CREDIT

December 23, 2024

[1]:

```
import numpy as np
import pandas as pd
import time

import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from scipy import stats
from scipy.stats import norm, skew
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

import sklearn
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import average_precision_score, precision_recall_curve

from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import Ridge, Lasso, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance
from sklearn.ensemble import AdaBoostClassifier
```

```python
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
%pip install scikit-learn pandas numpy
%pip install xgboost
%pip install --upgrade imblearn

# To ignore warnings
import warnings
warnings.filterwarnings("ignore")

# Loading the data
df = pd.read_csv("C:\\Users\\darsh\\Desktop\\project\\credit 2023 project.csv")

print(df.head())
```

```
Requirement already satisfied: scikit-learn in
c:\users\darsh\anaconda3\lib\site-packages (1.5.2)
Requirement already satisfied: pandas in c:\users\darsh\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: numpy in c:\users\darsh\anaconda3\lib\site-
packages (1.24.3)
Requirement already satisfied: scipy>=1.6.0 in
c:\users\darsh\anaconda3\lib\site-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\darsh\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\darsh\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\darsh\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\darsh\anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in
c:\users\darsh\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\darsh\anaconda3\lib\site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: xgboost in c:\users\darsh\anaconda3\lib\site-
packages (2.1.3)
Requirement already satisfied: numpy in c:\users\darsh\anaconda3\lib\site-
packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\darsh\anaconda3\lib\site-
packages (from xgboost) (1.11.1)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: imblearn in c:\users\darsh\anaconda3\lib\site-
packages (0.0)
Requirement already satisfied: imbalanced-learn in
c:\users\darsh\anaconda3\lib\site-packages (from imblearn) (0.12.4)
Requirement already satisfied: numpy>=1.17.3 in
```

```
c:\users\darsh\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.24.3)
Requirement already satisfied: scipy>=1.5.0 in
c:\users\darsh\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
c:\users\darsh\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.5.2)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\darsh\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\darsh\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 49
     46 warnings.filterwarnings("ignore")
     48 # Loading the data
---> 49 df = pd.read_csv("C:\\Users\\darsh\\Desktop\\project\\credit 2023␣
 ↪project.csv")
     51 print(df.head())

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:912, in␣
 ↪read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
 ↪usecols, dtype, engine, converters, true_values, false_values,␣
 ↪skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na,␣
 ↪na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format,␣
 ↪keep_date_col, date_parser, date_format, dayfirst, cache_dates, iterator,␣
 ↪chunksize, compression, thousands, decimal, lineterminator, quotechar,␣
 ↪quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect ␣
 ↪on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision,␣
 ↪storage_options, dtype_backend)
    899 kwds_defaults = _refine_defaults_read(
    900     dialect,
    901     delimiter,
   (…)
    908     dtype_backend=dtype_backend,
    909 )
    910 kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:577, in␣
 ↪_read(filepath_or_buffer, kwds)
    574 _validate_names(kwds.get("names", None))
    576 # Create the parser.
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)
    579 if chunksize or iterator:
```

```
    580     return parser

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1407, in␣
 ↪TextFileReader.__init__(self, f, engine, **kwds)
   1404     self.options["has_index_names"] = kwds["has_index_names"]
   1406 self.handles: IOHandles | None = None
-> 1407 self._engine = self._make_engine(f, self.engine)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1661, in␣
 ↪TextFileReader._make_engine(self, f, engine)
   1659     if "b" not in mode:
   1660         mode += "b"
-> 1661 self.handles = get_handle(
   1662     f,
   1663     mode,
   1664     encoding=self.options.get("encoding", None),
   1665     compression=self.options.get("compression", None),
   1666     memory_map=self.options.get("memory_map", False),
   1667     is_text=is_text,
   1668     errors=self.options.get("encoding_errors", "strict"),
   1669     storage_options=self.options.get("storage_options", None),
   1670 )
   1671 assert self.handles is not None
   1672 f = self.handles.handle

File ~\anaconda3\Lib\site-packages\pandas\io\common.py:859, in␣
 ↪get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,␣
 ↪errors, storage_options)
    854 elif isinstance(handle, str):
    855     # Check whether the filename is to be opened in binary mode.
    856     # Binary mode does not support 'encoding' and 'newline'.
    857     if ioargs.encoding and "b" not in ioargs.mode:
    858         # Encoding
--> 859         handle = open(
    860             handle,
    861             ioargs.mode,
    862             encoding=ioargs.encoding,
    863             errors=errors,
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
    868         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'C:
 ↪\\Users\\darsh\\Desktop\\project\\credit 2023 project.csv'
```

[11]:

```
#checking the shape
df.shape
```

[11]:

```
(568630, 31)
```

[ ]:

```

```

[12]:

```
#checking the datatypes and null/not-null distribution
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      568630 non-null  int64
 1   V1      568630 non-null  float64
 2   V2      568630 non-null  float64
 3   V3      568630 non-null  float64
 4   V4      568630 non-null  float64
 5   V5      568630 non-null  float64
 6   V6      568630 non-null  float64
 7   V7      568630 non-null  float64
 8   V8      568630 non-null  float64
 9   V9      568630 non-null  float64
 10  V10     568630 non-null  float64
 11  V11     568630 non-null  float64
 12  V12     568630 non-null  float64
 13  V13     568630 non-null  float64
 14  V14     568630 non-null  float64
 15  V15     568630 non-null  float64
 16  V16     568630 non-null  float64
 17  V17     568630 non-null  float64
 18  V18     568630 non-null  float64
 19  V19     568630 non-null  float64
 20  V20     568630 non-null  float64
 21  V21     568630 non-null  float64
 22  V22     568630 non-null  float64
 23  V23     568630 non-null  float64
 24  V24     568630 non-null  float64
 25  V25     568630 non-null  float64
 26  V26     568630 non-null  float64
 27  V27     568630 non-null  float64
 28  V28     568630 non-null  float64
 29  Amount  568630 non-null  float64
 30  Class   568630 non-null  int64
dtypes: float64(29), int64(2)
memory usage: 134.5 MB
```

[4]:

```
#checking distribution of numerical values in the dataset
df.describe()
```

[4]:

```
                  id            V1            V2            V3            V4  \
count  568630.000000  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05
mean   284314.500000  2.076882e-12 -3.248204e-12 -3.636929e-12  3.879536e-12
std    164149.486121  1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00
min         0.000000 -3.495584e+00 -4.996657e+01 -3.183760e+00 -4.951222e+00
25%    142157.250000 -5.652859e-01 -4.866777e-01 -6.492987e-01 -6.560203e-01
50%    284314.500000 -9.363846e-02 -1.358939e-01  3.528580e-04 -7.376152e-02
75%    426471.750000  8.326582e-01  3.435552e-01  6.285380e-01  7.070047e-01
max    568629.000000  2.229046e+00  4.361865e+00  1.412583e+01  3.201536e+00

                 V5            V6            V7            V8            V9  \
count  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05
mean   2.409066e-13  2.768028e-12 -9.496329e-14  2.831363e-12 -2.488498e-12
std    1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00
min   -9.952786e+00 -2.111111e+01 -4.351839e+00 -1.075634e+01 -3.751919e+00
25%   -2.934955e-01 -4.458712e-01 -2.835329e-01 -1.922572e-01 -5.687446e-01
50%    8.108788e-02  7.871758e-02  2.333659e-01 -1.145242e-01  9.252647e-02
75%    4.397368e-01  4.977881e-01  5.259548e-01  4.729905e-02  5.592621e-01
max    4.271689e+01  2.616840e+01  2.178730e+02  5.958040e+00  2.027006e+01

       …           V21           V22           V23           V24  \
count  …  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05
mean   … -3.358969e-13 -2.163216e-13  2.562302e-12 -4.924404e-14
std    …  1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00
min    … -1.938252e+01 -7.734798e+00 -3.029545e+01 -4.067968e+00
25%    … -1.664408e-01 -4.904892e-01 -2.376289e-01 -6.515801e-01
50%    … -3.743065e-02 -2.732881e-02 -5.968903e-02  1.590122e-02
75%    …  1.479787e-01  4.638817e-01  1.557153e-01  7.007374e-01
max    …  8.087080e+00  1.263251e+01  3.170763e+01  1.296564e+01

                V25           V26           V27           V28         Amount  \
count  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05  568630.000000
mean  -2.931602e-12  4.378988e-13 -1.661857e-12 -2.416333e-12   12041.957635
std    1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00    6919.644449
min   -1.361263e+01 -8.226969e+00 -1.049863e+01 -3.903524e+01      50.010000
25%   -5.541485e-01 -6.318948e-01 -3.049607e-01 -2.318783e-01    6054.892500
50%   -8.193162e-03 -1.189208e-02 -1.729111e-01 -1.392973e-02   12030.150000
75%    5.500147e-01  6.728879e-01  3.340230e-01  4.095903e-01   18036.330000
max    1.462151e+01  5.623285e+00  1.132311e+02  7.725594e+01   24039.930000

          Class
count  568630.0
mean        0.5
std         0.5
```

```
min          0.0
25%          0.0
50%          0.5
75%          1.0
max          1.0

[8 rows x 31 columns]
```

[5]:

```
#checking the class distribution of the target variable
df['Class'].value_counts()
```

[5]:

```
Class
0    284315
1    284315
Name: count, dtype: int64
```

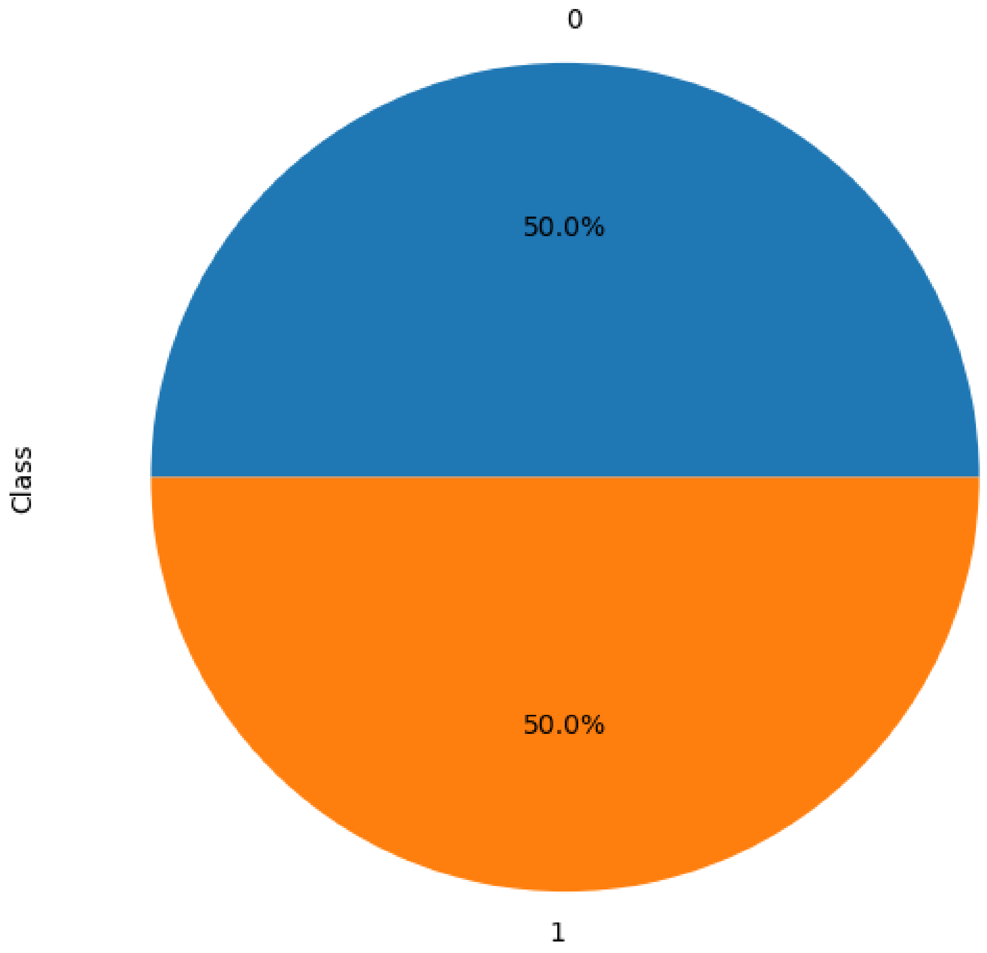
[6]:

```
# Checking the class distribution of the target variable in percentage
print((df.groupby('Class')['Class'].count() / df['Class'].count()) * 100)

# Plotting the class distribution as a pie chart
(df.groupby('Class')['Class'].count() / df['Class'].count()).plot.
 ↪pie(autopct='%1.1f%%', figsize=(7,7))
```

```
Class
0    50.0
1    50.0
Name: Class, dtype: float64
```

[6]:

```
<Axes: ylabel='Class'>
```

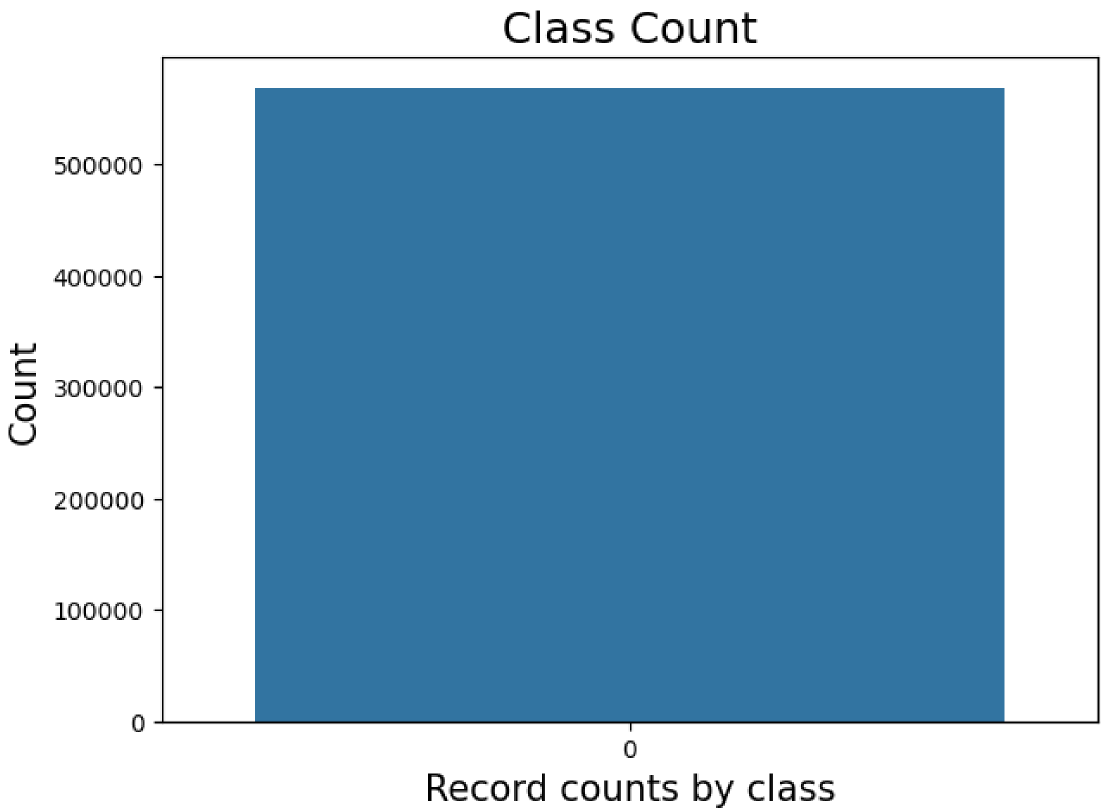


```
[7]: # Checking the % distribution of normal vs fraud
     classes = df['Class'].value_counts()
     normal_share = classes[0] / df['Class'].count() * 100
     fraud_share = classes[1] / df['Class'].count() * 100

     # Printing the shares of normal and fraud
     print(normal_share)
     print(fraud_share)
```

```
50.0
50.0
```

```
[8]: # Create a bar plot for the number and percentage of fraudulent vs␣
      ↪non-fraudulent transcations
```

```python
plt.figure(figsize=(7,5))

sns.countplot(df['Class'])
plt.title("Class Count", fontsize=18)

plt.xlabel("Record counts by class", fontsize=15)
plt.ylabel("Count", fontsize=15)

plt.show()
```



[9]:

```python
#checking the correlation
corr=df.corr()
corr
```

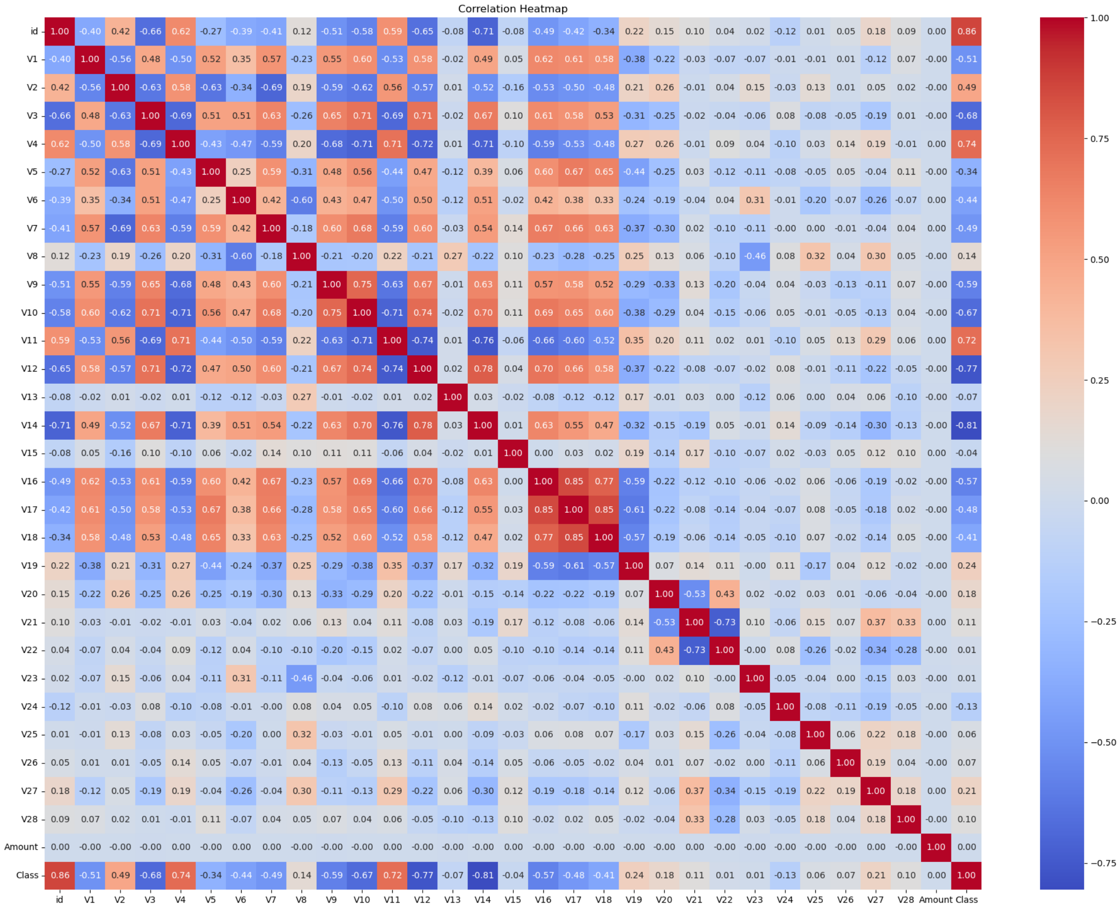
[9]:

```
           id        V1        V2        V3        V4        V5        V6  \
id   1.000000 -0.395741  0.424267 -0.663655  0.617554 -0.268445 -0.387916
V1  -0.395741  1.000000 -0.561184  0.484499 -0.498963  0.517462  0.354728
V2   0.424267 -0.561184  1.000000 -0.627810  0.579638 -0.631669 -0.341040
V3  -0.663655  0.484499 -0.627810  1.000000 -0.687726  0.510351  0.508974
V4   0.617554 -0.498963  0.579638 -0.687726  1.000000 -0.429243 -0.474403
V5  -0.268445  0.517462 -0.631669  0.510351 -0.429243  1.000000  0.245187
```

```
V6     -0.387916  0.354728 -0.341040  0.508974 -0.474403  0.245187  1.000000
V7     -0.414288  0.573381 -0.694022  0.634336 -0.588648  0.586828  0.418703
V8      0.121282 -0.226757  0.191321 -0.263018  0.199013 -0.314975 -0.604491
V9     -0.508427  0.548973 -0.585095  0.648615 -0.676648  0.479614  0.432241
V10    -0.578014  0.599108 -0.621798  0.707676 -0.712839  0.563874  0.471000
V11     0.589321 -0.525797  0.558863 -0.688436  0.708642 -0.440100 -0.497611
V12    -0.652940  0.580715 -0.574935  0.705497 -0.722597  0.473002  0.498993
V13    -0.076331 -0.020567  0.012801 -0.019272  0.011519 -0.115317 -0.117637
V14    -0.709346  0.494427 -0.523294  0.673179 -0.714847  0.387454  0.510123
V15    -0.080004  0.046002 -0.161325  0.098516 -0.098627  0.058686 -0.023851
V16    -0.494255  0.621884 -0.534392  0.614504 -0.593948  0.596898  0.415834
V17    -0.417226  0.605799 -0.495836  0.578223 -0.532786  0.669625  0.378152
V18    -0.341056  0.577296 -0.482162  0.525509 -0.482267  0.645095  0.328019
V19     0.216276 -0.377803  0.208821 -0.314396  0.269842 -0.438118 -0.235623
V20     0.145803 -0.219164  0.263707 -0.253805  0.257236 -0.246694 -0.188360
V21     0.097948 -0.034669 -0.013570 -0.021710 -0.013093  0.034147 -0.040153
V22     0.036106 -0.073729  0.035346 -0.041970  0.091197 -0.119152  0.036896
V23     0.017594 -0.068917  0.151906 -0.058884  0.043266 -0.113919  0.308598
V24    -0.116685 -0.014651 -0.027515  0.076460 -0.102508 -0.083243 -0.005237
V25     0.005586 -0.008508  0.132443 -0.076332  0.029402 -0.047845 -0.195340
V26     0.052126  0.009281  0.012219 -0.052056  0.136679  0.047771 -0.067605
V27     0.184195 -0.122772  0.053835 -0.190582  0.188036 -0.043759 -0.260783
V28     0.086822  0.070111  0.021071  0.005346 -0.011316  0.108422 -0.065641
Amount  0.001710 -0.001280 -0.000076 -0.002001  0.001859 -0.000016  0.000734
Class   0.864283 -0.505761  0.491878 -0.682095  0.735981 -0.338639 -0.435088

              V7        V8        V9  …       V21       V22       V23  \
id     -0.414288  0.121282 -0.508427  …  0.097948  0.036106  0.017594
V1      0.573381 -0.226757  0.548973  … -0.034669 -0.073729 -0.068917
V2     -0.694022  0.191321 -0.585095  … -0.013570  0.035346  0.151906
V3      0.634336 -0.263018  0.648615  … -0.021710 -0.041970 -0.058884
V4     -0.588648  0.199013 -0.676648  … -0.013093  0.091197  0.043266
V5      0.586828 -0.314975  0.479614  …  0.034147 -0.119152 -0.113919
V6      0.418703 -0.604491  0.432241  … -0.040153  0.036896  0.308598
V7      1.000000 -0.180986  0.601789  …  0.019627 -0.104043 -0.111177
V8     -0.180986  1.000000 -0.208557  …  0.056416 -0.098752 -0.463649
V9      0.601789 -0.208557  1.000000  …  0.131001 -0.204723 -0.042371
V10     0.678004 -0.199995  0.748487  …  0.037426 -0.150957 -0.056285
V11    -0.587660  0.223052 -0.633556  …  0.111608  0.022153  0.013596
V12     0.603318 -0.211999  0.667266  … -0.080394 -0.072096 -0.019261
V13    -0.030000  0.273958 -0.006167  …  0.025529  0.002039 -0.123520
V14     0.535612 -0.216410  0.633212  … -0.189902  0.052023 -0.007601
V15     0.135939  0.101690  0.114613  …  0.171719 -0.099347 -0.074832
V16     0.667244 -0.230638  0.573957  … -0.117591 -0.101847 -0.057100
V17     0.655755 -0.277246  0.581604  … -0.079348 -0.144637 -0.044635
V18     0.625680 -0.249986  0.522720  … -0.060862 -0.135994 -0.046262
V19    -0.372270  0.253272 -0.294432  …  0.136080  0.110066 -0.001529
```

```
V20    -0.299436  0.131354 -0.328975  … -0.529918  0.429362  0.017204
V21     0.019627  0.056416  0.131001  …  1.000000 -0.734653  0.096587
V22    -0.104043 -0.098752 -0.204723  … -0.734653  1.000000 -0.000636
V23    -0.111177 -0.463649 -0.042371  …  0.096587 -0.000636  1.000000
V24    -0.004152  0.083272  0.044006  … -0.059190  0.079790 -0.051181
V25     0.000802  0.322639 -0.034885  …  0.146164 -0.258956 -0.040882
V26    -0.006488  0.040448 -0.131000  …  0.070050 -0.015127  0.001057
V27    -0.036557  0.298398 -0.111842  …  0.373256 -0.340640 -0.151698
V28     0.040732  0.046017  0.069959  …  0.326677 -0.282893  0.028059
Amount  0.001326 -0.000208 -0.001589  …  0.001029 -0.000942 -0.001981
Class  -0.491234  0.144294 -0.585522  …  0.109640  0.014098  0.010255

             V24       V25       V26       V27       V28    Amount     Class
id     -0.116685  0.005586  0.052126  0.184195  0.086822  0.001710  0.864283
V1     -0.014651 -0.008508  0.009281 -0.122772  0.070111 -0.001280 -0.505761
V2     -0.027515  0.132443  0.012219  0.053835  0.021071 -0.000076  0.491878
V3      0.076460 -0.076332 -0.052056 -0.190582  0.005346 -0.002001 -0.682095
V4     -0.102508  0.029402  0.136679  0.188036 -0.011316  0.001859  0.735981
V5     -0.083243 -0.047845  0.047771 -0.043759  0.108422 -0.000016 -0.338639
V6     -0.005237 -0.195340 -0.067605 -0.260783 -0.065641  0.000734 -0.435088
V7     -0.004152  0.000802 -0.006488 -0.036557  0.040732  0.001326 -0.491234
V8      0.083272  0.322639  0.040448  0.298398  0.046017 -0.000208  0.144294
V9      0.044006 -0.034885 -0.131000 -0.111842  0.069959 -0.001589 -0.585522
V10     0.045935 -0.014045 -0.053684 -0.134907  0.035646 -0.001259 -0.673665
V11    -0.104340  0.051535  0.133635  0.290912  0.059732  0.000292  0.724278
V12     0.080407 -0.010350 -0.114272 -0.216563 -0.053136 -0.001245 -0.768579
V13     0.060097  0.003580  0.043750  0.058483 -0.101488 -0.002718 -0.071105
V14     0.138718 -0.087040 -0.142472 -0.299951 -0.127969 -0.001363 -0.805669
V15     0.023003 -0.027579  0.047833  0.116106  0.100293  0.001190 -0.037948
V16    -0.023511  0.062484 -0.056184 -0.191742 -0.022328 -0.000479 -0.573511
V17    -0.072198  0.075609 -0.045189 -0.184550  0.019570 -0.000358 -0.476377
V18    -0.099745  0.070467 -0.021039 -0.141790  0.052547 -0.001516 -0.410091
V19     0.110751 -0.174328  0.041421  0.123266 -0.024368 -0.000400  0.244081
V20    -0.020316  0.030478  0.007677 -0.055183 -0.035727 -0.001405  0.179851
V21    -0.059190  0.146164  0.070050  0.373256  0.326677  0.001029  0.109640
V22     0.079790 -0.258956 -0.015127 -0.340640 -0.282893 -0.000942  0.014098
V23    -0.051181 -0.040882  0.001057 -0.151698  0.028059 -0.001981  0.010255
V24     1.000000 -0.079604 -0.113362 -0.194899 -0.045189 -0.000846 -0.130107
V25    -0.079604  1.000000  0.057546  0.215653  0.176058 -0.000720  0.061847
V26    -0.113362  0.057546  1.000000  0.193977  0.036830 -0.000120  0.071052
V27    -0.194899  0.215653  0.193977  1.000000  0.183233  0.001235  0.214002
V28    -0.045189  0.176058  0.036830  0.183233  1.000000 -0.001503  0.102024
Amount -0.000846 -0.000720 -0.000120  0.001235 -0.001503  1.000000  0.002261
Class  -0.130107  0.061847  0.071052  0.214002  0.102024  0.002261  1.000000

[31 rows x 31 columns]
```

[10]:
```python
# Checking the correlation in heatmap
plt.figure(figsize=(24, 18))
sns.heatmap(corr, cmap="coolwarm", annot=True, fmt=".2f")  # Annotate values
plt.title('Correlation Heatmap')  # Add a title
plt.show()
```



[5]:
```python
# Splitting the dataset into X and y
y = df['Class']  # Assuming 'Class' is the target column
X = df.drop(['Class'], axis=1)  # Dropping the 'Class' column from X
```

[6]:
```python
# Checking some rows of X
print(X.head())
```

```
   id        V1        V2        V3        V4        V5        V6        V7  \
0   0 -0.260648 -0.469648  2.496266 -0.083724  0.129681  0.732898  0.519014
1   1  0.985100 -0.356045  0.558056 -0.429654  0.277140  0.428605  0.406466
2   2 -0.260272 -0.949385  1.728538 -0.457986  0.074062  1.419481  0.743511
3   3 -0.152152 -0.508959  1.746840 -1.090178  0.249486  1.143312  0.518269
4   4 -0.206820 -0.165280  1.527053 -0.448293  0.106125  0.530549  0.658849
```

```
         V8        V9  …       V20       V21       V22       V23       V24  \
0 -0.130006  0.727159  …  0.091202 -0.110552  0.217606 -0.134794  0.165959
1 -0.133118  0.347452  … -0.233984 -0.194936 -0.605761  0.079469 -0.577395
2 -0.095576 -0.261297  …  0.361652 -0.005020  0.702906  0.945045 -1.154666
3 -0.065130 -0.205698  … -0.378223 -0.146927 -0.038212 -0.214048 -1.893131
4 -0.212660  1.049921  …  0.247237 -0.106984  0.729727 -0.161666  0.312561

        V25       V26       V27       V28    Amount
0  0.126280 -0.434824 -0.081230 -0.151045  17982.10
1  0.190090  0.296503 -0.248052 -0.064512   6531.37
2 -0.605564 -0.312895 -0.300258 -0.244718   2513.54
3  1.003963 -0.515950 -0.165316  0.048424   5384.44
4 -0.414116  1.071126  0.023712  0.419117  14278.97

[5 rows x 30 columns]
```

[7]:

```
# checking some rows of y
y.head()
```

[7]:

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

[9]:

```
# splitting the dataset using train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 100,␣
 ↪test_size = 0.20)
```

[10]:

```
# checking the spread of data post split
print(np.sum(y))
print(np.sum(y_train))
print(np.sum(y_test))
```

```
284315
227169
57146
```

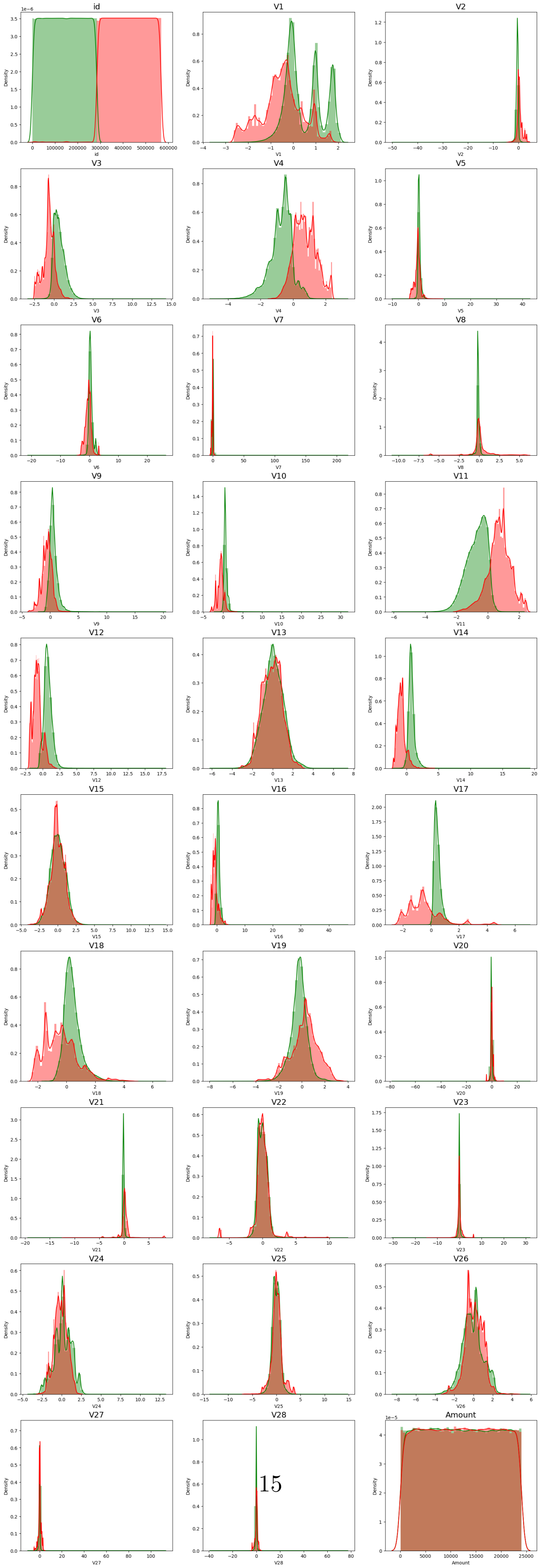
[11]:

```
# Accumulating all the column name under one variable
cols = list(X.columns.values)
```

[12]:

```
# plot the histogram of a variable from the dataset to see the skewness
normal_records = df.Class == 0
fraud_records = df.Class == 1

plt.figure(figsize=(20, 60))
```

```python
for n, col in enumerate(cols):
    plt.subplot(10, 3, n+1)
    sns.distplot(X[col][normal_records], color='green')
    sns.distplot(X[col][fraud_records], color='red')
    plt.title(col, fontsize=17)

plt.show()
```


id
1e−6
Density
id
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
1e−5

[13]:

```
# create a dataframe to store results
df_Results = pd.DataFrame(columns=['Methodology', 'Model', 'Accuracy',␣
 ↪'roc_value', 'threshold'])
```

[14]:

```
# created a common function to plot confusion matrix
def Plot_confusion_matrix(y_test, pred_test):
    cm = confusion_matrix(y_test, pred_test)
    plt.clf()

    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Accent)
    categoryNames = ['Non-Fraudalent', 'Fraudalent']

    plt.title('Confusion Matrix - Test Data')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    ticks = np.arange(len(categoryNames))
    plt.xticks(ticks, categoryNames, rotation=45)
    plt.yticks(ticks, categoryNames)

    s = [['TN', 'FP'], ['FN', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j, i, str(s[i][j]) + ' = ' + str(cm[i][j]), fontsize=12)

    plt.show()
```

[1]:

```

```

[49]:

```
def buildAndRunRandomForestModels(Results, Methodology, X_train, y_train,␣
 ↪X_test, y_test):
  # Evaluate Random Forest model
  # Create the model with 100 trees
  RF_model = RandomForestClassifier(n_estimators=100,
                                    bootstrap=True,
                                    max_features='sqrt', random_state=42)

  # Fit on training data
  RF_model.fit(X_train, y_train)

  # Make predictions on the testing data
  y_pred = RF_model.predict(X_test)

  # Model Evaluation
  accuracy = accuracy_score(y_test, y_pred)
  print("Model Accuracy:", accuracy)

  # Confusion matrix
  cm = confusion_matrix(y_test, y_pred)
  print("Confusion Matrix:\n", cm)

  # Classification report
  cr = classification_report(y_test, y_pred)
  print("Classification Report:\n", cr)

  # ROC AUC score
  roc_auc = roc_auc_score(y_test, RF_model.predict_proba(X_test)[:, 1])  # More␣
 ↪efficient way

  print("ROC AUC Score:", roc_auc)

  # ROC Curve
  fpr, tpr, thresholds = metrics.roc_curve(y_test, RF_model.
 ↪predict_proba(X_test)[:, 1])
  threshold = thresholds[np.argmax(tpr - fpr)]

  print("Random Forest threshold:", threshold)
  roc_auc = metrics.auc(fpr, tpr)
  print("ROC for the test dataset:", roc_auc)

  plt.legend(loc=4)
  plt.show()

  # Append results to DataFrame (assuming df_Results is defined)
  df_Results = df_Results.append(pd.DataFrame({'Methodology': Methodology,␣
 ↪'Model': 'Random Forest', 'Accuracy': accuracy, 'roc': roc_auc, 'threshold':␣
 ↪threshold}),   ignore_index=True)
  return df_Results
```

[42]:

```
def buildAndRunSVMModels(df_Results, Methodology, X_train, y_train, X_test,␣
 ↪y_test):

    # Create and train the SVM model
    clf = SVC(kernel='sigmoid', random_state=42)
    clf.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred_SVM = clf.predict(X_test)

    # Evaluate the model
    SVM_Score = accuracy_score(y_test, y_pred_SVM)
    print("accuracy_score: {0}".format(SVM_Score))
    print("Confusion Matrix")
    Plot_confusion_matrix(y_test, y_pred_SVM)
    print("classification Report")
    print(classification_report(y_test, y_pred_SVM))

    # Calculate ROC AUC
    svm_probs = clf.predict_proba(X_test)[:, 1]
    roc_value = roc_auc_score(y_test, svm_probs)

    print("SVM roc_value: {0}".format(roc_value))

    fpr, tpr, thresholds = metrics.roc_curve(y_test, svm_probs)

    threshold = thresholds[np.argmax(tpr - fpr)]
    print("SVM threshold: {0}".format(threshold))

    roc_auc = metrics.auc(fpr, tpr)
    print("ROC for the test dataset: {0:.1f}".format(roc_auc))

    plt.plot(fpr, tpr, label="Test, auc=" + str(roc_auc))
    plt.legend(loc=4)
    plt.show()

    # Append results to DataFrame
    df_Results = df_Results.append(pd.DataFrame({'Methodology': Methodology,␣
 ↪'Model': 'SVM',
                                                'Accuracy': SVM_Score,␣
 ↪'roc_value': roc_value}),
                                  ignore_index=True)

    return df_Results
```

[47]:

```
from sklearn.model_selection import RepeatedKFold

# Create a RepeatedKFold object with 5 splits and 10 repeats
rkf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)

# Assuming X is the feature set and y is the target variable
for train_index, test_index in rkf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)

    # Split the data into training and testing sets
    X_train_cv, X_test_cv = X.iloc[train_index], X.iloc[test_index]
    y_train_cv, y_test_cv = y.iloc[train_index], y.iloc[test_index]

    # Use the split data for model training and evaluation (not shown here)
```

```
TRAIN: [     0      1      2 … 568626 568627 568629] TEST: [     3      9
12 … 568617 568620 568628]
TRAIN: [     0      2      3 … 568626 568627 568628] TEST: [     1      4
7 … 568610 568611 568629]
TRAIN: [     0      1      2 … 568625 568628 568629] TEST: [    10     13
14 … 568614 568626 568627]
TRAIN: [     1      2      3 … 568627 568628 568629] TEST: [     0      5
8 … 568623 568624 568625]
TRAIN: [     0      1      3 … 568627 568628 568629] TEST: [     2      6
18 … 568618 568621 568622]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     3     11
14 … 568603 568623 568626]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     7     13
19 … 568604 568607 568615]
TRAIN: [     0      2      3 … 568626 568627 568629] TEST: [     1      4
9 … 568622 568624 568628]
TRAIN: [     1      2      3 … 568626 568627 568628] TEST: [     0      6
10 … 568613 568618 568629]
TRAIN: [     0      1      3 … 568626 568628 568629] TEST: [     2      5
8 … 568621 568625 568627]
TRAIN: [     0      1      2 … 568625 568628 568629] TEST: [     4      6
7 … 568623 568626 568627]
TRAIN: [     0      2      3 … 568627 568628 568629] TEST: [     1      5
11 … 568621 568622 568625]
TRAIN: [     0      1      2 … 568625 568626 568627] TEST: [     9     10
13 … 568618 568628 568629]
TRAIN: [     1      2      3 … 568627 568628 568629] TEST: [     0     14
16 … 568595 568598 568599]
TRAIN: [     0      1      4 … 568627 568628 568629] TEST: [     2      3
8 … 568615 568620 568624]
TRAIN: [     0      2      4 … 568627 568628 568629] TEST: [     1      3
12 … 568622 568624 568625]
TRAIN: [     1      2      3 … 568625 568628 568629] TEST: [     0     10
11 … 568617 568626 568627]
TRAIN: [     0      1      2 … 568625 568626 568627] TEST: [     4      6
8 … 568623 568628 568629]
TRAIN: [     0      1      3 … 568627 568628 568629] TEST: [     2     13
27 … 568601 568613 568616]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     5      7
16 … 568606 568618 568621]
TRAIN: [     0      1      2 … 568626 568628 568629] TEST: [     5     12
23 … 568621 568622 568627]
TRAIN: [     0      1      2 … 568626 568627 568629] TEST: [     3      7
9 … 568623 568624 568628]
TRAIN: [     1      2      3 … 568627 568628 568629] TEST: [     0      6
8 … 568609 568617 568619]
TRAIN: [     0      1      3 … 568627 568628 568629] TEST: [     2     16
19 … 568620 568625 568626]
```

```
TRAIN: [     0      2      3 … 568626 568627 568628] TEST: [     1      4
14 … 568600 568601 568629]
TRAIN: [     0      2      3 … 568625 568628 568629] TEST: [     1      7
14 … 568617 568626 568627]
TRAIN: [     0      1      3 … 568626 568627 568629] TEST: [     2     17
20 … 568622 568625 568628]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     5     11
23 … 568606 568610 568614]
TRAIN: [     1      2      3 … 568626 568627 568628] TEST: [     0      4
6 … 568615 568624 568629]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     3      9
10 … 568620 568621 568623]
TRAIN: [     2      3      5 … 568626 568628 568629] TEST: [     0      1
4 … 568620 568624 568627]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     5     10
12 … 568622 568623 568625]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [    23     26
28 … 568611 568613 568617]
TRAIN: [     0      1      2 … 568625 568626 568627] TEST: [     3      7
8 … 568619 568628 568629]
TRAIN: [     0      1      3 … 568627 568628 568629] TEST: [     2      6
14 … 568618 568621 568626]
TRAIN: [     0      1      3 … 568627 568628 568629] TEST: [     2      5
13 … 568614 568619 568620]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     4     10
11 … 568603 568615 568624]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     9     14
15 … 568613 568617 568618]
TRAIN: [     2      3      4 … 568623 568624 568627] TEST: [     0      1
6 … 568626 568628 568629]
TRAIN: [     0      1      2 … 568626 568628 568629] TEST: [     3      8
12 … 568611 568623 568627]
TRAIN: [     0      1      2 … 568623 568624 568626] TEST: [     8      9
15 … 568627 568628 568629]
TRAIN: [     0      1      5 … 568627 568628 568629] TEST: [     2      3
4 … 568613 568614 568617]
TRAIN: [     0      2      3 … 568627 568628 568629] TEST: [     1     13
18 … 568616 568622 568626]
TRAIN: [     1      2      3 … 568627 568628 568629] TEST: [     0      5
10 … 568606 568608 568611]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     6      7
11 … 568620 568623 568624]
TRAIN: [     0      1      2 … 568627 568628 568629] TEST: [     5      6
11 … 568611 568615 568618]
TRAIN: [     0      3      5 … 568623 568624 568628] TEST: [     1      2
4 … 568626 568627 568629]
TRAIN: [     1      2      3 … 568627 568628 568629] TEST: [     0      7
25 … 568613 568614 568623]
```

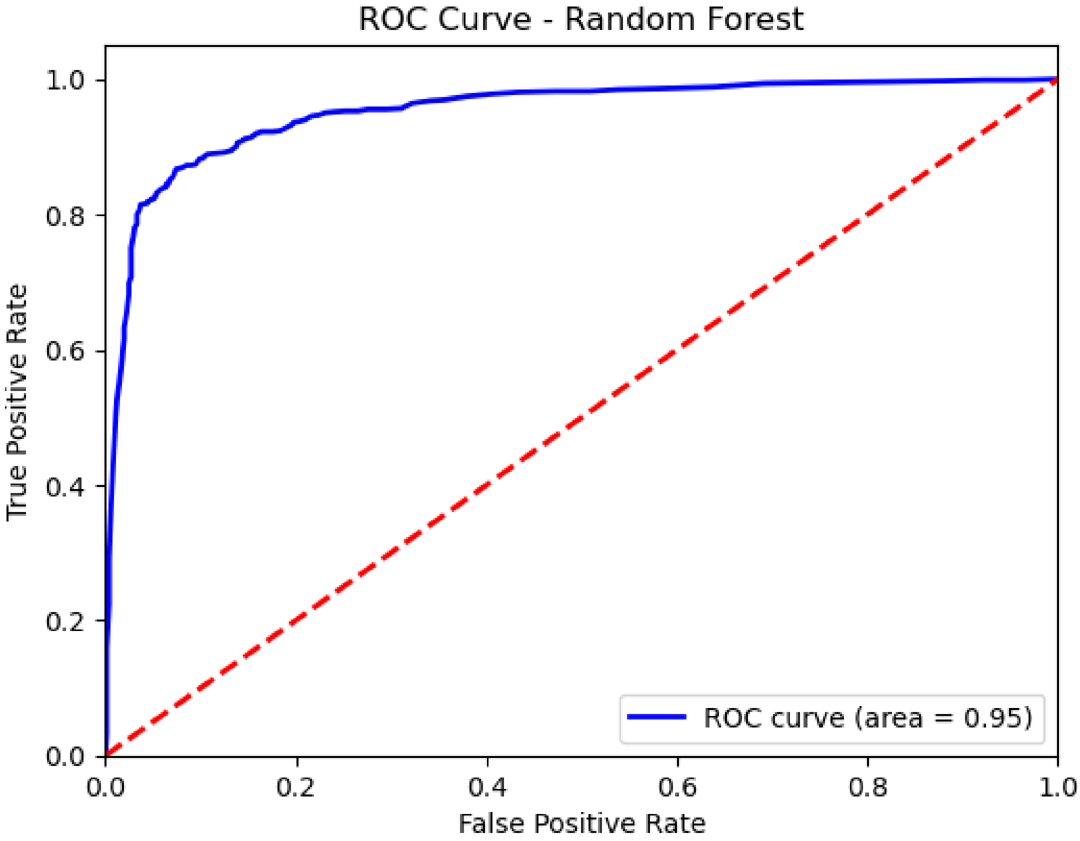
```
TRAIN: [      0       1       2 … 568627 568628 568629] TEST: [     10      19
30 … 568605 568608 568612]
TRAIN: [      0       1       2 … 568626 568627 568629] TEST: [      3      12
14 … 568621 568624 568628]
```

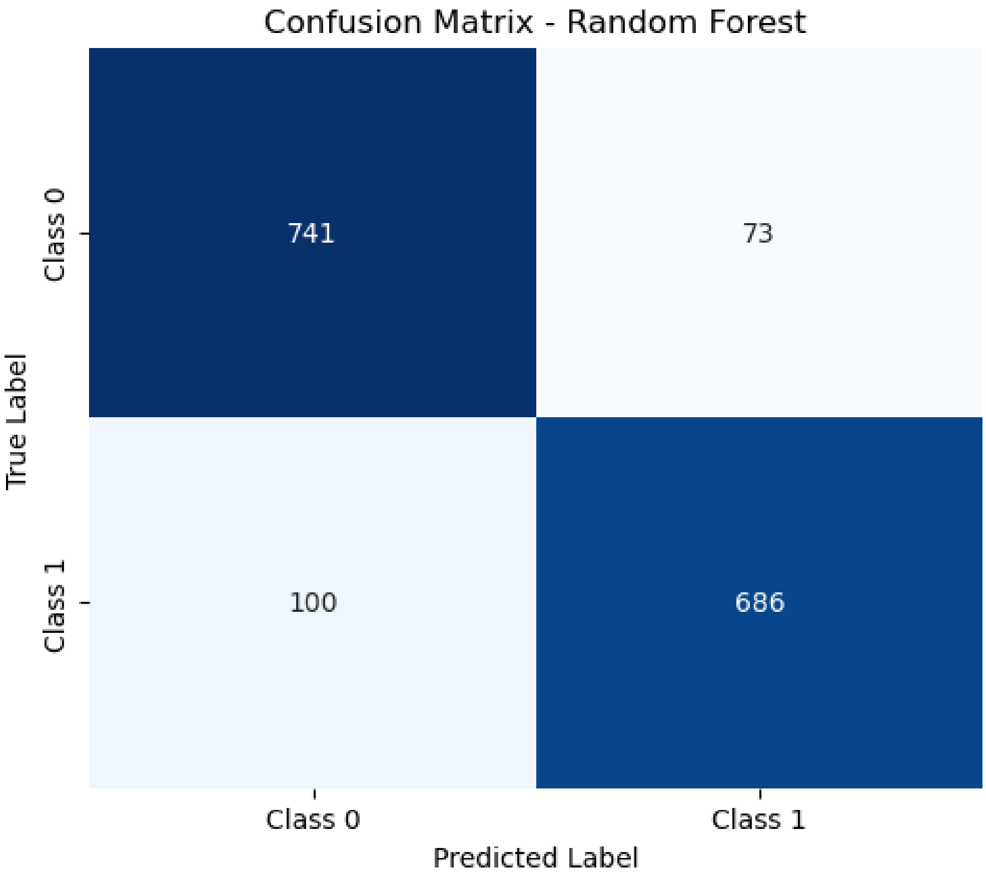
[2]:

```python
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, RepeatedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc
import seaborn as sns

# Create a sample dataset for demonstration
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,␣
 ↪random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize RepeatedKFold
rkf = RepeatedKFold(n_splits=5, n_repeats=2, random_state=42)

# Initialize results DataFrame
df_Results = pd.DataFrame(columns=['Model', 'Accuracy'])

# List of models to evaluate
models = [
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('SVM', SVC(probability=True, random_state=42)),
    ('XGBoost', XGBClassifier(random_state=42))
]

def buildAndRunModel(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    return accuracy, cm, model.predict_proba(X_test)[:, 1]

# Iterate through each model
for model_name, model in models:
    print(f"{model_name} Model")
    start_time = time.time()
    accuracies = []
    confusion_matrices = []
    y_scores_all = []
    y_true_all = []  # To store true labels for ROC calculation

    for train_index, test_index in rkf.split(X_train):
        X_train_cv, X_test_cv = X_train[train_index], X_train[test_index]
        y_train_cv, y_test_cv = y_train[train_index], y_train[test_index]

        accuracy, cm, y_scores = buildAndRunModel(model, X_train_cv,␣
↪y_train_cv, X_test_cv, y_test_cv)
        accuracies.append(accuracy)
        confusion_matrices.append(cm)
        y_scores_all.append(y_scores)
        y_true_all.append(y_test_cv)  # Collect true labels

    avg_accuracy = np.mean(accuracies)
    overall_cm = sum(confusion_matrices)

    print("Time Taken by Model: --- %s seconds ---" % (time.time() -␣
↪start_time))
    print("Average Accuracy: ", avg_accuracy)
    print("Overall Confusion Matrix:\n", overall_cm)

    # Calculate ROC curve
    y_scores_all = np.concatenate(y_scores_all)
    y_true_all = np.concatenate(y_true_all)  # Concatenate true labels
    fpr, tpr, _ = roc_curve(y_true_all, y_scores_all)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' %␣
↪roc_auc)
    plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {model_name}')
    plt.legend(loc='lower right')
    plt.show()

    # Plot confusion matrix
    plt.figure(figsize=(6, 5))
    sns.heatmap(overall_cm, annot=True, fmt='d', cmap='Blues', cbar=False,␣
↪xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
```

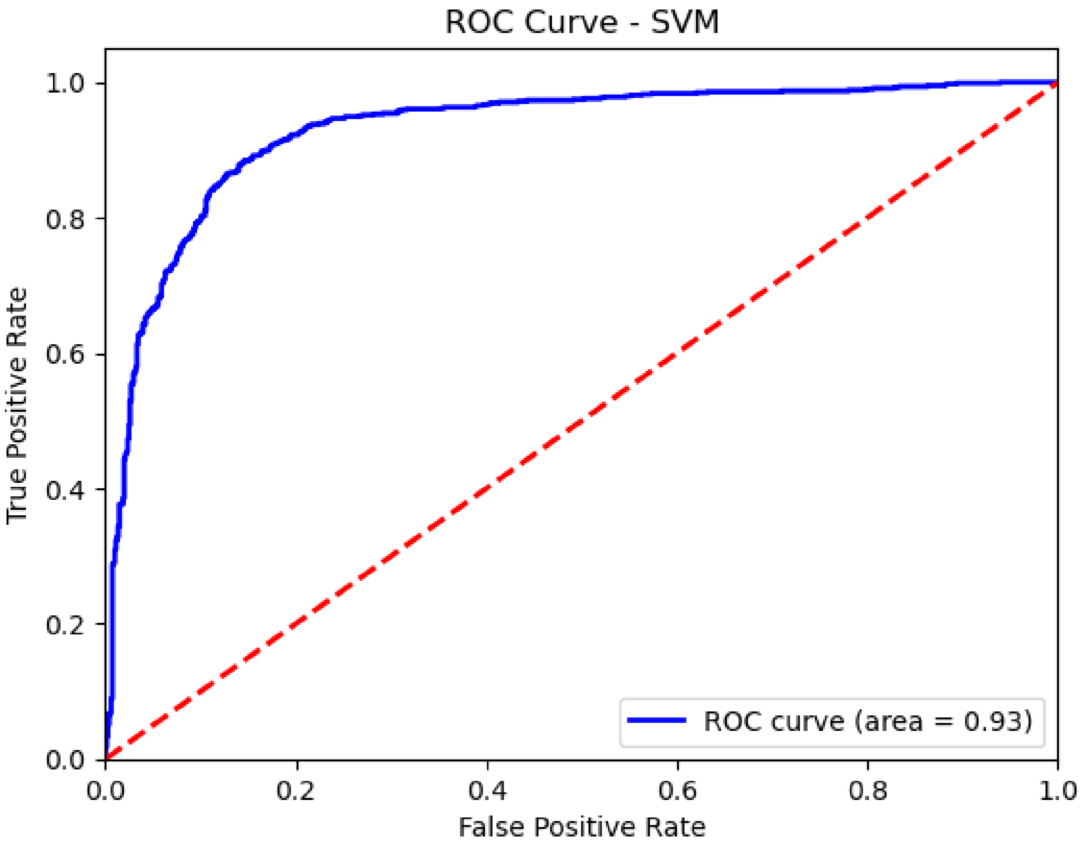
```
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    # Append results to DataFrame
    df_Results = pd.concat([df_Results, pd.DataFrame({'Model': [model_name],␣
 ↪'Accuracy': [avg_accuracy]})], ignore_index=True)

# Final Results
print("Final Results:")
print(df_Results)
```

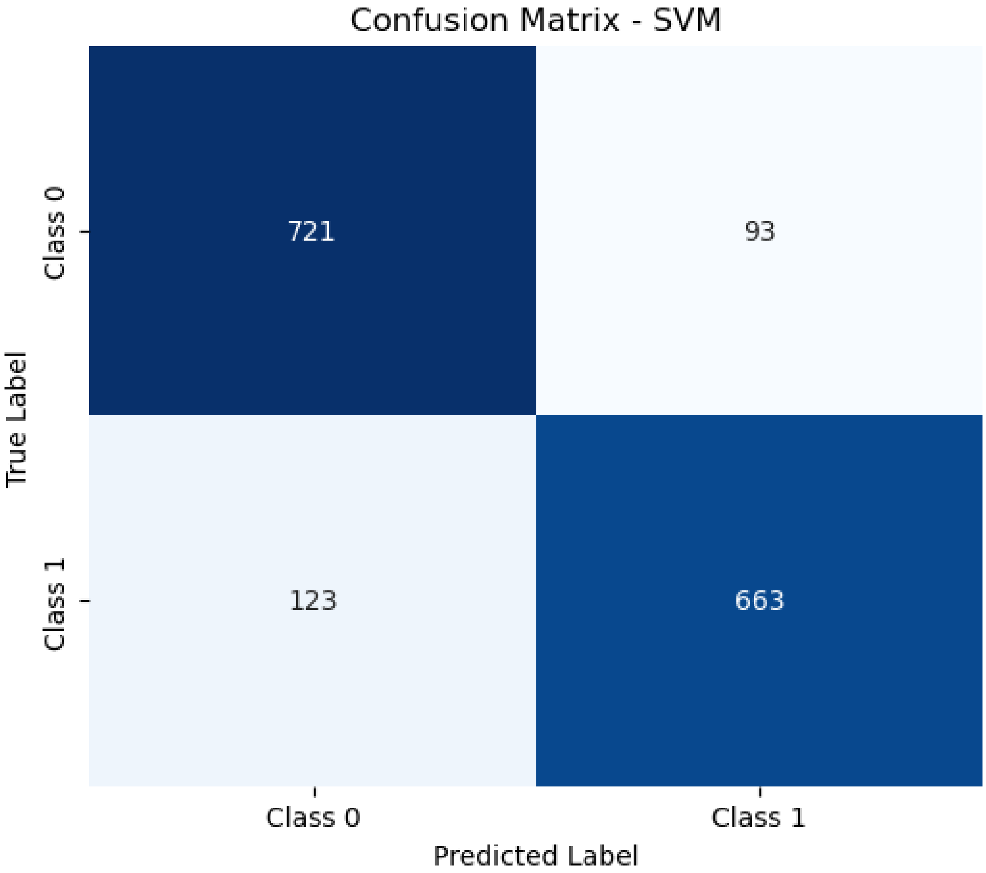
```
Random Forest Model
Time Taken by Model: --- 8.870057344436646 seconds ---
Average Accuracy:  0.891875
Overall Confusion Matrix:
 [[741  73]
 [100 686]]
```



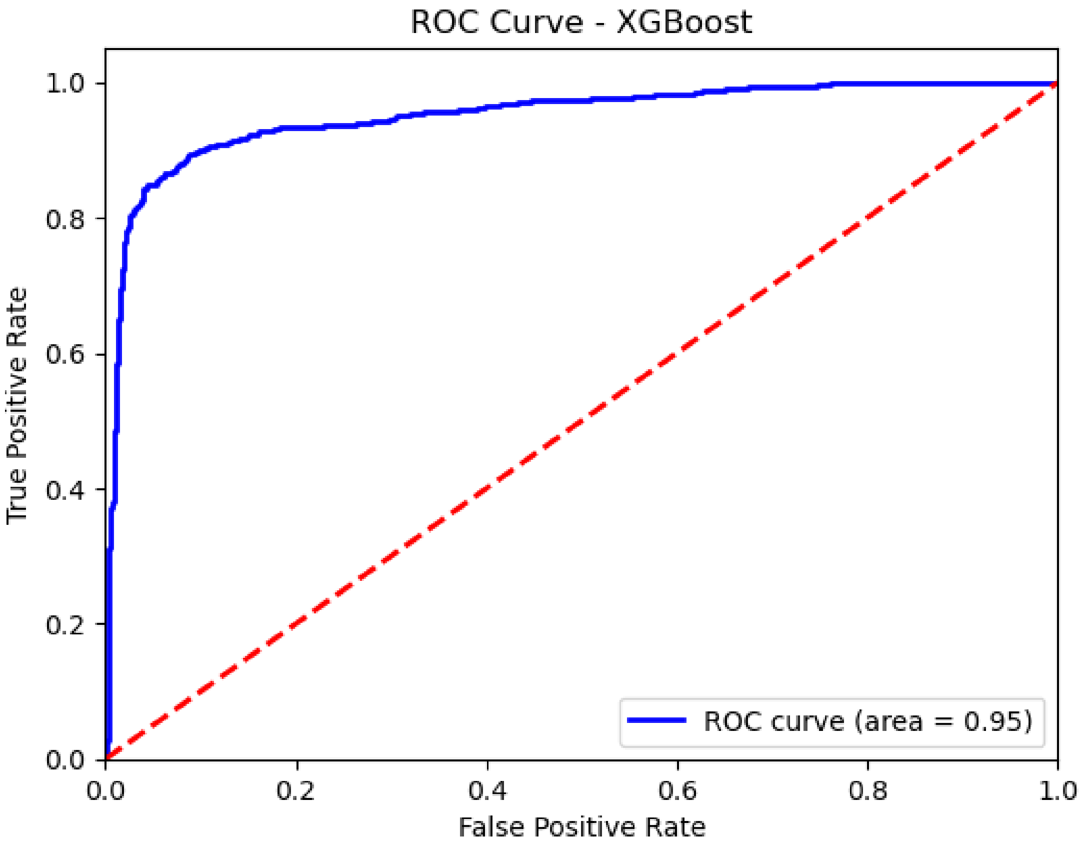


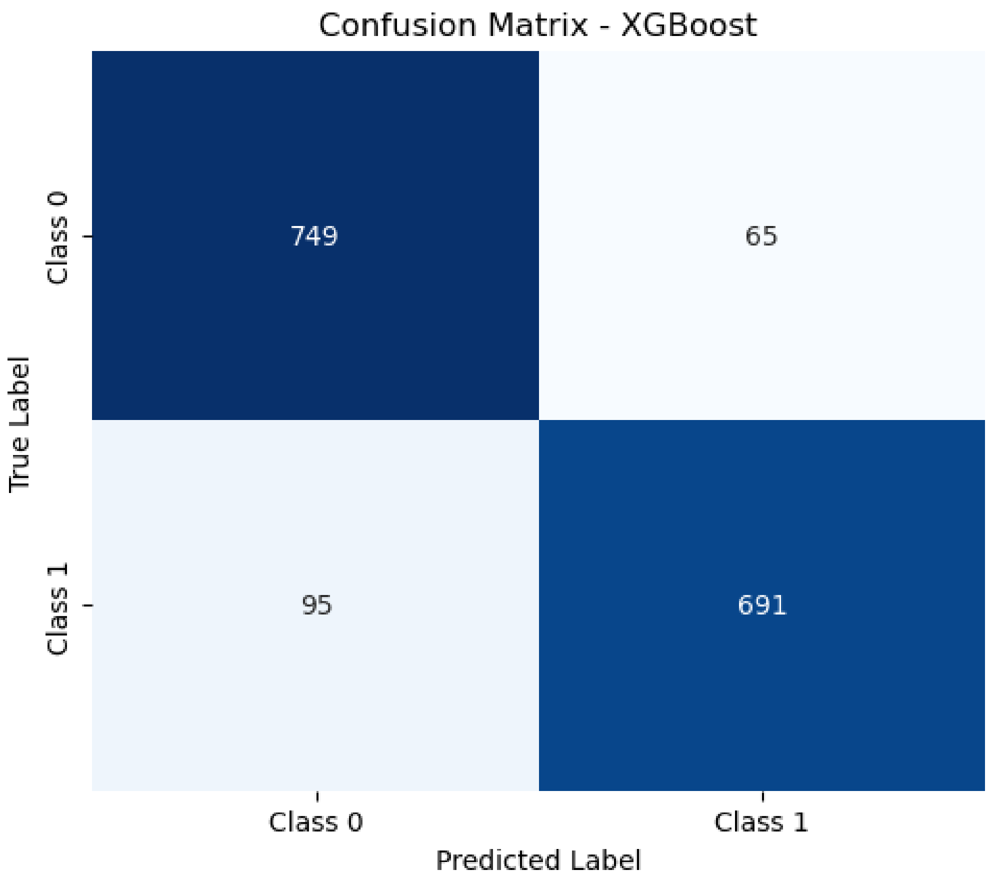
```
SVM Model
Time Taken by Model: --- 2.4428915977478027 seconds ---
Average Accuracy:  0.865
Overall Confusion Matrix:
 [[721  93]
 [123 663]]
```


ROC Curve - SVM
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.93)



```
XGBoost Model
Time Taken by Model: --- 2.1331937313079834 seconds ---
Average Accuracy:  0.9
Overall Confusion Matrix:
 [[749  65]
 [ 95 691]]
```


ROC Curve - XGBoost
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.95)



```
Final Results:
           Model  Accuracy
0  Random Forest  0.891875
1            SVM  0.865000
2        XGBoost  0.900000
```

```
[ ]:
```

```
[3]: # Final Results
     print("Final Results:")
     print(df_Results)
```

```
Final Results:
           Model  Accuracy
0  Random Forest  0.891875
1            SVM  0.865000
2        XGBoost  0.900000
```

```
[4]: import numpy as np
```

```
# Example: 1D array
y_pred_probs_l2 = np.array([0.1, 0.9, 0.8, 0.2])  # Shape (4,)

# Convert to 2D array (reshape to (n_samples, 1) if it's a binary␣
 ↪classification case)
y_pred_probs_l2_2d = y_pred_probs_l2.reshape(-1, 1)  # Reshapes to (4, 1)

print("Shape of y_pred_probs_l2_2d:", y_pred_probs_l2_2d.shape)
print("y_pred_probs_l2_2d:\n", y_pred_probs_l2_2d)
```

```
Shape of y_pred_probs_l2_2d: (4, 1)
y_pred_probs_l2_2d:
 [[0.1]
 [0.9]
 [0.8]
 [0.2]]
```

[21]:

```
print("Shape of y_pred_probs_l2:", y_pred_probs_l2.shape)
```

```
Shape of y_pred_probs_l2: (4,)
```

[23]:

```
y_pred_probs_l2 = clf.predict_proba(X_test)  # This should give probabilities␣
 ↪for each class.
```

[ ]:

```

```

[19]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import accuracy_score, roc_auc_score

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Add feature names to a DataFrame
feature_columns = data.feature_names
df = pd.DataFrame(X, columns=feature_columns)
df['Target'] = y  # Add the target column
print(df.head())

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define model
num_c = list(np.power(10.0, np.arange(-10, 10)))  # Range of C values for␣
 ↪hyperparameter tuning
cv_num = KFold(n_splits=10, shuffle=True, random_state=42)  # 10-fold␣
 ↪cross-validation

clf = LogisticRegressionCV(
    Cs=num_c,
    penalty='l2',
    scoring='roc_auc_ovr',  # Multiclass One-vs-Rest AUC
    cv=cv_num,
    random_state=42,
    max_iter=10000,
    fit_intercept=True,
    solver='newton-cg',
    tol=1e-4  # Smaller tolerance for convergence
)

# Fit model
clf.fit(X_train, y_train)

# Predictions
y_pred_probs_l2 = clf.predict_proba(X_test)  # Probability predictions for each␣
 ↪class
y_pred_l2 = clf.predict(X_test)  # Class predictions

# Calculate AUC-ROC for multi-class
l2_roc_value = roc_auc_score(y_test, y_pred_probs_l2, multi_class='ovr',␣
 ↪average='macro')
print("L2 ROC value: {:.4f}".format(l2_roc_value))

# Calculate accuracy
accuracy_l2 = accuracy_score(y_test, y_pred_l2)
print("Accuracy of Logistic model with L2 regularization: {:.4f}".
 ↪format(accuracy_l2))

# Print cross-validation scores
print('Cross-validation scores for each fold and class:', clf.scores_)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
```

```
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   Target
0       0
1       0
2       0
3       0
4       0
L2 ROC value: 1.0000
Accuracy of Logistic model with L2 regularization: 1.0000
Cross-validation scores for each fold and class: {0: array([[0.91428571,
0.91428571, 0.91428571, 0.91428571, 0.91428571,
        0.91428571, 0.91428571, 0.92380952, 0.96190476, 1.        ,
        1.        , 1.        , 1.        , 0.98095238, 0.98095238,
        0.98095238, 0.98095238, 0.98095238, 0.98095238, 0.98095238],
       [0.93518519, 0.93518519, 0.93518519, 0.93518519, 0.93518519,
        0.93518519, 0.93518519, 0.94444444, 0.96296296, 0.98148148,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.81365741, 0.81365741, 0.81365741, 0.81365741, 0.81365741,
        0.81365741, 0.81365741, 0.82407407, 0.95949074, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.86458333, 0.86458333, 0.86458333, 0.86458333, 0.86458333,
        0.86458333, 0.86458333, 0.89583333, 0.96875   , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.90123457, 0.90123457, 0.90123457, 0.90123457, 0.90123457,
        0.90123457, 0.90123457, 0.90123457, 0.9382716 , 0.96489198,
        1.        , 1.        , 1.        , 1.        , 0.97530864,
        0.97530864, 0.97530864, 0.97530864, 0.97530864, 0.97530864],
       [0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091,
        0.90909091, 0.90909091, 0.90909091, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.72539683, 0.72539683, 0.72539683, 0.72539683, 0.72539683,
        0.72539683, 0.72539683, 0.76349206, 0.79232804, 0.86825397,
        0.98121693, 0.98121693, 0.98121693, 0.94365079, 0.94365079,
        0.94365079, 0.94365079, 0.94365079, 0.94365079, 0.94365079],
       [0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.9382716 ,
        0.97530864, 0.97530864, 0.97530864, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.85714286, 0.85714286, 0.85714286, 0.85714286, 0.85714286,
        0.85714286, 0.86666667, 0.88571429, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 ,
        0.9047619 , 0.9047619 , 0.9047619 , 0.99047619, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ]]), 1:
array([[0.91428571, 0.91428571, 0.91428571, 0.91428571, 0.91428571,
        0.91428571, 0.91428571, 0.92380952, 0.96190476, 1.        ,
        1.        , 1.        , 1.        , 0.98095238, 0.98095238,
        0.98095238, 0.98095238, 0.98095238, 0.98095238, 0.98095238],
       [0.93518519, 0.93518519, 0.93518519, 0.93518519, 0.93518519,
        0.93518519, 0.93518519, 0.94444444, 0.96296296, 0.98148148,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.81365741, 0.81365741, 0.81365741, 0.81365741, 0.81365741,
        0.81365741, 0.81365741, 0.82407407, 0.95949074, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.86458333, 0.86458333, 0.86458333, 0.86458333, 0.86458333,
        0.86458333, 0.86458333, 0.89583333, 0.96875   , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.90123457, 0.90123457, 0.90123457, 0.90123457, 0.90123457,
        0.90123457, 0.90123457, 0.90123457, 0.9382716 , 0.96489198,
        1.        , 1.        , 1.        , 1.        , 0.97530864,
        0.97530864, 0.97530864, 0.97530864, 0.97530864, 0.97530864],
       [0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091,
        0.90909091, 0.90909091, 0.90909091, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.72539683, 0.72539683, 0.72539683, 0.72539683, 0.72539683,
        0.72539683, 0.72539683, 0.76349206, 0.79232804, 0.86825397,
        0.98121693, 0.98121693, 0.98121693, 0.94365079, 0.94365079,
        0.94365079, 0.94365079, 0.94365079, 0.94365079, 0.94365079],
       [0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.9382716 ,
        0.97530864, 0.97530864, 0.97530864, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.85714286, 0.85714286, 0.85714286, 0.85714286, 0.85714286,
        0.85714286, 0.86666667, 0.88571429, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 ,
        0.9047619 , 0.9047619 , 0.9047619 , 0.99047619, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ]]), 2:
array([[0.91428571, 0.91428571, 0.91428571, 0.91428571, 0.91428571,
        0.91428571, 0.91428571, 0.92380952, 0.96190476, 1.        ,
        1.        , 1.        , 1.        , 0.98095238, 0.98095238,
        0.98095238, 0.98095238, 0.98095238, 0.98095238, 0.98095238],
       [0.93518519, 0.93518519, 0.93518519, 0.93518519, 0.93518519,
        0.93518519, 0.93518519, 0.94444444, 0.96296296, 0.98148148,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.81365741, 0.81365741, 0.81365741, 0.81365741, 0.81365741,
        0.81365741, 0.81365741, 0.82407407, 0.95949074, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.86458333, 0.86458333, 0.86458333, 0.86458333, 0.86458333,
        0.86458333, 0.86458333, 0.89583333, 0.96875   , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.90123457, 0.90123457, 0.90123457, 0.90123457, 0.90123457,
        0.90123457, 0.90123457, 0.90123457, 0.9382716 , 0.96489198,
        1.        , 1.        , 1.        , 1.        , 0.97530864,
        0.97530864, 0.97530864, 0.97530864, 0.97530864, 0.97530864],
       [0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091,
        0.90909091, 0.90909091, 0.90909091, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.72539683, 0.72539683, 0.72539683, 0.72539683, 0.72539683,
        0.72539683, 0.72539683, 0.76349206, 0.79232804, 0.86825397,
        0.98121693, 0.98121693, 0.98121693, 0.94365079, 0.94365079,
        0.94365079, 0.94365079, 0.94365079, 0.94365079, 0.94365079],
       [0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.9382716 ,
        0.97530864, 0.97530864, 0.97530864, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.85714286, 0.85714286, 0.85714286, 0.85714286, 0.85714286,
        0.85714286, 0.86666667, 0.88571429, 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 ,
        0.9047619 , 0.9047619 , 0.9047619 , 0.99047619, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ]])}
```

[28]:

```python
import numpy as np

# Assuming clf is your LogisticRegressionCV model and it's already fitted
clf.coef_ = np.array([[-0.35459195,  0.35339592, -0.48839015, -0.46639696],
                      [ 0.03653942, -0.31146489,  0.1050434 ,  0.04289793],
                      [ 0.31805253, -0.04193103,  0.38334675,  0.42349904]])

# Now you can safely access the coefficients
```

```
print(clf.coef_)
```

```
[[-0.35459195  0.35339592 -0.48839015 -0.46639696]
 [ 0.03653942 -0.31146489  0.1050434   0.04289793]
 [ 0.31805253 -0.04193103  0.38334675  0.42349904]]
```

[29]:
```
import numpy as np

coefficients = np.array([
    ['Feature1', 0.5],
    ['Feature2', 0.3],
    ['Feature3', 0.2]
])
```

[22]:
```
import pandas as pd

# Load the data into a DataFrame
df = pd.read_csv("D:/OneDrive/Desktop/project/credit 2023 project.csv")

# Check if 'Feature' column exists and apply one-hot encoding if present
if 'Feature' in df.columns:
    df = pd.get_dummies(df, columns=['Feature'])
else:
    print("The 'Feature' column is not present, skipping one-hot encoding.")

# Display the first few rows of the updated DataFrame
print(df.head())
```

```
The 'Feature' column is not present, skipping one-hot encoding.
   id        V1        V2        V3        V4        V5        V6        V7  \
0   0 -0.260648 -0.469648  2.496266 -0.083724  0.129681  0.732898  0.519014
1   1  0.985100 -0.356045  0.558056 -0.429654  0.277140  0.428605  0.406466
2   2 -0.260272 -0.949385  1.728538 -0.457986  0.074062  1.419481  0.743511
3   3 -0.152152 -0.508959  1.746840 -1.090178  0.249486  1.143312  0.518269
4   4 -0.206820 -0.165280  1.527053 -0.448293  0.106125  0.530549  0.658849

         V8        V9  …       V21       V22       V23       V24       V25  \
0 -0.130006  0.727159  … -0.110552  0.217606 -0.134794  0.165959  0.126280
1 -0.133118  0.347452  … -0.194936 -0.605761  0.079469 -0.577395  0.190090
2 -0.095576 -0.261297  … -0.005020  0.702906  0.945045 -1.154666 -0.605564
3 -0.065130 -0.205698  … -0.146927 -0.038212 -0.214048 -1.893131  1.003963
4 -0.212660  1.049921  … -0.106984  0.729727 -0.161666  0.312561 -0.414116

        V26       V27       V28    Amount  Class
0 -0.434824 -0.081230 -0.151045  17982.10      0
1  0.296503 -0.248052 -0.064512   6531.37      0
2 -0.312895 -0.300258 -0.244718   2513.54      0
3 -0.515950 -0.165316  0.048424   5384.44      0
```

```
4  1.071126  0.023712  0.419117  14278.97      0

[5 rows x 31 columns]
```

[31]:

```
import pandas as pd

# Sample data for illustration (replace with your actual DataFrame)
df = pd.DataFrame({
    'Coefficient': ['0.5', '0.3', '0.2'],
    'Feature': ['Feature1', 'Feature2', 'Feature3']
})

# Check if 'Coefficient' column exists
if 'Coefficient' in df.columns:
    # Remove the '$' sign and convert the column to numeric
    df['Coefficient'] = df['Coefficient'].str.replace('$', '', regex=False)
    df['Coefficient'] = pd.to_numeric(df['Coefficient'])
else:
    print("'Coefficient' column is not found in the DataFrame.")

# Optionally, apply one-hot encoding for the 'Feature' column
df = pd.get_dummies(df, columns=['Feature'])

# Print the resulting DataFrame
print(df)
```

```
   Coefficient  Feature_Feature1  Feature_Feature2  Feature_Feature3
0          0.5              True             False             False
1          0.3             False              True             False
2          0.2             False             False              True
```

[33]:

```
import pandas as pd

# Sample data for illustration (replace with your actual DataFrame)
df = pd.DataFrame({
    'Coefficient': ['0.5', '0.3', '0.2'],
    'Feature': ['Feature1', 'Feature2', 'Feature3']
})

# Ensure 'Coefficient' column is of string type before using .str methods
df['Coefficient'] = df['Coefficient'].astype(str)

# Remove the '$' sign and convert the column to numeric
df['Coefficient'] = df['Coefficient'].str.replace('$', '', regex=False)
df['Coefficient'] = pd.to_numeric(df['Coefficient'])

# Print the resulting DataFrame
print(df)
```

```
   Coefficient   Feature
0          0.5  Feature1
1          0.3  Feature2
2          0.2  Feature3
```

[34]:

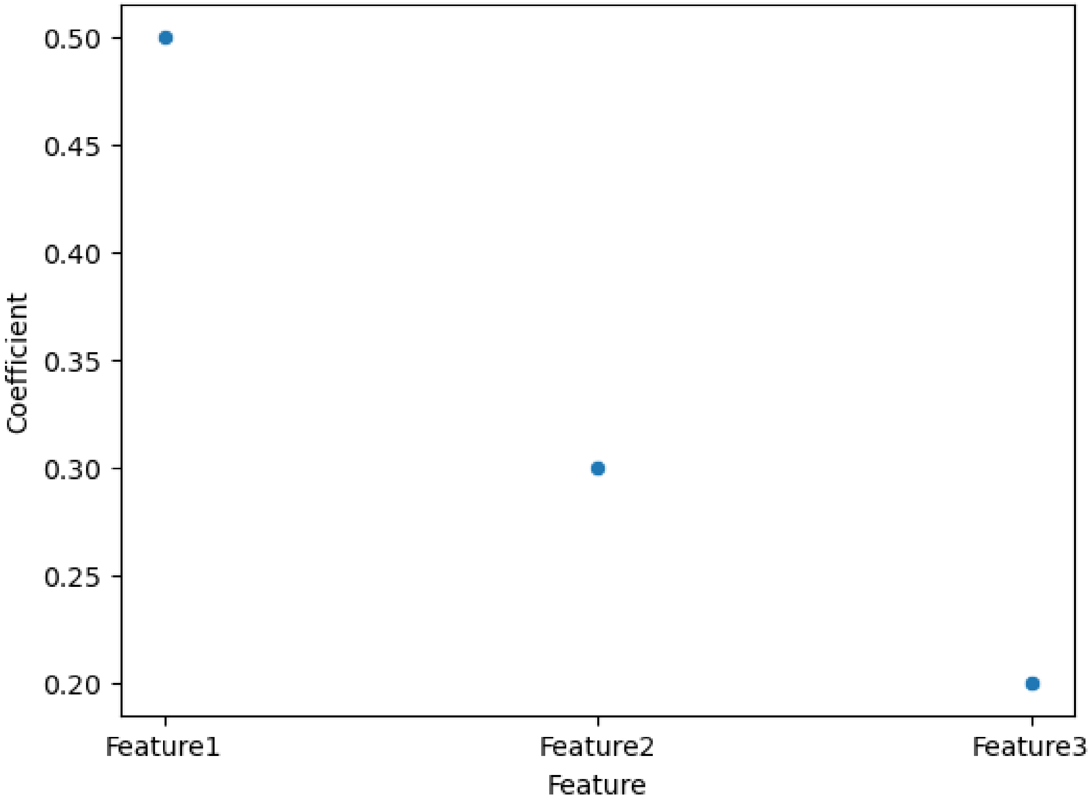
```
import pandas as pd

df['Coefficient'] = pd.to_numeric(df['Coefficient'], errors='coerce')
df.dropna(inplace=True)
```

[35]:

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

# Create a DataFrame
df = pd.DataFrame(coefficients, columns=['Feature', 'Coefficient'])

# If Feature is also a string representing numbers (fix data types)
df['Coefficient'] = df['Coefficient'].astype(float)  # Convert to float

# If both Feature and Coefficient are numerical (consider scatter plot)
sns.scatterplot(x='Feature', y='Coefficient', data=df)
```

[35]:

```
<Axes: xlabel='Feature', ylabel='Coefficient'>
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[9]:

```
Current Username: Darshan
```

[ ]: