Data Analysis and Machine-Learning

Chapter 10.3.

ML Modelling Applications (2): Coping with Data Imbalances

Box Plot Analysis, Tomeklink, CNN, ENN, SMOTE,

Undersampling and Oversampling Methods



*There is always a way to go,*

*If you look for it.*

*- Ernest A. Fitzgerald*

1. Introduction

In most cases, the real-world raw data provided to data scientists are often unstructured and disorganized, which adds up to the difficulties in processing big data with efficiency. Data imbalance is one example of such kind, which often becomes a critical obstacle in formulating an accurate model. Fortunately, however, there are several ways to cope with data imbalances. One way is to utilize statistical visualization such as box plots, in order to respectively examine the specific structure and dispersion of each category in an imbalanced variable. Another way is to eradicate imbalance per se, by either increasing or decreasing the total volume of data (or specific categories in such data), i.e., oversampling method (SMOTE), undersampling method (Tomeklink, CNN, ENN), and combined method (SMOTE+EEN, SMOTE+Tomeklink, etc.). This subchapter illustrates both methods, with applications on a real-world dataset with data imbalance. More specifically, we will be utilizing the credit card transaction dataset, where obviously the number of normal transactions is significantly larger than the number of fraud transactions, causing data imbalance problems in machine-learning modelling procedures.

2. EDA (Exploratory Data Analysis) and Visualizations of Variables

#Import Essential Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sys

sys.path.append('C:\\Users\\Master\\Desktop\\dataTools')

import dataTools as dt

from IPython.display import display

pd.options.display.max\_columns = 10000

pd.options.display.max\_colwidth = 1000

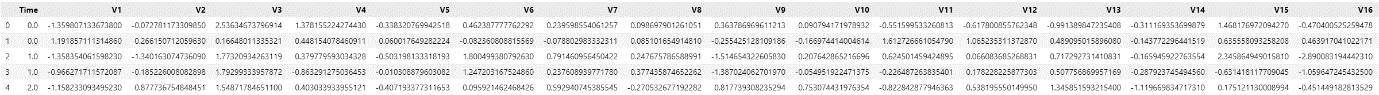
pd.options.display.max\_rows = 10000

pd.options.display.precision = 15

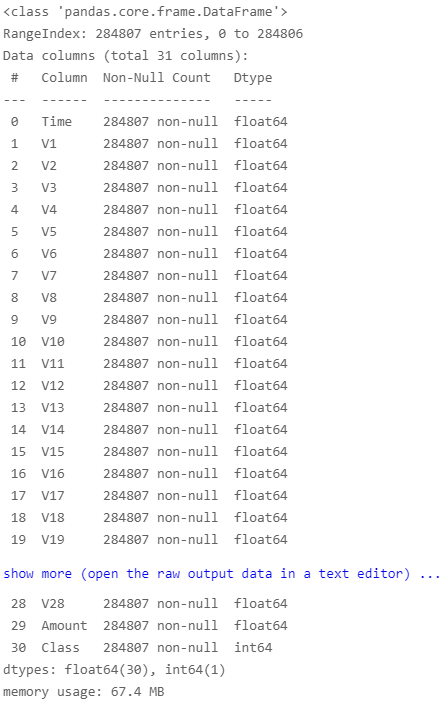
#Import Dataset

rawData = pd.read\_csv('Your File Path\\creditcard.csv')

rawData.head()



rawData.info()



rawData['Class'].unique()

OUTPUT:

array([0, 1], dtype=int64)

rawData['Class'].value\_counts()

OUTPUT:

0 284315

1 492

Name: Class, dtype: int64

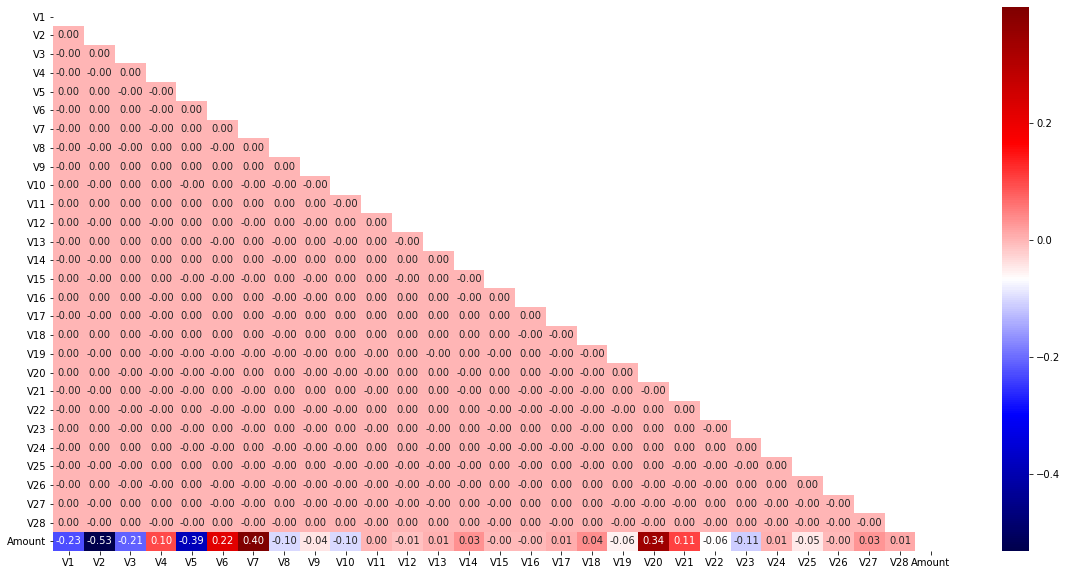
As the value\_count output depicts, the target variable is significantly imbalanced, with most of the data leant towards 0 (normal transaction) and less towards 1 (fraud). This would be a problem, as our purpose would be to generate a model that detects and predicts fraud transactions.

plt.figure(figsize=(20,2))

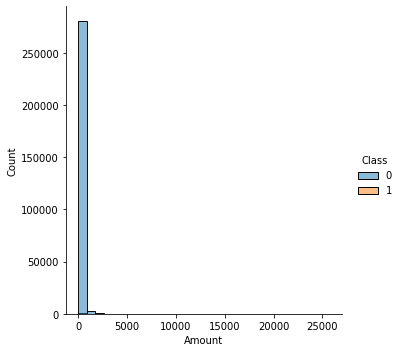
sns.countplot(data=rawData, y='Class')



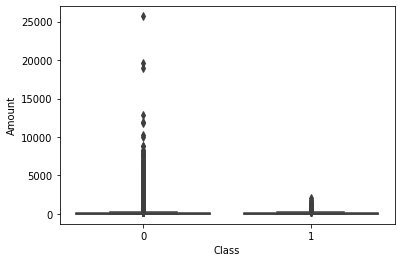
dt.visualCorr(rawData.drop(columns=['Time','Class']), width=20, height=10)



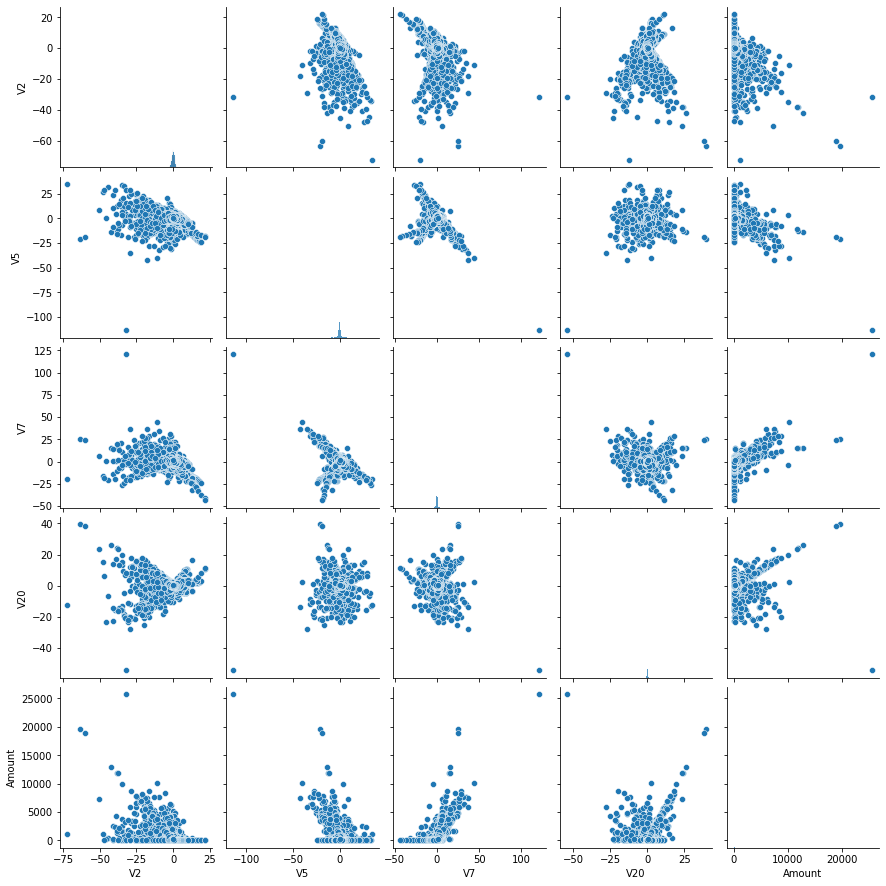
sns.displot(data=rawData, x='Amount', bins=30, hue='Class')



sns.boxplot(data=rawData, x='Class', y='Amount')



sns.pairplot(rawData[['V2', 'V5', 'V7', 'V20', 'Amount']])



sns.pairplot(rawData[

   [

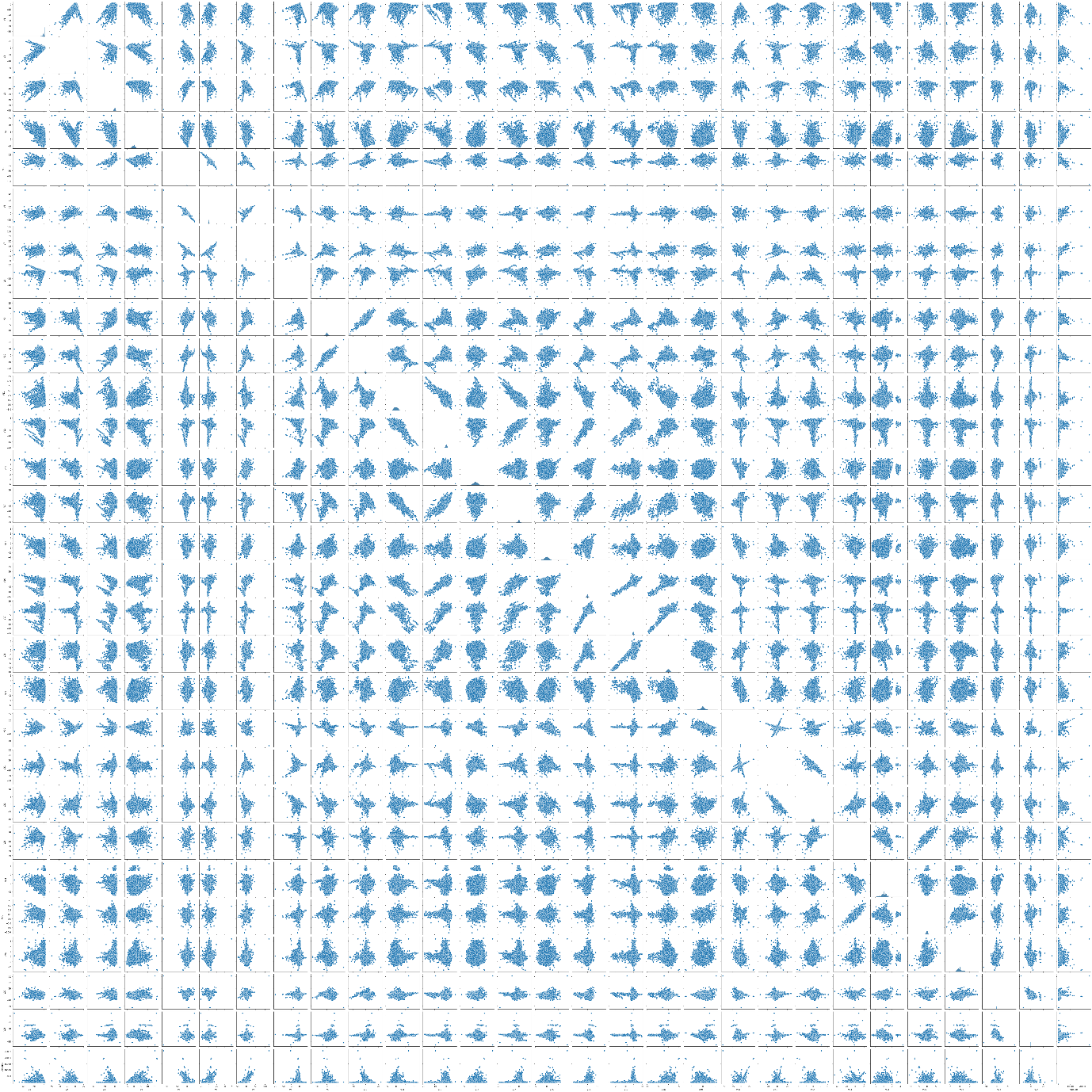
     'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',

     'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',

     'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',

   ]

])



2.1. Utilizing Box Plots

#Utilizing Box Plot (1): Detecting Possibly Significant Variables

col\_name = 'V3'

fig, ax = plt.subplots(10,2,figsize=(10,20))

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[0,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[0,1])

col\_name = 'V4'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[1,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[1,1])

col\_name = 'V9'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[2,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[2,1])

col\_name = 'V10'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[3,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[3,1])

col\_name = 'V11'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[4,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[4,1])

col\_name = 'V12'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[5,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[5,1])

col\_name = 'V14'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[6,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[6,1])

col\_name = 'V16'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[7,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[7,1])

col\_name = 'V17'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[8,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[8,1])

col\_name = 'V18'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[9,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[9,1])

plt.subplots\_adjust(left=0.1,

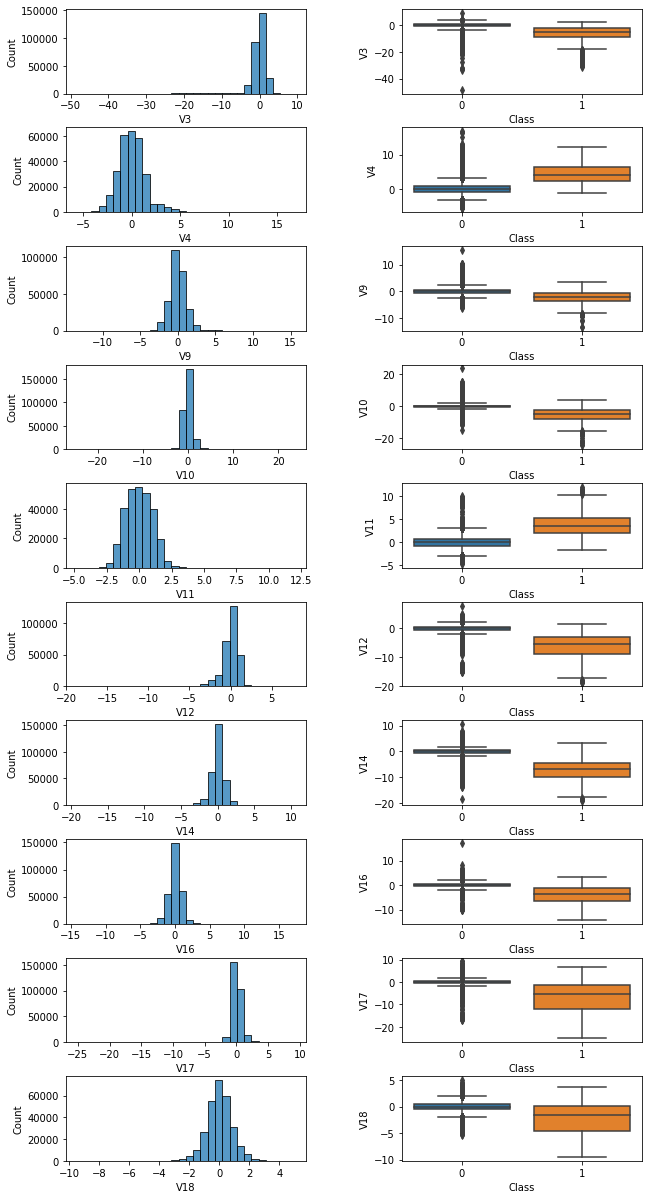
                    bottom=0.1,

                    right=0.9,

                    top=0.9,

                    wspace=0.4,

                    hspace=0.4)



Using box plots as above, we can draw a rational hypothesis that the variables V3, V4, V9,...,V18 may be significant in terms of their effects on the target variable (normal vs. fraud transactions). For instance, the respective distributions of data values in V14 differ distinctly in relation to the categories 0 and 1. In a similar manner, we can hypothesize that the variables V1, V2, V5, V7, V19 are possibly significant as well (yet with lesser significance compared to the variables above), and the other remaining variables (V6, V8, V13, V15, V20~V28) possibly have no or significantly less effects on the target variable.

#Utilizing Box Plot (2): Detecting Possibly Less Significant Variables

col\_name = 'V1'

fig, ax = plt.subplots(5,2,figsize=(10,20))

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[0,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[0,1])

col\_name = 'V2'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[1,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[1,1])

col\_name = 'V5'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[2,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[2,1])

col\_name = 'V7'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[3,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[3,1])

col\_name = 'V19'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[4,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[4,1])

plt.subplots\_adjust(left=0.1,

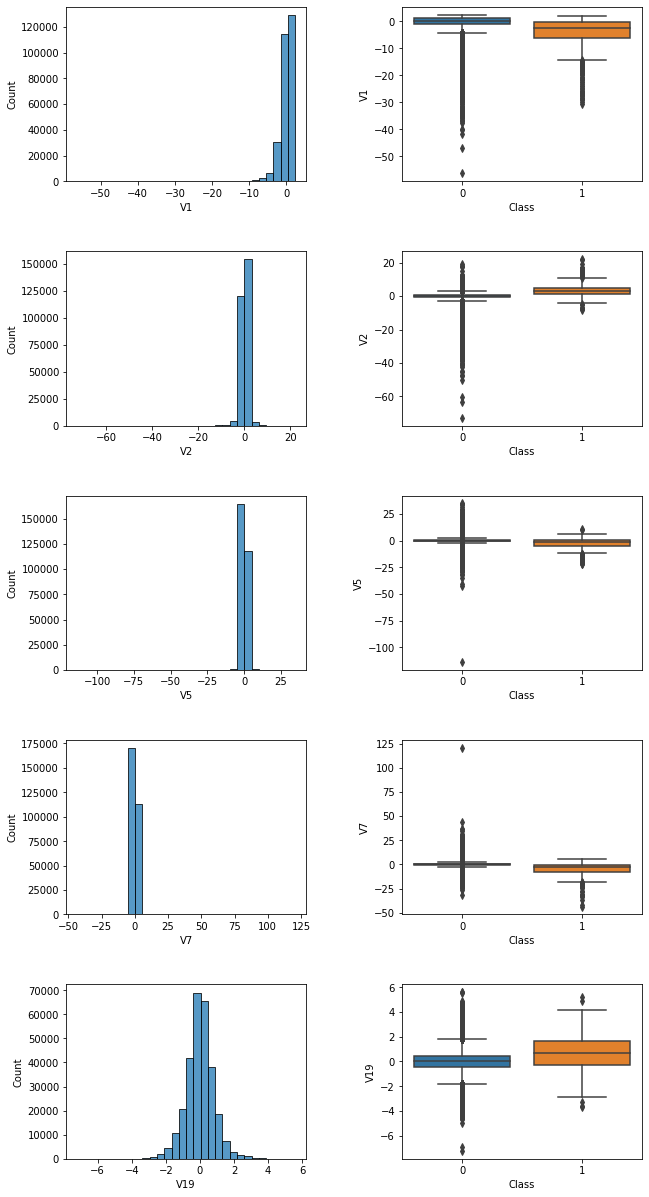
                    bottom=0.1,

                    right=0.9,

                    top=0.9,

                    wspace=0.4,

                    hspace=0.4)



#Utilizing Box Plot (1): Detecting Possibly Non-significant Variables

col\_name = 'V6'

fig, ax = plt.subplots(13,2,figsize=(10,30))

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[0,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[0,1])

col\_name = 'V8'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[1,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[1,1])

col\_name = 'V13'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[2,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[2,1])

col\_name = 'V15'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[3,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[3,1])

col\_name = 'V20'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[4,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[4,1])

col\_name = 'V21'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[5,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[5,1])

col\_name = 'V22'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[6,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[6,1])

col\_name = 'V23'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[7,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[7,1])

col\_name = 'V24'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[8,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[8,1])

col\_name = 'V25'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[9,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[9,1])

col\_name = 'V26'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[10,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[10,1])

col\_name = 'V27'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[11,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[11,1])

col\_name = 'V28'

sns.histplot(data=rawData, x=col\_name, bins=30, ax=ax[12,0])

sns.boxplot(data=rawData, x='Class', y=col\_name, ax=ax[12,1])

plt.subplots\_adjust(left=0.1,

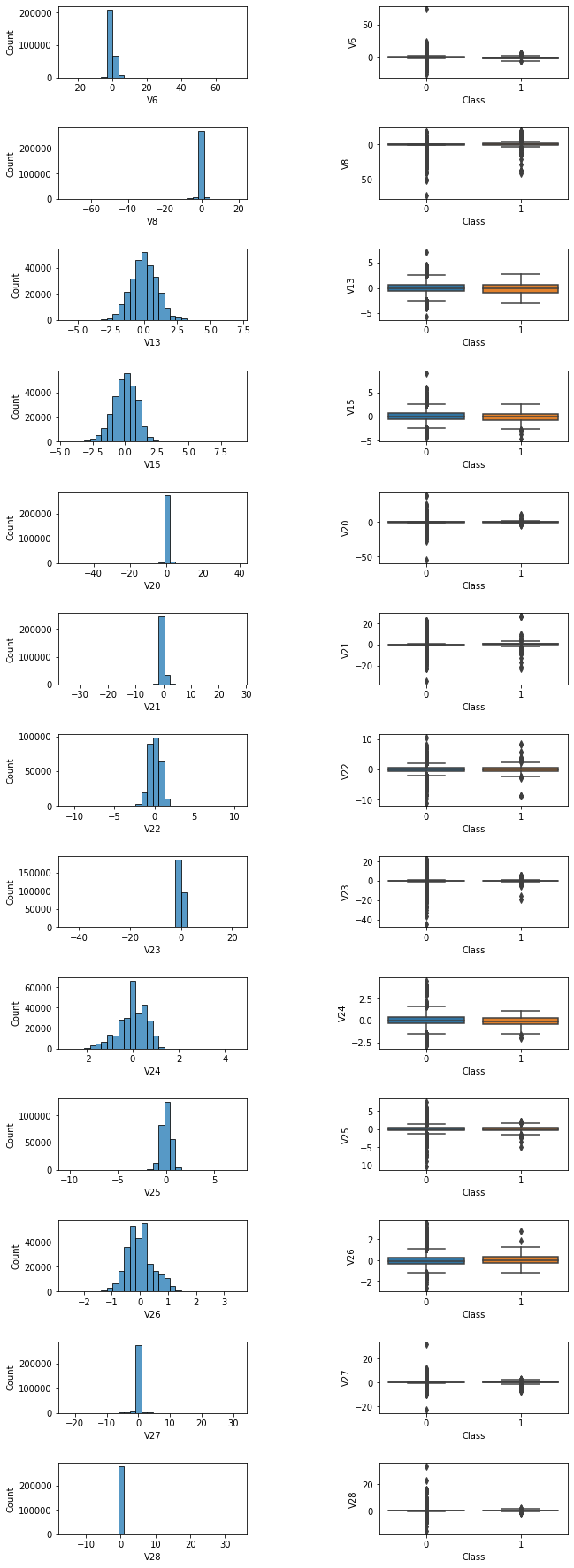
                    bottom=0.1,

                    right=0.9,

                    top=0.9,

                    wspace=0.7,

                    hspace=0.7)



3. Base Modelling

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import StratifiedKFold, GridSearchCV, train\_test\_split

rawData.columns

OUTPUT:

Index(['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'], dtype='object')

x = rawData[[

    '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'

]]

y = rawData['Class']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

folds = StratifiedKFold(n\_splits=10, shuffle=True)

model = LogisticRegression(class\_weight='balanced').fit(x\_train, y\_train)

yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

from sklearn.metrics import accuracy\_score, f1\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

accuracy\_score(y\_train, yhat\_train)

OUTPUT:

0.9732649826448105

accuracy\_score(y\_test, yhat\_test)

OUTPUT:

0.9730814695176901

f1\_score(y\_train, yhat\_train)

OUTPUT:

0.1042016806722689

f1\_score(y\_test, yhat\_test)

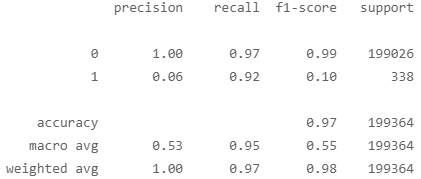
OUTPUT:

0.11059551430781128

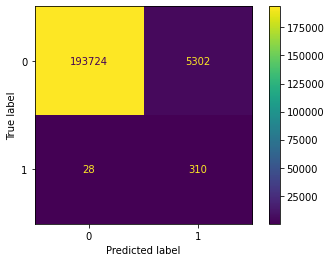
confusion\_matrix(y\_train, yhat\_train)



print(classification\_report(y\_train, yhat\_train))



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_train, yhat\_train)).plot()



As implied from the evaluation result and the confusion matrix, data imbalance is hindering the accurate modelling procedure. This calls for methods to solve data imbalance problems, namely, via adopting undersampling and oversampling methods.

4. Oversampling and Undersampling Methods

In order to illustrate the mechanisms of sampling, let us consider following sample data for better explanation.

from imblearn.over\_sampling import SMOTE

from sklearn import datasets

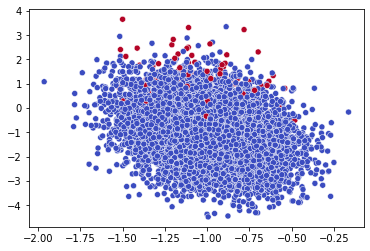
x, y = datasets.make\_classification(

    n\_samples=10000, n\_features=2, random\_state=12,

    n\_redundant=0, n\_clusters\_per\_class=1, weights=[0.99], flip\_y=0

)

sns.scatterplot(x = x[:,0], y=x[:,1], c=y, cmap='coolwarm')



from collections import Counter

Counter(y)

OUTPUT:

Counter({0: 9900, 1: 100})

sample = SMOTE()

x\_over, y\_over = sample.fit\_resample(x, y)

4.1. Oversampling

Oversampling SMOTE method increases the total number of data (for the category with less data, and thus in this case, for variable category ‘1’), accordingly and fitting to the original distribution shape of such data.

sample = SMOTE()

x\_over, y\_over = sample.fit\_resample(x, y)

Counter(y\_over)

OUTPUT:

Counter({0: 9900, 1: 9900})

4.2. Undersampling

Undersampling method, on the other hand, decreases the total number of majority class data, and there are several ways to do so, as follows.

4.2.1. Undersampling: Random Sampling

Random sampling method literally erases the data randomly, until the ratio between the classes become 1:1.

from imblearn.under\_sampling import \*

sample = RandomUnderSampler()

x\_under\_random, y\_under\_random = sample.fit\_resample(x, y)

Counter(y\_under\_random)

OUTPUT:

Counter({0: 100, 1: 100})

4.2.2. Undersampling: Tomeklink

Tomeklink refers to the data where majority and minority categorizations are close to each other. The Tomeklink method decreases the total number of majority data via locating these tomeklinks.

sample = TomekLinks()

x\_under\_tomek, y\_under\_tomek = sample.fit\_resample(x, y)

Counter(y\_under\_tomek)

Counter({0: 9841, 1: 100})

4.2.3. Undersampling: CNN (Condensed Nearest Neighbors)

CNN method is the application of KNN algorithm that we have covered in the previous chapter. More specifically, CNN method erases the neighboring majority class via setting K to 1.

sample = CondensedNearestNeighbour()

x\_under\_cnn, y\_under\_cnn = sample.fit\_resample(x, y)

Counter(y\_under\_cnn)

OUTPUT:

Counter({0: 355, 1: 100})

4.2.4. Undersampling: One-sided Selection

One-sided selection refers to the algorithm method that combines tomeklink method and CNN method together, as follows.

sample = OneSidedSelection()

x\_under\_tc, y\_under\_tc = sample.fit\_resample(x, y)

Counter(y\_under\_tc)

OUTPUT:

Counter({0: 7405, 1: 100})

4.2.5. Undersampling: ENN (Edited Nearest Neighbors)

Edited Nearest Neighbors method is another application of KNN algorithm. Majority class data neighboring the minority class data are preferentially erased, as follows.

sample = EditedNearestNeighbours()

x\_under\_enn, y\_under\_enn = sample.fit\_resample(x, y)

Counter(y\_under\_enn)

OUTPUT:

Counter({0: 9676, 1: 100})

4.3. Combined Method (Undersampling + Oversampling)

Combined method concurrently increases the total number of minority data while decreasing the total number of majority data. In most cases, this is done via combining various types of undersampling methods to the SMOTE method, as follows.

from imblearn.combine import \*

#SMOTE + ENN

sample = SMOTEENN()

x\_smote\_enn, y\_smote\_enn = sample.fit\_resample(x, y)

Counter(y\_smote\_enn)

OUTPUT:

Counter({0: 8438, 1: 8629})

#SMOTE + Tomeklink

sample = SMOTETomek()

x\_smote\_tomek, y\_smote\_tomek = sample.fit\_resample(x, y)

Counter(y\_smote\_tomek)

OUTPUT:

Counter({0: 9616, 1: 9616})

4.3. Comparisons and Visualization of Undersampling/Oversampling Methods

fig, ax = plt.subplots(3, 3, figsize=(20,10) )

ax[0,0].set\_title('Original Data')

sns.scatterplot(x = x[:,0], y = x[:, 1], c=y, cmap='coolwarm', ax=ax[0,0])

ax[0,1].set\_title('Oversampling: SMOTE')

sns.scatterplot(x = x\_over[:,0], y = x\_over[:, 1], c=y\_over, cmap='coolwarm', ax=ax[0,1])

ax[0,2].set\_title('Undersampling: Random Sampling')

sns.scatterplot(x = x\_under\_random[:,0], y = x\_under\_random[:, 1], c=y\_under\_random, cmap='coolwarm', ax=ax[0,2])

ax[1,0].set\_title('Undersampling: Tomeklink')

sns.scatterplot(x = x\_under\_tomek[:,0], y = x\_under\_tomek[:, 1], c=y\_under\_tomek, cmap='coolwarm', ax=ax[1,0])

ax[1,1].set\_title('Undersampling: CNN')

sns.scatterplot(x = x\_under\_cnn[:,0], y = x\_under\_cnn[:, 1], c=y\_under\_cnn, cmap='coolwarm', ax=ax[1,1])

ax[1,2].set\_title('Undersampling: Tomeklink + cnn')

sns.scatterplot(x = x\_under\_tc[:,0], y = x\_under\_tc[:, 1], c=y\_under\_tc, cmap='coolwarm', ax=ax[1,2])

ax[2,0].set\_title('Undersampling - ENN')

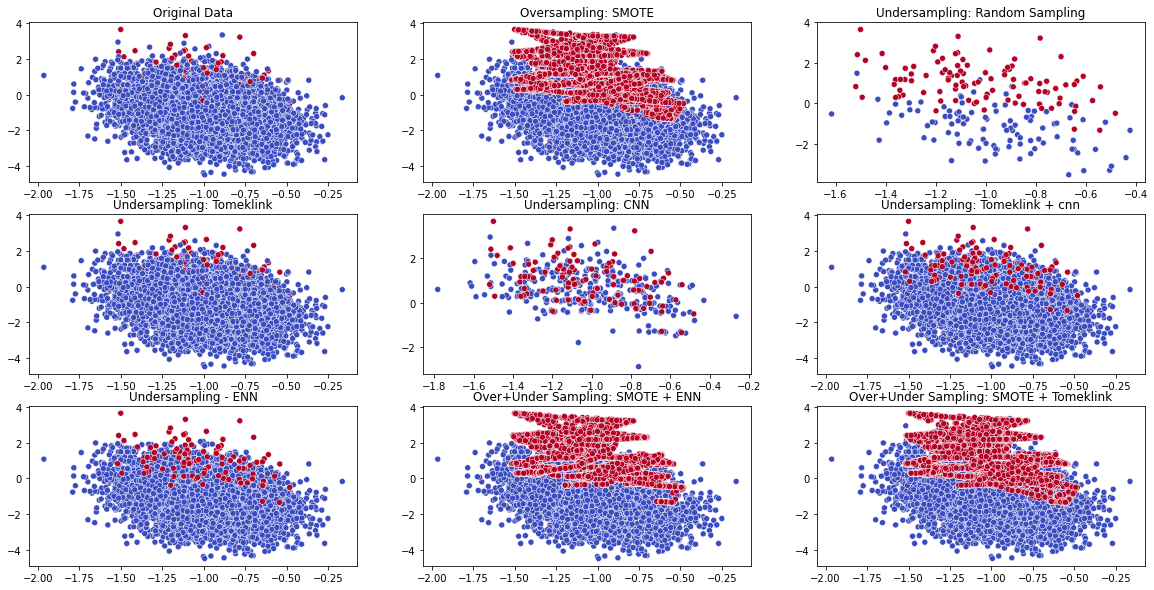
sns.scatterplot(x = x\_under\_enn[:,0], y = x\_under\_enn[:, 1], c=y\_under\_enn, cmap='coolwarm', ax=ax[2,0])

ax[2,1].set\_title('Over+Under Sampling: SMOTE + ENN')

sns.scatterplot(x = x\_smote\_enn[:,0], y = x\_smote\_enn[:, 1], c=y\_smote\_enn, cmap='coolwarm', ax=ax[2,1])

ax[2,2].set\_title('Over+Under Sampling: SMOTE + Tomeklink')

sns.scatterplot(x = x\_smote\_tomek[:,0], y = x\_smote\_tomek[:, 1], c=y\_smote\_tomek, cmap='coolwarm', ax=ax[2,2])



5. Sampling Application

Now, let us apply the sampling methods to the datasets to solve the data imbalance problem presented earlier.

x = rawData[[

  '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'

]]

y = rawData['Class']

#Oversampling

overSampler = SMOTE()

x\_over, y\_over = overSampler.fit\_resample(x, y)

Counter(y\_over)

OUTPUT:

Counter({0: 284315, 1: 284315})

x\_over = pd.DataFrame(x\_over, columns=x.columns)

y\_over = pd.DataFrame(y\_over, columns=['Class'])

rawData\_over = pd.concat([x\_over, y\_over], axis=1)

Let us check if the sampling process has been performed as desired via considering the respective shapes of the distributions for category 0 and 1:

plt.figure(figsize=(20,3))

plt.subplot(1,3,1)

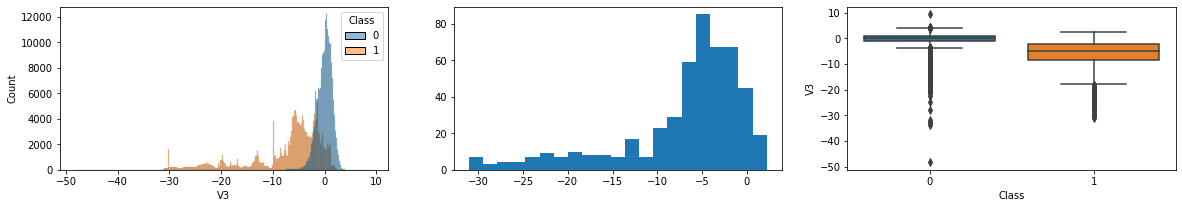
sns.histplot(data=rawData\_over, x='V3', hue='Class')

plt.subplot(1,3,2)

plt.hist(rawData[rawData['Class']==1]['V3'], bins='auto')

plt.subplot(1,3,3)

sns.boxplot(data=rawData, x='Class', y='V3')



plt.figure(figsize=(20,3))

plt.subplot(1,3,1)

sns.histplot(data=rawData\_over, x='V4', hue='Class')

plt.subplot(1,3,2)

plt.hist(rawData[rawData['Class']==1]['V4'], bins='auto')

plt.subplot(1,3,3)

sns.boxplot(data=rawData, x='Class', y='V4')



plt.figure(figsize=(20,3))

plt.subplot(1,3,1)

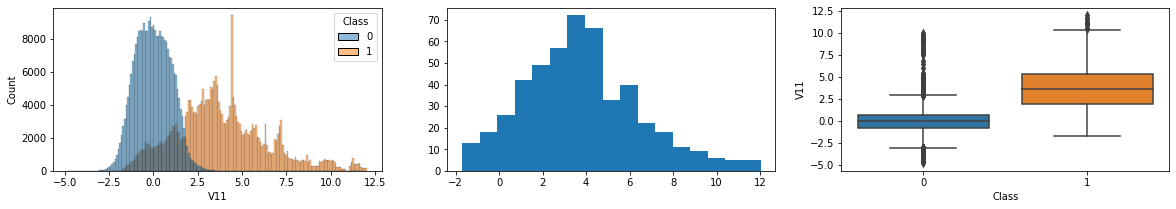
sns.histplot(data=rawData\_over, x='V11', hue='Class')

plt.subplot(1,3,2)

plt.hist(rawData[rawData['Class']==1]['V11'], bins='auto')

plt.subplot(1,3,3)

sns.boxplot(data=rawData, x='Class', y='V11')



plt.figure(figsize=(20,3))

plt.subplot(1,3,1)

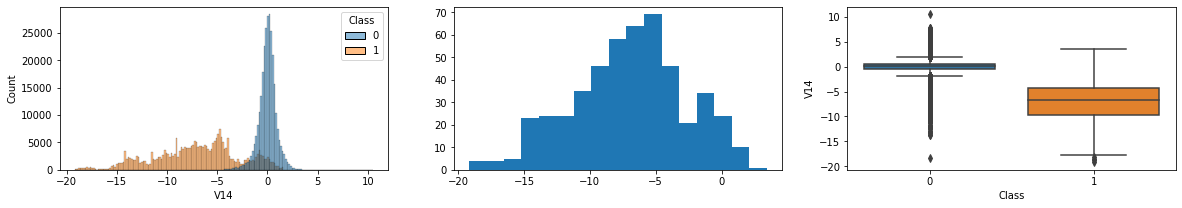
sns.histplot(data=rawData\_over, x='V14', hue='Class')

plt.subplot(1,3,2)

plt.hist(rawData[rawData['Class']==1]['V14'], bins='auto')

plt.subplot(1,3,3)

sns.boxplot(data=rawData, x='Class', y='V14')



#Undersampling

from imblearn.under\_sampling import RandomUnderSampler

from collections import Counter

sampler = RandomUnderSampler()

x\_under, y\_under = sampler.fit\_resample(x, y)

Counter(y\_under)

OUTPUT:

Counter({0: 492, 1: 492})

x\_under = pd.DataFrame(x\_under, columns=x.columns)

y\_under = pd.DataFrame(y\_under, columns=['Class'])

rawData\_under = pd.concat([x\_under, y\_under], axis=1)

plt.figure(figsize=(20,3))

plt.subplot(1,3,1)

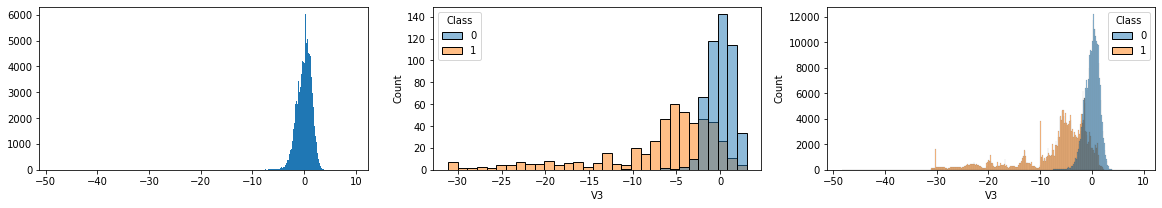
plt.hist(rawData[rawData['Class']==0]['V3'], bins='auto')

plt.subplot(1,3,2)

sns.histplot(data=rawData\_under, x='V3', hue='Class')

plt.subplot(1,3,3)

sns.histplot(data=rawData\_over, x='V3', hue='Class')



plt.figure(figsize=(20,3))

plt.subplot(1,3,1)

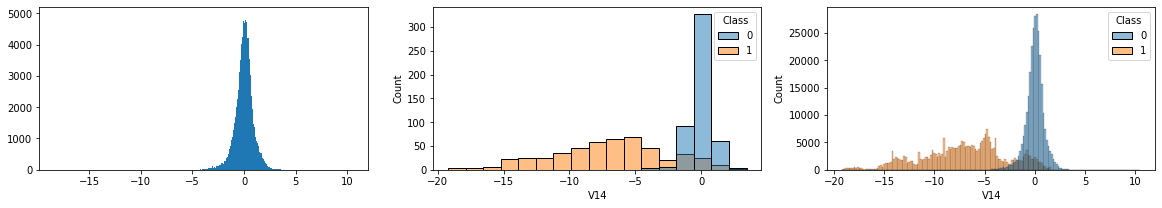
plt.hist(rawData[rawData['Class']==0]['V14'], bins='auto')

plt.subplot(1,3,2)

sns.histplot(data=rawData\_under, x='V14', hue='Class')

plt.subplot(1,3,3)

sns.histplot(data=rawData\_over, x='V14', hue='Class')



len(x\_under)

OUTPUT:

984

6. Modelling with Undersampling Data

6.1. Generalized Linear Model

from sklearn.model\_selection import train\_test\_split, StratifiedKFold

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_under, y\_under, test\_size=0.3)

folds = StratifiedKFold(n\_splits=10, shuffle=True)

model = LogisticRegression()

params = dict(

    penalty = ['elasticnet', 'none'],

    C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    class\_weight = ['balanced', None],

    solver= ['saga'],

    l1\_ratio = [0, 0.3, 0.5, 0.8, 1]

)

search = GridSearchCV(model, param\_grid=params, cv=folds, scoring='roc\_auc')

search.fit(x\_train, y\_train)

search.best\_params\_

OUTPUT:

{'C': 0.1, 'class\_weight': None, 'l1\_ratio': 0.3, 'penalty': 'elasticnet', 'solver': 'saga'}

search.best\_score\_

OUTPUT:

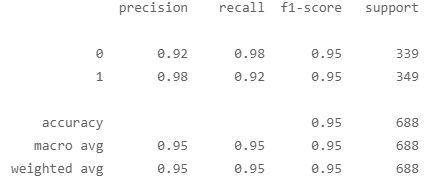
0.987230635569736

model = search.best\_estimator\_

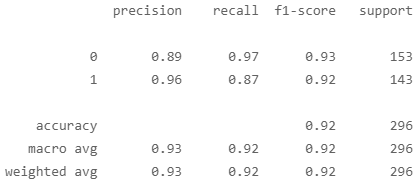
yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

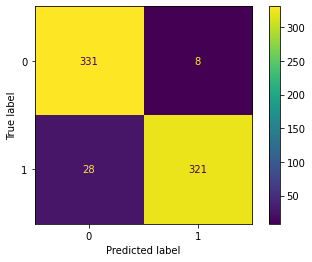
print(classification\_report(y\_train, yhat\_train))



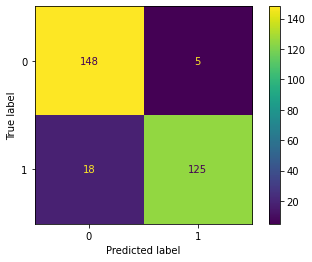
print(classification\_report(y\_test, yhat\_test))



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_train, yhat\_train)).plot()



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_test, yhat\_test)).plot()



6.2. Verification of Regression Coefficients for Feature Selection

model.coef\_

OUTPUT:

array([[-2.06394437e-01, -6.96322724e-02, -2.19088895e-01, 4.22964898e-01, 1.35239856e-03, -1.65781232e-01, -3.20149722e-02, -5.18386016e-02, 0.00000000e+00, -6.09655379e-02, 2.67844760e-01, -2.80885540e-01, 0.00000000e+00, -4.10145844e-01, -7.23437494e-03, -5.03087916e-02, -8.40552642e-02, 9.61049529e-02, 0.00000000e+00, -1.00016193e-02, 0.00000000e+00, 6.98620587e-03, -1.05289933e-01, 3.28328258e-06, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])

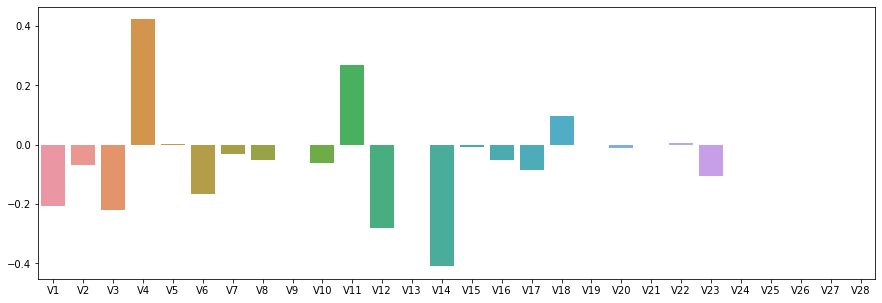
model.coef\_[0]

OUTPUT:

array([-2.06394437e-01, -6.96322724e-02, -2.19088895e-01, 4.22964898e-01, 1.35239856e-03, -1.65781232e-01, -3.20149722e-02, -5.18386016e-02, 0.00000000e+00, -6.09655379e-02, 2.67844760e-01, -2.80885540e-01, 0.00000000e+00, -4.10145844e-01, -7.23437494e-03, -5.03087916e-02, -8.40552642e-02, 9.61049529e-02, 0.00000000e+00, -1.00016193e-02, 0.00000000e+00, 6.98620587e-03, -1.05289933e-01, 3.28328258e-06, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00])

plt.figure(figsize=(15,5))

sns.barplot(x = x\_test.columns, y = model.coef\_[0])



x\_under\_selected = x\_under[['V1', 'V3', 'V4', 'V6', 'V11', 'V12', 'V14']]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_under\_selected, y\_under, test\_size=0.3)

model = LogisticRegression(solver='saga')

params = dict(

    penalty = ['elasticnet', 'none'],

    C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    l1\_ratio = [0, 0.3, 0.5, 0.8, 1]

)

search = GridSearchCV(model, param\_grid=params, cv=folds, scoring='roc\_auc')

search.fit(x\_train, y\_train)

search.best\_params\_

OUTPUT:

{'C': 0.01, 'l1\_ratio': 0, 'penalty': 'elasticnet'}

search.best\_score\_

OUTPUT:

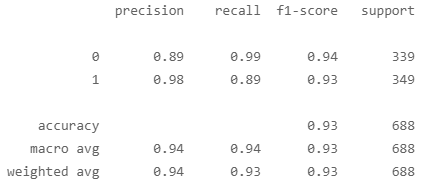
0.9814616755793226

model = search.best\_estimator\_

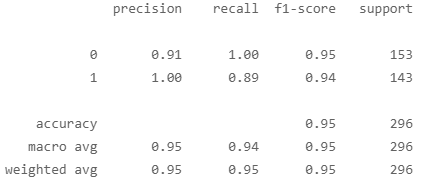
yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

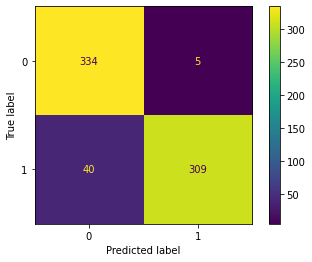
print(classification\_report(y\_train, yhat\_train))



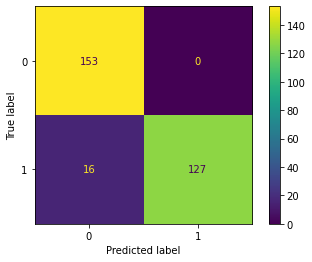
print(classification\_report(y\_test, yhat\_test))



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_train, yhat\_train)).plot()



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_test, yhat\_test)).plot()



As the evaluation result depicts, the target variable can be explained sufficiently enough with the following variables:

plt.figure(figsize=(15,5))

sns.barplot(x\_test.columns, model.coef\_[0])



6.3. Support Vector Machine

x\_under = pd.DataFrame(x\_under, columns=x.columns)

y\_under = pd.DataFrame(y\_under, columns=['Class'])

x\_under\_selected = x\_under[['V1','V3','V4','V6','V11','V12','V14']]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_under\_selected, y\_under, test\_size=0.3)

folds = StratifiedKFold(n\_splits=10, shuffle=True)

from sklearn.svm import SVC

model = SVC()

params = dict(

    C=[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

)

search = GridSearchCV(model, param\_grid=params, cv=folds, scoring='roc\_auc')

search.fit(x\_train, y\_train)

search.best\_params\_

OUTPUT:

{'C': 10, 'gamma': 0.0001}

search.best\_score\_

OUTPUT:

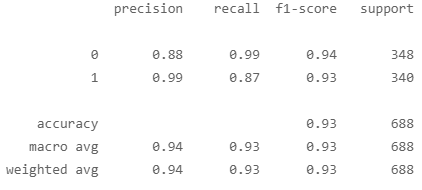
0.9826520019772615

model = search.best\_estimator\_

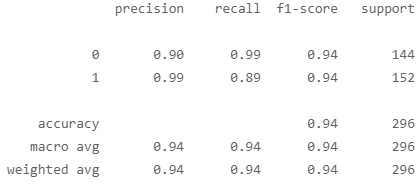
yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

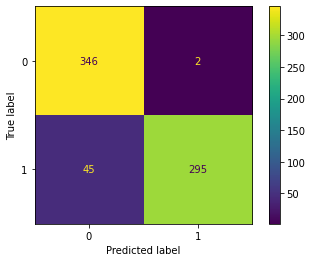
print(classification\_report(y\_train, yhat\_train))



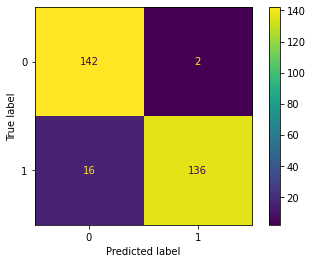
print(classification\_report(y\_test, yhat\_test))



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_train, yhat\_train)).plot()



ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_test, yhat\_test)).plot()

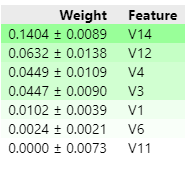


import eli5

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(model, random\_state=123).fit(x\_under\_selected, y\_under)

eli5.show\_weights(perm, feature\_names = x\_under\_selected.columns.tolist())



undersampling methods are combined to the SMOTE method

in relation to the categories 0 and 1

It can be implied from the box plots that variables V3, V4, V9,...V18 can be