Credit Fraud Detection

INTRODUCTION:

In this project, we'll apply 3 different predictive approaches to compare:

- How accurate we can be;
- How much we can minimize false positives (this becomes a problem when clients have their credit card blocked erroneously).

Since we can't see the real labels of each feature (for security reasons), we cannot infer additional business insights. Nevertheless, we can still deliver a good solution to this issue.

APPROACHES:

- Pycaret anomaly detection;
- Pycaret classifier (100% of our data);
- Classifier with under-over sample (using SMOTE technique).

GOALS:

- Implement models using pycaret modules to see how fast and accurate we can be with few code lines;
- Compare un-supervised models (anomaly detection) with supervised models (Logistic Regression);
- Implement a solution with under & over sample dataframe (using SMOTE technique).

In [1]:

```
# Standard Libraries
import os
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# Pycaret libraries
from pycaret.anomaly import *
# from pycaret.classification import * <- Will be imported when necessary
from pycaret.utils import enable_colab
enable_colab()


# Sklearn libraries
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import accuracy_score, roc_auc_score, recall_score, precision_scor
e, f1_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import StratifiedShuffleSplit, KFold, StratifiedKFold
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
from sklearn.linear_model import LogisticRegression


# Other libraries
from imblearn.over_sampling import SMOTE
import seaborn as sns
from functools import reduce


# Change pandas columns limit
pd.set_option('display.max_columns', 100)
```

```
Colab mode enabled.
```

In [2]:

```
# Path
os.chdir('C:/Users/ttandozia/Desktop/KAGGLE_FRAUD')
```

In [3]:

```
# Dataset
df = pd.read_csv('creditcard.csv', sep=',')
```

In [4]:

```
# Scale features "Time" and "Amount"
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

# Drop original columns
df.drop(['Time','Amount'], axis=1, inplace=True)

# Reorder columns
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']
df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)
```

We'll create the "original" train and test so we can compare the 3 approches with the same test set.

For each approach, we will split the data using random_state (or setion_id in pycaret) = 1313

In [5]:

```
# Spliting data
X = df.drop('Class', axis=1)
y = df['Class']

skf = StratifiedKFold(n_splits=5, random_state=1313, shuffle=False)

for train_index, test_index in skf.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

original_df_train = pd.merge(original_Xtrain, original_ytrain, left_index=True, right_i
ndex=True)
original_df_test = pd.merge(original_Xtest, original_ytest, left_index=True, right_inde
x=True)
```

1º APPROACH - ANOMALY DETECTION WITH PYCARET

In [6]:

```
# Pycaret Anomaly Setup
anom_setup = setup(data = original_df_train,
                   session_id=1313, # sklearn "random_state"
                   ignore_features=['Class'],
                   preprocess=True,
                   imputation_type='iterative',
                   iterative_imputation_iters=30,
                   combine_rare_levels=True,
                   rare_level_threshold=0.01,
                   numeric_imputation='median')
```

	Description	Value
0	session_id	1313
1	Original Data	(227846, 31)
2	Missing Values	False
3	Numeric Features	30
4	Categorical Features	0
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(227846, 30)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	anomaly-default-name
13	USI	e13c
14	Imputation Type	iterative
15	Iterative Imputation Iteration	30
16	Numeric Imputer	median
17	Iterative Imputation Numeric Model	Light Gradient Boosting Machine
18	Categorical Imputer	mode
19	Iterative Imputation Categorical Model	Light Gradient Boosting Machine
20	Unknown Categoricals Handling	least_frequent
21	Normalize	False
22	Normalize Method	None
23	Transformation	False
24	Transformation Method	None
25	PCA	False
26	PCA Method	None
27	PCA Components	None
28	Ignore Low Variance	False
29	Combine Rare Levels	True
30	Rare Level Threshold	0.010000
31	Numeric Binning	False
32	Remove Outliers	False
33	Outliers Threshold	None
34	Remove Multicollinearity	False
35	Multicollinearity Threshold	None
36	Clustering	False
37	Clustering Iteration	None

	Description	Value
38	Polynomial Features	False
39	Polynomial Degree	None
40	Trignometry Features	False
41	Polynomial Threshold	None
42	Group Features	False
43	Feature Selection	False
44	Features Selection Threshold	None
45	Feature Interaction	False
46	Feature Ratio	False
47	Interaction Threshold	None

ANOMALY MODELS:

KNN

In [7]:

```
# Creating model
t0 = time.time()
anom_model_knn = create_model('knn')
t1 = time.time()
print("Fitting knn took :{} sec".format(t1 - t0))
```

```
Fitting knn took :402.725647687912 sec
```

In [8]:

```
# Assign Label with data
anom_df_knn = assign_model(anom_model_knn)
```

In [9]:

```
# Check volumes
anom_df_knn['Anomaly'].value_counts()
```

Out[9]:

```
0    216454
1     11392
Name: Anomaly, dtype: int64
```

In [10]:

```
# Crosstab wit target
pd.crosstab(anom_df_knn['Class'], anom_df_knn['Anomaly']).stack().reset_index(name='Fre
q')
```

Out[10]:

	Class	Anomaly	Freq
0	0	0	216376
1	0	1	11076
2	1	0	78
3	1	1	316

In [11]:

```
# Saving model
save_model(anom_df_knn, 'ANOM_KNN', verbose=False)
```

Out[11]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['Class'], id_colum
ns=[],
                                       ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Iterative_Imputer(add_indicator=False,
                                    classifier=LGBMClassifier(boosting_typ
e='gbdt',
                                                              class_weigh
t...
 227865  0.488747 -0.063826 -0.610194  0.007487 -0.013918      0        0
 227866  0.468036  0.571794 -0.403076  0.259078  0.077267      0        0
 227867  0.700880 -0.769575 -0.193723  0.143983  0.134559      0        0
 227868  0.609969 -0.695184  0.463574  0.119990  0.134411      0        0

         Anomaly_Score
 0            1.881069
 1            0.416298
 2            3.652251
 3            1.880649
 4            1.969569
 ...               ...
 227864       3.258724
 227865       0.583978
 227866       3.166095
 227867       2.393118
 227868       1.101679

 [227846 rows x 33 columns]]],
          verbose=False),
 'ANOM_KNN.pkl')
```

Isolation Forest

In [12]:

```
# Creating model
t0 = time.time()
anom_model_ift = create_model('iforest')
t1 = time.time()
print("Fitting ift took :{} sec".format(t1 - t0))
```

```
Fitting ift took :23.881831884384155 sec
```

In [13]:

```
# Assign Label with data
anom_df_ift = assign_model(anom_model_ift)
```

In [14]:

```
# Check volumes
anom_df_ift['Anomaly'].value_counts()
```

Out[14]:

```
0    216453
1     11393
Name: Anomaly, dtype: int64
```

In [15]:

```
# Crosstab wit target
pd.crosstab(anom_df_ift['Class'], anom_df_ift['Anomaly']).stack().reset_index(name='Fre
q')
```

Out[15]:

	Class	Anomaly	Freq
0	0	0	216400
1	0	1	11052
2	1	0	53
3	1	1	341

In [16]:

```
# Saving model
save_model(anom_df_ift, 'ANOM_IFT', verbose=False)
```

Out[16]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['Class'], id_colum
ns=[],
                                       ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Iterative_Imputer(add_indicator=False,
                                    classifier=LGBMClassifier(boosting_typ
e='gbdt',
                                                              class_weigh
t...
 227865  0.488747 -0.063826 -0.610194  0.007487 -0.013918      0        0
 227866  0.468036  0.571794 -0.403076  0.259078  0.077267      0        0
 227867  0.700880 -0.769575 -0.193723  0.143983  0.134559      0        0
 227868  0.609969 -0.695184  0.463574  0.119990  0.134411      0        0

         Anomaly_Score
 0           -0.096091
 1           -0.108611
 2           -0.033654
 3           -0.063376
 4           -0.093594
 ...               ...
 227864      -0.006987
 227865      -0.099860
 227866      -0.058475
 227867      -0.022128
 227868      -0.064628

 [227846 rows x 33 columns]]],
          verbose=False),
 'ANOM_IFT.pkl')
```

Clustering-Based Local Outlier

In [17]:

```
# Creating model
t0 = time.time()
anom_model_clt = create_model('cluster')
t1 = time.time()
print("Fitting clt took :{} sec".format(t1 - t0))
```

```
Fitting clt took :20.332393646240234 sec
```

In [18]:

```
# Assign Label with data
anom_df_clt = assign_model(anom_model_clt)
```

In [19]:

```
# Check volumes
anom_df_clt['Anomaly'].value_counts()
```

Out[19]:

```
0    216453
1     11393
Name: Anomaly, dtype: int64
```

In [20]:

```
# Crosstab wit target
pd.crosstab(anom_df_clt['Class'], anom_df_clt['Anomaly']).stack().reset_index(name='Fre
q')
```

Out[20]:

	Class	Anomaly	Freq
0	0	0	216407
1	0	1	11045
2	1	0	46
3	1	1	348

In [21]:

```
# Saving model
save_model(anom_df_clt, 'ANOM_CLT', verbose=False)
```

Out[21]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['Class'], id_colum
ns=[],
                                       ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Iterative_Imputer(add_indicator=False,
                                    classifier=LGBMClassifier(boosting_typ
e='gbdt',
                                                              class_weigh
t...
 227865  0.488747 -0.063826 -0.610194  0.007487 -0.013918      0        0
 227866  0.468036  0.571794 -0.403076  0.259078  0.077267      0        0
 227867  0.700880 -0.769575 -0.193723  0.143983  0.134559      0        0
 227868  0.609969 -0.695184  0.463574  0.119990  0.134411      0        0

         Anomaly_Score
 0            3.802538
 1            2.585942
 2            5.757281
 3            4.202669
 4            3.252548
 ...               ...
 227864       6.945799
 227865       3.370518
 227866       4.965433
 227867       4.245115
 227868       6.144709

 [227846 rows x 33 columns]]],
          verbose=False),
 'ANOM_CLT.pkl')
```

Histogram-based Outlier Detection

In [22]:

```
# Creating model
t0 = time.time()
anom_model_his = create_model('histogram')
t1 = time.time()
print("Fitting his took :{} sec".format(t1 - t0))
```

```
Fitting his took :2.0970592498779297 sec
```

In [23]:

```
# Assign Label with data
anom_df_his = assign_model(anom_model_his)
```

In [24]:

```
# Check volumes
anom_df_his['Anomaly'].value_counts()
```

Out[24]:

```
0    216453
1     11393
Name: Anomaly, dtype: int64
```

In [25]:

```
# Crosstab wit target
pd.crosstab(anom_df_his['Class'], anom_df_his['Anomaly']).stack().reset_index(name='Fre
q')
```

Out[25]:

	Class	Anomaly	Freq
0	0	0	216398
1	0	1	11054
2	1	0	55
3	1	1	339

In [26]:

```
# Saving model
save_model(anom_df_his, 'ANOM_HIS', verbose=False)
```

Out[26]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['Class'], id_colum
ns=[],
                                       ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Iterative_Imputer(add_indicator=False,
                                    classifier=LGBMClassifier(boosting_typ
e='gbdt',
                                                              class_weigh
t...
 227865  0.488747 -0.063826 -0.610194  0.007487 -0.013918      0        0
 227866  0.468036  0.571794 -0.403076  0.259078  0.077267      0        0
 227867  0.700880 -0.769575 -0.193723  0.143983  0.134559      0        0
 227868  0.609969 -0.695184  0.463574  0.119990  0.134411      0        0

         Anomaly_Score
 0           49.820639
 1           48.483055
 2           54.262055
 3           55.701699
 4           49.609469
 ...               ...
 227864      53.114194
 227865      46.658967
 227866      55.866608
 227867      53.992354
 227868      54.743467

 [227846 rows x 33 columns]]],
          verbose=False),
 'ANOM_HIS.pkl')
```

Principal Component Analysis

In [27]:

```
# Creating model
t0 = time.time()
anom_model_pca = create_model('pca')
t1 = time.time()
print("Fitting pca took :{} sec".format(t1 - t0))
```

```
Fitting pca took :3.8145229816436768 sec
```

In [28]:

```
# Assign Label with data
anom_df_pca = assign_model(anom_model_pca)
```

In [29]:

```
# Check volumes
anom_df_pca['Anomaly'].value_counts()
```

Out[29]:

```
0    216453
1     11393
Name: Anomaly, dtype: int64
```

In [30]:

```
# Crosstab wit target
pd.crosstab(anom_df_pca['Class'], anom_df_pca['Anomaly']).stack().reset_index(name='Fre
q')
```

Out[30]:

	Class	Anomaly	Freq
0	0	0	216407
1	0	1	11045
2	1	0	46
3	1	1	348

In [31]:

```
# Saving model
save_model(anom_df_pca, 'ANOM_PCA', verbose=False)
```

Out[31]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['Class'], id_colum
ns=[],
                                       ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Iterative_Imputer(add_indicator=False,
                                    classifier=LGBMClassifier(boosting_typ
e='gbdt',
                                                              class_weigh
t...
 227866  0.468036  0.571794 -0.403076  0.259078  0.077267      0        0
 227867  0.700880 -0.769575 -0.193723  0.143983  0.134559      0        0
 227868  0.609969 -0.695184  0.463574  0.119990  0.134411      0        0

         Anomaly_Score
 0         6191.736288
 1         5487.197142
 2        10978.804240
 3         8259.160986
 4         6668.237090
 ...               ...
 227864   11803.761272
 227865    6443.536981
 227866    9706.444952
 227867   10653.463355
 227868    9367.867973

 [227846 rows x 33 columns]]],
          verbose=False),
 'ANOM_PCA.pkl')
```

Minimum Covariance Determinant

In [32]:

```
# Creating model
t0 = time.time()
anom_model_mcd = create_model('mcd')
t1 = time.time()
print("Fitting mcd took :{} sec".format(t1 - t0))
```

```
Fitting mcd took :142.09781455993652 sec
```

In [33]:

```
# Assign Label with data
anom_df_mcd = assign_model(anom_model_mcd)
```

In [34]:

```
# Check volumes
anom_df_mcd['Anomaly'].value_counts()
```

Out[34]:

```
0    216453
1     11393
Name: Anomaly, dtype: int64
```

In [35]:

```
# Crosstab wit target
pd.crosstab(anom_df_mcd['Class'], anom_df_mcd['Anomaly']).stack().reset_index(name='Fre
q')
```

Out[35]:

	Class	Anomaly	Freq
0	0	0	216218
1	0	1	11234
2	1	0	235
3	1	1	159

In [36]:

```
# Saving model
save_model(anom_df_mcd, 'ANOM_MCD', verbose=False)
```

Out[36]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True,
                                       features_todrop=['Class'], id_colum
ns=[],
                                       ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Iterative_Imputer(add_indicator=False,
                                    classifier=LGBMClassifier(boosting_typ
e='gbdt',
                                                              class_weigh
t...
 227865  0.488747 -0.063826 -0.610194  0.007487 -0.013918      0        0
 227866  0.468036  0.571794 -0.403076  0.259078  0.077267      0        0
 227867  0.700880 -0.769575 -0.193723  0.143983  0.134559      0        0
 227868  0.609969 -0.695184  0.463574  0.119990  0.134411      0        0

         Anomaly_Score
 0         5657.766140
 1           16.152076
 2        71542.468683
 3         7348.447465
 4         2264.473518
 ...               ...
 227864    1195.730918
 227865      12.976124
 227866    2978.723918
 227867     645.241759
 227868     798.000279

 [227846 rows x 33 columns]]],
          verbose=False),
 'ANOM_MCD.pkl')
```

In [37]:

```
# Sintetic information
KNN = pd.crosstab(anom_df_knn['Class'], anom_df_knn['Anomaly']).stack().reset_index(nam
e='Freq')
KNN = KNN.rename(columns={'Freq': 'KNN'})

IFT = pd.crosstab(anom_df_ift['Class'], anom_df_ift['Anomaly']).stack().reset_index(nam
e='Freq')
IFT = IFT.rename(columns={'Freq': 'IFT'})

CLT = pd.crosstab(anom_df_clt['Class'], anom_df_clt['Anomaly']).stack().reset_index(nam
e='Freq')
CLT = CLT.rename(columns={'Freq': 'CLT'})

HIS = pd.crosstab(anom_df_his['Class'], anom_df_his['Anomaly']).stack().reset_index(nam
e='Freq')
HIS = HIS.rename(columns={'Freq': 'HIS'})

PCA = pd.crosstab(anom_df_pca['Class'], anom_df_pca['Anomaly']).stack().reset_index(nam
e='Freq')
PCA = PCA.rename(columns={'Freq': 'PCA'})

MCD = pd.crosstab(anom_df_mcd['Class'], anom_df_mcd['Anomaly']).stack().reset_index(nam
e='Freq')
MCD = MCD.rename(columns={'Freq': 'MCD'})
```

In [38]:

```
# Merge train results
models_list = [KNN, IFT, CLT, HIS, PCA, MCD]
results = reduce(lambda  left,right: pd.merge(left,right, how='left'), models_list)
results.head()
```

Out[38]:

	Class	Anomaly	KNN	IFT	CLT	HIS	PCA	MCD
0	0	0	216376	216400	216407	216398	216407	216218
1	0	1	11076	11052	11045	11054	11045	11234
2	1	0	78	53	46	55	46	235
3	1	1	316	341	348	339	348	159

Now we'll predict our anomalies in the test set

In [39]:

```
# Test dataframes
KNN_TEST = predict_model(model=anom_model_knn, data=original_df_test)
IFT_TEST = predict_model(model=anom_model_ift, data=original_df_test)
CLT_TEST = predict_model(model=anom_model_clt, data=original_df_test)
HIS_TEST = predict_model(model=anom_model_his, data=original_df_test)
PCA_TEST = predict_model(model=anom_model_pca, data=original_df_test)
MCD_TEST = predict_model(model=anom_model_mcd, data=original_df_test)
```

In [40]:

```
# Sintetic information of test dataframe
KNN = pd.crosstab(KNN_TEST['Class'], KNN_TEST['Anomaly']).stack().reset_index(name='Fre
q')
KNN = KNN.rename(columns={'Freq': 'KNN'})

IFT = pd.crosstab(IFT_TEST['Class'], IFT_TEST['Anomaly']).stack().reset_index(name='Fre
q')
IFT = IFT.rename(columns={'Freq': 'IFT'})

CLT = pd.crosstab(CLT_TEST['Class'], CLT_TEST['Anomaly']).stack().reset_index(name='Fre
q')
CLT = CLT.rename(columns={'Freq': 'CLT'})

HIS = pd.crosstab(HIS_TEST['Class'], HIS_TEST['Anomaly']).stack().reset_index(name='Fre
q')
HIS = HIS.rename(columns={'Freq': 'HIS'})

PCA = pd.crosstab(PCA_TEST['Class'], PCA_TEST['Anomaly']).stack().reset_index(name='Fre
q')
PCA = PCA.rename(columns={'Freq': 'PCA'})

MCD = pd.crosstab(MCD_TEST['Class'], MCD_TEST['Anomaly']).stack().reset_index(name='Fre
q')
MCD = MCD.rename(columns={'Freq': 'MCD'})
```

In [41]:

```
# Merge test results
models_list = [KNN, IFT, CLT, HIS, PCA, MCD]
results = reduce(lambda  left,right: pd.merge(left,right, how='left'), models_list)
results.head()
```

Out[41]:

	Class	Anomaly	KNN	IFT	CLT	HIS	PCA	MCD
0	0	0	53755	53998	54206	53974	54041	54436
1	0	1	3108	2865	2657	2889	2822	2427
2	1	0	17	16	14	17	15	72
3	1	1	81	82	84	81	83	26

We can see that Clustering-Based Local Outlier "CLT" model had the best performance when alocating 5% of our training data as anomaly, and predicting in test set.

Note: Its development time was 95% less than knn, for example. (It took 20 sec. Knn took 402 sec.)

Let's see the best anomaly model metrics

In [42]:

```
# Fix formats
CLT_TEST['Anomaly'] = CLT_TEST['Anomaly'].astype(float)
CLT_TEST['Class'] = CLT_TEST['Class'].astype(float)
```

In [43]:

```
# Metrics
acc = accuracy_score(CLT_TEST['Class'], CLT_TEST['Anomaly'])
auc = roc_auc_score(CLT_TEST['Class'], CLT_TEST['Anomaly_Score'])
recall = recall_score(CLT_TEST['Class'], CLT_TEST['Anomaly'])
precision = precision_score(CLT_TEST['Class'], CLT_TEST['Anomaly'])
f1 = f1_score(CLT_TEST['Class'], CLT_TEST['Anomaly'])
```

In [44]:

```
# Visualize metrics
cols = ['Model','Accurary','AUC','Recall','Prec.','F1']
values = ['CLT',acc,auc,recall,precision,f1]
metrics_df = pd.DataFrame({tup[0]: [tup[1]] for tup in zip(cols, values)})
metrics_df
```

Out[44]:

	Model	Accurary	AUC	Recall	Prec.	F1
0	CLT	0.953108	0.960965	0.857143	0.030646	0.059176

In [45]:

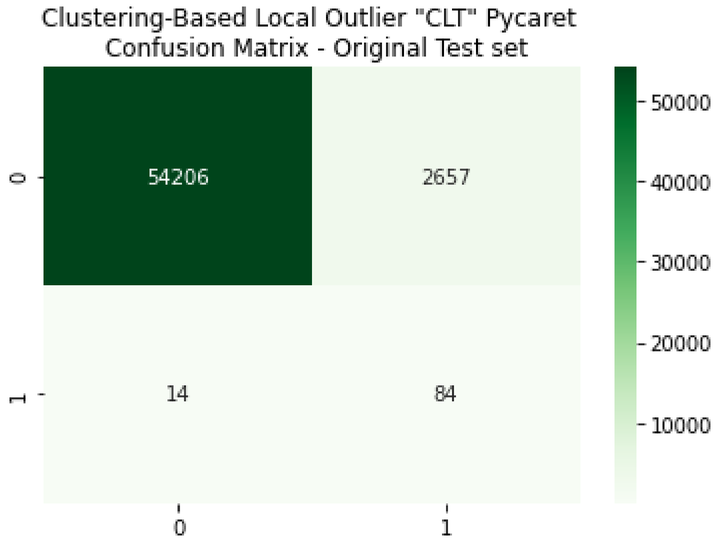
```
# See real target and predicted label
pd.crosstab(CLT_TEST['Class'], CLT_TEST['Anomaly']).stack().reset_index(name='Freq')
```

Out[45]:

	Class	Anomaly	Freq
0	0.0	0.0	54206
1	0.0	1.0	2657
2	1.0	0.0	14
3	1.0	1.0	84

In [46]:

```
# Plot confusion matrix
conf_matx_anom = confusion_matrix(original_ytest, CLT_TEST['Anomaly'])
ax = plt.axes()
sns.heatmap(conf_matx_anom, annot=True, cmap='Greens', fmt='.0f')
ax.set_title('Clustering-Based Local Outlier "CLT" Pycaret \n Confusion Matrix - Origin
al Test set')
plt.show()
```



2º APPROACH - CLASSIFIER WITH PYCARET

In [47]:

```
# Pycaret Classifier Setup

from pycaret.classification import *

ml_setup = setup(data = df,
                 target = 'Class',
                 session_id=1313,
                 feature_selection = True,
                 feature_selection_threshold=0.7,
                 fix_imbalance=True,
                 remove_multicollinearity = True,
                 multicollinearity_threshold = 0.4,
                 normalize=True,
                 normalize_method='robust',
                 remove_outliers=True,
                 train_size=0.8,
                 ignore_low_variance=True)
```

	Description	Value
0	session_id	1313
1	Target	Class
2	Target Type	Binary
3	Label Encoded	0: 0, 1: 1
4	Original Data	(284807, 31)
5	Missing Values	False
6	Numeric Features	30
7	Categorical Features	0
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(216453, 23)
12	Transformed Test Set	(56962, 23)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	0f5c
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	True
30	Normalize Method	robust
31	Transformation	False
32	Transformation Method	None
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	True
37	Combine Rare Levels	False

	Description	Value
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	True
41	Outliers Threshold	0.050000
42	Remove Multicollinearity	True
43	Multicollinearity Threshold	0.400000
44	Clustering	False
45	Clustering Iteration	None
46	Polynomial Features	False
47	Polynomial Degree	None
48	Trignometry Features	False
49	Polynomial Threshold	None
50	Group Features	False
51	Feature Selection	True
52	Features Selection Threshold	0.700000
53	Feature Interaction	False
54	Feature Ratio	False
55	Interaction Threshold	None
56	Fix Imbalance	True
57	Fix Imbalance Method	SMOTE

In [48]:

```
# Compare models
compare_models(fold=5)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9998	0.6647	0.0200	0.1000	0.0333	0.0333	0.0446	149.3700
et	Extra Trees Classifier	0.9998	0.7653	0.0644	0.4000	0.1097	0.1096	0.1585	37.6760
xgboost	Extreme Gradient Boosting	0.9994	0.7042	0.1089	0.0580	0.0749	0.0747	0.0787	110.2380
catboost	CatBoost Classifier	0.9990	0.6903	0.0867	0.0278	0.0415	0.0412	0.0482	89.9040
lightgbm	Light Gradient Boosting Machine	0.9987	0.7534	0.1533	0.0303	0.0505	0.0502	0.0676	5.8600
dt	Decision Tree Classifier	0.9986	0.5094	0.0200	0.0033	0.0056	0.0053	0.0076	20.4940
knn	K Neighbors Classifier	0.9981	0.5966	0.1511	0.0174	0.0312	0.0308	0.0506	148.5900
gbc	Gradient Boosting Classifier	0.9725	0.7448	0.3489	0.0025	0.0050	0.0046	0.0276	246.3660
qda	Quadratic Discriminant Analysis	0.9550	0.6392	0.1956	0.0010	0.0019	0.0015	0.0108	1.6720
ada	Ada Boost Classifier	0.9463	0.7168	0.4378	0.0017	0.0034	0.0030	0.0244	49.2880
nb	Naive Bayes	0.8928	0.7909	0.5933	0.0012	0.0023	0.0019	0.0226	0.6180
lr	Logistic Regression	0.8706	0.8064	0.6800	0.0011	0.0022	0.0018	0.0237	5.4500
svm	SVM - Linear Kernel	0.8638	0.0000	0.6778	0.0010	0.0021	0.0017	0.0229	1.0960
ridge	Ridge Classifier	0.8580	0.0000	0.6778	0.0010	0.0020	0.0016	0.0222	0.5440
lda	Linear Discriminant Analysis	0.8580	0.8107	0.6778	0.0010	0.0020	0.0016	0.0222	2.7160

Out[48]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=1313, verb
ose=0,
                       warm_start=False)
```

We'll create a Logistic Regression since it has the best Recall and its Accuracy and AUC was acceptable

In [49]:

```
# Creating model
ml_model = create_model(estimator='lr',fold=5)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8435	0.9437	0.8889	0.0012	0.0024	0.0019	0.0290
1	0.8715	0.7158	0.5556	0.0009	0.0018	0.0014	0.0184
2	0.8817	0.7319	0.4000	0.0008	0.0016	0.0011	0.0133
3	0.8886	0.7514	0.6667	0.0012	0.0025	0.0021	0.0254
4	0.8675	0.8893	0.8889	0.0014	0.0028	0.0024	0.0322
Mean	0.8706	0.8064	0.6800	0.0011	0.0022	0.0018	0.0237
SD	0.0154	0.0922	0.1904	0.0002	0.0005	0.0005	0.0069

In [50]:

```
# Predict
predict_model(ml_model)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.8518	0.9754	0.9500	0.0111	0.0220	0.0186	0.0940

Out[50]:

	V3	V8	V18	V27	V13	V7	V10	V5
0	0.556695	0.170458	0.372034	1.618953	0.356857	0.507318	-0.061292	0.363580
1	0.377311	-0.081743	0.128819	0.520264	1.711790	-1.321332	1.302402	-1.190163
2	-0.287048	-0.381197	0.167499	0.741556	-0.381046	0.864501	0.085946	0.877828
3	0.419974	0.006018	0.492145	0.060495	1.141342	0.338917	-0.410922	0.829842
4	-0.660386	-0.044080	0.121514	2.358651	0.634921	1.493415	-0.532053	-0.206764
...	...	...	...	...	...	...	...	...
56957	0.048762	1.510391	0.572402	-2.422383	-1.261917	-0.011089	-0.599377	-0.443520
56958	0.960143	1.793332	-0.354809	-0.455405	-0.148154	-0.351384	-0.863984	-0.158361
56959	-0.525572	0.335239	1.287645	0.163196	-0.142998	-0.493511	0.366743	-0.073525
56960	0.944113	1.549350	0.477104	1.429708	-0.489745	-0.217013	-0.223748	-0.347772
56961	0.347771	-0.543829	-0.285312	0.090078	-0.218384	1.071489	-0.500105	0.537405

56962 rows × 26 columns

The tune model will try to optimize the recall using hyperparameters

In [51]:

```
# Tune model
ml_model_tuned = tune_model(ml_model, fold=5,
                            optimize='Recall',
                            n_iter=30)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8435	0.9437	0.8889	0.0012	0.0024	0.0019	0.0290
1	0.8715	0.7168	0.5556	0.0009	0.0018	0.0014	0.0184
2	0.8816	0.7298	0.4000	0.0008	0.0016	0.0011	0.0133
3	0.8886	0.7520	0.6667	0.0012	0.0025	0.0021	0.0254
4	0.8675	0.8891	0.8889	0.0014	0.0028	0.0024	0.0321
Mean	0.8705	0.8063	0.6800	0.0011	0.0022	0.0018	0.0237
SD	0.0154	0.0923	0.1904	0.0002	0.0005	0.0005	0.0069

In [52]:

```
# Tune Predict
predictions = predict_model(ml_model_tuned)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.8519	0.9754	0.9500	0.0111	0.0220	0.0186	0.0941

Even though the training had 68% Recall, when applied to the test set, not only the model was able to maintain the accuracy and AUC, but it improved the Recall significantly.

Test set metrics:

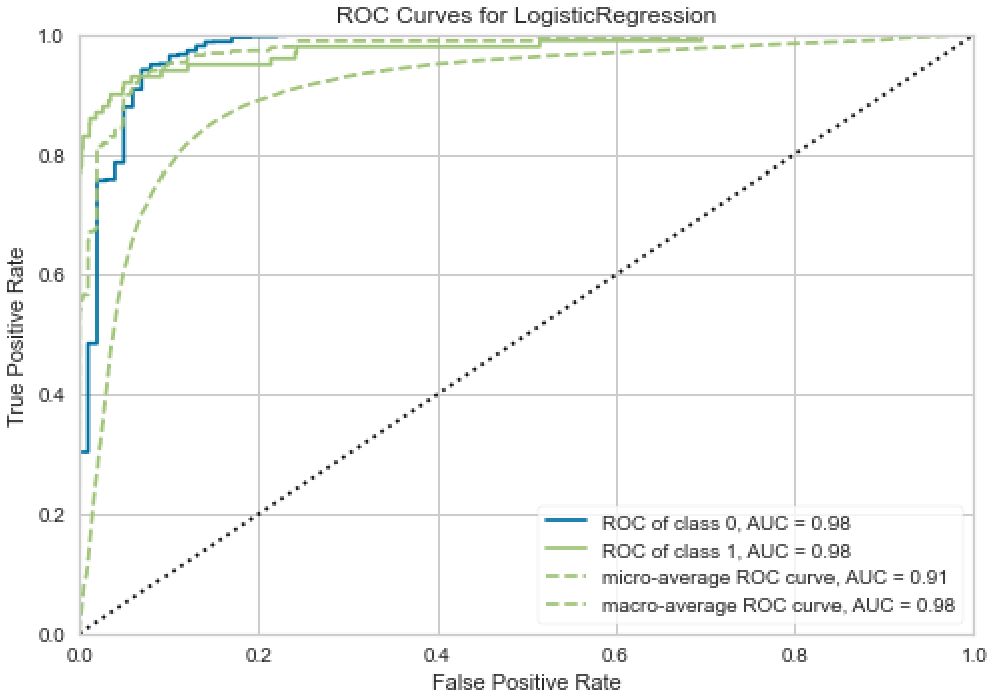
In [53]:

```
# Evaluating
evaluate_model(ml_model_tuned)
```

	Parameters
C	3.872
class_weight	balanced
dual	False
fit_intercept	True
intercept_scaling	1
l1_ratio	None
max_iter	1000
multi_class	auto
n_jobs	None
penalty	l2
random_state	1313
solver	lbfgs
tol	0.0001
verbose	0
warm_start	False

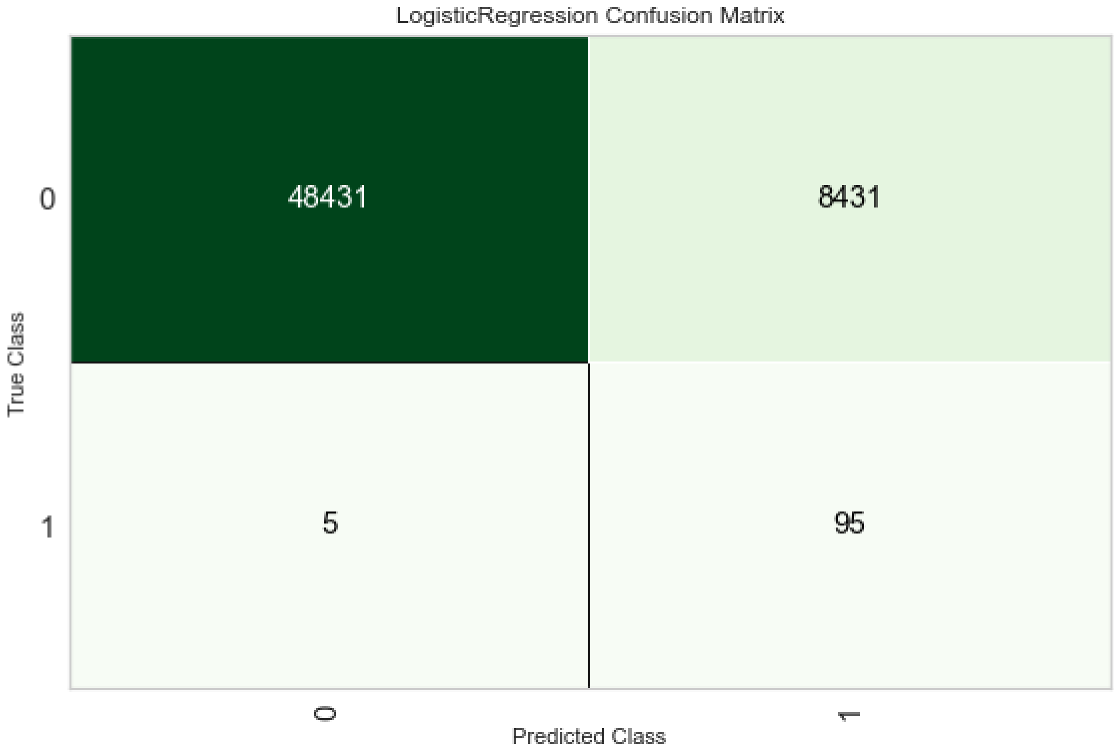
In [54]:

```
# AUC
plot_model(ml_model_tuned)
```



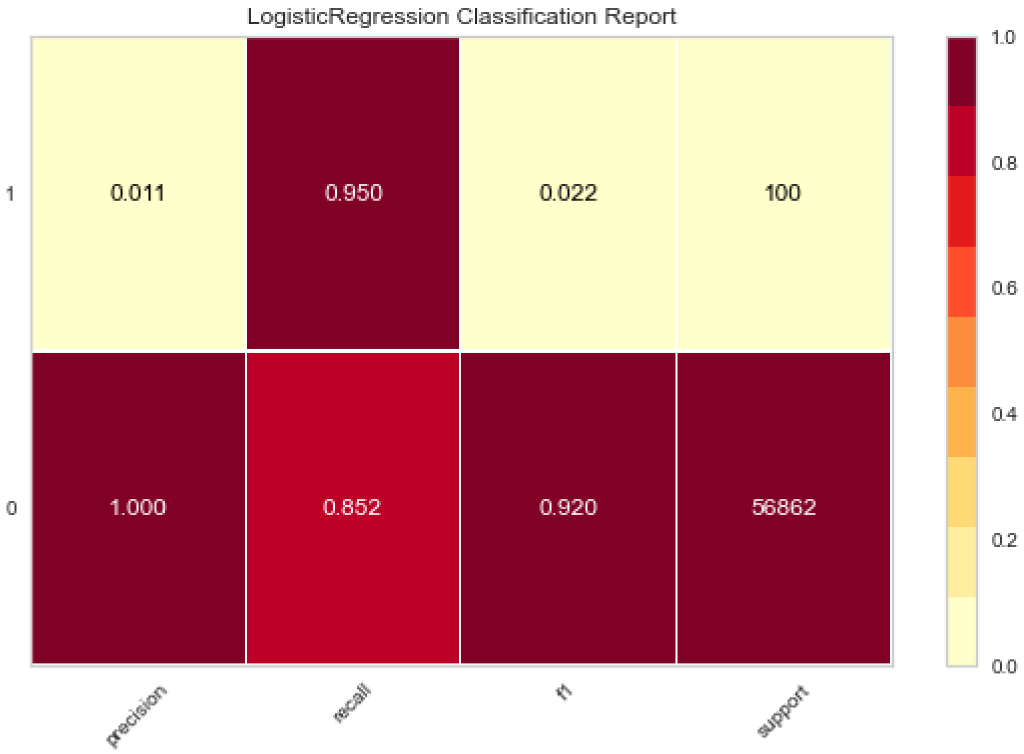
In [55]:

```
# Confusion matrix
plot_model(ml_model_tuned, plot='confusion_matrix')
```



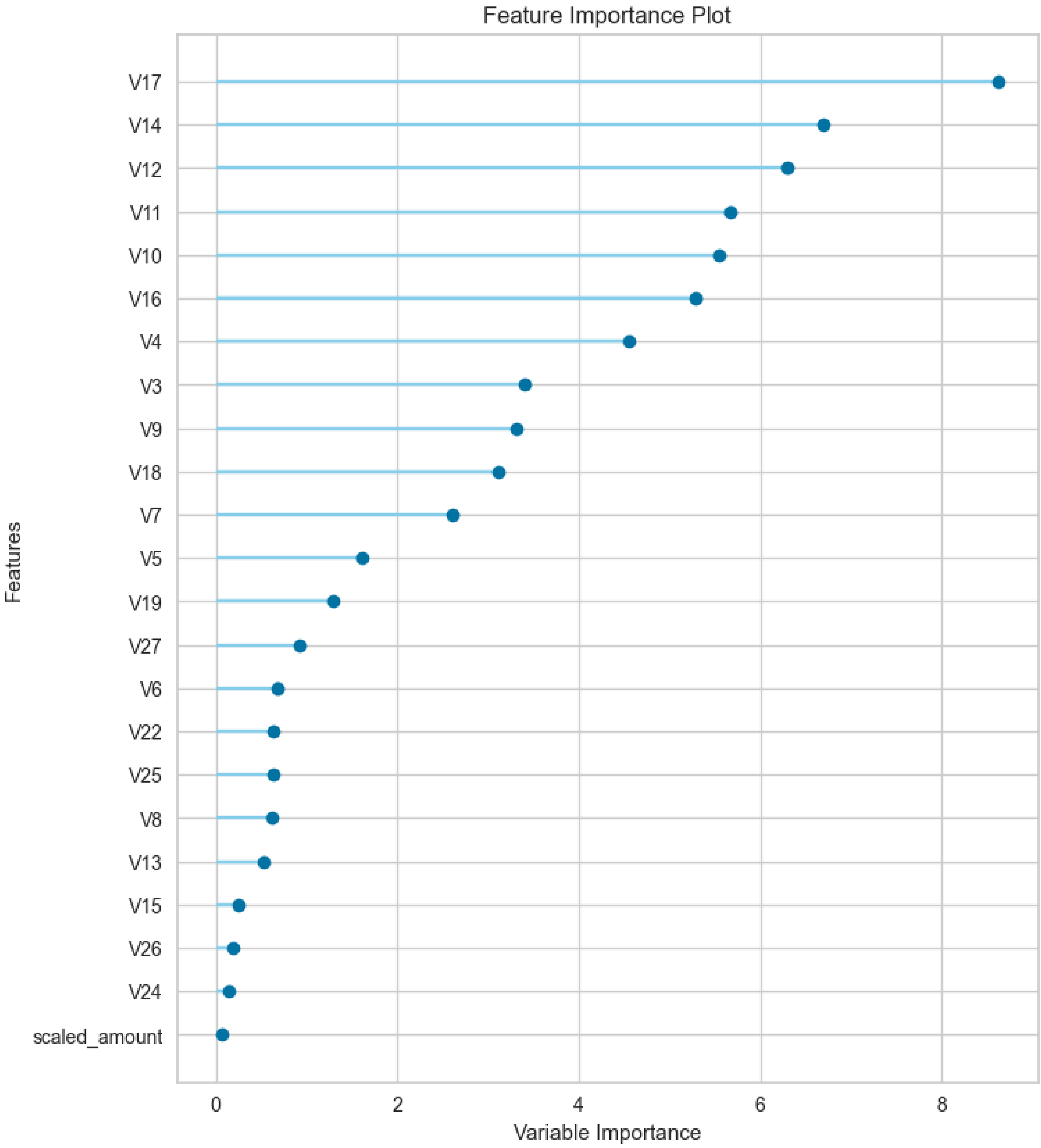
In [56]:

```
# Class report
plot_model(ml_model_tuned, plot='class_report')
```



In [57]:

```
# Feature importance
plot_model(ml_model_tuned, plot='feature_all')
```



In [58]:

```
# Finalize model
final_ml_tuned = finalize_model(ml_model_tuned)
```

In [59]:

```
# Saving model
save_model(final_ml_tuned, 'LR_PYCARET')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[59]:

```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop
=[],
                                       id_columns=[],
                                       ml_usecase='classification',
                                       numerical_features=[], target='Clas
s',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='not_available',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None,
                                 numeric_strate...
                                        target_variable='Class',
                                        threshold=0.4)),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  LogisticRegression(C=3.872, class_weight='balanced',
                                     dual=False, fit_intercept=True,
                                     intercept_scaling=1, l1_ratio=None,
                                     max_iter=1000, multi_class='auto',
                                     n_jobs=None, penalty='l2',
                                     random_state=1313, solver='lbfgs',
                                     tol=0.0001, verbose=0, warm_start=Fal
se)]],
          verbose=False),
 'LR_PYCARET.pkl')
```

In [60]:

```
# Fix formats
predictions['Label'] = predictions['Label'].astype(float)
predictions['Class'] = predictions['Class'].astype(float)
```

In [61]:

```
# Metrics
acc = accuracy_score(predictions['Class'], predictions['Label'])
auc = roc_auc_score(predictions['Class'], predictions['Score'])
recall = recall_score(predictions['Class'], predictions['Label'])
precision = precision_score(predictions['Class'], predictions['Label'])
f1 = f1_score(predictions['Class'], predictions['Label'])
```

In [62]:

```
# Visualize metrics
cols = ['Model','Accurary','AUC','Recall','Prec.','F1']
values = ['Random Forest',acc,auc,recall,precision,f1]
metrics_df = pd.DataFrame({tup[0]: [tup[1]] for tup in zip(cols, values)})
metrics_df
```

Out[62]:

	Model	Accurary	AUC	Recall	Prec.	F1
0	Random Forest	0.851901	0.879819	0.95	0.011142	0.022026

In [63]:

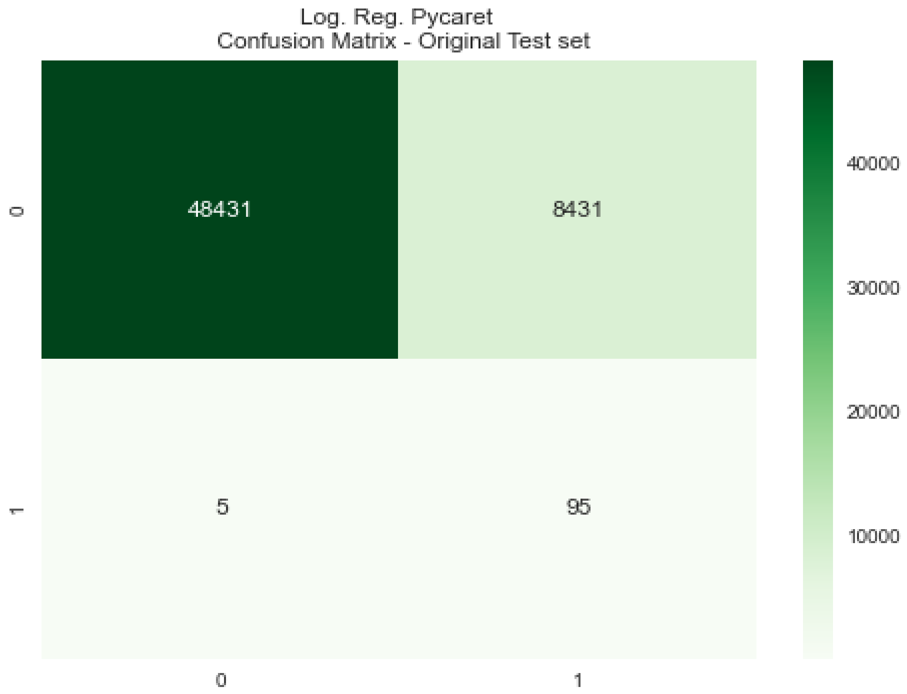
```
# See real target and predicted label
pd.crosstab(predictions['Class'], predictions['Label']).stack().reset_index(name='Freq'
)
```

Out[63]:

	Class	Label	Freq
0	0.0	0.0	48431
1	0.0	1.0	8431
2	1.0	0.0	5
3	1.0	1.0	95

In [64]:

```
# Plot confusion matrix
conf_matx_clf = confusion_matrix(predictions['Class'], predictions['Label'])
ax = plt.axes()
sns.heatmap(conf_matx_clf, annot=True, cmap='Greens', fmt='.0f')
ax.set_title('Log. Reg. Pycaret \n Confusion Matrix - Original Test set')
plt.show()
```



3º APPROACH - CLASSIFIER WITH UNDER & OVER SAMPLES

We'll apply some techniques to help our model identify the outliers.

- First, we'll create a dataframe with 492 non-fraud observations and 492 fraud observations (all fraud available);
- Second, we'll use smote (over-sample) in this dataframe with 984 observations and create our model with these new data (smote after under-sampling);
- After that, we'll predict the under-sampled test set (984 observations);
- Finally, we'll predict the original test set with 56,961 observations.

In [65]:

```
# Shuffle the data before creating the subsamples

df = df.sample(frac=1)

# Fraud classes = 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=1313)

new_df.head()
```

Out[65]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5
151006	-0.293440	0.113606	-26.457745	16.497472	-30.177317	8.904157	-17.892600
248419	2.019842	0.813520	1.939494	-1.228433	-1.822550	-1.123825	-0.153692
154670	1.145812	0.209084	-2.296987	4.064043	-5.957706	4.680008	-2.080938
15566	1.089779	-0.678239	-23.237920	13.487386	-25.188773	6.261733	-17.345188
33295	-0.293440	-0.558230	-0.615139	1.424862	0.825301	0.259156	0.738616

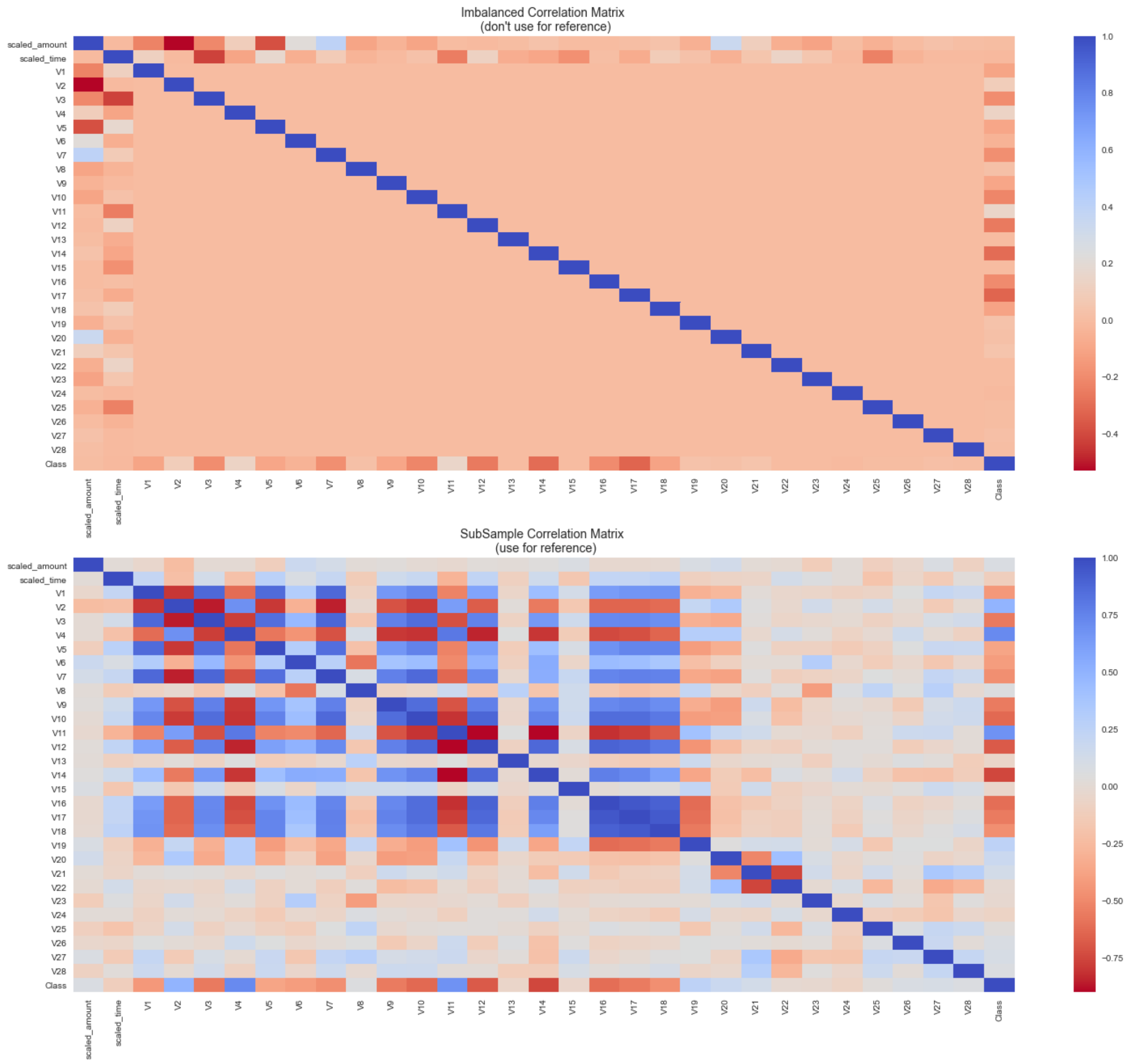
5 rows × 31 columns

In [66]:

```
# Plot the original and the undersampled datasets to see the differences
f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix \n (don't use for reference)", fontsize=14
)

# Subsample dataframe
sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix \n (use for reference)', fontsize=14)
plt.show()
```



We'll remove extreme outliers from features with high correlation with the class.

We'll use the interquartile range method, calculating the interquartile range (IQR) between the 75th percentile and the 25th percentile. This technique aims to define a threshold so if some instance passes it, it will be deleted.

In [67]:

```
# We will analyze only the 3 higher and the 3 lower correlations to remove extreme outl
ies
correlations = sub_sample_corr.drop(sub_sample_corr.columns.difference(['Class']), 1)
correlations = correlations.drop('Class', 0)
correlations = correlations.sort_values(by=['Class'])
```

In [68]:

```
# Lower
lower_3 = correlations[:3]

# Higher
higher_3 = correlations[-3:]
```

We could join the lower and higher correlations to apply only once. But, we'll keep separated so we can see the differences.

In [69]:

```
# Function to remove the extreme correlations
def extremeOutliers(df, correlations, cut):

    global new_df

    for row in correlations.index:
        var = row

        fraud_temp = df[var].loc[df['Class']==1].values
        q25, q75 = np.percentile(fraud_temp, 25), np.percentile(fraud_temp, 75)
        print('Feature: {}'.format(var))
        print('Quartile 25: {} | Quartile 75: {}'.format(round(q25,3), round(q75,3)))
        iqr_temp = q75 - q25
        print('iqr: {}'.format(round(iqr_temp,3)))

        cut_off_temp = iqr_temp * cut
        lower_temp, upper_temp = q25 - cut_off_temp, q75 + cut_off_temp
        print('Cut Off: {}'.format(round(cut_off_temp,3)))
        print('Lower: {}'.format(round(lower_temp,3)))
        print('Upper: {}'.format(round(upper_temp,3)))

        outliers = [x for x in fraud_temp if x < lower_temp or x > upper_temp]
        print('Feature Outliers for Fraud Cases: {}'.format(len(outliers)))
        print('outliers:{}'.format(outliers))

        print('----' * 30)

        df = df.drop(df[(df[var] > upper_temp) | (df[var] < lower_temp)].index)

    new_df = df

    return new_df
```

In [70]:

```
# Removing the lower correlations
extremeOutliers(new_df, lower_3, 1.5)
```

```
Feature: V14
Quartile 25: -9.693 | Quartile 75: -4.283
iqr: 5.41
Cut Off: 8.115
Lower: -17.808
Upper: 3.832
Feature Outliers for Fraud Cases: 4
outliers:[-18.8220867423816, -19.2143254902614, -18.4937733551053, -18.049
997689859396]
---------------------------------------------------------------------------
----------------------------------------------
Feature: V12
Quartile 25: -8.673 | Quartile 75: -2.893
iqr: 5.78
Cut Off: 8.67
Lower: -17.343
Upper: 5.777
Feature Outliers for Fraud Cases: 4
outliers:[-18.4311310279993, -18.553697009645802, -18.047596570821604, -1
8.683714633344298]
---------------------------------------------------------------------------
----------------------------------------------
Feature: V10
Quartile 25: -7.467 | Quartile 75: -2.512
iqr: 4.955
Cut Off: 7.432
Lower: -14.899
Upper: 4.92
Feature Outliers for Fraud Cases: 27
outliers:[-22.1870885620007, -15.1237521803455, -15.2399619587112, -16.649
6281595399, -18.9132433348732, -18.2711681738888, -15.2318333653018, -16.2
556117491401, -16.7460441053944, -15.2399619587112, -22.1870885620007, -2
2.1870885620007, -16.3035376590131, -14.9246547735487, -20.94919155436110
4, -16.6011969664137, -22.1870885620007, -19.836148851696, -15.12416281449
4698, -17.141513641289198, -24.403184969972802, -15.346098846877501, -14.9
246547735487, -15.563791338730098, -23.2282548357516, -24.5882624372475, -
15.563791338730098]
---------------------------------------------------------------------------
----------------------------------------------
```

Out[70]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5
248419	2.019842	0.813520	1.939494	-1.228433	-1.822550	-1.123825	-0.153692
154670	1.145812	0.209084	-2.296987	4.064043	-5.957706	4.680008	-2.080938
15566	1.089779	-0.678239	-23.237920	13.487386	-25.188773	6.261733	-17.345188
33295	-0.293440	-0.558230	-0.615139	1.424862	0.825301	0.259156	0.738616
239501	3.007895	0.768888	-6.682832	-2.714268	-5.774530	1.449792	-0.661836
...	...	...	...	...	...	...	...
117044	-0.269825	-0.119492	-0.641963	1.221503	1.285376	-0.024410	-0.160326
224407	-0.097813	0.694146	2.345023	-1.421737	-1.447076	-1.896845	-0.648271
8312	-0.293440	-0.864672	0.378275	3.914797	-5.726872	6.094141	1.698875
30442	-0.243695	-0.572916	-3.896583	4.518355	-4.454027	5.547453	-4.121459
279080	0.025152	0.986172	-1.138516	-0.762543	-1.142460	0.022805	3.478549

946 rows × 31 columns

In [71]:

```
# Removing the higher correlations
extremeOutliers(new_df, higher_3, 1.5)
```

```
Feature: V2
Quartile 25: 1.133 | Quartile 75: 4.142
iqr: 3.009
Cut Off: 4.513
Lower: -3.38
Upper: 8.655
Feature Outliers for Fraud Cases: 46
outliers:[13.4873857909274, 9.067613427317669, 15.3658043803315, -3.952320
08590575, 12.785970638297998, 12.785970638297998, 12.785970638297998, 12.9
305051249875, 15.876922987953598, 10.5417508026636, -3.93073139597263, 10.
114815724665402, 11.817921989785301, 12.6521968313004, -3.4204679837707, 1
0.5586001882538, -5.1983601992332895, 8.7759971528627, 10.8196653713117, -
3.93591892431521, 16.1557014298057, 13.7659421584186, 16.4345245512223, 1
5.598192662555402, -6.976420007546411, -8.402153677689151, -7.159041717094
45, 16.7133892350242, 14.6019980426299, 12.095893225929899, 9.669900173040
97, 14.323253809723301, -7.4490151587267395, 12.3739891389716, 10.39391714
27504, -4.8144607395562105, 12.785970638297998, 14.044566781510598, -7.196
97963053735, 12.785970638297998, 12.785970638297998, 13.208904284417601, 1
1.586380519818402, 8.71325017095966, 9.22369194937548, -3.488130181185609
6]
----------------------------------------------------------------------------
-----------------------------------------------
Feature: V11
Quartile 25: 1.845 | Quartile 75: 4.775
iqr: 2.93
Cut Off: 4.396
Lower: -2.55
Upper: 9.171
Feature Outliers for Fraud Cases: 11
outliers:[9.36907905765884, 9.939819741725689, 10.5452629545898, 10.853011
6481991, 10.187587324166401, 10.2777688628065, 9.32879925655782, 10.063789
7462894, 10.446846814514, 11.277920727806698, 11.152490598583698]
----------------------------------------------------------------------------
-----------------------------------------------
Feature: V4
Quartile 25: 2.084 | Quartile 75: 5.506
iqr: 3.422
Cut Off: 5.133
Lower: -3.048
Upper: 10.639
Feature Outliers for Fraud Cases: 2
outliers:[10.6485054461688, 10.6485054461688]
----------------------------------------------------------------------------
-----------------------------------------------
```

Out[71]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5
248419	2.019842	0.813520	1.939494	-1.228433	-1.822550	-1.123825	-0.153692
154670	1.145812	0.209084	-2.296987	4.064043	-5.957706	4.680008	-2.080938
33295	-0.293440	-0.558230	-0.615139	1.424862	0.825301	0.259156	0.738616
239501	3.007895	0.768888	-6.682832	-2.714268	-5.774530	1.449792	-0.661836
191690	-0.307413	0.524900	1.183931	3.057250	-6.161997	5.543972	1.617041
...	...	...	...	...	...	...	...
117044	-0.269825	-0.119492	-0.641963	1.221503	1.285376	-0.024410	-0.160326
224407	-0.097813	0.694146	2.345023	-1.421737	-1.447076	-1.896845	-0.648271
8312	-0.293440	-0.864672	0.378275	3.914797	-5.726872	6.094141	1.698875
30442	-0.243695	-0.572916	-3.896583	4.518355	-4.454027	5.547453	-4.121459
279080	0.025152	0.986172	-1.138516	-0.762543	-1.142460	0.022805	3.478549

872 rows × 31 columns

In [72]:

```
# Separating X and y
X = new_df.drop('Class', axis=1)
y = new_df['Class']

# Spliting train and test with the undersampled dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
313)
```

In [73]:

```
# SMOTE Technique (OverSampling)
sm = SMOTE(sampling_strategy='minority', random_state=1313)
```

In [74]:

```
# This is the data we will train the model
Xsm_train, ysm_train = sm.fit_sample(original_Xtrain, original_ytrain)
```

In [75]:

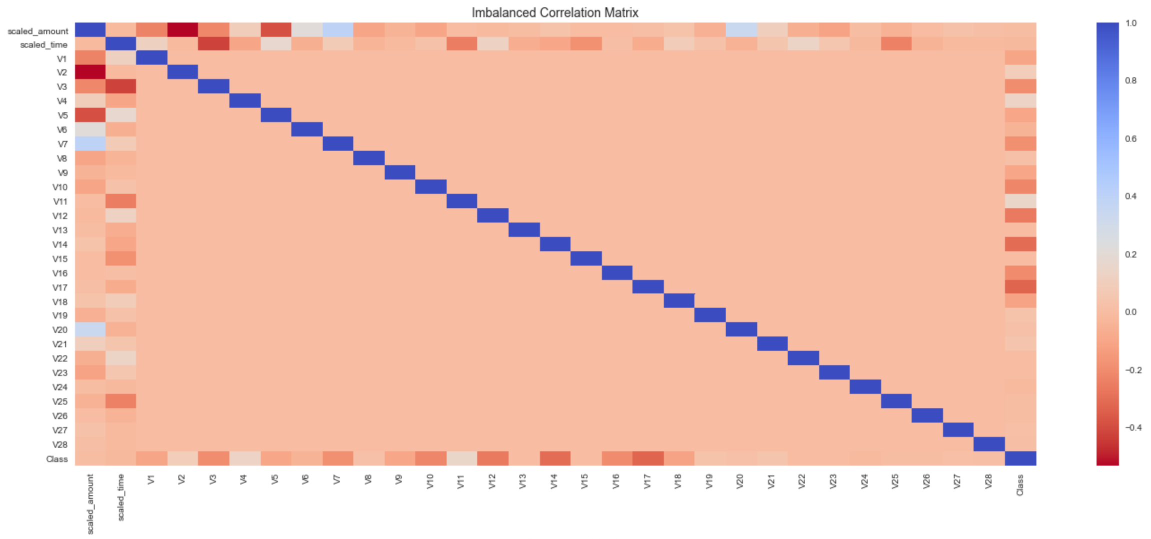
```
# Join X and y in the smote dataset
df_smote = pd.merge(Xsm_train, ysm_train, left_index=True, right_index=True)
```

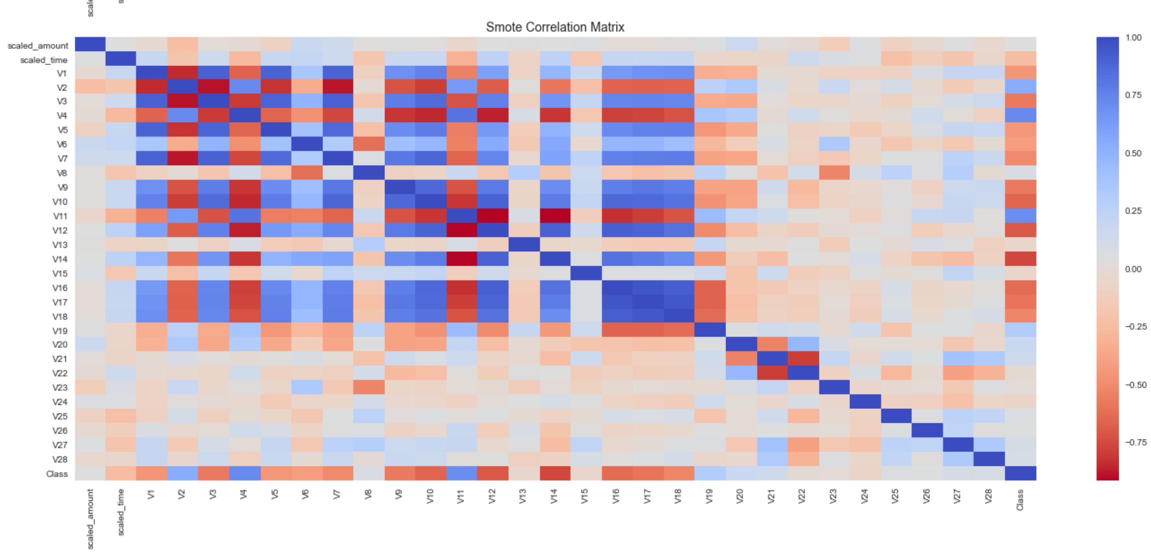
In [76]:

```
# Plot the original and the smote datasets to see the differences
f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix", fontsize=14)

# Smote dataFrame
sub_sample_corr = df_smote.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('Smote Correlation Matrix', fontsize=14)
plt.show()
```





In [77]:

```
# We will analyze only the 3 higher and the 3 lower correlations to remove extreme outl
ies in the smote dataset
correlations = sub_sample_corr.drop(sub_sample_corr.columns.difference(['Class']), 1)
correlations = correlations.drop('Class', 0)
correlations = correlations.sort_values(by=['Class'])
```

In [78]:

```
# Lower (smote)
lower_3 = correlations[:3]

# Higher (smote)
higher_3 = correlations[-3:]
```

In [79]:

```
# Removing the lower correlations (smote)
extremeOutliers(df_smote, lower_3, 3)
```

```
Feature: V14
Quartile 25: -9.863 | Quartile 75: -4.542
iqr: 5.321
Cut Off: 15.963
Lower: -25.827
Upper: 11.421
Feature Outliers for Fraud Cases: 0
outliers:[]
--------------------------------------------------------------------------
----------------------------------------------
Feature: V12
Quartile 25: -9.809 | Quartile 75: -3.294
iqr: 6.515
Cut Off: 19.546
Lower: -29.355
Upper: 16.252
Feature Outliers for Fraud Cases: 0
outliers:[]
--------------------------------------------------------------------------
----------------------------------------------
Feature: V10
Quartile 25: -8.512 | Quartile 75: -2.91
iqr: 5.601
Cut Off: 16.804
Lower: -25.316
Upper: 13.894
Feature Outliers for Fraud Cases: 0
outliers:[]
--------------------------------------------------------------------------
----------------------------------------------
```

Out[79]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193
...	...	...	...	...	...	...	...
454899	0.671712	0.108302	-12.307217	7.602335	-19.788195	7.189385	-11.061654
454900	-0.292028	-0.415486	0.624241	0.390222	0.974385	-0.045120	-0.225802
454901	-0.293440	-0.847775	-2.679379	7.022383	-13.548242	10.223529	-2.893169
454902	4.458440	-0.075704	-1.550026	-0.996501	0.810719	0.259955	-0.195255
454903	4.097408	0.107626	-5.840251	7.151384	-12.817413	7.034110	-9.645873

454900 rows × 31 columns

In [80]:

```
# Removing the higher correlations (smote)
extremeOutliers(df_smote, higher_3, 3)
```

Feature: V2
Quartile 25: 1.328 | Quartile 75: 6.186
iqr: 4.858
Cut Off: 14.574
Lower: -13.245
Upper: 20.76
Feature Outliers for Fraud Cases: 522
outliers:[21.4672029942752, 22.0577289904909, 21.58361433063832, 21.988850
878626053, 21.477239748906246, 21.905512688746356, 21.793438500680477, 21.
582099744560637, 21.730665633383843, 21.20372471555416, 21.81629433060585
7, 21.491680761406492, 21.84003188924353, 20.97386125190294, 22.0527198291
16526, 21.347634785251575, 21.618596524417452, 21.546770870493475, 20.9933
8684963587, 21.415749552492965, 21.560575584225305, 21.060962005469456, 2
1.731172015153852, 21.938702340551508, 21.96480314439898, 21.5327973206481
22, 21.357857833850346, 21.58089093915473, 21.185296861703765, 20.99491021
429548, 21.04782975130339, 21.70657221371062, 21.030098493071225, 21.74246
242961703, 21.32169322403962, 21.57014748179804, 21.747112748864108, 21.71
411424457876, 21.625782050534113, 20.764413052178707, 21.383524644229098,
21.582375026145396, 21.070782553898688, 21.941165162874057, 21.94834472882
034, 20.804657300623724, 21.641470856125643, 21.47411027918042, 21.990029
15916604, 22.00137706675789, 21.134661052776657, 21.558793751342844, 20.81
9378647748508, 21.557568789248478, 21.972179856267, 20.922019472333538, 2
1.809272649045834, 22.03160836895243, 21.726662663964174, 20.9762093758136
74, 21.876377063839406, 21.991697016815483, 21.710731645482095, 21.8526797
38274155, 21.259282152509897, 21.35684735048615, 21.730564480547216, 20.93
56824190607, 21.963788485416877, 21.7952179064955, 21.14096540507757, 21.5
59563111279676, 21.682020715908855, 21.564384928564376, 21.13336860007916,
21.866825249707684, 21.67396075022436, 21.645977751303754, 21.051518524451
666, 21.855055995541885, 21.755992725483924, 20.973201571464394, 21.041823
178860458, 21.111791262112884, 21.540824785062366, 20.79414868381397, 21.9
57653548354752, 21.676549008649697, 21.025933172283743, 21.59689622556507,
21.36398702163707, 21.460019824286753, 21.880529657424724, 21.959183708723
245, 21.50034489222224, 21.59604091830481, 21.53191267687703, 21.869103819
76007, 21.620483001138787, 21.79901635305907, 20.777639736932294, 21.83451
2893194347, 21.612536233754735, 21.160603321613635, 21.930429211478287, 2
0.89021770289382, 21.181704837569818, 21.849958429428295, 21.4096145133892
9, 20.780954132172308, 21.795135785813972, 21.68251528704212, 21.444349371
231766, 21.74394680692352, 21.774765089825394, 21.82004927881603, 21.21232
8448091334, 21.50213506641068, 21.669719333387036, 20.97997141979147, 22.0
3219398779399, 21.24127641876442, 21.745055753453187, 21.499103455593637,
21.899602412179238, 21.877804074503842, 21.717837454625812, 21.39602022509
393, 21.480790887637628, 21.025320516519642, 20.89301349405206, 20.840001
54175813, 22.042579563692062, 21.4211639800276, 21.961460568150304, 20.937
11022737927, 21.256442920262458, 21.802983430548437, 21.730811518972907, 2
0.862698778171076, 21.863268294372126, 21.125264319633644, 21.972027880831
604, 21.96846723200345, 21.87165323573523, 21.42266550151578, 20.891541846
858942, 20.991956784078432, 21.94254166741445, 21.55104626728692, 21.33270
367600723, 21.481604568068153, 21.883848274616494, 20.78014117670568, 22.0
0242627090853, 21.178922695812048, 21.685956030155918, 21.407371714614516,
21.89888646341627, 21.526096170876958, 21.72341090202639, 20.9977933648423
73, 20.857174623517647, 20.97418500885202, 21.88331842355969, 21.313689420
448423, 21.717662542321808, 21.954988550983654, 21.50124901693135, 20.9493
1781822974, 20.94422471609679, 21.955439971273915, 21.681767282398066, 21.
01727386737092, 21.95288939254612, 21.573368431873266, 21.60777562770699,
21.6609645699247, 22.051852751515018, 21.228151188729452, 21.5017673187011
8, 22.03139323936807, 21.397525064185377, 21.904016388149557, 21.611851872
78068, 21.99919283204061, 21.98303971723646, 21.99960580186763, 21.4741941
97272144, 20.94426815358466, 21.00824167102818, 20.98653425051201, 21.2108
48127513078, 21.2174034254758, 21.788339664700764, 22.008195492451158, 20.
779992466672358, 21.05237796650242, 21.945561552666653, 21.908724349877, 2
1.498328044056375, 21.110633190749734, 20.907483225071406, 21.871067666375

204, 21.945376348897835, 21.126663010298035, 21.474195658545376, 21.465888
40915013, 21.396926361294817, 21.652026614346365, 21.85967374625822, 21.81
5177180095702, 20.783981413199697, 21.498053161352075, 21.634762603391966,
21.24875882642824, 21.34420487389161, 21.918652332005205, 21.7291442461478
2, 21.481203127928243, 20.929857918462822, 22.051866393044605, 21.88586581
6084827, 21.5582319988035, 21.069513184557543, 21.412937793785563, 21.566
32857456761, 21.837305073071082, 20.9783360918774, 21.033574476295964, 22.
054472604090552, 21.892892657582337, 20.980802633522284, 21.65587324915016
5, 21.409499614188412, 21.9273376287832, 21.358173874996417, 21.5085382224
3187, 21.747209297656426, 21.44681645688882, 21.655468928852667, 21.930530
785818664, 21.883396391794992, 20.858029356411727, 21.9889238156589, 20.85
6838285187397, 21.899531161926944, 21.026217320005998, 20.866651886439698,
21.641946433826572, 21.907913202385043, 22.01787608349703, 21.699049837739
85, 21.82464667989276, 21.382251400301477, 21.797443238657294, 20.86351484
525713, 21.73029021790195, 21.036790743953546, 20.987711363624832, 21.8671
86577274534, 21.693534329901826, 20.940121875811265, 21.97405394006389, 2
1.622782585529684, 21.30977264107701, 21.85636219372181, 21.73160610201802
4, 21.07720891753697, 20.821903642948218, 22.006667725044515, 21.670729609
44248, 20.869609498608288, 22.037899566422517, 20.772323967769506, 21.6565
2778301642, 21.006544336829773, 20.788241702736666, 20.821248005465808, 2
1.985454709899006, 21.125019865937354, 21.88150757460992, 21.2478239216367
34, 21.916562542205934, 21.6749604223129, 21.99442760821961, 21.9853353725
40288, 21.37185875459047, 21.27480848612921, 21.7821019969087, 21.8999046
45131397, 21.362921982557204, 21.618409264998622, 21.845695090060975, 21.4
497757444193, 20.7786851095046, 21.82329375354908, 21.867990729595878, 21.
63350251701724, 21.347324478289487, 21.700786892661707, 21.40124075564080
6, 21.31349911526344, 21.942519109793395, 21.45451166197378, 21.5075170797
60106, 21.420381940880844, 21.218338870340048, 21.975572058416965, 21.8933
20351558472, 21.79779086743855, 21.653234965821767, 21.25468562478617, 21.
104195964323495, 20.82152990277572, 21.929687564132312, 21.81652351467491
3, 20.90735987568377, 21.835927996580917, 21.46502377829335, 20.7790862418
76627, 21.36277648130581, 20.878895649937917, 21.800866548767637, 21.80160
283045922, 21.79717785677284, 21.980571722047966, 21.868474504478506, 21.5
7843304664191, 21.82570749294608, 21.76911407040542, 21.745346535010942, 2
1.983338525571675, 21.042754663767887, 21.574954272574587, 20.947071656749
44, 21.750174195510088, 21.796943170236563, 21.629233902821333, 20.9100602
13579726, 21.83673770653361, 20.801673402382526, 21.67878552054515, 21.927
329485074825, 21.620631136129656, 21.865205960177423, 21.636634086712572,
21.641051550345157, 20.949111466281575, 21.866899907752074, 21.54923041242
7384, 20.849915165836066, 21.96799058417775, 21.09061712537259, 21.3454960
03154633, 21.057712242627403, 21.712051184963467, 21.051452223505215, 20.8
71889289206152, 21.86703900488594, 21.75101899698401, 21.633436176038995,
22.024809832489797, 20.809688862065407, 21.692173417339948, 20.82414092926
3283, 21.77653288968035, 21.891073019290666, 21.90214019066017, 21.1214124
52262085, 22.003918342883164, 22.049113323526942, 21.28621224777671, 21.09
675877262672, 21.47310048911429, 21.702493190302384, 20.85066729308452, 2
2.010108610165275, 21.88355760198862, 21.981059342324908, 21.4738354936271
8, 21.157315290048157, 21.81919980404386, 21.737977882524426, 21.315059921
59324, 21.289946517204346, 20.871853024965304, 21.55551032916131, 20.90456
7943439165, 21.94453566002533, 20.981716349746904, 21.825545240464646, 21.
406032045485546, 22.01415260390231, 21.111409677423165, 21.03538419333652
6, 21.7760912211518, 21.993636773409907, 20.8177964963898, 21.883092221143
603, 21.99194409276532, 21.359073666963106, 21.507899842274718, 21.3190499
19463602, 22.02721873488286, 21.97672714742287, 21.702912841970395, 21.879
589356713687, 21.23473188739938, 21.685356422778383, 22.004335479977005, 2
1.48830063159286, 21.10803511104165, 21.344679802642577, 21.67663286862078
2, 22.003837604766407, 20.846765734826697, 21.72259100038032, 21.997013801
415847, 21.405186673671814, 22.007676784554366, 20.795151202074422, 21.570
479343390296, 21.361868817655346, 21.42106442291969, 22.00534002070864, 2
1.937225024383604, 21.549166777067867, 21.646478199855757, 21.777577045680
91, 21.498086847318145, 21.731425764268806, 21.590552013972502, 21.7731479

```
311821, 21.29399845886333, 21.446840664448136, 21.810909206778803, 21.5825
299019122, 21.903320204735184, 21.982792709078296, 21.662426045897657, 21.
662671925001906, 21.301741170751388, 21.738842055248302, 21.23994129833010
5, 21.299731381119667, 21.193513610726267, 21.746984986778408, 22.03426198
0112923, 21.04601310513233, 21.80559458296019, 21.42300676025127, 21.83641
4121403045, 21.61505562159961, 21.25206120542931, 21.794839221779892, 21.5
46272848283845, 22.008639125435128, 21.788667085801876, 21.73366657114188
5, 20.801560074677564, 21.250698585941947, 20.800469843540682, 21.49779023
2421693, 21.770856671160544, 21.084461198355473, 21.158843014566845, 21.52
4493667478534, 21.91688208454768, 21.749007011241577, 21.470552261945112,
21.63343300824308, 20.994026576452793, 20.79439803645135, 21.7208464044420
03, 21.31828337893549, 21.85197489648874, 21.87113535148518, 20.9776992061
70783, 21.660371420381907, 21.98572538862507, 21.890018807183836, 21.88245
9328987284, 21.731587860668167, 21.85424346056888, 21.170515972190156, 21.
445235563796476, 21.355827089001124, 21.239422075258254, 21.5845174737033
5, 21.541123276007927, 21.60533055614015, 21.80064456256733, 21.8140294689
58974, 20.887772563698054, 21.186712934576605, 20.922009365261523, 21.6286
58420897857, 21.637674509947956, 21.873625958623727, 21.57665682128328, 2
1.87334497644559, 22.015123385104914, 22.03969265033535, 22.02654993129772
3, 21.688729743699746, 20.82661136114901, 21.348707280269984, 21.766863658
282208, 21.768776511119732, 21.708069012285115, 21.686876559805683, 21.926
204780539983, 21.033689883459612, 21.713192851022832, 21.886318394867043,
20.79932322147618, 21.935859675431818, 20.937723381368176, 21.568093971922
3, 20.825674741373383, 20.806198884220677]
---------------------------------------------------------------------------
---------------------------------------------
Feature: V11
Quartile 25: 2.365 | Quartile 75: 5.809
iqr: 3.444
Cut Off: 10.332
Lower: -7.967
Upper: 16.141
Feature Outliers for Fraud Cases: 0
outliers:[]
---------------------------------------------------------------------------
---------------------------------------------
Feature: V4
Quartile 25: 2.395 | Quartile 75: 6.423
iqr: 4.028
Cut Off: 12.084
Lower: -9.688
Upper: 18.507
Feature Outliers for Fraud Cases: 0
outliers:[]
---------------------------------------------------------------------------
---------------------------------------------
```

Out[80]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193
...	...	...	...	...	...	...	...
454899	0.671712	0.108302	-12.307217	7.602335	-19.788195	7.189385	-11.061654
454900	-0.292028	-0.415486	0.624241	0.390222	0.974385	-0.045120	-0.225802
454901	-0.293440	-0.847775	-2.679379	7.022383	-13.548242	10.223529	-2.893169
454902	4.458440	-0.075704	-1.550026	-0.996501	0.810719	0.259955	-0.195255
454903	4.097408	0.107626	-5.840251	7.151384	-12.817413	7.034110	-9.645873

454068 rows × 31 columns

In [81]:

```
# Separating X and y (smote)
X = df_smote.drop('Class', axis=1)
y = df_smote['Class']

# Spliting train and test (smote dataframe).
Xsm_train, X1_test, ysm_train, y1_test = train_test_split(X, y, test_size=0.2, random_s
tate=1313)
```

In [82]:

```
# Logistic Regression parameters
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
```

In [83]:

```
# Logistic Regression fitted in smote dataframe
t0 = time.time()
log_reg_sm = grid_log_reg.estimator
log_reg_sm.fit(Xsm_train, ysm_train)
t1 = time.time()
print("Fitting SMOTE data took :{} sec".format(t1 - t0))
```

```
Fitting SMOTE data took :6.92450475692749 sec
```

In [84]:

```
# Logistic Regression predicted in the under-sampled dataframe
y_pred_log_reg = log_reg_sm.predict(X_test)
```

In [85]:

```
# Report (comparing with under-sampled class)
print('Logistic Regression:')
print(classification_report(y_test, y_pred_log_reg))
```

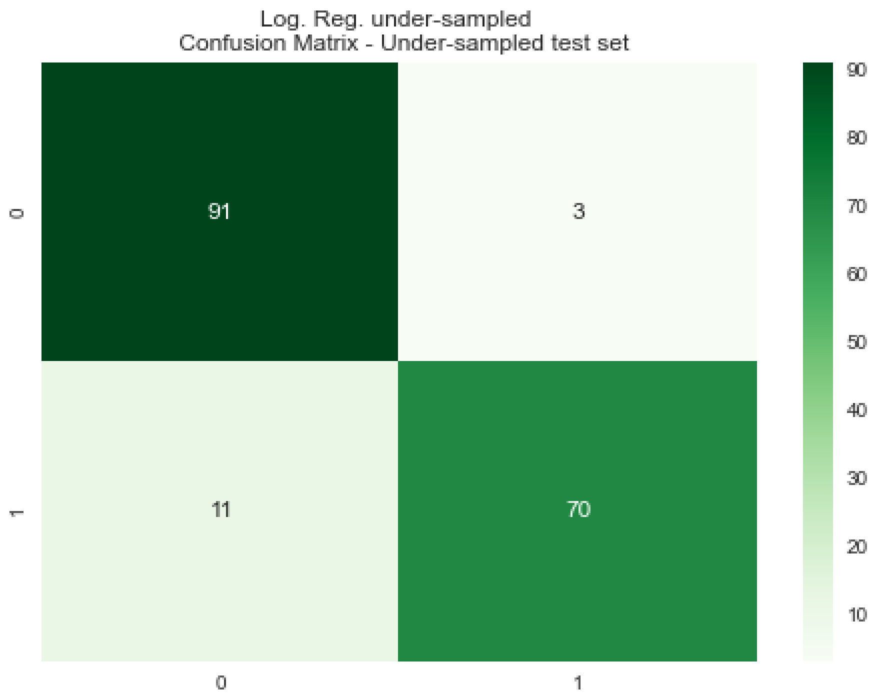
```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        94
           1       0.96      0.86      0.91        81

    accuracy                           0.92       175
   macro avg       0.93      0.92      0.92       175
weighted avg       0.92      0.92      0.92       175
```

In [86]:

```
# Plot confusion matrix
conf_matx_smt = confusion_matrix(y_test, y_pred_log_reg)
ax = plt.axes()
sns.heatmap(conf_matx_smt, annot=True, cmap='Greens', fmt='.0f')
ax.set_title('Log. Reg. under-sampled \n Confusion Matrix - Under-sampled test set')
plt.show()
```



In [87]:

```
# Logistic Regression predicted in the original test set
y_pred_sm = log_reg_sm.predict(original_Xtest)
```

In [88]:

```
# Report
labels = ['No Fraud', 'Fraud']
print(classification_report(original_ytest, y_pred_sm, target_names=labels))
```

```
              precision    recall  f1-score   support

    No Fraud       1.00      0.99      0.99     56863
       Fraud       0.12      0.87      0.21        98

    accuracy                           0.99     56961
   macro avg       0.56      0.93      0.60     56961
weighted avg       1.00      0.99      0.99     56961
```

In [89]:

```
# Merge labels with X_test
original_Xtest['Label'] = y_pred_sm
original_Xtest['Class'] = original_ytest
```

In [90]:

```
# Metrics
acc = accuracy_score(original_Xtest['Class'], original_Xtest['Label'])
auc = roc_auc_score(original_Xtest['Class'], original_Xtest['Label'])
recall = recall_score(original_Xtest['Class'], original_Xtest['Label'])
precision = precision_score(original_Xtest['Class'], original_Xtest['Label'])
f1 = f1_score(original_Xtest['Class'], original_Xtest['Label'])
```

In [91]:

```
# Visualize metrics
cols = ['Model','Accurary','AUC','Recall','Prec.','F1']
values = ['Log. Reg. Smote',acc,auc,recall,precision,f1]
metrics_df = pd.DataFrame({tup[0]: [tup[1]] for tup in zip(cols, values)})
metrics_df
```

Out[91]:

	Model	Accurary	AUC	Recall	Prec.	F1
0	Log. Reg. Smote	0.988905	0.928231	0.867347	0.120739	0.21197

In [92]:

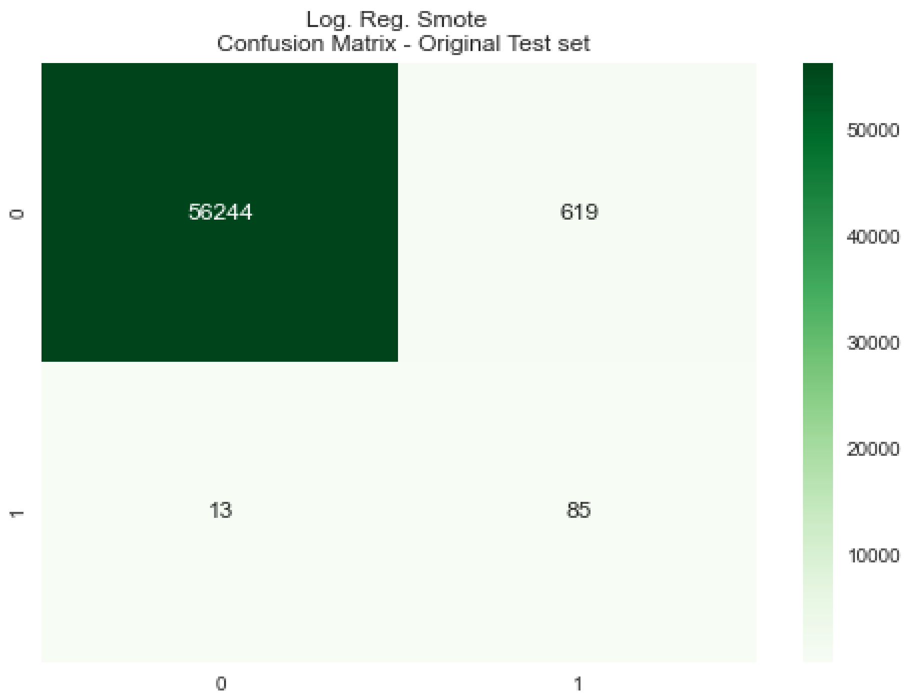
```
# See real target and predicted label
pd.crosstab(original_Xtest['Class'], original_Xtest['Label']).stack().reset_index(name=
'Freq')
```

Out[92]:

	Class	Label	Freq
0	0	0	56244
1	0	1	619
2	1	0	13
3	1	1	85

In [93]:

```
# Plot confusion matrix
conf_matx_smt = confusion_matrix(original_ytest, y_pred_sm)
ax = plt.axes()
sns.heatmap(conf_matx_smt, annot=True, cmap='Greens', fmt='.0f')
ax.set_title('Log. Reg. Smote \n Confusion Matrix - Original Test set')
plt.show()
```



CONCLUSION

The implementations made in this project show us that we have a trade-off when it comes to models. On the one hand, a model that can detect 95% of anomaly cases, but which attributes a huge volume of "non-fraud" transactions to fraud (false positive). On the other hand, a little prediction is lost using an under-over sample model (we achieved 87% success in detecting fraud), but we labeled 92% less reputable transactions as fraud, thus reducing our volume of false positives.

Some points about the models:

- The model with under-over sample was the best. There were only 619 false-positive and 13 false-negative observations;
- The pycaret classification model was better at solving frauds, however, there were 8431 cases of false positives, which can generate dissatisfaction among customers for having their cards blocked due to suspected fraud;
- The pycaret anomaly model managed to detect fraud as well as the under-over sample model, but it also had a high number of false positives (2657).

Important to say: Pycaret is a powerful prediction tool because with few lines of code it is possible to develop a model with great metrics. For models where there are no metrics designed and/or that need a quick solution, albeit a palliative one, pycaret will surely exceed expectations.

Personal note: I have been using pycaret in work projects. A scenario where we have few resources and a lot of demand. We are gaining agility with projects that need a quick response (due to our industry) or new customers, where there are still no defined metrics/goals.

Here is a summary of the 3 confusion matrices applied on the same test set:

In [98]:

```
# Plot confusion matrix for the 3 approaches. Everyone built with the original test set

f, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(20,5))

# 1º Approach (Anomaly model with Pycaret)
sns.heatmap(conf_matx_anom, ax=ax1, annot=True, cmap='Greens', fmt='.0f')
ax1.set_title('Clustering-Based Local Outlier "CLT" Pycaret \n Confusion Matrix - Original Test set', fontsize=14)

# 2º Approach (Classifier model with Pycaret)
sns.heatmap(conf_matx_clf, ax=ax2, annot=True, cmap='Greens', fmt='.0f')
ax2.set_title('Log. Reg. Pycaret \n Confusion Matrix - Original Test set', fontsize=14)

# 3º Approach (Classifier with under & over sample)
sns.heatmap(conf_matx_smt, ax=ax3, annot=True, cmap='Greens', fmt='.0f')
ax3.set_title('Log. Reg. Smote \n Confusion Matrix - Original Test set', fontsize=14)
```

Out[98]:

```
Text(0.5, 1.0, 'Log. Reg. Smote \n Confusion Matrix - Original Test set')
```