In [1]:

```
## main libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.ticker as mticker

# !pip install squarify
import squarify as sq

import scipy.stats as stats
from scipy.cluster.hierarchy import linkage, dendrogram
import statsmodels.api as sm
import statsmodels.formula.api as smf
import datetime as dt
from datetime import datetime
# from pyclustertend import hopkins

## pre-processing
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.decomposition import PCA
from sklearn.dummy import DummyClassifier
from sklearn.impute import SimpleImputer, KNNImputer

## feature Selection
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif, f_regr

## scaling
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler

## regression/prediction
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet, LogisticR
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, ExtraTr
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor

## ann
from sklearn.neural_network import MLPRegressor

## classification
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoost
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
```

```
from xgboost import XGBClassifier, plot_importance

## metrics
from sklearn.metrics import confusion_matrix, r2_score, mean_absolute_error, mean_squa
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import make_scorer, precision_score, precision_recall_curve, prec
from sklearn.metrics import roc_auc_score, roc_curve, f1_score, accuracy_score, recall
from sklearn.metrics import silhouette_samples,silhouette_score, average_precision_scc
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics import auc

## model selection
from sklearn import model_selection
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold, KFold, cross_val_predict,
from sklearn.model_selection import StratifiedKFold, GridSearchCV, cross_val_score, cr

## MLearning
from sklearn.pipeline import make_pipeline, Pipeline
import optuna
from sklearn.naive_bayes import GaussianNB

## clevers
# !pip install -U pandas-profiling --user
import ydata_profiling
from ydata_profiling.report.presentation.flavours.html.templates import create_html_as

import ipywidgets
from ipywidgets import interact
import missingno as msno
# !pip install wordcloud
from wordcloud import WordCloud

# !pip install termcolor
import colorama
from colorama import Fore, Style  # makes strings colored
from termcolor import colored
from termcolor import cprint
# grey red green yellow blue magenta cyan white (on_grey ..)
# bold dark underline blink reverse concealed
# cprint("Have a first look to:","blue","on_grey", attrs=['bold'])

## plotly and cufflinks
import plotly
import plotly.express as px
import cufflinks as cf
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

## Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")

## Figure&Display options
```

```
plt.rcParams["figure.figsize"] = (10,6)
pd.set_option('max_colwidth',200)
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 200)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

In [4]:
```
## Some Useful User-Defined-Functions
def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=Fa
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missi
    return missing_values[missing_values['Missing_Number']>0]
```

In [3]:
```
def first_looking(df):
    print(colored("Shape:", 'yellow', attrs=['bold']), df.shape,'\n',
          colored('*'*100, 'red', attrs=['bold']),
          colored("\nInfo:\n",'yellow', attrs=['bold']), sep='')
    print(df.info(), '\n',
          colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", 'yellow', attrs=['bold']), df.nunique(),'\n'
          colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values:\n", 'yellow', attrs=['bold']), missing_values(df),'
          colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", 'yellow', attrs=['bold']), *list(df.columns), sep='\
    print(colored('*'*100, 'red', attrs=['bold']), sep='')

    df.columns= df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')

    print(colored("Columns after rename:", 'yellow', attrs=['bold']), *list(df.columns
    print(colored('*'*100, 'red', attrs=['bold']), sep='')
```

In [5]:
```
# To view summary information about the columns
def summary(column):
    print(colored("Column: ",'yellow', attrs=['bold']), column)
    print(colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("Missing values: ", 'yellow', attrs=['bold']), df[column].isnull().s
    print(colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("Missing values(%): ", 'yellow', attrs=['bold']), round(df[column].i
    print(colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("Unique values: ", 'yellow', attrs=['bold']), df[column].nunique())
    print(colored('*'*100, 'red', attrs=['bold']), sep='')
    print(colored("Value counts: \n", 'yellow', attrs=['bold']), df[column].value_cour
    print(colored('*'*100, 'red', attrs=['bold']), sep='')
```

In [6]:
```
def multicolinearity_control(df):
    df_temp = df.corr()
    count = 'Done'
    feature =[]
    collinear= []
    for col in df_temp.columns:
        for i in df_temp.index:
            if abs(df_temp[col][i] > .8 and df_temp[col][i] < 1):
                    feature.append(col)
                    collinear.append(i)
                    cprint(f"multicolinearity alert in between {col} - {i}", "red", at
    else:
        cprint(f"There is NO multicollinearity between the features.", "blue", attrs=[
```

In [7]:
```
def duplicate_values(df):
    print(colored("Duplicate check...", 'yellow', attrs=['bold']), sep='')
    duplicate_values = df.duplicated(subset=None, keep='first').sum()
    if duplicate_values > 0:
        df.drop_duplicates(keep='first', inplace=True)
        print(duplicate_values, colored(" Duplicates were dropped!"),'\n',
              colored('*'*100, 'red', attrs=['bold']), sep='')
    else:
        print(colored("There are no duplicates"),'\n',
              colored('*'*100, 'red', attrs=['bold']), sep='')
```

In [8]:
```
def drop_columns(df, drop_columns):
    if drop_columns !=[]:
        df.drop(drop_columns, axis=1, inplace=True)
        print(drop_columns, 'were dropped')
    else:
        print(colored('Missing value control...', 'yellow', attrs=['bold']),'\n',
              colored('If there is a missing value above the limit you have given, the
```

In [9]:
```
def drop_null(df, limit):
    for i in df.isnull().sum().index:
        if (df.isnull().sum()[i]/df.shape[0]*100)>limit:
            print(df.isnull().sum()[i], 'percent of', i ,'were null and dropped')
            df.drop(i, axis=1, inplace=True)
    print(colored('Last shape after missing value control:', 'yellow', attrs=['bold'])
          colored('*'*100, 'red', attrs=['bold']), sep='')
```

In [10]:
```
def shape_control():
    print('df.shape:', df.shape)
    print('X.shape:', X.shape)
    print('y.shape:', y.shape)
    print('X_train.shape:', X_train.shape)
    print('y_train.shape:', y_train.shape)
    print('X_test.shape:', X_test.shape)
    print('y_test.shape:', y_test.shape)
```

In [11]:
```
# show values in bar graphic
def show_values_on_bars(axs):
    def _show_on_single_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() / 2
            _y = p.get_y() + p.get_height()
            value = '{:.2f}'.format(p.get_height())
            ax.text(_x, _y, value, ha="center")
    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)
```

In [12]:
```
'''This function detects the best z-score for outlier detection in the specified colum
def outlier_zscore(df, col, min_z=1, max_z = 5, step = 0.05, print_list = False):
    z_scores = stats.zscore(df[col].dropna())
    threshold_list = []

    for threshold in np.arange(min_z, max_z, step):
        threshold_list.append((threshold, len(np.where(z_scores > threshold)[0])))
```

```
    df_outlier = pd.DataFrame(threshold_list, columns = ['threshold', 'outlier_count']
    df_outlier['pct'] = (df_outlier.outlier_count - df_outlier.outlier_count.shift(-1)
    df_outlier['pct'] = df_outlier['pct'].apply(lambda x : x-100 if x == 100 else x)
    best_treshold = round(df_outlier.iloc[df_outlier.pct.argmax(), 0],2)
    IQR_coef = round((best_treshold - 0.675) / 1.35, 2)
    outlier_limit = int(df[col].dropna().mean() + (df[col].dropna().std()) * df_outlie
    num_outlier = df_outlier.iloc[df_outlier.pct.argmax(), 1]
    percentile_threshold = stats.percentileofscore(df[col].dropna(), outlier_limit)
    plt.plot(df_outlier.threshold, df_outlier.outlier_count)
    plt.vlines(best_treshold, 0, df_outlier.outlier_count.max(), colors="r", ls = ":")
    plt.annotate("Zscore : {}\nIQR_coef : {}\nValue : {}\nNum_outlier : {}\nPercentile
                                                                    IQR_coef,
                                                                    outlier_limi
                                                                    num_outlier,
                                                                    (np.round(pe
                                                                     np.round(10
                                                                    (best_tresho
    plt.show()
    if print_list:
        print(df_outlier)
    return (plt, df_outlier, best_treshold, IQR_coef, outlier_limit, num_outlier, perc
```

In [13]:
```
'''This function plots histogram, boxplot and z-score/outlier graphs for the specified

def outlier_inspect(df, col, min_z = 1, max_z = 5, step = 0.05, max_hist = None, bins
    fig = plt.figure(figsize=(20, 6))
    fig.suptitle(col, fontsize=16)
    plt.subplot(1,3,1)
    if max_hist == None:
        sns.distplot(df[col], kde=False, bins = 50)
    else :
        sns.distplot(df[df[col]<=max_hist][col], kde=False, bins = 50)
    plt.subplot(1,3,2)
    sns.boxplot(df[col])
    plt.subplot(1,3,3)
    z_score_inspect = outlier_zscore(df, col, min_z = min_z, max_z = max_z, step = ste
    plt.show()
```

In [14]:
```
"""This function gives max/min threshold, number of data, number of outlier and plots
according to the tree type and the entered z-score value for the relevant column."""

def num_outliers(df, col, whis = 1.5):
    q1 = df.groupby("class")[col].quantile(0.25)
    q3 = df.groupby("class")[col].quantile(0.75)
    iqr = q3 - q1
    print("Column_name :", col)
    print("whis :", whis)
    print("-------------------------------------------")
    for i in np.sort(df['class'].unique()):
        min_threshold = q1.loc[i] - whis*iqr.loc[i]
        max_threshold = q3.loc[i] + whis*iqr.loc[i]
        print("min_threshold:", min_threshold, "\nmax_threshold:", max_threshold)
        num_outliers = len(df[df["class"]==i][col][(df[col]<min_threshold) | (df[col]>
        print(f"Num_of_values for {i} :", len(df[df["class"]==i]))
        print(f"Num_of_outliers for {i} :", num_outliers)
        print("-------------------------------------------")
    return sns.boxplot(y = df[col], x = df["class"], whis=whis)
```

In [15]:
```
"""This function assigns the NaN-value first and then drop related rows, according to
whis value and plots the boxplot for the relevant column. """

def remove_outliers(df, col, whis=1.5):
    q1 = df.groupby("class")[col].quantile(0.25)
    q3 = df.groupby("class")[col].quantile(0.75)
    iqr = q3 - q1
    for i in np.sort(df['class'].unique()):
        min_threshold = q1.loc[i] - whis*iqr.loc[i]
        max_threshold = q3.loc[i] + whis*iqr.loc[i]
        df.loc[((df["class"]==i) & ((df[col]<min_threshold) | (df[col]>max_threshold))
    return sns.boxplot(y = df[col], x = df["class"], whis=whis)
```

In [16]:
```
# df0 = pd.read_csv('creditcard.csv')
df0=pd.read_csv(r'C:\Users\SHAKOUAT HOSSEN\String_List_Dictionaries\dictionary\creditc
df = df0.copy()
df.head(3)
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V1
0	0.000	-1.360	-0.073	2.536	1.378	-0.338	0.462	0.240	0.099	0.364	0.091	-0.552	-0.618	-0.99
1	0.000	1.192	0.266	0.166	0.448	0.060	-0.082	-0.079	0.085	-0.255	-0.167	1.613	1.065	0.48
2	1.000	-1.358	-1.340	1.773	0.380	-0.503	1.800	0.791	0.248	-1.515	0.208	0.625	0.066	0.71

In [17]:
```
df.tail()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
284802	172786.000	-11.881	10.072	-9.835	-2.067	-5.364	-2.607	-4.918	7.305	1.914	4.356	-1.593
284803	172787.000	-0.733	-0.055	2.035	-0.739	0.868	1.058	0.024	0.295	0.585	-0.976	-0.150
284804	172788.000	1.920	-0.301	-3.250	-0.558	2.631	3.031	-0.297	0.708	0.432	-0.485	0.412
284805	172788.000	-0.240	0.530	0.703	0.690	-0.378	0.624	-0.686	0.679	0.392	-0.399	-1.934
284806	172792.000	-0.533	-0.190	0.703	-0.506	-0.013	-0.650	1.577	-0.415	0.486	-0.915	-1.040

In [18]:
```
first_looking(df)
duplicate_values(df)
drop_columns(df, [])
drop_null(df, 90)
```

```
Shape:(284807, 31)
********************************************************************************
***************
Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
********************************************************************************
***************
Number of Uniques:
Time      124592
V1        275663
V2        275663
V3        275663
V4        275663
V5        275663
V6        275663
V7        275663
V8        275663
V9        275663
V10       275663
V11       275663
V12       275663
```

```
V13       275663
V14       275663
V15       275663
V16       275663
V17       275663
V18       275663
V19       275663
V20       275663
V21       275663
V22       275663
V23       275663
V24       275663
V25       275663
V26       275663
V27       275663
V28       275663
Amount     32767
Class          2
dtype: int64
********************************************************************************
***************
Missing Values:
Empty DataFrame
Columns: [Missing_Number, Missing_Percent]
Index: []
********************************************************************************
***************
All Columns:
- Time
- 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
- Class
********************************************************************************
```

```
***************
Columns after rename:
- time
- 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
- class
********************************************************************************
***************
Duplicate check...
1081 Duplicates were dropped!
********************************************************************************
***************
Missing value control...
If there is a missing value above the limit you have given, the relevant columns are
dropped and an information is given.
Last shape after missing value control:(283726, 31)
********************************************************************************
***************
```

In [19]:

```
#############a general look at data
df.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	75%	max
time	283726.000	94811.078	47481.048	0.000	54204.750	84692.500	139298.000	172792.000
v1	283726.000	0.006	1.948	-56.408	-0.916	0.020	1.316	2.455
v2	283726.000	-0.004	1.647	-72.716	-0.600	0.064	0.800	22.058
v3	283726.000	0.002	1.509	-48.326	-0.890	0.180	1.027	9.383
v4	283726.000	-0.003	1.414	-5.683	-0.850	-0.022	0.740	16.875
v5	283726.000	0.002	1.377	-113.743	-0.690	-0.053	0.612	34.802
v6	283726.000	-0.001	1.332	-26.161	-0.769	-0.275	0.397	73.302
v7	283726.000	0.002	1.228	-43.557	-0.553	0.041	0.570	120.589
v8	283726.000	-0.001	1.179	-73.217	-0.209	0.022	0.326	20.007
v9	283726.000	-0.002	1.095	-13.434	-0.644	-0.053	0.596	15.595
v10	283726.000	-0.001	1.076	-24.588	-0.536	-0.093	0.454	23.745
v11	283726.000	0.000	1.019	-4.797	-0.762	-0.032	0.740	12.019
v12	283726.000	-0.001	0.995	-18.684	-0.406	0.139	0.617	7.848
v13	283726.000	0.001	0.995	-5.792	-0.648	-0.013	0.663	7.127
v14	283726.000	0.000	0.952	-19.214	-0.426	0.050	0.492	10.527
v15	283726.000	0.001	0.915	-4.499	-0.581	0.049	0.650	8.878
v16	283726.000	0.001	0.874	-14.130	-0.467	0.067	0.524	17.315
v17	283726.000	0.000	0.843	-25.163	-0.484	-0.066	0.399	9.254
v18	283726.000	0.002	0.837	-9.499	-0.498	-0.002	0.502	5.041
v19	283726.000	-0.000	0.813	-7.214	-0.456	0.003	0.459	5.592
v20	283726.000	0.000	0.770	-54.498	-0.211	-0.062	0.133	39.421
v21	283726.000	-0.000	0.724	-34.830	-0.228	-0.029	0.186	27.203
v22	283726.000	-0.000	0.725	-10.933	-0.543	0.007	0.528	10.503
v23	283726.000	0.000	0.624	-44.808	-0.162	-0.011	0.148	22.528
v24	283726.000	0.000	0.606	-2.837	-0.354	0.041	0.440	4.585
v25	283726.000	-0.000	0.521	-10.295	-0.317	0.016	0.351	7.520
v26	283726.000	0.000	0.482	-2.605	-0.327	-0.052	0.240	3.517
v27	283726.000	0.002	0.396	-22.566	-0.071	0.001	0.091	31.612
v28	283726.000	0.001	0.328	-15.430	-0.053	0.011	0.078	33.848
amount	283726.000	88.473	250.399	0.000	5.600	22.000	77.510	25691.160
class	283726.000	0.002	0.041	0.000	0.000	0.000	0.000	1.000

In [20]:

```
### Examination of features ana data insight
### class Colmun-Terget Clomun
cprint('Have a first look to "class" column', "white", "on_black", attrs =["bold"])
```

```
print("_____"*10)
summary('class')
```

```
Have a first look to "class" column

_________________________________________________________
Column:  class
**********************************************************************************
***************
Missing values:  0
**********************************************************************************
***************
Missing values(%):  0.0
**********************************************************************************
***************
Unique values:  2
**********************************************************************************
***************
Value counts:
class
0    283253
1       473
Name: count, dtype: int64
**********************************************************************************
***************
```

In [21]:
```
cprint('Mean value according to the "class" column', "white", "on_black", attrs=["bold
df.groupby('class').mean()
```

```
Mean value according to the "class" column
```

Out[21]:

class	time	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	
0	94835.058	0.013	-0.010	0.013	-0.010	0.007	0.001	0.010	-0.002	0.003	0.008	-0.006	0
1	80450.514	-4.498	3.406	-6.730	4.473	-2.957	-1.433	-5.176	0.953	-2.522	-5.453	3.716	-6

In [22]:
```
cprint('Descriptive Static according to the to the "class==0 Non-Fraudlent"', "white",
df[df['class']==1].describe().T.style.background_gradient(subset=['mean', 'min', '50%'
```

```
Descriptive Static according to the to the "class==0 Non-Fraudlent"
```

Out[22]:

	count	mean	std	min	25%	50%	75
time	473.000000	80450.513742	48636.179973	406.000000	41203.000000	73408.000000	129095.0000
v1	473.000000	-4.498280	6.593145	-30.552380	-5.603690	-2.271755	-0.3614
v2	473.000000	3.405965	4.122500	-8.402154	1.145381	2.617105	4.5717
v3	473.000000	-6.729599	6.909647	-31.103685	-7.926507	-4.875397	-2.1714
v4	473.000000	4.472591	2.871523	-1.313275	2.288644	4.100098	6.2909
v5	473.000000	-2.957197	5.278831	-22.105532	-4.278983	-1.372245	0.2608
v6	473.000000	-1.432518	1.715347	-6.406267	-2.450444	-1.420468	-0.4136
v7	473.000000	-5.175912	6.858024	-43.557242	-6.989195	-2.902079	-0.9071
v8	473.000000	0.953255	5.585950	-41.044261	-0.161518	0.617738	1.7094
v9	473.000000	-2.522124	2.465047	-13.434066	-3.796760	-2.099049	-0.7883
v10	473.000000	-5.453274	4.706451	-24.588262	-7.297803	-4.466284	-2.4474
v11	473.000000	3.716347	2.672817	-1.702228	1.928568	3.525726	5.2241
v12	473.000000	-6.103254	4.582331	-18.683715	-8.601648	-5.437354	-2.8249
v13	473.000000	-0.094324	1.108001	-3.127795	-0.978065	-0.063634	0.6949
v14	473.000000	-6.835946	4.253210	-19.214325	-9.505141	-6.590550	-4.2524
v15	473.000000	-0.072830	1.045631	-4.498945	-0.638498	-0.038632	0.6339
v16	473.000000	-4.000956	3.831724	-14.129855	-6.469187	-3.302899	-1.1424
v17	473.000000	-6.463285	6.965744	-25.162799	-11.588544	-5.157596	-1.1290
v18	473.000000	-2.157071	2.901815	-9.498746	-4.568859	-1.417917	0.1164
v19	473.000000	0.669143	1.534246	-3.681904	-0.300931	0.647709	1.6610
v20	473.000000	0.405043	1.289414	-4.128186	-0.160163	0.285559	0.8214
v21	473.000000	0.466550	2.731191	-22.797604	0.027935	0.573898	1.1926
v22	473.000000	0.086639	1.181295	-8.887017	-0.521934	0.055179	0.6163
v23	473.000000	-0.096464	1.508570	-19.254328	-0.341881	-0.075034	0.2876
v24	473.000000	-0.106643	0.517900	-2.028024	-0.436539	-0.061263	0.2820
v25	473.000000	0.040615	0.806785	-4.781606	-0.320311	0.077913	0.4638
v26	473.000000	0.050456	0.463016	-1.152671	-0.263078	0.012792	0.3955
v27	473.000000	0.213774	1.245779	-7.263482	-0.015551	0.394682	0.8210
v28	473.000000	0.078270	0.533100	-1.869290	-0.097223	0.145895	0.3723
amount	473.000000	123.871860	260.211041	0.000000	1.000000	9.820000	105.8900
class	473.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.0000

In [23]:
```
cprint('Descriptive Static according to the to the "class==1 Fraudlent"', "white", "or
df[df['class']==0].describe().T.style.background_gradient(subset=['mean', 'min', '50%'
```

```
Descriptive Static according to the to the "class==1 Fraudlent"
```

Out[23]:

	count	mean	std	min	25%	50%	
time	283253.000000	94835.058093	47475.550607	0.000000	54233.000000	84711.000000	139308.(
v1	283253.000000	0.013439	1.922179	-56.407510	-0.913431	0.022562	1.3
v2	283253.000000	-0.009829	1.633520	-72.715728	-0.601398	0.062561	0.7
v3	283253.000000	0.012853	1.457593	-48.325589	-0.883966	0.182247	1.(
v4	283253.000000	-0.010440	1.398575	-5.683171	-0.851605	-0.024500	0.7
v5	283253.000000	0.006769	1.355816	-113.743307	-0.687888	-0.052807	0.(
v6	283253.000000	0.001251	1.329914	-26.160506	-0.767543	-0.274172	0.3
v7	283253.000000	0.010447	1.177480	-31.764946	-0.550146	0.041664	0.5
v8	283253.000000	-0.002448	1.157140	-73.216718	-0.208841	0.021633	0.3
v9	283253.000000	0.002613	1.086902	-6.290730	-0.641649	-0.051368	0.5
v10	283253.000000	0.007663	1.036321	-14.741096	-0.533155	-0.092120	0.4
v11	283253.000000	-0.006004	1.002257	-4.797473	-0.762737	-0.034446	0.7
v12	283253.000000	0.009476	0.945382	-15.144988	-0.402882	0.140573	0.(
v13	283253.000000	0.000762	0.995226	-5.791881	-0.647502	-0.012905	0.(
v14	283253.000000	0.011668	0.894379	-18.392091	-0.422708	0.051486	0.4
v15	283253.000000	0.001166	0.914657	-4.391307	-0.581407	0.049435	0.(
v16	283253.000000	0.007845	0.844608	-10.115560	-0.464241	0.068100	0.5
v17	283253.000000	0.010963	0.748513	-17.098444	-0.482794	-0.065096	0.3
v18	283253.000000	0.005120	0.824952	-5.366660	-0.496635	-0.001436	0.5
v19	283253.000000	-0.001382	0.811183	-7.213527	-0.456350	0.002737	0.4
v20	283253.000000	-0.000489	0.768649	-54.497720	-0.211517	-0.062507	0.1
v21	283253.000000	-0.001150	0.715629	-34.830382	-0.228406	-0.029798	0.1
v22	283253.000000	-0.000160	0.723541	-10.933144	-0.542737	0.006675	0.5
v23	283253.000000	0.000360	0.621165	-44.807735	-0.161490	-0.011077	0.1
v24	283253.000000	0.000393	0.605748	-2.836627	-0.354306	0.041115	0.4
v25	283253.000000	-0.000301	0.520612	-10.295397	-0.317476	0.016190	0.3
v26	283253.000000	0.000065	0.482080	-2.604551	-0.326853	-0.052293	0.2
v27	283253.000000	0.001409	0.392700	-22.565679	-0.070650	0.001368	0.(
v28	283253.000000	0.000418	0.327563	-15.430084	-0.052808	0.011238	0.(
amount	283253.000000	88.413575	250.379023	0.000000	5.670000	22.000000	77.4
class	283253.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(

In [24]:
```
fig = px.pie(df, names='class', title='"class" Column Distribution')
fig.show()
```

In [25]:
```
y = df['class']
print(f'Percentage of class-1: %{round(y.value_counts(normalize= True)[1] * 100, 2 )}
```

```
Percentage of class-1: %0.17 -->(473 Obserbation of class-1)
Percentage of class-1 %0.17 -->(283253 Obserbation of class-0)
```

In [26]:
```
cprint("\nExamination of Features and Data Insights\n", 'green', 'on_yellow', attrs =
cprint("In the given dataset, feature 'Class' is the\
response variable and it takes value 1 in case of fraud and 0 otherwise. \nSo, we can
groups and compare their characteristics. Here, we can find the average of\nboth the g
function.\n", 'green', 'on_yellow', attrs = ['bold'])

cprint("Have a look to 'time' Column ", "black", "on_white", attrs=['bold'])
print("_____"*10)
summary('time')
```

Examination of Features and Data Insights

In the given dataset, feature 'Class' is theresponse variable and it takes value 1 in case of fraud and 0 otherwise.
So, we can divide data into two groups and compare their characteristics. Here, we can find the average of
both the groups using groupby() and mean() function.

Have a look to 'time' Column

```
_____________________________________________________
Column:  time
**********************************************************************************
***************
Missing values:  0
**********************************************************************************
***************
Missing values(%):  0.0
**********************************************************************************
***************
Unique values:  124592
**********************************************************************************
***************
Value counts:
time
3767.000      21
3770.000      20
19912.000     19
3750.000      19
73885.000     17
              ..
127732.000     1
62217.000      1
127739.000     1
127741.000     1
172792.000     1
Name: count, Length: 124592, dtype: int64
**********************************************************************************
***************
```

In [27]:
```
cprint("Describe value of column 'time' ", "black", "on_white", attrs=['bold'])
df.time.describe()
```

Describe value of column 'time'

Out[27]:
```
count   283726.000
mean     94811.078
std      47481.048
min          0.000
25%      54204.750
50%      84692.500
75%     139298.000
max     172792.000
Name: time, dtype: float64
```

In [28]:
```
fig = px.box(df, x = df['time'], color = df['class'], title = 'Boxplot of column "time
fig.show()
```

In [29]:
```
cprint("Describe value of column 'amount' ", "black", "on_white", attrs=['bold'])
df.amount.describe()
```

Out[29]:
```
Describe value of column 'amount'
count   283726.000
mean        88.473
std        250.399
min          0.000
25%          5.600
50%         22.000
75%         77.510
max      25691.160
Name: amount, dtype: float64
```

In [30]:
```
fig = px.box(df, x = df['amount'], color = df['class'], title = 'Boxplot of column "am
fig.show()
```

In [31]:
```python
def transaction_transformer(price):
    if price <= 5.600:
        return 'Q-1'
    if 5.600 < price <= 22.700:
        return 'Q-2'
    if 22.700 < price <= 75.510:
        return 'Q-3'
    else:
        return 'Q-4'
```

In [32]:
```python
df['transaction_class'] = df['amount'].apply(transaction_transformer)
```

In [33]:
```python
values = df[df['class'] == 0]['transaction_class'].value_counts().values
names = df[df['class'] == 0]['transaction_class'].value_counts().index

fig = px.pie(
    values=values,
    names=names,
    title='transaction_class & class==0 Distribution'
)

fig.show()
```

In [34]:

```
values = df[df['class'] == 1]['transaction_class'].value_counts().values
names = df[df['class'] == 1]['transaction_class'].value_counts().index

fig = px.pie(
    values=values,
    names=names,
    title='transaction_class & class==1 Distribution'
)

fig.show()
```

In [35]:
```
print('Number of Fraudulent transcations for Q-1          : ', df[(df['class']==1) &(
print('Total amount of Fraudulent transcations for Q-1    : ', df[(df['class']==1) &(
print('Mean amount of Fraudulent transcations for Q-1     : ', df[(df['class']==1) &(
```

```
Number of Fraudulent transcations for Q-1          :  213
Total amount of Fraudulent transcations for Q-1    :  251.73000000000002
Mean amount of Fraudulent transcations for Q-1     :  1.181830985915493
```

In [36]:
```
print('Number of Fraudulent transcations for Q-2          : ', df[(df['class']==1) &(
print('Total amount of Fraudulent transcations for Q-2    : ', df[(df['class']==1) &(
print('Mean amount of Fraudulent transcations for Q-2     : ', df[(df['class']==1) &(
```

```
Number of Fraudulent transcations for Q-2          :  49
Total amount of Fraudulent transcations for Q-2    :  591.53
Mean amount of Fraudulent transcations for Q-2     :  12.07204081632653
```

In [37]:
```
print('Number of Fraudulent transcations for Q-3          : ', df[(df['class']==1) &(
print('Total amount of Fraudulent transcations for Q-3    : ', df[(df['class']==1) &(
print('Mean amount of Fraudulent transcations for Q-3     : ', df[(df['class']==1) &(
```

```
Number of Fraudulent transcations for Q-3          :  41
Total amount of Fraudulent transcations for Q-3    :  1733.79
Mean amount of Fraudulent transcations for Q-3     :  42.28756097560976
```

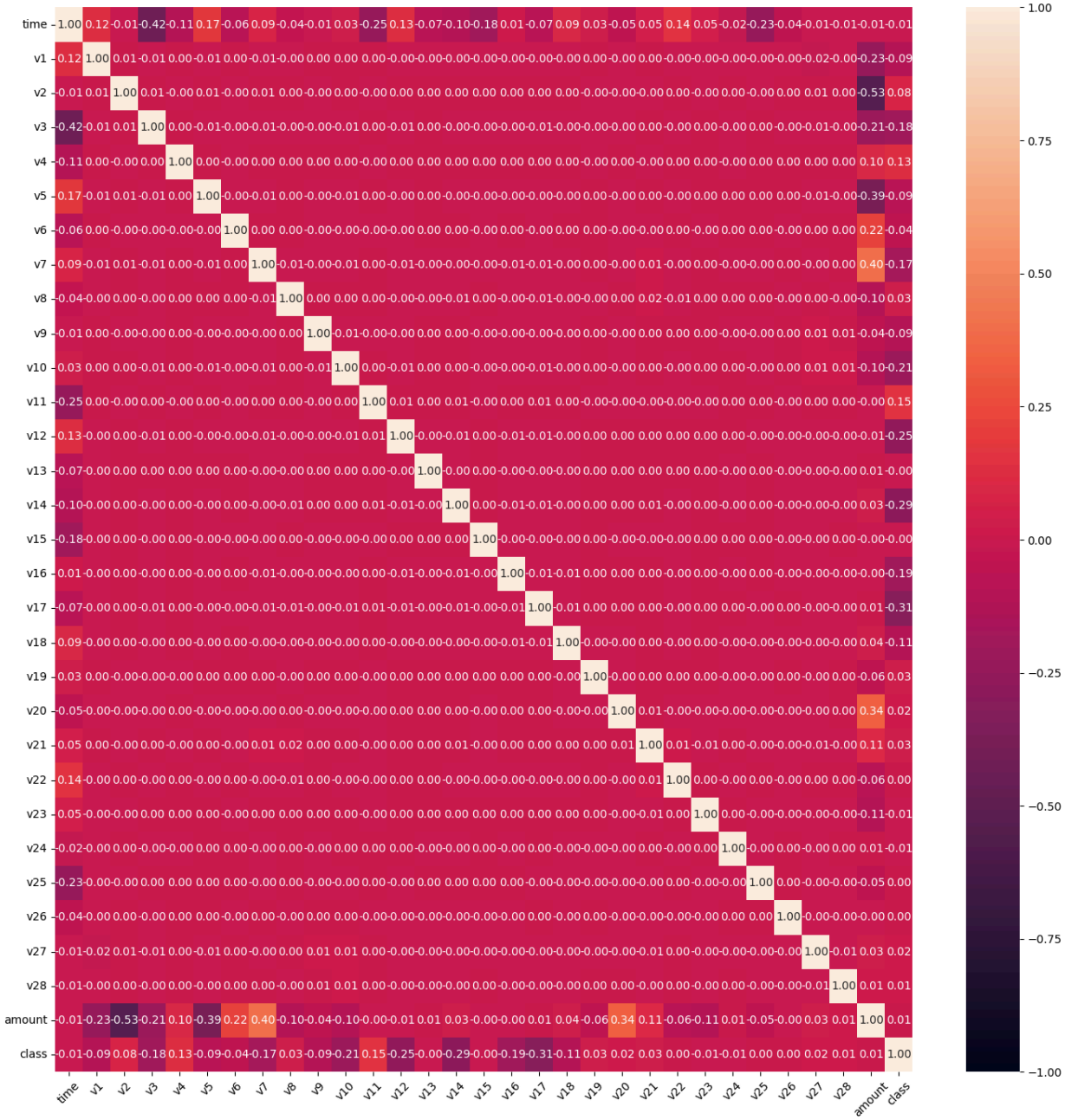
In [38]:
```
print('Number of Fraudulent transcations for Q-4          : ', df[(df['class']==1) &(
print('Total amount of Fraudulent transcations for Q-4    : ', df[(df['class']==1) &(
print('Mean amount of Fraudulent transcations for Q-4     : ', df[(df['class']==1) &(
```

```
Number of Fraudulent transcations for Q-4           :  170
Total amount of Fraudulent transcations for Q-4     :  56014.34
Mean amount of Fraudulent transcations for Q-4      :  329.4961176470588
```

In [39]:
```
df.drop(['transaction_class'], axis = 1, inplace = True)
```

In [40]:
```
cprint("Heat Map of df", 'black', 'on_white', attrs = ['bold'])
print("_____"*10)
plt.figure(figsize =(17, 17))
sns.heatmap(df.corr(), annot = True, fmt = '.2f', vmin = -1, vmax = 1)
plt.xticks(rotation = 45);
```

Heat Map of df



In [41]:
```
multicolinearity_control(df)
```

There is NO multicollinearity between the features.

In [42]:
```
df.corr()['class'].sort_values().drop('class').iplot(kind='barh', title = 'Correlatior
```

In [43]:
```
cprint("\nData Cleaning\n", 'green', 'on_yellow', attrs = ['bold'])
```

Data Cleaning

In [44]:
```
cprint("Check Missing Values and Outliers", 'black', 'on_white', attrs = ['bold'])
missing_values(df)
```

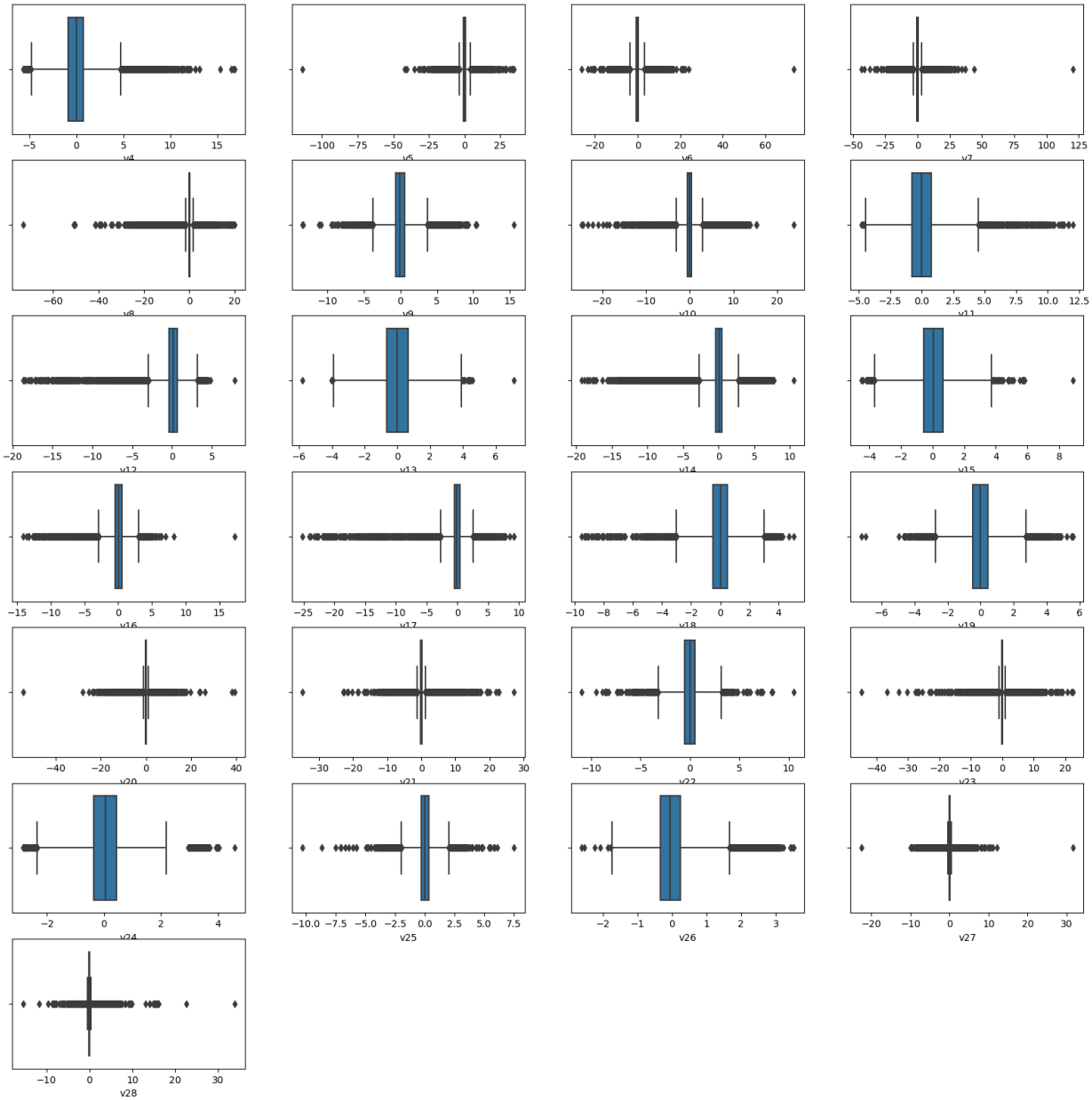
Check Missing Values and Outliers

Out[44]:

	Missing_Number	Missing_Percent

In [48]:
```
cprint("BoxPlot", 'black', 'on_white', attrs = ['bold'])
%matplotlib inline
index = 0
plt.figure(figsize=(20, 20))
for feauture in df.columns[2:29]:
    index += 1
    plt.subplot(7, 4, index)
    sns.boxplot(x=feauture, data = df, whis = 2.5)
```

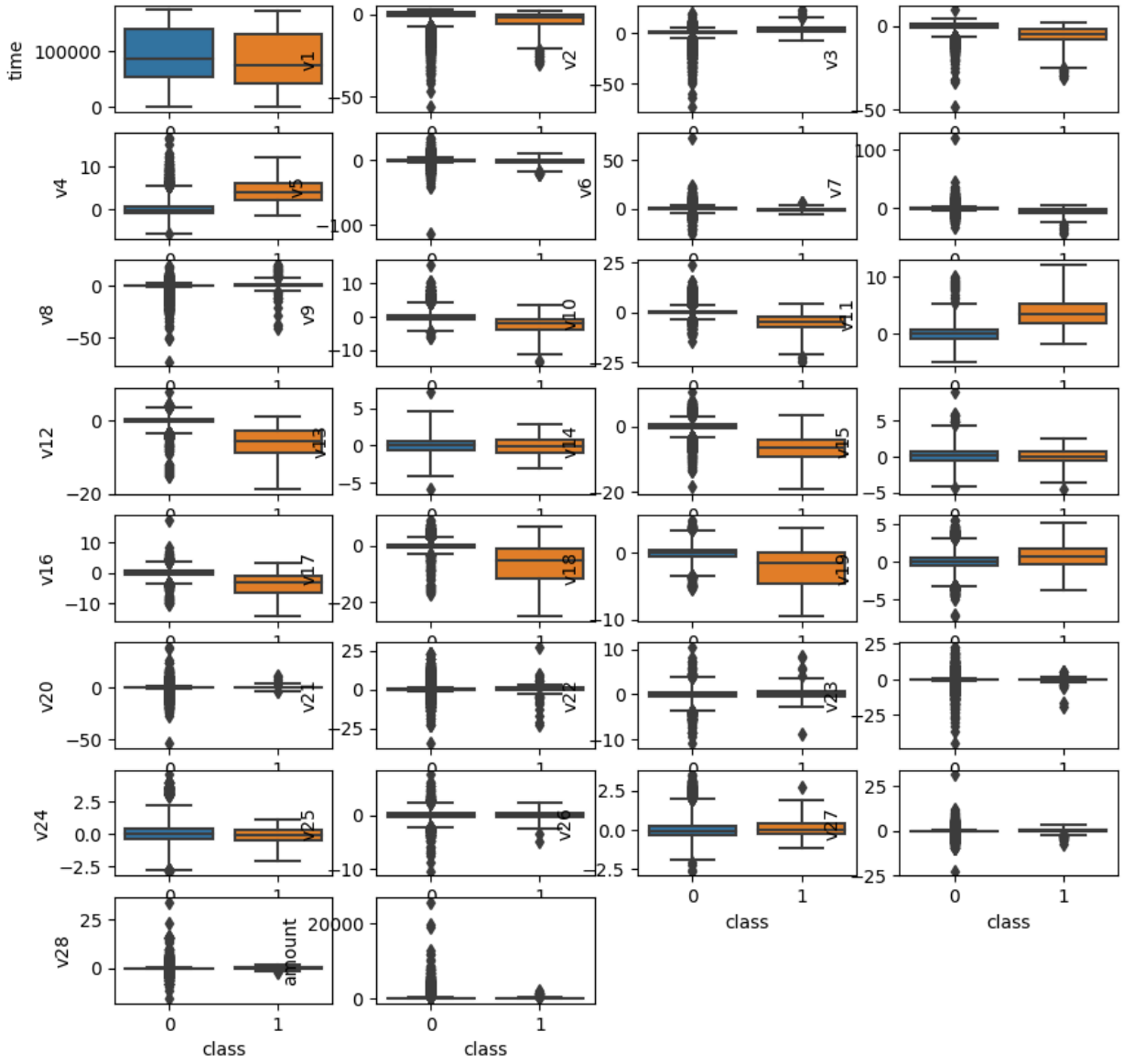
BoxPlot



In [46]:

```
cprint("BoxPlot", 'black', 'on_white', attrs = ['bold'])
%matplotlib inline
index = 0
plt.figure(figsize=(10, 10))
for feauture in df.columns[:30]:
    index += 1
    plt.subplot(8, 4, index)
    sns.boxplot(y=feauture, x = "class", data = df, whis = 3)
```

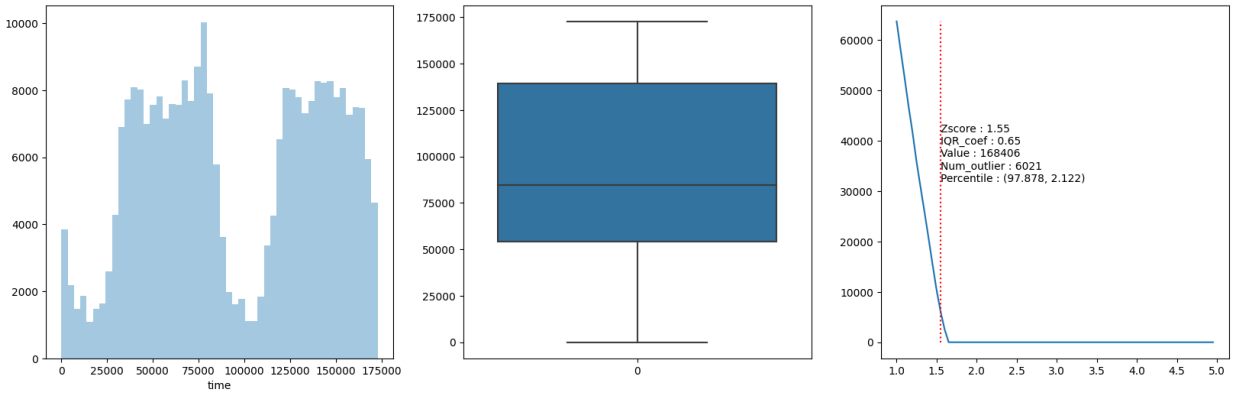
BoxPlot



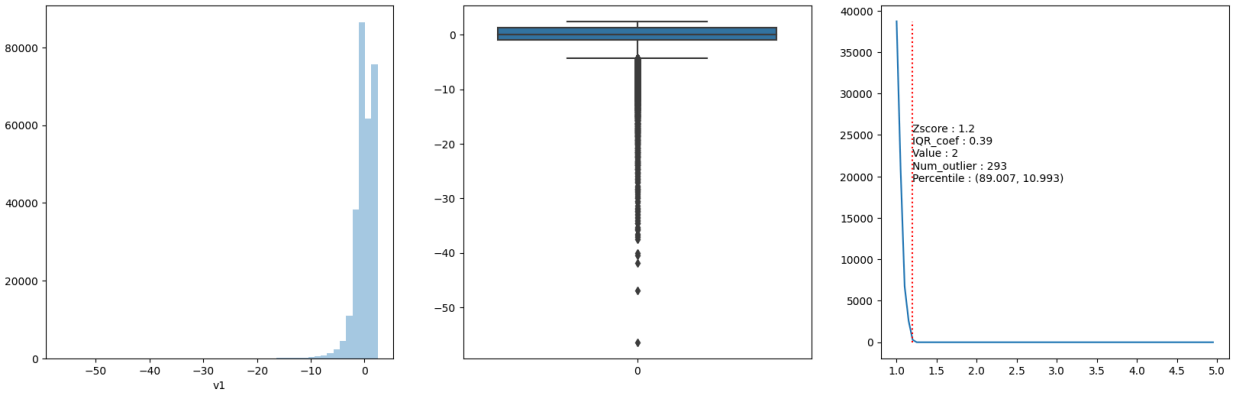
In [34]:
```
from scipy.stats import zscore
from scipy import stats
from numpy import percentile
cprint("'outlier_zscore' function detects the best z-score for outlier detection in th
cprint("'outlier_inspect' function plots histogram, boxplot and z-score/outlier graphs
cprint("Outlier Inception\n", 'black', 'on_white', attrs = ['bold'])
for col in df.columns[:10]:
    outlier_inspect(df, col)
```

```
'outlier_zscore' function detects the best z-score for outlier detection in the speci
fied column.

'outlier_inspect' function plots histogram, boxplot and z-score/outlier graphs for th
e specified column.

Outlier Inception
```

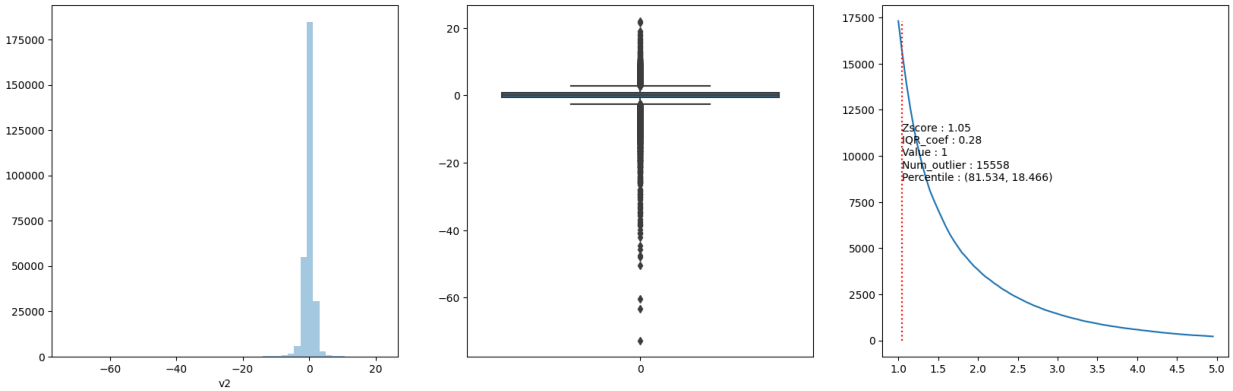
time


time
Zscore : 1.55
IQR_coef : 0.65
Value : 168406
Num_outlier : 6021
Percentile : (97.878, 2.122)

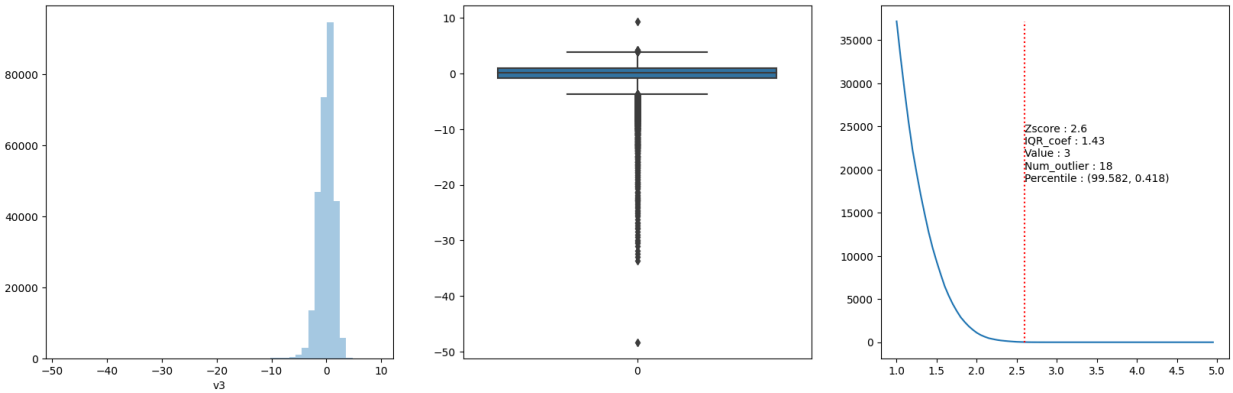
v1


v1
Zscore : 1.2
IQR_coef : 0.39
Value : 2
Num_outlier : 293
Percentile : (89.007, 10.993)

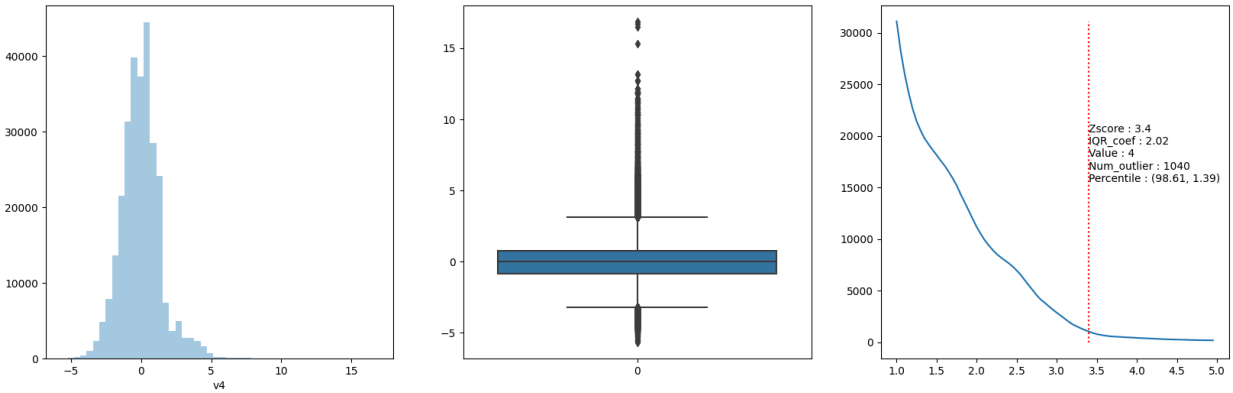
v2


v2
Zscore : 1.05
IQR_coef : 0.28
Value : 1
Num_outlier : 15558
Percentile : (81.534, 18.466)

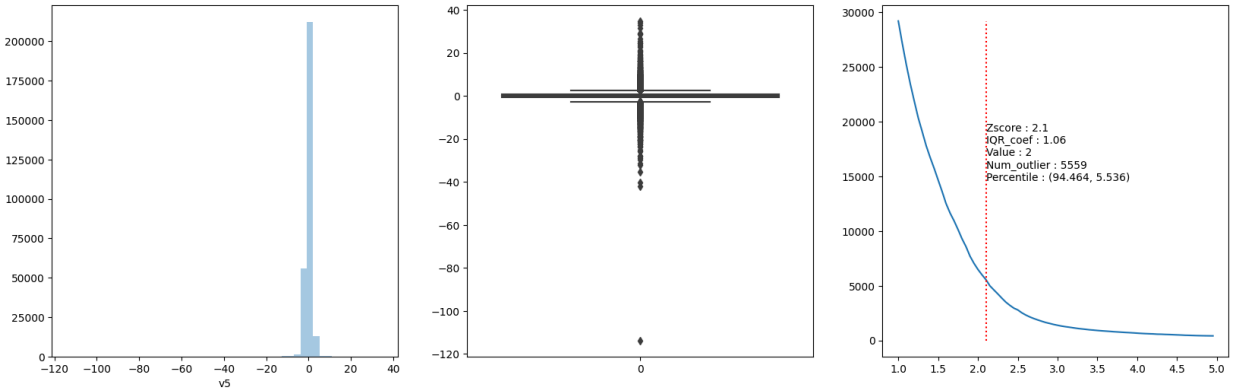
v3


v3
Zscore : 2.6
IQR_coef : 1.43
Value : 3
Num_outlier : 18
Percentile : (99.582, 0.418)

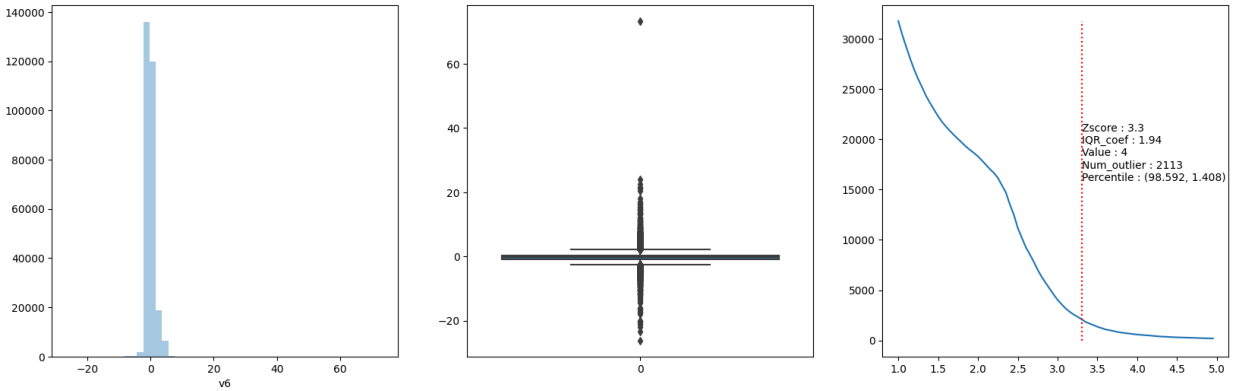
v4


v4
Zscore : 3.4
IQR_coef : 2.02
Value : 4
Num_outlier : 1040
Percentile : (98.61, 1.39)

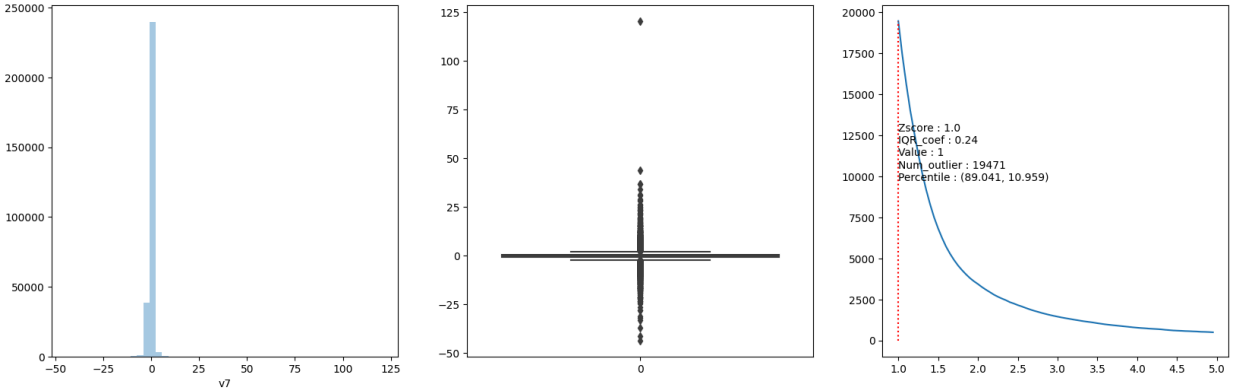
v5

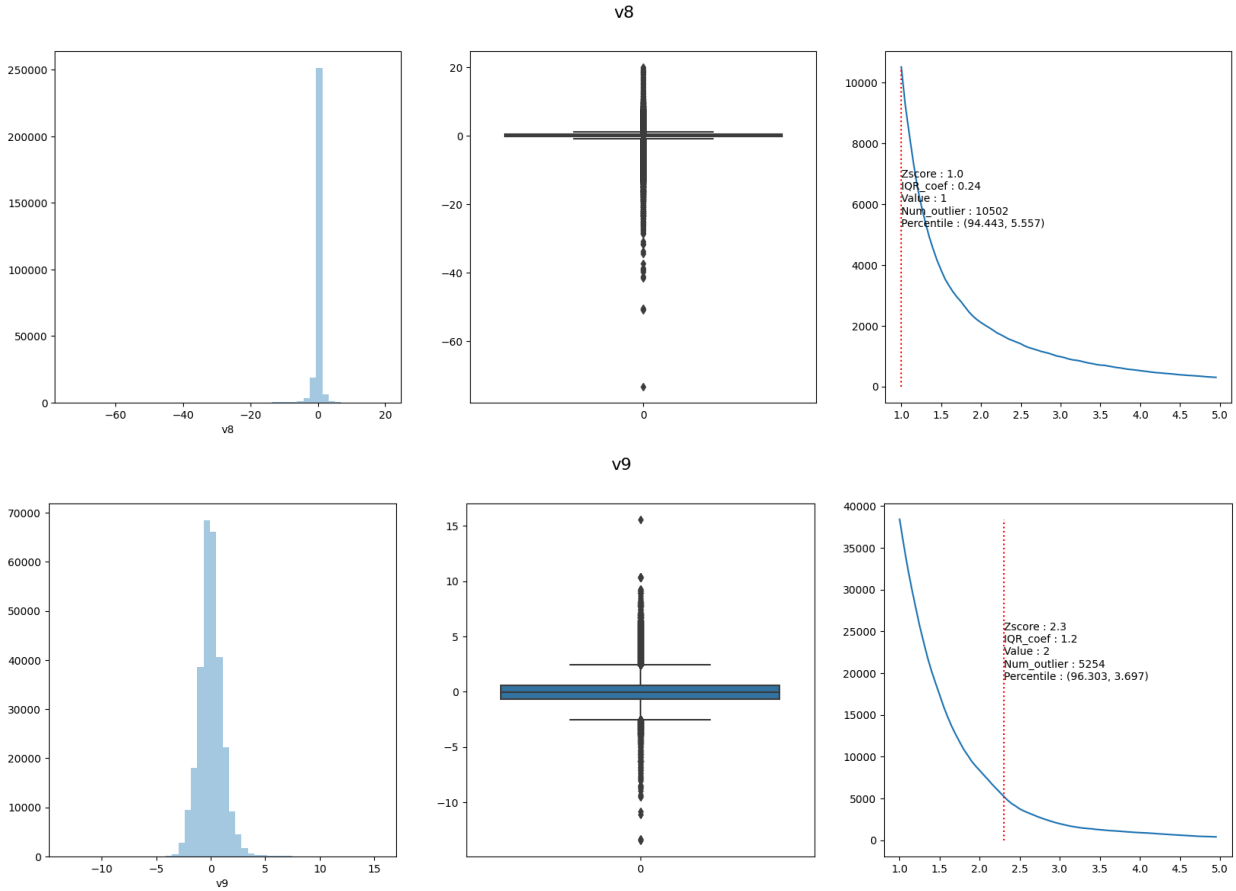

v5
Zscore : 2.1
IQR_coef : 1.06
Value : 2
Num_outlier : 5559
Percentile : (94.464, 5.536)

v6


v6
Zscore : 3.3
IQR_coef : 1.94
Value : 4
Num_outlier : 2113
Percentile : (98.592, 1.408)

v7


v7
Zscore : 1.0
IQR_coef : 0.24
Value : 1
Num_outlier : 19471
Percentile : (89.041, 10.959)



In [35]:
```
cprint("\nEvaluation\n", 'black', 'on_white', attrs = ['bold'])
IQR_coef = 3
z_score = round(0.675 + IQR_coef * 1.35, 2)
z_score
```

Evaluation

Out[35]: 4.73

In [36]:
```
z_score = 3
iqr_coef = round((z_score - 0.675)/1.35, 2)
iqr_coef
```

Out[36]: 1.72

In [37]:
```
cprint("\nLet's create a new df before outlier operation\n", 'black', 'on_white', attr
df_out = df.copy()
```

Let's create a new df before outlier operation

In [38]:
```
cprint("\nShape of new df before outlier check\n", 'black', 'on_white', attrs = ['bold
df_out.shape
```

Shape of new df before outlier check

Out[38]: (283726, 31)

In [39]:
```
cprint("\nValue counts of 'class' Column before outlier check \n", 'black', 'on_white'
df_out['class'].value_counts(dropna=False)
```

Value counts of 'class' Column before outlier check

Out[39]:
```
class
0    283253
1       473
Name: count, dtype: int64
```

In [40]:
```
cprint("\nMissing value control before outlier check \n", 'black', 'on_white', attrs =
missing_values(df_out)
```

Missing value control before outlier check

Out[40]:

	Missing_Number	Missing_Percent

In [41]:
```
for col in df_out.columns[:30]:
    num_outliers(df_out, col, whis = 3)
```

```
Column_name : time
whis : 3
-----------------------------------------
min_threshold: -200992.0
max_threshold: 394533.0
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 0
-----------------------------------------
min_threshold: -222473.0
max_threshold: 392771.0
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v1
whis : 3
-----------------------------------------
min_threshold: -7.604089690316541
max_threshold: 8.007446923764526
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 1870
-----------------------------------------
min_threshold: -21.330477603798716
max_threshold: 15.365359485400905
Num_of_values for 1 : 473
Num_of_outliers for 1 : 19
-----------------------------------------
Column_name : v2
whis : 3
-----------------------------------------
min_threshold: -4.796627511937092
max_threshold: 4.992241486644909
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 4330
-----------------------------------------
min_threshold: -9.13370526362269
max_threshold: 14.85082955491482
Num_of_values for 1 : 473
Num_of_outliers for 1 : 12
-----------------------------------------
Column_name : v3
whis : 3
-----------------------------------------
min_threshold: -6.620648975136031
max_threshold: 6.764944223798885
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 599
-----------------------------------------
min_threshold: -25.19166402031824
max_threshold: 15.093703276050778
Num_of_values for 1 : 473
Num_of_outliers for 1 : 14
-----------------------------------------
Column_name : v4
whis : 3
-----------------------------------------
min_threshold: -5.609112666525339
max_threshold: 5.491739158975989
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 306
-----------------------------------------
```

```
min_threshold: -9.7181792669762
max_threshold: 18.2977407988582
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v5
whis : 3
-----------------------------------------
min_threshold: -4.588876019418242
max_threshold: 4.513429853879736
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 2169
-----------------------------------------
min_threshold: -17.89839372337706
max_threshold: 13.880232458346873
Num_of_values for 1 : 473
Num_of_outliers for 1 : 11
-----------------------------------------
Column_name : v6
whis : 3
-----------------------------------------
min_threshold: -4.2632034355050346
max_threshold: 3.8933385454012366
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 5392
-----------------------------------------
min_threshold: -8.560835548030997
max_threshold: 5.696743857658712
Num_of_values for 1 : 473
Num_of_outliers for 1 : 3
-----------------------------------------
Column_name : v7
whis : 3
-----------------------------------------
min_threshold: -3.9136729424264733
max_threshold: 3.934555461413029
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 2824
-----------------------------------------
min_threshold: -25.235213519245576
max_threshold: 17.338830312073696
Num_of_values for 1 : 473
Num_of_outliers for 1 : 7
-----------------------------------------
Column_name : v8
whis : 3
-----------------------------------------
min_threshold: -1.8087808106549712
max_threshold: 1.9244129780498362
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 11037
-----------------------------------------
min_threshold: -5.774323492620118
max_threshold: 7.322222032639315
Num_of_values for 1 : 473
Num_of_outliers for 1 : 48
-----------------------------------------
Column_name : v9
whis : 3
-----------------------------------------
```

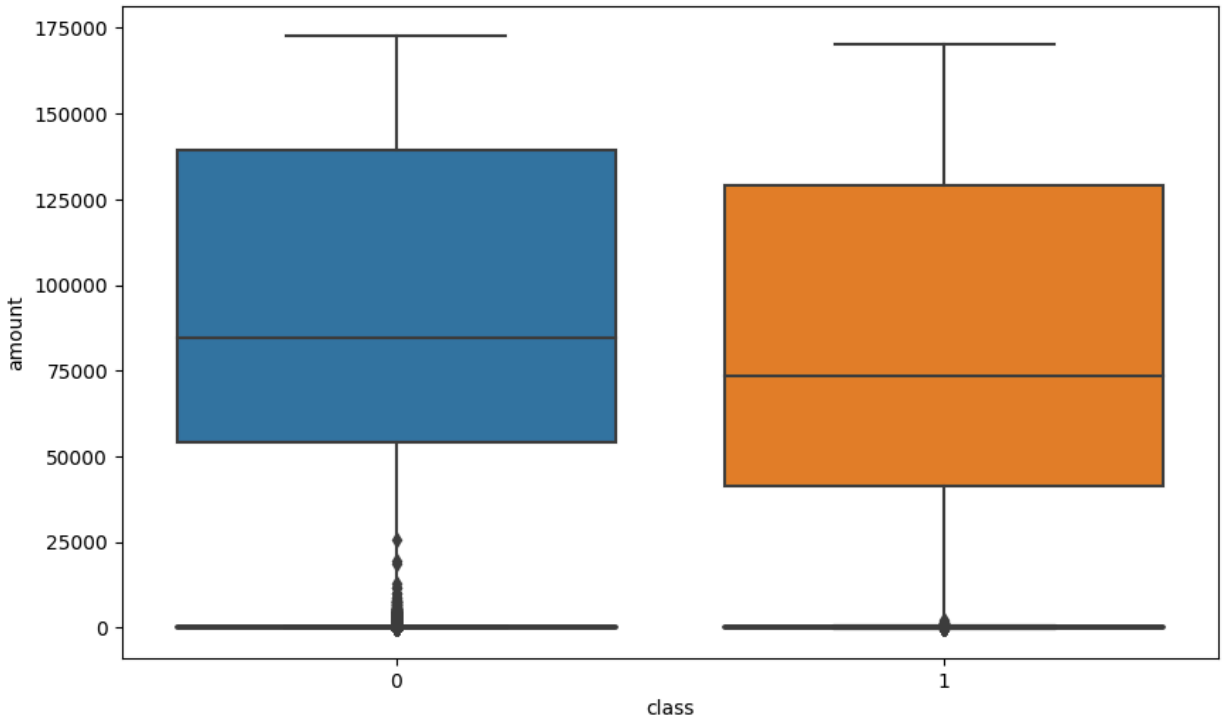
```
min_threshold: -4.35750131151855
max_threshold: 4.312820912032102
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 954
-----------------------------------------
min_threshold: -12.821875790991772
max_threshold: 8.236728667382978
Num_of_values for 1 : 473
Num_of_outliers for 1 : 2
-----------------------------------------
Column_name : v10
whis : 3
-----------------------------------------
min_threshold: -3.496992687296603
max_threshold: 3.418629644001385
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 2802
-----------------------------------------
min_threshold: -21.848805634723913
max_threshold: 12.103533029597791
Num_of_values for 1 : 473
Num_of_outliers for 1 : 4
-----------------------------------------
Column_name : v11
whis : 3
-----------------------------------------
min_threshold: -5.260608937463697
max_threshold: 5.234425018513713
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 26
-----------------------------------------
min_threshold: -7.958101484508561
max_threshold: 15.110793120758721
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v12
whis : 3
-----------------------------------------
min_threshold: -3.4648674747502413
max_threshold: 3.6797664225826425
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 987
-----------------------------------------
min_threshold: -25.931756176757737
max_threshold: 14.505162289226156
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v13
whis : 3
-----------------------------------------
min_threshold: -4.579325782140245
max_threshold: 4.594930415001589
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 2
-----------------------------------------
min_threshold: -5.9971911220712
max_threshold: 5.714104298273796
Num_of_values for 1 : 473
```

```
Num_of_outliers for 1 : 0
----------------------------------------
Column_name : v14
whis : 3
----------------------------------------
min_threshold: -3.1700128779302617
max_threshold: 3.24036445340293
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 1969
----------------------------------------
min_threshold: -25.2631665066491
max_threshold: 11.50556008227446
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
----------------------------------------
Column_name : v15
whis : 3
----------------------------------------
min_threshold: -4.276068002526584
max_threshold: 4.344806772961216
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 17
----------------------------------------
min_threshold: -4.4559304244943885
max_threshold: 4.451411449754766
Num_of_values for 1 : 473
Num_of_outliers for 1 : 1
----------------------------------------
Column_name : v16
whis : 3
----------------------------------------
min_threshold: -3.428861112005019
max_threshold: 3.488585601020494
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 340
----------------------------------------
min_threshold: -22.44934574586738
max_threshold: 14.837691582857719
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
----------------------------------------
Column_name : v17
whis : 3
----------------------------------------
min_threshold: -3.12871397062705
max_threshold: 3.0450983928153663
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 1093
----------------------------------------
min_threshold: -42.967006766016056
max_threshold: 30.249407289699302
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
----------------------------------------
Column_name : v18
whis : 3
----------------------------------------
min_threshold: -3.4934236823009606
max_threshold: 3.4990844011183726
Num_of_values for 0 : 283253
```

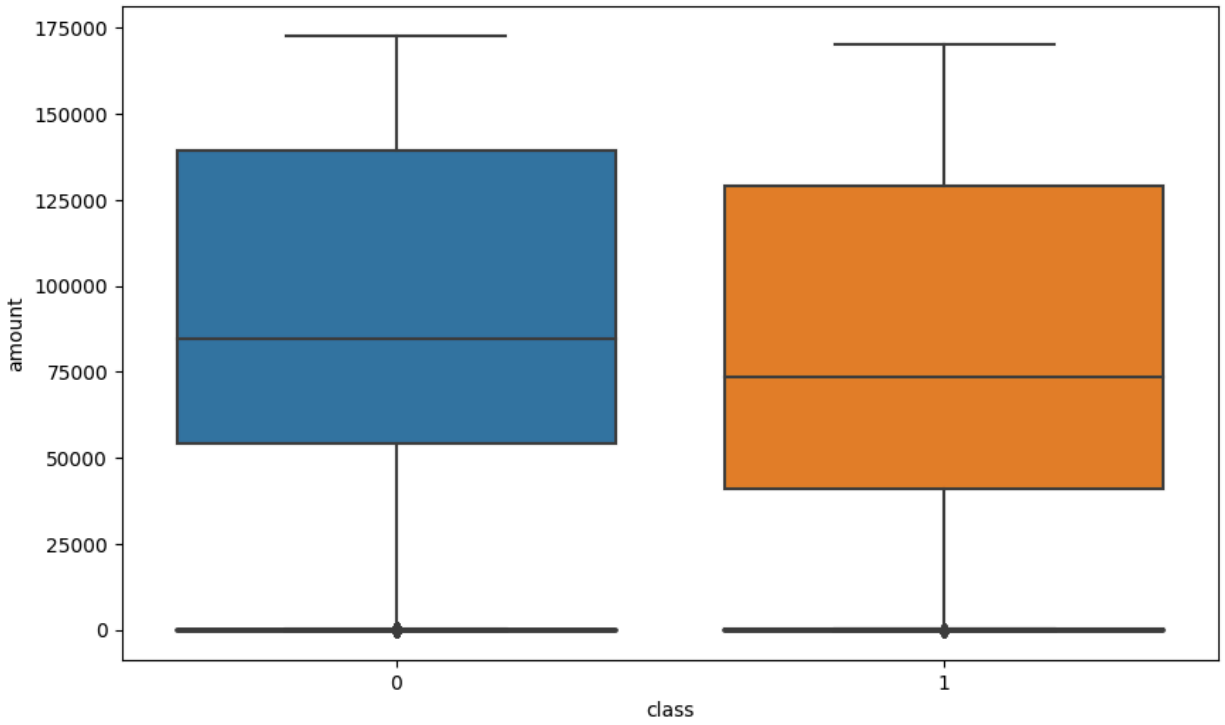
```
Num_of_outliers for 0 : 118
-----------------------------------------
min_threshold: -18.624794700101052
max_threshold: 14.172387937221927
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v19
whis : 3
-----------------------------------------
min_threshold: -3.196921623131294
max_threshold: 3.197745717679896
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 601
-----------------------------------------
min_threshold: -6.186811190339024
max_threshold: 7.546909770544928
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v20
whis : 3
-----------------------------------------
min_threshold: -1.2439444704105909
max_threshold: 1.1650536888539658
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 13181
-----------------------------------------
min_threshold: -3.104867563569706
max_threshold: 3.766108632852359
Num_of_values for 1 : 473
Num_of_outliers for 1 : 10
-----------------------------------------
Column_name : v21
whis : 3
-----------------------------------------
min_threshold: -1.470033572364138
max_threshold: 1.427098300597629
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 6761
-----------------------------------------
min_threshold: -3.4663407471882692
max_threshold: 4.686969254757649
Num_of_values for 1 : 473
Num_of_outliers for 1 : 23
-----------------------------------------
Column_name : v22
whis : 3
-----------------------------------------
min_threshold: -3.755353909843568
max_threshold: 3.740752815770414
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 180
-----------------------------------------
min_threshold: -3.936883462190517
max_threshold: 4.031332325372906
Num_of_values for 1 : 473
Num_of_outliers for 1 : 6
-----------------------------------------
Column_name : v23
```

```
whis : 3
-----------------------------------------
min_threshold: -1.088860637925542
max_threshold: 1.075003512755236
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 8047
-----------------------------------------
min_threshold: -2.230500525486926
max_threshold: 2.176278715195732
Num_of_values for 1 : 473
Num_of_outliers for 1 : 20
-----------------------------------------
Column_name : v24
whis : 3
-----------------------------------------
min_threshold: -2.737378019534058
max_threshold: 2.8231225026395528
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 137
-----------------------------------------
min_threshold: -2.592244021764005
max_threshold: 2.437735148316076
Num_of_values for 1 : 473
Num_of_outliers for 1 : 0
-----------------------------------------
Column_name : v25
whis : 3
-----------------------------------------
min_threshold: -2.321374847663067
max_threshold: 2.354388029476602
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 441
-----------------------------------------
min_threshold: -2.672725889220364
max_threshold: 2.8162422493989157
Num_of_values for 1 : 473
Num_of_outliers for 1 : 2
-----------------------------------------
Column_name : v26
whis : 3
-----------------------------------------
min_threshold: -2.027479169487471
max_threshold: 1.9406493204199993
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 513
-----------------------------------------
min_threshold: -2.2388948591668036
max_threshold: 2.371344658363276
Num_of_values for 1 : 473
Num_of_outliers for 1 : 1
-----------------------------------------
Column_name : v27
whis : 3
-----------------------------------------
min_threshold: -0.5548818682696173
max_threshold: 0.5749913671559825
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 16106
-----------------------------------------
min_threshold: -2.525348710414211
```

```
max_threshold: 3.3308448081560513
Num_of_values for 1 : 473
Num_of_outliers for 1 : 23
-----------------------------------------
Column_name : v28
whis : 3
-----------------------------------------
min_threshold: -0.44511299011134864
max_threshold: 0.47026575301826845
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 12308
-----------------------------------------
min_threshold: -1.5060693915192813
max_threshold: 1.7812392937319077
Num_of_values for 1 : 473
Num_of_outliers for 1 : 3
-----------------------------------------
Column_name : amount
whis : 3
-----------------------------------------
min_threshold: -209.7
max_threshold: 292.83
Num_of_values for 0 : 283253
Num_of_outliers for 0 : 18762
-----------------------------------------
min_threshold: -313.67
max_threshold: 420.56
Num_of_values for 1 : 473
Num_of_outliers for 1 : 41
-----------------------------------------
```



In [42]:
```
for col in df_out.columns[:30]:
    remove_outliers(df_out, col, whis = 3)
```



In [43]:
```
cprint("\nMissing value control after outlier check \n", 'black', 'on_white', attrs =
missing_values(df_out)
```

Missing value control after outlier check

Out[43]:

	Missing_Number	Missing_Percent
amount	18803	0.066
v27	16129	0.057
v20	13191	0.046
v28	12311	0.043
v8	11085	0.039
v23	8067	0.028
v21	6784	0.024
v6	5395	0.019
v2	4342	0.015
v7	2831	0.010
v10	2806	0.010
v5	2180	0.008
v14	1969	0.007
v1	1889	0.007
v17	1093	0.004
v12	987	0.003
v9	956	0.003
v3	613	0.002
v19	601	0.002
v26	514	0.002
v25	443	0.002
v16	340	0.001
v4	306	0.001
v22	186	0.001
v24	137	0.000
v18	118	0.000
v11	26	0.000
v15	18	0.000
v13	2	0.000

In [44]:
```
df_out.dropna(inplace=True)
cprint("\nMissing value control after outlier check \n", 'black', 'on_white', attrs =
missing_values(df_out)
```

```
Missing value control after outlier check
```

Out[44]:

	Missing_Number	Missing_Percent

In [45]:
```
cprint("\nShape of new df after outlier check \n", 'black', 'on_white', attrs = ['bold
df_out.shape
```

Shape of new df after outlier check

Out[45]:
```
(230994, 31)
```

In [46]:
```
cprint("\nValue of 'class' column after outlier check \n", 'black', 'on_white', attrs
df_out['class'].value_counts(dropna= False)
```

Value of 'class' column after outlier check

Out[46]:
```
class
0    230640
1       354
Name: count, dtype: int64
```

In [47]:
```
cprint("\nModel building without smote\n", 'green', 'on_white', attrs = ['bold'])
cprint("\nData Pre_Processing\n", 'black', 'on_blue', attrs = ['bold'])
```

Model building without smote

Data Pre_Processing

In [48]:
```
def eval(model, X_train, X_test):
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)

    print(confusion_matrix(y_test, y_pred))
    print("Test_Set")
    print(classification_report(y_test,y_pred))
    print("Train_Set")
    print(classification_report(y_train,y_pred_train))
    print("---"*20)
    from_predictions(model, X_test, y_test, cmap="plasma")
def from_predictions(model, X, y, cmap="plasma"):
    y_pred = model.predict(X)
    cm = confusion_matrix(y, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap=cmap)
    plt.xlabel("Predicted labels")
    plt.ylabel("True labels")
    plt.title("Confusion Matrix")
    plt.show()
```

In [49]:
```
def train_val(y_train, y_train_pred, y_test, y_pred):

    scores = {"train_set": {"Accuracy" : accuracy_score(y_train, y_train_pred),
                            "Precision" : precision_score(y_train, y_train_pred),
                            "Recall" : recall_score(y_train, y_train_pred),
                            "f1" : f1_score(y_train, y_train_pred),
```

```
                            "roc_auc" : roc_auc_score(y_train, y_train_pred),
                            "recall_auc" : auc(recall, precision)},

                "test_set": {"Accuracy" : accuracy_score(y_test, y_pred),
                             "Precision" : precision_score(y_test, y_pred),
                             "Recall" : recall_score(y_test, y_pred),
                             "f1" : f1_score(y_test, y_pred),
                             "roc_auc" : roc_auc_score(y_test, y_pred),
                             "recall_auc" : auc(recall, precision)}}

    return pd.DataFrame(scores)
```

In [50]:
```
cprint("\nScaling\n", 'green', 'on_yellow', attrs = ['bold'])
```

Scaling

In [51]:
```
df_ml = df_out.copy()
scaler = StandardScaler()

df_ml["amount"] = scaler.fit_transform(df_ml["amount"].values.reshape(-1,1))
df_ml["time"] = scaler.fit_transform(df_ml["time"].values.reshape(-1,1))
```

In [52]:
```
cprint("\nTrain-Test Split\n", 'green', 'on_yellow', attrs = ['bold'])
```

Train-Test Split

In [53]:
```
X = df_ml.drop(['class'], axis = 1)
y = df_ml['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.
```

In [54]:
```
print('X_train.shape : ', X_train.shape)
print('X_test.shape  : ', X_test.shape)
```

```
X_train.shape :  (184795, 30)
X_test.shape  :  (46199, 30)
```

In [55]:
```
cprint('y_train.value_counts', 'black', 'on_white', attrs = ['bold'])
y_train.value_counts()
```

y_train.value_counts

Out[55]:
```
class
0    184512
1       283
Name: count, dtype: int64
```

In [56]:
```
cprint('\nLogistic Regression without Smote\n', 'black', 'on_blue', attrs = ['bold'])
cprint('\nModel Building\n', 'green', 'on_yellow', attrs = ['bold'])
```

Logistic Regression without Smote

Model Building

In [57]:
```
LogReg_model = LogisticRegression(solver='liblinear', class_weight = 'balanced', randc
LogReg_model.fit(X_train, y_train)
y_pred = LogReg_model.predict(X_test)
y_train_pred = LogReg_model.predict(X_train)
```

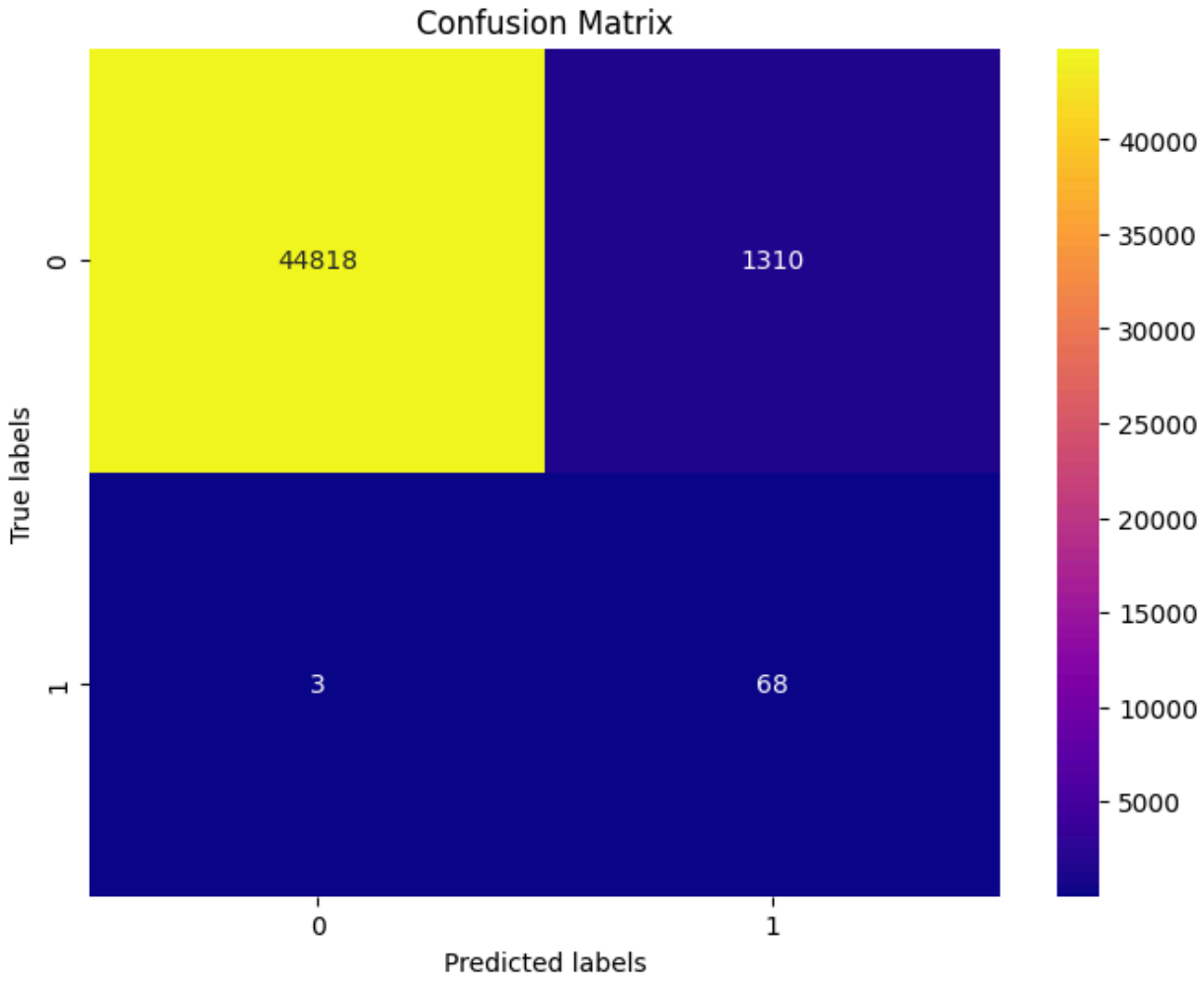
In [58]:
```
cprint('\nEvaluating Model Performance\n', 'green','on_yellow', attrs = ['bold'])
```

Evaluating Model Performance

In [59]:
```
LogReg_model_f1 = f1_score(y_test, y_pred)
LogReg_model_acc = accuracy_score(y_test, y_pred)
LogReg_model_recall = recall_score(y_test, y_pred)
LogReg_model_auc = roc_auc_score(y_test, y_pred)
LogReg_model_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
LogReg_model_recall_auc = auc(recall, precision)
```

In [60]:
```
print("LogReg_Model")
print ("------------------")
eval(LogReg_model, X_train, X_test)

cprint('\nLogReg Scores\n', 'white', 'on_black', attrs = ['bold'])
train_val (y_train, y_train_pred, y_test, y_pred)
```

```
LogReg_Model
------------------
[[44818  1310]
 [    3    68]]
Test_Set
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     46128
           1       0.05      0.96      0.09        71

    accuracy                           0.97     46199
   macro avg       0.52      0.96      0.54     46199
weighted avg       1.00      0.97      0.98     46199

Train_Set
              precision    recall  f1-score   support

           0       1.00      0.97      0.99    184512
           1       0.05      0.93      0.09       283

    accuracy                           0.97    184795
   macro avg       0.52      0.95      0.54    184795
weighted avg       1.00      0.97      0.98    184795

------------------------------------------------------------
```



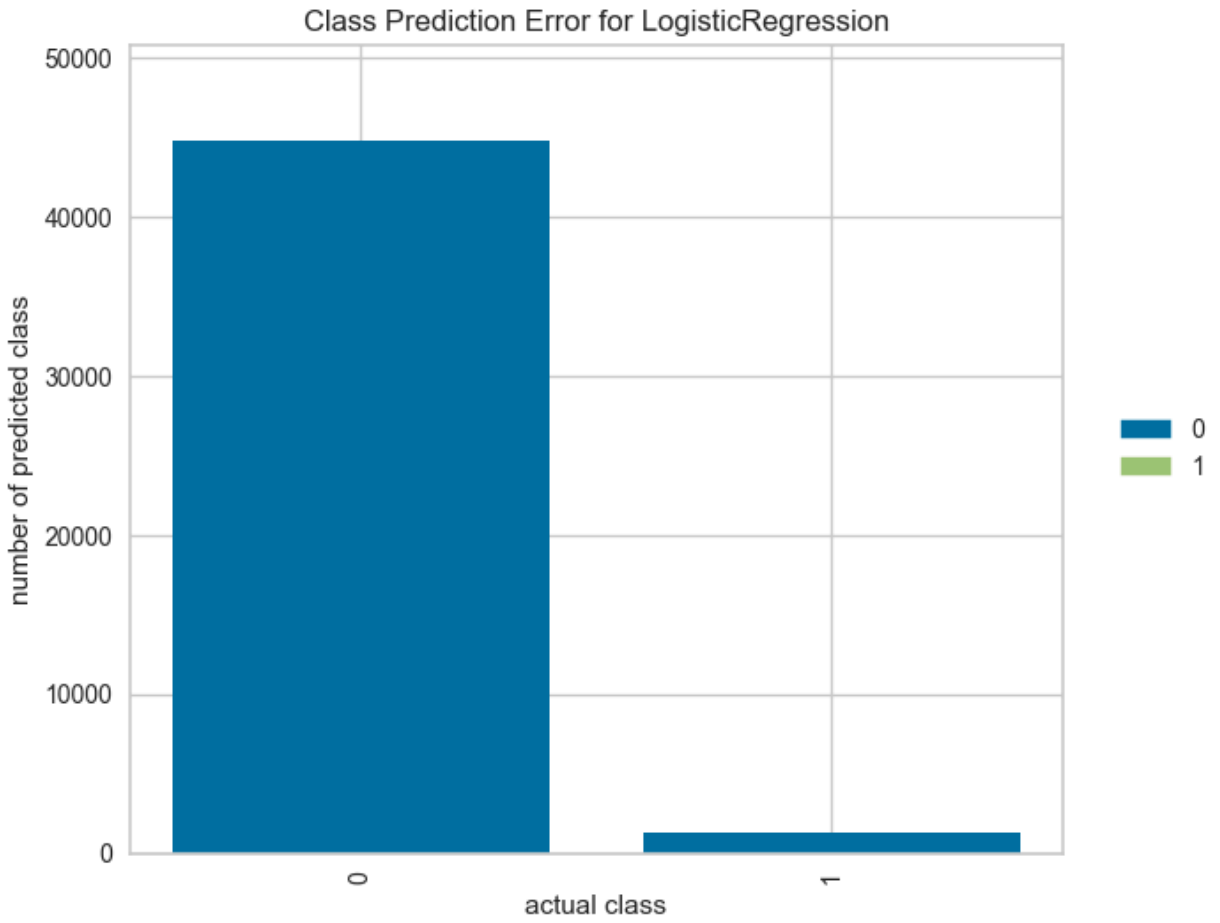
LogReg Scores

Out[60]:

	train_set	test_set
Accuracy	0.973	0.972
Precision	0.050	0.049
Recall	0.929	0.958
f1	0.094	0.094
roc_auc	0.951	0.965
recall_auc	0.504	0.504

In [61]:

```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(LogReg_model)
#fit the trinig data to visiualize
visualizer.fit (X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
#draw visulization
visualizer.poof();
```



In [62]:
```
cprint('\nLogistic Regression Cross Validation\n', 'green', 'on_yellow', attrs = ['bol
```

Logistic Regression Cross Validation

In [63]:
```
cprint('\nCross Validation scores for logistic regression with defult parmeters\n','wh
LogReg_cv = LogisticRegression(solver='liblinear', class_weight='balanced', random_sta
LogReg_cv_scores = cross_validate(LogReg_cv, X_train, y_train, scoring=['accuracy', 'p

# Printing cross-validation scores
#print("Accuracy:", LogReg_cv_scores['test_accuracy'].mean()[2:])
#print("Precision:", LogReg_cv_scores['test_precision'].mean()[2:])
#print("Recall:", LogReg_cv_scores['test_recall'].mean()[2:])
#print("F1 Score:", LogReg_cv_scores['test_f1'].mean()[2:])
#print("ROC AUC Score:", LogReg_cv_scores['test_roc_auc'].mean()[2:])
LogReg_cv_scores = pd.DataFrame(LogReg_cv_scores, index = range(1, 11))
LogReg_cv_scores.mean()[2:]
```

Cross Validation scores for logistic regression with defult parmeters

Out[63]:
```
test_accuracy    0.973
test_precision   0.048
test_recall      0.901
test_f1          0.092
test_roc_auc     0.978
dtype: float64
```

In [64]:
```
cprint('\nLogistic Regression RandomizedSearchCV\n', 'green', 'on_yellow',attrs= ['bol
```

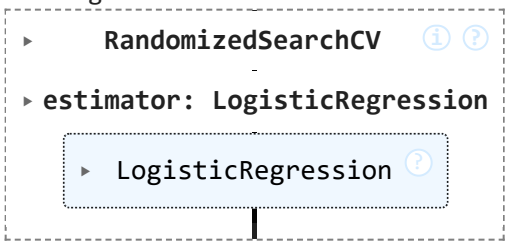
Logistic Regression RandomizedSearchCV

In [65]:
```
param_grid = {
    "class_weight": [None, "balanced"],
    "penalty": ["l1", "l2"],
    "solver": ['liblinear', 'lbfgs']
}
```

In [244…
```
# Initialize Logistic Regression with default parameters
LogReg_grid = LogisticRegression(random_state=42)

# Perform Grid Search vs RandomizedSearchCV
LogReg_grid_model = RandomizedSearchCV(LogReg_grid, param_grid, scoring="f1", verbose=
LogReg_grid_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

Out[244]:



In [245…
```
print(colored('\033[1mBest Estimator of RandomizedSearchCV for Logistic Regression Mod
```

```
Best Estimator of RandomizedSearchCV for Logistic Regression Model: LogisticRegressio
n(random_state=42, solver='liblinear')
```

In [246…
```
print(colored('\033[1mBest Parameters of RandomizedSearchCV for Logistic Regression Mo
```

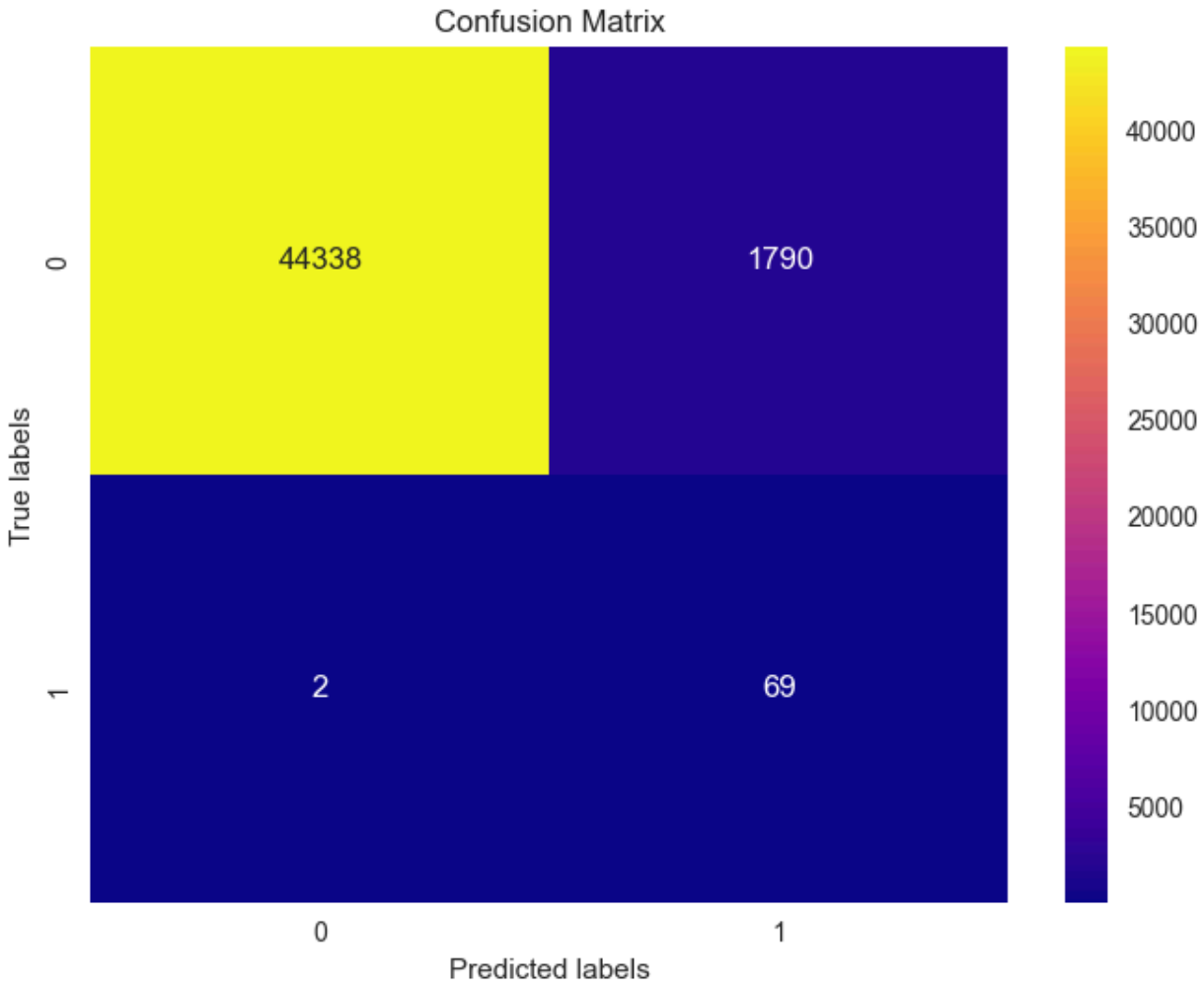
```
Best Parameters of RandomizedSearchCV for Logistic Regression Model: {'solver': 'libl
inear', 'penalty': 'l2', 'class_weight': None}
```

In [66]:
```
LogReg_tuned = LogisticRegression(class_weight = 'balanced', penalty = 'l1',solver = '
```

In [67]:
```
y_pred = LogReg_tuned.predict(X_test)
y_train_pred = LogReg_tuned.predict(X_train)

LogReg_tuned_f1 = f1_score(y_test, y_pred)
LogReg_tuned_acc = accuracy_score(y_test, y_pred)
LogReg_tuned_recall = recall_score(y_test, y_pred)
LogReg_tuned_auc = roc_auc_score(y_test, y_pred)
LogReg_tuned_pre = precision_score(y_test, y_pred)
LogReg_tuned_recall_auc = auc(recall, precision)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
LogReg_tuned_recall_auc = auc(recall, precision)
```

In [68]:
```
print("LogReg_tuned")
print("------------")
eval(LogReg_tuned, X_train, X_test)
```

```
LogReg_tuned
------------
[[44338  1790]
 [    2    69]]
Test_Set
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     46128
           1       0.04      0.97      0.07        71

    accuracy                           0.96     46199
   macro avg       0.52      0.97      0.53     46199
weighted avg       1.00      0.96      0.98     46199

Train_Set
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    184512
           1       0.04      0.94      0.07       283

    accuracy                           0.96    184795
   macro avg       0.52      0.95      0.53    184795
weighted avg       1.00      0.96      0.98    184795

------------------------------------------------------------
```



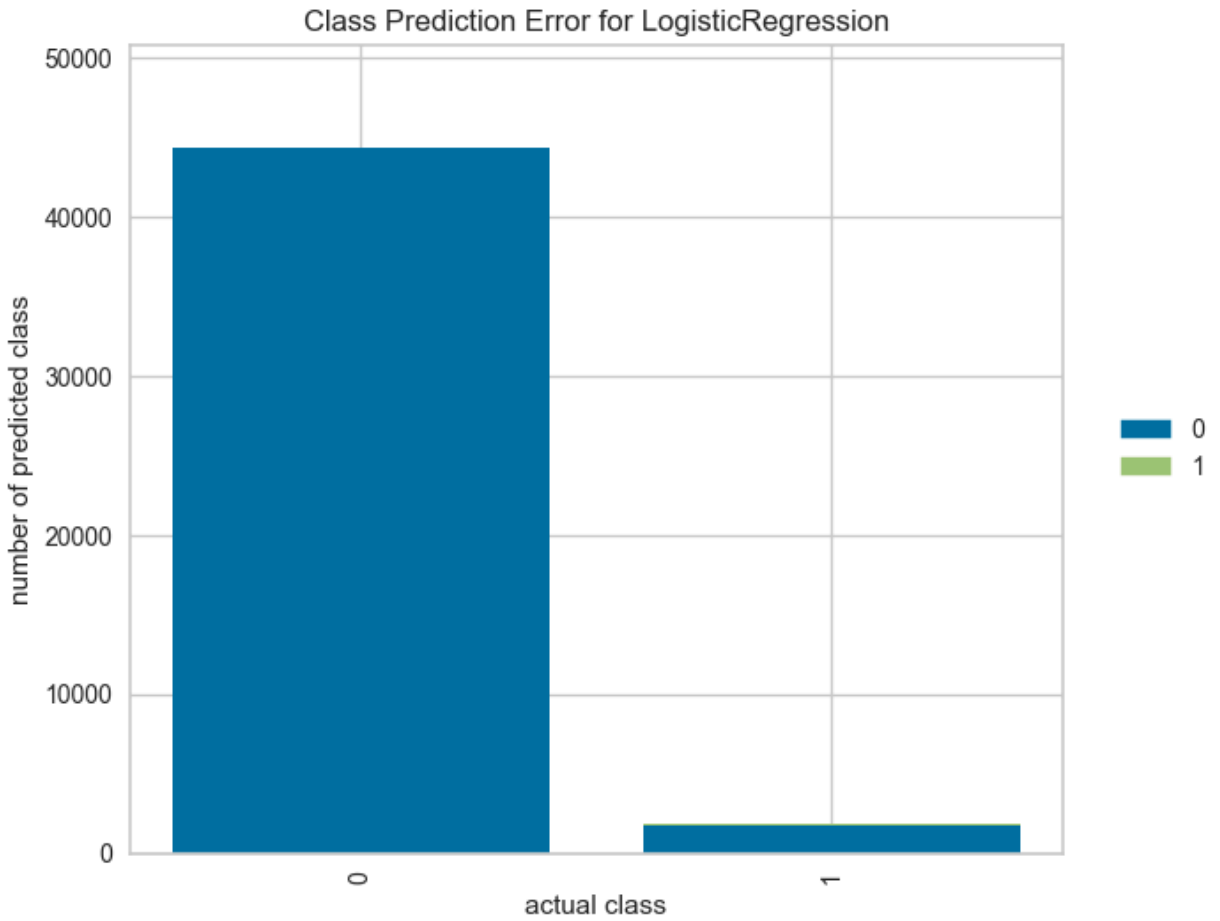
In [69]:
```
cprint('\nLogistic Regression_tuned Scores\n', 'black', 'on_white', attrs=['bold'])
train_val(y_train, y_train_pred, y_test, y_pred)
```

Logistic Regression_tuned Scores

Out[69]:

	train_set	test_set
Accuracy	0.962	0.961
Precision	0.037	0.037
Recall	0.940	0.972
f1	0.071	0.072
roc_auc	0.951	0.967
recall_auc	0.504	0.504

In [70]:
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(LogReg_tuned)
#fit the trining data to the visualizer
visualizer.fit(X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
#Draw visualization
visualizer.poof()
```



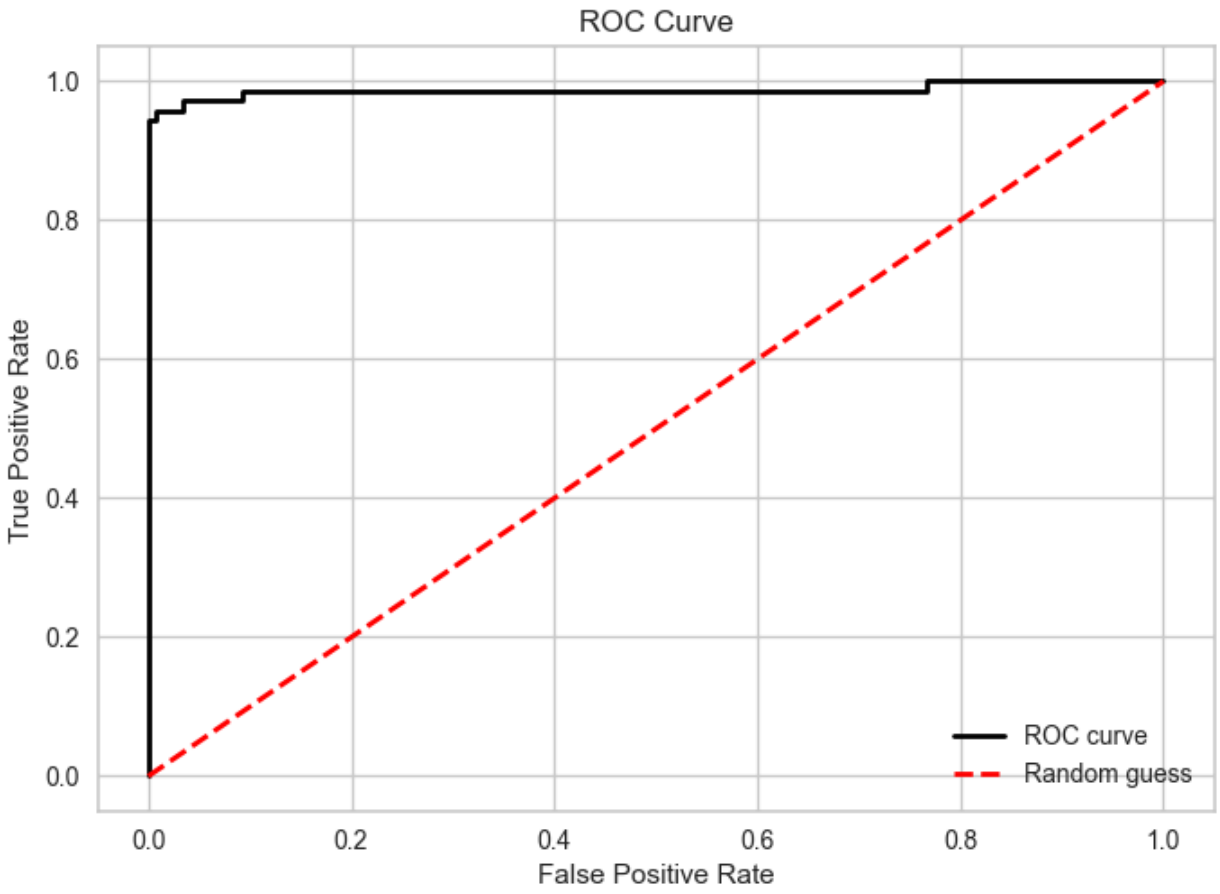
Out[70]:
```
<Axes: title={'center': 'Class Prediction Error for LogisticRegression'}, xlabel='act
ual class', ylabel='number of predicted class'>
```

In [71]:
```
cprint('\nLogistic Regression Roc (Receiver operating Curve) and AUC(Area Under curve)
cprint('\nroc_curve\n', 'black', 'on_white',attrs = ['bold'])
```

```
# Compute ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, LogReg_tuned.predict_proba(X_test)[:,1])

# Plot ROC curve
plt.plot(fpr, tpr, color='black', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

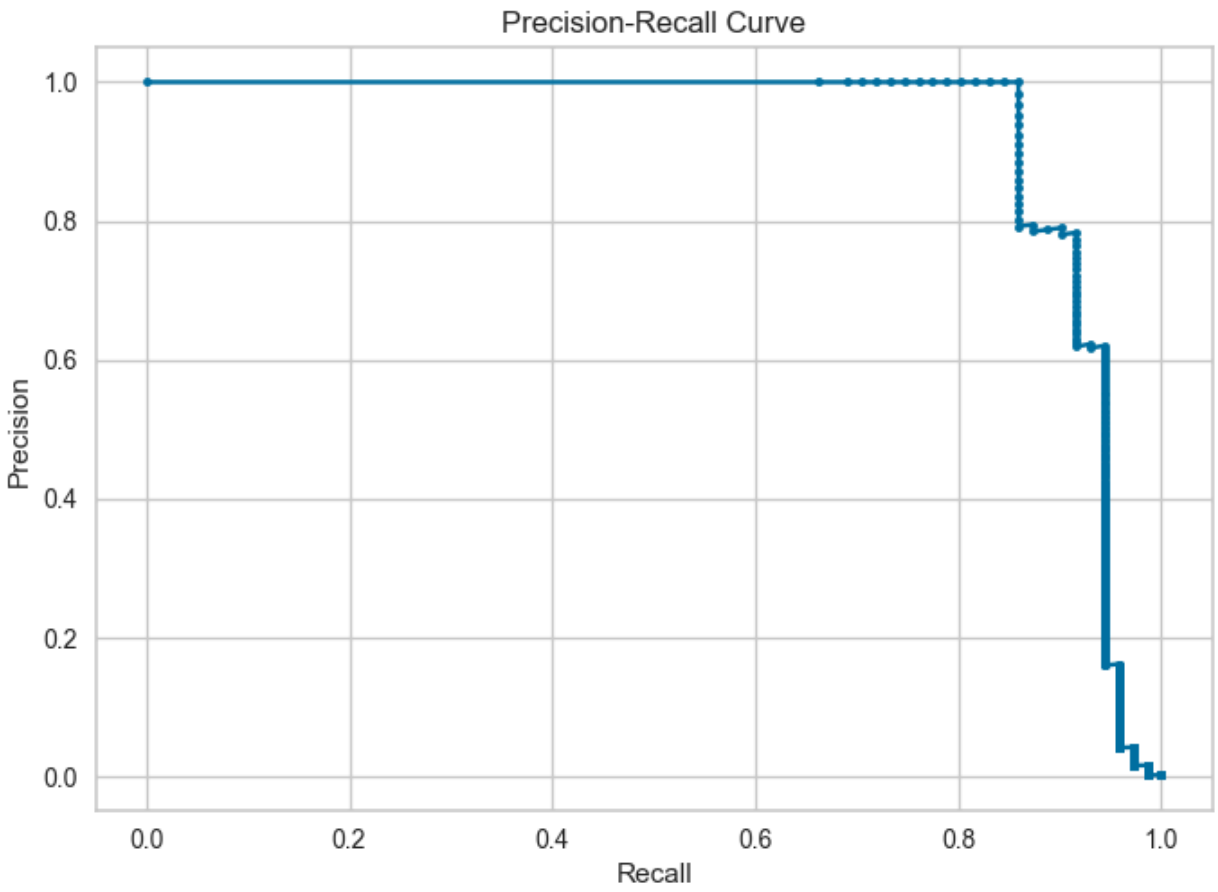
Logistic Regression Roc (Receiver operating Curve) and AUC(Area Under curve)

roc_curve



In [72]:
```
cprint('\nprecision_recall_curve\n', 'black', 'on_white',attrs=['bold'])
# Compute precision-recall pairs
precision, recall, _ = precision_recall_curve(y_test, LogReg_tuned.predict_proba(X_tes

## Plot precision-recall curve
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```

precision_recall_curve



In [73]:
```
cprint('\nLogReg_tuned Predictions\n','black', 'on_white',attrs=['bold'])
LogReg_Pred = {"Actual": y_test, "LogReg_Pred":y_pred}
LogReg_Pred = pd.DataFrame.from_dict(LogReg_Pred)
LogReg_Pred.sample(5)
```

LogReg_tuned Predictions

Out[73]:

	Actual	LogReg_Pred
6422	0	0
283870	0	1
19739	0	0
52233	0	0
162159	0	0

In [74]:
```
pd.crosstab(LogReg_Pred['Actual'], LogReg_Pred['LogReg_Pred'])
```

Out[74]:

LogReg_Pred	0	1
Actual		
0	44338	1790
1	2	69

In [75]:
```
cprint('\nPrediction\n', 'green', 'on_yellow', attrs=['bold'])
Model_Preds = LogReg_Pred
Model_Preds.sample(5)
```

Prediction

Out[75]:

	Actual	LogReg_Pred
92386	0	0
271263	0	0
271569	0	0
270647	0	0
112638	0	0

In [76]:
```
#LogisticRegresson = pickle.dump(LogReg_tuned, open('LogisticRegression_Model(tuned)',
```

In [77]:
```
cprint('\nRandom Forest Classifier without Smote\n','black', 'on_blue', attrs = ['bold
cprint('\nModel building with Random Forest Classifier\n','green','on_yellow', attrs =
RF_model = RandomForestClassifier(class_weight = 'balanced', random_state = 42)
RF_model.fit(X_train, y_train)
y_pred = RF_model.predict(X_test)
y_train_pred = RF_model.predict(X_train)
```

Random Forest Classifier without Smote

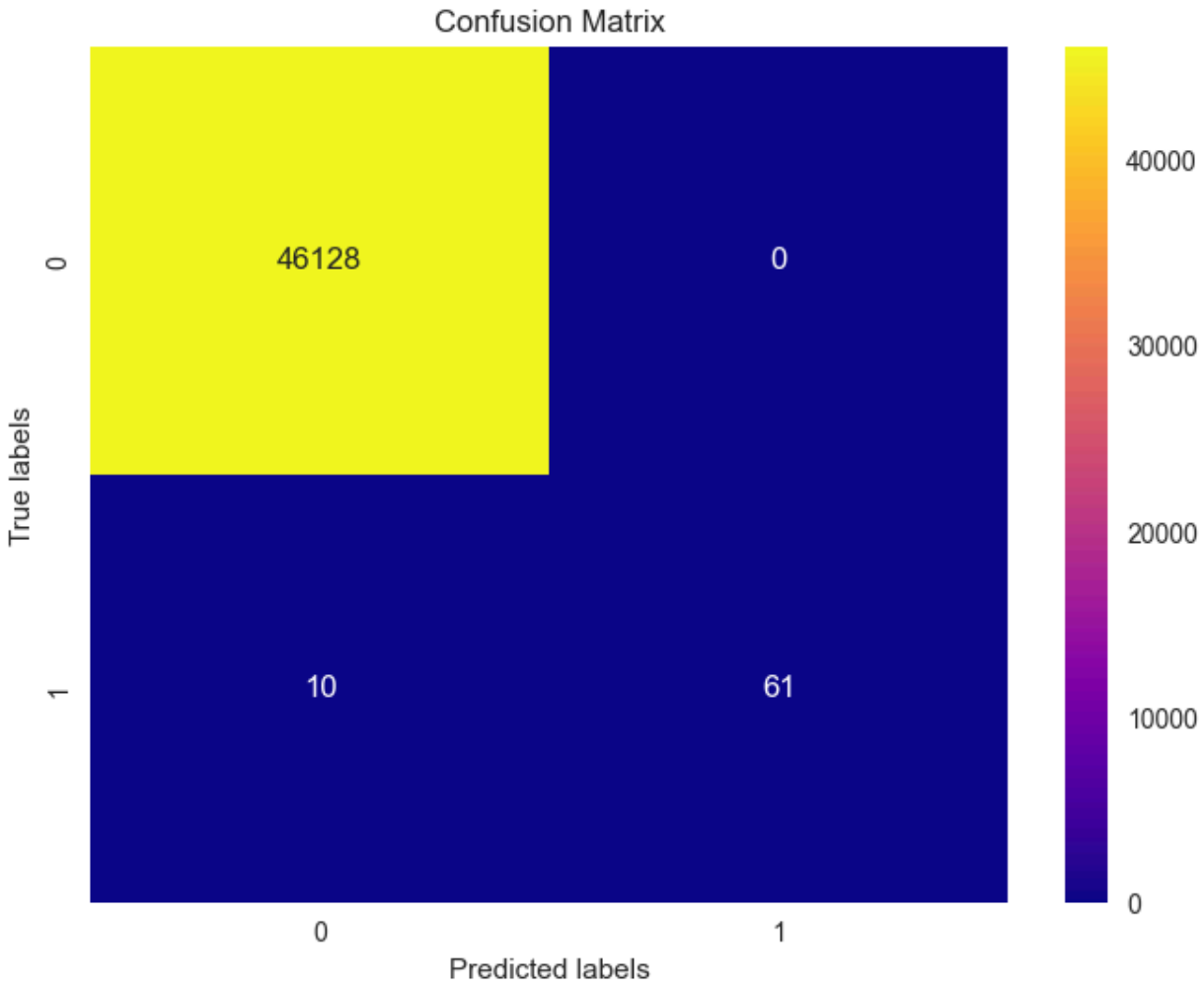
Model building with Random Forest Classifier

In [78]:
```
cprint('\nEvaluating model Performance\n', 'green', 'on_yellow', attrs=['bold'])
RF_model_f1 = f1_score(y_test, y_pred)
RF_model_acc = accuracy_score(y_test, y_pred)
RF_model_recall = recall_score(y_test, y_pred)
RF_model_auc = roc_auc_score(y_test, y_pred)
RF_model_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
RF_model_recall_auc = auc(recall, precision)
```

Evaluating model Performance

In [79]:
```
print("RF_Model")
print("-----------")
eval(RF_model, X_train, X_test)
```

```
RF_Model
-----------
[[46128     0]
 [   10    61]]
Test_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     46128
           1       1.00      0.86      0.92        71

    accuracy                           1.00     46199
   macro avg       1.00      0.93      0.96     46199
weighted avg       1.00      1.00      1.00     46199

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    184512
           1       1.00      1.00      1.00       283

    accuracy                           1.00    184795
   macro avg       1.00      1.00      1.00    184795
weighted avg       1.00      1.00      1.00    184795

------------------------------------------------------------
```



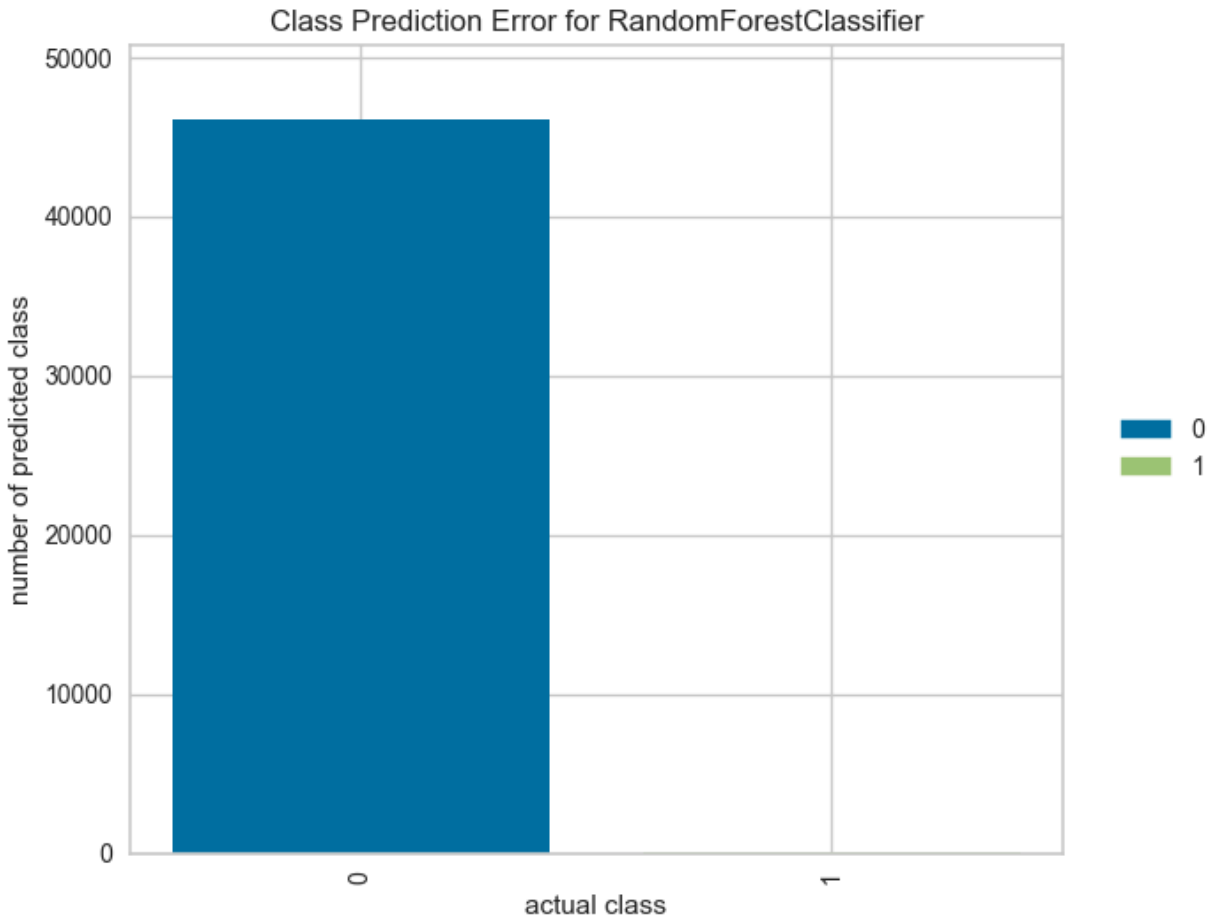
In [80]:
```
cprint('\nRF_Model Score\n', 'black', 'on_white', attrs = ['bold'])
train_val(y_train, y_train_pred, y_test, y_pred)
```

RF_Model Score

Out[80]:

	train_set	test_set
Accuracy	1.000	1.000
Precision	1.000	1.000
Recall	1.000	0.859
f1	1.000	0.924
roc_auc	1.000	0.930
recall_auc	0.930	0.930

In [81]:
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(RF_model)
#fit the trining data to the visualizer
visualizer.fit(X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
#Draw visualization
visualizer.poof()
```



Out[81]:
```
<Axes: title={'center': 'Class Prediction Error for RandomForestClassifier'}, xlabel
='actual class', ylabel='number of predicted class'>
```

In [82]:
```
cprint('\nFeature Importance for Random Forest Model\n', 'green', 'on_yellow', attrs =
RF_feature_imp  = pd.DataFrame(index=X.columns, data=RF_model.feature_importances_, cc
```

```
RF_feature_imp
```

Feature Importance for Random Forest Model

Out[82]:

	Importance
v14	0.191
v4	0.106
v10	0.102
v12	0.098
v17	0.077
v2	0.055
v3	0.051
v11	0.051
v16	0.031
v7	0.029
v9	0.024
v21	0.022
v19	0.018
v20	0.016
v8	0.014
v27	0.013
amount	0.012
v1	0.010
v28	0.009
v23	0.008
v18	0.008
v15	0.008
v26	0.007
v6	0.007
v5	0.006
v22	0.006
v13	0.006
v25	0.005
v24	0.005
time	0.004

In [83]:
```
fig= px.bar(RF_feature_imp.sort_values('Importance', ascending = False),x = RF_feature
fig.show()
```

In [84]:
```
cprint('\nRandom Forest Cross Validation\n', 'green', 'on_yellow', attrs = ['bold'])
cprint('\nCross VAlidation Score for Random Forest with defult parameters\n', 'black',
RF_cv = RandomForestClassifier(class_weight = 'balanced', random_state = 42)
RF_cv_scores = cross_validate(RF_cv, X_train, y_train, scoring = ['accuracy', 'precisi
RF_cv_scores = pd.DataFrame(RF_cv_scores, index = range(1, 11))
RF_cv_scores.mean()[2:]
```

Random Forest Cross Validation

Cross VAlidation Score for Random Forest with defult parameters

Out[84]:
```
test_accuracy    1.000
test_precision   1.000
test_recall      0.774
test_f1          0.871
test_roc_auc     0.949
dtype: float64
```

In [85]:
```
cprint('\nRandom Forest Classifier RandomizedSearchCV\n', 'green', 'on_yellow', attrs
RF_grid = RandomForestClassifier(class_weight='balanced', random_state=42)
```

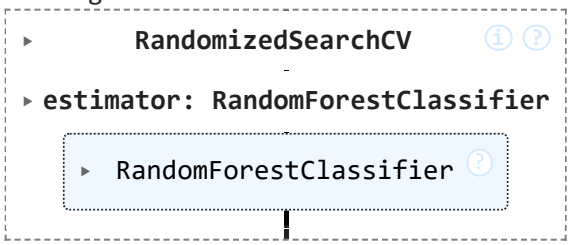
```
RF_param_distributions = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

Random Forest Classifier RandomizedSearchCV

In [247…
```
RF_grid_model = RandomizedSearchCV(
    estimator=RF_grid,
    param_distributions=RF_param_distributions,
    scoring="recall",
    n_jobs=-1,
    verbose=2
)

RF_grid_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[247]:



In [248…
```
print(colored('\033[1mBest Estimator of GridSearchCV for Random Forest Model:\033[0m',
```

```
Best Estimator of GridSearchCV for Random Forest Model: RandomForestClassifier(class_
weight='balanced', min_samples_leaf=2,
                       n_estimators=200, random_state=42)
```

In [249…
```
print(colored('\033[1mBest Parameters of GridSearchCV for Random Forest Model:\033[0m'
```

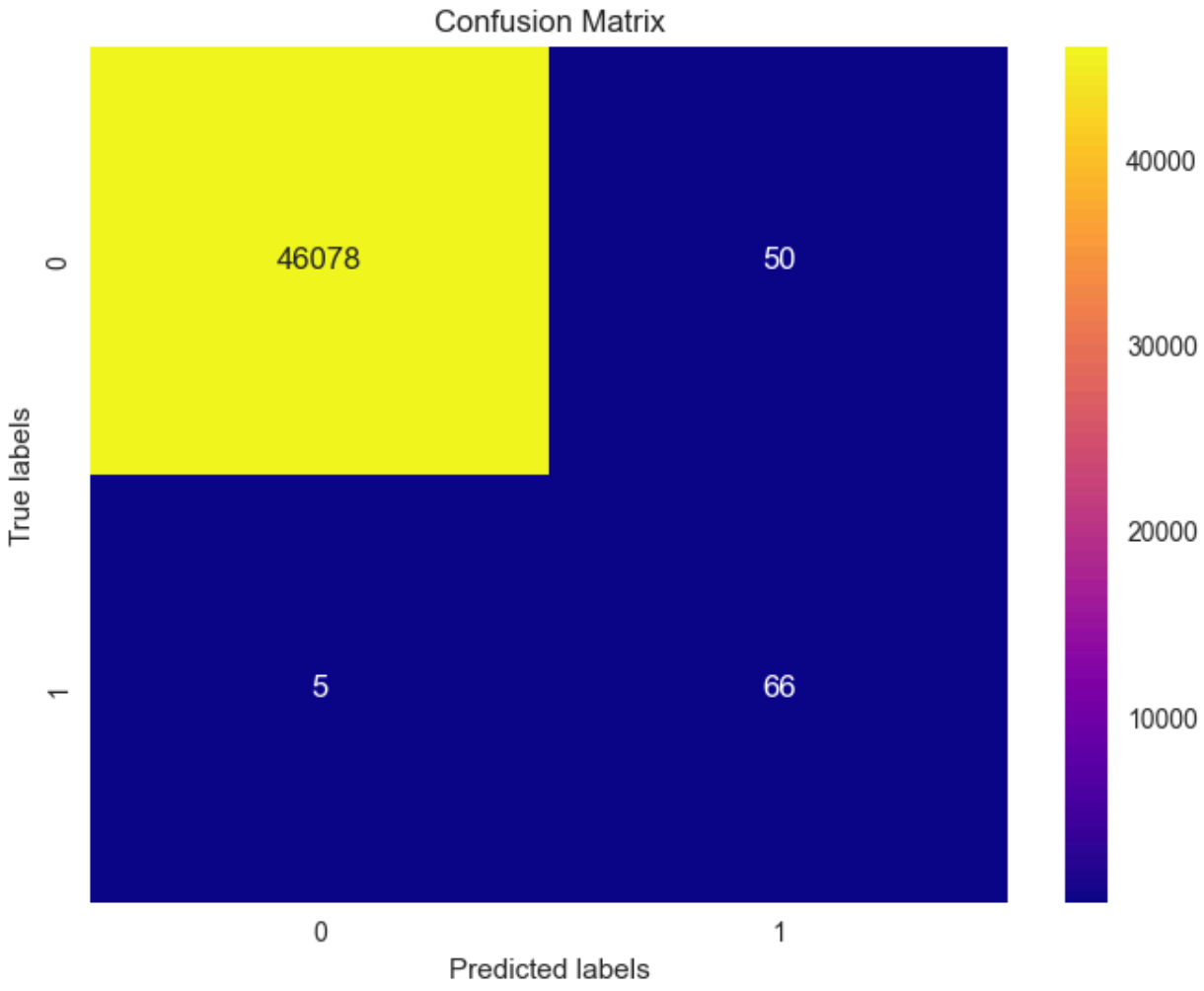
```
Best Parameters of GridSearchCV for Random Forest Model: {'n_estimators': 200, 'min_s
amples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': None}
```

In [86]:
```
RF_tuned = RandomForestClassifier (class_weight = 'balanced', max_depth = 5, max_featu
```

In [87]:
```
y_pred = RF_tuned.predict(X_test)
y_train_pred = RF_tuned.predict(X_train)

RF_tuned_f1 = f1_score(y_test, y_pred)
RF_tuned_acc = accuracy_score(y_test, y_pred)
RF_tuned_recall = recall_score(y_test, y_pred)
RF_tuned_auc = roc_auc_score(y_test, y_pred)
RF_tuned_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
RF_tuned_recall_auc = auc(recall, precision)
```

In [88]:
```
print("RF_Model")
print("---------")
eval(RF_tuned, X_train, X_test)
```

```
RF_Model
---------
[[46078    50]
 [    5    66]]
Test_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     46128
           1       0.57      0.93      0.71        71

    accuracy                           1.00     46199
   macro avg       0.78      0.96      0.85     46199
weighted avg       1.00      1.00      1.00     46199

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    184512
           1       0.60      0.88      0.71       283

    accuracy                           1.00    184795
   macro avg       0.80      0.94      0.85    184795
weighted avg       1.00      1.00      1.00    184795

------------------------------------------------------------
```



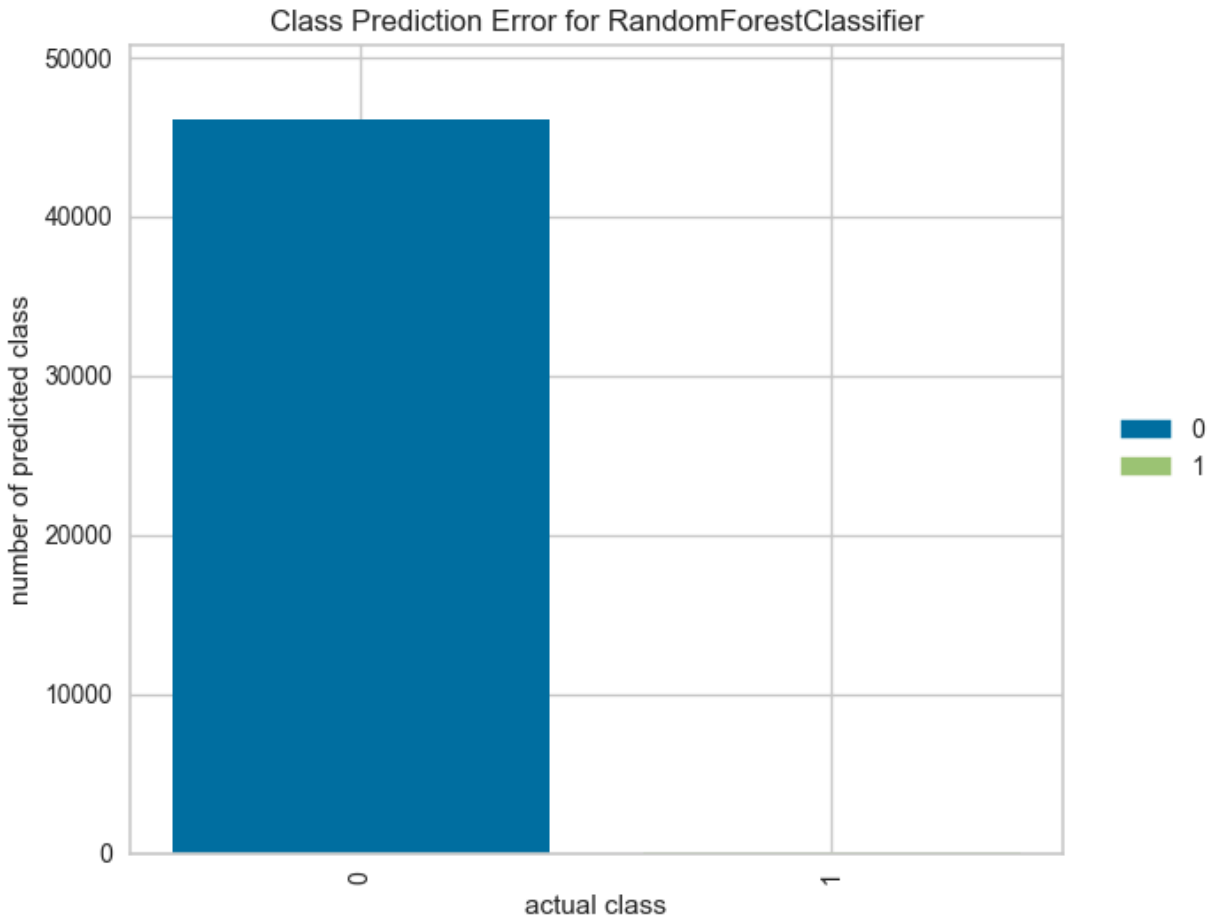
In [89]:
```
cprint('\nRF_tuned score\n', 'black', 'on_white', attrs=['bold'])
train_val(y_train, y_train_pred, y_test, y_pred)
```

RF_tuned score

Out[89]:

	train_set	test_set
Accuracy	0.999	0.999
Precision	0.596	0.569
Recall	0.876	0.930
f1	0.710	0.706
roc_auc	0.938	0.964
recall_auc	0.749	0.749

In [90]:
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(RF_tuned)
#fit the trining data to the visualizer
visualizer.fit(X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
#Draw visualization
visualizer.poof()
```

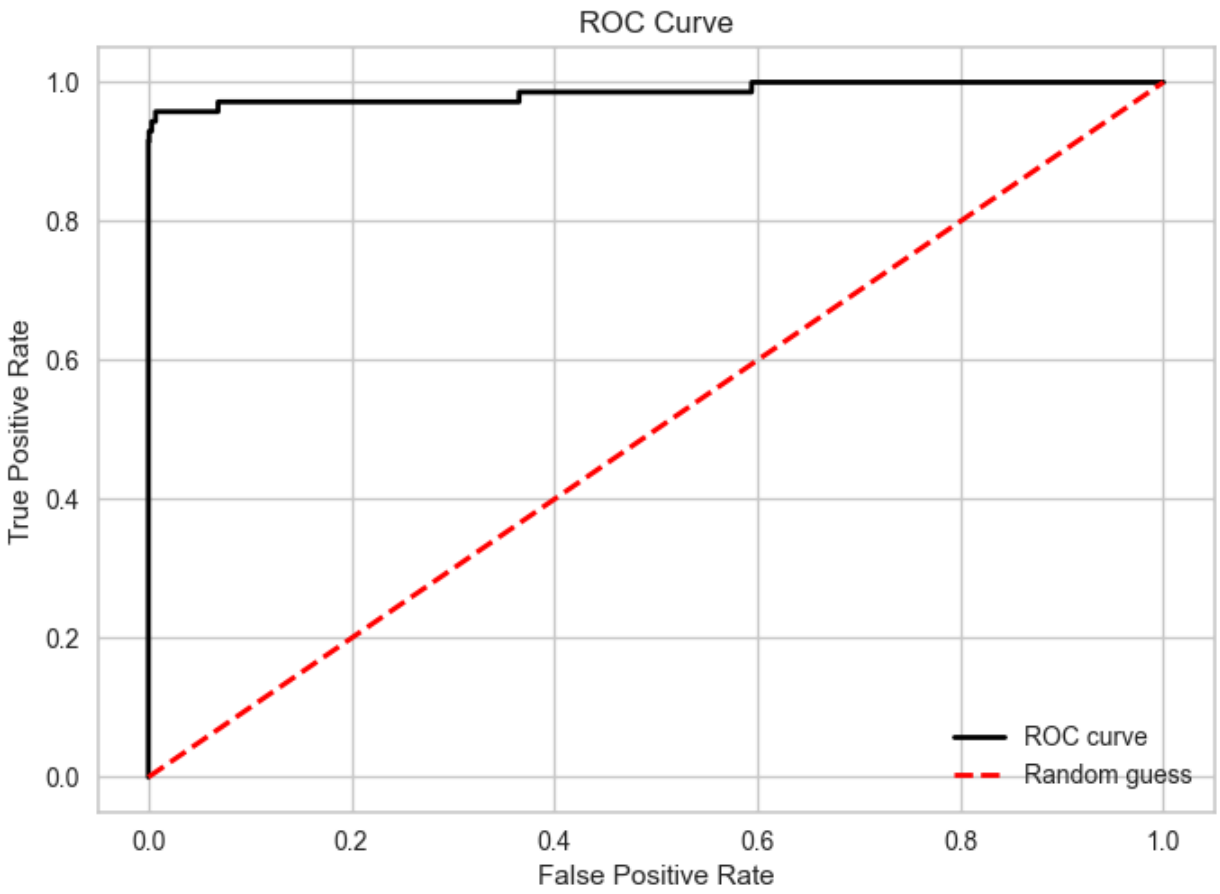


Out[90]:
```
<Axes: title={'center': 'Class Prediction Error for RandomForestClassifier'}, xlabel
='actual class', ylabel='number of predicted class'>
```

In [91]:
```
cprint('\nRandom Forest Roc (Receiver operating Curve) and AUC(Area Undercurve)\n', 'g
# Compute ROC curve points
```

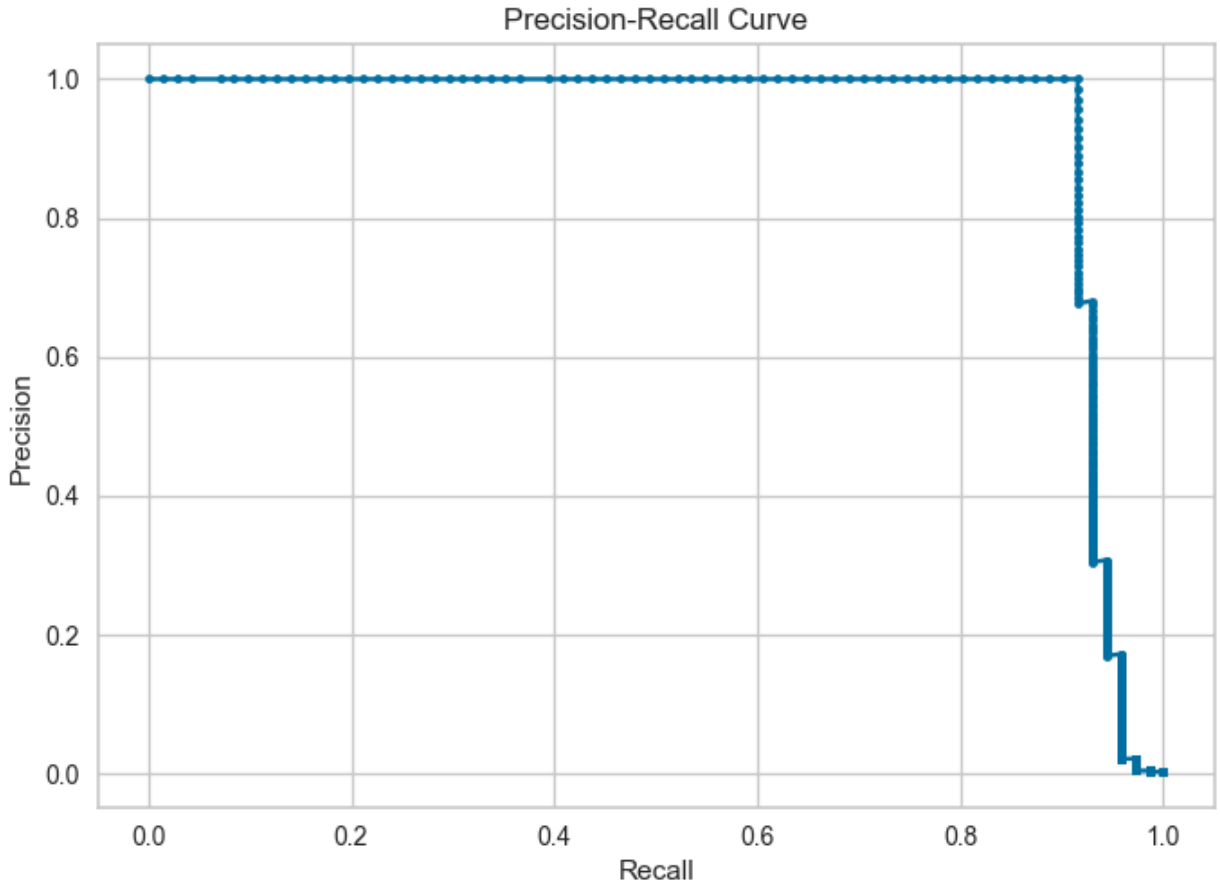
```
fpr, tpr, thresholds = roc_curve(y_test, RF_tuned.predict_proba(X_test)[:,1])

# Plot ROC curve
plt.plot(fpr, tpr, color='black', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

Random Forest Roc (Receiver operating Curve) and AUC(Area Undercurve)



In [92]:
```
cprint('\nprecision_recall_curve\n', 'black', 'on_white',attrs=['bold'])
# Compute precision-recall pairs
precision, recall, _ = precision_recall_curve(y_test, RF_tuned.predict_proba(X_test)[:

## Plot precision-recall curve
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```

precision_recall_curve



In [93]:
```
cprint('\nRandom Forest Prediction\n', 'black', 'on_white', attrs = ['bold'])
RF_Pred = {"Actual": y_test, "RF_Pred": y_pred}
RF_Pred = pd.DataFrame.from_dict(RF_Pred)
RF_Pred.sample(5)
```

```
Random Forest Prediction
```

Out[93]:

	Actual	RF_Pred
264017	0	0
111824	0	0
6281	0	0
225478	0	0
80616	0	0

In [94]:
```
# Assuming RF_Pred is your DataFrame containing 'Actual' and 'RF_Pred' columns
pd.crosstab(RF_Pred['Actual'], RF_Pred['RF_Pred'])
```

Out[94]:

RF_Pred	0	1
Actual		
0	46078	50
1	5	66

In [95]:
```
cprint('\nPrediction\n', 'green', 'on_yellow', attrs = ['bold'])
RF_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, RF_Pred, left_index=True, right_index = True)
Model_Preds.sample(5)
```

Prediction

Out[95]:

	Actual	LogReg_Pred	RF_Pred
29115	0	0	0
94981	0	0	0
15595	0	0	0
106592	0	0	0
281801	0	0	0

In [96]:
```
#random_forest_classifier = pickle.dump(RF_tuned, open('random_forest_model(tuned)','w
```

In [97]:
```
cprint("\nModel building with smote\n", 'green', 'on_white', attrs = ['bold'])
cprint('\nData pre-processing\n', 'black', 'on_blue', attrs=['bold'])
cprint('\nScaling\n','green', 'on_yellow', attrs = ['bold'])
```

Model building with smote

Data pre-processing

Scaling

In [98]:
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
```

In [99]:
```
df_smote = df_out.copy()
scaler = StandardScaler()

df_smote["amount"] = scaler.fit_transform(df_ml["amount"].values.reshape(-1,1))
df_smote["time"] = scaler.fit_transform(df_ml["time"].values.reshape(-1,1))
```

In [100…
```
cprint("\nApplying SMOTE and Train-Test Split\n", 'green', 'on_yellow', attrs = ['bold
```

Applying SMOTE and Train-Test Split

In [101…
```
X = df_smote.drop(['class'], axis = 1)
y = df_smote['class']
```

In [102…
```
over = SMOTE(sampling_strategy = {1: 10000})
under = RandomUnderSampler(sampling_strategy = {0: 10000})
steps = [('o', over), ('u', under)]
```

```
pipeline = Pipeline(steps = steps)
X, y = pipeline.fit_resample(X, y)
```

In [103…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.
```

In [104…
```
print('X_train.shape : ', X_train.shape)
print('X_test.shape  : ', X_test.shape)
```

```
X_train.shape :  (16000, 30)
X_test.shape  :  (4000, 30)
```

In [105…
```
cprint('y_train.value_counts', 'black', 'on_white', attrs = ['bold'])
y_train.value_counts()
```

Out[105]:
y_train.value_counts

```
class
1    8000
0    8000
Name: count, dtype: int64
```

In [106…
```
cprint('y_test.value_counts', 'black', 'on_white', attrs = ['bold'])
y_test.value_counts()
```

Out[106]:
y_test.value_counts

```
class
1    2000
0    2000
Name: count, dtype: int64
```

In [107…
```
cprint('\nLogistic Regression with Smote\n', 'black', 'on_blue', attrs = ['bold'])
cprint('\nModel Building with Logistic Rgression\n', 'green', 'on_yellow', attrs = ['b
```

Logistic Regression with Smote

Model Building with Logistic Rgression

In [108…
```
LogRegSmote_model = LogisticRegression(solver='liblinear', class_weight = 'balanced',
LogRegSmote_model.fit(X_train, y_train)
y_pred = LogRegSmote_model.predict(X_test)
y_train_pred = LogRegSmote_model.predict(X_train)
```

In [109…
```
cprint('\nEvaluating Model Performance\n', 'green','on_yellow', attrs = ['bold'])
```

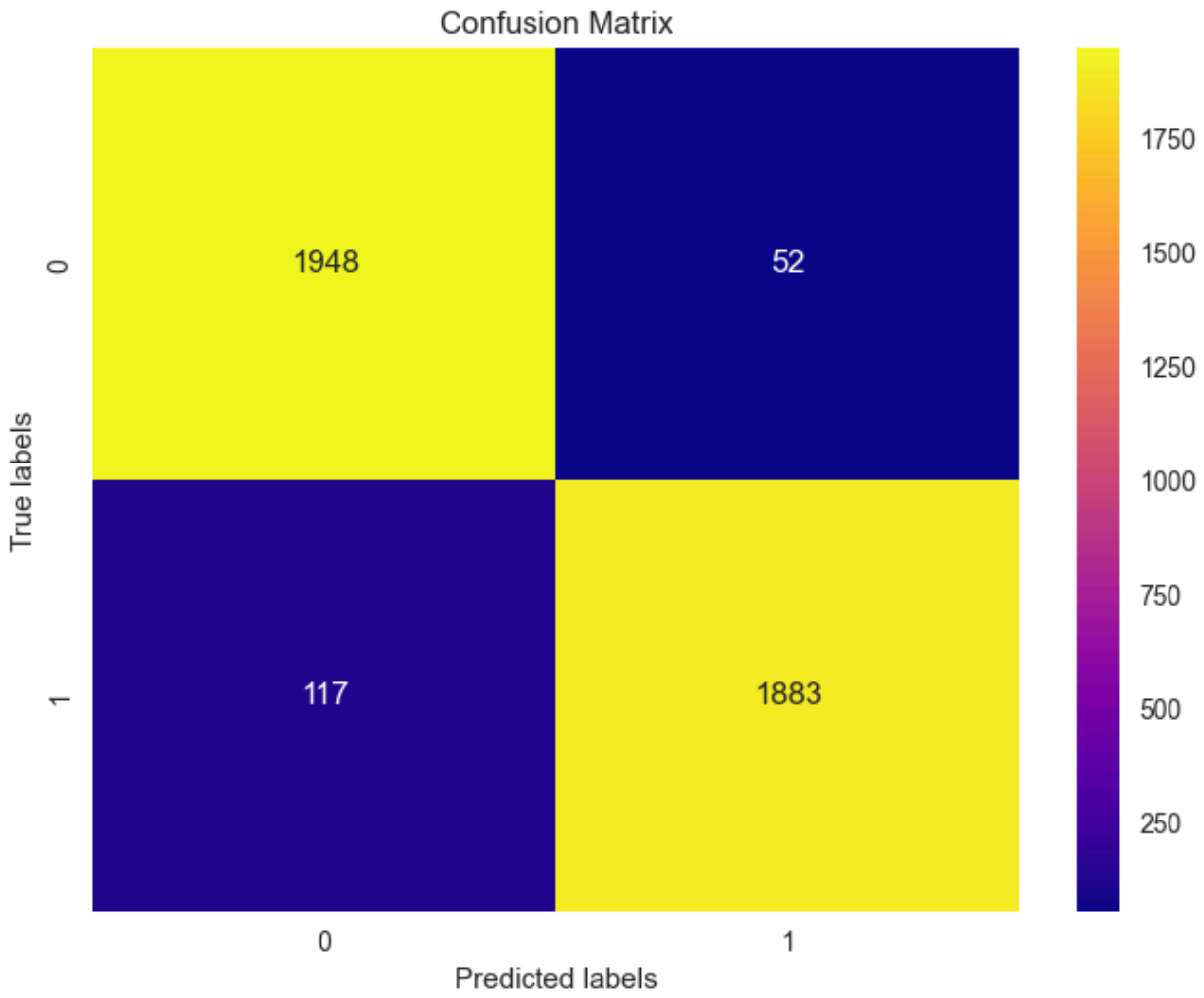
Evaluating Model Performance

In [110…
```
LogRegSmote_model_f1 = f1_score(y_test, y_pred)
LogRegSmote_model_acc = accuracy_score(y_test, y_pred)
LogRegSmote_model_recall = recall_score(y_test, y_pred)
LogRegSmote_model_auc = roc_auc_score(y_test, y_pred)
LogRegSmote_model_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
LogRegSmote_model_recall_auc = auc(recall, precision)
```

In [111…
```
print("LogReg_Model")
print ("------------------")
```

```
eval(LogRegSmote_model, X_train, X_test)
```

```
LogReg_Model
------------------
[[1948   52]
 [ 117 1883]]
Test_Set
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      2000
           1       0.97      0.94      0.96      2000

    accuracy                           0.96      4000
   macro avg       0.96      0.96      0.96      4000
weighted avg       0.96      0.96      0.96      4000

Train_Set
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      8000
           1       0.97      0.93      0.95      8000

    accuracy                           0.95     16000
   macro avg       0.96      0.95      0.95     16000
weighted avg       0.96      0.95      0.95     16000

------------------------------------------------------------
```



In [112…

```
cprint('\nLogReg Smote Model Scores\n', 'white', 'on_black', attrs = ['bold'])
train_val (y_train, y_train_pred, y_test, y_pred)
```

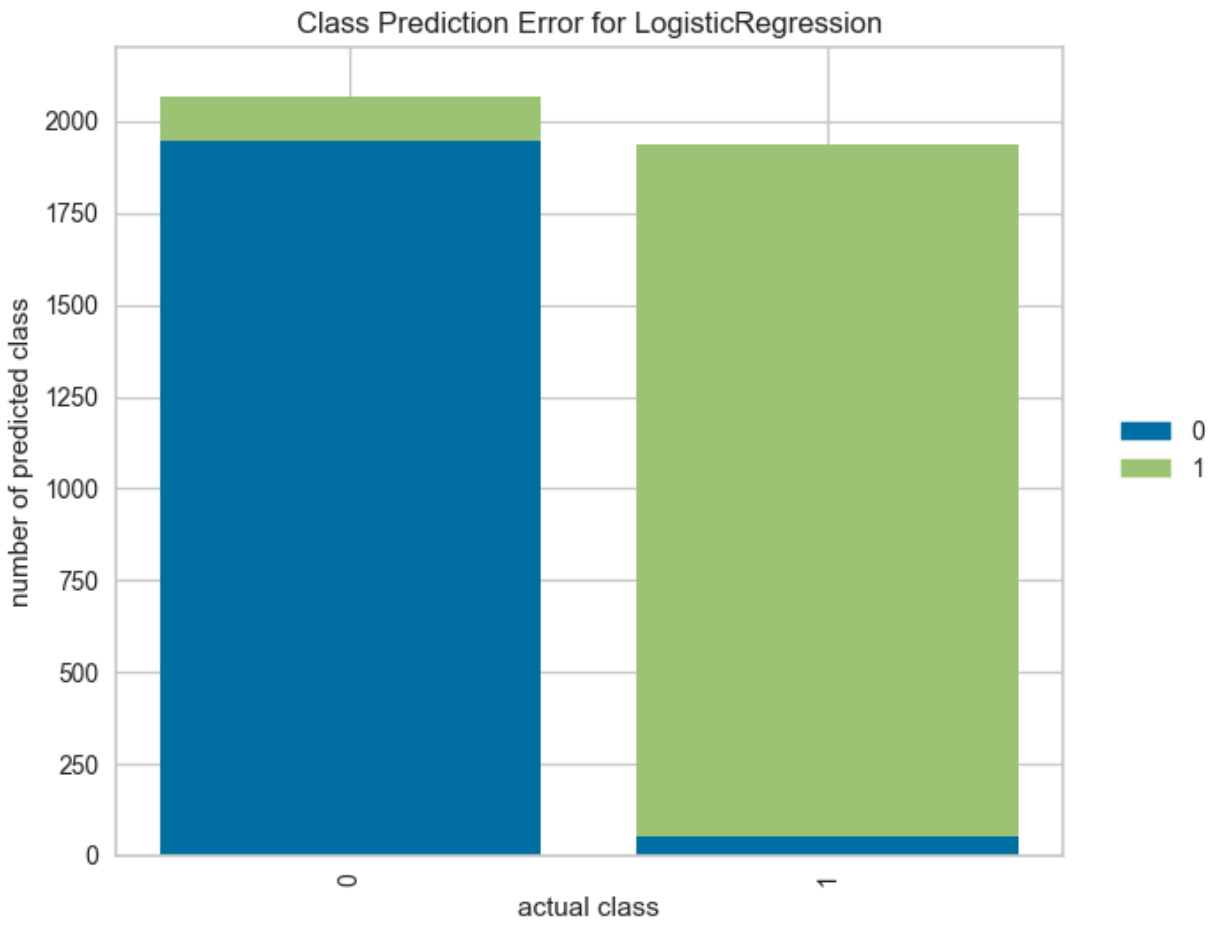
```
LogReg Smote Model Scores
```

Out[112]:

	train_set	test_set
Accuracy	0.955	0.958
Precision	0.975	0.973
Recall	0.934	0.942
f1	0.954	0.957
roc_auc	0.955	0.958
recall_auc	0.972	0.972

In [115…

```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(LogRegSmote_model)
#fit the trinig data to visiualize
visualizer.fit (X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
#draw visulization
visualizer.poof();
```



In [116…
```
cprint('\nLogistic Regression Cross Validation\n', 'green', 'on_yellow', attrs = ['bol
```

Logistic Regression Cross Validation

In [117…
```
cprint('\nCross Validation scores for logistic regression with defult parmeters(SMOTE)
LogRegSmote_cv = LogisticRegression(solver='liblinear', class_weight='balanced', randc
LogRegSmote_cv_scores = cross_validate(LogRegSmote_cv, X_train, y_train, scoring=['acc
LogRegSmote_cv_scores = pd.DataFrame(LogRegSmote_cv_scores, index = range(1, 11))
LogRegSmote_cv_scores.mean()[2:]
```

Cross Validation scores for logistic regression with defult parmeters(SMOTE)

Out[117]:
```
test_accuracy    0.954
test_precision   0.974
test_recall      0.934
test_f1          0.953
test_roc_auc     0.992
dtype: float64
```

In [118…
```
cprint('\nLogistic Regression Smote RandomizedSearchCV\n', 'green', 'on_yellow',attrs=
```

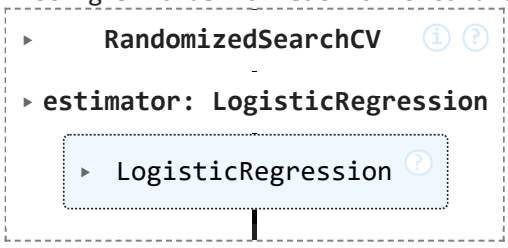
Logistic Regression Smote RandomizedSearchCV

In [119…
```
param_grid = {
    "class_weight": [None, "balanced"],
    "penalty": ["l1", "l2"],
    "solver": ['liblinear', 'lbfgs']
}
```

In [120…
```
# Initialize Logistic Regression with default parameters
LogRegSmote_grid = LogisticRegression(random_state=42)

# Perform Grid Search vs RandomizedSearchCV
LogRegSmote_grid_model = RandomizedSearchCV(LogRegSmote_grid, param_grid, scoring="f1"
LogRegSmote_grid_model.fit(X_train, y_train)
```

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Out[120]:



In [121…
```
print(colored('\033[1mBest Estimator of  RandomizedSearchCV for Logistic Regression Mc
print(colored('\033[1mBest Parameters of  RandomizedSearchCV for Logistic Regression M
```

Best Estimator of RandomizedSearchCV for Logistic Regression Model: LogisticRegressi
on(random_state=42, solver='liblinear')
Best Parameters of RandomizedSearchCV for Logistic Regression Model: {'solver': 'lib
linear', 'penalty': 'l2', 'class_weight': None}

In [122…
```
LogRegSmote_tuned = LogisticRegression(class_weight = 'balanced', penalty = 'l1',solve
```

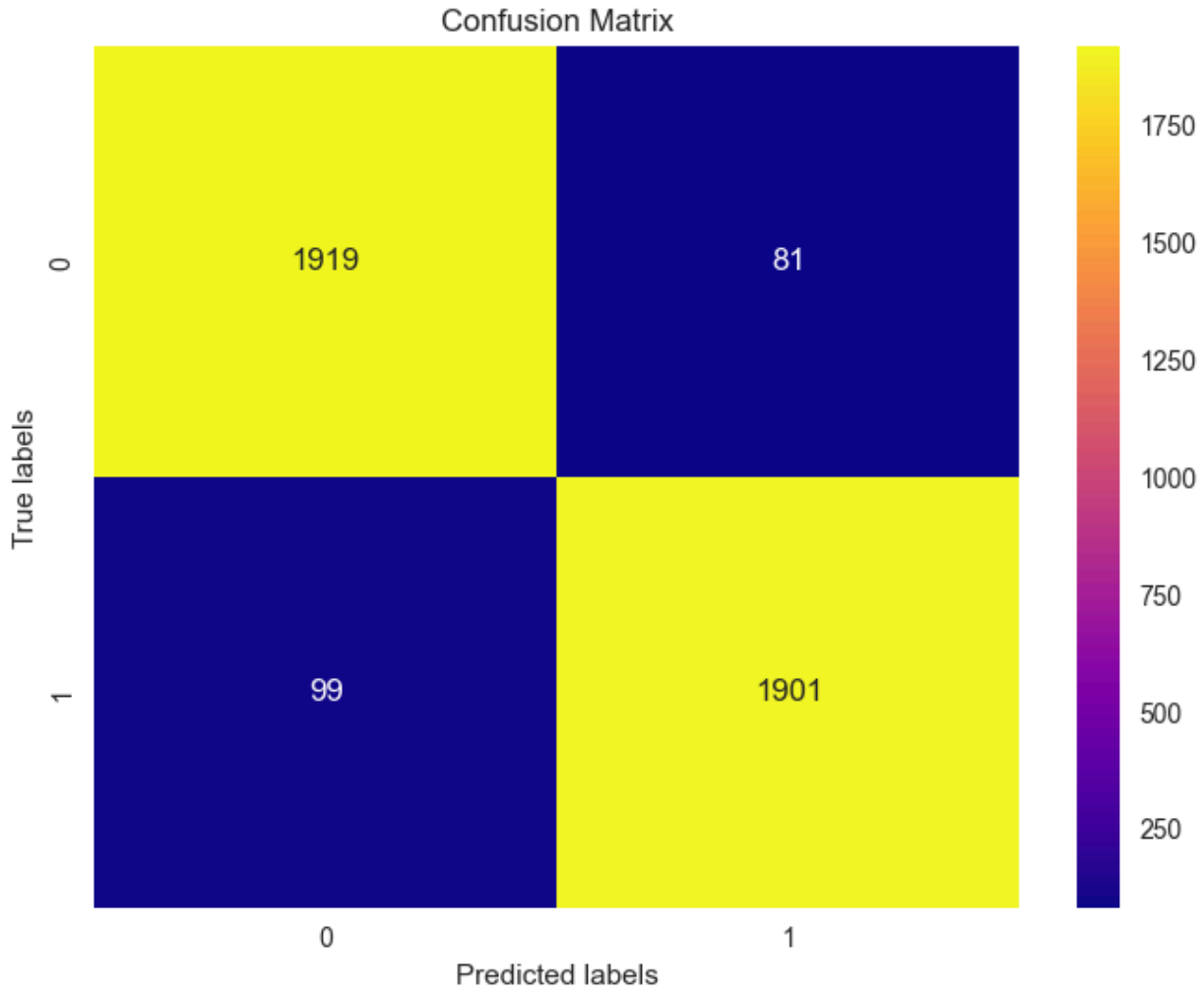
In [123…

```
y_pred = LogRegSmote_tuned.predict(X_test)
y_train_pred = LogRegSmote_tuned.predict(X_train)
LogRegSmote_tuned_f1 = f1_score(y_test, y_pred)
LogRegSmote_tuned_acc = accuracy_score(y_test, y_pred)
LogRegSmote_tuned_recall = recall_score(y_test, y_pred)
LogRegSmote_tuned_auc = roc_auc_score(y_test, y_pred)
LogRegSmote_tuned_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
LogRegSmote_tuned_recall_auc = auc(recall, precision)
```

In [124…

```
print("LogReg_tuned")
print("------------")
eval(LogReg_tuned, X_train, X_test)
```

```
LogReg_tuned
------------
[[1919   81]
 [  99 1901]]
Test_Set
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      2000
           1       0.96      0.95      0.95      2000

    accuracy                           0.95      4000
   macro avg       0.96      0.96      0.95      4000
weighted avg       0.96      0.95      0.95      4000

Train_Set
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      8000
           1       0.96      0.94      0.95      8000

    accuracy                           0.95     16000
   macro avg       0.95      0.95      0.95     16000
weighted avg       0.95      0.95      0.95     16000

------------------------------------------------------------
```



In [125…
```
cprint('\nLogistic Regression_tuned Scores\n', 'black', 'on_white', attrs=['bold'])
train_val(y_train, y_train_pred, y_test, y_pred)
```

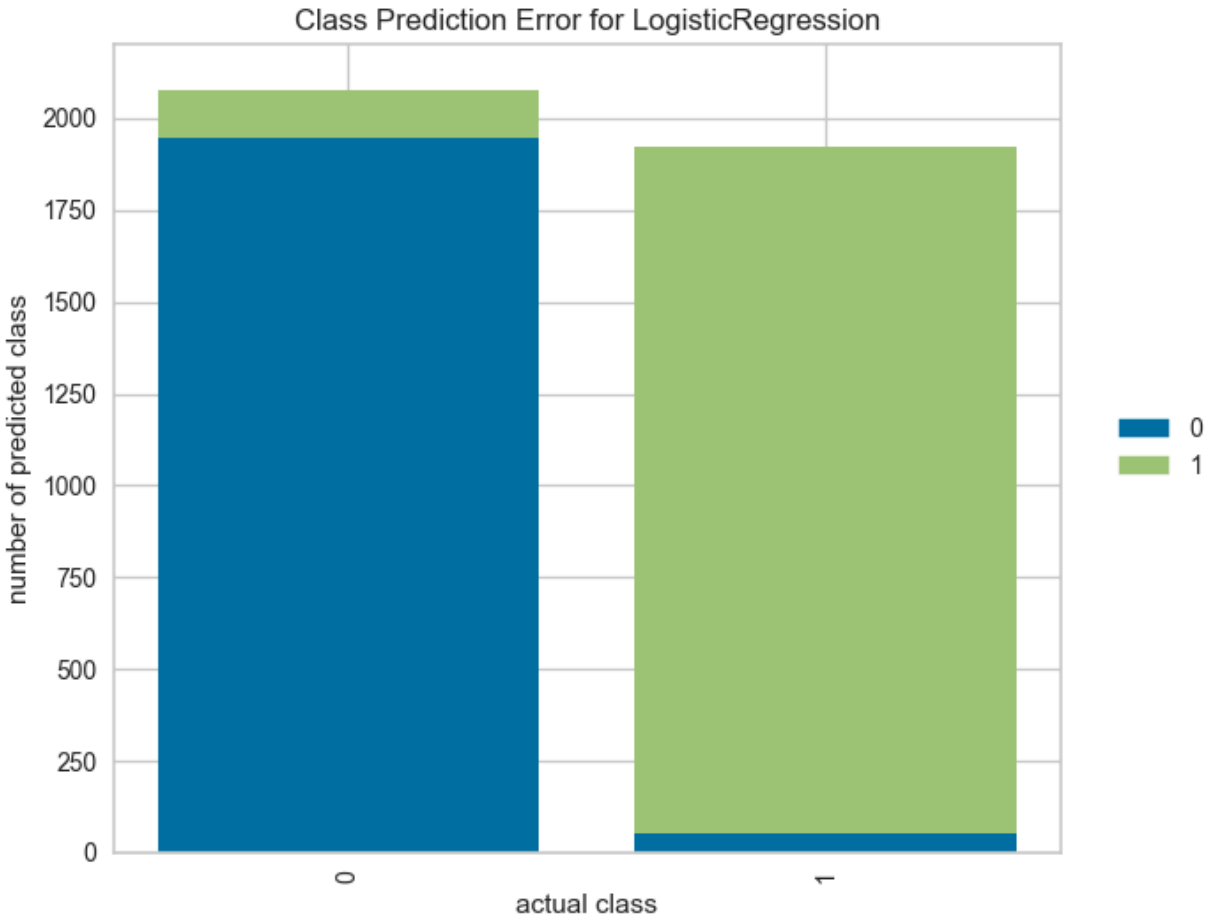
Logistic Regression_tuned Scores

Out[125]:

	train_set	test_set
Accuracy	0.954	0.955
Precision	0.976	0.973
Recall	0.931	0.936
f1	0.953	0.954
roc_auc	0.954	0.955
recall_auc	0.971	0.971

In [126…
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(LogRegSmote_tuned)
#fit the trining data to the visualizer
visualizer.fit(X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
```

```
#Draw visualization
visualizer.poof()
```



Out[126]:
```
<Axes: title={'center': 'Class Prediction Error for LogisticRegression'}, xlabel='act
ual class', ylabel='number of predicted class'>
```

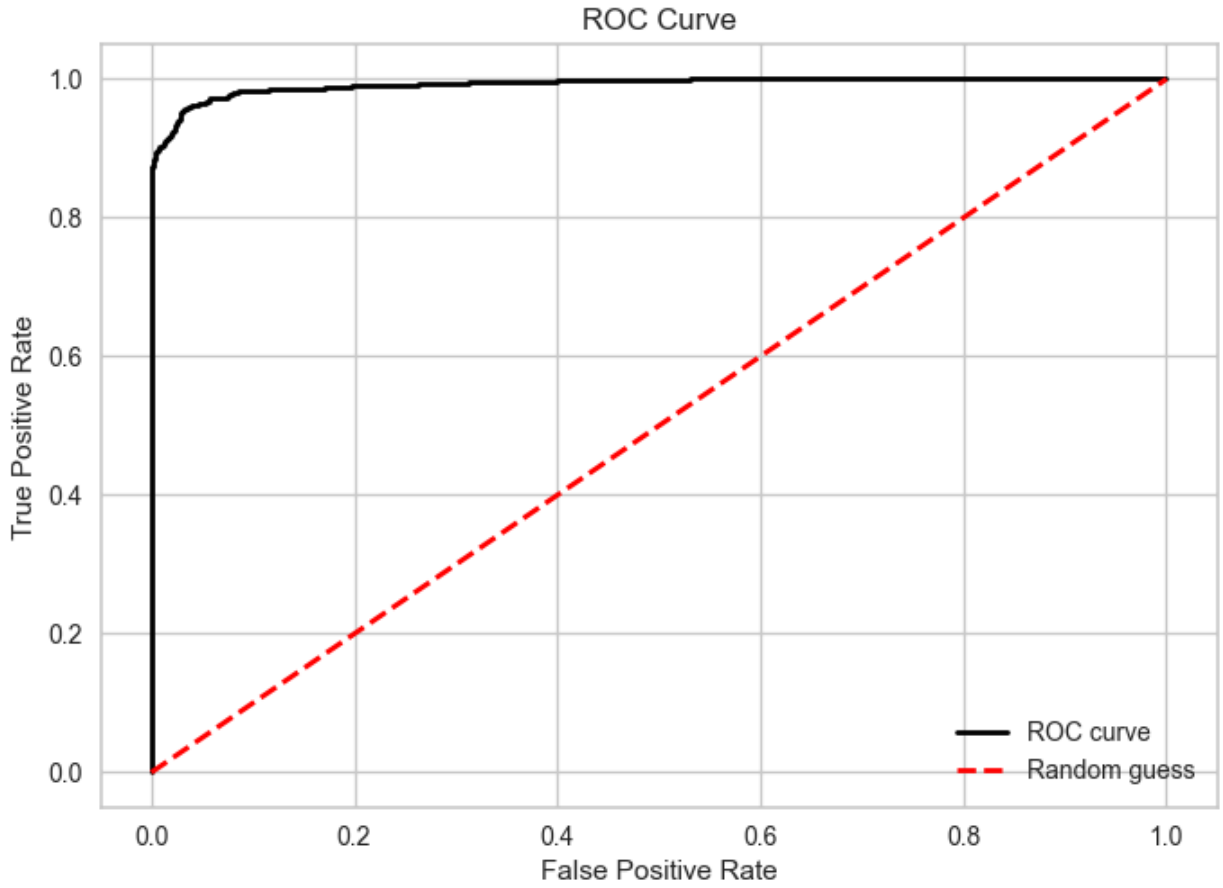
In [127…
```
cprint('\nLogistic Regression Roc (Receiver operating Curve) and AUC(Area Under curve)
cprint('\nroc_curve\n', 'black', 'on_white',attrs = ['bold'])
```

Logistic Regression Roc (Receiver operating Curve) and AUC(Area Under curve)

roc_curve

In [128…
```
# Compute ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, LogRegSmote_tuned.predict_proba(X_test)[:,1])
```

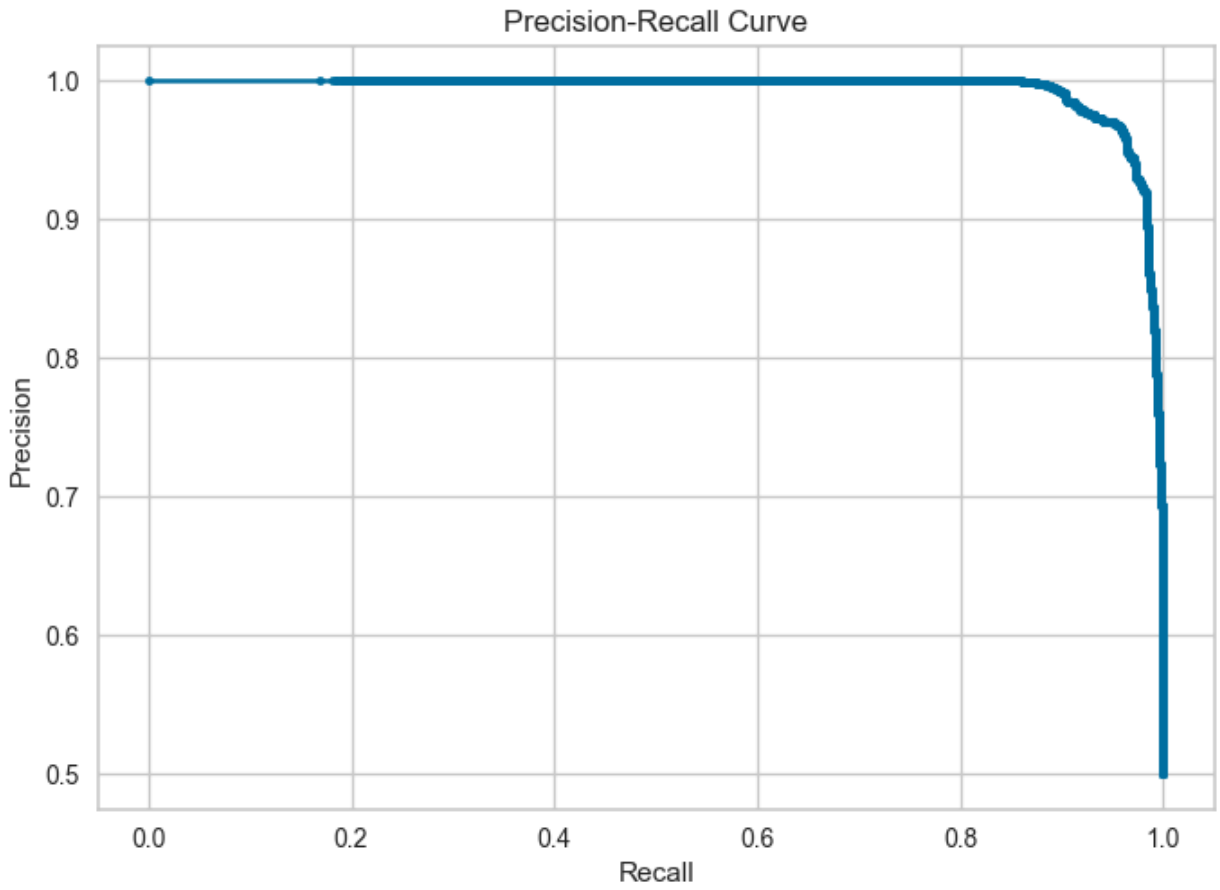
In [129…
```
# Plot ROC curve
plt.plot(fpr, tpr, color='black', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



In [130…

```
cprint('\nprecision_recall_curve\n', 'black', 'on_white',attrs=['bold'])
# Compute precision-recall pairs
precision, recall, _ = precision_recall_curve(y_test, LogRegSmote_tuned.predict_proba(

## Plot precision-recall curve
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```

precision_recall_curve



In [131…
```
cprint('\nLogReg_tuned Predictions\n','black', 'on_white',attrs=['bold'])
LogRegSmote_Pred = {"Actual": y_test, "LogRegSmote_Pred":y_pred}
LogRegSmote_Pred = pd.DataFrame.from_dict(LogRegSmote_Pred)
LogRegSmote_Pred.sample(5)
```

```
LogReg_tuned Predictions
```

Out[131]:

	Actual	LogRegSmote_Pred
108386	0	0
238067	1	1
235255	1	1
239212	1	1
234734	1	1

In [132…
```
pd.crosstab(LogRegSmote_Pred['Actual'], LogRegSmote_Pred['LogRegSmote_Pred'])
```

Out[132]:

LogRegSmote_Pred	**0**	**1**
Actual		
0	1949	51
1	128	1872

In [133…
```
cprint('\nPrediction\n', 'green', 'on_yellow', attrs=['bold'])
LogRegSmote_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, LogRegSmote_Pred, left_index = True, right_index =
Model_Preds.sample(5)
```

Prediction

Out[133]:

	Actual	LogReg_Pred	RF_Pred	LogRegSmote_Pred
232471	0	0	0	1
120335	0	0	0	0
19390	0	0	0	0
32854	0	0	0	0
231695	0	0	0	1

In [134…
```
#Logistic_regresson_smote = pickle.dump(LogRegSmote_tuned, open('logistic_regresson_sm
```

In [135…
```
cprint('\nRandom Forest Classifier with Smote\n','black', 'on_blue', attrs = ['bold'])
cprint('\nModel building with Random Forest Classifier\n','green','on_yellow', attrs =
RFSmote_model = RandomForestClassifier(class_weight = 'balanced', random_state = 42)
RFSmote_model.fit(X_train, y_train)
y_pred = RFSmote_model.predict(X_test)
y_train_pred = RFSmote_model.predict(X_train)
```

Random Forest Classifier with Smote

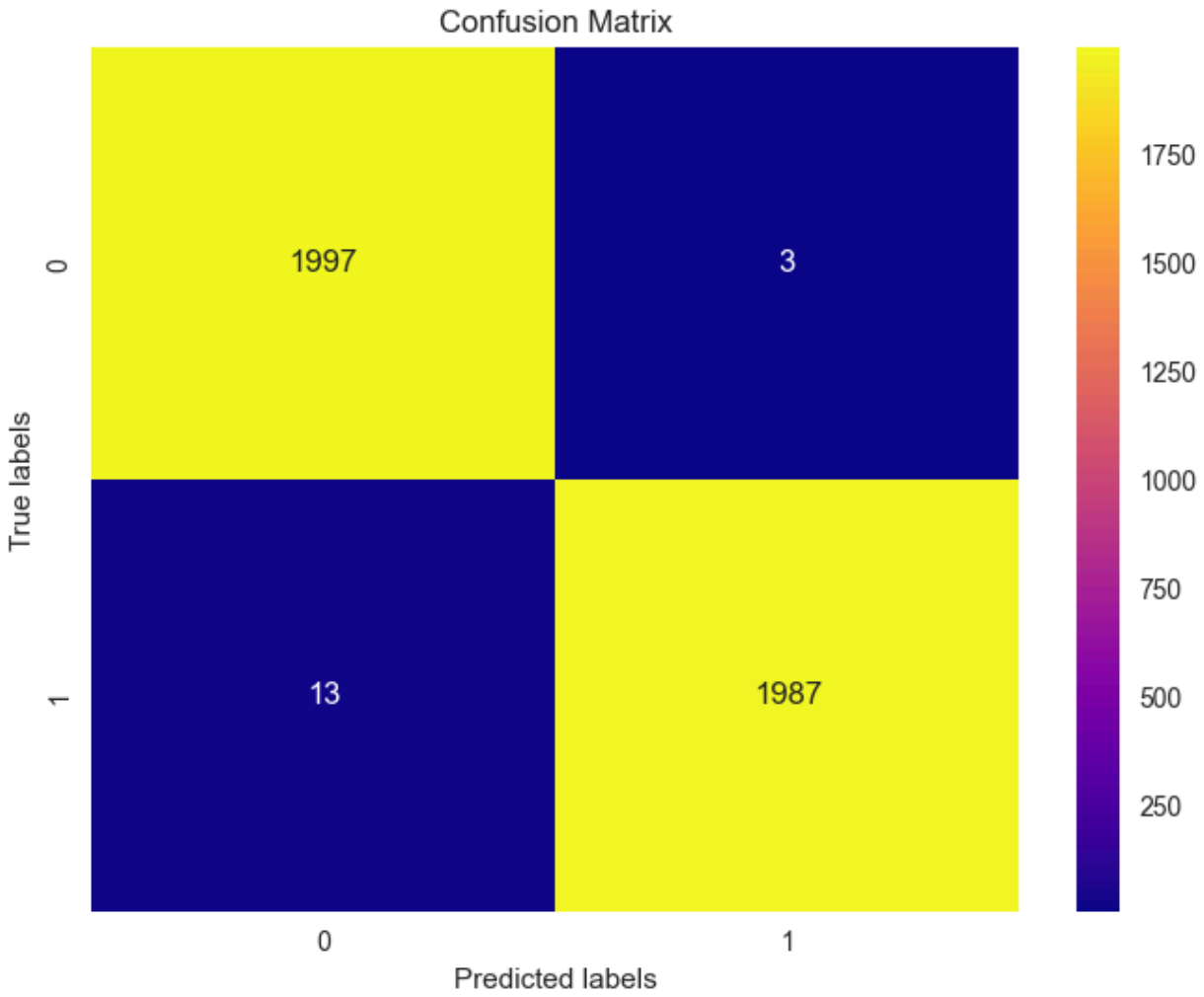
Model building with Random Forest Classifier

In [136…
```
cprint('\nEvaluating model Performance\n', 'green', 'on_yellow', attrs=['bold'])
RFSmote_model_f1 = f1_score(y_test, y_pred)
RFSmote_model_acc = accuracy_score(y_test, y_pred)
RFSmote_model_recall = recall_score(y_test, y_pred)
RFSmote_model_auc = roc_auc_score(y_test, y_pred)
RFSmote_model_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
RFSmote_model_recall_auc = auc(recall, precision)
```

Evaluating model Performance

In [137…
```
print("RFSmote_Model")
print("-----------")
eval(RFSmote_model, X_train, X_test)
```

```
RFSmote_Model
-----------
[[1997    3]
 [  13 1987]]
Test_Set
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2000
           1       1.00      0.99      1.00      2000

    accuracy                           1.00      4000
   macro avg       1.00      1.00      1.00      4000
weighted avg       1.00      1.00      1.00      4000

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8000
           1       1.00      1.00      1.00      8000

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

------------------------------------------------------------
```



In [138…

```
cprint('\nRF_smote_Model Score\n', 'black', 'on_white', attrs = ['bold'])
train_val(y_train, y_train_pred, y_test, y_pred)
```

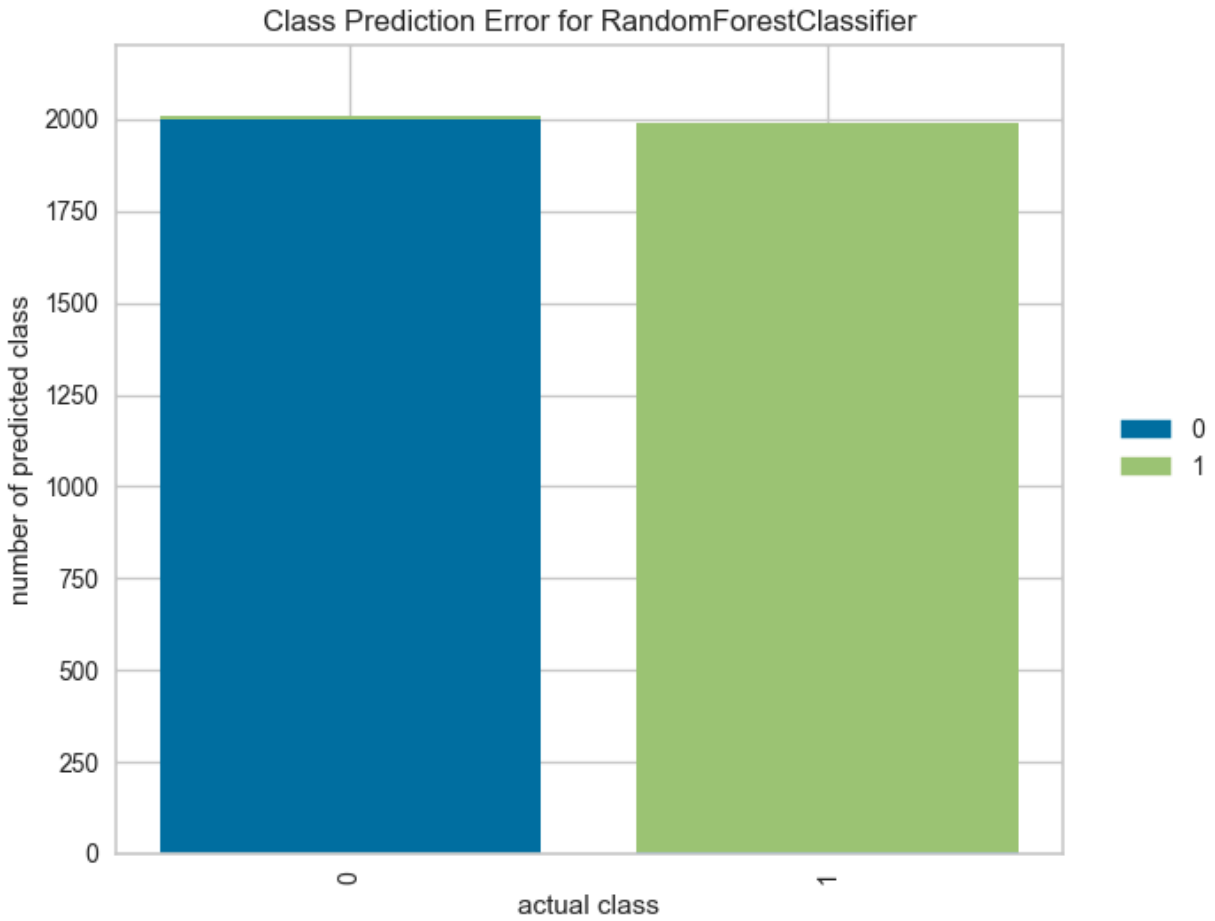
RF_smote_Model Score

Out[138]:

	train_set	test_set
Accuracy	1.000	0.996
Precision	1.000	0.998
Recall	1.000	0.994
f1	1.000	0.996
roc_auc	1.000	0.996
recall_auc	0.998	0.998

In [140…

```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(RFSmote_model)
#fit the trining data to the visualizer
visualizer.fit(X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
#Draw visualization
visualizer.poof()
```



Out[140]:

```
<Axes: title={'center': 'Class Prediction Error for RandomForestClassifier'}, xlabel
='actual class', ylabel='number of predicted class'>
```

In [141…

```
cprint('\nRandom Forest Cross Validation\n', 'green', 'on_yellow', attrs = ['bold'])
cprint('\nCross VAlidation Score for Random Forest with defult parameters(SMOTE)\n', '
```

```
RF_smote_cv = RandomForestClassifier(class_weight = 'balanced', random_state = 42)
RF_smote_cv_scores = cross_validate(RF_smote_cv, X_train, y_train, scoring = ['accurac
RF_smote_cv_scores = pd.DataFrame(RF_smote_cv_scores, index = range(1, 11))
RF_smote_cv_scores.mean()[2:]
```

Random Forest Cross Validation

Cross VAlidation Score for Random Forest with defult parameters(SMOTE)

Out[141]:
```
test_accuracy    0.996
test_precision   0.999
test_recall      0.993
test_f1          0.996
test_roc_auc     1.000
dtype: float64
```

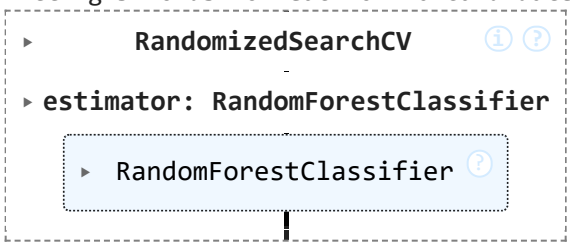
In [142…
```
cprint('\nRandom Forest Classifier RandomizedSearchCV\n', 'green', 'on_yellow', attrs
RF_smote_grid = RandomForestClassifier(class_weight='balanced', random_state=42)

RF_param_distributions = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

Random Forest Classifier RandomizedSearchCV

In [143…
```
RF_smote_grid_model = RandomizedSearchCV(
    estimator=RF_grid,
    param_distributions=RF_param_distributions,
    scoring="recall",
    n_jobs=-1,
    verbose=2
)

RF_smote_grid_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[143]:



In [144…
```
print(colored('\033[1mBest Estimator of RandomizedSearchCV for Random Forest Model:\03
print(colored('\033[1mBest Parameters of RandomizedSearchCV for Random Forest Model:\0
```

```
Best Estimator of RandomizedSearchCV for Random Forest Model: RandomForestClassifier
(class_weight='balanced', max_depth=30,
                       min_samples_leaf=2, min_samples_split=5,
                       random_state=42)
Best Parameters of RandomizedSearchCV for Random Forest Model: {'n_estimators': 100,
'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 3
0}
```

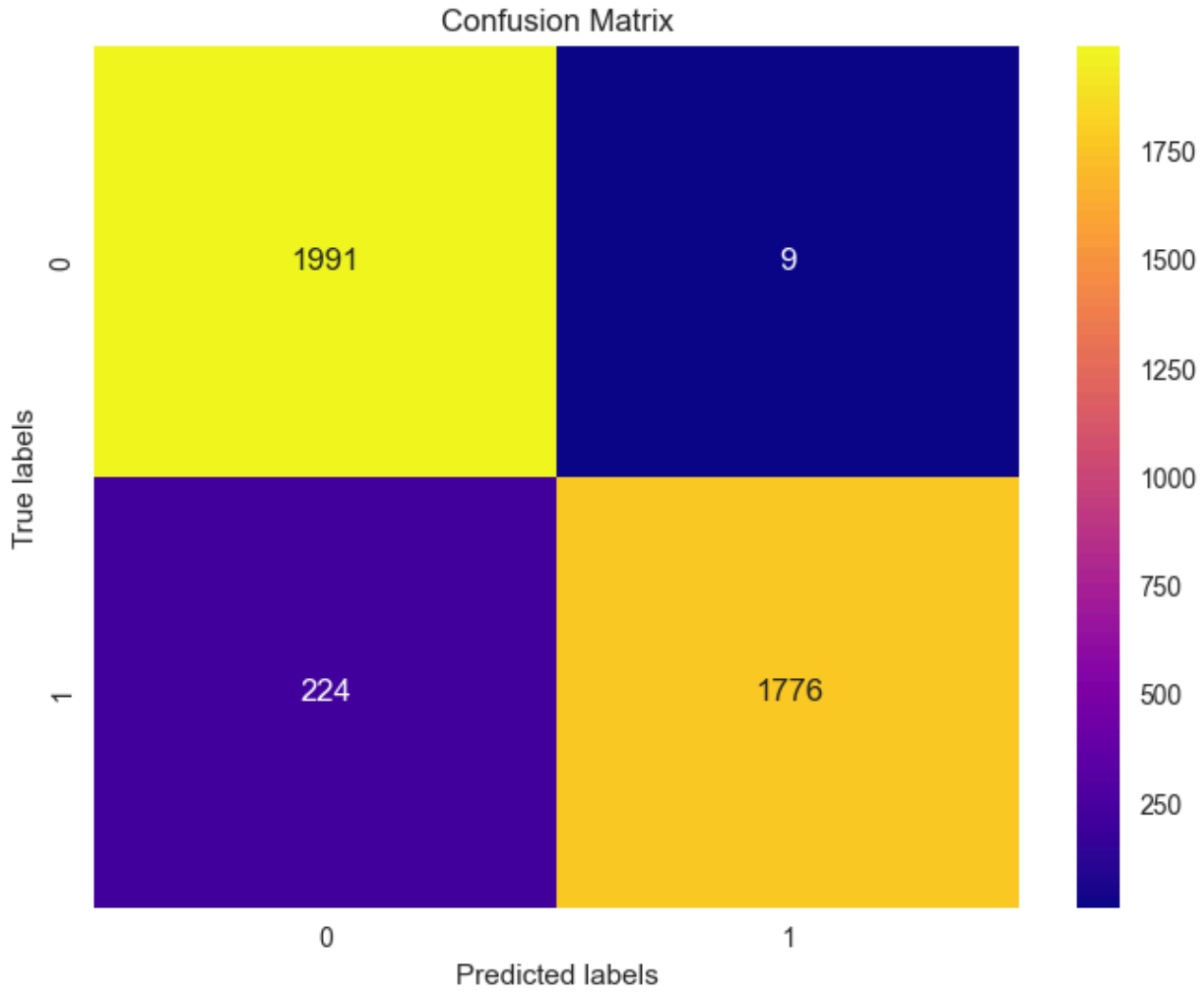
In [145…
```
RFSmote_tuned = RandomForestClassifier (class_weight = 'balanced', max_depth = 5, max_
```

In [146…
```
y_pred = RFSmote_tuned.predict(X_test)
y_train_pred = RFSmote_tuned.predict(X_train)
```

In [147…
```
RFSmote_tuned_f1 = f1_score(y_test, y_pred)
RFSmote_tuned_acc = accuracy_score(y_test, y_pred)
RFSmote_tuned_recall = recall_score(y_test, y_pred)
RFSmote_tuned_auc = roc_auc_score(y_test, y_pred)
RFSmote_tuned_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
RFSmote_tuned_recall_auc = auc(recall, precision)
```

In [148…
```
print("RF_Model")
print("---------")
eval(RFSmote_tuned, X_train, X_test)
```

```
RF_Model
---------
[[1991    9]
 [ 224 1776]]
Test_Set
              precision    recall  f1-score   support

           0       0.90      1.00      0.94      2000
           1       0.99      0.89      0.94      2000

    accuracy                           0.94      4000
   macro avg       0.95      0.94      0.94      4000
weighted avg       0.95      0.94      0.94      4000

Train_Set
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      8000
           1       1.00      0.88      0.94      8000

    accuracy                           0.94     16000
   macro avg       0.95      0.94      0.94     16000
weighted avg       0.95      0.94      0.94     16000

------------------------------------------------------------
```



In [149…
```
cprint('\nRF_tuned score\n', 'black', 'on_white', attrs=['bold'])
train_val(y_train, y_train_pred, y_test, y_pred)
```

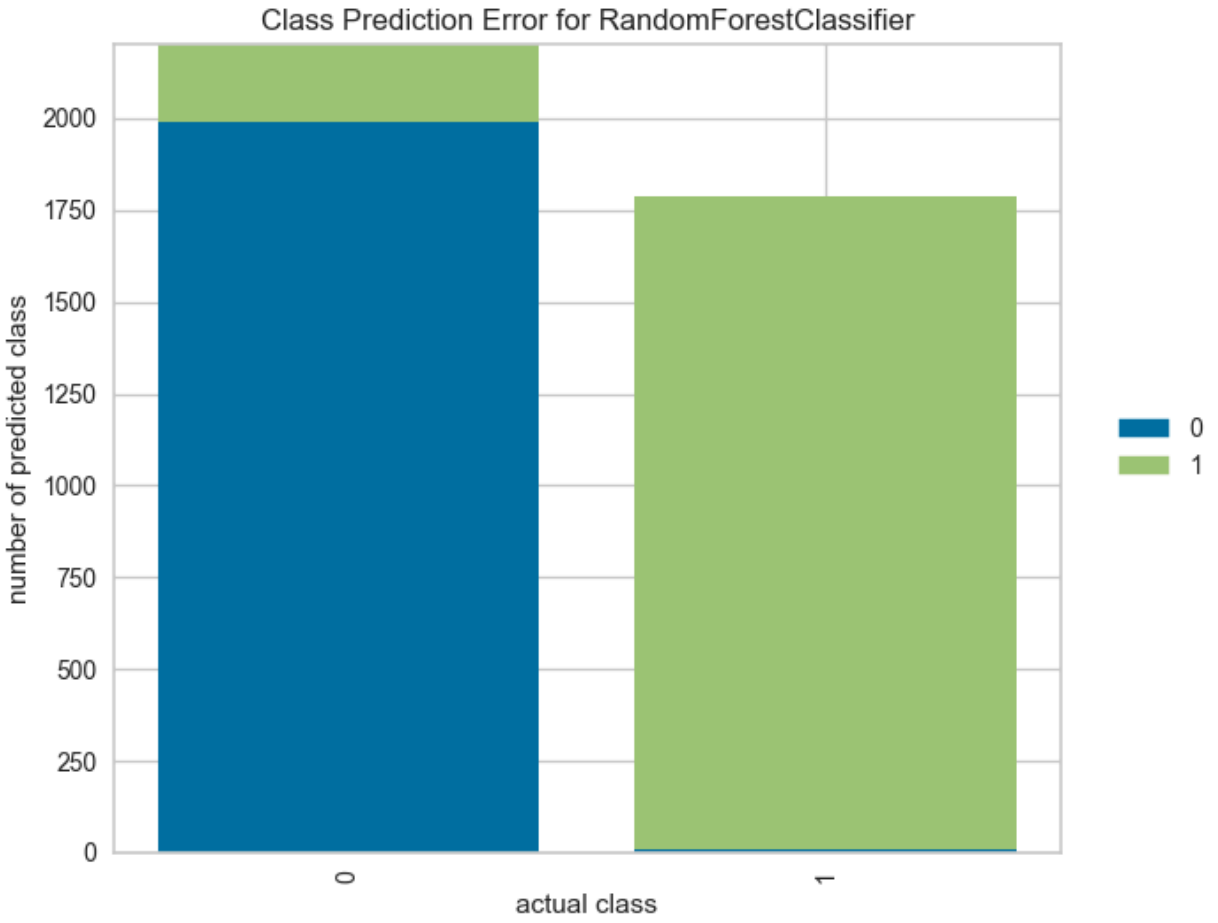
RF_tuned score

Out[149]:

	train_set	test_set
Accuracy	0.939	0.942
Precision	0.998	0.995
Recall	0.879	0.888
f1	0.935	0.938
roc_auc	0.939	0.942
recall_auc	0.969	0.969

In [150…
```
from yellowbrick.classifier import ClassPredictionError
visualizer = ClassPredictionError(RFSmote_tuned)
#fit the trining data to the visualizer
visualizer.fit(X_train, y_train)
#evalute the model on the test data
visualizer.score(X_test, y_test)
```

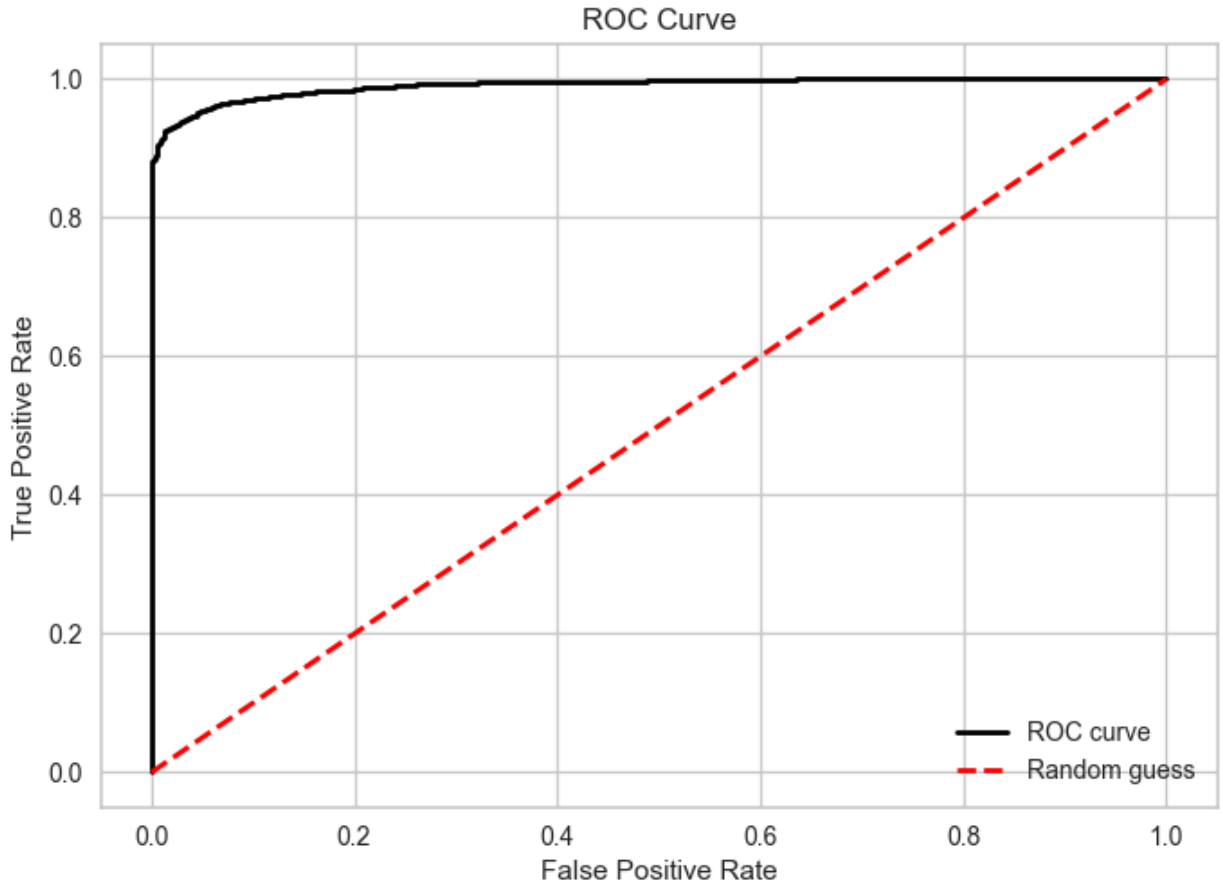
```
#Draw visualization
visualizer.poof()
```



Out[150]:
```
<Axes: title={'center': 'Class Prediction Error for RandomForestClassifier'}, xlabel
='actual class', ylabel='number of predicted class'>
```

In [151…
```
cprint('\nRandom Forest Roc (Receiver operating Curve) and AUC(Area Undercurve)\n', 'g
# Compute ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, RFSmote_tuned.predict_proba(X_test)[:,1])

# Plot ROC curve
plt.plot(fpr, tpr, color='black', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

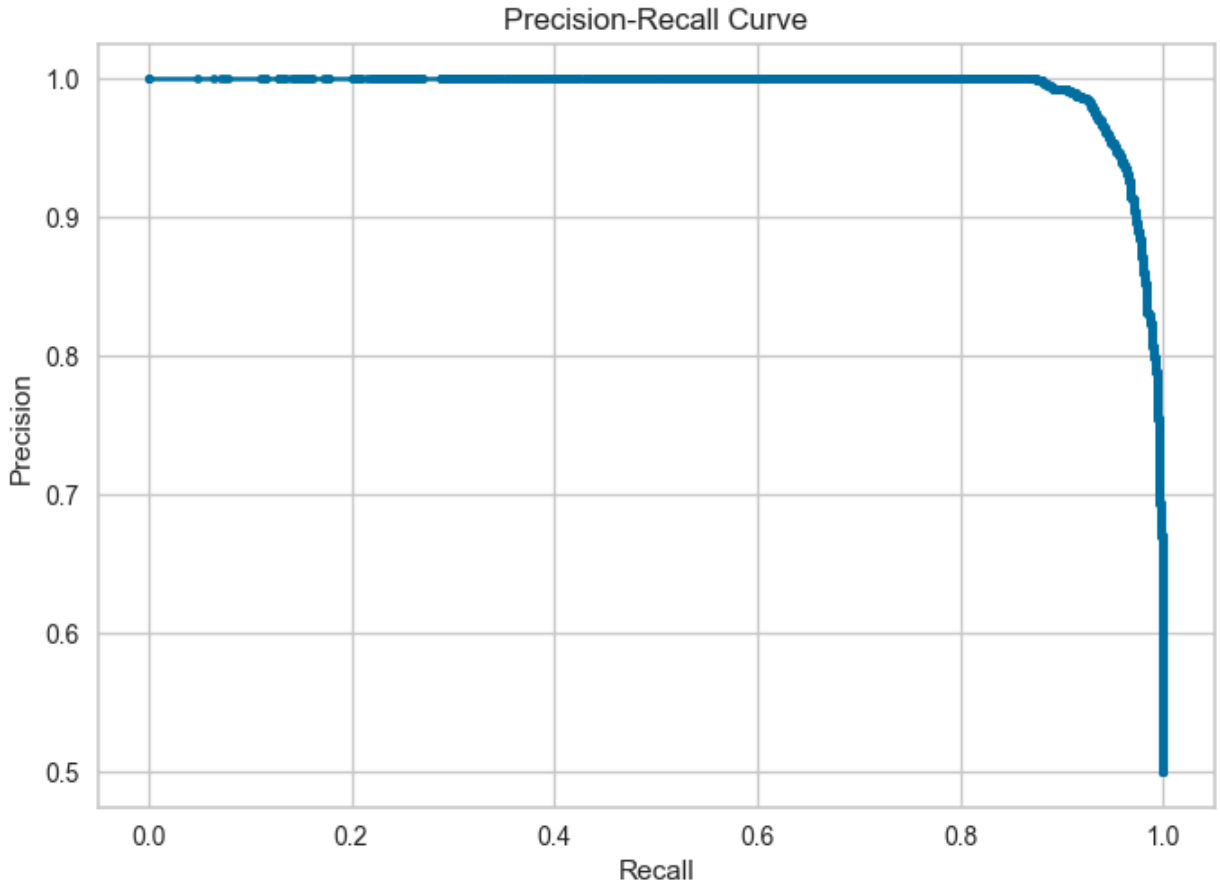
Random Forest Roc (Receiver operating Curve) and AUC(Area Undercurve)



In [152…

```
cprint('\nprecision_recall_curve\n', 'black', 'on_white',attrs=['bold'])
# Compute precision-recall pairs
precision, recall, _ = precision_recall_curve(y_test, RFSmote_tuned.predict_proba(X_te

## Plot precision-recall curve
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```

precision_recall_curve



In [153…
```
cprint('\nRandom Forest Prediction\n', 'black', 'on_white', attrs = ['bold'])
RFSmote_Pred = {"Actual": y_test, "RFSmote_Pred": y_pred}
RFSmote_Pred = pd.DataFrame.from_dict(RFSmote_Pred)
RFSmote_Pred.sample(5)
```

```
Random Forest Prediction
```

Out[153]:

	Actual	RFSmote_Pred
233517	1	1
166268	0	0
237090	1	1
41179	0	0
210911	0	0

In [154…
```
# Assuming RF_Pred is your DataFrame containing 'Actual' and 'RF_Pred' columns
pd.crosstab(RFSmote_Pred['Actual'], RFSmote_Pred['RFSmote_Pred'])
```

Out[154]:

RFSmote_Pred	0	1
Actual		
0	1991	9
1	224	1776

In [155…
```
cprint('\nPrediction\n', 'green', 'on_yellow', attrs = ['bold'])
RFSmote_Pred.drop("Actual", axis = 1, inplace = True)
Model_Preds = pd.merge(Model_Preds, RFSmote_Pred, left_index=True, right_index = True)
Model_Preds.sample(5)
```

Prediction

Out[155]:

	Actual	LogReg_Pred	RF_Pred	LogRegSmote_Pred	RFSmote_Pred
232524	0	0	0	1	1
147278	0	0	0	0	0
236231	0	0	0	1	1
148786	0	0	0	0	0
238115	0	0	0	1	1

In [156…
```
#random_forest_smote = pickle.dump(RFSmote_tuned, open('random_forest_smote(tuned)','w
```

In [157…
```
cprint("\nNeural Network\n", 'green', 'on_white', attrs = ['bold'])
cprint('\nData pre-processing\n', 'black', 'on_blue', attrs=['bold'])
cprint('\nScaling\n','green', 'on_yellow', attrs = ['bold'])
```

Neural Network

Data pre-processing

Scaling

In [158…
```
df_dl = df_out.copy()
```

In [159…
```
scaler = StandardScaler()
df_dl["amount"] = scaler.fit_transform(df_dl["amount"].values.reshape(-1, 1))
df_dl["time"] = scaler.fit_transform(df_dl["time"].values.reshape(-1, 1))
```

In [160…
```
cprint('\nTrain - Test Split\n','green', 'on_yellow', attrs = ['bold'])
X = df_dl.drop(['class'], axis = 1)
y = df_dl['class']
# Applying SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.
```

Train - Test Split

In [161…
```
print('X_train.shape : ', X_train.shape)
print('X_test.shape  : ', X_test.shape)
```

```
X_train.shape :  (369024, 30)
X_test.shape  :  (92256, 30)
```

In [162…
```
cprint('y_train.value_counts', 'black', 'on_white', attrs = ['bold'])
print("----"*10)
y_train.value_counts()
```

```
y_train.value_counts
----------------------------------------
```

Out[162]:
```
class
1    184512
0    184512
Name: count, dtype: int64
```

In [163…
```
cprint('y_test.value_counts', 'black', 'on_white', attrs = ['bold'])
print("----"*10)
y_test.value_counts()
```

```
y_test.value_counts
----------------------------------------
```

Out[163]:
```
class
0    46128
1    46128
Name: count, dtype: int64
```

In [164…
```
cprint('\nImport libraries\n', 'green', 'on_yellow', attrs=['bold'])
```

Import libraries

In [165…
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam, SGD, Adamax, RMSprop,Adadelta
from tensorflow.keras.layers import Dropout
from sklearn.utils import class_weight
from sklearn.utils import compute_class_weight
from scikeras.wrappers import KerasClassifier
# Assuming you have defined build_classifier function earlier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve, average_precision_score
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [166…
```
cprint('\nDefine Model\n', 'green', 'on_yellow', attrs=['bold'])
import tensorflow as tf
print(tf.__version__)
```

Define Model

```
2.14.0
```

In [167…
```
dl_model = Sequential()
dl_model.add(Dense(32, activation="relu"))  # Specify activation as argument
dl_model.add(Dropout(0.25))
dl_model.add(Dense(16, activation="relu"))  # Specify activation as argument
dl_model.add(Dropout(0.25))
dl_model.add(Dense(1, activation="sigmoid"))
```

In [168…
```
cprint('\nCompile Model\n', 'green', 'on_yellow', attrs=['bold'])
```

Compile Model

In [169…
```
optimizer = Adam(learning_rate = 0.001)
dl_model.compile(loss = 'binary_crossentropy',
                 optimizer = optimizer,
                 metrics = ["Recall"])
```

In [170…
```
train_classes = y_train
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(train_
class_weights = dict(zip(np.unique(train_classes), class_weights))
print("class_weights : ", class_weights)
```

```
class_weights :  {0: 1.0, 1: 1.0}
```

In [171…
```
early_stop = EarlyStopping(monitor= "val_loss", mode = "auto", verbose = 1, patience=
```

In [172…
```
cprint('\nFit model\n', 'green', 'on_yellow', attrs=['bold'])
```

Fit model

In [173…
```
dl_model.fit(x=X_train,
             y=y_train,
             validation_split=0.1,
             batch_size=64,
             epochs=500,
             verbose=0,
             callbacks=[early_stop],

             class_weight={0: 0.500766887790496, 1: 326.4929328621908})
```

```
Restoring model weights from the end of the best epoch: 119.
Epoch 139: early stopping
```

Out[173]:
```
<keras.src.callbacks.History at 0x267a900a1d0>
```

In [174…
```
dl_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                992

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 16)                528

 dropout_1 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 1537 (6.00 KB)
Trainable params: 1537 (6.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [175…
```
cprint('\nEvaluating model Performance\n', 'green', 'on_yellow', attrs=['bold'])
```

Evaluating model Performance

In [176…
```
cprint('\nDL_model loss_df\n', 'black', 'on_white', attrs=['bold'])
loss_df = pd.DataFrame(dl_model.history.history)
loss_df.head(1)
```

DL_model loss_df

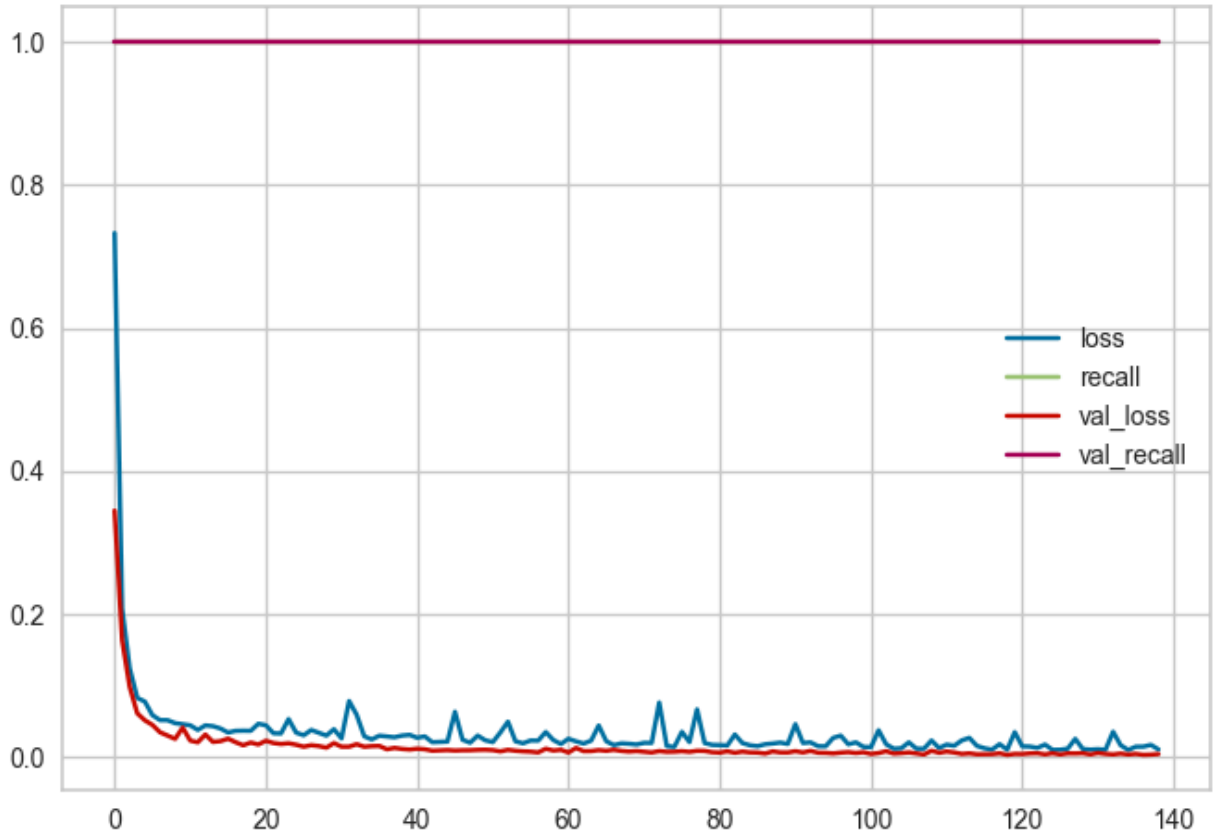
Out[176]:

	loss	recall	val_loss	val_recall
0	0.733	1.000	0.345	1.000

In [177…
```
cprint('\nDL_model loss_df\n', 'black', 'on_white', attrs=['bold'])
loss_df.plot()
```

DL_model loss_df

Out[177]:
```
<Axes: >
```



In [178…
```
cprint('\nDL_model Evaluating (train_set)\n', 'black', 'on_white', attrs=['bold'])
print("------"*10)
dl_model.evaluate(X_train, y_train)
```

```
DL_model Evaluating (train_set)

------------------------------------------------------------
11532/11532 [==============================] - 12s 1ms/step - loss: 0.0034 - recall:
1.0000
```
Out[178]:
```
[0.0034319499973207712, 1.0]
```

In [179…
```
cprint('\nDL_model Evaluating (test_set)\n', 'black', 'on_white', attrs=['bold'])
print("------"*10)
dl_model.evaluate(X_test, y_test)
```

```
DL_model Evaluating (test_set)

------------------------------------------------------------
2883/2883 [==============================] - 3s 1ms/step - loss: 0.0044 - recall: 1.0
000
```
Out[179]:
```
[0.004393872804939747, 1.0]
```

In [180…
```
cprint('\nDL_model Evaluating (test_set)\n', 'black', 'on_white', attrs=['bold'])
print("------"*10)
loss, accuracy = dl_model.evaluate(X_test, y_test, verbose = 0)
print("loss: ", loss)
print("accuracy: ", accuracy)
```

DL_model Evaluating (test_set)

```
------------------------------------------------------------
loss:  0.004393872804939747
accuracy:  1.0
```

In [181…
```
cprint('\nConfusion_Matrix and Clasification_Report\n', 'black', 'on_white', attrs=['b
print("----"*10)
y_pred = (dl_model.predict(X_test) > 0.5 ).astype("int32")
#
print(confusion_matrix(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Confusion_Matrix and Clasification_Report

```
----------------------------------------
2883/2883 [==============================] - 3s 827us/step
[[46059    69]
 [    0 46128]]
[[46059    69]
 [    0 46128]]
```

In [182…
```
y_pred_proba = dl_model.predict(X_test)
DL_acc = accuracy_score(y_test, y_pred)
DL_AP = average_precision_score(y_test, y_pred_proba)
DL_f1 = f1_score (y_test, y_pred)
DL_rec = recall_score(y_test, y_pred)
DL_roc_auc =  roc_auc_score(y_test, y_pred)
DL_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
DL_recall_auc = auc(recall, precision)
```

```
2883/2883 [==============================] - 3s 868us/step
```

In [183…
```
cprint('\nDL_model score \n', 'black', 'on_white', attrs=['bold'])
print("----"*10)
print("DL_acc : ", DL_acc)
print("DL_AP : ", DL_AP)
print("DL_f1 : ", DL_f1)
print("DL_rec : ", DL_rec)
print("DL_roc_auc : ", DL_roc_auc)
print("DL_pre : ", DL_pre)
print("DL_recall_auc : ", DL_recall_auc)
```

DL_model score

```
----------------------------------------
DL_acc :  0.9992520811654526
DL_AP :  0.9999929620582615
DL_f1 :  0.9992526401299756
DL_rec :  1.0
DL_roc_auc :  0.9992520811654527
DL_pre :  0.9985063965192545
DL_recall_auc :  0.9992531982596273
```

In [184…
```
cprint('\nNeural Network RandomizedSearchCV\n', 'green', 'on_yellow', attrs=['bold'])
```

Neural Network RandomizedSearchCV

In [185…
```
#def build_classifier(optimizer):
    #classifier = Sequential()
    #classifier.add(Dense(32, activation="relu"))  # Specify activation as argument
    #classifier.add(Dropout(0.25))
    #classifier.add(Dense(16, activation="relu"))  # Specify activation as argument
    #classifier.add(Dropout(0.25))
    #classifier.add(Dense(1, activation="sigmoid"))
    #classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics =
    #return classifier

def build_classifier(optimizer='adam'):
    classifier = Sequential()
    classifier.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
    classifier.add(Dropout(0.5))
    classifier.add(Dense(units=1, activation='sigmoid'))
    classifier.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accu
    return classifier
```

In [186…
```
print("class_weights : ", class_weights)
```

```
class_weights :  {0: 1.0, 1: 1.0}
```

In [187…
```
#early_stop = EarlyStopping(monitor= "val_loss", mode = "auto", verbose = 1, patience=
# Define the early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1, restore_best_wei
```

In [ ]:
```
# Define the KerasClassifier with the simplified model
classifier = KerasClassifier(build_fn=build_classifier, epochs=20)

# Define the parameter grid
parameters = {'batch_size': [32, 64]}

# Create the GridSearchCV object
grid_model = GridSearchCV(estimator=classifier,
                          param_grid=parameters,
                          scoring='recall',
                          cv=10,
                          n_jobs=-1,
                          verbose=0)

# Fit the GridSearchCV object with the preprocessed training data
grid_model.fit(X_train, y_train, callbacks=[early_stop], class_weight={0: 0.5007668877
```

In [ ]:
```
print("grid_model.best_score_: ",grid_model.best_score_)
```

In [ ]:
```
cprint('\nConfusion_Matrix and Clasification_Report\n', 'black', 'on_white', attrs=['b
print("----"*10)
y_pred = (grid_model.predict(X_test) > 0.5 ).astype("int32")
#
print(confusion_matrix(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

In [188…
```
dl_tuned = Sequential()
dl_tuned.add(Dense(32, activation="relu"))  # Specify activation as argument
```

```
dl_tuned.add(Dropout(0.25))
dl_tuned.add(Dense(16, activation="relu"))  # Specify activation as argument
dl_tuned.add(Dropout(0.25))
dl_tuned.add(Dense(1, activation="sigmoid"))
```

In [189…
```
optimizer = SGD(learning_rate = 0.001)
dl_tuned.compile(loss = 'binary_crossentropy',
                 optimizer = optimizer,
                 metrics = ["Recall"])
```

In [190…
```
train_classes = y_train
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(train_
class_weights = dict(zip(np.unique(train_classes), class_weights))
print("class_weights : ", class_weights)
```

```
class_weights :  {0: 1.0, 1: 1.0}
```

In [191…
```
early_stop = EarlyStopping(monitor= "val_loss", mode = "auto", verbose = 1, patience=
```

In [192…
```
dl_tuned.fit(x=X_train,
             y=y_train,
             validation_split=0.1,
             batch_size=64,
             epochs=500,
             verbose=0,
             callbacks=[early_stop],
             class_weight={0: 0.500766887790496, 1: 326.4929328621908})
```

```
Restoring model weights from the end of the best epoch: 186.
Epoch 206: early stopping
```
Out[192]:
```
<keras.src.callbacks.History at 0x267ac474990>
```

In [193…
```
dl_tuned.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 32)                992

 dropout_2 (Dropout)         (None, 32)                0

 dense_4 (Dense)             (None, 16)                528

 dropout_3 (Dropout)         (None, 16)                0

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 1537 (6.00 KB)
Trainable params: 1537 (6.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [194…
```
cprint('\nDL_model loss_df\n', 'black', 'on_white', attrs=['bold'])
loss_df = pd.DataFrame(dl_tuned.history.history)
loss_df.head(1)
```

DL_model loss_df

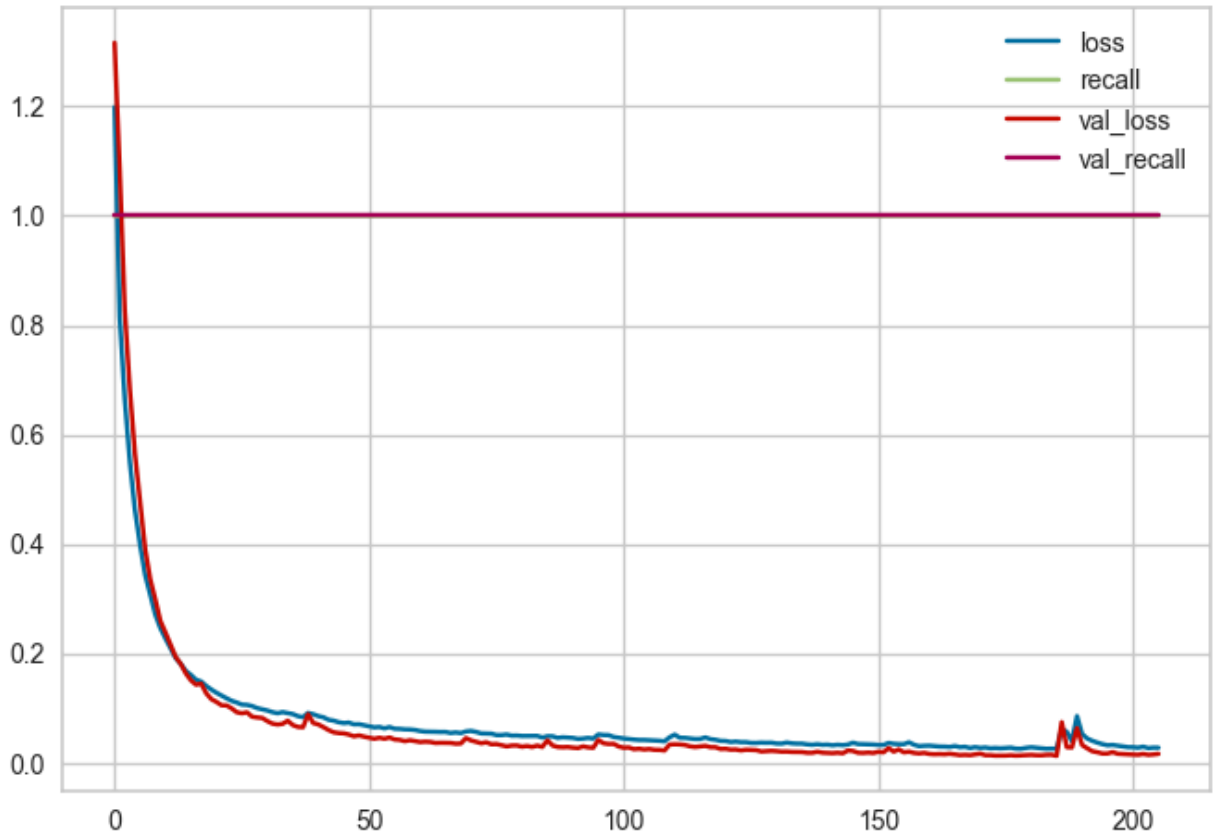
Out[194]:

	loss	recall	val_loss	val_recall
0	1.199	1.000	1.316	1.000

In [195…
```
cprint('\nDL_model loss_df\n', 'black', 'on_white', attrs=['bold'])
loss_df.plot()
```

DL_model loss_df

Out[195]: <Axes: >



In [196…
```
cprint('\nDL_tuned Evaluating (train_set)\n', 'black', 'on_white', attrs=['bold'])
print("------"*10)
dl_tuned.evaluate(X_train, y_train)
```

DL_tuned Evaluating (train_set)

Out[196]:
```
------------------------------------------------------------
11532/11532 [==============================] - 12s 1ms/step - loss: 0.0127 - recall:
1.0000
[0.012672311626374722, 1.0]
```

In [197…
```
cprint('\nDL_tuned Evaluating (test_set)\n', 'black', 'on_white', attrs=['bold'])
print("------"*10)
dl_tuned.evaluate(X_test, y_test)
```

```
DL_tuned Evaluating (test_set)

------------------------------------------------------------
2883/2883 [==============================] - 3s 998us/step - loss: 0.0145 - recall:
1.0000
```

Out[197]:
```
[0.014509974047541618, 1.0]
```

In [198…
```
cprint('\nDL_tuned Evaluating (test_set)\n', 'black', 'on_white', attrs=['bold'])
print("------"*10)
loss, accuracy = dl_tuned.evaluate(X_test, y_test, verbose = 0)
print("loss: ", loss)
print("accuracy: ", accuracy)
```

```
DL_tuned Evaluating (test_set)

------------------------------------------------------------
loss:  0.014509974047541618
accuracy:  1.0
```

In [199…
```
cprint('\nConfusion_Matrix and Clasification_Report\n', 'black', 'on_white', attrs=['b
print("----"*10)
y_pred = (dl_tuned.predict(X_test) > 0.5 ).astype("int32")
print(confusion_matrix(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
Confusion_Matrix and Clasification_Report

----------------------------------------
2883/2883 [==============================] - 3s 856us/step
[[45894   234]
 [    0 46128]]
[[45894   234]
 [    0 46128]]
```

In [200…
```
y_pred_proba = dl_tuned.predict(X_test)
DLTuned_acc = accuracy_score(y_test, y_pred)
DLTuned_AP = average_precision_score(y_test, y_pred_proba)
DLTuned_f1 = f1_score (y_test, y_pred)
DLTuned_rec = recall_score(y_test, y_pred)
DLTuned_roc_auc =  roc_auc_score(y_test, y_pred)
DLTuned_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
DLTuned_recall_auc = auc(recall, precision)
```

```
2883/2883 [==============================] - 3s 961us/step
```

In [201…
```
cprint('\nDL_tuned score \n', 'black', 'on_white', attrs=['bold'])
print("----"*10)
print("DLTuned_acc : ", DLTuned_acc)
print("DLTuned_AP : ", DLTuned_AP)
print("DLTuned_f1 : ", DLTuned_f1)
print("DLTuned_rec : ", DLTuned_rec)
print("DLTuned_roc_auc : ", DLTuned_roc_auc)
print("DLTuned_pre : ", DLTuned_pre)
print("DLTuned_recall_auc : ", DLTuned_recall_auc)
```

DL_tuned score

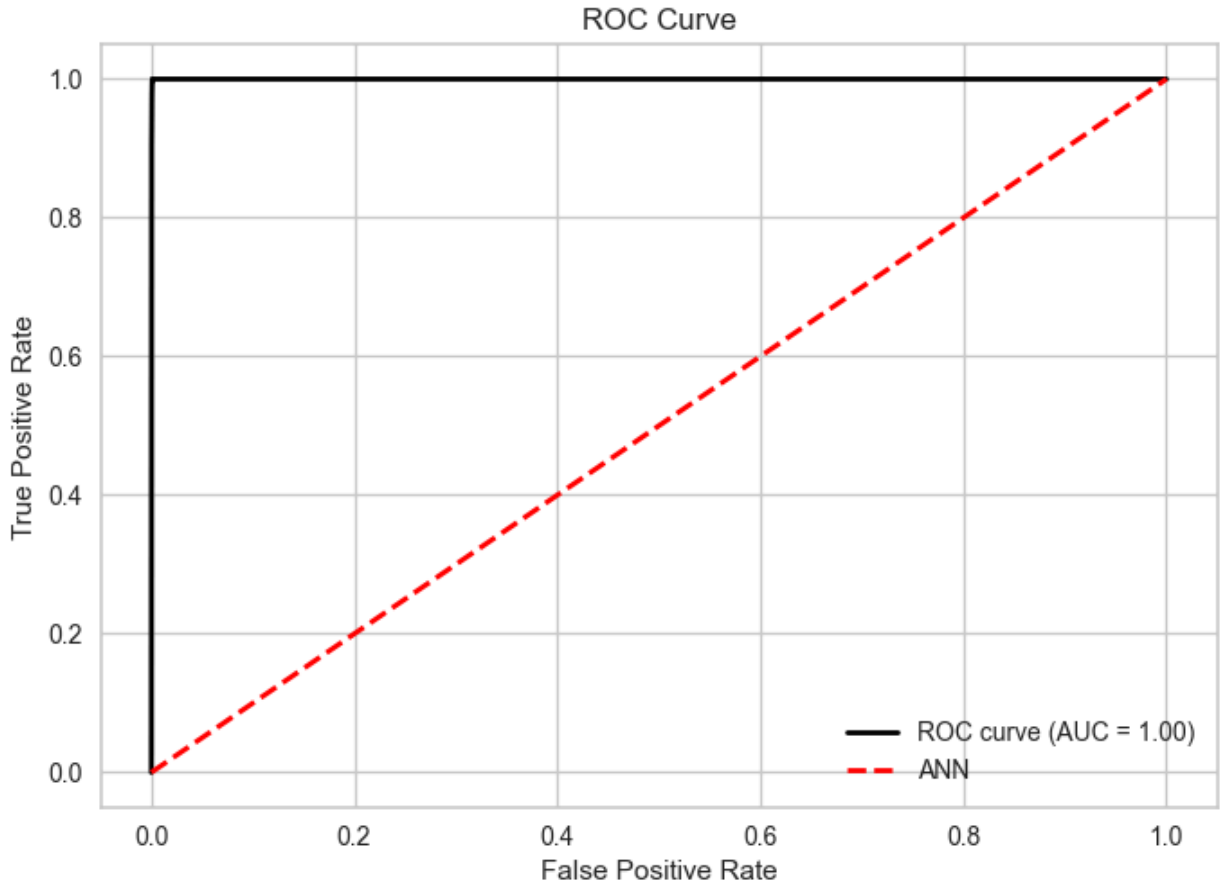
```
----------------------------------------
DLTuned_acc :  0.9974635796045785
DLTuned_AP :  0.9999022746870562
DLTuned_f1 :  0.9974699967564061
DLTuned_rec :  1.0
DLTuned_roc_auc :  0.9974635796045785
DLTuned_pre :  0.9949527630386955
DLTuned_recall_auc :  0.9974763815193477
```

In [202…

```
cprint('\nNeural Network Roc (Receiver operating Curve) and AUC(Area Undercurve)\n', '
# Get predicted probabilities
y_pred_prob = dl_tuned.predict(X_test)

# Compute ROC curve points
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute AUC
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, color='black', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='ANN')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

Neural Network Roc (Receiver operating Curve) and AUC(Area Undercurve)

```
2883/2883 [==============================] - 3s 863us/step
```

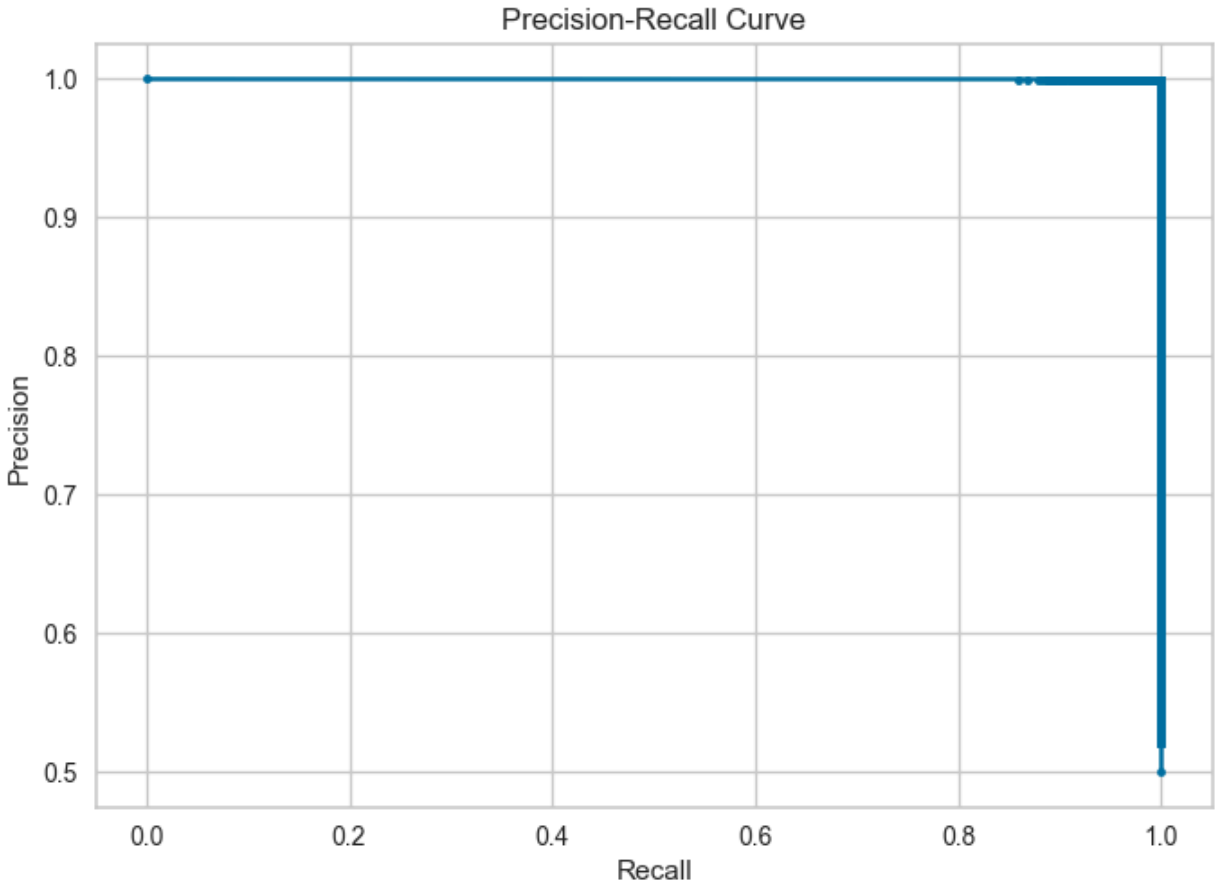


In [203…

```
cprint('\nprecision_recall_curve\n', 'black', 'on_white',attrs=['bold'])
# Get predicted probabilities
y_pred_prob = dl_tuned.predict(X_test)

# Compute precision-recall pairs
precision, recall, _ = precision_recall_curve(y_test, y_pred_prob)

## Plot precision-recall curve
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```

precision_recall_curve

```
2883/2883 [==============================] - 2s 852us/step
```



In [204…

```
cprint('\nPrediction\n', 'green', 'on_yellow', attrs=['bold'])
```

Prediction

In [205…

```
cprint('\ndl_tuned Predictions\n','black', 'on_white',attrs=['bold'])
# Assuming y_test and y_pred are your arrays
# Flatten them if they are not already 1-dimensional
y_test_flat = np.array(y_test).flatten()
y_pred_flat = np.array(y_pred).flatten()

# Create the dictionary for DataFrame
dl_Pred = {"Actual": y_test_flat, "dl_Pred": y_pred_flat}

# Create DataFrame from the dictionary
dl_Pred_df = pd.DataFrame(dl_Pred)

# Now you can sample from dl_Pred_df
dl_Pred_df.sample(5)
```

dl_tuned Predictions

Out[205]:

	Actual	dl_Pred
77363	0	0
19721	1	1
28329	1	1
33117	1	1
68406	1	1

In [206…
```
pd.crosstab(dl_Pred['Actual'], dl_Pred['dl_Pred'])
```

Out[206]:

col_0	0	1
row_0		
0	45894	234
1	0	46128

In [207…
```
cprint('\nPrediction\n', 'green', 'on_yellow', attrs=['bold'])
# Assuming dl_Pred is your dictionary
dl_Pred_df = pd.DataFrame(dl_Pred)

# Now you can sample from dl_Pred_df
dl_Pred_df.sample(5)
```

Prediction

Out[207]:

	Actual	dl_Pred
82025	0	0
76568	1	1
53037	1	1
78722	0	0
2992	0	0

In [208…
```
cprint('\nThe Comparision Of Model\n', 'green', 'on_yellow', attrs=['bold'])
```

The Comparision Of Model

In [209…
```
# Create DataFrame
compare = pd.DataFrame({
    "Model": ["LogReg_model", "LogReg_tuned", "RF_model", "RF_tuned", "LogRegSmote_mod
    "Accuracy_Score": [LogReg_model_acc, LogReg_tuned_acc, RF_model_acc, RF_tuned_acc,
    "F1_Score": [LogReg_model_f1, LogReg_tuned_f1, RF_model_f1, RF_tuned_f1, LogRegSmo
    "Precision_Score": [LogReg_model_pre, LogReg_tuned_pre, RF_model_pre, RF_tuned_pre
    "Recall_Score": [LogReg_model_recall, LogReg_tuned_recall, RF_model_recall, RF_tur
    "Roc_Auc_Score": [LogReg_model_auc, LogReg_tuned_auc, RF_model_auc, RF_tuned_auc,
    "Recall_Auc_Score": [LogReg_model_recall_auc, LogReg_tuned_recall_auc, RF_model_re
})
```

```python
# Sorting and plotting each metric
metrics = ["Recall_Score", "F1_Score", "Roc_Auc_Score", "Recall_Auc_Score", "Precisior
titles = ["Recall Score", "F1 Score", "ROC AUC Score", "Recall AUC Score", "Precision

for metric, title in zip(metrics, titles):
    compare_sorted = compare.sort_values(by=metric, ascending=True)
    fig = px.bar(compare_sorted, x=metric, y="Model", title=title)
    fig.show()
```

In [210…
```
compare.sort_values(by= "Recall_Score", ascending = False)
```

Out[210]:

	Model	Accuracy_Score	F1_Score	Precision_Score	Recall_Score	Roc_Auc_Score	Recall
8	dL_model	0.999	0.999	0.999	1.000	0.999	
9	dL_Tuned	0.997	0.997	0.995	1.000	0.997	
6	RFSmote_model	0.996	0.996	0.998	0.994	0.996	
1	LogReg_tuned	0.961	0.072	0.037	0.972	0.967	
0	LogReg_model	0.972	0.094	0.049	0.958	0.965	
4	LogRegSmote_model	0.958	0.957	0.973	0.942	0.958	
5	LogRegSmote_tuned	0.955	0.954	0.973	0.936	0.955	
3	RF_tuned	0.999	0.706	0.569	0.930	0.964	
7	RFSmote_tuned	0.942	0.938	0.995	0.888	0.942	
2	RF_model	1.000	0.924	1.000	0.859	0.930	

In [211…
```
cprint('\nFinal Model\n', 'green', 'on_yellow', attrs=['bold'])
```

Final Model

In [212…
```
df_out.corr()['class'].sort_values().drop('class').iplot(kind = 'barh', title = 'Corre
```

In [214…
```
# Assuming RF_smote_model is already fitted and has the attribute feature_importances_
RFSmote_feature_imp = pd.DataFrame(index=X.columns, data = RFSmote_model.feature_impor
fig = px.bar(RFSmote_feature_imp.sort_values('Importance', ascending = False), x = RFS
             ascending = False).index, y = 'Importance', title = "RFSmote Feature_Impo
             labels = dict(x = "Features", y ="Importance"))
fig.show()
```

In [215…

```
# Assuming RF_smote_model is already fitted and has the attribute feature_importances_
RFSmote_feature_imp = pd.DataFrame(index=X.columns, data = RFSmote_tuned.feature_impor
fig = px.bar(RFSmote_feature_imp.sort_values('Importance', ascending = False), x = RFS
             ascending = False).index, y = 'Importance', title = "RFSmote_tuned Featur
             labels = dict(x = "Features", y ="Importance"))
fig.show()
```

In [216…
```
df_deploy = df_out[['v2', 'v3', 'v4', 'v7', 'v10', 'v11', 'v12', 'v14', 'v16', 'v17',
df_deploy.head(1)
```

Out[216]:

	v2	v3	v4	v7	v10	v11	v12	v14	v16	v17	class
0	-0.073	2.536	1.378	0.240	0.091	-0.552	-0.618	-0.311	-0.470	0.208	0

In [217…
```
X = df_deploy.drop(['class'], axis = 1)
y = df_deploy['class']
```

In [218…
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

over = SMOTE(sampling_strategy = {1:10000})
under = RandomUnderSampler(sampling_strategy = {0:10000})
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps = steps)
X, y  = pipeline.fit_resample(X, y)
```

In [219…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.
```

In [220…
```
print('X_train.shape : ', X_train.shape)
print('X_test.shape : ', X_test.shape)
```

```
X_train.shape :  (16000, 10)
X_test.shape :  (4000, 10)
```

In [221…
```
cprint('\ny_trian.value_counts\n', 'black', 'on_white', attrs=['bold'])
```

y_trian.value_counts

In [222…
```
y_train.value_counts()
```

Out[222]:
```
class
1    8000
0    8000
Name: count, dtype: int64
```

In [223…
```
cprint('\ny_test.value_counts\n', 'black', 'on_white', attrs=['bold'])
```

y_test.value_counts

In [224…
```
y_test.value_counts()
```

Out[224]:
```
class
1    2000
0    2000
Name: count, dtype: int64
```

In [236…
```
models = []
models.append(("LogReg_Deploy", LogisticRegression(class_weight = 'balanced', penalty
models.append(("RandomForest_Deploy", RandomForestClassifier(class_weight = 'balanced'
models.append(("KNN_trial", KNeighborsClassifier()))
models.append(("SVC_trial",SVC(class_weight = 'balanced', random_state = 42)))
models.append(("DTC_trial",DecisionTreeClassifier(class_weight = 'balanced', random_st
models.append(("ADA_trial",AdaBoostClassifier(random_state = 42)))
models.append(("GBC_trial",GradientBoostingClassifier(random_state = 42)))
models.append(("XGB_trial",XGBClassifier(random_state = 42, verbosity = 0)))
models.append(("LGB_trial",LGBMClassifier(random_state=42)))
#evaluate each model in turn
results=[]
names = []
f1_scores = []
recall_scores = []
roc_auc_scores = []

for name, model in models :
    kfold = StratifiedKFold(n_splits = 10, shuffle  = True)
    cv_results = cross_val_score(model, X_train, y_train, cv = kfold, scoring = "recal
    results.append(cv_results)
    names.append(name)
    print(f"{name}MODEL:{round(cv_results.mean(),4)}")
    y_pred = model.fit(X_train, y_train).predict(X_test)
    f1_scores.append(f1_score(y_test, y_pred))
    recall_scores.append(recall_score(y_test, y_pred))
    roc_auc_scores.append(roc_auc_score(y_test, y_pred))
result_df = pd.DataFrame(results, columns=[i for i in range(1, 11)], index=names).T
result_df.iplot(kind="box", boxpoints="all", title="CV Results")

compare = pd.DataFrame({"F1": f1_scores, "Recall": recall_scores, "ROC_AUC": roc_auc_s
```

```
for score in compare.columns:
    compare[score].sort_values().iplot(kind="barh", title=f"{score} Score")
```

```
LogReg_DeployMODEL:0.9036
RandomForest_DeployMODEL:0.9484
KNN_trialMODEL:0.998
SVC_trialMODEL:0.8849
DTC_trialMODEL:0.9814
ADA_trialMODEL:0.9381
GBC_trialMODEL:0.9574
XGB_trialMODEL:0.9941
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was
0.000666 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000571 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000639 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000618 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000516 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000324 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000365 seconds.
```

```
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000318 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000606 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 7200, number of negative: 7200
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000310 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 14400, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
LGB_trialMODEL:0.9914
[LightGBM] [Info] Number of positive: 8000, number of negative: 8000
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was
0.000347 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2550
[LightGBM] [Info] Number of data points in the train set: 16000, number of used featu
res: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
```

In [230…

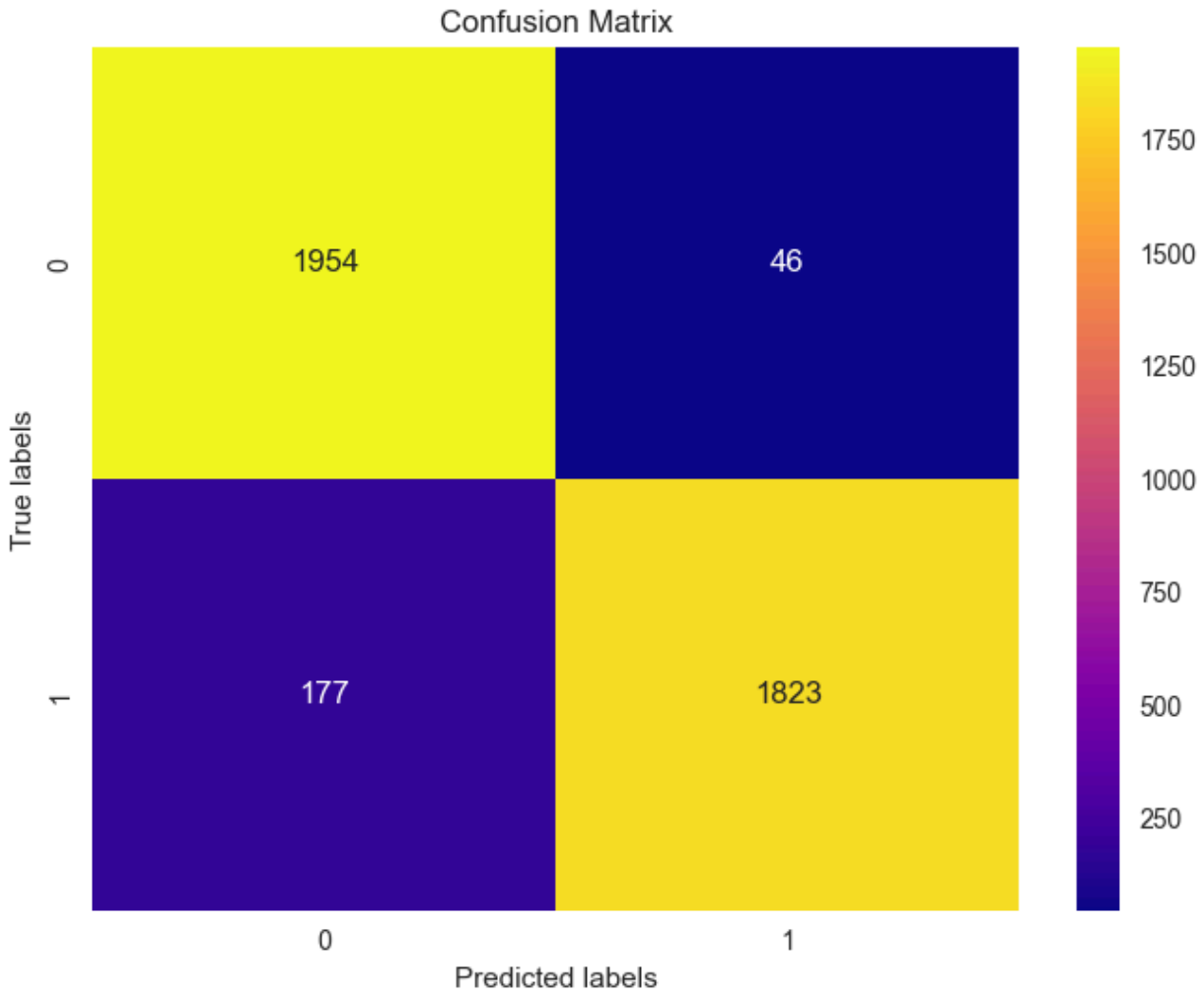
```
cprint('\nFinal model with Logistic Regression (With Feature Importance with SMOTE)\n'
```

Final model with Logistic Regression (With Feature Importance with SMOTE)

In [232…

```
LogReg_Deploy = LogisticRegression(solver='liblinear', class_weight = 'balanced', rand
y_pred = LogReg_Deploy.predict(X_test)
y_train_pred = LogReg_Deploy.predict(X_train)

LogReg_Deploy_f1 = f1_score(y_test, y_pred)
LogReg_Deploy_acc = accuracy_score(y_test, y_pred)
LogReg_Deploy_recall = recall_score(y_test, y_pred)
LogReg_Deploy_auc = roc_auc_score(y_test, y_pred)
LogReg_Deploy_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
LogReg_Deploy_recall_auc = auc(recall, precision)

print("LogReg_Deploy")
print ("------------------")
eval(LogReg_Deploy, X_train, X_test)
```

```
LogReg_Deploy
------------------
[[1954   46]
 [ 177 1823]]
Test_Set
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      2000
           1       0.98      0.91      0.94      2000

    accuracy                           0.94      4000
   macro avg       0.95      0.94      0.94      4000
weighted avg       0.95      0.94      0.94      4000

Train_Set
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      8000
           1       0.98      0.90      0.94      8000

    accuracy                           0.94     16000
   macro avg       0.94      0.94      0.94     16000
weighted avg       0.94      0.94      0.94     16000

------------------------------------------------------------
```

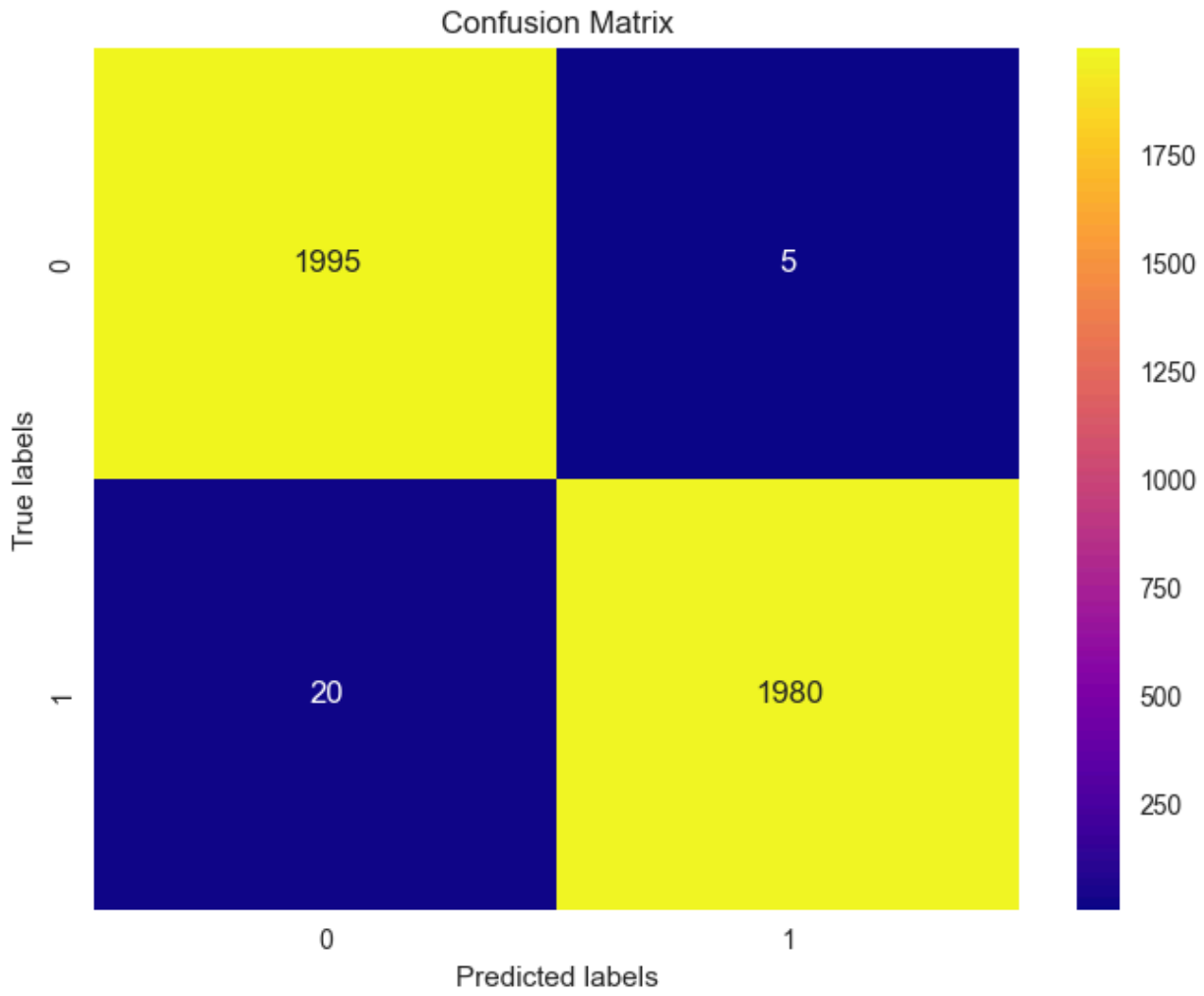


In [ ]:
```
cprint('\nFinal model with Random Forest Classifier(With Feature Importance with SMOTE
```

In [233…

```
RF_Deploy = RandomForestClassifier(class_weight = 'balanced', random_state = 42).fit(X
y_pred = RF_Deploy.predict(X_test)
y_train_pred = RF_Deploy.predict(X_train)

RF_Deploy_f1 = f1_score(y_test, y_pred)
RF_Deploy_acc = accuracy_score(y_test, y_pred)
RF_Deploy_recall = recall_score(y_test, y_pred)
RF_Deploy_auc = roc_auc_score(y_test, y_pred)
RF_Deploy_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
RF_Deployl_recall_auc = auc(recall, precision)

print("RF_Deploy")
print("-----------")
eval(RF_Deploy, X_train, X_test)
```

```
RF_Deploy
-----------
[[1995    5]
 [  20 1980]]
Test_Set
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      2000
           1       1.00      0.99      0.99      2000

    accuracy                           0.99      4000
   macro avg       0.99      0.99      0.99      4000
weighted avg       0.99      0.99      0.99      4000

Train_Set
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8000
           1       1.00      1.00      1.00      8000

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

------------------------------------------------------------
```



In [ ]:
```
cprint('\nFinal model with Artifical Neural Network(With Feature Importance with SMOTE
```

In [243…
```
DL_deploy = Sequential()
DL_deploy.add(Dense(32, activation="relu"))  # Specify activation as argument
DL_deploy.add(Dropout(0.25))
DL_deploy.add(Dense(16, activation="relu"))  # Specify activation as argument
DL_deploy.add(Dropout(0.25))
DL_deploy.add(Dense(1, activation="sigmoid"))

optimizer = Adam(learning_rate=0.001)
DL_deploy.compile(loss='binary_crossentropy',
                  optimizer=optimizer,
                  metrics=["Recall"])

# Predict probabilities
y_pred_proba = DL_deploy.predict(X_test)

# Threshold probabilities to obtain class labels
y_pred = (y_pred_proba > 0.5).astype(int)

# Calculate evaluation metrics
DL_acc = accuracy_score(y_test, y_pred)
DL_AP = average_precision_score(y_test, y_pred_proba)
DL_f1 = f1_score(y_test, y_pred)
DL_rec = recall_score(y_test, y_pred)
DL_roc_auc = roc_auc_score(y_test, y_pred)
```

```
DL_pre = precision_score(y_test, y_pred)
precision, recall, _ = precision_recall_curve(y_test, y_pred)
DL_recall_auc = auc(recall, precision)
```

```
125/125 [==============================] - 0s 929us/step
```