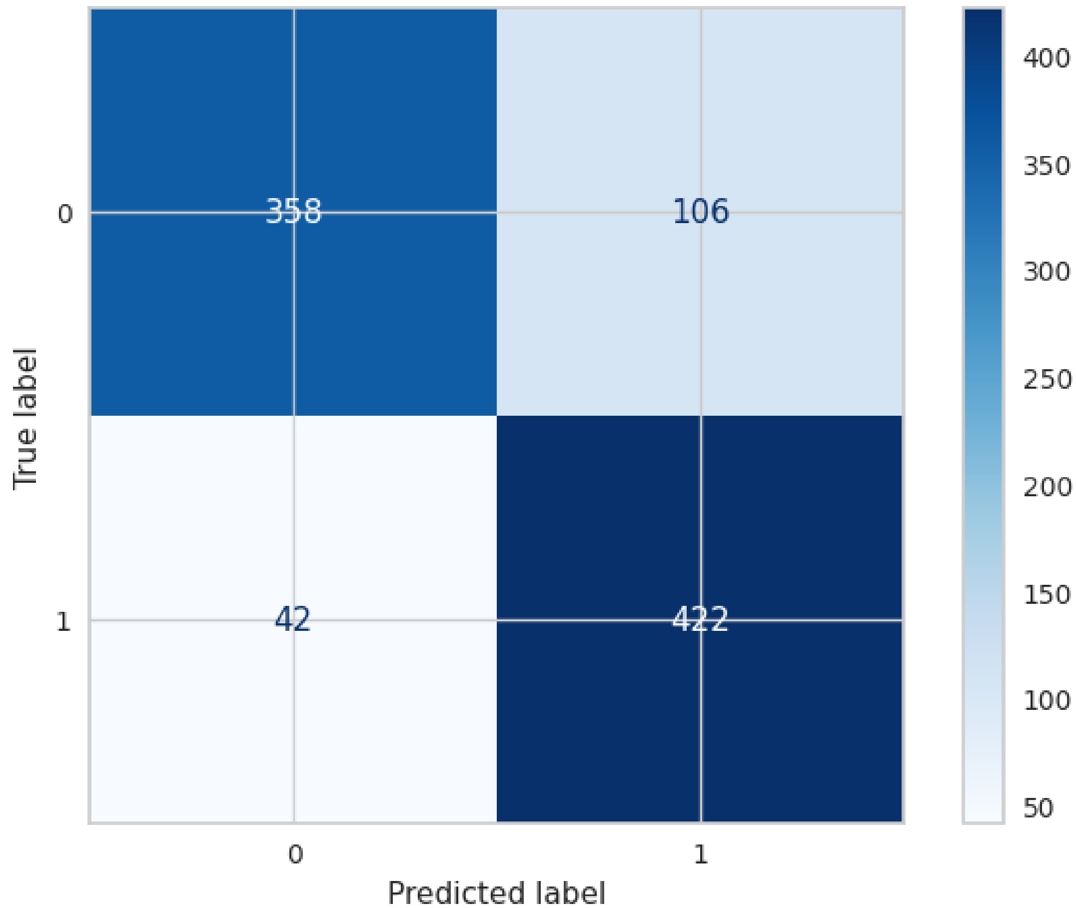

369
95
43
421
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


385
79
42
422
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


379
85
54
410
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100


389
75
48
416
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100
50


390
74
48
416
True label
0
1
0
1
Predicted label
400
350
300
250
200
150
100
50



[343]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','Test␣
↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['SGD Under','SGD Under With Feature','SGD Under Scaling','SGD␣
↪Under With Normalize','SGD Under With PCA'
                ,'SGD Under With PCA and Scaling',
                'SGD Under With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[343]:
```
                                  Train Accuracy  Test Accuracy   Test F1  \
Models
SGD Under                               0.846946       0.855603  0.865731
SGD Under With Feature                  0.836766       0.851293  0.859184
SGD Under Scaling                       0.856766       0.869612  0.874611
SGD Under With Normalize                0.831856       0.850216  0.855057
SGD Under With PCA                      0.850419       0.867457  0.871204
SGD Under With PCA and Scaling          0.857006       0.868534  0.872117
SGD Under With PCA and Normalize        0.823593       0.840517  0.850806
```

```
                                  Test Recall  Test Precision       AUC
Models
SGD Under                            0.931034        0.808989  0.855603
SGD Under With Feature               0.907328        0.815891  0.851293
SGD Under Scaling                    0.909483        0.842315  0.869612
SGD Under With Normalize             0.883621        0.828283  0.850216
SGD Under With PCA                   0.896552        0.847251  0.867457
SGD Under With PCA and Scaling       0.896552        0.848980  0.868534
SGD Under With PCA and Normalize     0.909483        0.799242  0.840517
```

[344]:
```
models_draw(df)
```

Regression

LinearRegression

[345]:
```
def Check_R(model,X_train,y_train,X_test,y_test):
    y_pred = model.predict(X_test)
    print('R2 Score Train :',r2_score(y_train,model.predict(X_train)))
    print('R2 Score Test :',r2_score(y_test,y_pred))
    MAEValue = mean_absolute_error(y_test, y_pred)
    print('Mean Absolute Error Value is : ', MAEValue)
    MSEValue = mean_squared_error(y_test, y_pred)
    print('Mean Squared Error Value is : ', MSEValue)
    MdSEValue = median_absolute_error(y_test, y_pred)
    print('Median Absolute Error Value is : ', MdSEValue )
    return [r2_score(y_train,model.
 ↪predict(X_train)),r2_score(y_test,y_pred),MAEValue,MSEValue,MdSEValue]
def PipeLine2(model):
    steps = [
    ('poly',PolynomialFeatures(degree=3)),
    ('scaling', MinMaxScaler()),
    ('model', model)
    ]
    return Pipeline(steps).fit(X_train,y_train)
def Models(models, X_train, y_train, X_test, y_test):
    print('Apply Model With Normal Data : \n')
    model = PipeLine(models, X_train, y_train)
    value1 = Check_R(model, X_train, y_train, X_test, y_test)
    print("\n\n Apply Model With Feature Selection :\n")
    try:
        feature = SelectFeature(model, X_train, y_train)
    except:
        feature = SelectFeature(RandomForestRegressor(max_depth=20), X_train,␣
 ↪y_train)
    X_train1 = X_train.loc[:, feature]
    X_test1 = X_test.loc[:, feature]
```

```python
    model = PipeLine(models, X_train1, y_train, flage=1)
    value2 = Check_R(model, X_train1, y_train, X_test1, y_test)
    print("\n\n Apply Model With Normal Data With Scaling :\n")
    model = PipeLine(models, X_train, y_train, flage=1)
    value3 = Check_R(model, X_train, y_train, X_test, y_test)
    print("\n\n Apply Model With Normal Data With Normalize :\n")
    model = PipeLine(models, X_train, y_train, flage=2)
    value4 = Check_R(model, X_train, y_train, X_test, y_test)
    print("\n\n Apply Model With Normal Data With PCA :\n")
    model = PipeLine(models, X_train, y_train, flage=3)
    value5 = Check_R(model, X_train, y_train, X_test, y_test)
    print("\n\n Apply Model With Normal Data With PCA and Scaling :\n")
    model = PipeLine(models, X_train, y_train, flage=4)
    value6 = Check_R(model, X_train, y_train, X_test, y_test)
    print("\n\n Apply Model With Normal Data With PCA and Normalize :\n")
    model = PipeLine(models, X_train, y_train, flage=5)
    value7 = Check_R(model, X_train, y_train, X_test, y_test)
    return [value1, value2, value3, value4, value5, value6, value7]
```

[346]:

```python
X_train,y_train,X_test,y_test=Split(X_regression,y_regression,classification=0)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[347]:

```python
cross_validation(LinearRegression(),X_train,y_train)
```

```
Train Score Value :  [0.17496898 0.17965153 0.17770749 0.18636847 0.18140432]
Mean 0.18002015861264872
Test Score Value :  [0.19872269 0.18039782 0.18837726 0.15336561 0.17303233]
Mean 0.17877914309362816
```

[348]:

```python
Values = Models(LinearRegression(),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : 0.1799131533956465
R2 Score Test : 0.17228606829267468
Mean Absolute Error Value is :  0.19619725204659433
Mean Squared Error Value is :  0.06186401164417418
Median Absolute Error Value is :  0.1680957394151842


 Apply Model With Feature Selection :

R2 Score Train : 0.17615493090143175
R2 Score Test : 0.1686980538859445
Mean Absolute Error Value is :  0.19667992701103928
```

```
Mean Squared Error Value is :  0.06213218275563484
Median Absolute Error Value is :  0.16843801909472556


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.1799131533956465
R2 Score Test : 0.17228606829267468
Mean Absolute Error Value is :  0.19619725204659433
Mean Squared Error Value is :  0.06186401164417418
Median Absolute Error Value is :  0.16809573941518405


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.07600027203446813
R2 Score Test : 0.07535690410276075
Mean Absolute Error Value is :  0.20962336185918254
Mean Squared Error Value is :  0.06910857611554425
Median Absolute Error Value is :  0.1808791241205233


 Apply Model With Normal Data With PCA :

R2 Score Train : 0.1799131533956465
R2 Score Test : 0.17228606829267468
Mean Absolute Error Value is :  0.1961972520465943
Mean Squared Error Value is :  0.06186401164417419
Median Absolute Error Value is :  0.16809573941518413


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.1799131533956465
R2 Score Test : 0.17228606829267468
Mean Absolute Error Value is :  0.19619725204659433
Mean Squared Error Value is :  0.06186401164417419
Median Absolute Error Value is :  0.1680957394151841


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.07600027203446813
R2 Score Test : 0.07535690410276075
Mean Absolute Error Value is :  0.20962336185918254
Mean Squared Error Value is :  0.06910857611554425
Median Absolute Error Value is :  0.1808791241205233
```

[349]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['Linear','Linear With Feature','Linear Scaling','Linear With␣
↪Normalize','Linear With PCA'
                ,'Linear With PCA and Scaling',
                'Linear With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[349]:
```
                               Train Accuracy  Test Accuracy       MAE  \
Models
Linear                               0.179913       0.172286  0.196197
Linear With Feature                  0.176155       0.168698  0.196680
Linear Scaling                       0.179913       0.172286  0.196197
Linear With Normalize                0.076000       0.075357  0.209623
Linear With PCA                      0.179913       0.172286  0.196197
Linear With PCA and Scaling          0.179913       0.172286  0.196197
Linear With PCA and Normalize        0.076000       0.075357  0.209623

                                    MSE      MdSE
Models
Linear                         0.061864  0.168096
Linear With Feature            0.062132  0.168438
Linear Scaling                 0.061864  0.168096
Linear With Normalize          0.069109  0.180879
Linear With PCA                0.061864  0.168096
Linear With PCA and Scaling    0.061864  0.168096
Linear With PCA and Normalize  0.069109  0.180879
```

[350]:
```
models_draw(df)
```

RandomForestRegressor

[351]:
```
Search(RandomForestRegressor(max_depth=20),{'max_depth':
↪[20,25,30,35,40]},X_train,y_train)
```

[351]:
```
RandomForestRegressor(max_depth=20)
```

[352]:
```
cross_validation(RandomForestRegressor(max_depth=20),X_train,y_train)
```

```
Train Score Value :  [0.74415434 0.74063758 0.73495586 0.74516589 0.73725465]
Mean 0.7404336671675078
Test Score Value :  [0.22012049 0.19951875 0.20272295 0.17715486 0.20306069]
Mean 0.20051554728514
```

[353]:
```
Values =␣
↪Models(RandomForestRegressor(max_depth=20),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :
```

```
R2 Score Train : 0.7235870419837179
R2 Score Test : 0.1642813655855161
Mean Absolute Error Value is :  0.19425288366657895
Mean Squared Error Value is :  0.062462289627078685
Median Absolute Error Value is :  0.15761151669122453


 Apply Model With Feature Selection :

R2 Score Train : 0.6908577382279188
R2 Score Test : 0.1251847846804185
Mean Absolute Error Value is :  0.19749169026426083
Mean Squared Error Value is :  0.06538439984379493
Median Absolute Error Value is :  0.16073871557098646


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.7251155544706664
R2 Score Test : 0.1652335111899077
Mean Absolute Error Value is :  0.19424345269438678
Mean Squared Error Value is :  0.062391125491136776
Median Absolute Error Value is :  0.15759616493404496


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.6414595481216068
R2 Score Test : 0.18957811135899805
Mean Absolute Error Value is :  0.19026381180052132
Mean Squared Error Value is :  0.060571590298311356
Median Absolute Error Value is :  0.15502489890250598


 Apply Model With Normal Data With PCA :

R2 Score Train : 0.615030835334744
R2 Score Test : 0.20088648644403706
Mean Absolute Error Value is :  0.1889425015063232
Mean Squared Error Value is :  0.059726393158165955
Median Absolute Error Value is :  0.15830763236069512


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.5370176102561741
R2 Score Test : 0.2104153913267489
```

```
Mean Absolute Error Value is :  0.18803508408618938
Mean Squared Error Value is :  0.059014195066484254
Median Absolute Error Value is :  0.15516793812864793


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.5940394243979192
R2 Score Test : 0.1963536741149311
Mean Absolute Error Value is :  0.19098710931058116
Mean Squared Error Value is :  0.060065179233845814
Median Absolute Error Value is :  0.15863160908372492
```

[354]:
```python
df = pd.DataFrame(Values,columns=['Train Accuracy','Test Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['Random','Random With Feature','Random Scaling','Random With Normalize','Random With PCA'
                ,'Random With PCA and Scaling',
                'Random With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[354]:
```
                               Train Accuracy  Test Accuracy       MAE  \
Models
Random                               0.723587       0.164281  0.194253
Random With Feature                  0.690858       0.125185  0.197492
Random Scaling                       0.725116       0.165234  0.194243
Random With Normalize                0.641460       0.189578  0.190264
Random With PCA                      0.615031       0.200886  0.188943
Random With PCA and Scaling          0.537018       0.210415  0.188035
Random With PCA and Normalize        0.594039       0.196354  0.190987

                                    MSE      MdSE
Models
Random                         0.062462  0.157612
Random With Feature            0.065384  0.160739
Random Scaling                 0.062391  0.157596
Random With Normalize          0.060572  0.155025
Random With PCA                0.059726  0.158308
Random With PCA and Scaling    0.059014  0.155168
Random With PCA and Normalize  0.060065  0.158632
```

[355]:
```python
models_draw(df)
```

Ridge

[356]:
```python
Search(Ridge(alpha=1.0),{'alpha':[1,2,.5,5,10,15,40]},X_train,y_train)
```

[356]: Ridge(alpha=0.5)

[357]:
```
cross_validation(Ridge(alpha=.5),X_train,y_train)
```

```
Train Score Value :  [0.17496744 0.17965065 0.17770661 0.1863676  0.18140344]
Mean 0.18001914741418806
Test Score Value :  [0.19872663 0.18036918 0.18837539 0.15336924 0.17303511]
Mean 0.17877511144388875
```

[358]:
```
Values = Models(Ridge(alpha=.5),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : 0.17991244273646634
R2 Score Test : 0.1722874119661726
Mean Absolute Error Value is :  0.19619805603088958
Mean Squared Error Value is :  0.06186391121692543
Median Absolute Error Value is :  0.16809458644419698


 Apply Model With Feature Selection :

R2 Score Train : 0.17615492417052525
R2 Score Test : 0.16869974303902646
Mean Absolute Error Value is :  0.19668147742933953
Mean Squared Error Value is :  0.06213205650696132
Median Absolute Error Value is :  0.1684306773400222


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.179912390913616
R2 Score Test : 0.17229516682164658
Mean Absolute Error Value is :  0.19619697248792486
Mean Squared Error Value is :  0.061863331612727696
Median Absolute Error Value is :  0.16809538243749234


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.040785827125842666
R2 Score Test : 0.04250004455645562
Mean Absolute Error Value is :  0.21354464332853368
Mean Squared Error Value is :  0.07156432448910473
Median Absolute Error Value is :  0.18612895660766393


 Apply Model With Normal Data With PCA :
```

```
R2 Score Train : 0.17991244273646645
R2 Score Test : 0.1722874119661726
Mean Absolute Error Value is :  0.1961980560308896
Mean Squared Error Value is :  0.06186391121692542
Median Absolute Error Value is :  0.16809458644419709


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.179912390913616
R2 Score Test : 0.17229516682164658
Mean Absolute Error Value is :  0.19619697248792495
Mean Squared Error Value is :  0.06186333161272769
Median Absolute Error Value is :  0.16809538243749184


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.040785827125842666
R2 Score Test : 0.04250004455645562
Mean Absolute Error Value is :  0.21354464332853368
Mean Squared Error Value is :  0.07156432448910473
Median Absolute Error Value is :  0.18612895660766393
```

[359]:

```python
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['Ridge','Ridge With Feature','Ridge Scaling','Ridge With␣
↪Normalize','Ridge With PCA'
                ,'Ridge With PCA and Scaling',
                'Ridge With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[359]:

```
                              Train Accuracy  Test Accuracy       MAE  \
Models
Ridge                               0.179912       0.172287  0.196198
Ridge With Feature                  0.176155       0.168700  0.196681
Ridge Scaling                       0.179912       0.172295  0.196197
Ridge With Normalize                0.040786       0.042500  0.213545
Ridge With PCA                      0.179912       0.172287  0.196198
Ridge With PCA and Scaling          0.179912       0.172295  0.196197
Ridge With PCA and Normalize        0.040786       0.042500  0.213545

                                   MSE      MdSE
Models
Ridge                         0.061864  0.168095
```

```
Ridge With Feature            0.062132  0.168431
Ridge Scaling                 0.061863  0.168095
Ridge With Normalize          0.071564  0.186129
Ridge With PCA                0.061864  0.168095
Ridge With PCA and Scaling    0.061863  0.168095
Ridge With PCA and Normalize  0.071564  0.186129
```

[360]:
```
models_draw(df)
```

DecisionTreeRegressor

[361]:
```
Search(DecisionTreeRegressor(max_depth=20),{'max_depth':
↪[20,25,30,35,40]},X_train,y_train)
```

[361]:
```
DecisionTreeRegressor(max_depth=20)
```

[362]:
```
cross_validation(DecisionTreeRegressor(max_depth=20),X_train,y_train)
```

```
Train Score Value :  [0.70625918 0.70679501 0.71844423 0.75649423 0.66250076]
Mean 0.710098681226547
Test Score Value :  [-0.22357831 -0.27612165 -0.25777573 -0.3281922
-0.22963324]        Mean -0.26306022787286915
```

[363]:
```
Values =␣
↪Models(DecisionTreeRegressor(max_depth=20),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : 0.7034968420136125
R2 Score Test : -0.3611559368949344
Mean Absolute Error Value is :  0.23582177743781374
Mean Squared Error Value is :  0.10173390045025843
Median Absolute Error Value is :  0.17059327916377443


 Apply Model With Feature Selection :

R2 Score Train : 0.6402537094958773
R2 Score Test : -0.22615805917692788
Mean Absolute Error Value is :  0.22553155242435627
Mean Squared Error Value is :  0.09164404940491128
Median Absolute Error Value is :  0.16842513576415827


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.7034968420136125
R2 Score Test : -0.3411230733774848
Mean Absolute Error Value is :  0.2346813973424926
```

```
Mean Squared Error Value is :  0.10023662795738965
Median Absolute Error Value is :  0.17082886943243802


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.6399606705654064
R2 Score Test : -0.1361105929324644
Mean Absolute Error Value is :  0.2144272686022767
Mean Squared Error Value is :  0.08491382862828958
Median Absolute Error Value is :  0.15730123161150206


 Apply Model With Normal Data With PCA :

R2 Score Train : 0.5724284499023398
R2 Score Test : -0.1353803265279654
Mean Absolute Error Value is :  0.21296813397041536
Mean Squared Error Value is :  0.08485924792398987
Median Absolute Error Value is :  0.15353761120790443


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.5239140964635978
R2 Score Test : -0.05872568567852521
Mean Absolute Error Value is :  0.2078646545143188
Mean Squared Error Value is :  0.07913001779697232
Median Absolute Error Value is :  0.15472217745504815


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.563817374089383
R2 Score Test : -0.1627571939851402
Mean Absolute Error Value is :  0.21799342072327252
Mean Squared Error Value is :  0.08690541723717057
Median Absolute Error Value is :  0.16261882432657992
```

[364]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['Decision','Decision With Feature','Decision Scaling','Decision␣
↪With Normalize','Decision With PCA'
                ,'Decision With PCA and Scaling',
                'Decision With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[364]:
```
                              Train Accuracy  Test Accuracy       MAE  \
Models
Decision                            0.703497      -0.361156  0.235822
Decision With Feature               0.640254      -0.226158  0.225532
Decision Scaling                    0.703497      -0.341123  0.234681
Decision With Normalize             0.639961      -0.136111  0.214427
Decision With PCA                   0.572428      -0.135380  0.212968
Decision With PCA and Scaling       0.523914      -0.058726  0.207865
Decision With PCA and Normalize     0.563817      -0.162757  0.217993

                                     MSE      MdSE
Models
Decision                        0.101734  0.170593
Decision With Feature           0.091644  0.168425
Decision Scaling                0.100237  0.170829
Decision With Normalize         0.084914  0.157301
Decision With PCA               0.084859  0.153538
Decision With PCA and Scaling   0.079130  0.154722
Decision With PCA and Normalize 0.086905  0.162619
```

[365]:
```
models_draw(df)
```

KNeighborsRegressor

[366]:
```
Search(KNeighborsRegressor(n_neighbors = 5),{'n_neighbors':
 ↪[3,5,7,9,11]},X_train,y_train)
```

[366]:
```
KNeighborsRegressor(n_neighbors=11)
```

[367]:
```
cross_validation(KNeighborsRegressor(n_neighbors = 11),X_train,y_train)
```

```
Train Score Value :  [0.15036278 0.14235686 0.14552232 0.15015352 0.14530772]
Mean 0.14674064002252052
Test Score Value :  [-0.03321914 -0.01232461 -0.02066934 -0.02904956
-0.03249353]        Mean -0.02555123549454876
```

[368]:
```
Values = Models(KNeighborsRegressor(n_neighbors =␣
 ↪11),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : 0.151576976882271
R2 Score Test : -0.04295235954272725
Mean Absolute Error Value is :  0.2196140746901452
Mean Squared Error Value is :  0.07795110658821741
Median Absolute Error Value is :  0.17857394738698074


 Apply Model With Feature Selection :
```

```
R2 Score Train : 0.29509380492304615
R2 Score Test : 0.11695556130499152
Mean Absolute Error Value is :  0.19917175668440434
Mean Squared Error Value is :  0.0659994586838338
Median Absolute Error Value is :  0.1629875167501234


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.3105140936233737
R2 Score Test : 0.15682229020792626
Mean Absolute Error Value is :  0.19565752333114378
Mean Squared Error Value is :  0.06301978698013418
Median Absolute Error Value is :  0.16207066788913185


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.14980694817592155
R2 Score Test : -0.027872771276059582
Mean Absolute Error Value is :  0.2177136179578107
Mean Squared Error Value is :  0.07682404591135503
Median Absolute Error Value is :  0.1797376401720855


 Apply Model With Normal Data With PCA :

R2 Score Train : 0.15112528819412496
R2 Score Test : -0.040344334411982485
Mean Absolute Error Value is :  0.219728205538011
Mean Squared Error Value is :  0.07775618067133222
Median Absolute Error Value is :  0.17899710839974609


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.31076802852280105
R2 Score Test : 0.15610918435195043
Mean Absolute Error Value is :  0.1957374012234676
Mean Squared Error Value is :  0.063073085091097
Median Absolute Error Value is :  0.16217645814232312


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.1496660314028736
R2 Score Test : -0.02811330180523397
```

```
Mean Absolute Error Value is :  0.21777100878482125
Mean Squared Error Value is :  0.07684202335849902
Median Absolute Error Value is :  0.1797376401720855
```

[369]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['KNN','KNN With Feature','KNN Scaling','KNN With␣
↪Normalize','KNN With PCA'
                ,'KNN With PCA and Scaling',
                'KNN With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[369]:
```
                            Train Accuracy  Test Accuracy       MAE       MSE  \
Models
KNN                               0.151577      -0.042952  0.219614  0.077951
KNN With Feature                  0.295094       0.116956  0.199172  0.065999
KNN Scaling                       0.310514       0.156822  0.195658  0.063020
KNN With Normalize                0.149807      -0.027873  0.217714  0.076824
KNN With PCA                      0.151125      -0.040344  0.219728  0.077756
KNN With PCA and Scaling          0.310768       0.156109  0.195737  0.063073
KNN With PCA and Normalize        0.149666      -0.028113  0.217771  0.076842

                                MdSE
Models
KNN                         0.178574
KNN With Feature            0.162988
KNN Scaling                 0.162071
KNN With Normalize          0.179738
KNN With PCA                0.178997
KNN With PCA and Scaling    0.162176
KNN With PCA and Normalize  0.179738
```

[370]:
```
models_draw(df)
```

SVR

[371]:
```
Search(SVR(C = 1.0),{'C':[1,.5,2,3,5,10]},X_train,y_train)
```

[371]:
```
SVR(C=10)
```

[372]:
```
cross_validation(SVR(C = 10),X_train,y_train)
```

```
Train Score Value :  [0.13397628 0.13917285 0.13878077 0.14855674 0.14015961]
Mean 0.1401292522035146
Test Score Value :  [0.14612083 0.13909309 0.14292087 0.12060939 0.14106212]
Mean 0.13796125964321812
```

[373]:
```
Values = Models(SVR(C = 10),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : 0.15259411734317463
R2 Score Test : 0.14929750423378996
Mean Absolute Error Value is :  0.1871221322467224
Mean Squared Error Value is :  0.06358219559655516
Median Absolute Error Value is :  0.14017116730696383


 Apply Model With Feature Selection :

R2 Score Train : 0.20114119993723467
R2 Score Test : 0.1657206194846481
Mean Absolute Error Value is :  0.18188834951726046
Mean Squared Error Value is :  0.06235471862148843
Median Absolute Error Value is :  0.13168278289261764


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.2930056884734811
R2 Score Test : 0.17098762253982425
Mean Absolute Error Value is :  0.18155693735828865
Mean Squared Error Value is :  0.06196105853452672
Median Absolute Error Value is :  0.13470236083783463


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.09120751932124083
R2 Score Test : 0.09169581781503067
Mean Absolute Error Value is :  0.1930102426595141
Mean Squared Error Value is :  0.0678873924318722
Median Absolute Error Value is :  0.14308272327121532


 Apply Model With Normal Data With PCA :

R2 Score Train : 0.20576148560345964
R2 Score Test : 0.1902712962766141
Mean Absolute Error Value is :  0.17869072138647818
Mean Squared Error Value is :  0.060519781094464195
Median Absolute Error Value is :  0.13049148173223732


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.3530363870420393
```

```
R2 Score Test : 0.12912110064986704
Mean Absolute Error Value is :  0.1871817490537347
Mean Squared Error Value is :  0.06509019639059606
Median Absolute Error Value is :  0.13833055606689298


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.22520287942031858
R2 Score Test : 0.13948806014575543
Mean Absolute Error Value is :  0.18437929052979685
Mean Squared Error Value is :  0.064315361416337
Median Absolute Error Value is :  0.1341850275879029
```

[374]:

```python
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['SVR','SVR With Feature','SVR Scaling','SVR With␣
↪Normalize','SVR With PCA'
                ,'SVR With PCA and Scaling',
                'SVR With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[374]:

```
                            Train Accuracy  Test Accuracy       MAE       MSE  \
Models
SVR                               0.152594       0.149298  0.187122  0.063582
SVR With Feature                  0.201141       0.165721  0.181888  0.062355
SVR Scaling                       0.293006       0.170988  0.181557  0.061961
SVR With Normalize                0.091208       0.091696  0.193010  0.067887
SVR With PCA                      0.205761       0.190271  0.178691  0.060520
SVR With PCA and Scaling          0.353036       0.129121  0.187182  0.065090
SVR With PCA and Normalize        0.225203       0.139488  0.184379  0.064315

                                MdSE
Models
SVR                         0.140171
SVR With Feature            0.131683
SVR Scaling                 0.134702
SVR With Normalize          0.143083
SVR With PCA                0.130491
SVR With PCA and Scaling    0.138331
SVR With PCA and Normalize  0.134185
```

[375]:

```python
models_draw(df)
```

SGDRegressor

[376]:

```python
Search(SGDRegressor(alpha=0.1),{'alpha':[.1,1,.5,2,3,5,10]},X_train,y_train)
```

[376]: SGDRegressor(alpha=0.1)

[377]:
```
cross_validation(SGDRegressor(alpha=.5),X_train,y_train)
```

```
Train Score Value :  [-1.74515879e+26 -8.26981994e+24 -1.24679307e+27
-1.61345357e+27
 -3.17005226e+26]        Mean -6.720075134924045e+26
Test Score Value :  [-1.71938922e+26 -7.98146956e+24 -1.26248809e+27
-1.61593695e+27
 -3.29142639e+26]        Mean -6.774976134429883e+26
```

[378]:
```
Values = Models(SGDRegressor(alpha=.5),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : -5.741588667337575e+25
R2 Score Test : -5.770198700782382e+25
Mean Absolute Error Value is :  1979307709209.2761
Mean Squared Error Value is :  4.312693382822287e+24
Median Absolute Error Value is :  2126675631138.131


 Apply Model With Feature Selection :

R2 Score Train : 0.04980513066561476
R2 Score Test : 0.04825384154312595
Mean Absolute Error Value is :  0.21658273224938457
Mean Squared Error Value is :  0.07113428102825906
Median Absolute Error Value is :  0.19188840696525453


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.057883642885369846
R2 Score Test : 0.05783319804683314
Mean Absolute Error Value is :  0.21167362757902528
Mean Squared Error Value is :  0.07041830304115947
Median Absolute Error Value is :  0.18252468742246236


 Apply Model With Normal Data With Normalize :

R2 Score Train : -7.883667650210313e-05
R2 Score Test : -0.0005043315116277647
Mean Absolute Error Value is :  0.21985131158315532
Mean Squared Error Value is :  0.07477850649077612
Median Absolute Error Value is :  0.1908918716052896
```

```
 Apply Model With Normal Data With PCA :

R2 Score Train : -9.247806710323317e+21
R2 Score Test : -1.0264011042776051e+22
Mean Absolute Error Value is :  18627255437.397522
Mean Squared Error Value is :  7.671405232439082e+20
Median Absolute Error Value is :  13557219352.917046


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.05461088587298801
R2 Score Test : 0.05427611300658375
Mean Absolute Error Value is :  0.2136782254694331
Mean Squared Error Value is :  0.07068417156692469
Median Absolute Error Value is :  0.18718922160770612


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.00012653230309889185
R2 Score Test : -8.298003730722314e-05
Mean Absolute Error Value is :  0.2192071300287922
Mean Squared Error Value is :  0.0747470143393031
Median Absolute Error Value is :  0.18967438604261685
```

[379]:

```python
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['SGD','SGD With Feature','SGD Scaling','SGD With␣
↪Normalize','SGD With PCA'
                ,'SGD With PCA and Scaling',
                'SGD With PCA and Normalize']
df.set_index('Models', inplace=True)
df
```

[379]:

```
                            Train Accuracy  Test Accuracy           MAE  \
Models
SGD                          -5.741589e+25  -5.770199e+25  1.979308e+12
SGD With Feature              4.980513e-02   4.825384e-02  2.165827e-01
SGD Scaling                   5.788364e-02   5.783332e-02  2.116736e-01
SGD With Normalize           -7.883668e-05  -5.043315e-04  2.198513e-01
SGD With PCA                 -9.247807e+21  -1.026401e+22  1.862726e+10
SGD With PCA and Scaling      5.461089e-02   5.427611e-02  2.136782e-01
SGD With PCA and Normalize    1.265323e-04  -8.298004e-05  2.192071e-01

                                     MSE          MdSE
```

```
Models
SGD                         4.312693e+24  2.126676e+12
SGD With Feature            7.113428e-02  1.918884e-01
SGD Scaling                 7.041830e-02  1.825247e-01
SGD With Normalize          7.477851e-02  1.908919e-01
SGD With PCA                7.671405e+20  1.355722e+10
SGD With PCA and Scaling    7.068417e-02  1.871892e-01
SGD With PCA and Normalize  7.474701e-02  1.896744e-01
```

[380]:
```
models_draw(df)
```

GradientBoostingRegressor

[381]:
```
Search(GradientBoostingRegressor(max_depth=2),{'max_depth':
 ↪[5,10,15,20,25,30,35,40]},X_train,y_train)
```

[381]:
```
GradientBoostingRegressor(max_depth=5)
```

[382]:
```
cross_validation(GradientBoostingRegressor(max_depth=5),X_train,y_train)
```

```
Train Score Value :  [0.28806374 0.29342476 0.29203183 0.30102895 0.29190881]
Mean 0.2932916189080985
Test Score Value :  [0.26953323 0.25149182 0.24963486 0.22338702 0.24944391]
Mean 0.24869816762398522
```

[383]:
```
Values =␣
 ↪Models(GradientBoostingRegressor(max_depth=5),X_train,y_train,X_test,y_test)
```

```
Apply Model With Normal Data :

R2 Score Train : 0.287898719320049527
R2 Score Test : 0.236127005533337673
Mean Absolute Error Value is :  0.18466873538194234
Mean Squared Error Value is :  0.05709248813400777
Median Absolute Error Value is :  0.15125449551905054


 Apply Model With Feature Selection :

R2 Score Train : 0.2722153116530236
R2 Score Test : 0.23074245690816586
Mean Absolute Error Value is :  0.1851873945543714
Mean Squared Error Value is :  0.05749493367236129
Median Absolute Error Value is :  0.15221935871073447


 Apply Model With Normal Data With Scaling :

R2 Score Train : 0.287898719320049527
```

```
R2 Score Test : 0.23609034774701987
Mean Absolute Error Value is :  0.18467448999600752
Mean Squared Error Value is :  0.057095227966738805
Median Absolute Error Value is :  0.1512756991703233


 Apply Model With Normal Data With Normalize :

R2 Score Train : 0.30498481102426067
R2 Score Test : 0.23737862939907983
Mean Absolute Error Value is :  0.18460457959509896
Mean Squared Error Value is :  0.0569989407495361
Median Absolute Error Value is :  0.1517326619250577


 Apply Model With Normal Data With PCA :

R2 Score Train : 0.3050864461207925
R2 Score Test : 0.23178883402922656
Mean Absolute Error Value is :  0.1856607032134361
Mean Squared Error Value is :  0.05741672659631513
Median Absolute Error Value is :  0.1537172317296088


 Apply Model With Normal Data With PCA and Scaling :

R2 Score Train : 0.29819168715101185
R2 Score Test : 0.22338482198821352
Mean Absolute Error Value is :  0.18686371631834767
Mean Squared Error Value is :  0.05804484928320842
Median Absolute Error Value is :  0.15415055056320198


 Apply Model With Normal Data With PCA and Normalize :

R2 Score Train : 0.3039890198899693
R2 Score Test : 0.2191657562205488
Mean Absolute Error Value is :  0.18802122444901345
Mean Squared Error Value is :  0.058360185686016045
Median Absolute Error Value is :  0.15723956602054193
```

[384]:
```
df = pd.DataFrame(Values,columns=['Train Accuracy','Test␣
↪Accuracy','MAE','MSE','MdSE'])
df['Models'] = ['Gradient','Gradient With Feature','Gradient Scaling','Gradient␣
↪With Normalize','Gradient With PCA'
                ,'Gradient With PCA and Scaling',
                'Gradient With PCA and Normalize']
```

```
df.set_index('Models', inplace=True)
df
```

[384]:

```
                                 Train Accuracy  Test Accuracy       MAE  \
Models
Gradient                               0.287899       0.236127  0.184669
Gradient With Feature                  0.272215       0.230742  0.185187
Gradient Scaling                       0.287899       0.236090  0.184674
Gradient With Normalize                0.304985       0.237379  0.184605
Gradient With PCA                      0.305086       0.231789  0.185661
Gradient With PCA and Scaling          0.298192       0.223385  0.186864
Gradient With PCA and Normalize        0.303989       0.219166  0.188021

                                      MSE      MdSE
Models
Gradient                         0.057092  0.151254
Gradient With Feature            0.057495  0.152219
Gradient Scaling                 0.057095  0.151276
Gradient With Normalize          0.056999  0.151733
Gradient With PCA                0.057417  0.153717
Gradient With PCA and Scaling    0.058045  0.154151
Gradient With PCA and Normalize  0.058360  0.157240
```

[385]:

```
models_draw(df)
```

Clustering

Feature Scaling

[386]:

```
Columns = X_cluster.columns
```

[387]:

```
MS = MinMaxScaler()
X_cluster = MS.fit_transform(X_cluster)
```

[388]:

```
X_cluster = pd.DataFrame(X_cluster,columns=Columns)
X_cluster.head()
```

[388]:

```
        age  job  marital  education  default  housing  loan  contact  \
0  0.735849  0.3      0.5   0.000000      0.0      0.0   0.0      1.0
1  0.754717  0.7      0.5   0.500000      0.0      0.0   0.0      1.0
2  0.377358  0.7      0.5   0.500000      0.0      1.0   0.0      1.0
3  0.433962  0.0      0.5   0.166667      0.0      0.0   0.0      1.0
4  0.735849  0.7      0.5   0.500000      0.0      0.0   1.0      1.0

      month  day_of_week  …  campaign  pdays  previous  poutcome  \
0  0.666667         0.25  …       0.0    1.0       0.0       0.5
1  0.666667         0.25  …       0.0    1.0       0.0       0.5
2  0.666667         0.25  …       0.0    1.0       0.0       0.5
```

```
3  0.666667          0.25  …       0.0     1.0       0.0       0.5
4  0.666667          0.25  …       0.0     1.0       0.0       0.5

   emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.employed    y
0      0.888889            0.72           0.64   0.911111          0.8  0.0
1      0.888889            0.72           0.64   0.911111          0.8  0.0
2      0.888889            0.72           0.64   0.911111          0.8  0.0
3      0.888889            0.72           0.64   0.911111          0.8  0.0
4      0.888889            0.72           0.64   0.911111          0.8  0.0

[5 rows x 21 columns]
```

[389]:
```
X_train,X_test=Split(X_cluster,classification=2)
```

```
X_train shape is  (37056, 21)
X_test shape is  (4118, 21)
```

[390]:
```
X_train.y = X_train.y.astype(int)
X_test.y = X_test.y.astype(int)
y_train=X_train.iloc[:,-1]
y_test=X_test.iloc[:,-1]
```

PCA

[391]:
```
PCAModel = PCA(n_components=2, svd_solver='auto')
PCAModel.fit(X_train.iloc[:,:-1])
print('PCAModel Explained Variance is : ' , PCAModel.explained_variance_)
print('PCAModel Explained Variance ratio is : ' , PCAModel.
 ↪explained_variance_ratio_)
```

```
PCAModel Explained Variance is :  [0.3236848  0.24030428]
PCAModel Explained Variance ratio is :  [0.18894046 0.1402698 ]
```

[392]:
```
X_train_pca = PCAModel.transform(X_train.iloc[:,:-1])
X_test_pca = PCAModel.transform(X_test.iloc[:,:-1])
X_train_pca = pd.DataFrame(X_train_pca,columns=['Feature1','Feature2'])
X_test_pca = pd.DataFrame(X_test_pca,columns=['Feature1','Feature2'])
X_train_pca.head()
```

[392]:

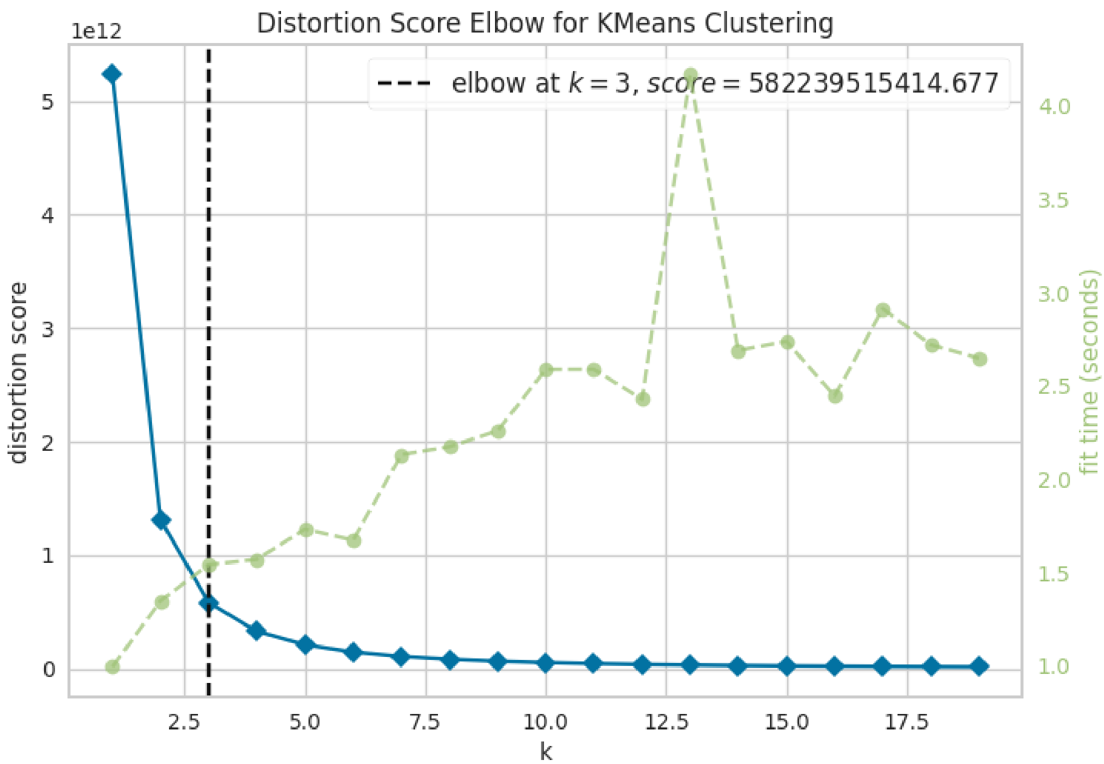
	Feature1	Feature2
0	0.956740	-0.200840
1	0.231611	-0.408457
2	-0.806489	0.276320
3	-0.434794	-0.580563
4	0.416368	0.718046

[393]:
```
X_train.reset_index(inplace=True)
X_test.reset_index(inplace=True)
```

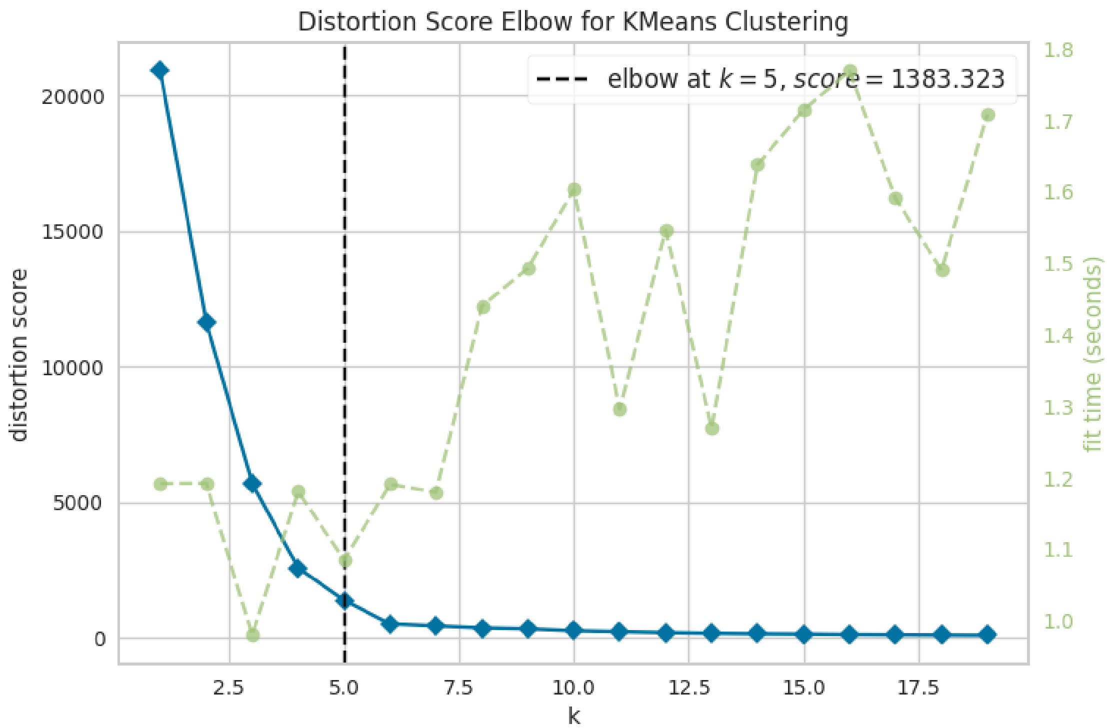
[394]:
```
fig = go.Figure()
for color,y_ in zip(['red','orange'],X_train.y.unique()):
    pca = X_train_pca[X_train.y==y_]
    scatter_trace = go.Scatter(
        x=pca['Feature1'],
        y=pca['Feature2'],
        mode='markers',
        marker=dict(
            size=20,
            color=color,
            symbol='circle',
            opacity=0.8
        ),
        name=f'Train Cluster {y_}'
    )
    fig.add_trace(scatter_trace)
for color,y_ in zip(['green','blue'],X_test.y.unique()):
    pca = X_test_pca[X_test.y==y_]
    scatter_trace = go.Scatter(
        x=pca['Feature1'],
        y=pca['Feature2'],
        mode='markers',
        marker=dict(
            size=10,
            color=color,
            symbol='circle',
            opacity=0.8
        ),
        name=f'Test Cluster {y_}'
    )
    fig.add_trace(scatter_trace)
fig.update_layout(
    title_text='PCA',
    title_x=0.5,
    title_font=dict(size=20),
    xaxis_title='Feature1',
    yaxis_title='Feature2',
    font=dict(size=15),
    width=1000,
    height=700,
    xaxis=dict(tickangle=-90),
    template='plotly_dark'
)
fig.update_annotations(font=dict(size=20))
fig.show()
```

```
Elbow
```

[395]:
```
kmeans = KMeans(init='k-means++', random_state=44)
visualizer = KElbowVisualizer(kmeans, k=(1,20))
visualizer.fit(X_train.iloc[:,:-1])
visualizer.show()
plt.show()
```



[396]:
```
kmeans = KMeans(init='k-means++', random_state=44)
visualizer = KElbowVisualizer(kmeans, k=(1,20))
visualizer.fit(X_train_pca)
visualizer.show()
plt.show()
```



K-Means model with two clusters PCA

[397]:
```
kmeans1 = KMeans(n_clusters=2,random_state=44)
kmeans1.fit(X_train_pca)
```

[397]:
```
KMeans(n_clusters=2, random_state=44)
```

[398]:
```
kmeans1.cluster_centers_
```

[398]:
```
array([[-0.21870611,  0.50662691],
       [ 0.17940626, -0.41558986]])
```

[399]:
```
kmeans1.inertia_
```

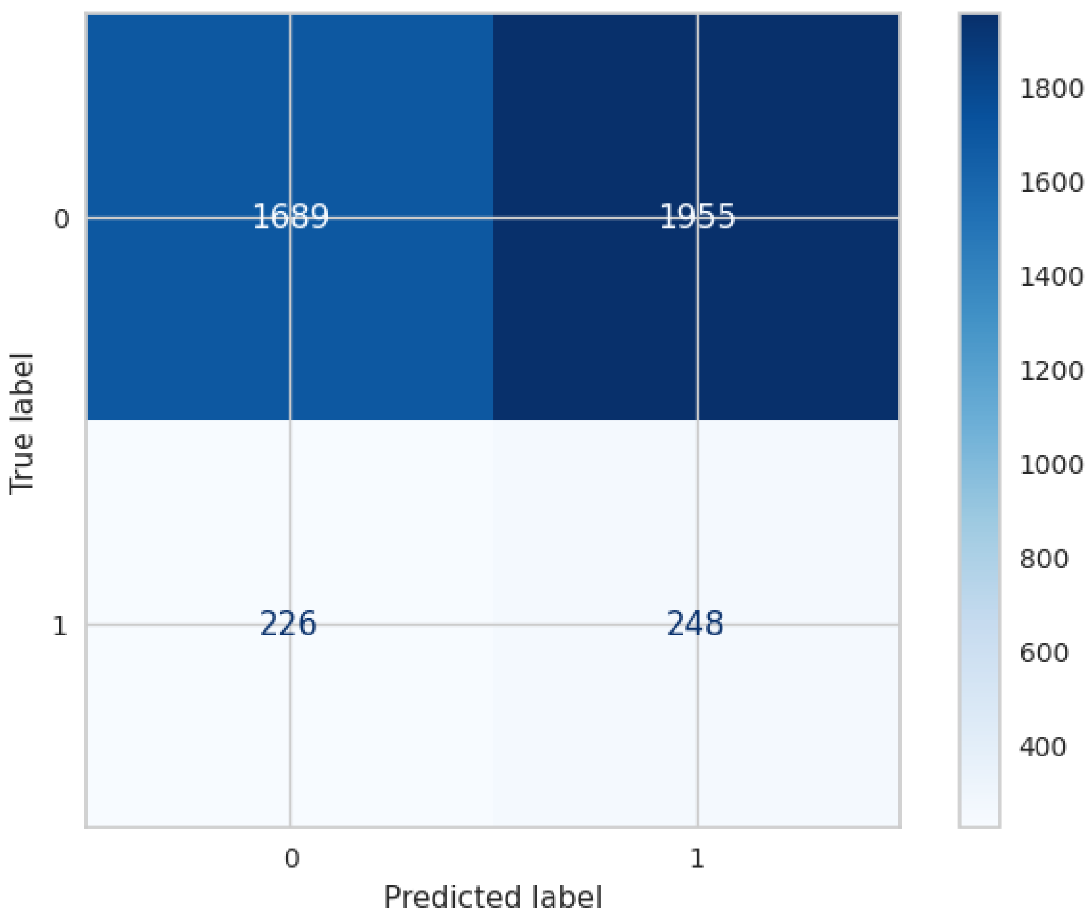
[399]:
```
11642.536148789948
```

Evaluation

[400]:
```
y_train_pred = kmeans1.predict(X_train_pca)
y_pred = kmeans1.predict(X_test_pca)
```

[401]:
```
value_kmeans_pca =
 ↪Check(cluster=1,y_train2=y_train,y_train_pred=y_train_pred,y_test2=y_test,y_pred=y_pred)
```

```
Model Train Score is :  0.46594343696027635
Model Test Score is :  0.47037396794560465
F1 Score is :  0.18528203212551367
Recall Score is :  0.5232067510548524
Precision Score is :  0.11257376305038584
AUC Value  :  0.4933541987985568
Classification Report is :                precision    recall  f1-score
support

           0       0.88      0.46      0.61      3644
           1       0.11      0.52      0.19       474

    accuracy                           0.47      4118
   macro avg       0.50      0.49      0.40      4118
weighted avg       0.79      0.47      0.56      4118

Confusion Matrix is :
 [[1689 1955]
 [ 226  248]]
```



[402]:
```python
kmeans = KMeans(n_clusters=5,random_state=44)
kmeans.fit(X_train_pca)
```

[402]:
```
KMeans(n_clusters=5, random_state=44)
```

[403]:
```python
kmeans.cluster_centers_
```

[403]:
```
array([[ 0.16768794,  0.62574502],
       [ 0.23082459, -0.41251061],
       [-0.83283837,  0.31730133],
       [-0.52876862, -0.63462027],
       [ 0.77951734, -0.21431802]])
```

[404]:
```python
kmeans.inertia_
```

[404]:
```
1383.3226044319376
```

[405]:
```python
fig = go.Figure()
for color,y_ in zip(['red','orange'],[0,1]):
    pca = X_train_pca[y_train_pred==y_]
    scatter_trace = go.Scatter(
        x=pca['Feature1'],
        y=pca['Feature2'],
        mode='markers',
        marker=dict(
            size=20,
            color=color,
            symbol='circle',
            opacity=0.8
        ),
        name=f'Train Cluster {y_}'
    )
    fig.add_trace(scatter_trace)
for color,y_ in zip(['green','blue'],[0,1]):
    pca = X_test_pca[y_pred==y_]
    scatter_trace = go.Scatter(
        x=pca['Feature1'],
        y=pca['Feature2'],
        mode='markers',
        marker=dict(
            size=10,
            color=color,
            symbol='circle',
            opacity=0.8
        ),
        name=f'Test Cluster {y_}'
```

```
    )
    fig.add_trace(scatter_trace)
fig.update_layout(
    title_text='PCA',
    title_x=0.5,
    title_font=dict(size=20),
    xaxis_title='Feature1',
    yaxis_title='Feature2',
    font=dict(size=15),
    width=1000,
    height=700,
    xaxis=dict(tickangle=-90),
    template='plotly_dark'
)
fig.update_annotations(font=dict(size=20))
fig.show()
```

K-Means model with two clusters

[406]:
```
kmeans2 = KMeans(n_clusters=2,random_state=44)
kmeans2.fit(X_train.iloc[:,:-1])
```

[406]:
```
KMeans(n_clusters=2, random_state=44)
```

[407]:
```
kmeans2.cluster_centers_
```

[407]:
```
array([[ 3.08556103e+04,  4.32962071e-01,  3.77643750e-01,
         6.04478823e-01,  6.60554263e-01,  1.62276194e-04,
         5.92037648e-01,  1.54757397e-01,  9.24433386e-02,
         4.76484977e-01,  4.96740953e-01,  3.63398632e-01,
         2.23984421e-01,  9.44268530e-01,  4.94092374e-02,
         4.28841889e-01,  5.06974871e-01,  4.03940066e-01,
         3.80094120e-01,  6.77451937e-01,  6.45394061e-01],
       [ 1.02762860e+04,  4.33140241e-01,  3.48317088e-01,
         5.64543056e-01,  5.74335362e-01, -2.71050543e-19,
         5.06866283e-01,  1.50573537e-01,  6.39237439e-01,
         4.64023073e-01,  5.06098874e-01,  3.66534504e-01,
         2.86122031e-01,  1.00000000e+00, -1.54043445e-15,
         5.00000000e-01,  9.57922703e-01,  7.32142819e-01,
         4.45999246e-01,  9.46602077e-01,  9.24260865e-01]])
```

[408]:
```
kmeans2.inertia_
```

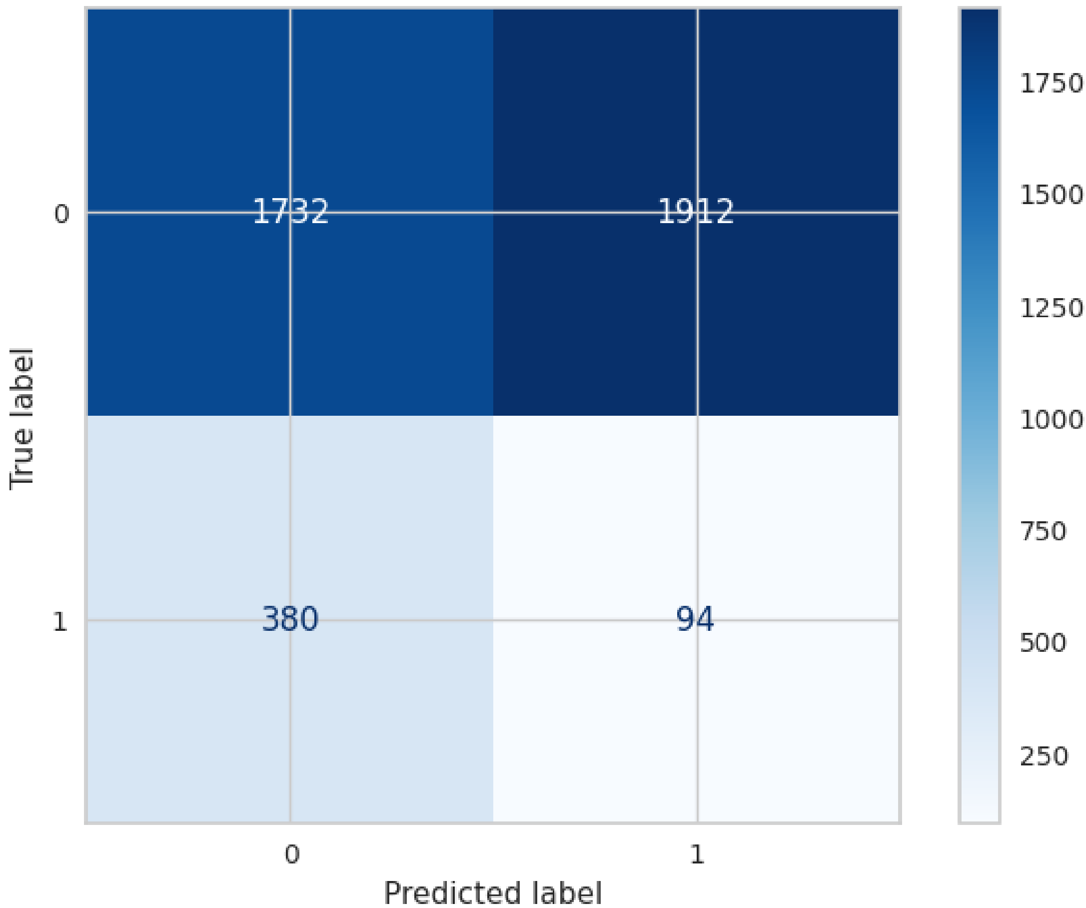
[408]:
```
1309740502783.25
```

Evaluation

[409]:
```
y_train_pred = kmeans2.predict(X_train.iloc[:,:-1])
y_pred = kmeans2.predict(X_test.iloc[:,:-1])
```

[410]:
```
value_kmeans =␣
↪Check(cluster=1,y_train2=y_train,y_train_pred=y_train_pred,y_test2=y_test,y_pred=y_pred)
```

```
Model Train Score is :  0.4335330310880829
Model Test Score is :  0.4434191355026712
F1 Score is :  0.07580645161290323
Recall Score is :  0.19831223628691982
Precision Score is :  0.04685942173479561
AUC Value  :  0.3368070511840746

Classification Report is :                precision    recall  f1-score
support

           0       0.82      0.48      0.60      3644
           1       0.05      0.20      0.08       474

    accuracy                           0.44      4118
   macro avg       0.43      0.34      0.34      4118
weighted avg       0.73      0.44      0.54      4118

Confusion Matrix is :
 [[1732 1912]
 [ 380   94]]
```



- The lesser the model inertia, the better the model fit.
- We can see that the model has very high inertia. So, this is not a good model fit to the data.

Use elbow method to find optimal number of clusters

[411]:
```
inertia_values = []
k_range =  np.arange(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=600, n_init=10,␣
 ↪random_state=44)
    kmeans.fit(X_cluster)
    inertia_values.append(kmeans.inertia_)

fig = go.Figure()
fig.add_trace(go.Scatter(x=k_range, y=inertia_values, mode='lines+markers'))
fig.update_layout(
    title='The Elbow Method for Optimal Number of Clusters',
    xaxis=dict(title='Number of Clusters'),
```

```
    yaxis=dict(title='Inertia'),
    template='plotly_dark'
)
fig.show()
```

- By the above plot, we can see that there is a kink at k=3.
- Hence k=3 can be considered a good number of the cluster to cluster this data.

K-Means model with different clusters

K-Means model with 3 clusters

[412]:
```
kmeans = KMeans(n_clusters=3,random_state=44)
kmeans.fit(X_train.iloc[:,:-1])
```

[412]:
```
KMeans(n_clusters=3, random_state=44)
```

[413]:
```
kmeans.cluster_centers_
```

[413]:
```
array([[ 6.83742978e+03,  4.34689478e-01,  3.41394635e-01,
         5.52844215e-01,  5.50083495e-01, -2.57498016e-19,
         4.84890110e-01,  1.43018746e-01,  9.12330317e-01,
         5.57934712e-01,  5.08140756e-01,  3.64231140e-01,
         2.66402715e-01,  1.00000000e+00, -1.14491749e-15,
         5.00000000e-01,  9.36867055e-01,  7.78765352e-01,
         4.92624434e-01,  9.32633566e-01,  8.86360698e-01],
       [ 3.42898556e+04,  4.24729872e-01,  3.69404501e-01,
         6.25274190e-01,  6.32328648e-01, -2.57498016e-19,
         5.90381022e-01,  1.53871151e-01,  9.97644000e-02,
         4.90806185e-01,  4.73007555e-01,  3.83575731e-01,
         2.08432854e-01,  9.18274308e-01,  6.68848578e-02,
         4.16362012e-01,  3.15685903e-01,  3.65088959e-01,
         3.30051182e-01,  5.57602838e-01,  4.92688277e-01],
       [ 2.05766839e+04,  4.39692342e-01,  3.78085846e-01,
         5.75499151e-01,  6.69738367e-01,  2.42502627e-04,
         5.73033708e-01,  1.61102579e-01,  8.56842616e-02,
         3.62047441e-01,  5.22997332e-01,  3.47196787e-01,
         2.90291811e-01,  9.98031998e-01,  7.28662656e-03,
         4.76881416e-01,  9.44116617e-01,  5.60255436e-01,
         4.16234743e-01,  9.45412787e-01,  9.74852478e-01]])
```

[414]:
```
kmeans.inertia_
```

[414]:
```
582242281750.3159
```

K-Means model with 6 clusters

[415]:
```
kmeans = KMeans(n_clusters=6,random_state=44)
kmeans.fit(X_train.iloc[:,:-1])
```

[415]: KMeans(n_clusters=6, random_state=44)

[416]:
```
kmeans.cluster_centers_
```

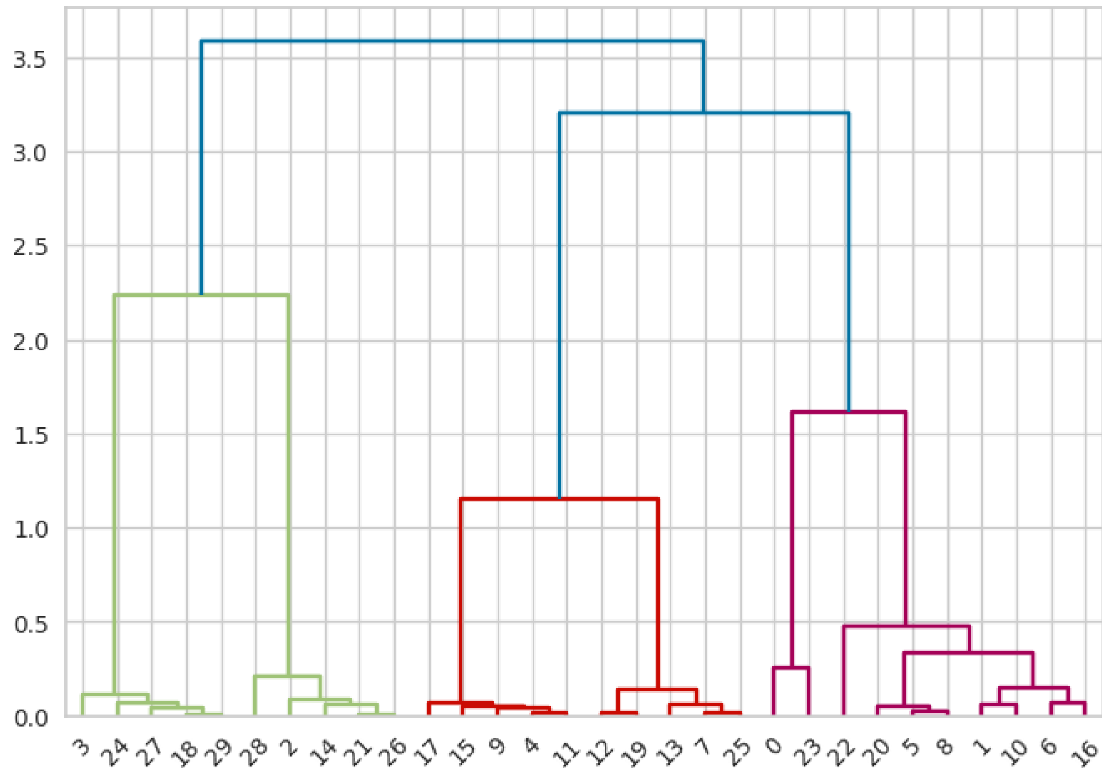
[416]:
```
array([[ 1.71108019e+04,  4.29488175e-01,  3.60936239e-01,
         5.88297014e-01,  6.22195319e-01, -2.57498016e-19,
         5.49636804e-01,  1.65456013e-01,  9.49152542e-02,
         2.77553583e-01,  5.00686037e-01,  3.70920266e-01,
         3.23066990e-01,  1.00000000e+00,  4.16333634e-17,
         5.00000000e-01,  1.00000000e+00,  6.39838579e-01,
         3.50443906e-01,  9.74465839e-01,  1.00000000e+00],
       [ 3.08666946e+04,  4.08326927e-01,  3.46029957e-01,
         6.07183121e-01,  5.76126054e-01, -2.84603070e-19,
         5.94942825e-01,  1.61217587e-01,  8.02061524e-02,
         4.26370323e-01,  4.96255436e-01,  3.80644304e-01,
         2.17361894e-01,  9.81428943e-01,  4.58320871e-02,
         3.72765341e-01,  3.49635833e-01,  3.51283621e-01,
         1.45240780e-01,  6.92780321e-01,  6.10822999e-01],
       [ 3.40819509e+03,  4.41887584e-01,  3.40293690e-01,
         5.47200258e-01,  5.43461890e-01, -2.57498016e-19,
         4.89269001e-01,  1.48297563e-01,  1.00000000e+00,
         6.66666667e-01,  5.45505890e-01,  3.78761064e-01,
         2.37857028e-01,  1.00000000e+00,  4.16333634e-17,
         5.00000000e-01,  8.88888889e-01,  7.20000000e-01,
         6.40000000e-01,  9.11485076e-01,  8.00000000e-01],
       [ 2.39843280e+04,  4.49240651e-01,  3.94250646e-01,
         5.62903747e-01,  7.16489018e-01,  4.84496124e-04,
         5.96091731e-01,  1.56169251e-01,  7.67118863e-02,
         4.45144272e-01,  5.46794251e-01,  3.24522069e-01,
         2.55943152e-01,  9.96068128e-01,  1.45118125e-02,
         4.53972868e-01,  8.89032443e-01,  4.81679587e-01,
         4.81479328e-01,  9.16810734e-01,  9.50064599e-01],
       [ 3.77420802e+04,  4.41553836e-01,  3.93087106e-01,
         6.43814349e-01,  6.89791360e-01, -2.57498016e-19,
         5.85389770e-01,  1.46429155e-01,  1.20281092e-01,
         5.57080859e-01,  4.47949011e-01,  3.86045525e-01,
         2.00163425e-01,  8.54444542e-01,  8.80862886e-02,
         4.60696192e-01,  2.82671460e-01,  3.79329956e-01,
         5.17424416e-01,  4.21337650e-01,  3.74080732e-01],
       [ 1.02571554e+04,  4.27933987e-01,  3.43359375e-01,
         5.57861328e-01,  5.56586372e-01, -2.57498016e-19,
         4.80631510e-01,  1.38183594e-01,  8.27636719e-01,
         4.49544271e-01,  4.70499674e-01,  3.48994572e-01,
         2.96093750e-01,  1.00000000e+00,  6.24500451e-17,
         5.00000000e-01,  9.84899450e-01,  8.38600260e-01,
         3.45416667e-01,  9.53725405e-01,  9.72819010e-01]])
```

[417]:
```
kmeans.inertia_
```

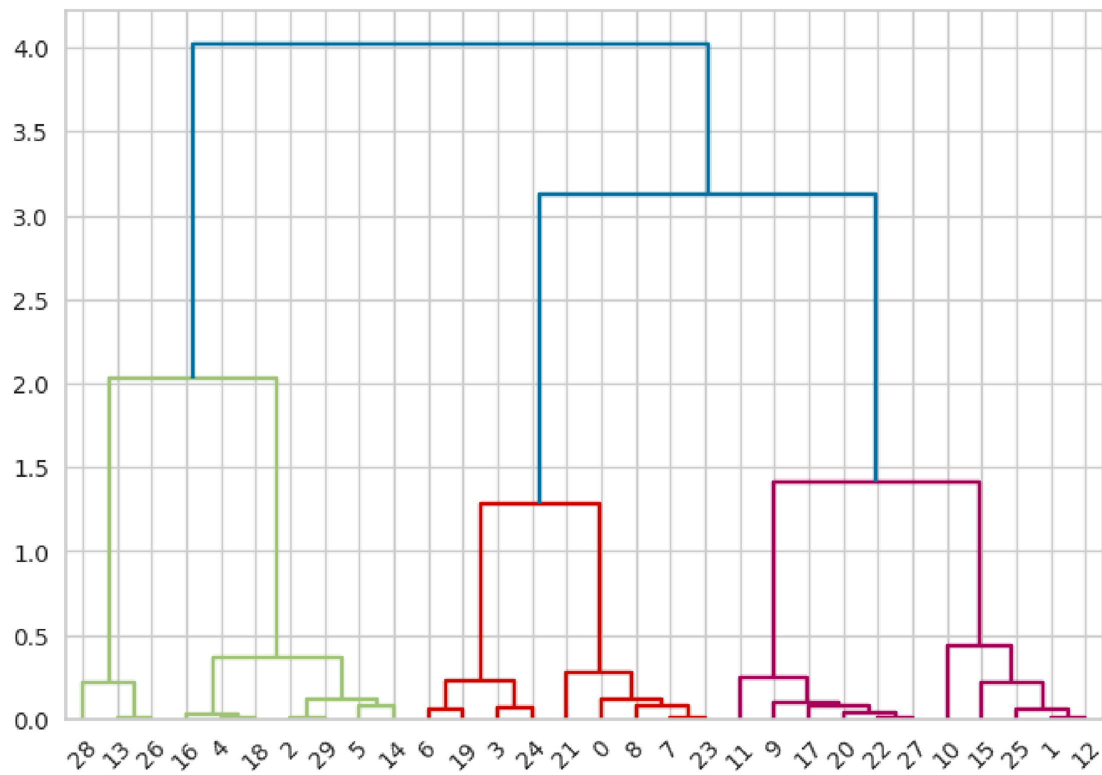
[417]: 145524897092.83978

AgglomerativeClustering

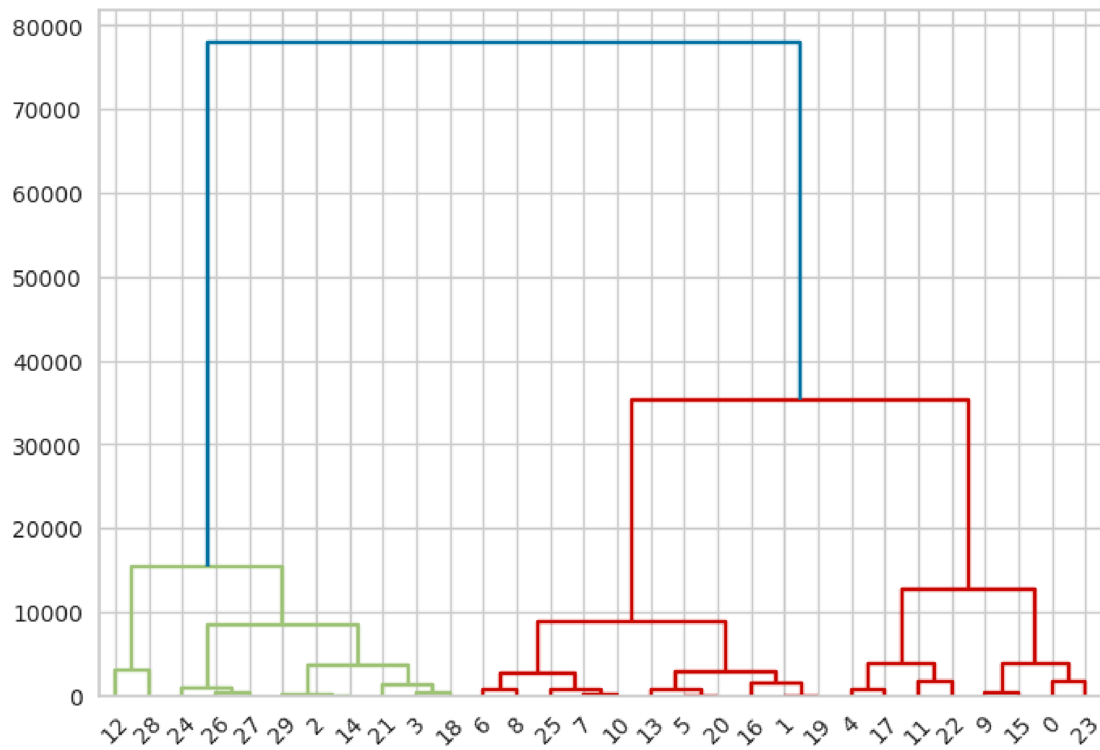
[418]:
```
den = sch.dendrogram(sch.linkage(X_train_pca.iloc[: 30], method = 'ward'))
```



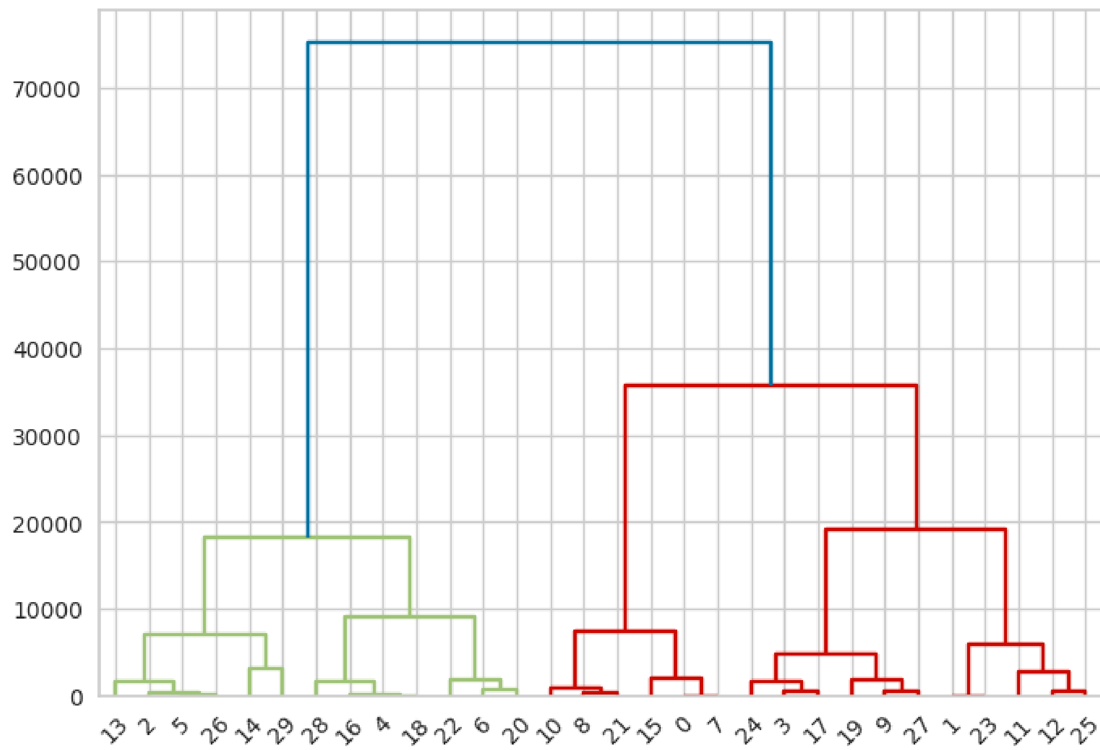
[419]:
```
den = sch.dendrogram(sch.linkage(X_test_pca.iloc[: 30], method = 'ward'))
```



[420]:
```
den = sch.dendrogram(sch.linkage(X_train.iloc[: 30,:-1], method = 'ward'))
```



[421]:
```
den = sch.dendrogram(sch.linkage(X_test.iloc[: 30,:-1], method = 'ward'))
```

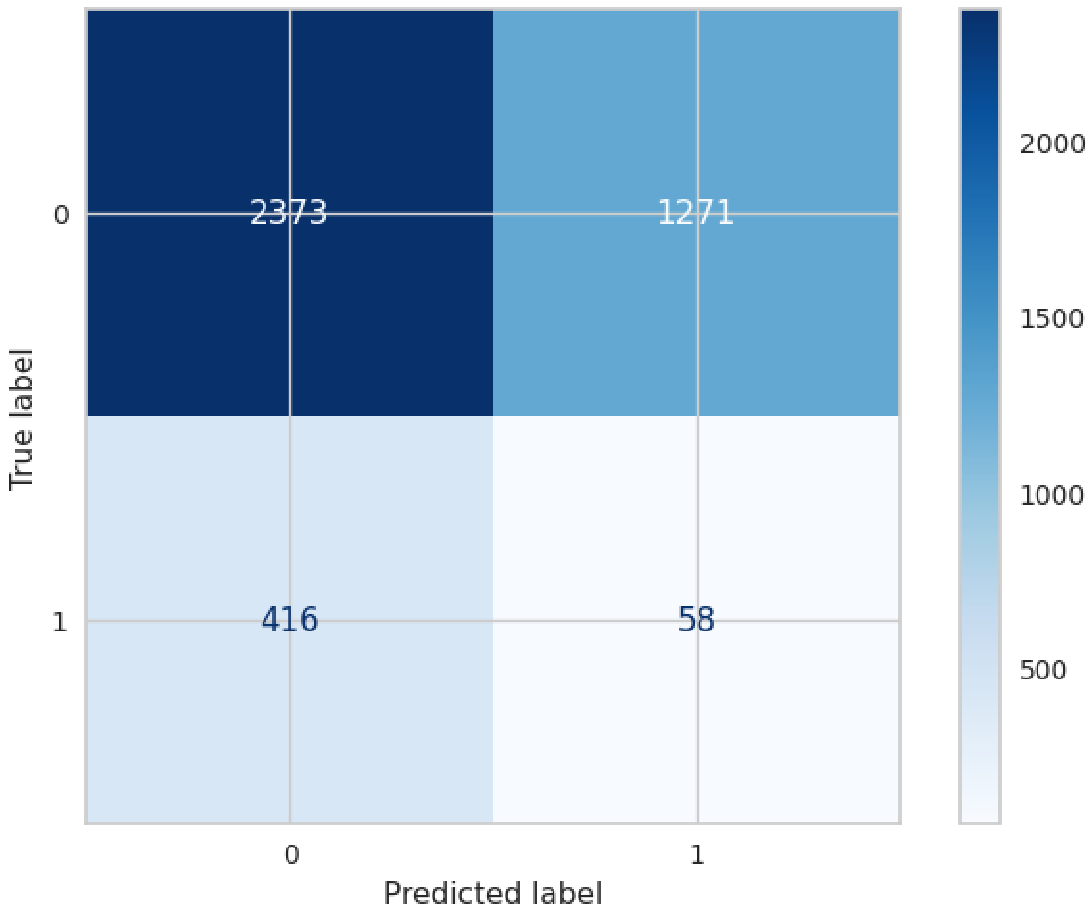


[422]:
```
AggClusteringModel =␣
↪AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage='ward')
y_train_pred = AggClusteringModel.fit_predict(X_train_pca)
y_pred = AggClusteringModel.fit_predict(X_test_pca)
```

[423]:
```
value_agg_pca =␣
↪Check(cluster=1,y_train2=y_train,y_train_pred=y_train_pred,y_test2=y_test,y_pred=y_pred)
```

```
Model Train Score is :  0.5795822538860104
Model Test Score is :  0.5903351141330743
F1 Score is :  0.06433721575152523
Recall Score is :  0.12236286919831224
Precision Score is :  0.0436418359668924
AUC Value  :  0.386785166761615

Classification Report is :                precision    recall  f1-score
support

           0       0.85      0.65      0.74      3644
           1       0.04      0.12      0.06       474

    accuracy                           0.59      4118
   macro avg       0.45      0.39      0.40      4118
weighted avg       0.76      0.59      0.66      4118
```

```
Confusion Matrix is :
 [[2373 1271]
 [ 416   58]]
```




```
[424]: fig = go.Figure()
for color,y_ in zip(['red','orange'],[0,1]):
    pca = X_train_pca[y_train_pred==y_]
    scatter_trace = go.Scatter(
        x=pca['Feature1'],
        y=pca['Feature2'],
        mode='markers',
        marker=dict(
            size=20,
            color=color,
            symbol='circle',
            opacity=0.8
        ),
```

```
        name=f'Train Cluster {y_}'
    )
    fig.add_trace(scatter_trace)
for color,y_ in zip(['green','blue'],[0,1]):
    pca = X_test_pca[y_pred==y_]
    scatter_trace = go.Scatter(
        x=pca['Feature1'],
        y=pca['Feature2'],
        mode='markers',
        marker=dict(
            size=10,
            color=color,
            symbol='circle',
            opacity=0.8
        ),
        name=f'Test Cluster {y_}'
    )
    fig.add_trace(scatter_trace)
fig.update_layout(
    title_text='PCA',
    title_x=0.5,
    title_font=dict(size=20),
    xaxis_title='Feature1',
    yaxis_title='Feature2',
    font=dict(size=15),
    width=1000,
    height=700,
    xaxis=dict(tickangle=-90),
    template='plotly_dark'
)
fig.update_annotations(font=dict(size=20))
fig.show()
```

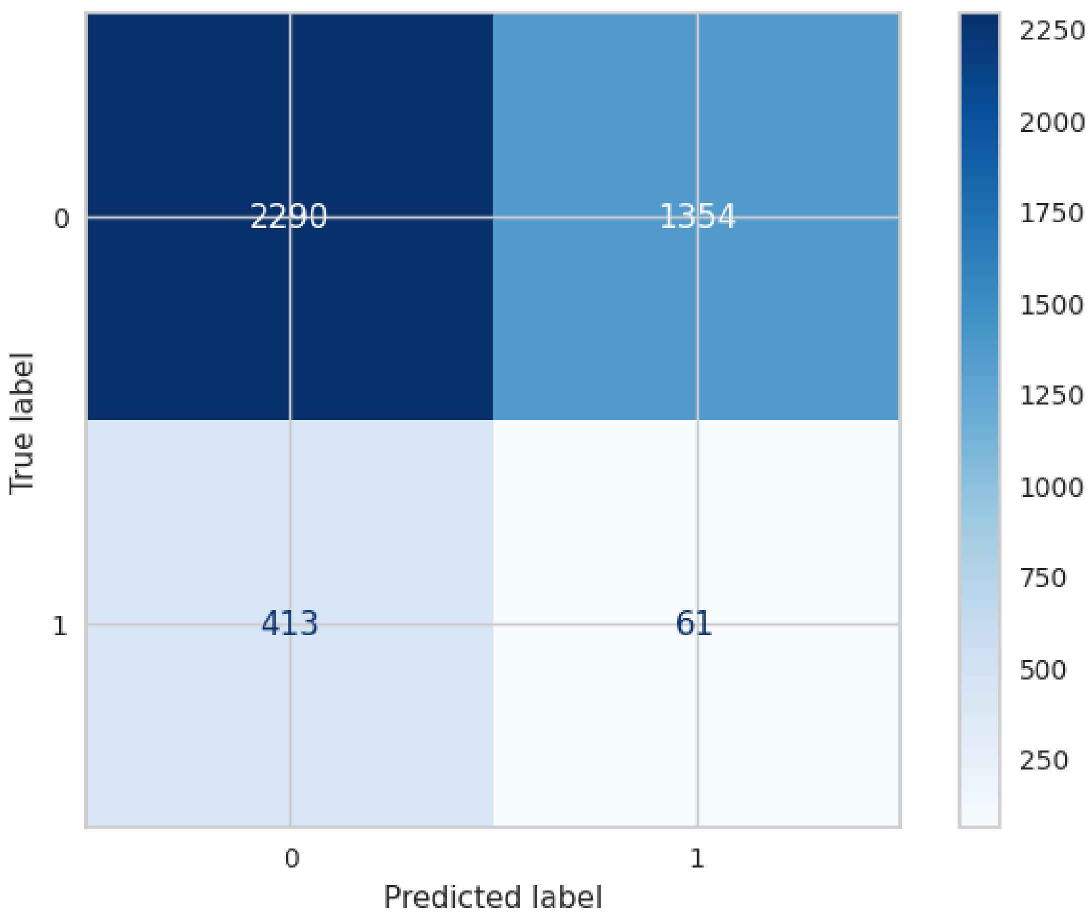
[425]:
```
AggClusteringModel =␣
 ↪AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage='ward')
y_train_pred = AggClusteringModel.fit_predict(X_train.iloc[:,:-1])
y_pred = AggClusteringModel.fit_predict(X_test.iloc[:,:-1])
```

[426]:
```
y_train = X_train.iloc[:,-1]
y_test = X_test.iloc[:,-1]
```

[427]:
```
value_agg =␣
 ↪Check(cluster=1,y_train2=y_train,y_train_pred=y_train_pred,y_test2=y_test,y_pred=y_pred)
```

```
Model Train Score is :  0.7206660189982729
Model Test Score is :  0.5709082078678971
F1 Score is :  0.06458443620963472
Recall Score is :  0.12869198312236288
```

```
Precision Score is :  0.0431095406360424
AUC Value  :  0.37856113974998495

Classification Report is :                 precision    recall  f1-score
support

           0       0.85      0.63      0.72      3644
           1       0.04      0.13      0.06       474

    accuracy                           0.57      4118
   macro avg       0.45      0.38      0.39      4118
weighted avg       0.75      0.57      0.65      4118

Confusion Matrix is :
 [[2290 1354]
 [ 413   61]]
```



[428]:
```
list = [value_kmeans,value_kmeans_pca,value_agg,value_agg_pca]
df = pd.DataFrame(list,columns=['Train Accuracy','Test Accuracy','Test␣
 ↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Kmeans','Kmeans␣
 ↪PCA','AgglomerativeClustering','AgglomerativeClustering PCA']
df.set_index('Models', inplace=True)
df
```

[428]:
```
                             Train Accuracy  Test Accuracy   Test F1  \
Models
Kmeans                             0.433533       0.443419  0.075806
Kmeans PCA                         0.465943       0.470374  0.185282
AgglomerativeClustering            0.720666       0.570908  0.064584
AgglomerativeClustering PCA        0.579582       0.590335  0.064337

                             Test Recall  Test Precision       AUC
Models
Kmeans                          0.198312        0.046859  0.336807
Kmeans PCA                      0.523207        0.112574  0.493354
AgglomerativeClustering         0.128692        0.043110  0.378561
AgglomerativeClustering PCA     0.122363        0.043642  0.386785
```

[429]:
```
models_draw(df)
```

** #

DL Models

Tabel of Contents

Deep Learning Models

[430]:
```
X_train_c,y_train_c,X_test_c,y_test_c=Split(X_classification,y_classification)
X_train_r,y_train_r,X_test_r,y_test_r=Split(X_regression,y_regression,classification=0)
```

```
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
X_train shape is  (37056, 20)
X_test shape is  (4118, 20)
y_train shape is  (37056,)
y_test shape is  (4118,)
```

[431]:
```
classification_Input = keras.Input(shape=(X_classification.shape[1],))
regression_Input = keras.Input(shape=(X_regression.shape[1],))

dense_layer1 = keras.layers.Dense(128, activation='relu', name='Dense_Layer1')
dense_layer2 = keras.layers.Dense(256, activation='relu', name='Dense_Layer2')
```

```
batch_norm1 = keras.layers.BatchNormalization(name='BatchNorm1')
dropout1 = keras.layers.Dropout(0.5, name='Dropout1')

batch_norm2 = keras.layers.BatchNormalization(name='BatchNorm2')
dropout2 = keras.layers.Dropout(0.5, name='Dropout2')

classification_output = dense_layer1(classification_Input)
classification_output = batch_norm1(classification_output)
classification_output = dropout1(classification_output)

regression_output = dense_layer1(regression_Input)
regression_output = batch_norm1(regression_output)
regression_output = dropout1(regression_output)

layer = dense_layer2(classification_output)
layer2 = dense_layer2(regression_output)

layer = batch_norm2(layer)
layer = dropout2(layer)

layer2 = batch_norm2(layer2)
layer2 = dropout2(layer2)

layer_C = keras.layers.Dense(1, activation='sigmoid',␣
 ↪name='Dense_Layer3')(layer)
layer_R = keras.layers.Dense(1, name='Dense_Layer4')(layer2)

model = keras.Model(inputs=[classification_Input, regression_Input],␣
 ↪outputs=[layer_C, layer_R])
```

[432]:
```
model.summary()
```

```
Model: "functional_1"


 Layer (type)           Output Shape          Param #  Connected to

 input_layer            (None, 20)                  0  -
 (InputLayer)

 input_layer_1          (None, 20)                  0  -
 (InputLayer)

 Dense_Layer1           (None, 128)             2,688  input_layer[0][0…
 (Dense)                                               input_layer_1[0]…

 BatchNorm1             (None, 128)               512  Dense_Layer1[0][…
```

```
(BatchNormalizatio…                                 Dense_Layer1[1][…

Dropout1 (Dropout)     (None, 128)               0  BatchNorm1[0][0],
                                                    BatchNorm1[1][0]

Dense_Layer2           (None, 256)          33,024  Dropout1[0][0],
(Dense)                                             Dropout1[1][0]

BatchNorm2             (None, 256)           1,024  Dense_Layer2[0][…
(BatchNormalizatio…                                 Dense_Layer2[1][…

Dropout2 (Dropout)     (None, 256)               0  BatchNorm2[0][0],
                                                    BatchNorm2[1][0]

Dense_Layer3           (None, 1)               257  Dropout2[0][0]
(Dense)

Dense_Layer4           (None, 1)               257  Dropout2[1][0]
(Dense)
```

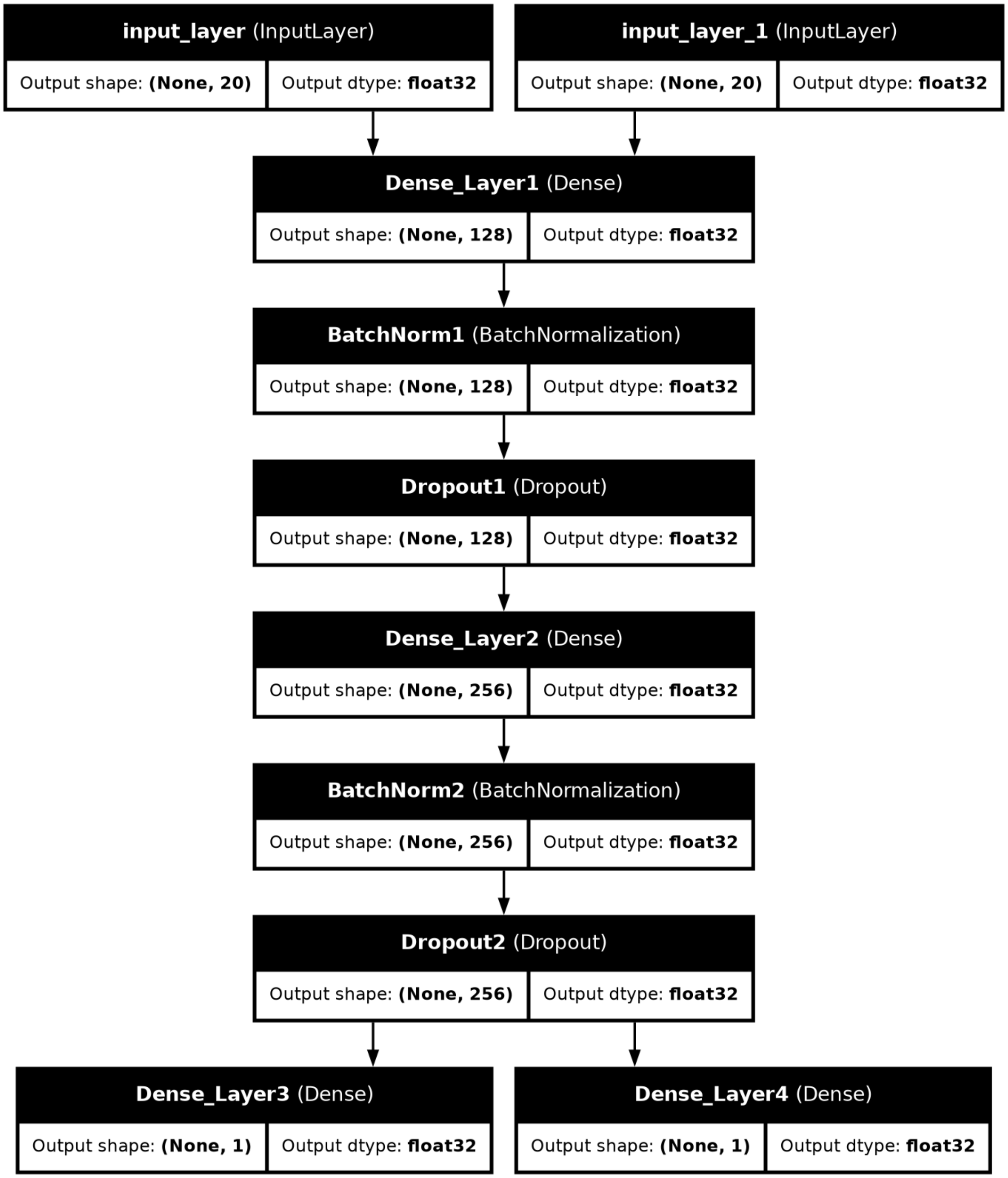
Total params: 37,762 (147.51 KB)

Trainable params: 36,994 (144.51 KB)

Non-trainable params: 768 (3.00 KB)

[433]:
```python
keras.utils.plot_model(model, to_file='model.png', show_shapes=True,␣
↪show_layer_names=True,show_dtype=True,dpi=120)
```

[433]:



[434]:
```
model.compile(optimizer='adam',
              loss={'Dense_Layer3': 'binary_crossentropy', 'Dense_Layer4':␣
 ↪'mse'},
              metrics={'Dense_Layer3': 'accuracy', 'Dense_Layer4': 'mae'})
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.keras",␣
 ↪save_best_only=True)
```

```
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,␣
↪restore_best_weights=True)
hist = model.fit([X_train_c, X_train_r], [y_train_c, y_train_r],
          epochs=50,
          batch_size=32,validation_split=.1,
          callbacks=[checkpoint_cb, early_stopping_cb])
```

```
Epoch 1/50
1043/1043          7s 3ms/step -
Dense_Layer3_accuracy: 0.8019 - Dense_Layer4_mae: 0.7588 - loss: 1.7538 -
val_Dense_Layer3_accuracy: 0.8972 - val_Dense_Layer4_mae: 0.2077 - val_loss:
0.3830
Epoch 2/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8894 - Dense_Layer4_mae: 0.2285 - loss: 0.3429 -
val_Dense_Layer3_accuracy: 0.8967 - val_Dense_Layer4_mae: 0.2126 - val_loss:
0.3752
Epoch 3/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8922 - Dense_Layer4_mae: 0.2250 - loss: 0.3144 -
val_Dense_Layer3_accuracy: 0.8929 - val_Dense_Layer4_mae: 0.2138 - val_loss:
0.4155
Epoch 4/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8998 - Dense_Layer4_mae: 0.2272 - loss: 0.3073 -
val_Dense_Layer3_accuracy: 0.8961 - val_Dense_Layer4_mae: 0.2652 - val_loss:
0.4769
Epoch 5/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8988 - Dense_Layer4_mae: 0.2307 - loss: 0.3082 -
val_Dense_Layer3_accuracy: 0.9039 - val_Dense_Layer4_mae: 0.2741 - val_loss:
0.4669
Epoch 6/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8989 - Dense_Layer4_mae: 0.2278 - loss: 0.3013 -
val_Dense_Layer3_accuracy: 0.9066 - val_Dense_Layer4_mae: 0.2396 - val_loss:
0.4158
Epoch 7/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8983 - Dense_Layer4_mae: 0.2266 - loss: 0.3001 -
val_Dense_Layer3_accuracy: 0.9012 - val_Dense_Layer4_mae: 0.2130 - val_loss:
0.4305
Epoch 8/50
1043/1043          3s 3ms/step -
Dense_Layer3_accuracy: 0.8975 - Dense_Layer4_mae: 0.2246 - loss: 0.3020 -
val_Dense_Layer3_accuracy: 0.9042 - val_Dense_Layer4_mae: 0.2043 - val_loss:
0.5365
Epoch 9/50
```

```
1043/1043            3s 3ms/step -
Dense_Layer3_accuracy: 0.9005 - Dense_Layer4_mae: 0.2213 - loss: 0.2972 -
val_Dense_Layer3_accuracy: 0.9007 - val_Dense_Layer4_mae: 0.1980 - val_loss:
0.4297
Epoch 10/50
1043/1043            3s 3ms/step -
Dense_Layer3_accuracy: 0.9014 - Dense_Layer4_mae: 0.2122 - loss: 0.2857 -
val_Dense_Layer3_accuracy: 0.9039 - val_Dense_Layer4_mae: 0.2104 - val_loss:
0.4719
Epoch 11/50
1043/1043            3s 3ms/step -
Dense_Layer3_accuracy: 0.9006 - Dense_Layer4_mae: 0.2120 - loss: 0.2871 -
val_Dense_Layer3_accuracy: 0.8940 - val_Dense_Layer4_mae: 0.1980 - val_loss:
0.4694
Epoch 12/50
1043/1043            3s 3ms/step -
Dense_Layer3_accuracy: 0.9008 - Dense_Layer4_mae: 0.2094 - loss: 0.2819 -
val_Dense_Layer3_accuracy: 0.9004 - val_Dense_Layer4_mae: 0.2136 - val_loss:
0.5121
```

[435]:
```
model.evaluate([X_test_c, X_test_r], [y_test_c, y_test_r])
```

```
129/129            0s 2ms/step -
Dense_Layer3_accuracy: 0.8902 - Dense_Layer4_mae: 0.2182 - loss: 0.3857
```

[435]:
```
[0.3855230510234833, 0.8897523283958435, 0.21595177054405212]
```

[436]:
```
hist_=pd.DataFrame(hist.history)
hist_
```

[436]:
```
    Dense_Layer3_accuracy  Dense_Layer4_mae      loss  \
0                0.857391          0.494668  0.959694
1                0.892084          0.224810  0.330215
2                0.895532          0.225326  0.312462
3                0.897901          0.227626  0.307862
4                0.898231          0.228529  0.304783
5                0.898561          0.228197  0.301962
6                0.898201          0.226457  0.300962
7                0.897691          0.224494  0.298259
8                0.901439          0.218621  0.292451
9                0.900750          0.211126  0.285878
10               0.900480          0.210736  0.285988
11               0.900600          0.209527  0.282296

    val_Dense_Layer3_accuracy  val_Dense_Layer4_mae  val_loss
0                    0.897194              0.207707  0.383009
1                    0.896654              0.212646  0.375190
2                    0.892876              0.213811  0.415541
```

3	0.896114	0.265242	0.476920
4	0.903940	0.274134	0.466892
5	0.906638	0.239562	0.415806
6	0.901241	0.212971	0.430500
7	0.904209	0.204327	0.536468
8	0.900702	0.197997	0.429677
9	0.903940	0.210409	0.471945
10	0.893956	0.197986	0.469387
11	0.900432	0.213580	0.512110

[437]:
```
def summary_plot():
    fig = make_subplots(rows=2, cols=2, subplot_titles=("Total Loss",'',␣
 ↪"Classification Accuracy", "Regression MAE"))
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['loss'], mode='lines',␣
 ↪name='Total Loss', line=dict(color='blue')), row=1, col=1)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['val_loss'], mode='lines',␣
 ↪name='Validation Loss', line=dict(color='orange')), row=1, col=1)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['Dense_Layer3_accuracy'],␣
 ↪mode='lines', name='Train Classification Accuracy', line=dict(color='red')),␣
 ↪row=2, col=1)
    fig.add_trace(go.Scatter(x=hist_.index,␣
 ↪y=hist_['val_Dense_Layer3_accuracy'], mode='lines', name='Validation␣
 ↪Classification Accuracy', line=dict(color='red')), row=2, col=1)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['Dense_Layer4_mae'],␣
 ↪mode='lines', name='Train Regression MAE', line=dict(color='purple')),␣
 ↪row=2, col=2)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['val_Dense_Layer4_mae'],␣
 ↪mode='lines', name='Validation Regression MAE', line=dict(color='purple')),␣
 ↪row=2, col=2)
    fig.update_layout(
        title_text="Training Summary",
        title_x=0.5,
        title_font=dict(size=20),
        font=dict(size=15),
        width=1100,
        height=1000,
        template='plotly_dark'
    )
    fig.update_annotations(font=dict(size=20))
    fig.show()
```

[438]:
```
summary_plot()
```

[439]:
```
predictions = model.predict([X_test_c,X_test_r])
```

```
129/129                0s 3ms/step
```

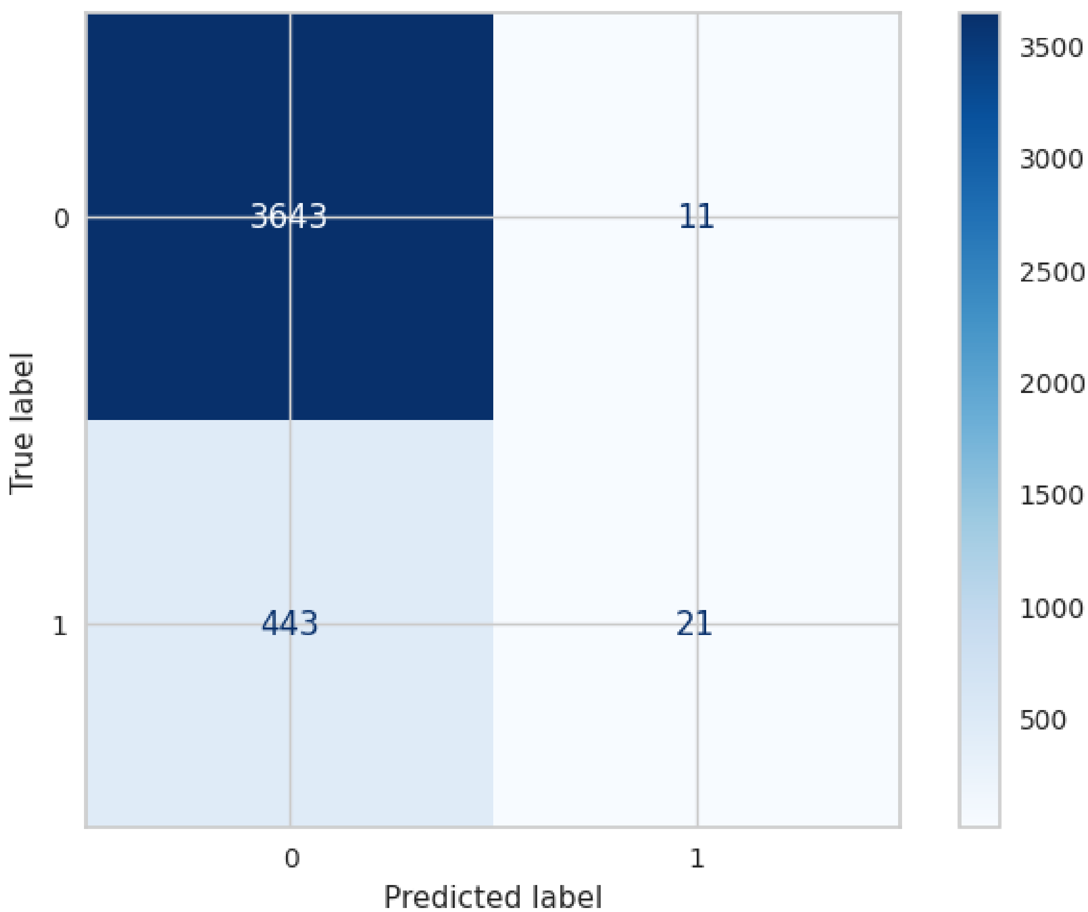
[440]:
```
classification_predictions = np.where(predictions[0]>=.5,1,0)
regression_predictions = predictions[1]
```

[441]:
```
def Check(model_22 = 1):
    if model_22:
        train = accuracy_score(y_train_c,np.where(model.
 ↪predict([X_train_c,X_train_r])[0]>=.5,1,0))
    else:
        train = accuracy_score(y_train_c,np.where(model2.predict(X_train_c)>=.
 ↪5,1,0))
    y_pred=classification_predictions
    test = accuracy_score(y_test_c,y_pred)
    print('Model Train Score is : ' , train)
    print('Model Test Score is : ' , test)
    F1Score = f1_score(y_test_c, y_pred)
    print('F1 Score is : ', F1Score)
    RecallScore = recall_score(y_test_c, y_pred)
    print('Recall Score is : ', RecallScore)
    PrecisionScore = precision_score(y_test_c, y_pred)
    print('Precision Score is : ', PrecisionScore)
    fprValue2, tprValue2, thresholdsValue2 = roc_curve(y_test_c,y_pred)
    AUCValue = auc(fprValue2, tprValue2)
    print('AUC Value  : ', AUCValue)
    Area(fprValue2,tprValue2,AUCValue)
    ClassificationReport = classification_report(y_test_c,y_pred)
    print('Classification Report is : ', ClassificationReport)
    CM = confusion_matrix(y_test_c, y_pred)
    print('Confusion Matrix is : \n', CM)
    disp = ConfusionMatrixDisplay(confusion_matrix=CM, display_labels=[0,1])
    disp.plot(cmap='Blues')
    values=[train,test,F1Score,RecallScore,PrecisionScore,AUCValue]
    return values
def Check_R():
    y_pred = regression_predictions
    print('R2 Score Train :',r2_score(y_train_c,model.
 ↪predict([X_train_c,X_train_r])[1]))
    print('R2 Score Test :',r2_score(y_test_c,y_pred))
    MAEValue = mean_absolute_error(y_test_c, y_pred)
    print('Mean Absolute Error Value is : ', MAEValue)
    MSEValue = mean_squared_error(y_test_c, y_pred)
    print('Mean Squared Error Value is : ', MSEValue)
    MdSEValue = median_absolute_error(y_test_c, y_pred)
    print('Median Absolute Error Value is : ', MdSEValue )
```

[442]:
```
values_d = Check()
```

```
1158/1158              2s 2ms/step
Model Train Score is :  0.8921901986183074
```

```
Model Test Score is :  0.889752306945119
F1 Score is :  0.08467741935483872
Recall Score is :  0.04525862068965517
Precision Score is :  0.65625
AUC Value  :  0.5211241105637657

Classification Report is :                precision    recall  f1-score
support

           0       0.89      1.00      0.94      3654
           1       0.66      0.05      0.08       464

    accuracy                           0.89      4118
   macro avg       0.77      0.52      0.51      4118
weighted avg       0.87      0.89      0.84      4118

Confusion Matrix is :
 [[3643   11]
 [ 443   21]]
```



[443]:
```
Check_R()
```

```
1158/1158            2s 2ms/step
R2 Score Train : -0.5967131996146555
R2 Score Test : -0.6066389344623884
Mean Absolute Error Value is :  0.3877328199928336
Mean Squared Error Value is :  0.16063202201329968
Median Absolute Error Value is :  0.3480357676744461
```

RandomOverSampler

[444]:
```
X_train_c,y_train_c,X_test_c,y_test_c=Split(X_classification_over,y_classification_over)
```

```
X_train shape is  (65763, 20)
X_test shape is  (7307, 20)
y_train shape is  (65763,)
y_test shape is  (7307,)
```

[445]:
```
model2 = keras.Model(inputs=[classification_Input], outputs=[layer_C])
model2.summary()
```

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 20)	0
Dense_Layer1 (Dense)	(None, 128)	2,688
BatchNorm1 (BatchNormalization)	(None, 128)	512
Dropout1 (Dropout)	(None, 128)	0
Dense_Layer2 (Dense)	(None, 256)	33,024
BatchNorm2 (BatchNormalization)	(None, 256)	1,024
Dropout2 (Dropout)	(None, 256)	0
Dense_Layer3 (Dense)	(None, 1)	257

Total params: 37,505 (146.50 KB)

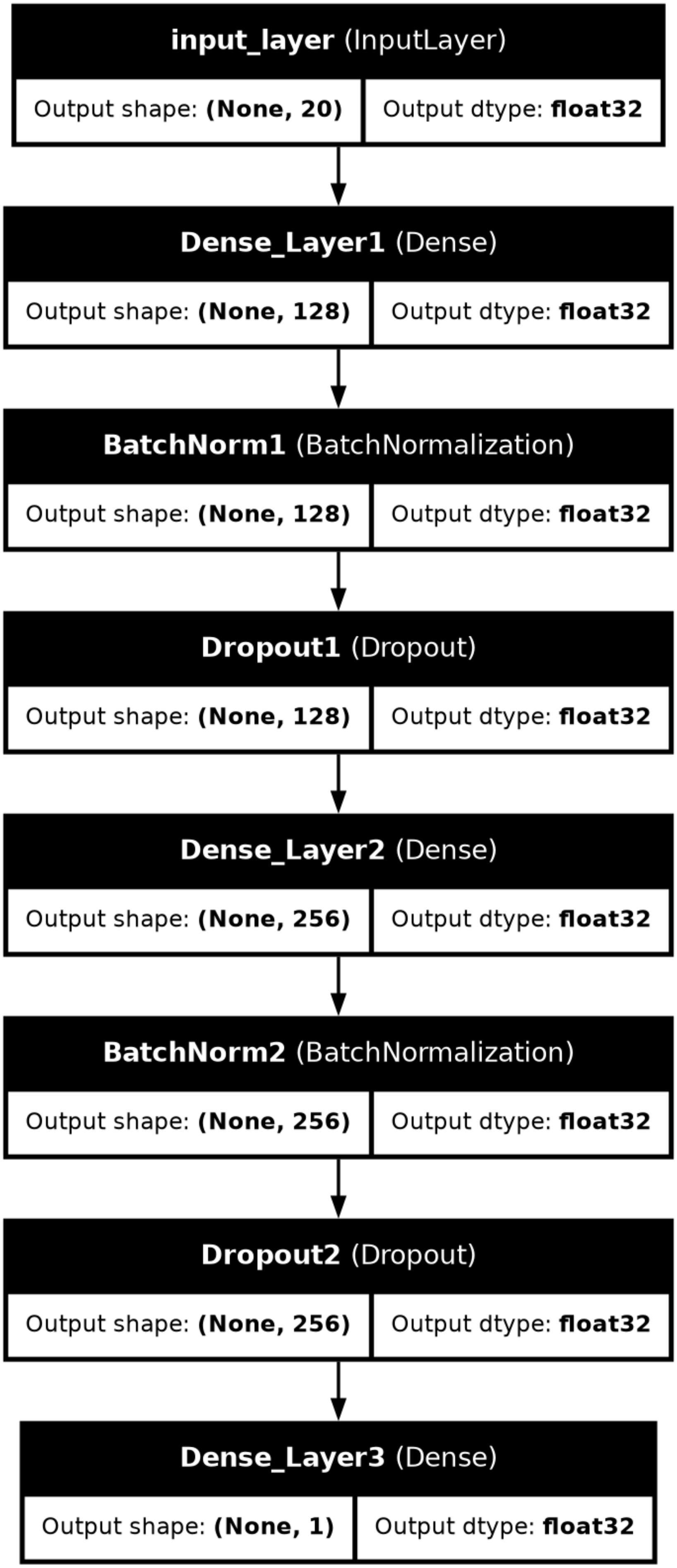
Trainable params: 36,737 (143.50 KB)

Non-trainable params: 768 (3.00 KB)

[446]:

```
keras.utils.plot_model(model2, to_file='model.png', show_shapes=True,␣
↪show_layer_names=True,show_dtype=True,dpi=120)
```

[446]:


input_layer (InputLayer)
Output shape: (None, 20)
Output dtype: float32
Dense_Layer1 (Dense)
Output shape: (None, 128)
Output dtype: float32
BatchNorm1 (BatchNormalization)
Output shape: (None, 128)
Output dtype: float32
Dropout1 (Dropout)
Output shape: (None, 128)
Output dtype: float32
Dense_Layer2 (Dense)
Output shape: (None, 256)
Output dtype: float32
BatchNorm2 (BatchNormalization)
Output shape: (None, 256)
Output dtype: float32
Dropout2 (Dropout)
Output shape: (None, 256)
Output dtype: float32
Dense_Layer3 (Dense)
Output shape: (None, 1)
Output dtype: float32

[447]:
```
model2.compile(optimizer='adam',
              loss={'Dense_Layer3': 'binary_crossentropy'},
              metrics={'Dense_Layer3': 'accuracy'})
hist = model2.fit(X_train_c,y_train_c,
          epochs=50,
          batch_size=32,validation_split=.1,
          callbacks=[checkpoint_cb, early_stopping_cb])
```

```
Epoch 1/50
1850/1850          8s 3ms/step -
accuracy: 0.8286 - loss: 0.4188 - val_accuracy: 0.8429 - val_loss: 0.3503
Epoch 2/50
1850/1850          5s 3ms/step -
accuracy: 0.8544 - loss: 0.3581 - val_accuracy: 0.8670 - val_loss: 0.3213
Epoch 3/50
1850/1850          5s 3ms/step -
accuracy: 0.8593 - loss: 0.3464 - val_accuracy: 0.8653 - val_loss: 0.3198
Epoch 4/50
1850/1850          5s 3ms/step -
accuracy: 0.8596 - loss: 0.3481 - val_accuracy: 0.8682 - val_loss: 0.3169
Epoch 5/50
1850/1850          5s 3ms/step -
accuracy: 0.8657 - loss: 0.3363 - val_accuracy: 0.8703 - val_loss: 0.3144
Epoch 6/50
1850/1850          5s 3ms/step -
accuracy: 0.8626 - loss: 0.3406 - val_accuracy: 0.8718 - val_loss: 0.3145
Epoch 7/50
1850/1850          5s 3ms/step -
accuracy: 0.8618 - loss: 0.3385 - val_accuracy: 0.8752 - val_loss: 0.3111
Epoch 8/50
1850/1850          5s 3ms/step -
accuracy: 0.8661 - loss: 0.3340 - val_accuracy: 0.8712 - val_loss: 0.3079
Epoch 9/50
1850/1850          5s 3ms/step -
accuracy: 0.8656 - loss: 0.3347 - val_accuracy: 0.8714 - val_loss: 0.3088
Epoch 10/50
1850/1850          5s 3ms/step -
accuracy: 0.8624 - loss: 0.3395 - val_accuracy: 0.8726 - val_loss: 0.3073
Epoch 11/50
1850/1850          5s 3ms/step -
accuracy: 0.8636 - loss: 0.3339 - val_accuracy: 0.8771 - val_loss: 0.2985
Epoch 12/50
1850/1850          6s 3ms/step -
accuracy: 0.8662 - loss: 0.3306 - val_accuracy: 0.8680 - val_loss: 0.3118
Epoch 13/50
```

```
1850/1850            5s 3ms/step -
accuracy: 0.8635 - loss: 0.3363 - val_accuracy: 0.8703 - val_loss: 0.3057
Epoch 14/50
1850/1850            5s 3ms/step -
accuracy: 0.8667 - loss: 0.3331 - val_accuracy: 0.8762 - val_loss: 0.2981
Epoch 15/50
1850/1850            5s 3ms/step -
accuracy: 0.8690 - loss: 0.3269 - val_accuracy: 0.8756 - val_loss: 0.3002
Epoch 16/50
1850/1850            5s 3ms/step -
accuracy: 0.8678 - loss: 0.3279 - val_accuracy: 0.8794 - val_loss: 0.3006
Epoch 17/50
1850/1850            5s 3ms/step -
accuracy: 0.8725 - loss: 0.3234 - val_accuracy: 0.8776 - val_loss: 0.2981
Epoch 18/50
1850/1850            5s 3ms/step -
accuracy: 0.8726 - loss: 0.3199 - val_accuracy: 0.8800 - val_loss: 0.2917
Epoch 19/50
1850/1850            5s 3ms/step -
accuracy: 0.8706 - loss: 0.3225 - val_accuracy: 0.8814 - val_loss: 0.2950
Epoch 20/50
1850/1850            5s 3ms/step -
accuracy: 0.8749 - loss: 0.3207 - val_accuracy: 0.8762 - val_loss: 0.3054
Epoch 21/50
1850/1850            5s 3ms/step -
accuracy: 0.8710 - loss: 0.3242 - val_accuracy: 0.8820 - val_loss: 0.2900
Epoch 22/50
1850/1850            5s 3ms/step -
accuracy: 0.8728 - loss: 0.3173 - val_accuracy: 0.8779 - val_loss: 0.3028
Epoch 23/50
1850/1850            5s 3ms/step -
accuracy: 0.8720 - loss: 0.3201 - val_accuracy: 0.8793 - val_loss: 0.2974
Epoch 24/50
1850/1850            5s 3ms/step -
accuracy: 0.8728 - loss: 0.3153 - val_accuracy: 0.8785 - val_loss: 0.2927
Epoch 25/50
1850/1850            5s 3ms/step -
accuracy: 0.8718 - loss: 0.3194 - val_accuracy: 0.8787 - val_loss: 0.2967
Epoch 26/50
1850/1850            5s 3ms/step -
accuracy: 0.8704 - loss: 0.3197 - val_accuracy: 0.8835 - val_loss: 0.2902
Epoch 27/50
1850/1850            5s 3ms/step -
accuracy: 0.8732 - loss: 0.3184 - val_accuracy: 0.8765 - val_loss: 0.2973
Epoch 28/50
1850/1850            5s 3ms/step -
accuracy: 0.8720 - loss: 0.3169 - val_accuracy: 0.8747 - val_loss: 0.3056
Epoch 29/50
```

```
1850/1850            5s 3ms/step -
accuracy: 0.8709 - loss: 0.3206 - val_accuracy: 0.8814 - val_loss: 0.2951
Epoch 30/50
1850/1850            5s 3ms/step -
accuracy: 0.8724 - loss: 0.3181 - val_accuracy: 0.8800 - val_loss: 0.2883
Epoch 31/50
1850/1850            5s 3ms/step -
accuracy: 0.8740 - loss: 0.3172 - val_accuracy: 0.8765 - val_loss: 0.2983
Epoch 32/50
1850/1850            5s 3ms/step -
accuracy: 0.8740 - loss: 0.3153 - val_accuracy: 0.8825 - val_loss: 0.2918
Epoch 33/50
1850/1850            5s 3ms/step -
accuracy: 0.8737 - loss: 0.3171 - val_accuracy: 0.8767 - val_loss: 0.2951
Epoch 34/50
1850/1850            5s 3ms/step -
accuracy: 0.8725 - loss: 0.3179 - val_accuracy: 0.8805 - val_loss: 0.2931
Epoch 35/50
1850/1850            5s 3ms/step -
accuracy: 0.8707 - loss: 0.3223 - val_accuracy: 0.8819 - val_loss: 0.2979
Epoch 36/50
1850/1850            5s 3ms/step -
accuracy: 0.8706 - loss: 0.3185 - val_accuracy: 0.8814 - val_loss: 0.2975
Epoch 37/50
1850/1850            5s 3ms/step -
accuracy: 0.8706 - loss: 0.3198 - val_accuracy: 0.8790 - val_loss: 0.2934
Epoch 38/50
1850/1850            5s 3ms/step -
accuracy: 0.8740 - loss: 0.3158 - val_accuracy: 0.8825 - val_loss: 0.2917
Epoch 39/50
1850/1850            10s 3ms/step -
accuracy: 0.8739 - loss: 0.3161 - val_accuracy: 0.8816 - val_loss: 0.2879
Epoch 40/50
1850/1850            5s 3ms/step -
accuracy: 0.8725 - loss: 0.3176 - val_accuracy: 0.8828 - val_loss: 0.2895
Epoch 41/50
1850/1850            5s 3ms/step -
accuracy: 0.8714 - loss: 0.3171 - val_accuracy: 0.8834 - val_loss: 0.2916
Epoch 42/50
1850/1850            5s 3ms/step -
accuracy: 0.8746 - loss: 0.3155 - val_accuracy: 0.8847 - val_loss: 0.2890
Epoch 43/50
1850/1850            6s 3ms/step -
accuracy: 0.8755 - loss: 0.3105 - val_accuracy: 0.8838 - val_loss: 0.2889
Epoch 44/50
1850/1850            6s 3ms/step -
accuracy: 0.8734 - loss: 0.3135 - val_accuracy: 0.8816 - val_loss: 0.2882
Epoch 45/50
```

```
1850/1850            6s 3ms/step -
accuracy: 0.8731 - loss: 0.3144 - val_accuracy: 0.8809 - val_loss: 0.2963
Epoch 46/50
1850/1850            6s 3ms/step -
accuracy: 0.8719 - loss: 0.3194 - val_accuracy: 0.8756 - val_loss: 0.2924
Epoch 47/50
1850/1850            6s 3ms/step -
accuracy: 0.8730 - loss: 0.3171 - val_accuracy: 0.8837 - val_loss: 0.2899
Epoch 48/50
1850/1850            5s 3ms/step -
accuracy: 0.8739 - loss: 0.3132 - val_accuracy: 0.8797 - val_loss: 0.2926
Epoch 49/50
1850/1850            5s 3ms/step -
accuracy: 0.8770 - loss: 0.3096 - val_accuracy: 0.8834 - val_loss: 0.2898
```

[448]:
```
model2.evaluate(X_test_c,y_test_c)
```

```
229/229            0s 1ms/step -
accuracy: 0.8854 - loss: 0.2838
```

[448]: [0.2808574140071869, 0.8851786255836487]

[449]:
```
hist_=pd.DataFrame(hist.history)
hist_
```

[449]:

	accuracy	loss	val_accuracy	val_loss
0	0.843916	0.378547	0.842938	0.350317
1	0.854172	0.354037	0.866961	0.321258
2	0.859021	0.347425	0.865288	0.319830
3	0.860153	0.345896	0.868177	0.316926
4	0.862197	0.342117	0.870306	0.314431
5	0.860930	0.344267	0.871826	0.314475
6	0.861488	0.338515	0.875171	0.311108
7	0.864258	0.336407	0.871218	0.307863
8	0.864563	0.335216	0.871370	0.308834
9	0.864698	0.335060	0.872586	0.307318
10	0.865441	0.332236	0.877148	0.298481
11	0.866708	0.329856	0.868025	0.311752
12	0.866759	0.330437	0.870306	0.305715
13	0.868297	0.329130	0.876235	0.298063
14	0.868533	0.325977	0.875627	0.300174
15	0.868246	0.327165	0.879428	0.300567
16	0.870138	0.324976	0.877604	0.298050
17	0.870932	0.323905	0.880036	0.291676
18	0.871236	0.322177	0.881405	0.295000
19	0.871946	0.323185	0.876235	0.305448
20	0.873281	0.319980	0.882013	0.289971
21	0.871760	0.320215	0.877908	0.302785

```
22  0.871574  0.320176      0.879276  0.297420
23  0.872487  0.317512      0.878516  0.292717
24  0.871203  0.320413      0.878668  0.296727
25  0.872250  0.319158      0.883534  0.290213
26  0.871524  0.319845      0.876539  0.297285
27  0.870780  0.320379      0.874715  0.305607
28  0.872250  0.318990      0.881405  0.295066
29  0.871574  0.320304      0.880036  0.288291
30  0.874092  0.317647      0.876539  0.298292
31  0.871388  0.319435      0.882469  0.291793
32  0.872115  0.319140      0.876692  0.295069
33  0.872977  0.317697      0.880493  0.293137
34  0.871439  0.318820      0.881861  0.297871
35  0.872858  0.316919      0.881405  0.297540
36  0.871929  0.318431      0.878972  0.293413
37  0.872740  0.317294      0.882469  0.291726
38  0.872470  0.318711      0.881557  0.287946
39  0.873247  0.316782      0.882773  0.289496
40  0.872487  0.314994      0.883381  0.291615
41  0.874447  0.314769      0.884750  0.288973
42  0.872892  0.315695      0.883838  0.288933
43  0.873179  0.313844      0.881557  0.288172
44  0.873095  0.315220      0.880949  0.296259
45  0.873585  0.315946      0.875627  0.292418
46  0.873585  0.315087      0.883686  0.289891
47  0.874734  0.313269      0.879732  0.292560
48  0.875815  0.311884      0.883381  0.289758
```

[450]:

```
def summary_plot2():
    fig = make_subplots(rows=1, cols=2, subplot_titles=("Total␣
↪Loss","Classification Accuracy"))
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['loss'], mode='lines',␣
↪name='Train Loss', line=dict(color='blue')), row=1, col=1)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['val_loss'], mode='lines',␣
↪name='Validation Loss', line=dict(color='blue')), row=1, col=1)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['accuracy'], mode='lines',␣
↪name='Train Accuracy', line=dict(color='red')), row=1, col=2)
    fig.add_trace(go.Scatter(x=hist_.index, y=hist_['val_accuracy'],␣
↪mode='lines', name='Validation Accuracy', line=dict(color='red')), row=1,␣
↪col=2)
    fig.update_layout(
        title_text="Training Summary",
        title_x=0.5,
        title_font=dict(size=20),
        font=dict(size=15),
        width=1100,
        height=600,
```

```
        template='plotly_dark'
    )
    fig.update_annotations(font=dict(size=20))
    fig.show()
```

[451]:
```
summary_plot2()
```

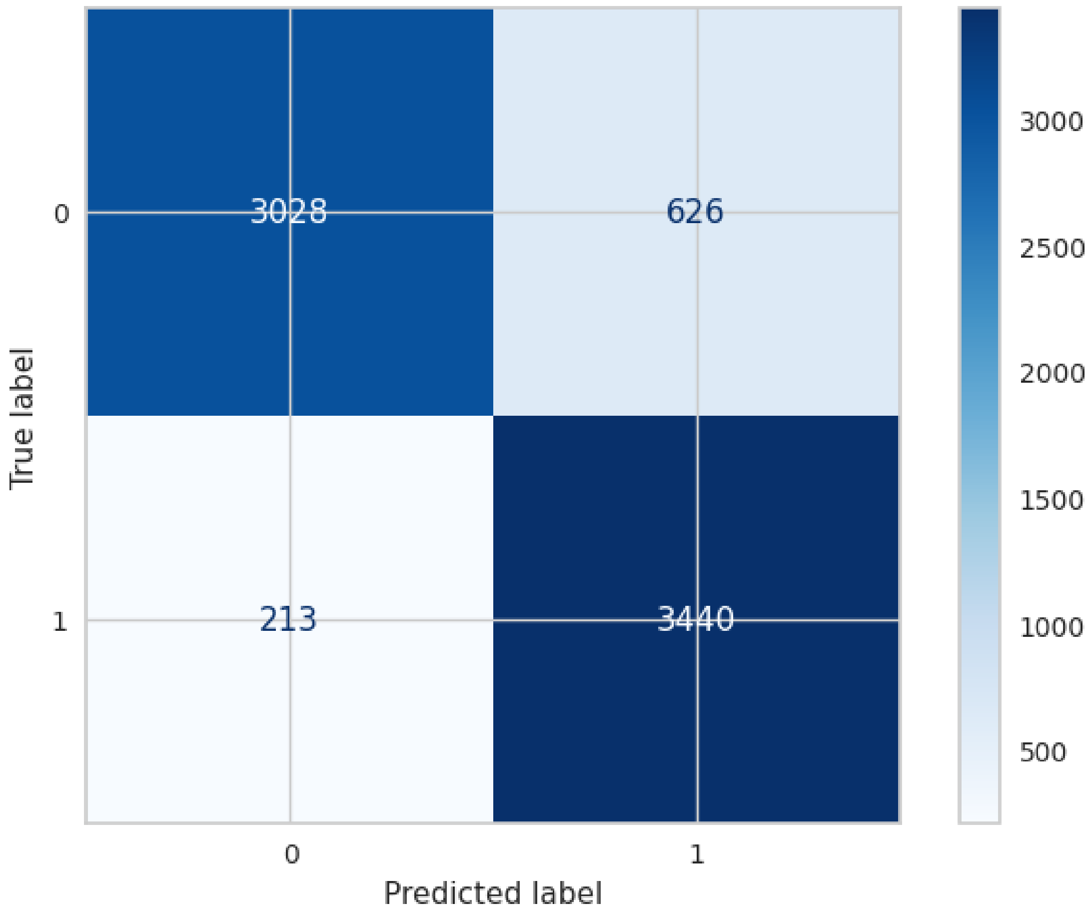
[452]:
```
predictions = model2.predict(X_test_c)
```

```
229/229              0s 2ms/step
```

[453]:
```
classification_predictions = np.where(predictions>=.5,1,0)
```

[454]:
```
value_d_over = Check(model_22=0)
```

```
2056/2056              3s 1ms/step
Model Train Score is :  0.8850569469154388
Model Test Score is :  0.8851785958669769
F1 Score is :  0.8913071641404327
Recall Score is :  0.9416917601970983
Precision Score is :  0.8460403344810624
AUC Value  :  0.8851863289217565

Classification Report is :                precision    recall  f1-score
support

           0       0.93      0.83      0.88      3654
           1       0.85      0.94      0.89      3653

    accuracy                           0.89      7307
   macro avg       0.89      0.89      0.88      7307
weighted avg       0.89      0.89      0.88      7307

Confusion Matrix is :
 [[3028  626]
 [ 213 3440]]
```



RandomUnderSampler

[455]:
```
X_train_c,y_train_c,X_test_c,y_test_c=Split(X_classification_under,y_classification_under)
```

```
X_train shape is  (8350, 20)
X_test shape is  (928, 20)
y_train shape is  (8350,)
y_test shape is  (928,)
```

[456]:
```
hist = model2.fit(X_train_c,y_train_c,
          epochs=50,
          batch_size=32,validation_split=.1,
          callbacks=[checkpoint_cb, early_stopping_cb])
```

```
Epoch 1/50
235/235              1s 3ms/step -
accuracy: 0.8747 - loss: 0.3104 - val_accuracy: 0.8719 - val_loss: 0.3217
Epoch 2/50
235/235              1s 3ms/step -
accuracy: 0.8659 - loss: 0.3160 - val_accuracy: 0.8790 - val_loss: 0.3049
```

```
Epoch 3/50
235/235              1s 3ms/step -
accuracy: 0.8650 - loss: 0.3246 - val_accuracy: 0.8802 - val_loss: 0.3075
Epoch 4/50
235/235              1s 3ms/step -
accuracy: 0.8686 - loss: 0.3120 - val_accuracy: 0.8754 - val_loss: 0.3108
Epoch 5/50
235/235              1s 3ms/step -
accuracy: 0.8677 - loss: 0.3175 - val_accuracy: 0.8790 - val_loss: 0.3106
Epoch 6/50
235/235              1s 3ms/step -
accuracy: 0.8781 - loss: 0.3048 - val_accuracy: 0.8766 - val_loss: 0.3093
Epoch 7/50
235/235              1s 3ms/step -
accuracy: 0.8755 - loss: 0.3149 - val_accuracy: 0.8754 - val_loss: 0.3119
Epoch 8/50
235/235              1s 3ms/step -
accuracy: 0.8739 - loss: 0.3180 - val_accuracy: 0.8754 - val_loss: 0.3112
Epoch 9/50
235/235              1s 3ms/step -
accuracy: 0.8712 - loss: 0.3205 - val_accuracy: 0.8802 - val_loss: 0.3112
Epoch 10/50
235/235              1s 3ms/step -
accuracy: 0.8688 - loss: 0.3113 - val_accuracy: 0.8743 - val_loss: 0.3121
```

[457]:
```
model2.evaluate(X_test_c,y_test_c)
```

```
29/29              0s 1ms/step -
accuracy: 0.8905 - loss: 0.2802
```

[457]: [0.2924061417579651, 0.8846982717514038]

[458]:
```
hist_=pd.DataFrame(hist.history)
hist_
```

[458]:

	accuracy	loss	val_accuracy	val_loss
0	0.870393	0.317221	0.871856	0.321699
1	0.868397	0.313290	0.879042	0.304934
2	0.869727	0.313788	0.880240	0.307521
3	0.869594	0.314728	0.875449	0.310806
4	0.873719	0.307169	0.879042	0.310580
5	0.872389	0.313188	0.876647	0.309325
6	0.873719	0.314463	0.875449	0.311888
7	0.870526	0.318851	0.875449	0.311185
8	0.874118	0.319702	0.880240	0.311241
9	0.866800	0.312994	0.874251	0.312073

[459]:
```
summary_plot2()
```

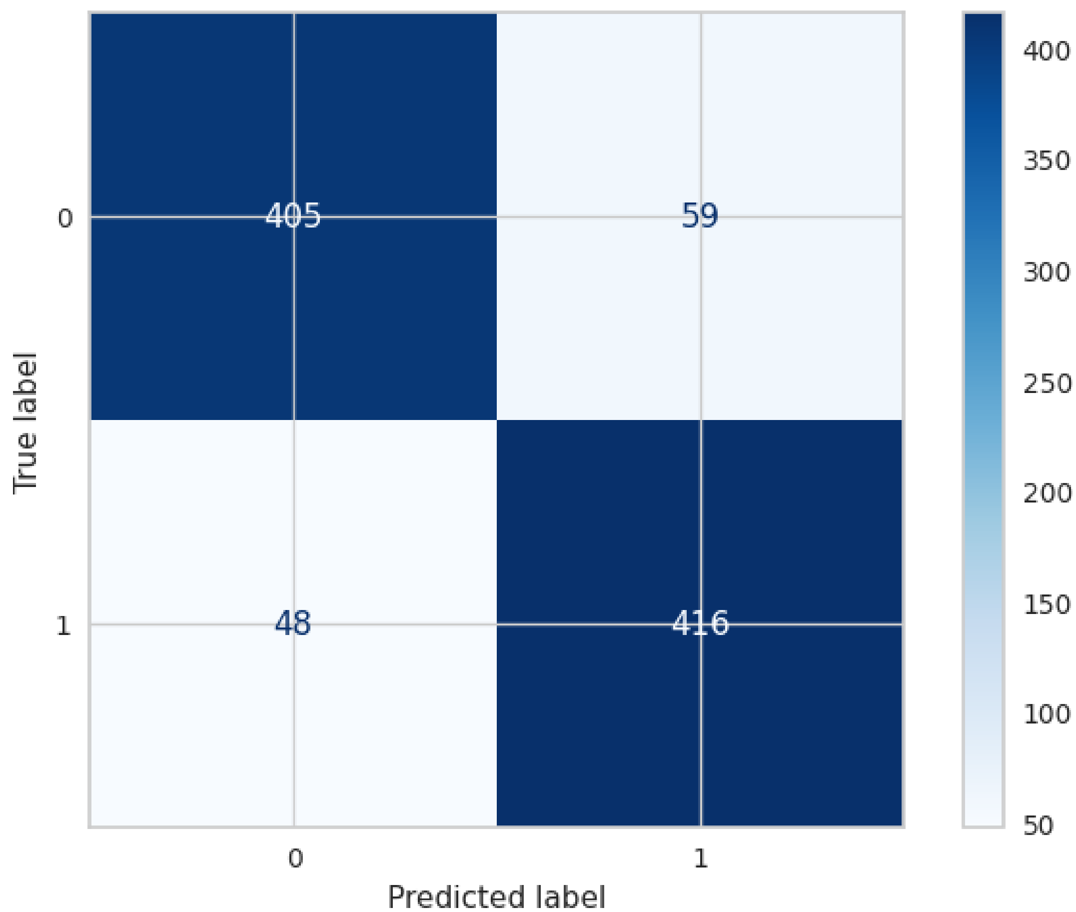
```
[460]: predictions = model2.predict(X_test_c)
```

```
29/29              0s 1ms/step
```

```
[461]: classification_predictions = np.where(predictions>=.5,1,0)
```

```
[462]: value_d_under = Check(model_22=0)
```

```
261/261              0s 1ms/step
Model Train Score is :  0.8795209580838323
Model Test Score is :  0.884698275862069
F1 Score is :  0.8860489882854099
Recall Score is :  0.896551724137931
Precision Score is :  0.8757894736842106
AUC Value  :  0.884698275862069

Classification Report is :                precision    recall  f1-score
support

           0       0.89      0.87      0.88       464
           1       0.88      0.90      0.89       464

    accuracy                           0.88       928
   macro avg       0.88      0.88      0.88       928
weighted avg       0.88      0.88      0.88       928

Confusion Matrix is :
 [[405  59]
 [ 48 416]]
```



[463]:
```
list = [values_d,value_d_over,value_d_under]
df = pd.DataFrame(list,columns=['Train Accuracy','Test Accuracy','Test␣
 ↪F1','Test Recall','Test Precision','AUC'])
df['Models'] = ['Deep Learning','Deep Learning With Over','Deep Learning With␣
 ↪Under']
df.set_index('Models', inplace=True)
df
```

[463]:
```
                         Train Accuracy  Test Accuracy   Test F1  \
Models
Deep Learning                  0.892190       0.889752  0.084677
Deep Learning With Over        0.885057       0.885179  0.891307
Deep Learning With Under       0.879521       0.884698  0.886049

                         Test Recall  Test Precision       AUC
Models
Deep Learning               0.045259        0.656250  0.521124
Deep Learning With Over     0.941692        0.846040  0.885186
```

```
Deep Learning With Under     0.896552        0.875789  0.884698
```

[464]:
```
models_draw(df)
```