Fundamentals of Machine Learning

Concepts, Techniques and Tools to Build Intelligent Systems

Module 5
Logistic Regression, Decision Tree & Supported Vector Machine(SVM)

Ali Samanipour

May. 2023

1. What is an Imbalanced Dataset?
2. Normalization
3. Scaling
4. Splitting Dataset
5. Handling Imbalance

6 Logistic Regression

7 Decision Tree

8 Support Vector Machine (SVM)

9 Data Preparation & Metric Trap

10 Training & Evaluation

Credit Card Fraud Detection

The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection

```
import pandas as pd
df = pd.read_csv("input/creditcard.csv")
df.shape
```

```
(284807, 31)
```

Let’s Understand the Data

Time: Number of seconds elapsed between this transaction and the first transaction in the dataset

```
df.columns
```

```
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')
```

Let’s Understand the Data …

Amount: Transaction amount

```
df.columns
```

```
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')
```

Let’s Understand the Data …

Class: 1 for fraudulent transactions, 0 non-fraudulent

```
df.columns
```

```
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')
```

Let's Understand the Data …

V1…, V28: These are the features of the dataset
(we will only be dependent on statistical metrics and relations to choose the best features for predicting the desired class)

```
df.columns

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')
```

1. What is an Imbalanced Dataset?
2. Normalization
3. Scaling
4. Splitting Dataset
5. Handling Imbalance

What is an Imbalanced Dataset?

An imbalanced dataset is when one output class has extremely high entries compared to the other output class

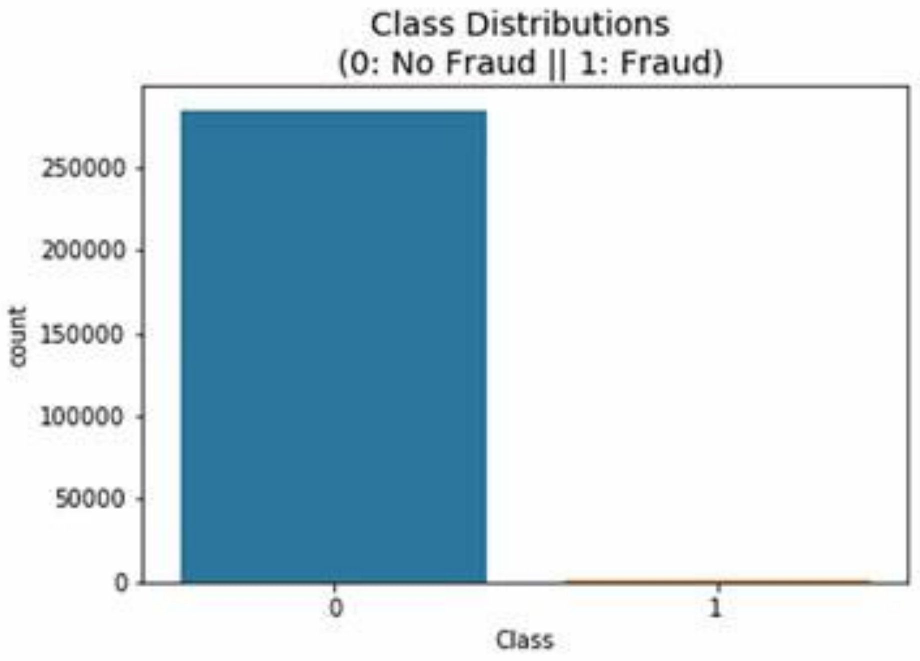
```
df.Class.value_counts()

0    284315
1       492
Name: Class, dtype: int64
```

Imbalanced Dataset

Let’s see how Imbalance Dataset looks like

```
sns.countplot('Class', data=df)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```



Counting Nulls

We do not have any NULL values to handle in the dataset. So, going ahead

```
df.isnull().sum().max()
```

```
0
```

Knowing the Features

Let us see some of the values for all the features and know, how the value looks

```
df.sort_index(axis=1).head(3)
```

	Amount	Class	Time	V1	V10	V11	V12	V13	V14	V15	...	V26
0	149.62	0	0.0	-1.359807	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	...	-0.189115
1	2.69	0	0.0	1.191857	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	...	0.125895
2	378.66	0	1.0	-1.358354	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	...	-0.139097

3 rows × 31 columns

Dataset Information

It's a relatively large dataset.

Training a model with all the features generally don't make sense.

We need to choose the features in such a way that those features will have some contribution to the decision of the output class.

```
df.sort_index(axis=1).info()
```

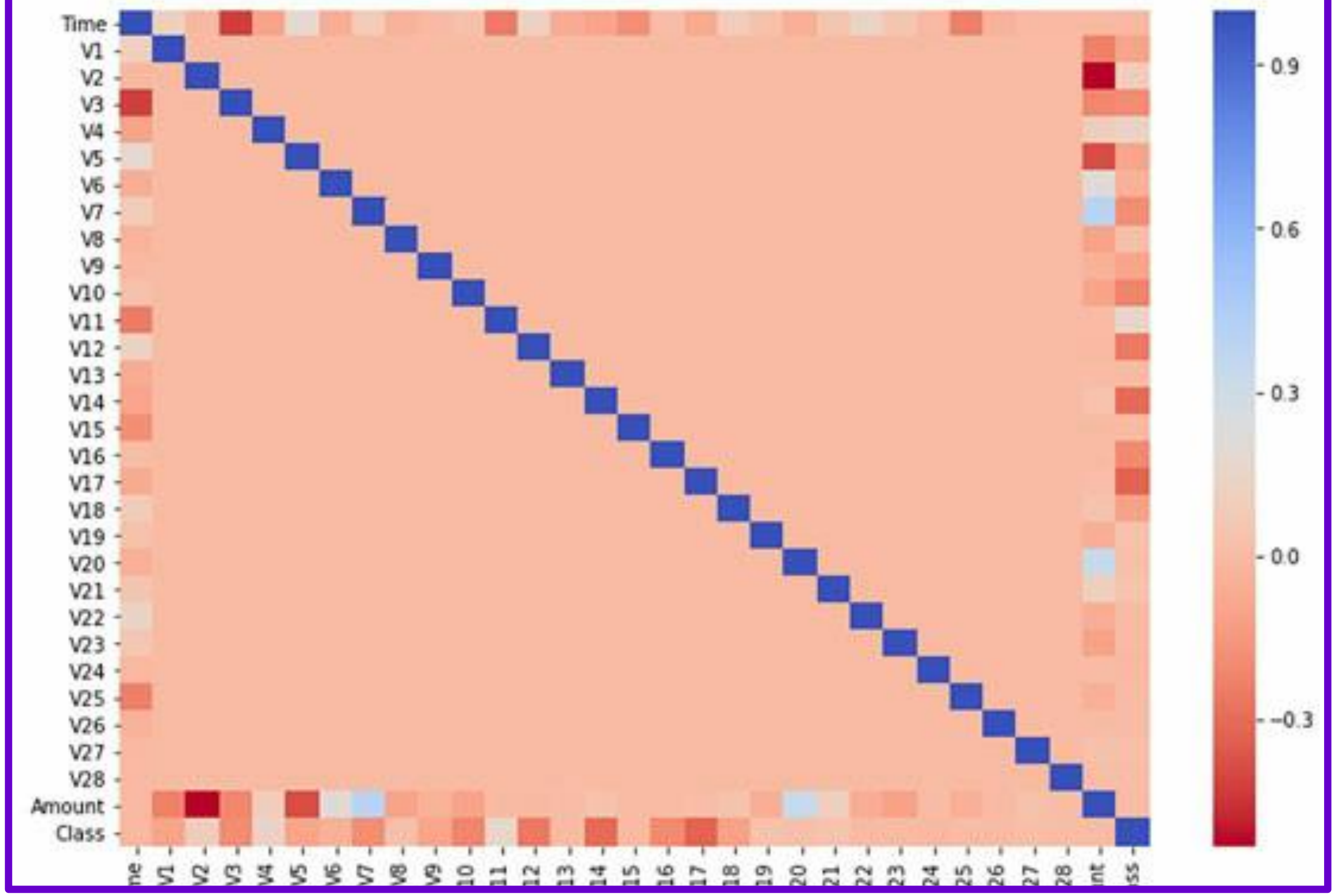
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Amount     284807 non-null float64
Class      284807 non-null int64
Time       284807 non-null float64
V1         284807 non-null float64
V10        284807 non-null float64
V11        284807 non-null float64
V12        284807 non-null float64
V13        284807 non-null float64
V14        284807 non-null float64
V15        284807 non-null float64
V16        284807 non-null float64
```

Correlation of heat map

We now need to find some features among 28 dimensionally reduced features that we can use to train the model

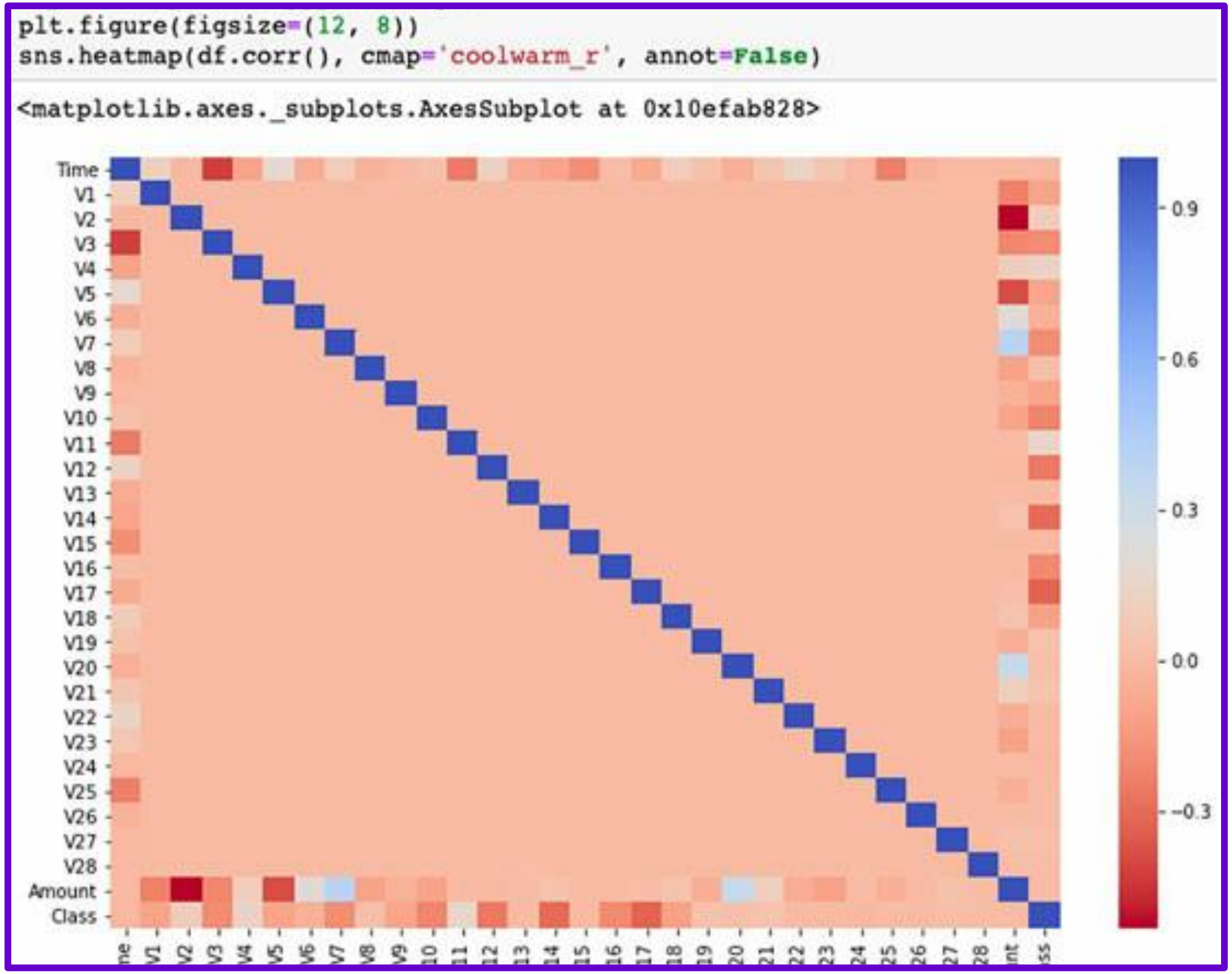
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap='coolwarm_r', annot=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10efab828>
```



Correlation of heat map …

Features like V1, V3, V5, V7, V9, V10, V12, V14, V16, V17, V18 have some relation with "Class" compared to the other features



1. What is an Imbalanced Dataset?
2. Normalization
3. Scaling
4. Splitting Dataset
5. Handling Imbalance

Normalization & Feature Scaling

All the 28 features are dimensionally reduced features through Principal Component Analysis (PCA). Before PCA is applied, the dataset is first normalized. So, we need to apply normalization to rest of the columns as well

```
df.sort_index(axis=1).head(3)
```

	Amount	Class	Time	V1	V10	V11	V12	V13	V14	V15	...	V26
0	149.62	0	0.0	-1.359807	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	...	-0.189115
1	2.69	0	0.0	1.191857	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	...	0.125895
2	378.66	0	1.0	-1.358354	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	...	-0.139097

3 rows × 31 columns

Normalization & Feature Scaling

Normalization is used in a variety of ways in statistics. It is also known as **feature scaling**. **The meaning we are going after, is the normalization of the range of the data to a standard scale.**

Case 1: Say we have four lengths; all of them are in centimeters, and only one of them is inches. So, in those cases, we need to follow one standard and generalize all the length units.

Case 2: We have two features like Age and Salary, and we can easily say that both of the features will have a different range. Hence, we will use normalization/scaling techniques to bring them on the same scale

Range for all columns

Seeing the feature ranges, we can see "Time" and "Amount" has not been scaled.

```
df.min()
```

Time	0.000000
V1	-56.407510
V2	-72.715728
V3	-48.325589
V4	-5.683171
V5	-113.743307
V6	-26.160506
V7	-43.557242
V8	-73.216718
V9	-13.434066
V10	-24.588262
V11	-4.797473
V12	-18.683715
V13	-5.791881
V14	-19.214325
V15	-4.498945
V16	-14.129855
V17	-25.162799
V18	-9.498746
V19	-7.213527
V20	-54.497720
V21	-34.830382
V22	-10.933144
V23	-44.807735
V24	-2.836627
V25	-10.295397
V26	-2.604551
V27	-22.565679
V28	-15.430084
Amount	0.000000
Class	0.000000

dtype: float64

```
df.max()
```

Time	172792.000000
V1	2.454930
V2	22.057729
V3	9.382558
V4	16.875344
V5	34.801666
V6	73.301626
V7	120.589494
V8	20.007208
V9	15.594995
V10	23.745136
V11	12.018913
V12	7.848392
V13	7.126883
V14	10.526766
V15	8.877742
V16	17.315112
V17	9.253526
V18	5.041069
V19	5.591971
V20	39.420904
V21	27.202839
V22	10.503090
V23	22.528412
V24	4.584549
V25	7.519589
V26	3.517346
V27	31.612198
V28	33.847808
Amount	25691.160000
Class	1.000000

dtype: float64

Min-max Feature Scaling (Rescaling)

Here we scale the entire column with any given range into a usable given range [a, b].

$$x' = a + \frac{(x_i - min(x))(b - a)}{max(x) - min(x)}$$

Scaling down using mix-max feature scaling, won’t change the distribution of the feature. It only changes the range and keeping the distribution the same for the feature

Standardization (**Z-score** Normalization)

In the process of standardization, each feature has **zero-mean** and **unit variance or standard deviation**

$$x' = \frac{x_i - \bar{x}}{\sigma}$$

Scaling down using standardization will change the distribution of the feature and try to make mean close to zero, and the standard deviation equals to one

Principal Component Analysis(PCA)

Principal component analysis (PCA) is a statistical method to explain variance and covariance structure of a set of variables through linear combination. It uses the concept of orthogonal transformation to convert the set of variables (mostly correlated variables) into a set of linearly uncorrelated values

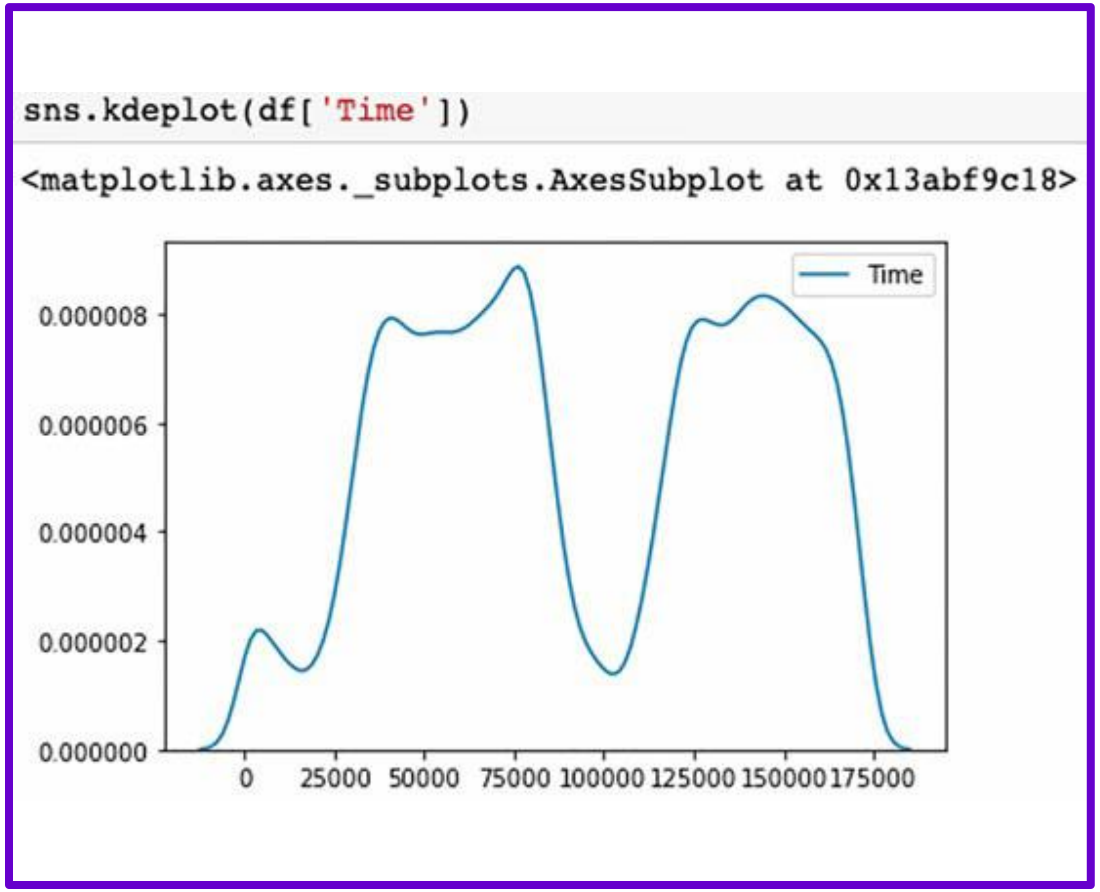
PCA is one of the popular methods to perform dimension reduction on a large dataset, and it helps in multiple ways like visualization, handling a smaller number of columns/features for modeling

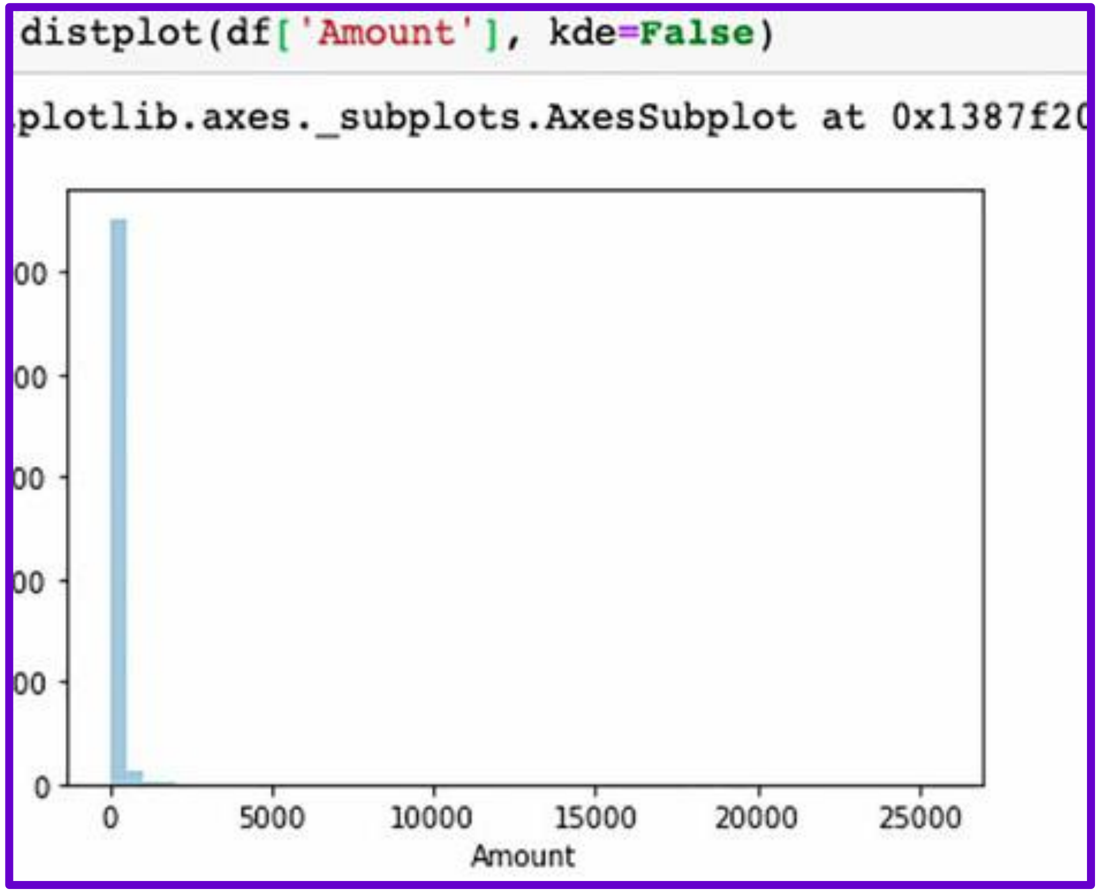
Cross-validation

Cross-validation is a model validation technique to ensure its performance, but it only works best when the training dataset distribution is similar to real-time data distribution.

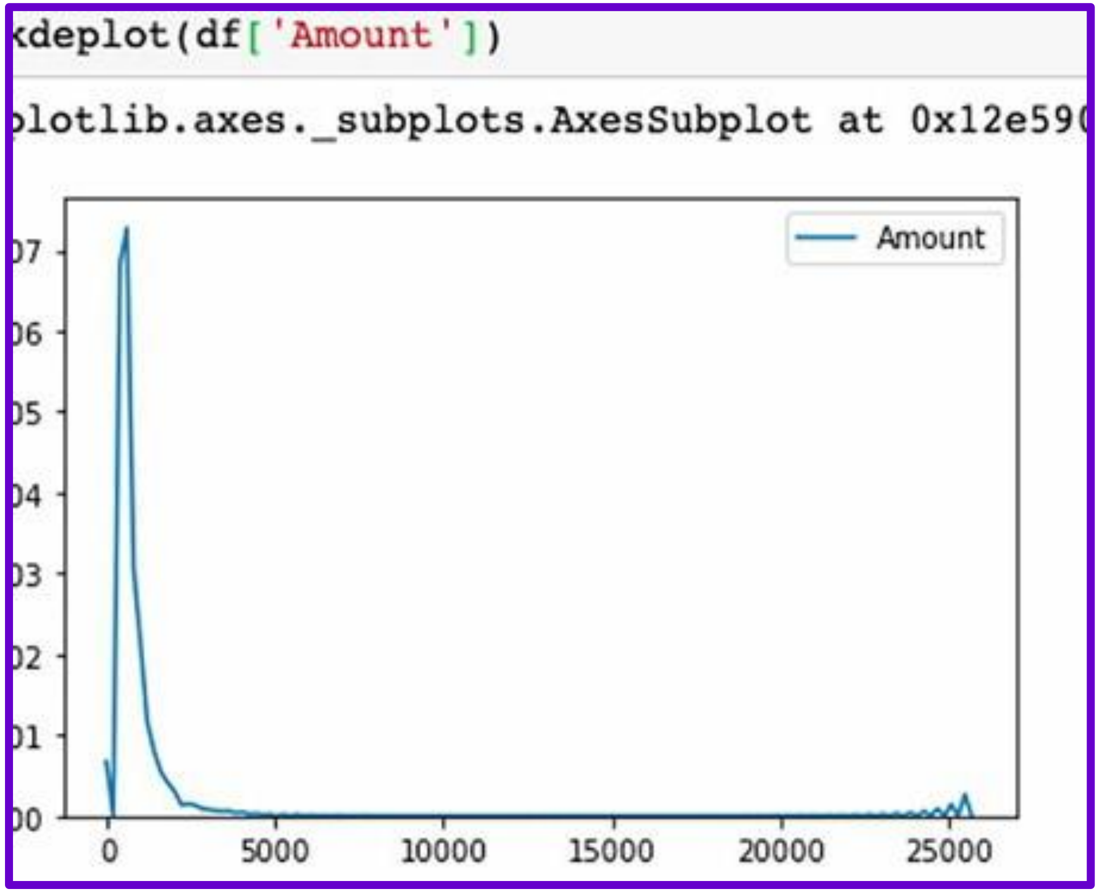
There are different types of cross-validation techniques: Exhaustive cross-validation like Leave-one-out cross-validation and Non-exhaustive cross-validation like k-fold cross-validation

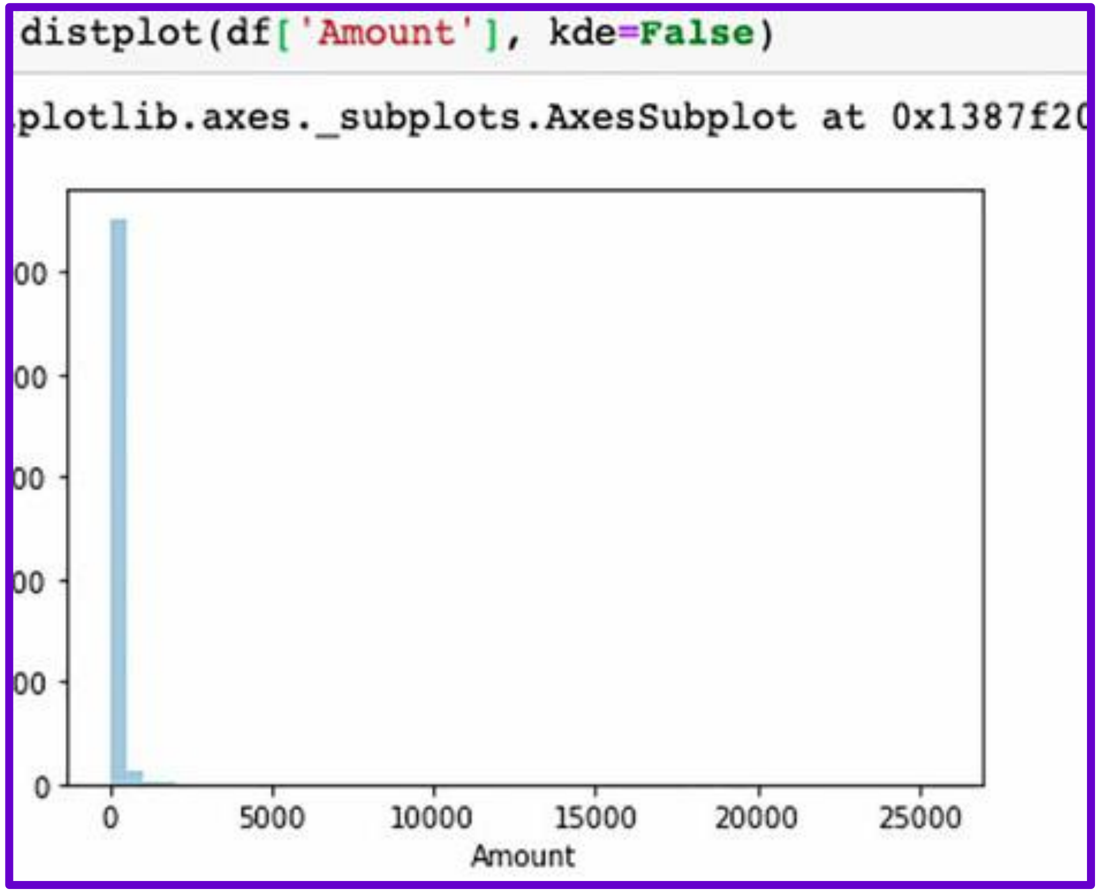
Bi-modal "Time" distribution plot



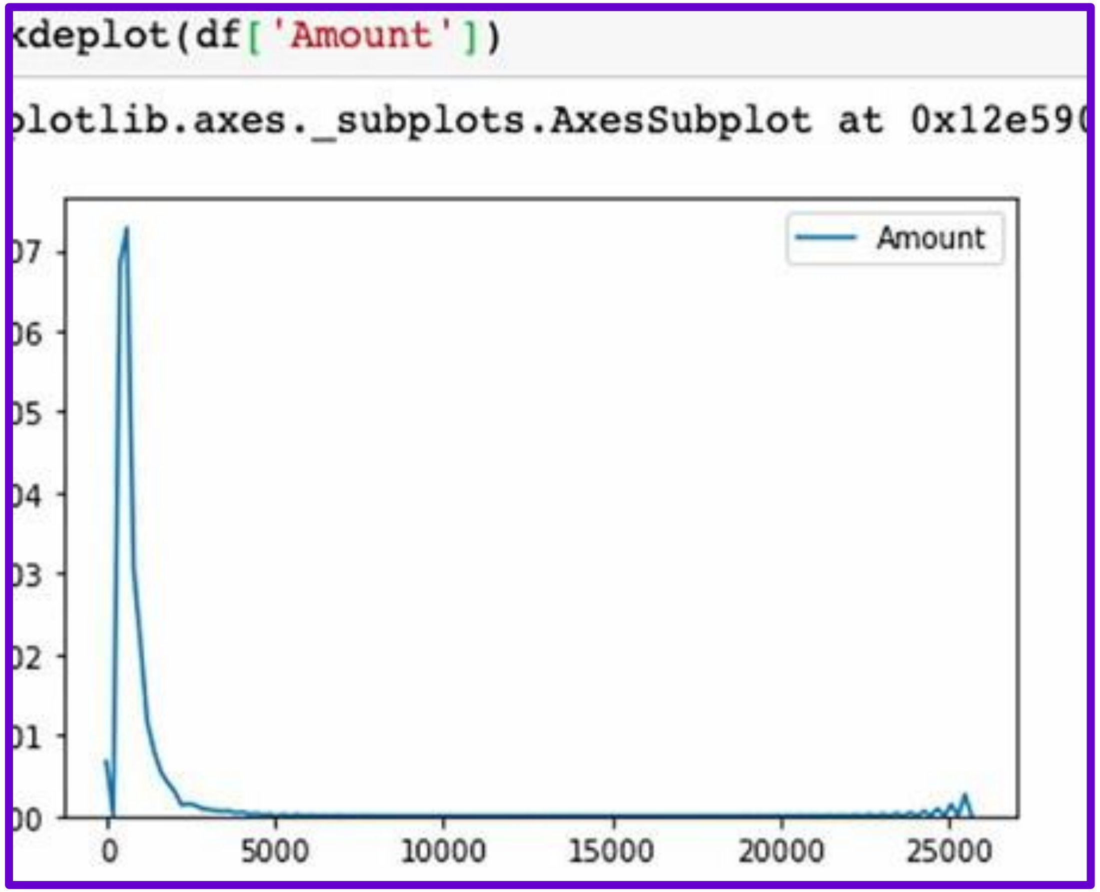


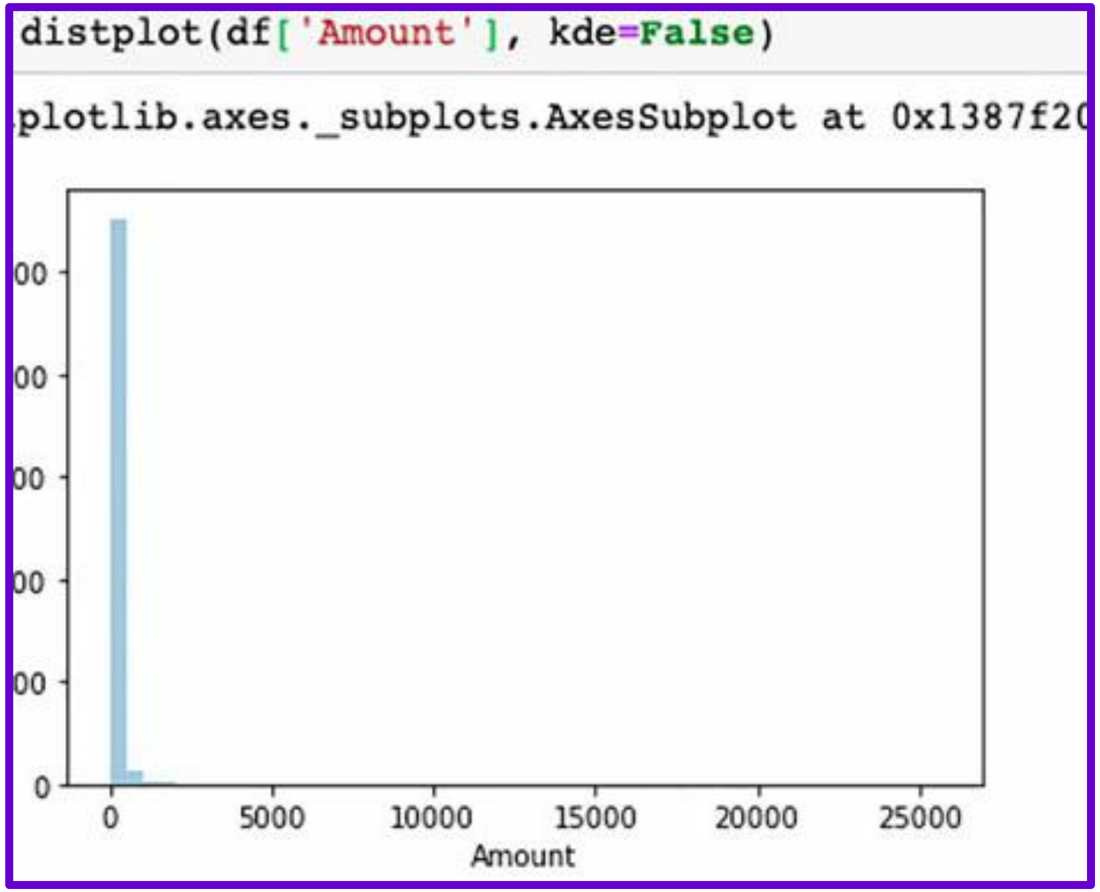
“Amount” Frequency Distribution





“Amount” Frequency Distribution

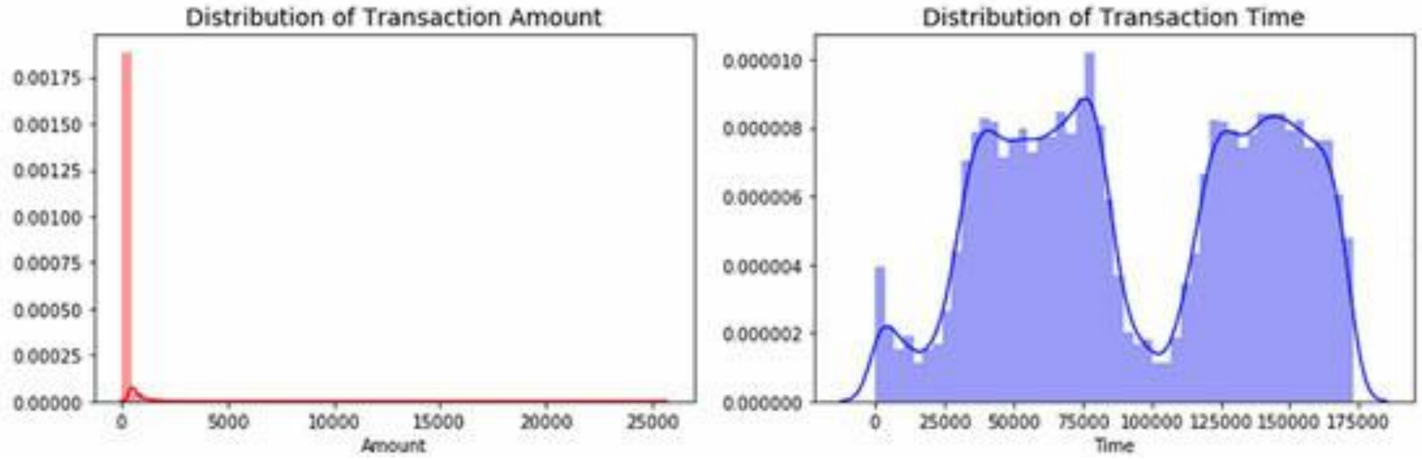




Comparison for an "Amount" and "Time" KDEs

We should try to see if there is any visual link/similarity among the distributions.

```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.distplot(df['Amount'], ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)

sns.distplot(df['Time'], ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)

plt.show()
```



Summed “Amount” grouped by “Time

we need to drop this idea to compare “Time” and “Amount” distribution separately.

```
df[['Time','Amount']].groupby(['Time']).head()
```

	Time	Amount
0	0.0	149.62
1	0.0	2.69
2	1.0	378.66
3	1.0	123.50
4	2.0	69.99
...	...	...
284802	172786.0	0.77
284803	172787.0	24.79
284804	172788.0	67.88
284805	172788.0	10.00
284806	172792.0	217.00

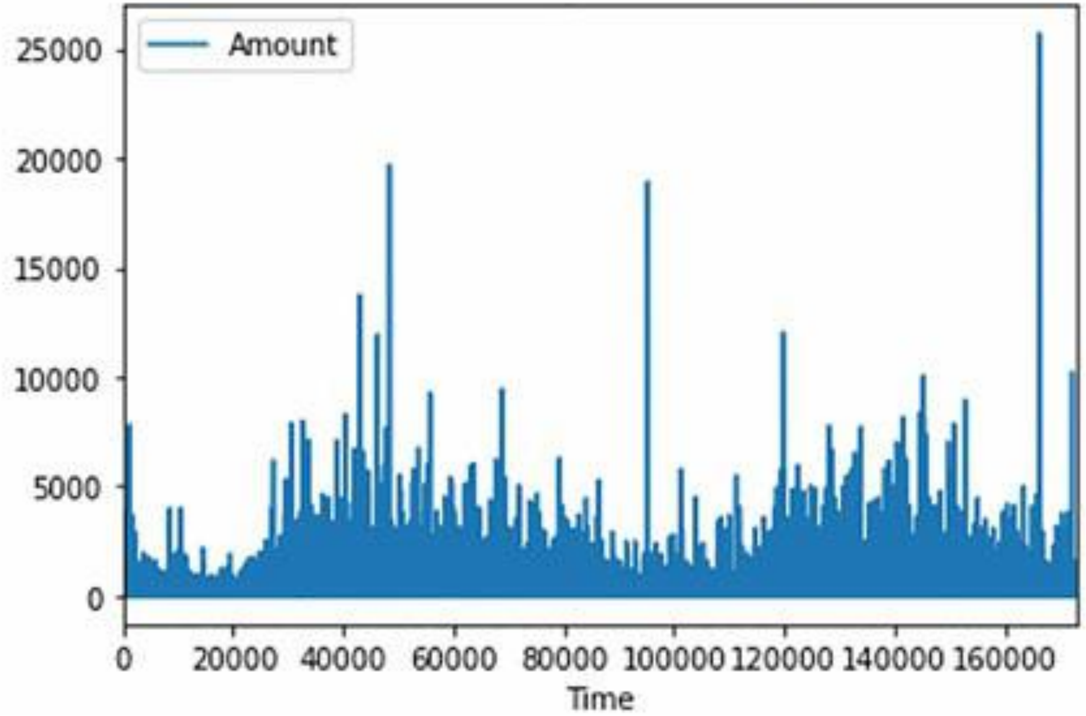
279146 rows × 2 columns

Summed “Amount” Bar Plot

It will make sense if we plot a chart between “Time” vs. “Amount.” We should know that for every “time” what will be the “amounts”

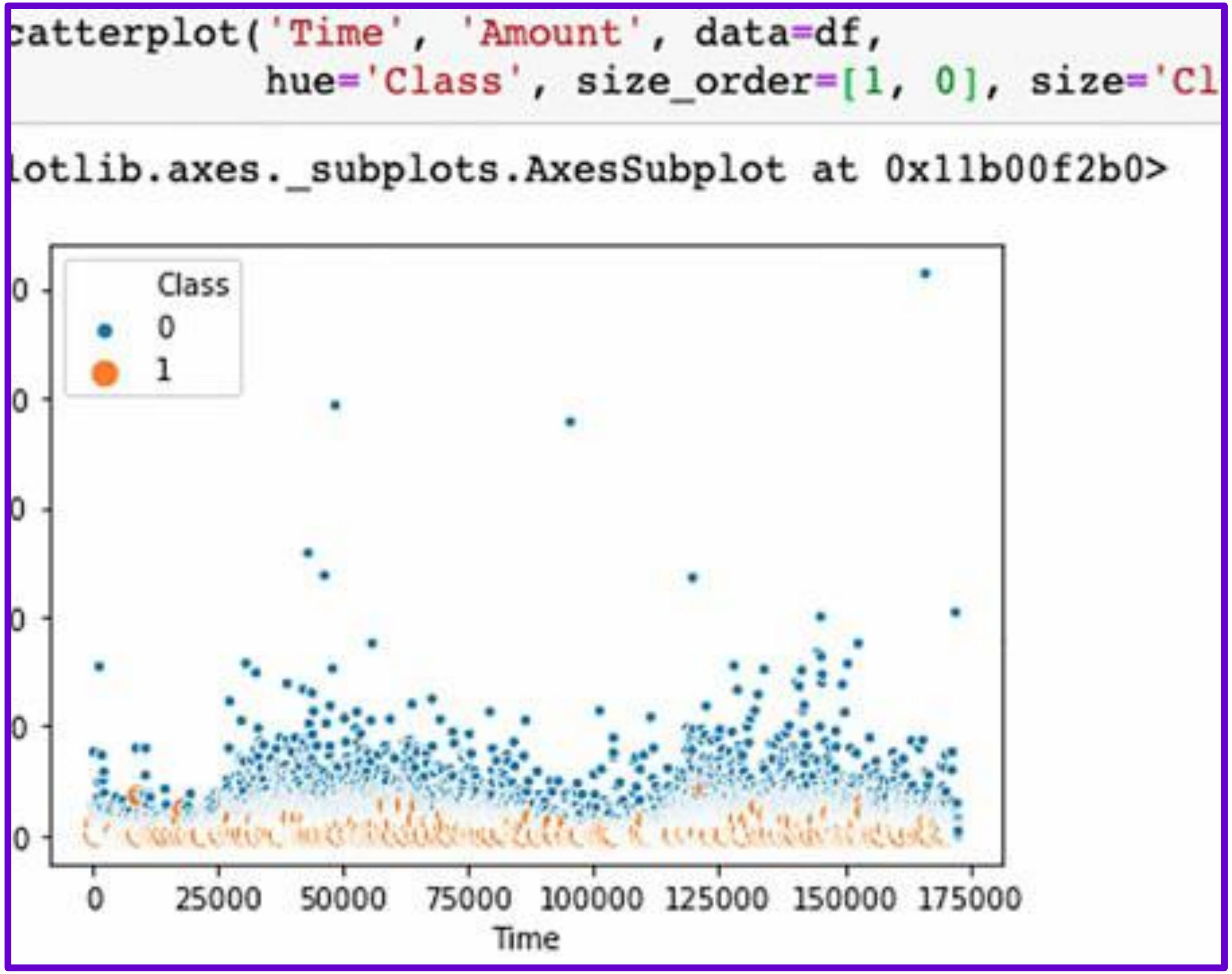
```
df[['Time','Amount']].groupby(['Time']).sum().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7126f5bad0>
```



Class-wise "Amount" vs. "Time" Scatter Plot

Need a plot that will help us to see the fraudulent transactions and the range for amounts



Range for Fraudulent Class

The above figure shows the range for the fraudulent transaction to be around [0, 2500]. We can test our hypothesis out and the range of the fraudulent transaction amount

```
f_df.Amount.min(), f_df.Amount.max()
```

```
(0.0, 2125.87)
```

Count of records where “Amount” is zero

How can there be a fraud when the transaction value is zero?

```
df[df["Amount"]==0].shape
```

```
(1825, 31)
```

Count of records where “Amount” is zero

We have a small chunk of records where “Amount” equals to zero. We need to see the breakdown down of “Class” as well and decide whether to keep it or ignore it.

```
df[df["Amount"]==0]["Class"].value_counts()
```

```
0    1798
1      27
Name: Class, dtype: int64
```

Count of records where “Amount” is zero

Seeing the breakdown, we can clearly say that it is not significant enough to impact the decision or prediction.
We will see more distributions and later decide to keep it or not.

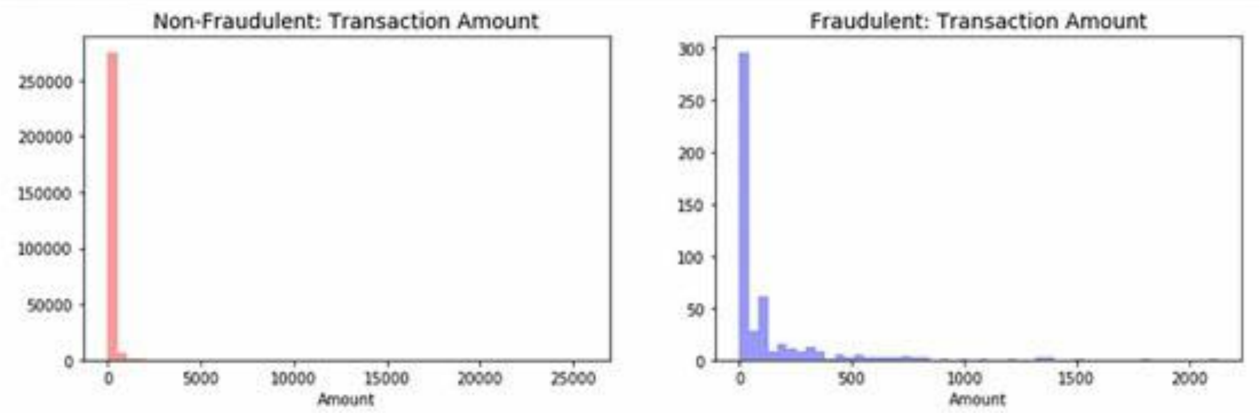
```
df[df["Amount"]==0]["Class"].value_counts()
```

```
0    1798
1      27
Name: Class, dtype: int64
```

Shape of Fraudulent and Non-fraudulent Class

Seeing the above image, we can see the distribution is same or nearly identical if we compare them.

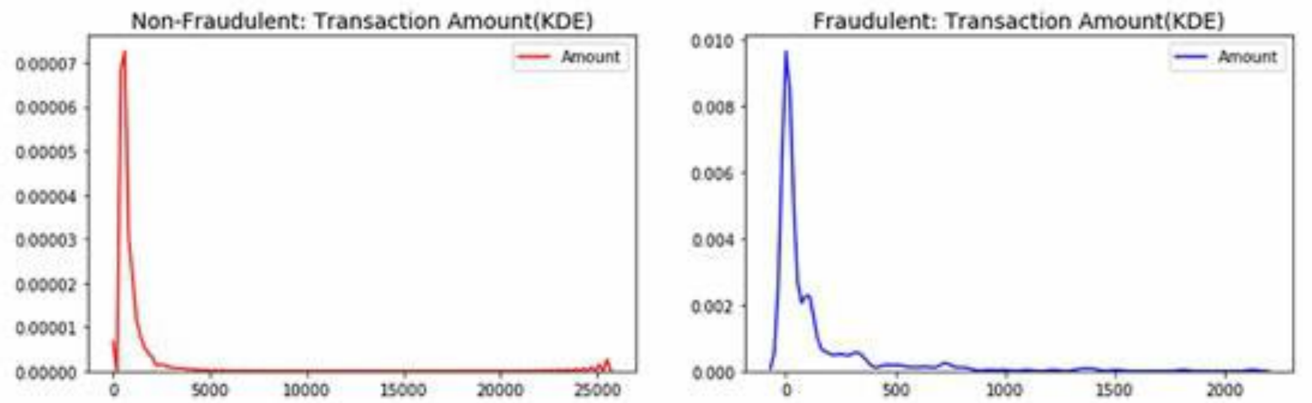
```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.distplot(n_df['Amount'], kde=False, ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Amount', fontsize=14)

sns.distplot(f_df['Amount'], kde=False, ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Amount', fontsize=14)

plt.show()
```



Shape of Fraudulent and Non-fraudulent Class

KDE will be same as well, which we can confirm by plotting it.

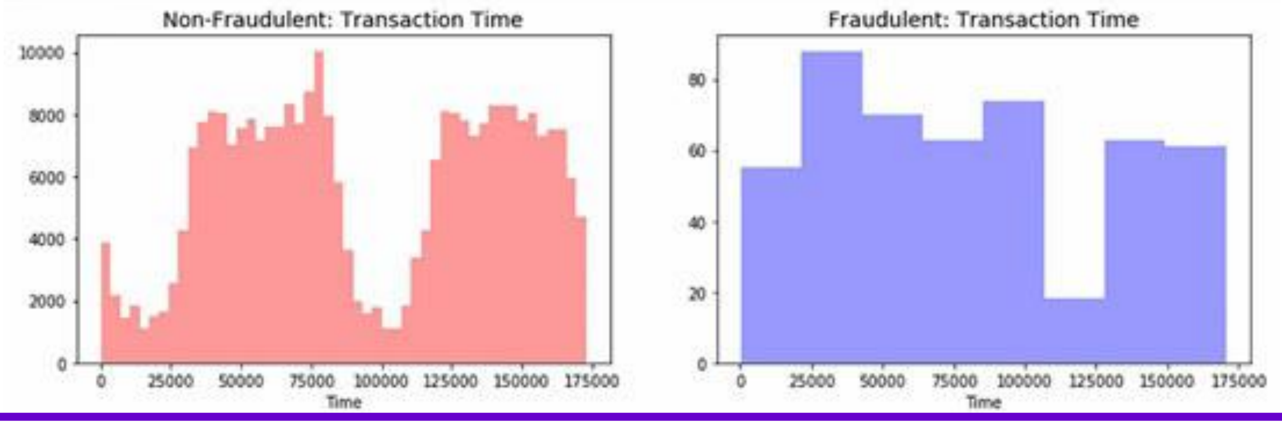
```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.kdeplot(n_df['Amount'], ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Amount(KDE)', fontsize=14)

sns.kdeplot(f_df['Amount'], ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Amount(KDE)', fontsize=14)

plt.show()
```



“Time” Frequency Distribution for Fraudulent and Non-fraudulent Class

We need to perform something similar for “Time” as well and see the distribution

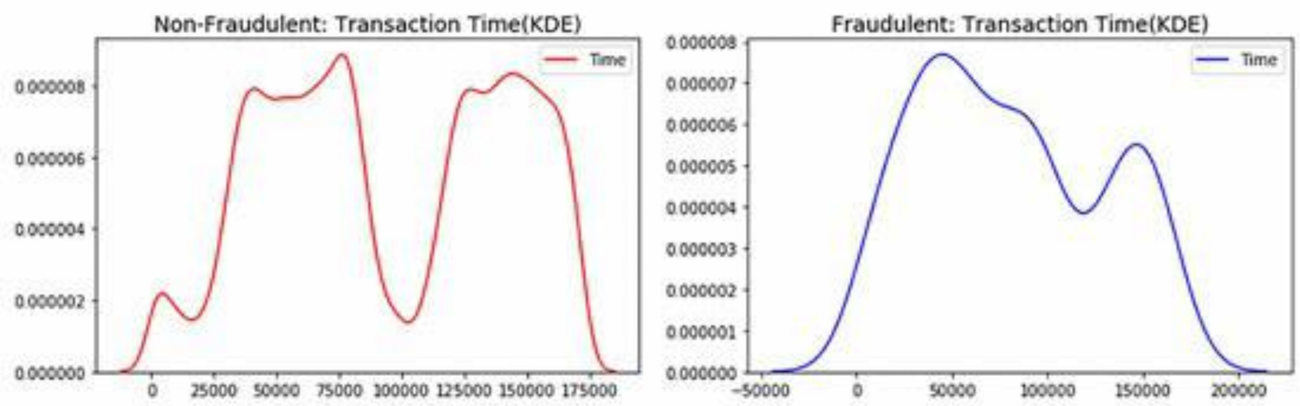
```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.distplot(n_df['Time'], kde=False, ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Time', fontsize=14)

sns.distplot(f_df['Time'], kde=False, ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Time', fontsize=14)

plt.show()
```



“Time” Frequency Distribution for Fraudulent and Non-fraudulent Class

Interestingly, the “Time” distribution for Fraudulent transaction is different. We can confirm and analyze after plotting the KDE.

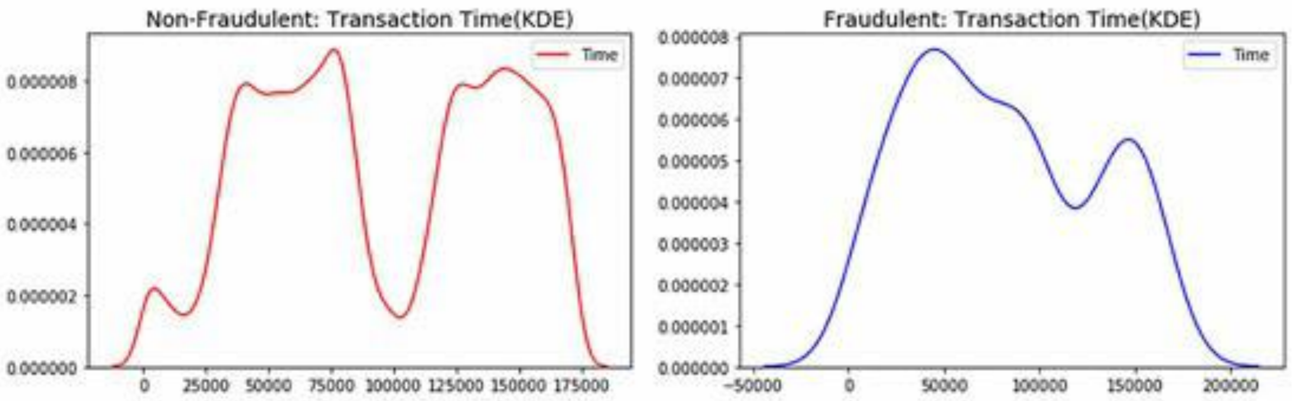
```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.kdeplot(n_df['Time'], ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Time(KDE)', fontsize=14)

sns.kdeplot(f_df['Time'], ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Time(KDE)', fontsize=14)

plt.show()
```



Interpreting Data and Make Conclusions

Interestingly, the “Time” distribution for Fraudulent transaction is different. We can confirm and analyze after plotting the KDE.

```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.kdeplot(n_df['Time'], ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Time(KDE)', fontsize=14)

sns.kdeplot(f_df['Time'], ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Time(KDE)', fontsize=14)

plt.show()
```



Interpreting Distributions and Make Conclusions

Non-fraudulent transaction for both “Time” and “Amount” looks the same when compared with the entire population. That is because the size of the Fraudulent dataset is so less the impact is not significant enough to change the distribution.

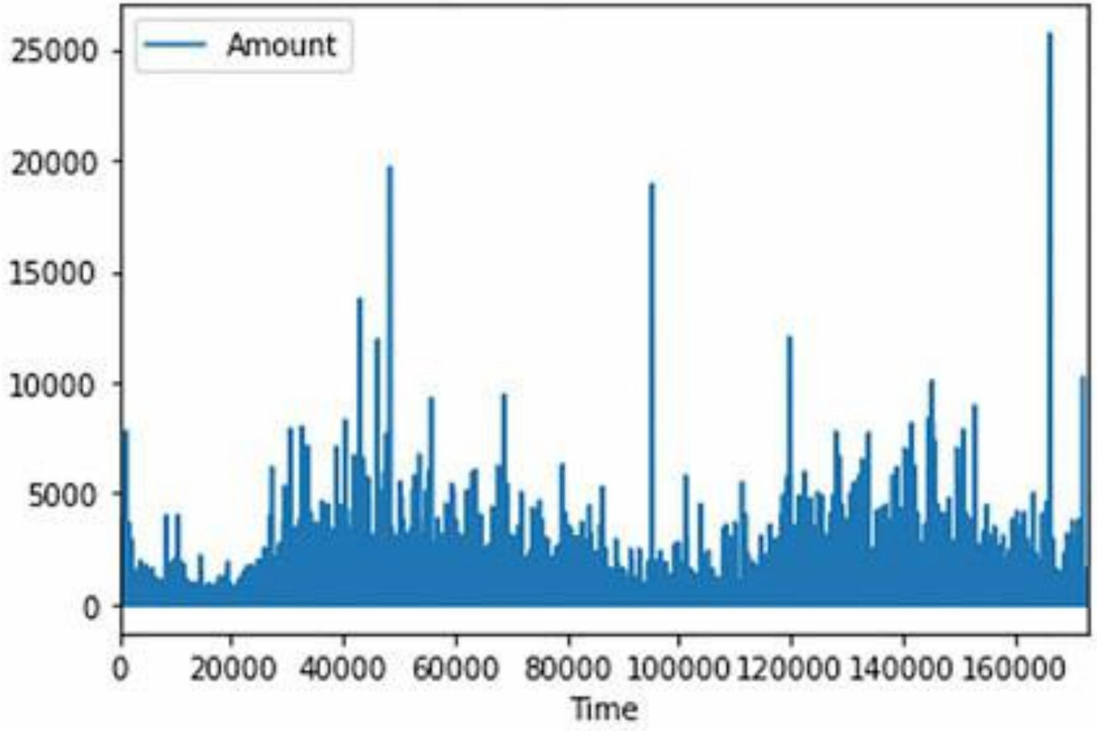
Fraudulent transaction for “Time” looks different, and that has some significance, i.e., the density of the distribution is higher at the beginning of time, and there is only one statistical model to the distribution. We need to keep this in mind, when we scale the data and make sure this kind of pattern is not lost.

Non-Fraudulent Summed "Amount" vs. "Time"

As with the previous plots for this class, this is highly similar to the plot for the entire population.

```
n_df[['Time','Amount']].groupby(['Time']).sum().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7131404890>
```

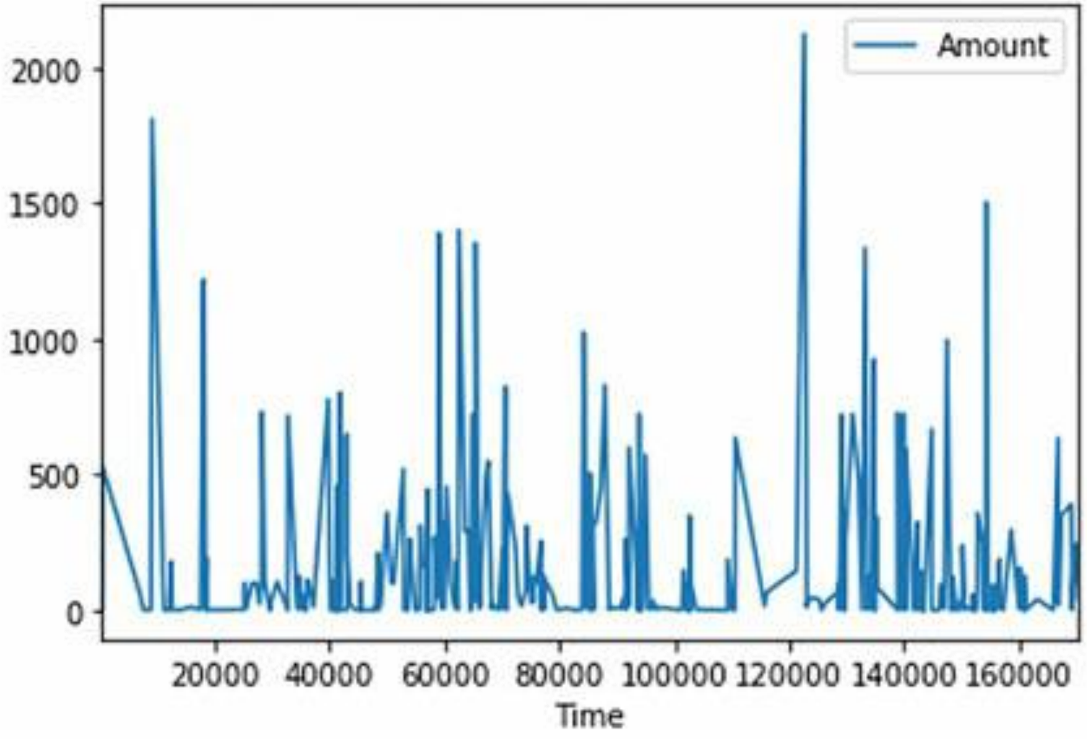


Non-Fraudulent Summed “Amount” vs. “Time”

As we thought, the plot will be different from the hypothesis, and we find the hypothesis to be true.

```
f_df[['Time','Amount']].groupby(['Time']).sum().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7131643250>
```



Total summed amount for different class

We can see the total amount for each group and removing the zero-amount transaction would not affect the "Amount" distribution, and it won't change the "Time" distribution as the size of the exception is very low.

```
print("Entire Dataset: " + str(df.Amount.sum()))
print("Non-Fraudulent Dataset: " + str(n_df.Amount.sum()))
print("Fraudulent Dataset: " + str(f_df.Amount.sum()))
```

```
Entire Dataset: 25162590.009999998
Non-Fraudulent Dataset: 25102462.04
Fraudulent Dataset: 60127.97
```

Removing Zero "Amount" Transaction

We can see the total amount for each group and removing the zero-amount transaction would not affect the "Amount" distribution, and it won't change the "Time" distribution as the size of the exception is very low.

```
df.drop(df[df.Amount == 0].index, inplace=True)
n_df = df[df["Class"]==0]
f_df = df[df["Class"]==1]
(df.shape, n_df.shape, f_df.shape)
```

```
((282982, 31), (282517, 31), (465, 31))
```

1. What is an Imbalanced Dataset?
2. Normalization
3. Scaling
4. Splitting Dataset
5. Handling Imbalance

Scaling, Standard Scaler

Now we need to make all the features same for the comparison.

Standard scalar from the sklearn implements standardization/Z-score normalization.

$$x' = \frac{x_i - \bar{x}}{\sigma}$$

Standard scaling the features

We know from the data description that all the features from V1…, V28 are scaled as those are the output result of a Dimension Reduction Algorithm, i.e., PCA. So, we will only implement standard scaling for “Time” and “Amount.”

```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()

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

visualize the change in the distribution

We need to visualize the change in the distribution compared with the original distribution so that we can decide whether to keep or discard it.

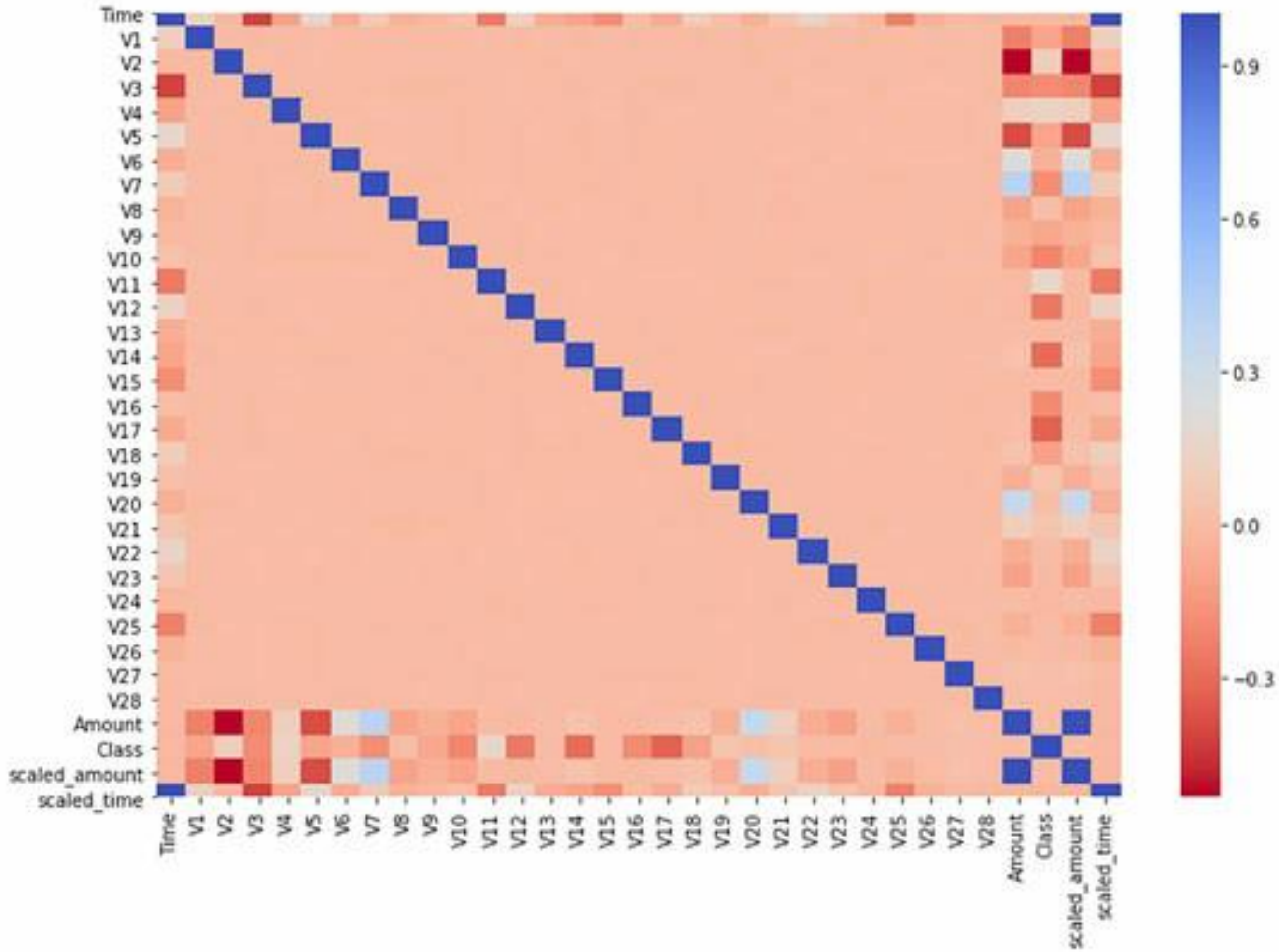
```
def compare_kde(col1, col2, name):
    fig, ax = plt.subplots(2, 2, figsize=(16,7))
    sns.kdeplot(df[df['Class']==0]['Amount'], ax=ax[0][0], color='r')
    ax[0][0].set_title('Non-Fraudulent: Transaction Amount(Original)', fontsize=12)
    sns.kdeplot(df[df['Class']==1]['Time'], ax=ax[0][1], color='b')
    ax[0][1].set_title('Fraudulent: Transaction Time(Original)', fontsize=12)
    sns.kdeplot(df[df['Class']==0][col1], ax=ax[1][0], color='r')
    ax[1][0].set_title('Non-Fraudulent: Transaction Amount('+ name +')', fontsize=12)
    sns.kdeplot(df[df['Class']==1][col2], ax=ax[1][1], color='b')
    ax[1][1].set_title('Fraudulent: Transaction Time(RobustScaler)', fontsize=12)
    plt.show()
```

Correlation heat map after scaling

Interestingly, after scaling, it did not make any visual change, and that is only expected because, “Amount” and “scaled_amount” features correlation equals to one.

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap='coolwarm_r', annot=False)
```

<matplotlib.axes._subplots.AxesSubplot at 0x129930750>



Robust Scaling

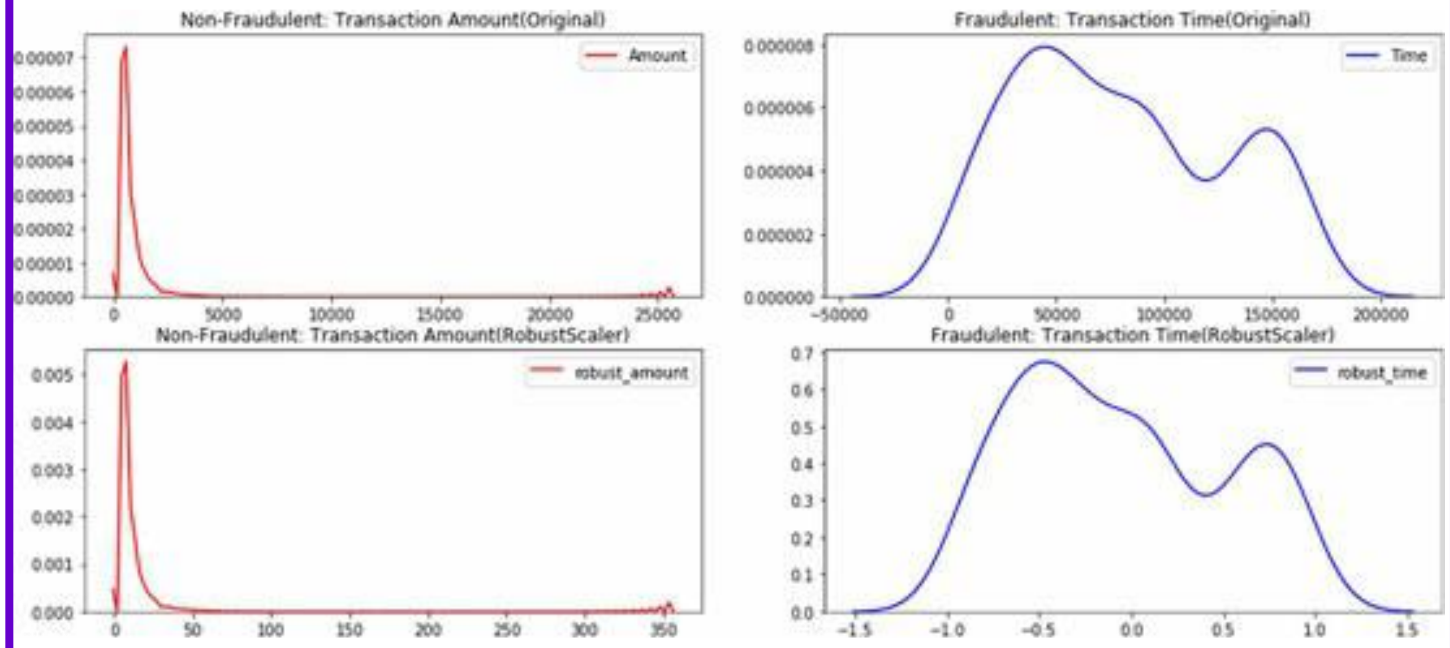
Robust scaling is similar to standard scaling, but it removes the median and scales the data points according to the IQR.

Robust scaling helps to handle the outliers better than standard scaling. We will also check the distribution and compare it with Standard Scalar.

```
from sklearn.preprocessing import RobustScaler

rob_scaler = RobustScaler()

df['robust_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['robust_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
```

Comparing Robust Scaling

It didn't change that distribution, but the range and the scale are changed. Compared with both the Original and Standard Scalar, it is quite similar



Power Transformer

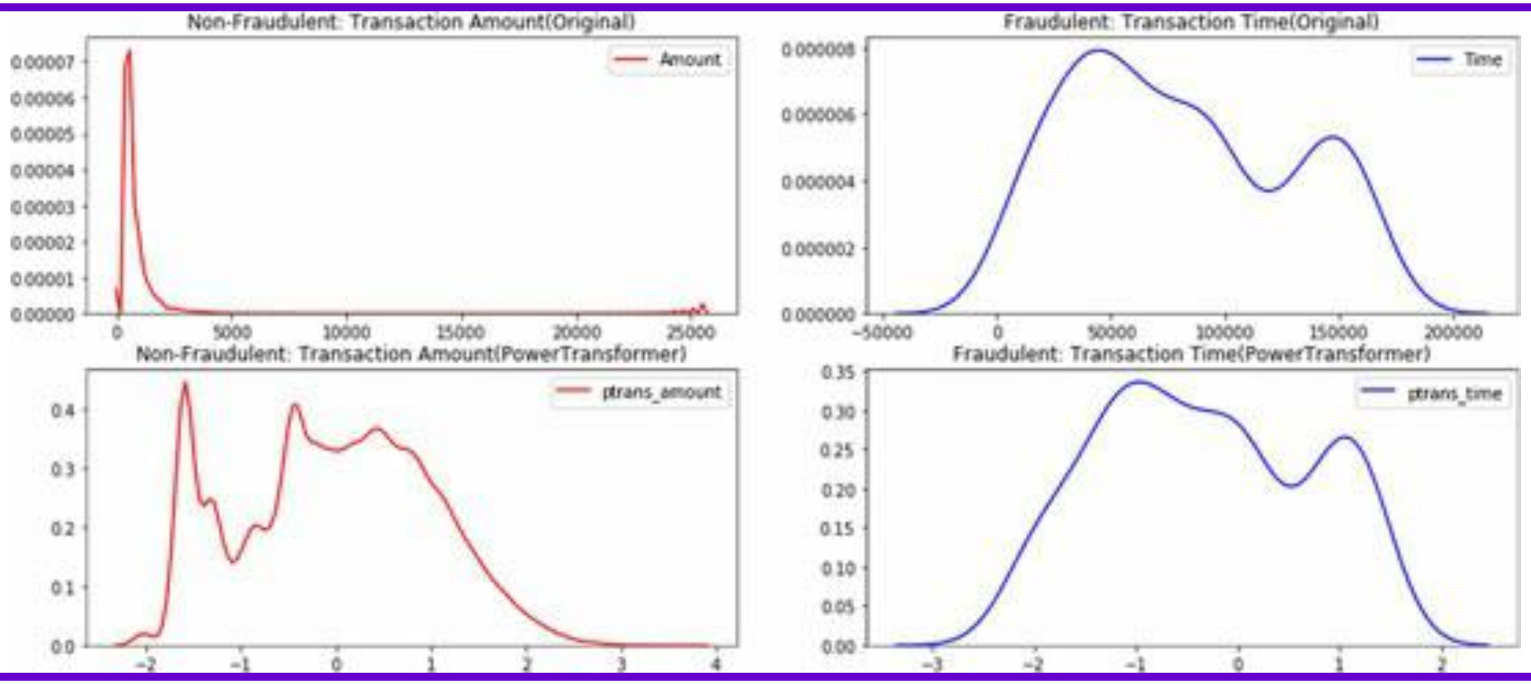
Power transformer belongs to a family of parametric, monotonic transformations that target to map/project data points from any distribution to as close to a Normal/Gaussian distribution.

This process helps to stabilize the variance of the data and minimize skewness. This method has the highest effects on skewed data.

```
from sklearn.preprocessing import PowerTransformer

p_trans = PowerTransformer()

df['ptrans_amount'] = p_trans.fit_transform(df['Amount'].values.reshape(-1,1))
df['ptrans_time'] = p_trans.fit_transform(df['Time'].values.reshape(-1,1))
```

Comparing power transformer

“Amount” feature had quite a lot of change, and it is not similar to a normal distribution but, when compared to “Time” distribution, the change is a lot



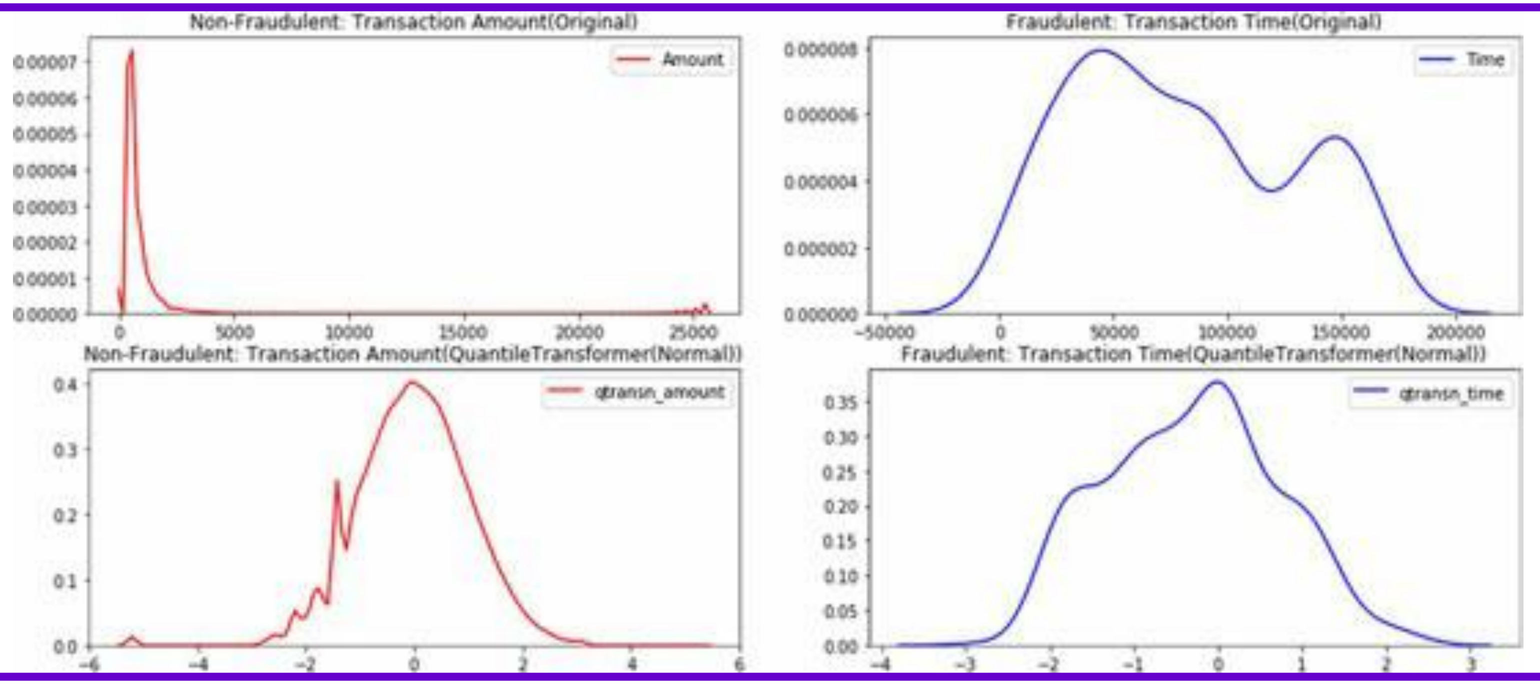
Quantile Transformer

Quantile transformer is a non-parametric method to transform the features such that it follows a uniform or a normal distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers: this is, therefore, a robust pre-processing scheme.

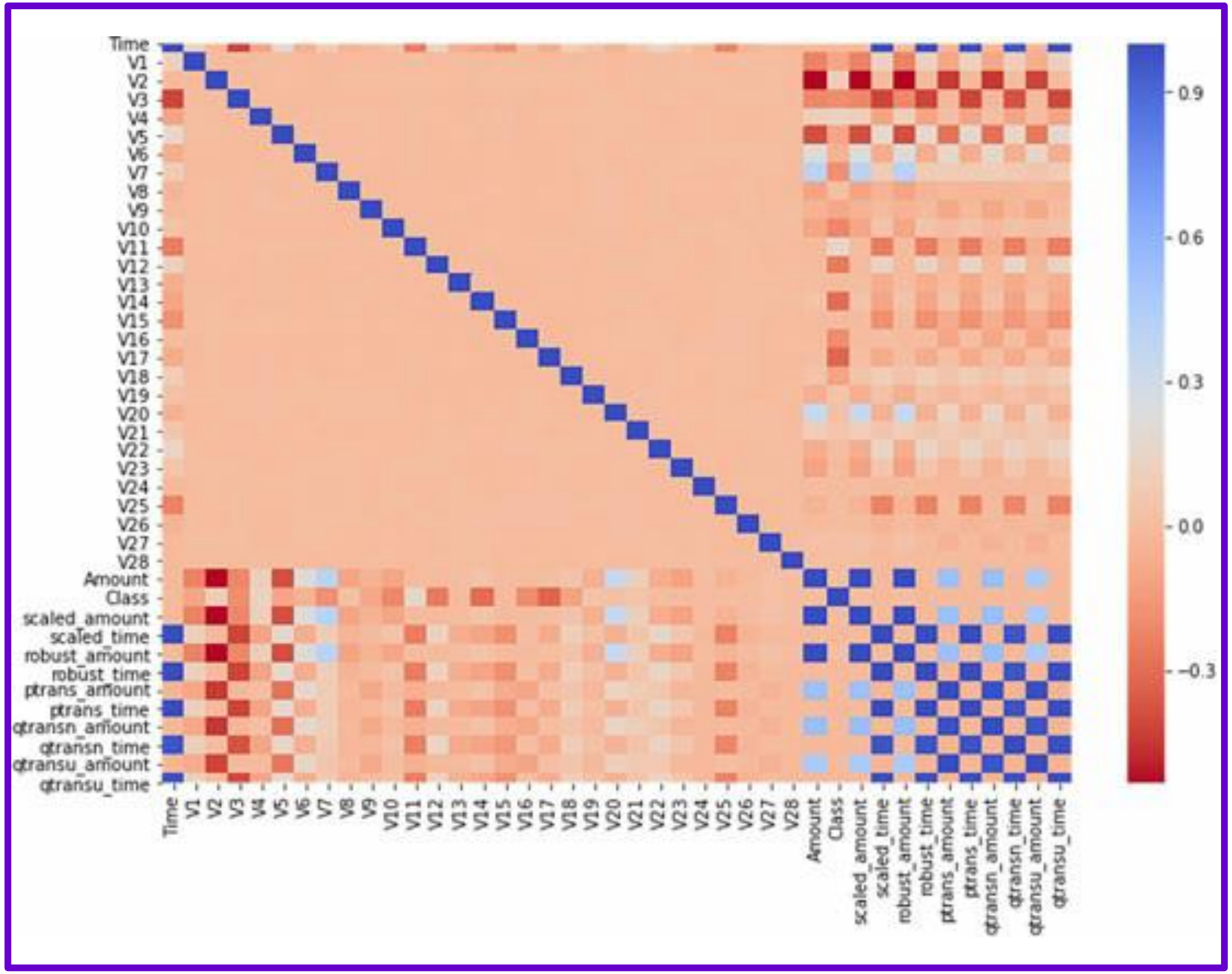
```
from sklearn.preprocessing import QuantileTransformer

q_trans = QuantileTransformer(output_distribution="normal")

df['qtransn_amount'] = q_trans.fit_transform(df['Amount'].values.reshape(-1,1))
df['qtransn_time'] = q_trans.fit_transform(df['Time'].values.reshape(-1,1))

q_trans = QuantileTransformer(output_distribution="uniform")

df['qtransu_amount'] = q_trans.fit_transform(df['Amount'].values.reshape(-1,1))
df['qtransu_time'] = q_trans.fit_transform(df['Time'].values.reshape(-1,1))
```

Comparing Quantile Transformer -Uniform

We can see that “Amount” is now a uniform distribution, and “Time” is close to a uniform distribution.



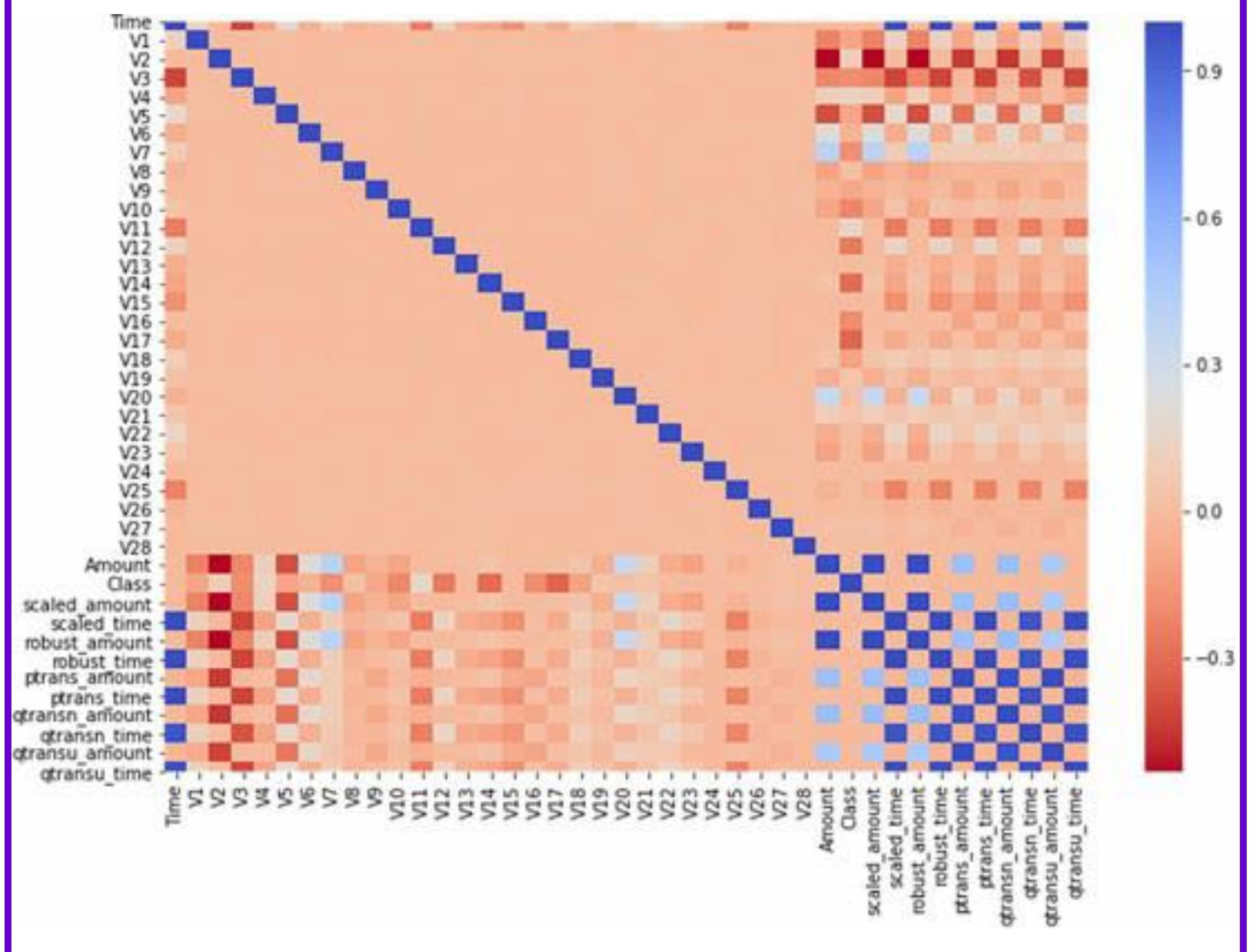
Correlation heat map after applying all scaling algorithms



Now, we need to analyze this correlation matrix and figure out the available scaled feature.

Correlation heat map after applying all scaling algorithms

We have one observation that none of the dimensionally reduced features changed its correlation with respect to “Time” and “Amount.”



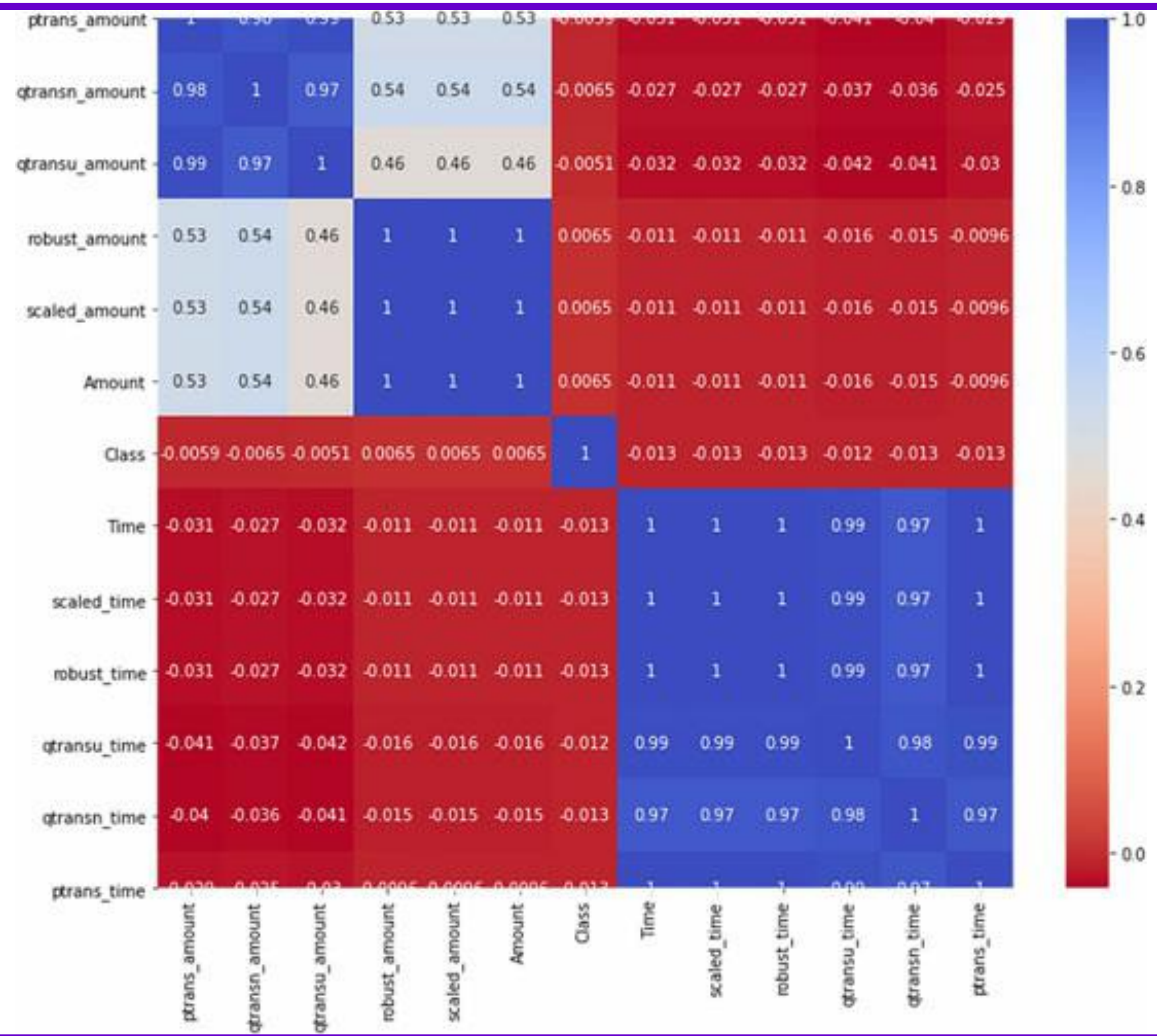
Code for plotting only scaled features

Analyzing above correlation heat map is very tough as we are not taking V1…, V28 into consideration as of now

```
plt.figure(figsize=(12, 10))
sns.heatmap(df[["ptrans_amount","qtransn_amount","qtransu_amount",
                "robust_amount","scaled_amount","Amount",
                "Class", "Time", 'scaled_time',"robust_time",
                "qtransu_time", "qtransn_time","ptrans_time"]].corr(),
           cmap='coolwarm_r', annot=True)
```

Correlation for all scaled features

Analyzing above correlation heat map is very tough as we are not taking V1..., V28 into consideration as of now. So, it is better to plot all the scaled amounts and time separately to analyze them.



Description of scaled "Amount."

We need to compare "Time," "Class," and "Amount" and see an interesting result that has nearly zero change in correlation concerning "Class."

```
df[["Amount", 'scaled_amount',"robust_amount",
    "qtransu_amount", "qtransn_amount","ptrans_amount"]].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Amount	282982.0	8.891940e+01	250.824374	0.010000	5.990000	22.490000	78.000000	25691.160000
scaled_amount	282982.0	3.586570e-15	1.000002	-0.354469	-0.330628	-0.264845	-0.043534	102.072560
robust_amount	282982.0	9.225024e-01	3.483188	-0.312179	-0.229135	0.000000	0.770865	356.459797
qtransu_amount	282982.0	5.001364e-01	0.288647	0.000000	0.250250	0.501011	0.748822	1.000000
qtransn_amount	282982.0	-4.655992e-03	1.019305	-5.199338	-0.673702	0.001756	0.671178	5.199338
ptrans_amount	282982.0	2.142510e-14	1.000002	-2.050775	-0.733013	0.030784	0.750683	3.649005

Description of scaled "Time."

"Time" also reflects the same characteristics as "Amount," but the change is drastic for some algorithm for the future "Amount."

```
df[["Time", 'scaled_time',"robust_time",
    "qtransu_time", "qtransn_time","ptrans_time"]].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Time	282982.0	9.484896e+04	47482.459589	0.000000	54251.250000	84707.500000	139363.750000	172792.000000
scaled_time	282982.0	6.155099e-15	1.000002	-1.997561	-0.855006	-0.213584	0.937501	1.641515
robust_time	282982.0	1.191536e-01	0.557879	-0.995242	-0.357835	0.000000	0.642165	1.034918
qtransu_time	282982.0	4.995409e-01	0.288562	0.000000	0.248568	0.499854	0.749264	1.000000
qtransn_time	282982.0	3.381409e-03	1.000297	-5.199338	-0.671536	0.005473	0.679379	5.199338
ptrans_time	282982.0	1.981191e-14	1.000002	-2.436283	-0.809483	-0.143207	0.928790	1.534947

Being an ML practitioner

Like we have seen, there is not much information we found out from this scaling section

Being an ML practitioner, we need to do the same thing again and again, and every work depends on the experience.

but, when we will go ahead with the next section, i.e., handling the imbalanced data, we will see a huge improvement with the results from scaling too

1. What is an Imbalanced Dataset?
2. Normalization
3. Scaling
4. Splitting Dataset
5. Handling Imbalance

Splitting Dataset

We need to split the data for many reasons. One of the major reasons is for testing the model

```
from sklearn.model_selection import train_test_split
import numpy as np

print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

X = df.drop('Class', axis=1)
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(y_train, return_counts=True)
test_unique_label, test_counts_label = np.unique(y_test, return_counts=True)

print('\nLabel Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
print("\nTrain:")
print('No Frauds', round(len(y_train[y_train==0])/len(X_train) * 100,2), '% of the dataset')
print('Frauds', round(len(y_train[y_train==1])/len(X_train) * 100,2), '% of the dataset')
print("\nTest:")
print('No Frauds', round(len(y_test[y_test==0])/len(X_test) * 100,2), '% of the dataset')
print('Frauds', round(len(y_test[y_test==1])/len(X_test) * 100,2), '% of the dataset')
```

Split Data Statistics (Traditional)

We need to make the fraud percentage the same or similar to the original fraud percentage. As we are dealing with an extremely small percentage of fraud data, our target should be the same.

```
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset

Label Distributions:

[0.99829936 0.00169622]
[0.99858647 0.0014312 ]

Train:
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset

Test:
No Frauds 99.86 % of the dataset
Frauds 0.14 % of the dataset
```

Split Data Statistics (StratifiedKFold)

There is a solution to this problem that is stratified K-fold, it is a similar concept like a cross validation K-fold technique. It gives us the same distribution as the original distribution

```python
from sklearn.model_selection import StratifiedKFold
import numpy as np

print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

X = df.drop('Class', axis=1)
y = df['Class']

skf = StratifiedKFold(n_splits=10, random_state=None, shuffle=False)

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(y_train, return_counts=True)
test_unique_label, test_counts_label = np.unique(y_test, return_counts=True)

print('\nLabel Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
print("\nTrain:")
print('No Frauds', round(len(y_train[y_train==0])/len(X_train) * 100,2), '% of the dataset')
print('Frauds', round(len(y_train[y_train==1])/len(X_train) * 100,2), '% of the dataset')
print("\nTest:")
print('No Frauds', round(len(y_test[y_test==0])/len(X_test) * 100,2), '% of the dataset')
print('Frauds', round(len(y_test[y_test==1])/len(X_test) * 100,2), '% of the dataset')
```

Splitting Dataset (StratifiedKFold)

From the above code snippet, we are expecting the training and testing dataset with the class distribution like the original one

```
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset

Label Distributions:

[1.12315249 0.00185082]
[0.49916955 0.00081278]

Train:
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset

Test:
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset
```

Splitting Dataset

We got the expected output, i.e., the percentage of the class distribution is the same for the train, test, and the original dataset. Now, we can go ahead with handling imbalance characteristic of the dataset.

1. What is an Imbalanced Dataset?
2. Normalization
3. Scaling
4. Splitting Dataset
5. Handling Imbalance

Handling Imbalance: Oversampling

In oversampling, we will generally create synthetic data points for the class, which has low counts.

So, for oversampling, we have to use the test data from the original data, because after performing oversampling, if we split the data that won’t be correct, and we will get a wrong result.

Handling Imbalance: Under-sampling

for under-sampling, we can use the original dataframe to make a sub-sample out of it.

But, in this case, we will only use the train test dataset.

What is a sub-sample?

Sub-sample just means a part of the original dataset, but in this scenario, the subsample will be 50-50.

We will take the entire Fraudulent class and under-sample the non-fraudulent class so that the ratio is 50-50.

Advantages of sub-subsample

Overfitting is a common problem when there is an imbalance because it assumes that, in most cases, there are non-fraudulent transactions. But for a subsample, this problem will be less prone to overfitting.

Correlation with the class will improve drastically as our subsample will have no imbalance problem. Although we don't know what the "V" features stand for (but, my assumption is "Vector"), it will be useful to understand how each of these features influences the result with the class (Fraud or No Fraud) by having an imbalance dataframe we are not able to see the true correlations between the class and features

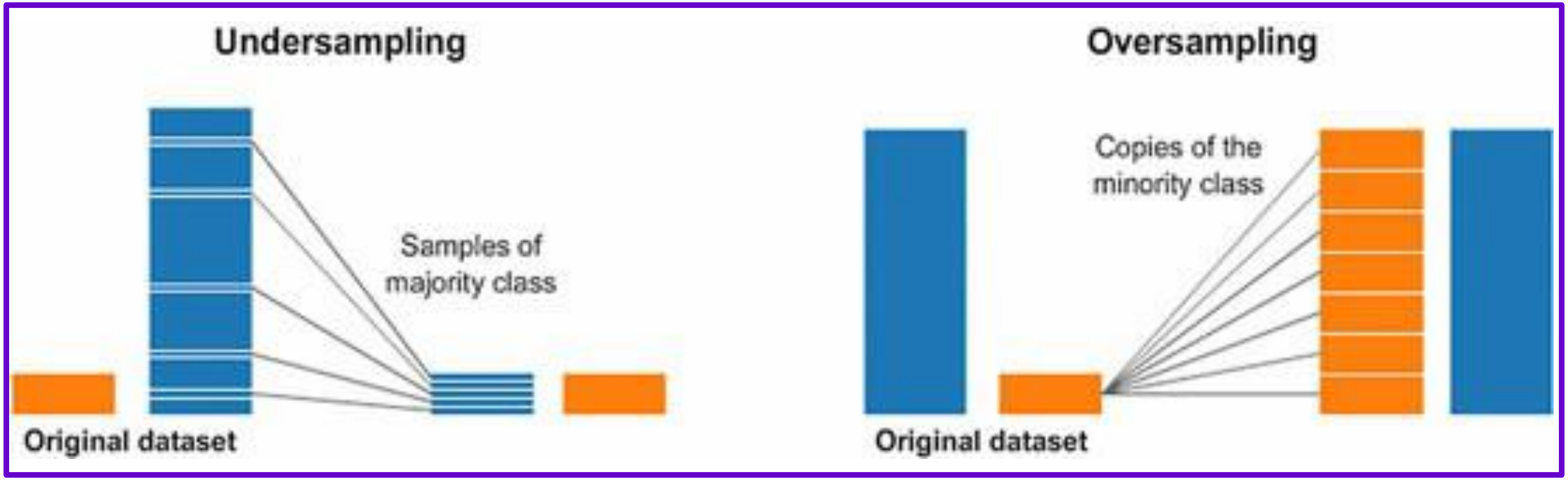
What is Resampling

Overfitting is a common problem when there is an imbalance because it assumes that, in most cases, there are non-fraudulent transactions. But for a subsample, this problem will be less prone to overfitting.

Correlation with the class will improve drastically as our subsample will have no imbalance problem. Although we don’t know what the “V” features stand for (but, my assumption is “Vector”), it will be useful to understand how each of these features influences the result with the class (Fraud or No Fraud) by having an imbalance dataframe we are not able to see the true correlations between the class and features

What is Resampling

Resampling is a technique to handle imbalance data by creating a sub-sampled dataset from the original data.



Resampling Train Datas

We need to make sure that we are using only the train dataset and not using the test dataset at all. The test dataset will only be used to validate the model and see how good it is.

```
train_df = X_train.copy()
train_df['Class'] = y_train
train_df.shape
```

```
(254685, 41)
```

Train Dataframe Class Distribution

Within that training dataframe, there will be a mixture of fraud and non-fraud datasets in the ratio of the Original dataset as we have used the Stratified K-fold technique

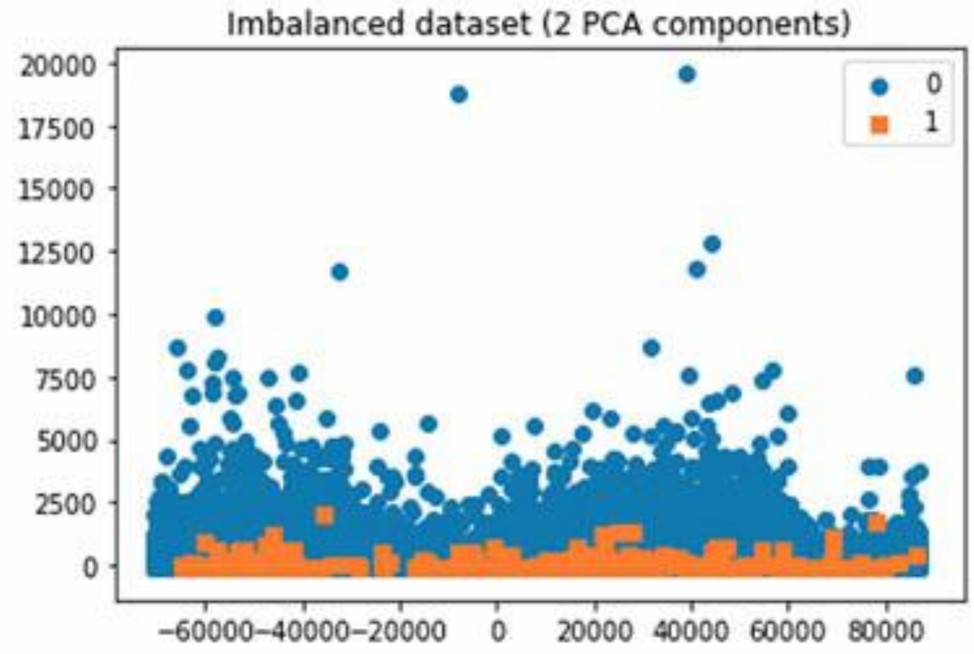
```
train_df['Class'].value_counts()
```

```
0    254266
1       419
Name: Class, dtype: int64
```

PCA plot for Original dataframe

Before going ahead, we need to find a way to visualize the 41 column features. We can use a dimensionality reduction algorithm like Principal Component Analysis to reduce to 2 principal components and plot in a 2D plot

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(X_train)

plot_2d_space(X, y_train, 'Imbalanced dataset (2 PCA components)')
```



Function to plot 2 component PCA

As we will plot a 2D graph, again and again, we are using a function for that. Then a simple function call is sufficient to plot it

```
def plot_2d_space(X, y, label='Class'):
    colors = ['#1F77B4', '#FF7F0E']
    markers = ['o', 's']
    for l, c, m in zip(np.unique(y), colors, markers):
        plt.scatter(
            X[y==l, 0],
            X[y==l, 1],
            c=c, label=l, marker=m
        )
    plt.title(label)
    plt.legend(loc='upper right')
    plt.show()
```

Random Undersampling

Random under-sampling will remove the majority class data points such that it is equivalent/equal/proportional to the minority class

```
# Class count
count_class_0, count_class_1 = train_df.Class.value_counts()

# Divide by class
train_df_0 = train_df[train_df['Class'] == 0]
train_df_1 = train_df[train_df['Class'] == 1]
```

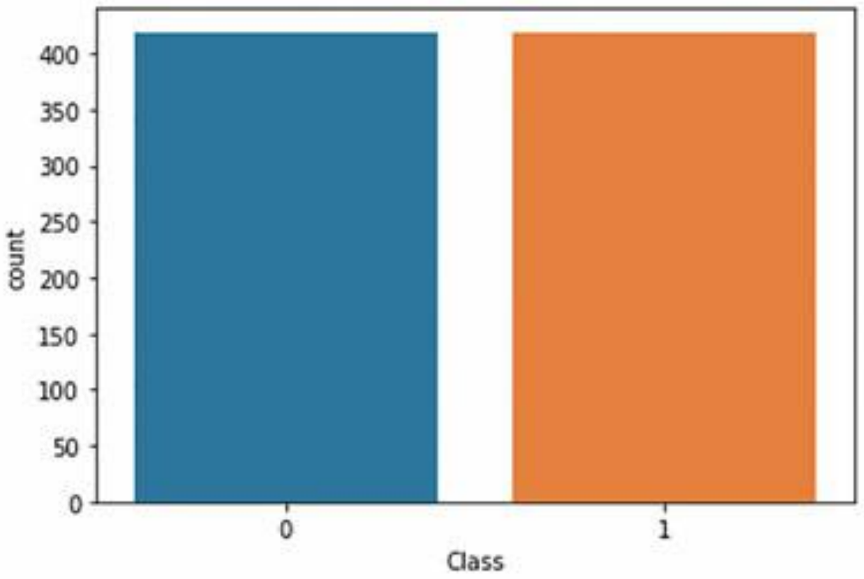
Random undersampling

Now, this dataset is equidistributed, and it is likely to give a better correlation and model results, as we have hypothesized before.

can arise multiple problems due to information loss

```
train_df_0_under = train_df_0.sample(count_class_1)
train_df_under = pd.concat([train_df_0_under, train_df_1], axis=0)

print('Random under-sampling:')
print(train_df_under.Class.value_counts())

sns.countplot('Class', data=train_df_under)
```

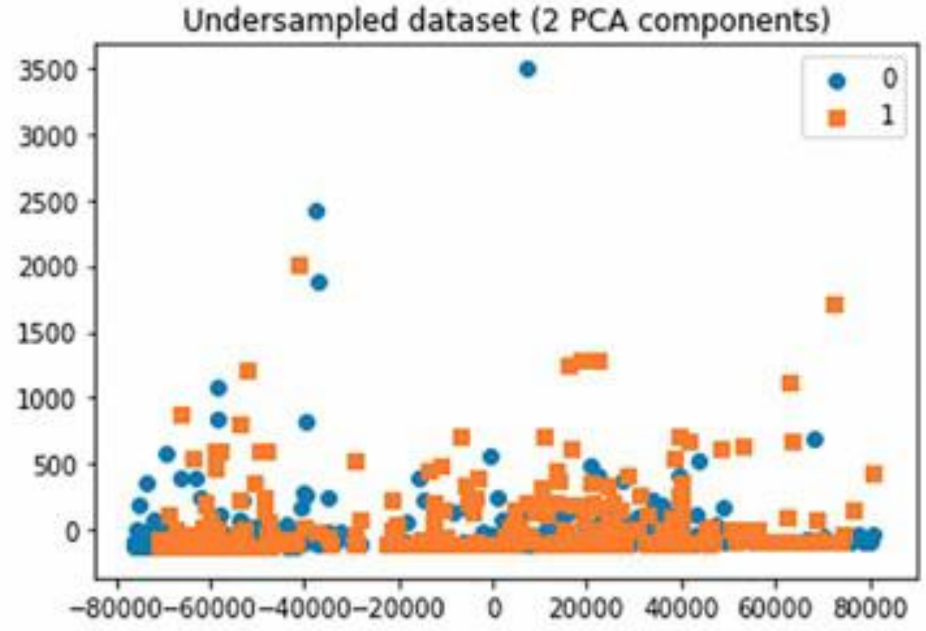
```
Random under-sampling:
1    419
0    419
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a31c29fd0>
```



PCA plot for random undersampling

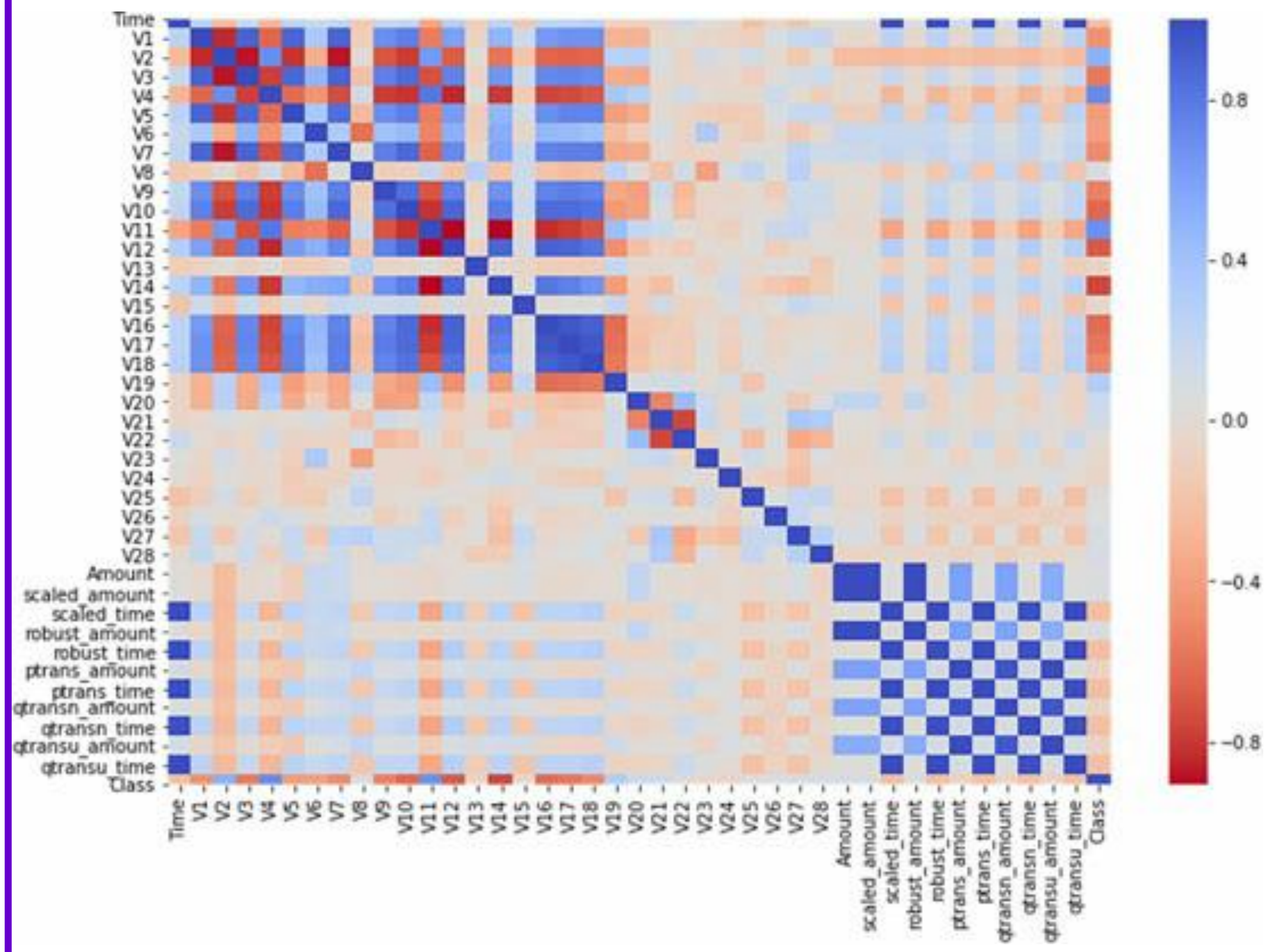
we can see the homogeneity in the data points of different classes. With this new homogeneous dataset, we are expecting a good correlation between all the features and primarily with the feature “class.”

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_under.drop(columns=["Class"]))

plot_2d_space(X, train_df_under["Class"], 'Undersampled dataset (2 PCA components)')
```



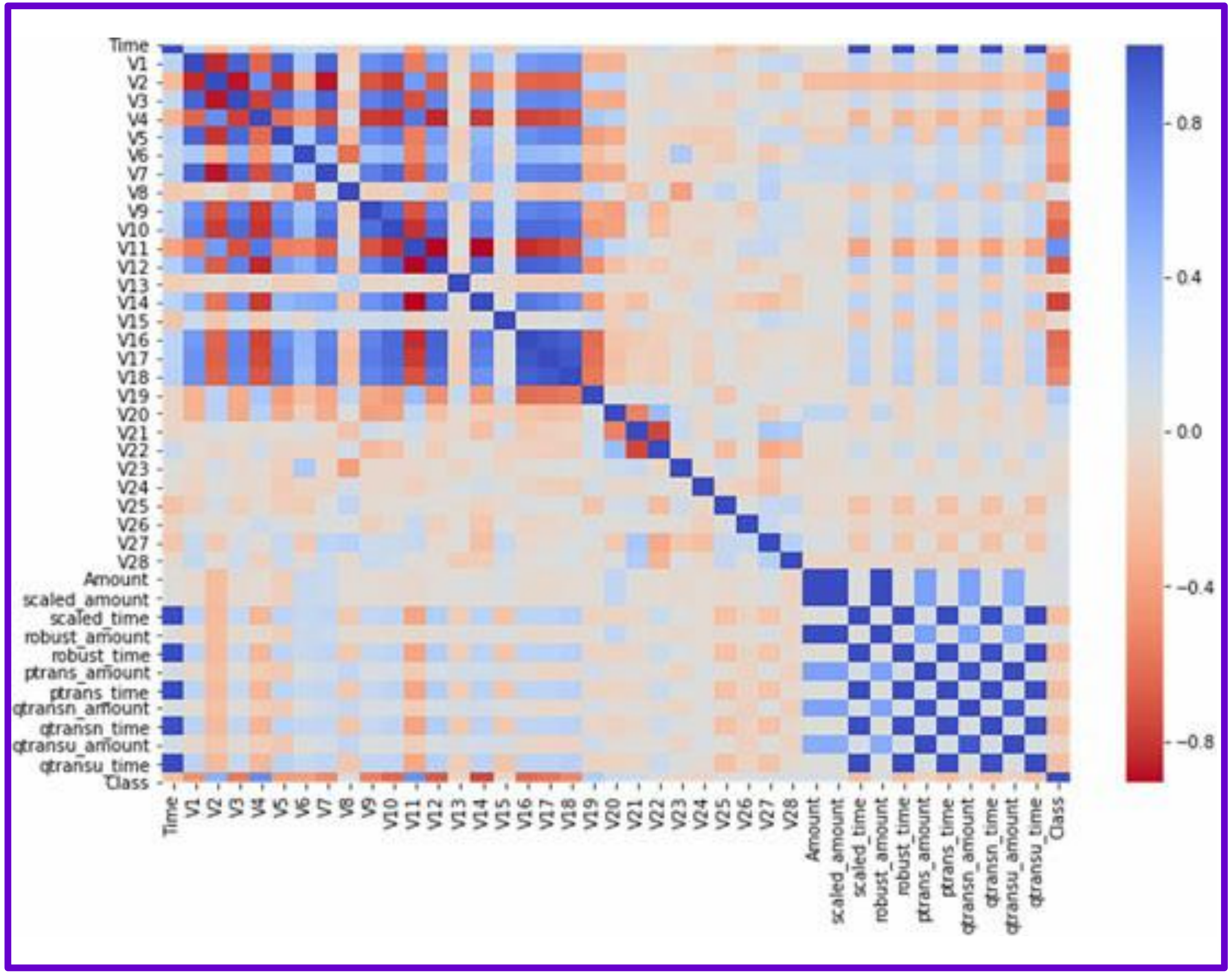
Correlation for Random Under Sampled data

We can see a drastic change with the correlation matrix for mostly all the features. Some features are now positively and negatively correlated



Correlation for Random Under Sampled data…

There is one more interesting observation that after random under-sampling, all the scaled features and their respective original features have the same or similar correlation

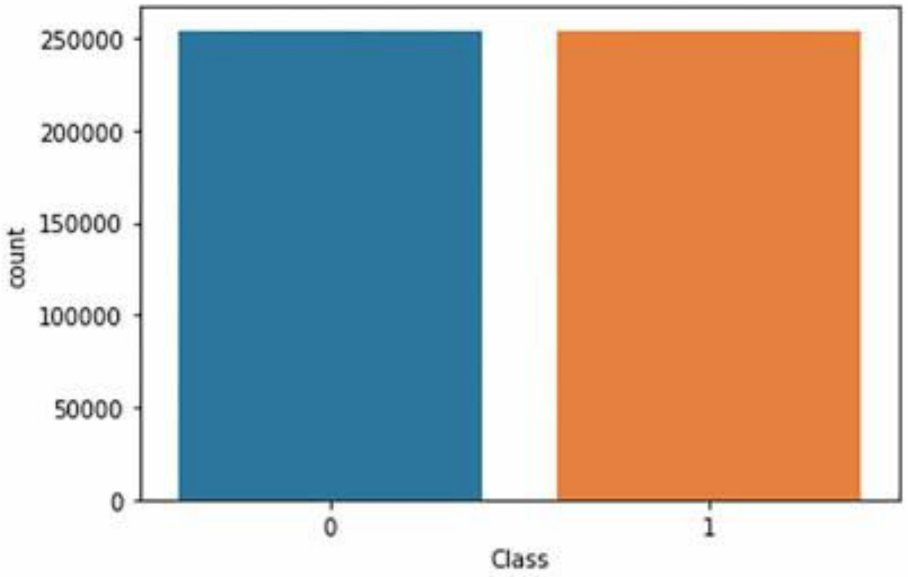


Random over-sampling

Similar to random under-sampling, we have random over-sampling where we will oversample the minor class, i.e., in this case, it will be the fraudulent transaction dataset.

```
train_df_1_over = train_df_1.sample(count_class_0, replace=True)
train_df_over = pd.concat([train_df_0, train_df_1_over], axis=0)

print('Random over-sampling:')
print(train_df_over.Class.value_counts())

sns.countplot('Class', data=train_df_over)
```

```
Random over-sampling:
1    254266
0    254266
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a336eee90>
```

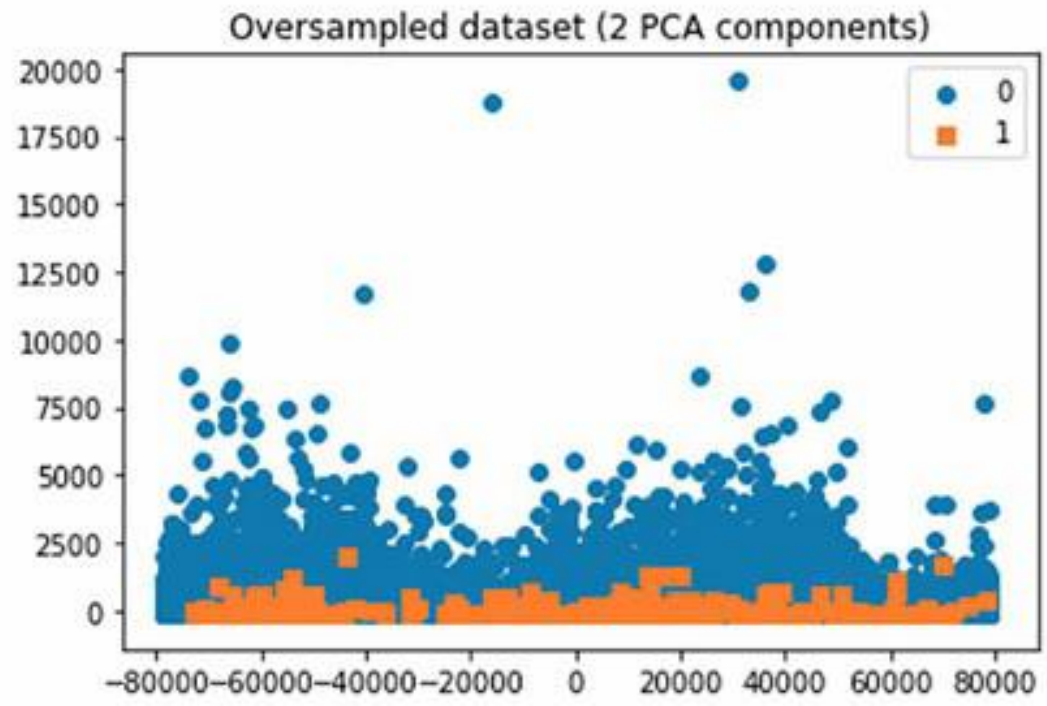


PCA plot for random over-sampling

The Oversampled data set shows the same plot as the original dataset plot.

So that it means the correlation heat map will be similar to the original one?? No

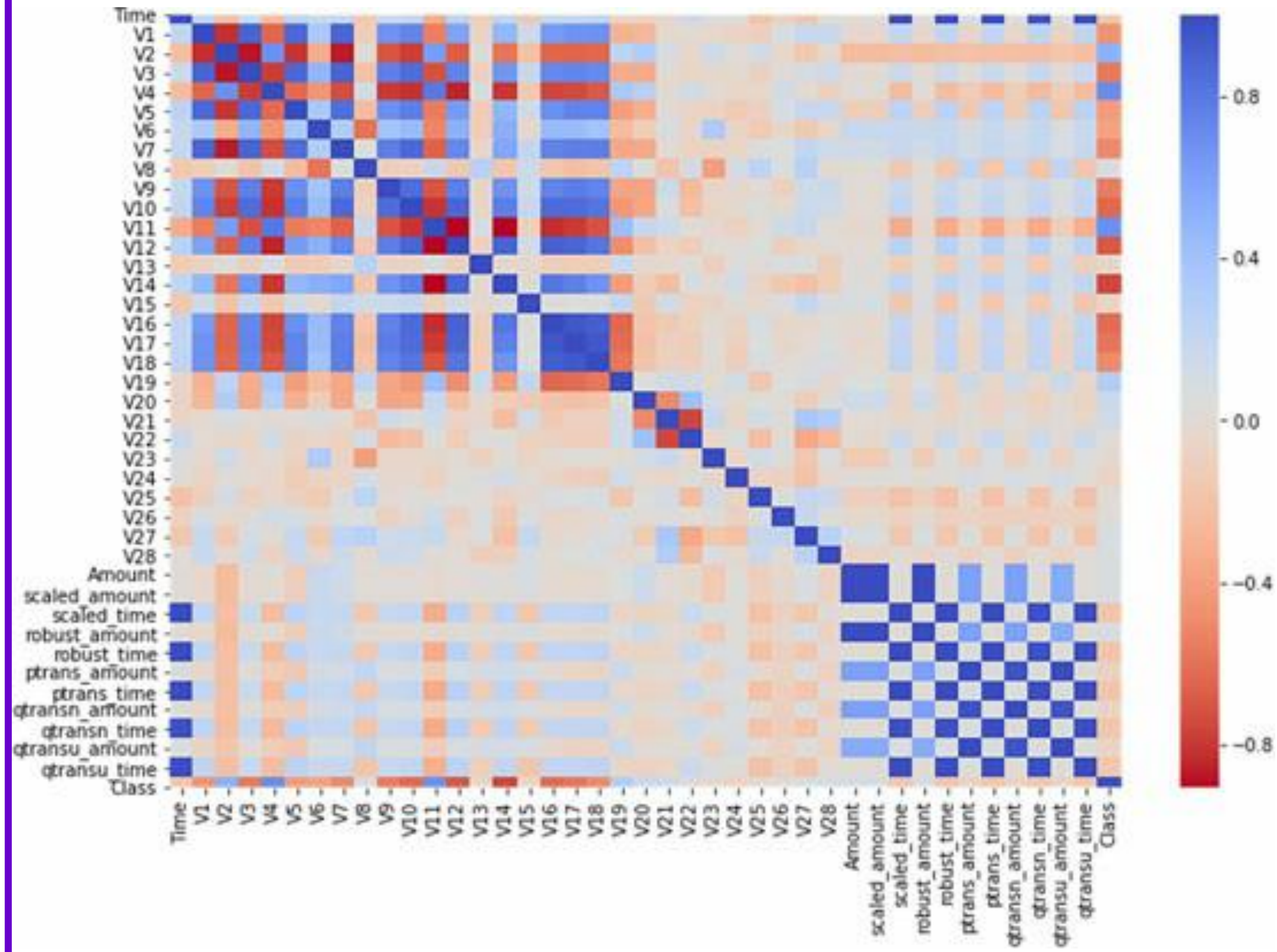
```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_over.drop(columns=["Class"]))

plot_2d_space(X, train_df_over["Class"], 'Oversampled dataset (2 PCA components)')
```



PCA plot for random over-sampling

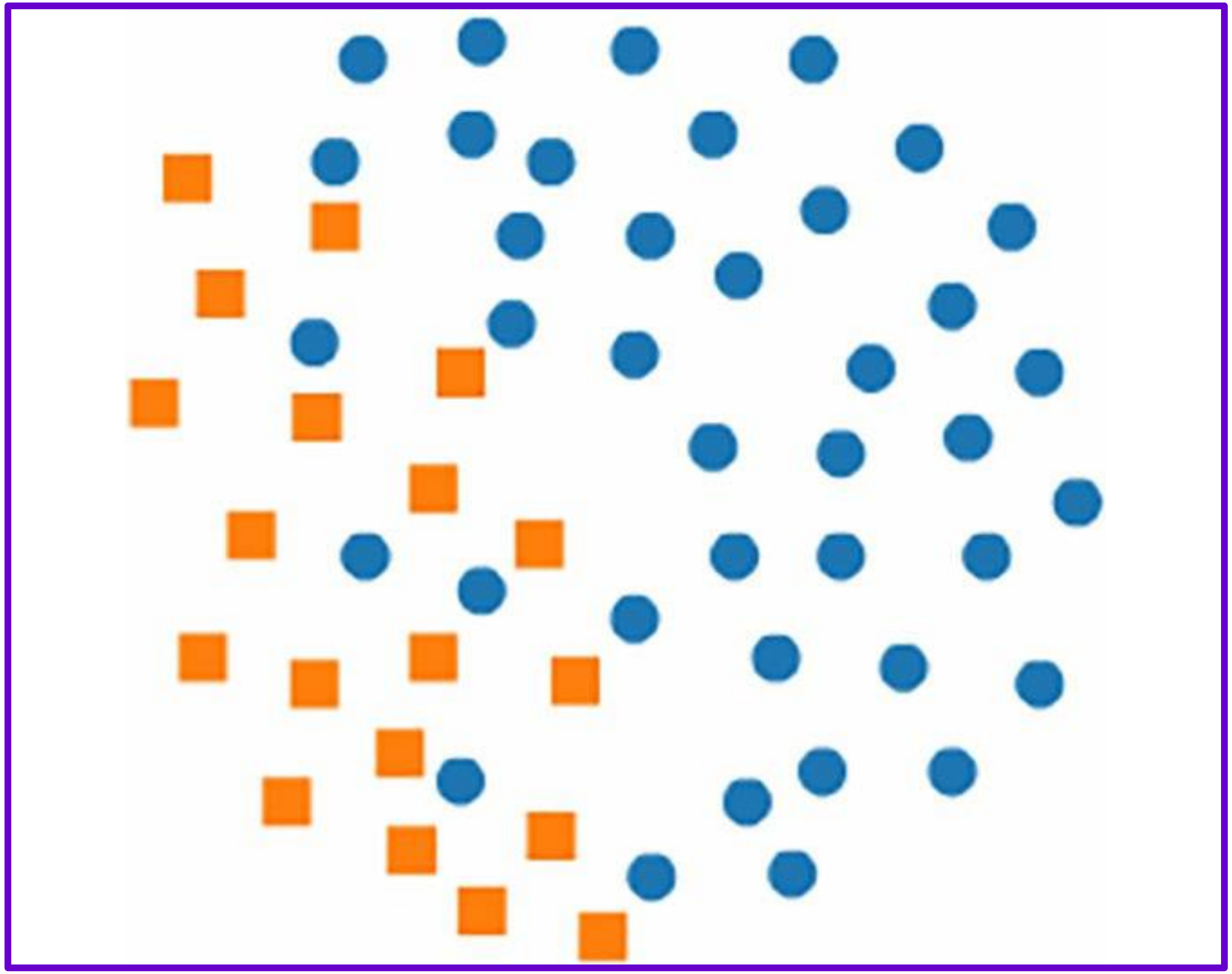
So that it means the correlation heat map will be similar to the original one??

But unfortunately, no, it will be extremely different because this oversampled dataset is equidistributed so it will have a better correlation and will be similar to the randomly under-sampled data set.



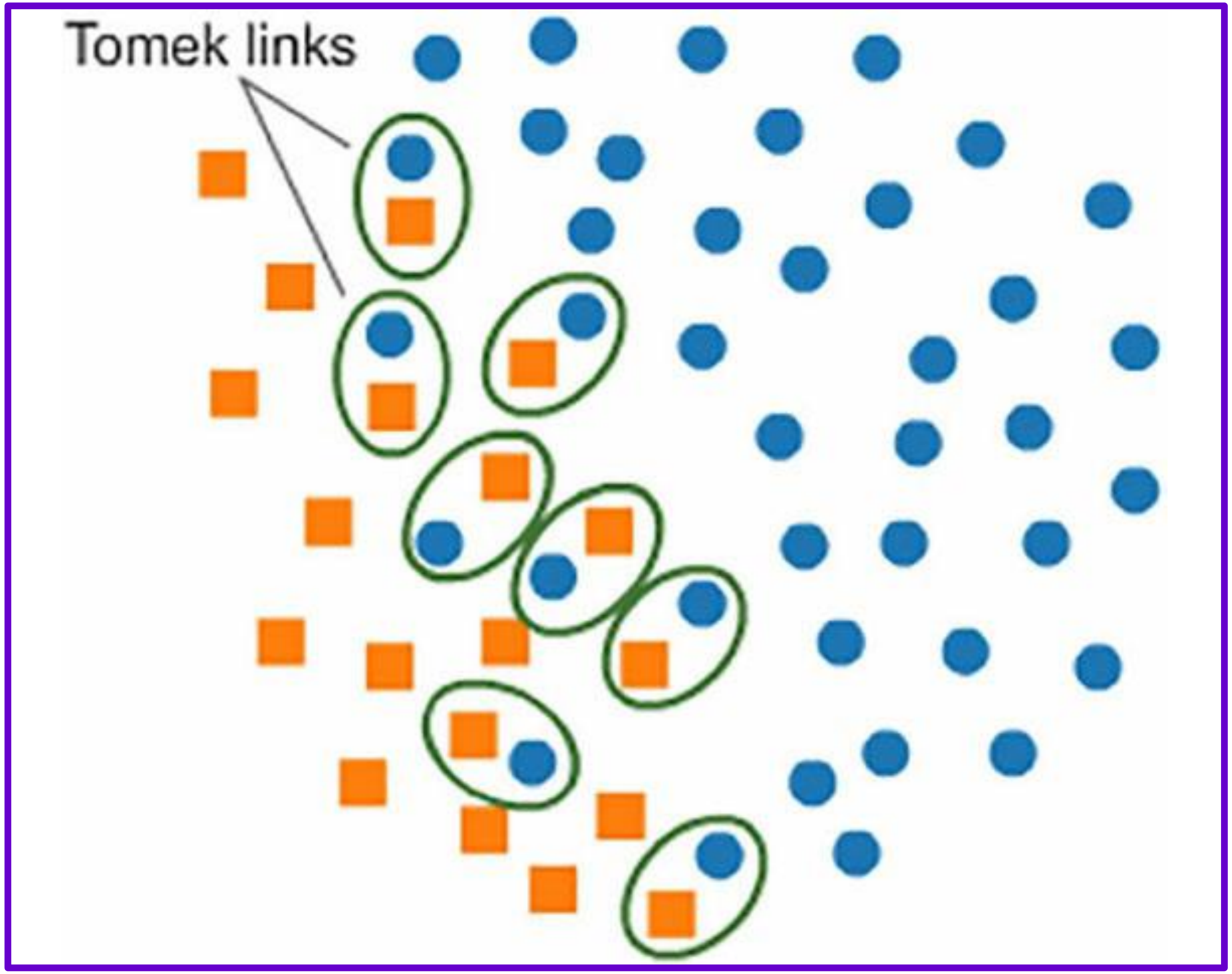
Tomek's Links for Undersampling: Step 1

The data points are plotted in n-dimensions(n-features) in this case for simplicity, we are using 2-axis and differentiate them with respect to class (here, there are 2 classes only).



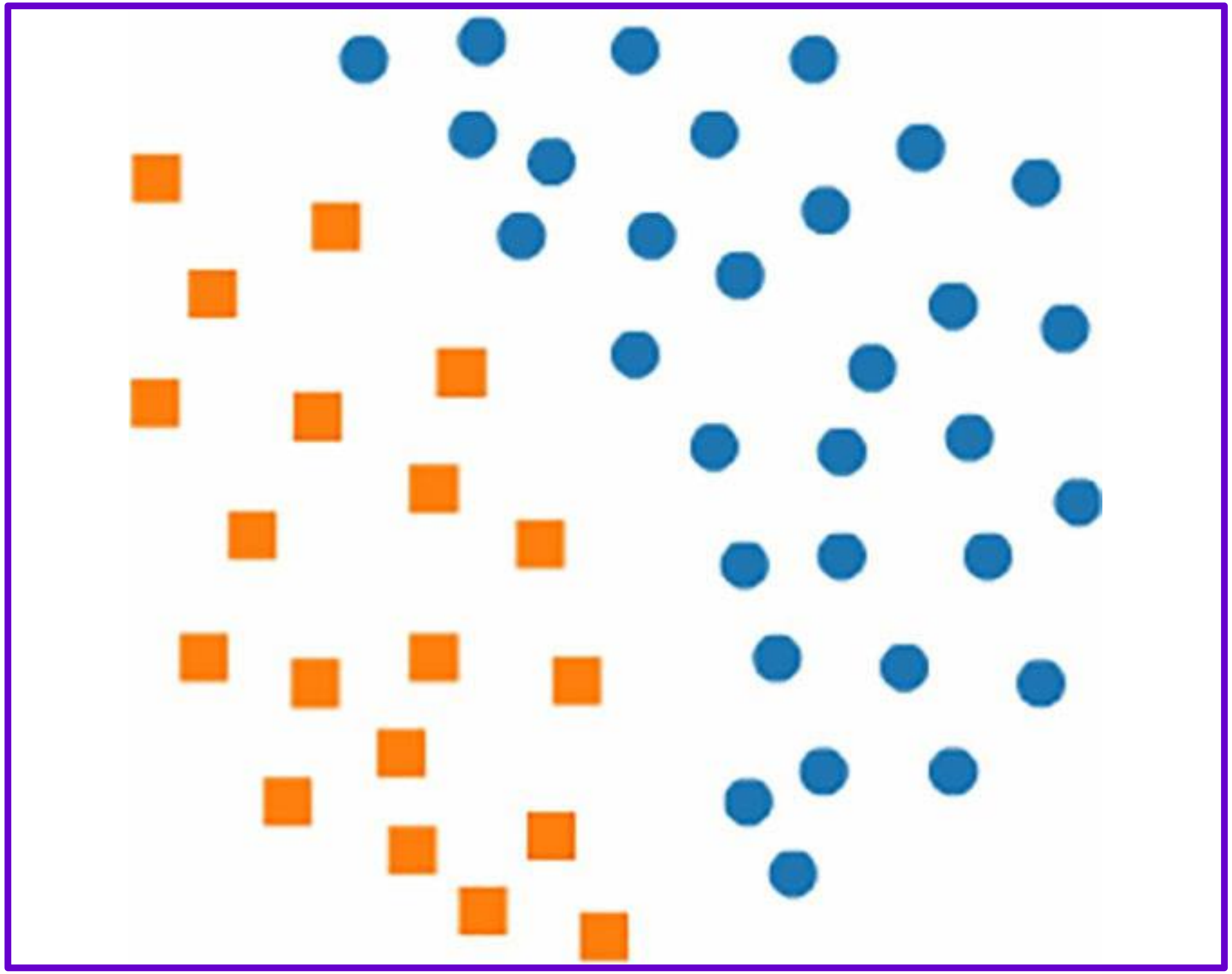
Tomek's Links for Undersampling: Step 2

We find the nearest neighbor where k=1 where the minor and the major classes are coupled.



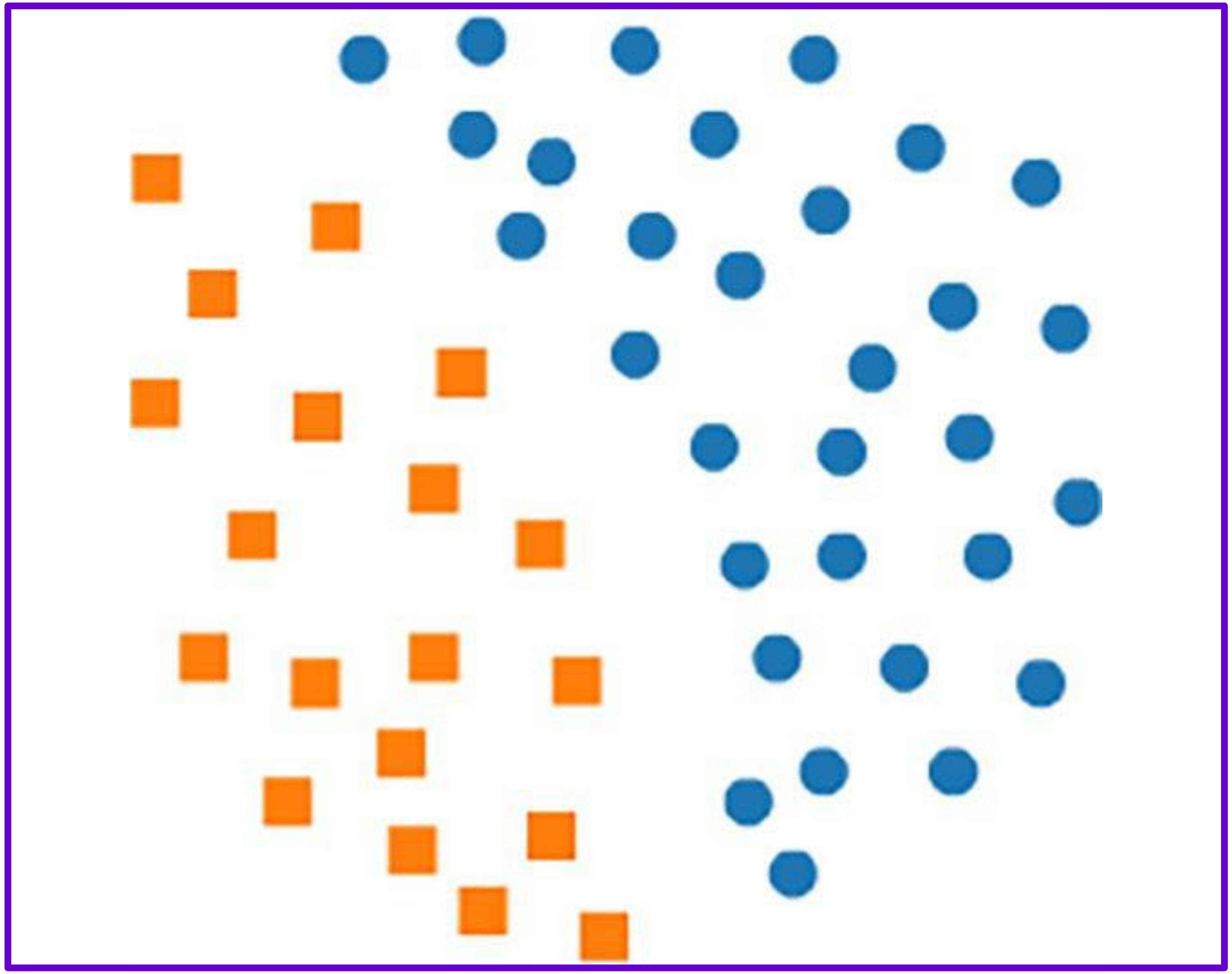
Tomek’s Links for Undersampling: Step 3

After we find the coupled data, we will remove the major class for under-sampling.



Tomek’s Links for Undersampling

The above illustrations give us a simple working of Tomek’s Link. This helps us to define a good decision boundary and remove some of the data points from the majority class.

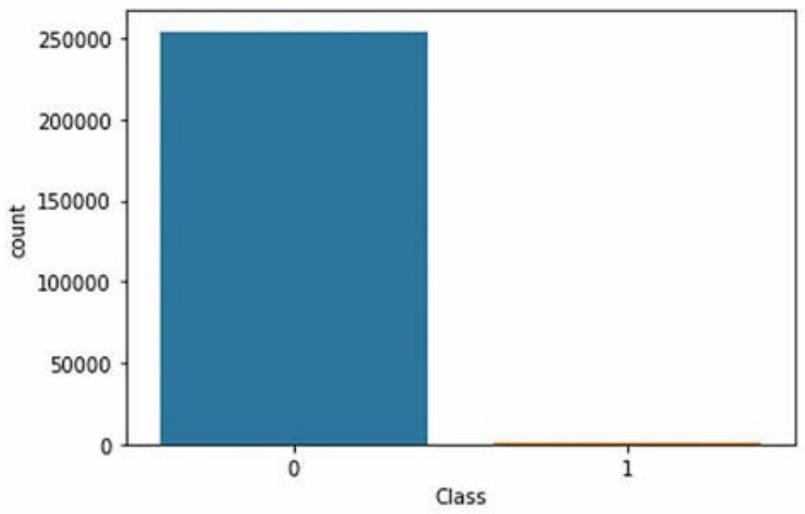


Tomek’s Links for Undersampling

We can clearly state that there won’t be any change from the original data as the change is not significant enough. Let’s plot the principal components and see how it looks.

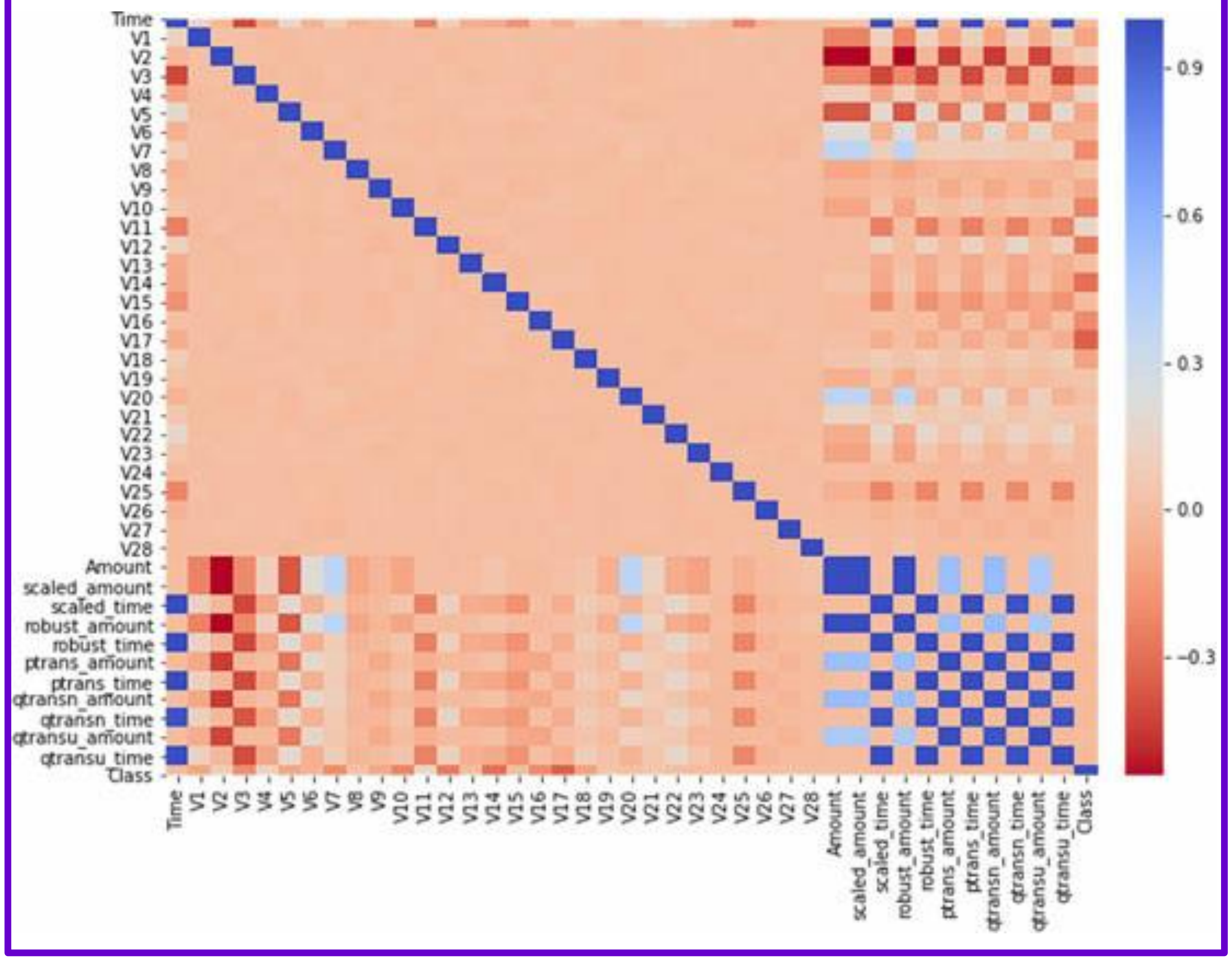
```python
from imblearn.under_sampling import TomekLinks

tkl = TomekLinks(sampling_strategy='auto', n_jobs=-1)
X_tl, y_tl = tkl.fit_sample(X_train, y_train)

train_df_tkl = X_tl
train_df_tkl['Class'] = y_tl

print('Tomek links under-sampling:')
print(train_df_tkl.Class.value_counts())
sns.countplot('Class', data=train_df_tkl)
```

```
Tomek links under-sampling:
0    254204
1       418
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a5e531890>
```



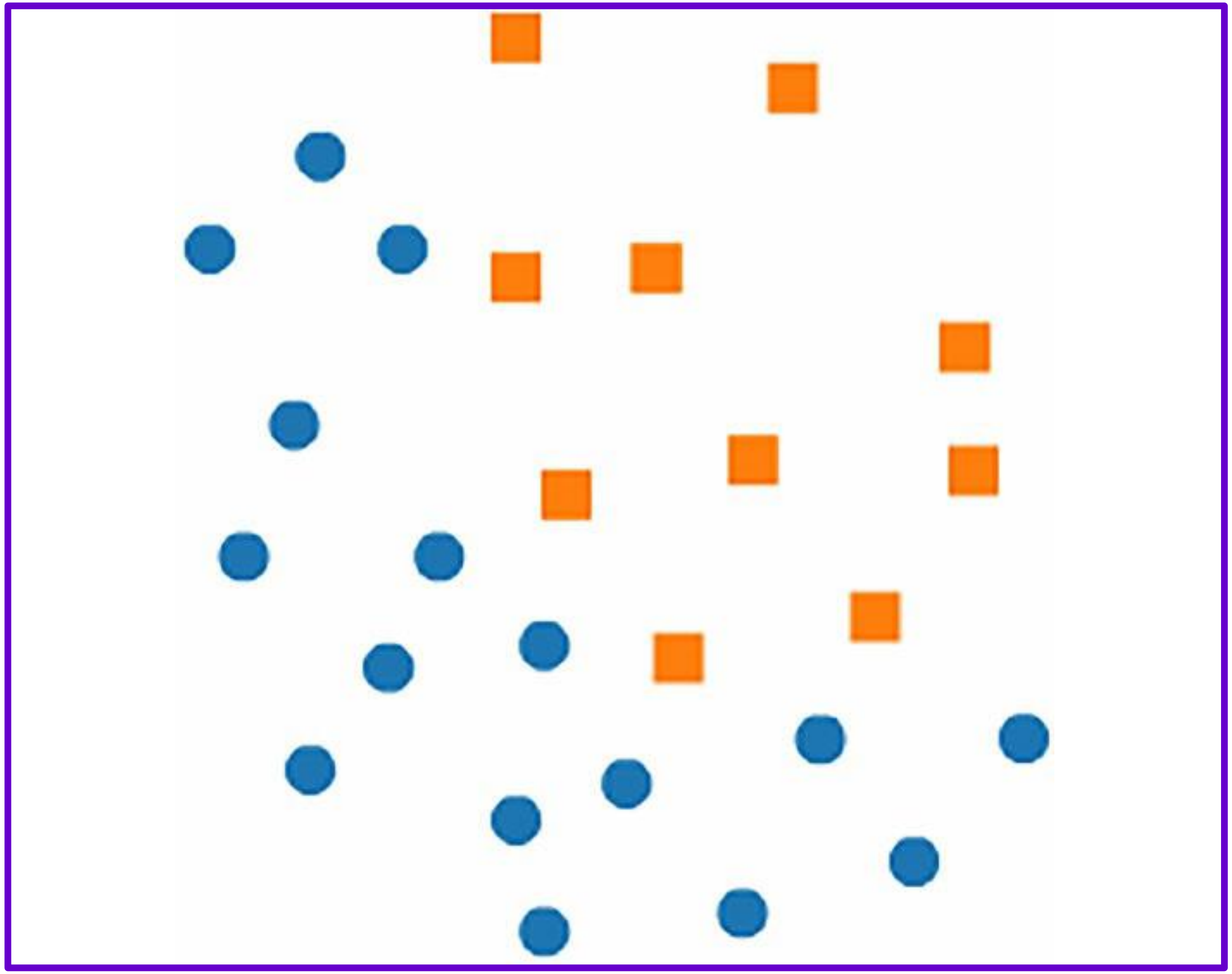
PCA plot for Tomek's Link

We can see it is the same distribution as the original dataframe. We can confirm that the heat map did not change at all, so this technique cannot be used for under-sampling.



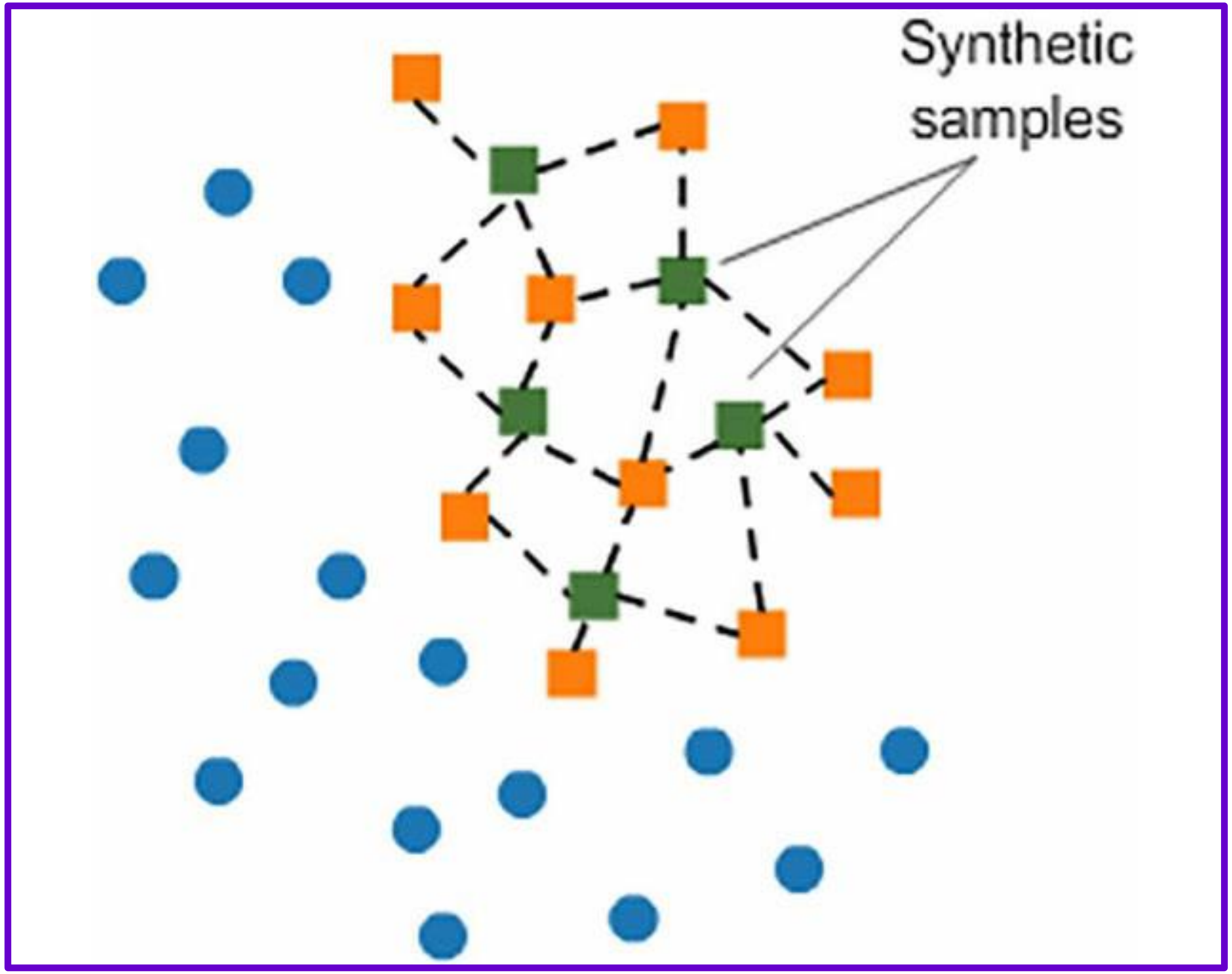
Synthetic Minority Oversampling Technique: Step 1

Let us plot the data points n dimensional space(n-features) and classify them as per the "Class."



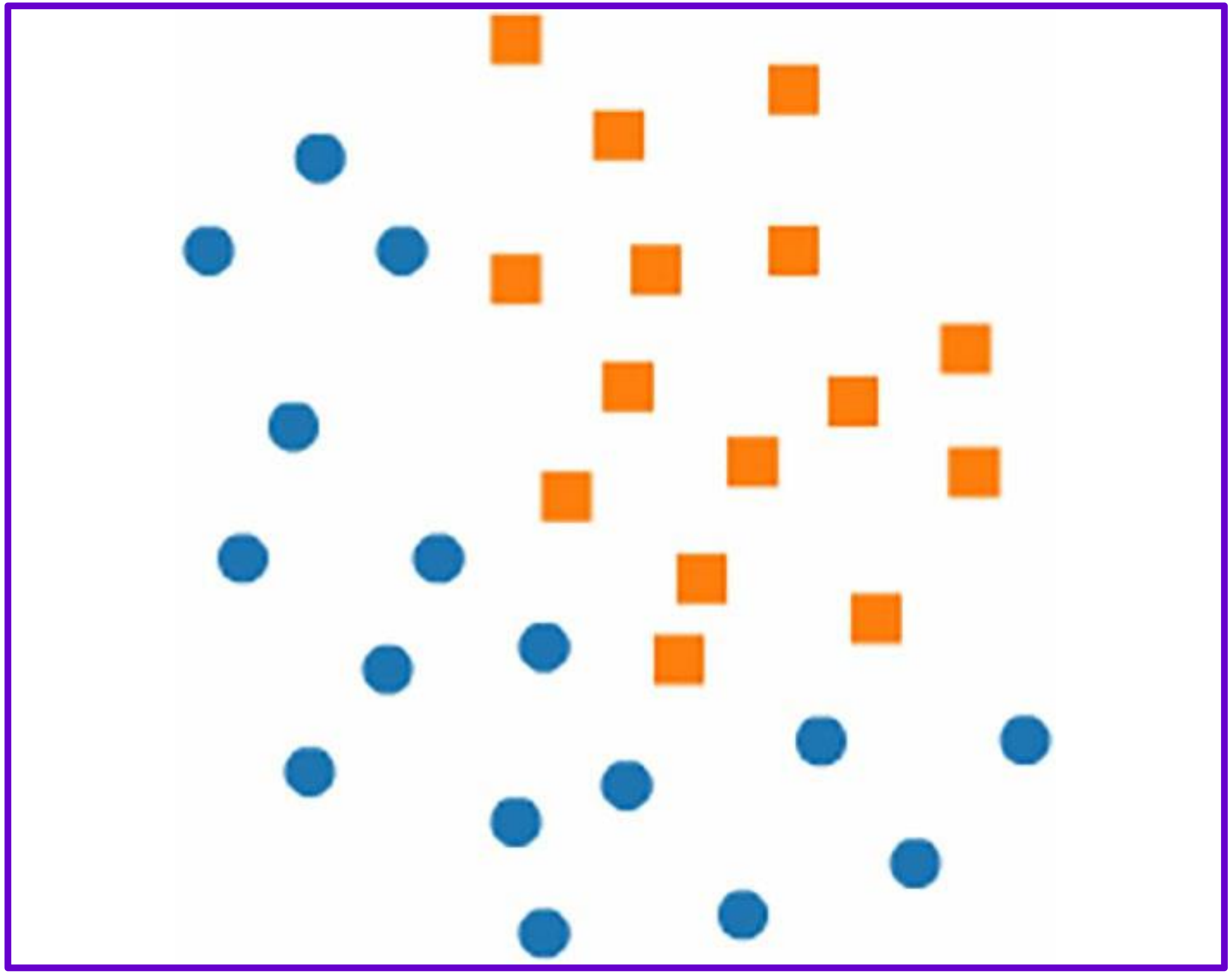
Synthetic Minority Oversampling Technique: Step 2

Create a pattern/boundary of the minority dataset with the data points. We can create a space using the boundary of the minority data and add synthetic values within the boundary where the nearest neighbor is also of the same class.



Synthetic Minority Oversampling Technique: Step 3

Plot those points and which meets step 2 conditions and assign them the minor class.

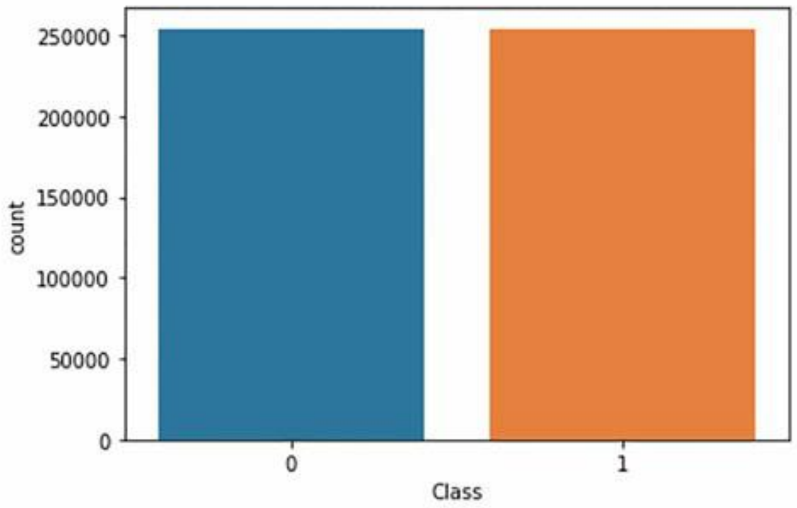


Synthetic Minority Oversampling Technique

We can see that it equidistributed, and we have the same number of records for both the classes.

```
from imblearn.over_sampling import SMOTE

sm = SMOTE()
X_sm, y_sm = sm.fit_sample(X_train, y_train)

train_df_sm = X_sm
train_df_sm['Class'] = y_sm

print('SMOTE over-sampling:')
print(train_df_sm.Class.value_counts())
sns.countplot('Class', data=train_df_sm)
```

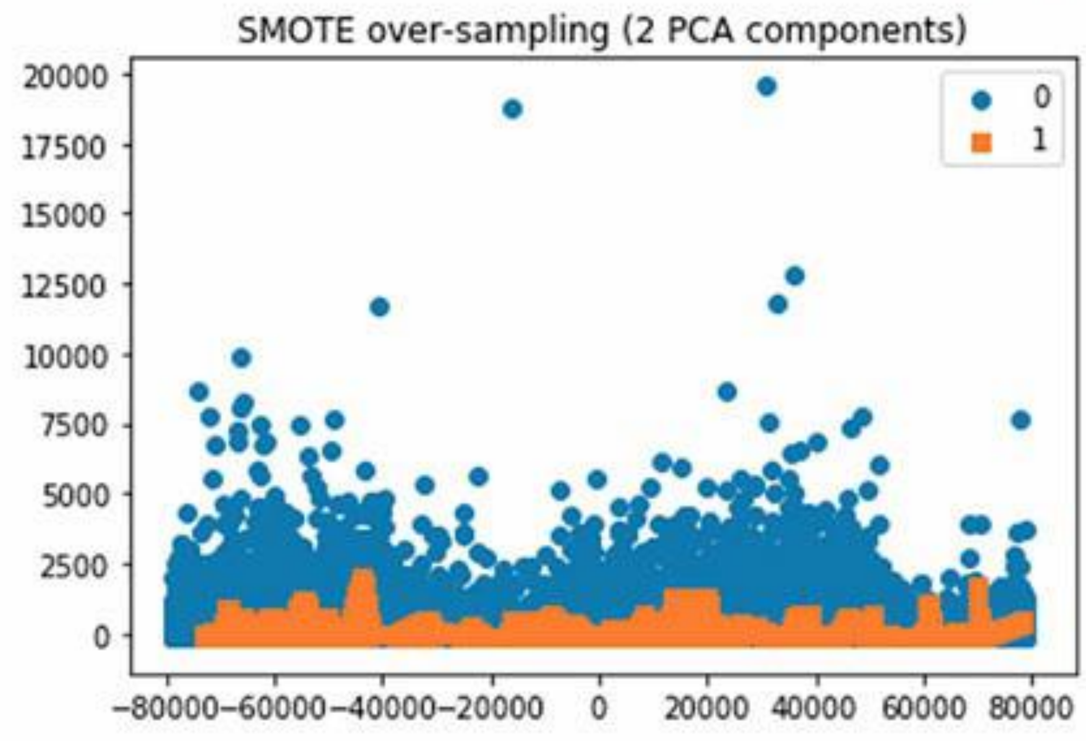
```
SMOTE over-sampling:
1    254266
0    254266
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a6b2f3cd0>
```



PCA plot for SMOTE

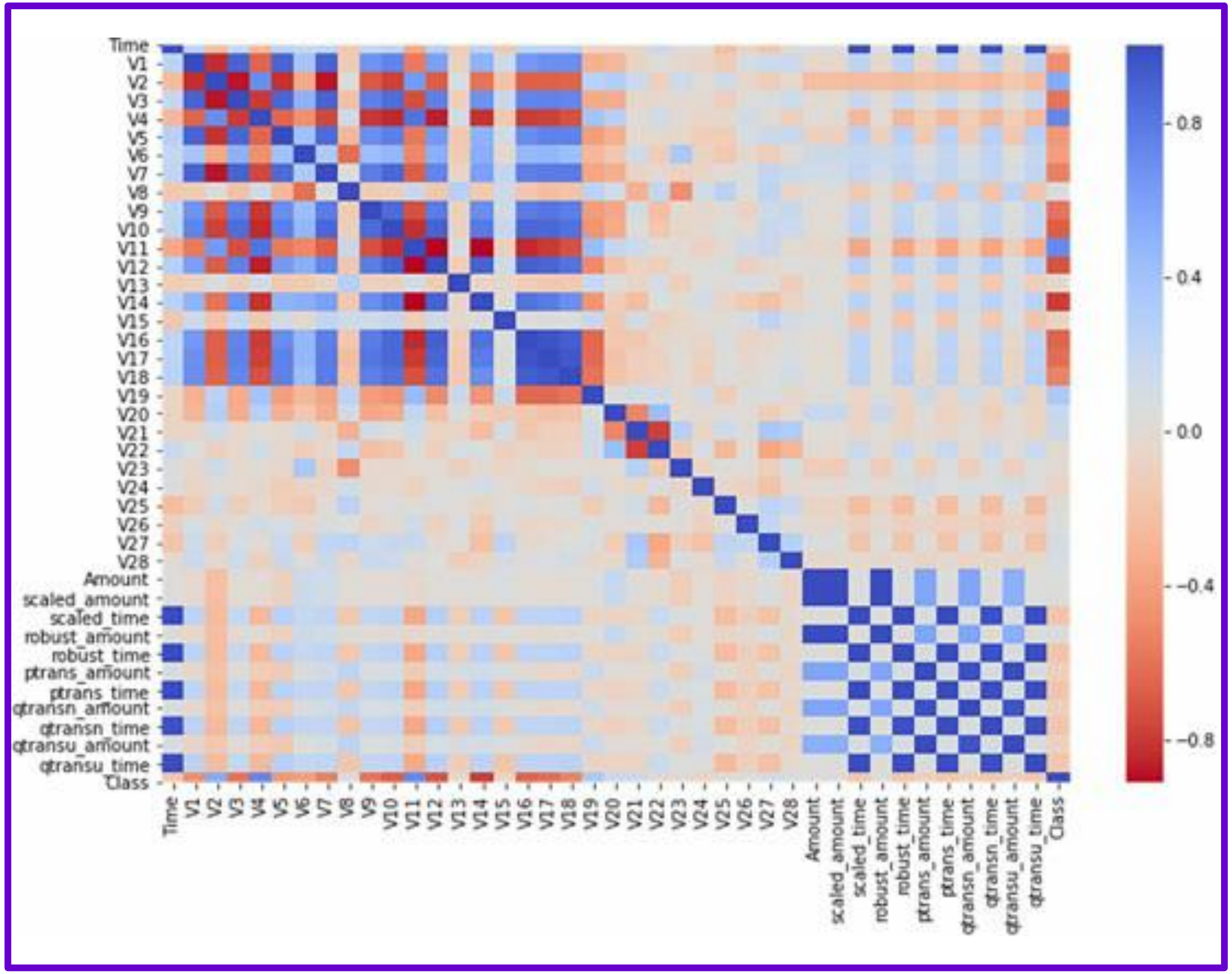
We can see there is a difference in the distribution. But it is not significant enough(it seems so)

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_sm.drop(columns=["Class"]))

plot_2d_space(X, train_df_sm["Class"], 'SMOTE over-sampling (2 PCA components)')
```



Correlation Heatmap for SMOTE dataset

Interestingly, we can see a similar heat map for both Random Oversampling and SMOTE



Function for plot Box-and-whiskers plot for Positive Correlated Data

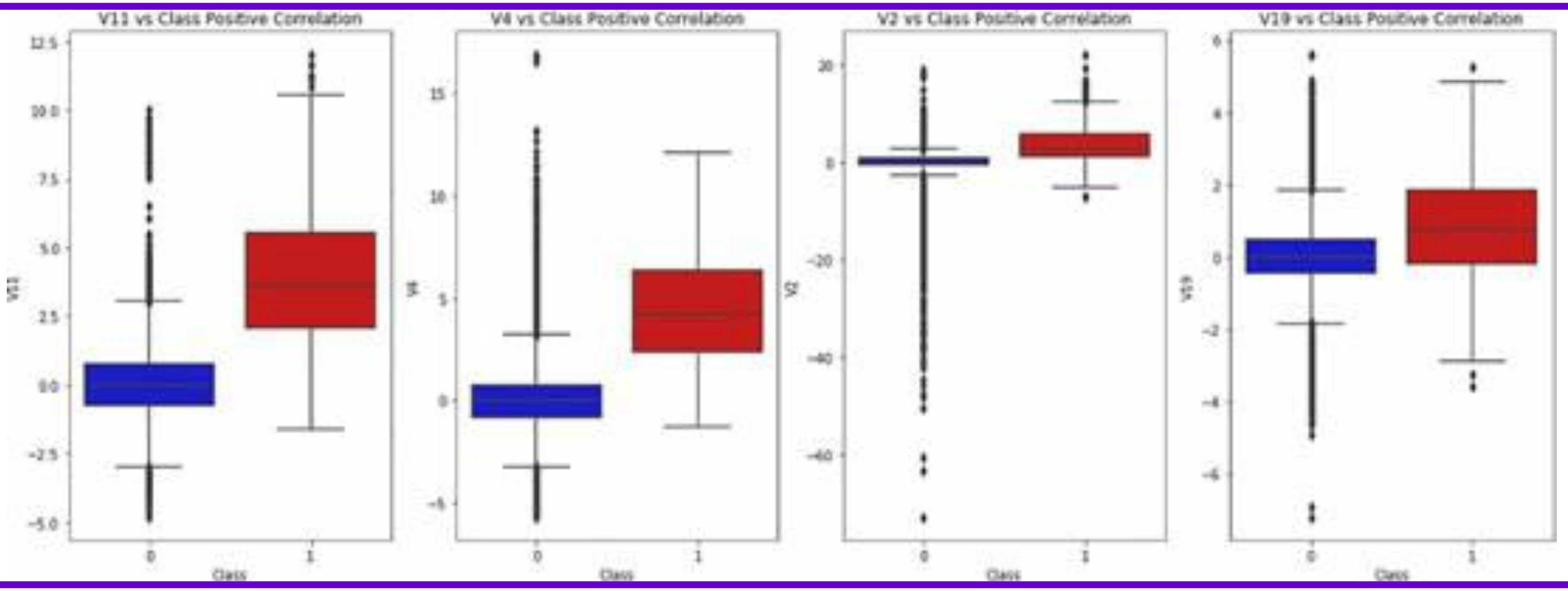
From the last section, we have seen the features V2, V4, V11, and V19 are positively correlated.

We will write a function that will help us to plot all the boxplots to compare them.

```
def plot_pos_box(df):
    f, axes = plt.subplots(ncols=4, figsize=(20,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="V11", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('V11 vs Class Positive Correlation')

    sns.boxplot(x="Class", y="V4", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('V4 vs Class Positive Correlation')

    sns.boxplot(x="Class", y="V2", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('V2 vs Class Positive Correlation')

    sns.boxplot(x="Class", y="V19", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('V19 vs Class Positive Correlation')

    plt.show()
```

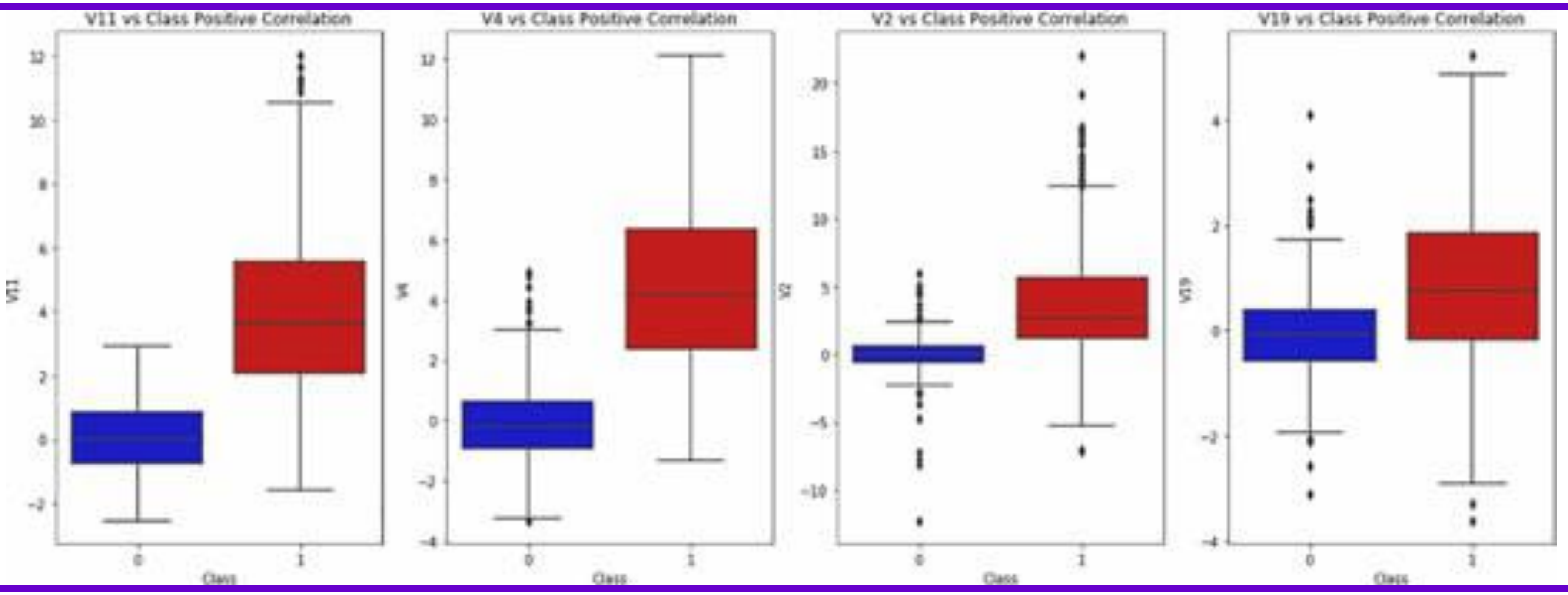
Original training data

We can see that the original training dataframe has an extremely large number of outliers for class 0, and that is very bad for modeling, so we went for resampling the dataset



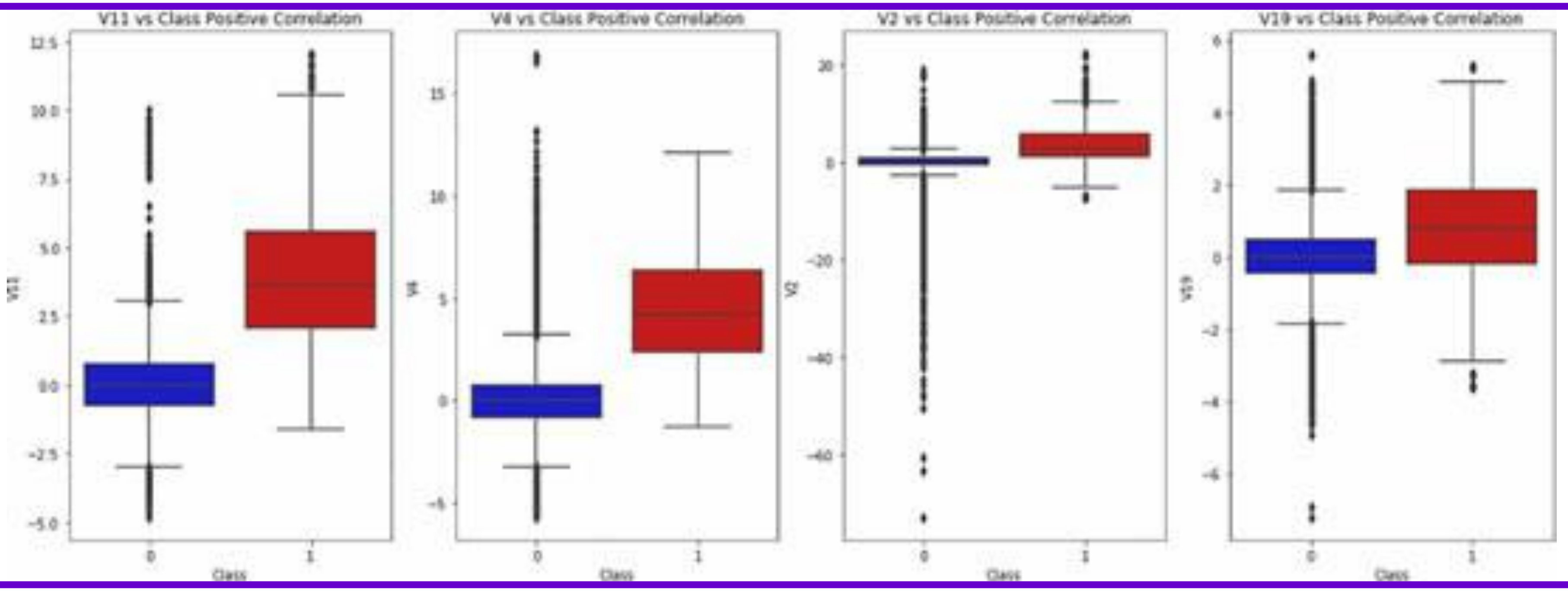
Random under-sampling

This plot looks way better as there are a smaller number of outliers, and this can be used for training machine learning models.



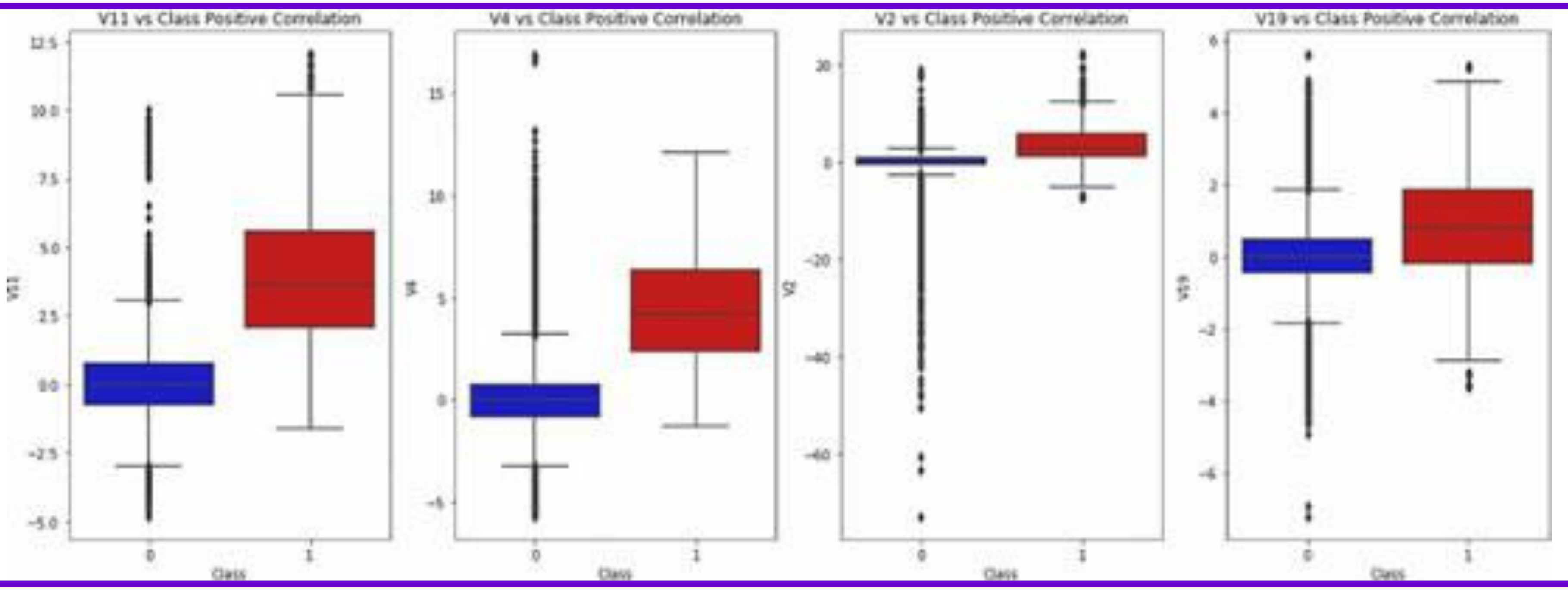
Random over-sampling

As this shows so many outliers, we cannot use this dataset for modeling. So, we will be ignoring this dataset.



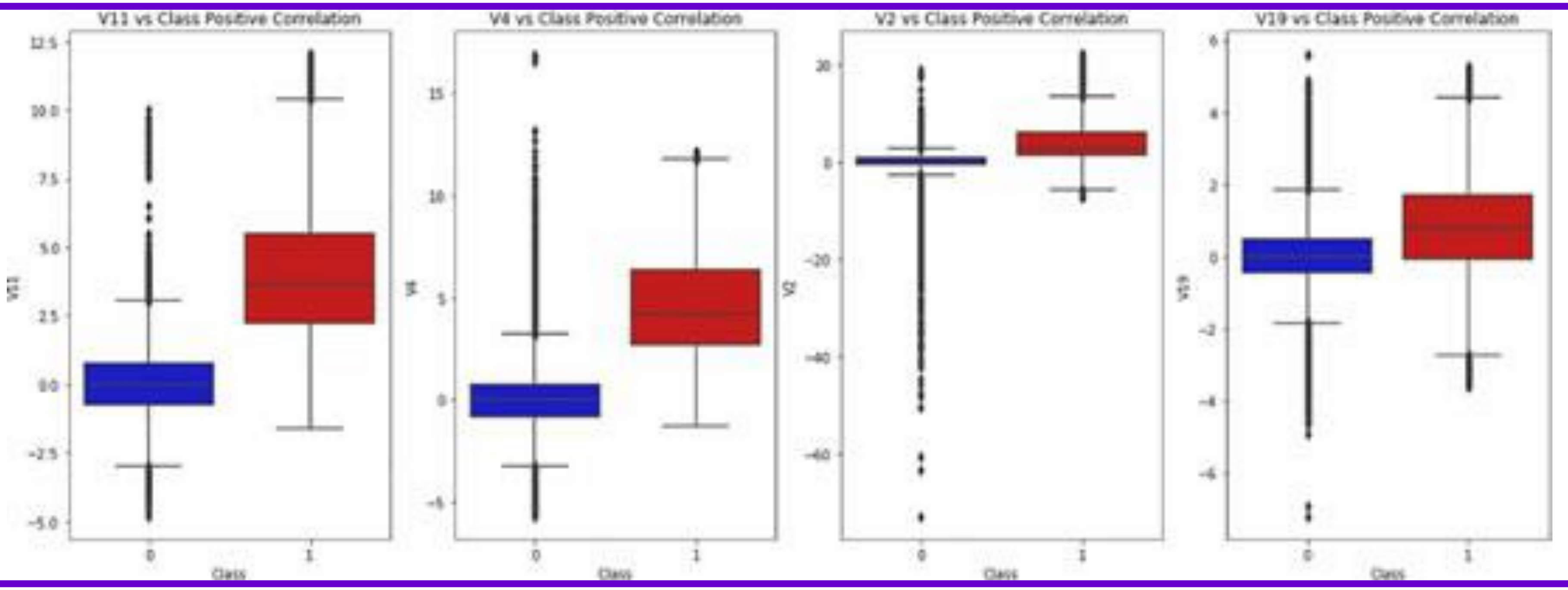
Tomek's Link under-sampling

This plot is exactly similar to the original training dataset, as there were only 62 records that were removed from the majority class.



SMOTE

SMOTE is also an oversampling technique so that it will look similar to Random Oversampling.



Function for plot Box-and-whiskers plot for Negative Correlated Data

Similarly, we will write a function for negatively correlated values to plot box-and-whiskers and find the optimal resampling technique

```
def plot_neg_box(df):
    f, axes = plt.subplots(ncols=5, figsize=(28,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="V17", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('V17 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V14", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('V14 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V12", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('V12 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V16", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('V16 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V10", data=df, palette=colors, ax=axes[4])
    axes[4].set_title('V10 vs Class Negative Correlation')

    plt.show()
```

Function for plot Box-and-whiskers plot for Scaled Amount

Now we have to select which scaling technique to use for the modeling purpose. We have selected only two resampling techniques from the previous section. With that dataframe, we will be looking at the scaled data and select the optimal algorithm for the scaled feature (“Time” and “Amount”).

```python
def plot_scaled_amt(df):
    f, axes = plt.subplots(ncols=5, figsize=(28,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="scaled_amount", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('Standard Scaling (Amount)')

    sns.boxplot(x="Class", y="robust_amount", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('Robust Scaling (Amount)')

    sns.boxplot(x="Class", y="ptrans_amount", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('Power Transformer (Amount)')

    sns.boxplot(x="Class", y="qtransn_amount", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('Quartile Transformer -Normal (Amount)')

    sns.boxplot(x="Class", y="qtransu_amount", data=df, palette=colors, ax=axes[4])
    axes[4].set_title('Quartile Tranformer -Uniform (Amount)')

    plt.show()
```

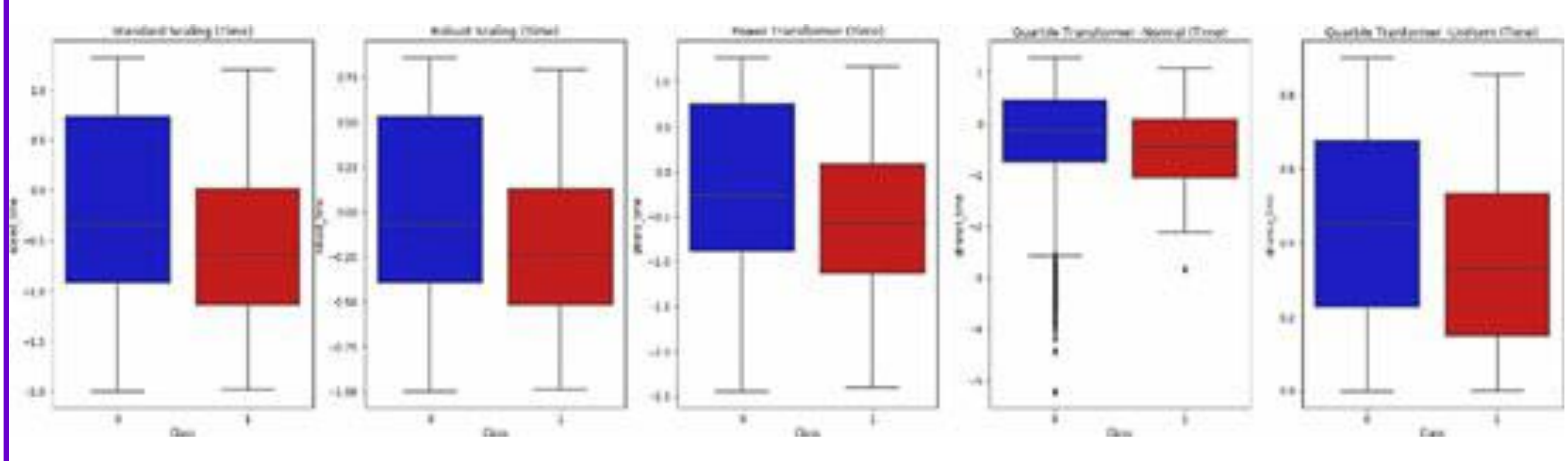
Function for plot Box-and-whiskers plot for Scaled Time

Now we have to select which scaling technique to use for the modeling purpose. We have selected only two resampling techniques from the previous section. With that dataframe, we will be looking at the scaled data and select the optimal algorithm for the scaled feature (“Time” and “Amount”).

```
def plot_scaled_time(df):
    f, axes = plt.subplots(ncols=5, figsize=(28,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="scaled_time", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('Standard Scaling (Time)')

    sns.boxplot(x="Class", y="robust_time", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('Robust Scaling (Time)')

    sns.boxplot(x="Class", y="ptrans_time", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('Power Transformer (Time)')

    sns.boxplot(x="Class", y="qtransn_time", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('Quartile Transformer -Normal (Time)')

    sns.boxplot(x="Class", y="qtransu_time", data=df, palette=colors, ax=axes[4])
    axes[4].set_title('Quartile Tranformer -Uniform (Time)')

    plt.show()
```

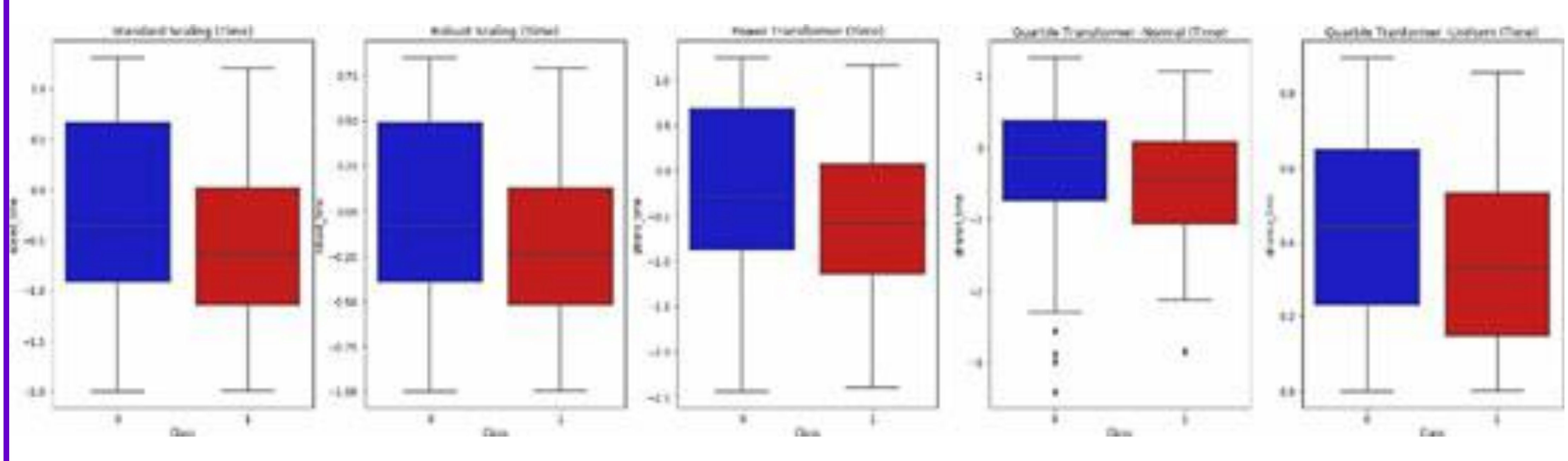
Time:
Original Training Data for Box plot We

We can see for “Quantile Transformer -Normal” there is skewness and outliers in the plot. We now have to compare this entire plot with Random Undersampling and SMOTE



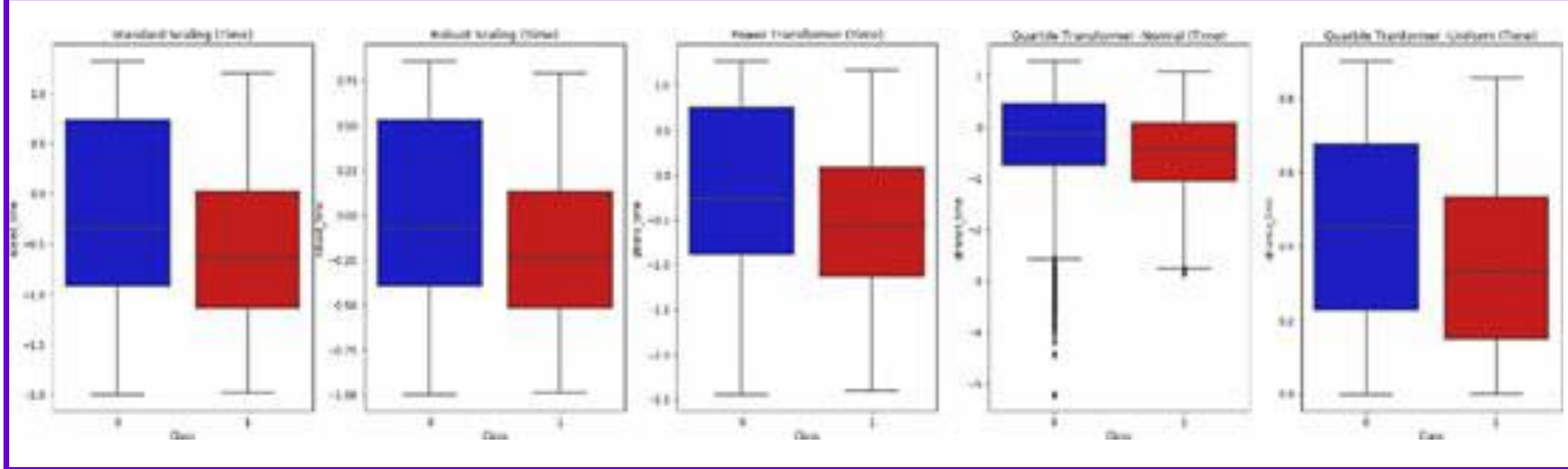
Time:
Original Training Data for Box plot We

For this dataset, we can see the same thing that for “Quantile Transformer - Normal,” there are outliers in the plot. Apart from that, all looks good for use. Previously, we have seen that “Quantile Transformer -Uniform” actually changed the distribution so that can be ignored as well.



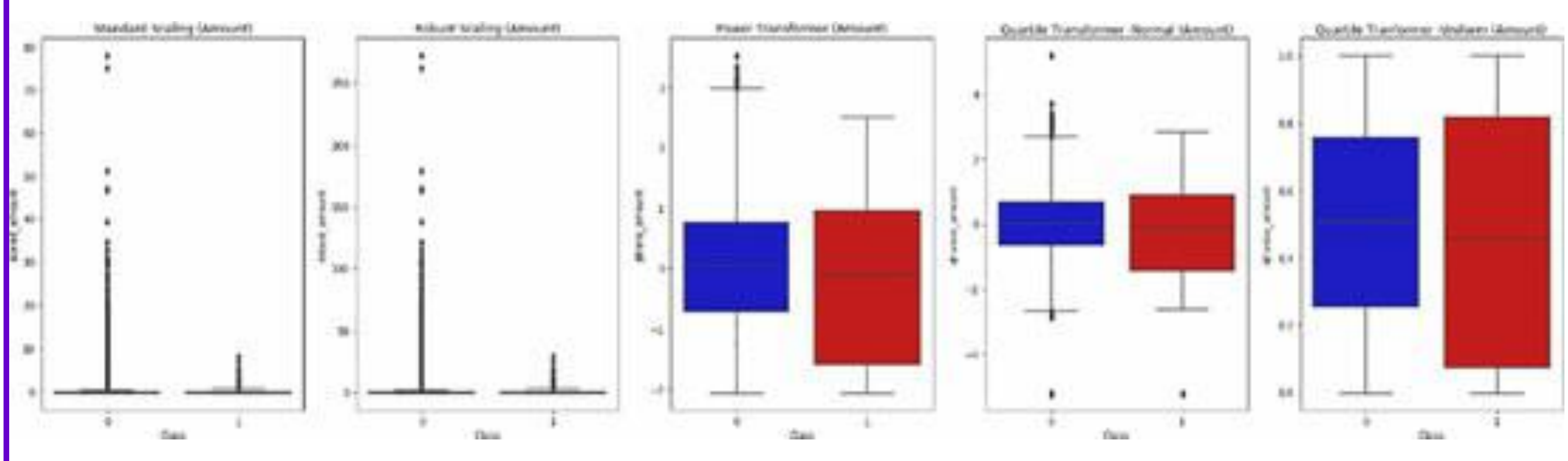
Time:
SMOTE for Box plot

For the SMOTE dataframe, we see the same result, so, for the feature “Time,” we can choose any algorithm apart from “Quantile Transformer.”



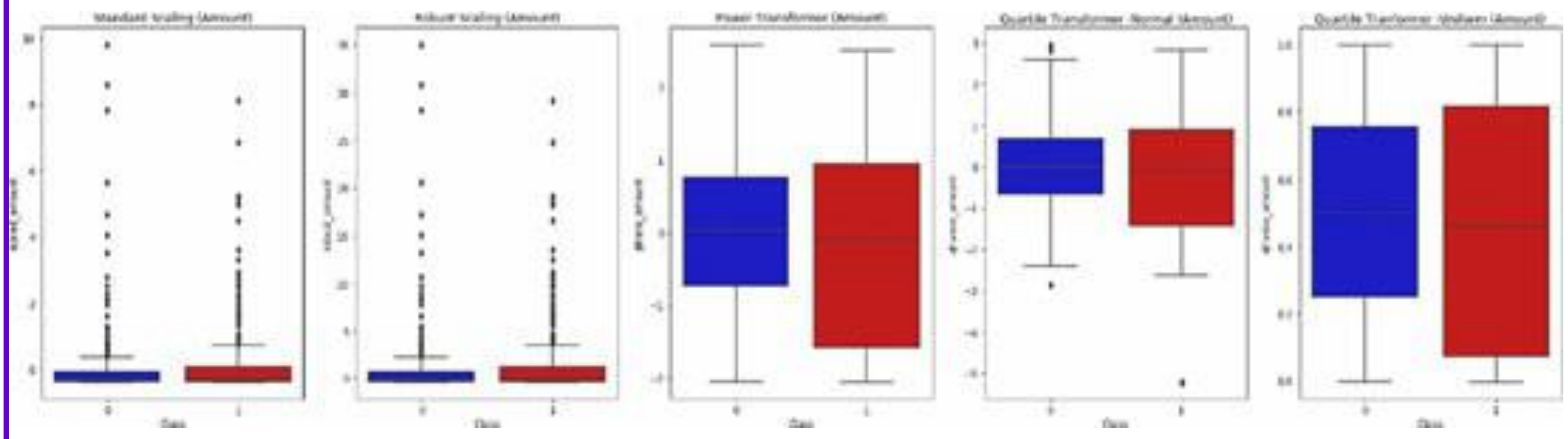
Amount:
Original Training Data for Box plot We

Here, for the amount, we cannot ignore or eliminate the outliers as they will give us a lot of information. We will try to stick with the original distribution with some minor changes.



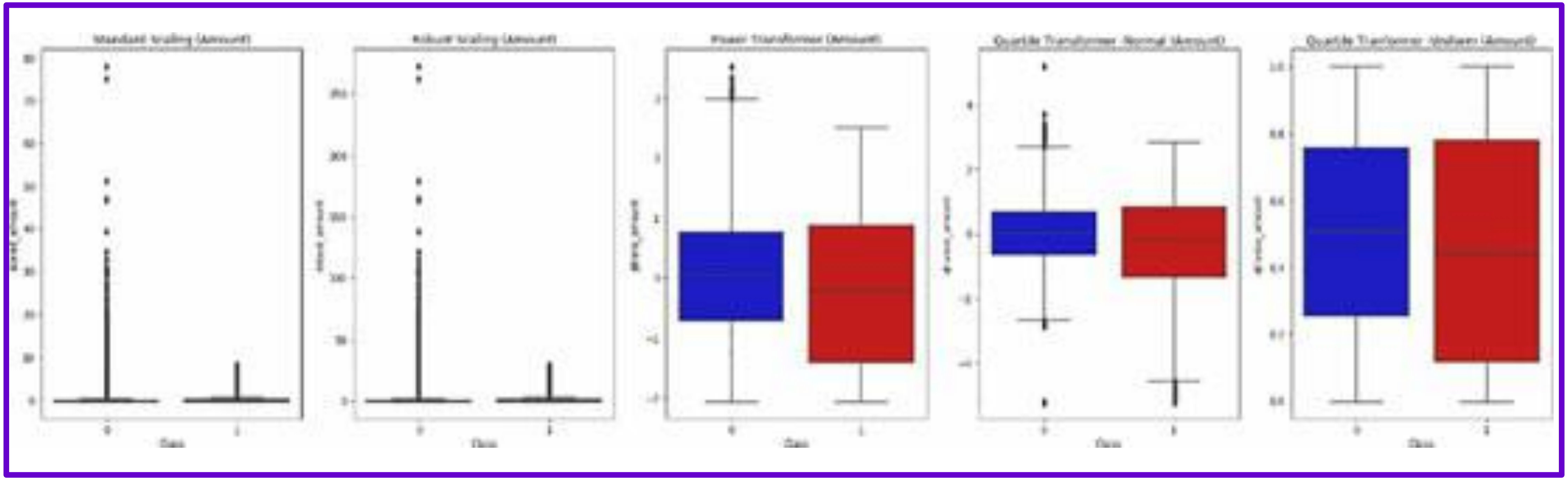
Amount:
Random under sampling for Box plot

We can see outliers have reduced, and Power Transformer looks good in terms of distribution. But we will also look at Robust Scaler as it more or less gives us the original distribution. “Quantile Transformer - Normal” looks good, but it has already performed badly for the future “Time,” so we will ignore it



Amount:
SMOTE for Box plot

SMOTE plot shows a similar thing like the original training dataset; hence we will stick with our last decision.



Find the best(Optimal) data for the model

SMOTE Dataset 1 - Robust Scaled Time, Robust Scaled Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

SMOTE Dataset 2 - Power Transformer Time, Power Transformer Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

Random Under-sample Dataset 1 - Robust Scaled Time, Robust Scaled Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

Random Under-sample Dataset 2 - Power Transformer Time, Power Transformer Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

6	Logistic Regression
7	Decision Tree
8	Support Vector Machine (SVM)
9	Data Preparation & Metric Trap
10	Training & Evaluation

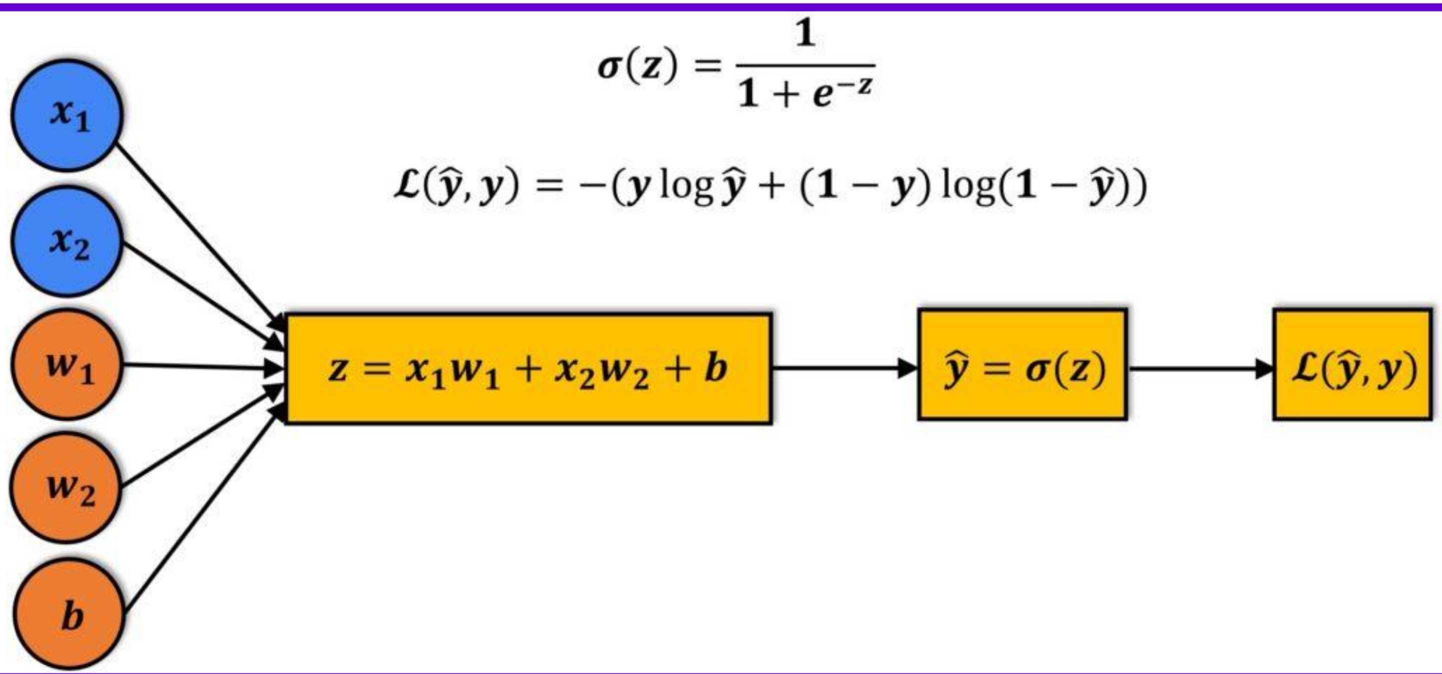
Logistic Regression Model

Logistic Regression is a predictive analysis which is used to explain the data and **relationship between** one dependent **binary variable** and **one or more nominal, ordinal, interval or ratio-level independent variables**.

$$p(x) = \frac{e^{(\beta_0+\beta_1 x_1+\cdots+\beta_k x_k)}}{1+e^{(\beta_0+\beta_1 x_1+\cdots+\beta_k x_k)}}$$

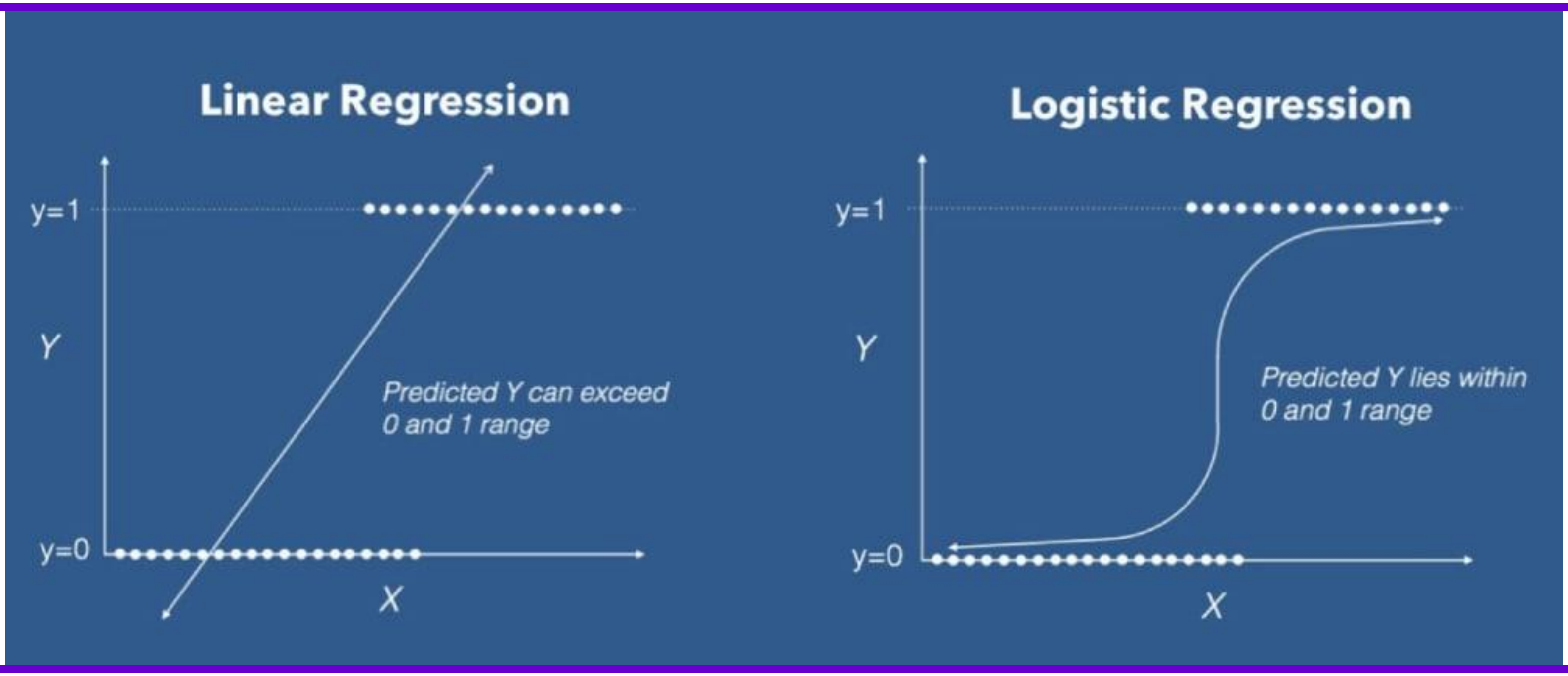
Logistic Regression Model

A linear equation (z) is given to a sigmoidal activation function (σ) to predict the output (ŷ).



Logistic Regression Vs Linear Regression

Unlike linear regression, we get an ‘S’ shaped curve in logistic regression.



6 Logistic Regression

7 Decision Tree

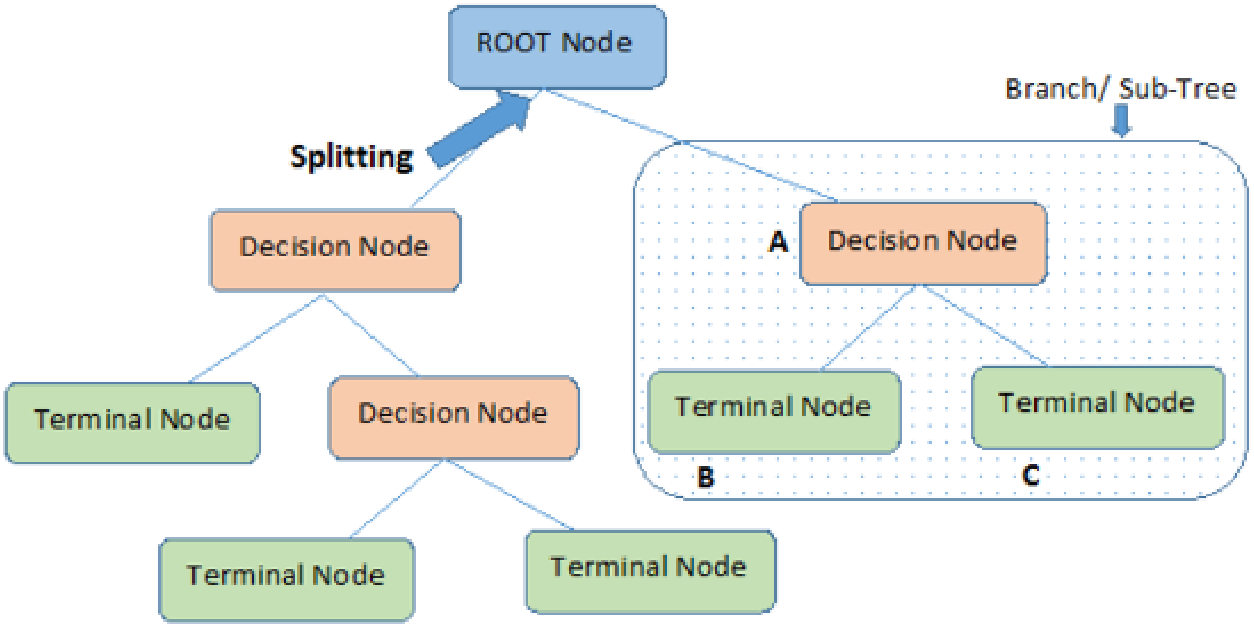
8 Support Vector Machine (SVM)

9 Data Preparation & Metric Trap

10 Training & Evaluation

Decision Tree Model

A decision tree is a predictive model that uses a flowchart-like structure to make decisions based on input data. It divides data into branches and assigns outcomes to leaf nodes.



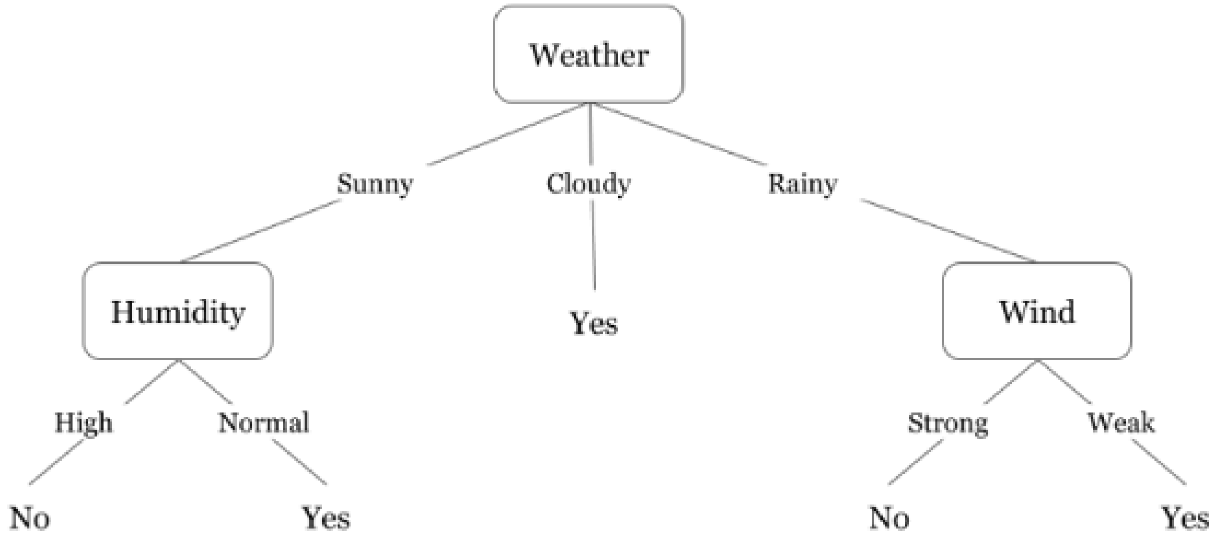
Example of Decision Tree

Decision trees are upside down which means the root is at the top and then this root is split into various several nodes.

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

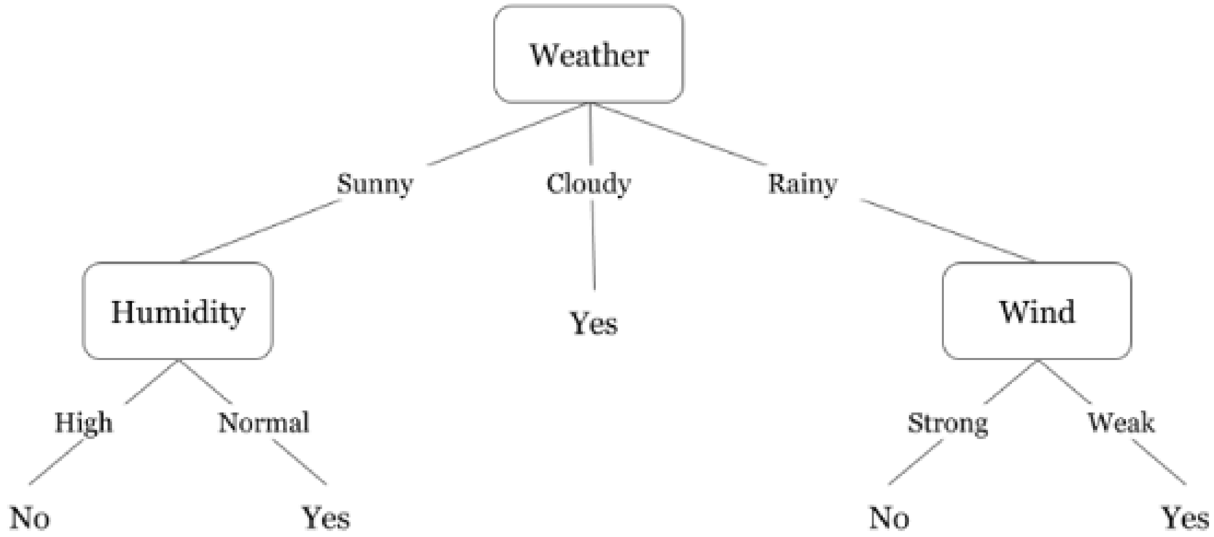
Example of Decision Tree

In the below diagram the tree will first ask what is the weather? Is it sunny, cloudy, or rainy? If yes then it will go to the next feature which is humidity and wind. It will again check if there is a strong wind or weak, if it's a weak wind and it's rainy then the person may go and play.



Example of Decision Tree (uncertainty in the dataset)

Now you must be thinking how do I know what should be the root node? what should be the decision node? when should I stop splitting? To decide this, there is a metric called “Entropy” which is the amount of uncertainty in the dataset.



Entropy

Entropy is nothing but the uncertainty in our dataset or measure of disorder.

Suppose you have a group of friends who decides which movie they can watch together on Sunday. There are 2 choices for movies, one is ***"Lucy"*** and the second is ***"Titanic"*** and now everyone has to tell their choice. After everyone gives their answer we see that *"Lucy" gets 4 votes* and *"Titanic" gets 4 votes*. Which movie do we watch now?

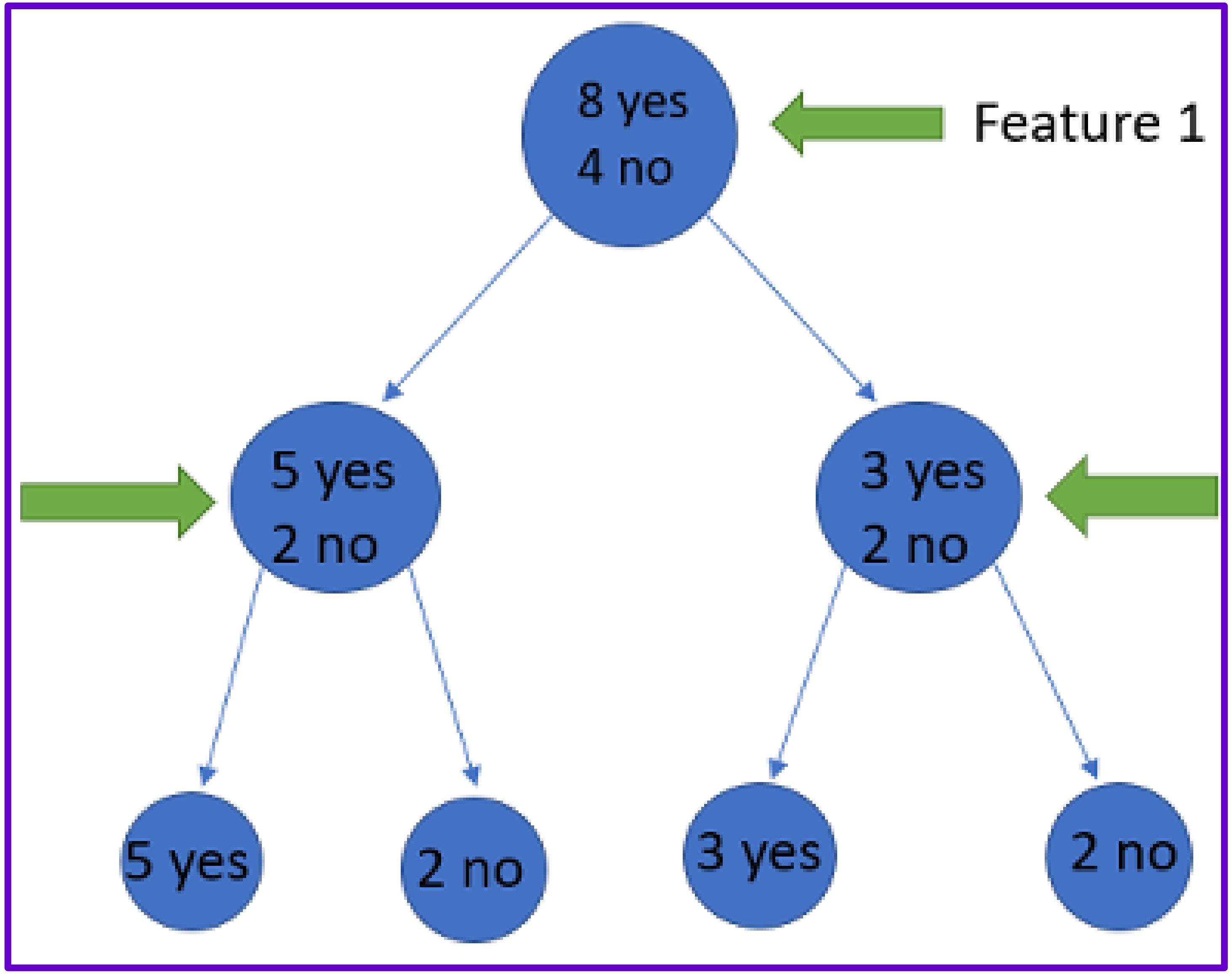
The formula for Entropy

Here p+ is the probability of positive class
p– is the probability of negative class
S is the subset of the training example

$$E(S) = -p_{(+)}\log p_{(+)} - p_{(-)}\log p_{(-)}$$

How do Decision Trees use Entropy?

Entropy basically measures the impurity of a node. Impurity is the degree of randomness; it tells how random our data is. A **pure sub-split** means that either you should be getting "yes", or you should be getting "no".



How do Decision Trees use Entropy?...
Is our splits pure?
(For feature 2)

We see here the split is not pure, why? Because we can still see some negative classes in both the nodes.

$$\Rightarrow -\left(\frac{5}{7}\right)log_2\left(\frac{5}{7}\right) - \left(\frac{2}{7}\right)log_2\left(\frac{2}{7}\right)$$

$$\Rightarrow -(0.71 * -0.49) - (0.28 * -1.83)$$

$$\Rightarrow -(-0.34) - (-0.51)$$

$$\Rightarrow 0.34 + 0.51$$

$$\Rightarrow 0.85$$

How do Decision Trees use Entropy?...
Is our splits pure?
(For feature 3)

We see here the split is not pure, why? Because we can still see some negative classes in both the nodes.

$$\Rightarrow -\left(\frac{3}{5}\right)log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)log_2\left(\frac{2}{5}\right)$$

$$\Rightarrow -(0.6 * -0.73) - (0.4 * -1.32)$$

$$\Rightarrow -(-0.438) - (-0.528)$$

$$\Rightarrow 0.438 + 0.528$$

$$\Rightarrow 0.966$$

Decrease the uncertainty or impurity in the dataset?

the goal of machine learning is to decrease the uncertainty or impurity in the dataset, here by using the entropy we are getting the impurity of a particular node, we don't know if the parent entropy or the entropy of a particular node has decreased or not

For this, we bring a new metric called "Information gain" which tells us how much the parent entropy has decreased after splitting it with some feature.

Information Gain

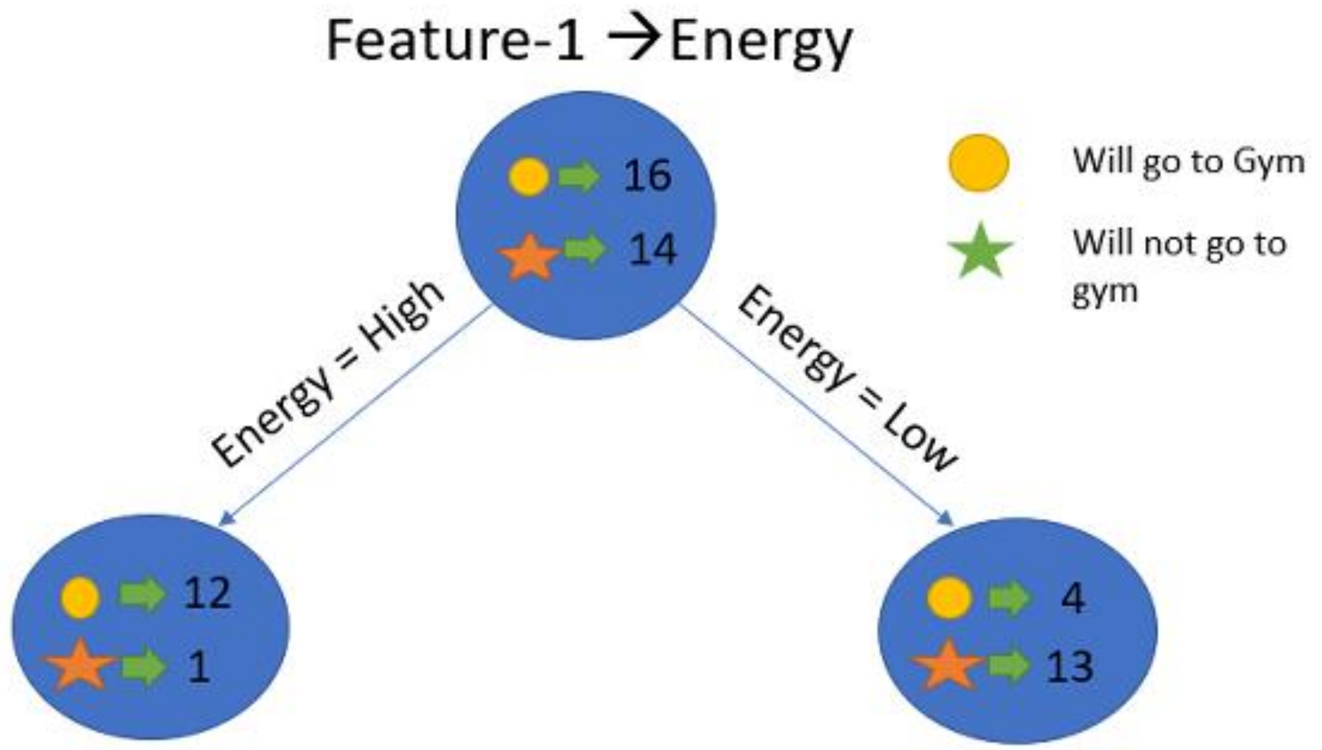
Information gain measures the reduction of uncertainty given some feature and it is also a deciding factor for which attribute should be selected as a decision node or root node.

It is just entropy of the full dataset – entropy of the dataset given some feature.

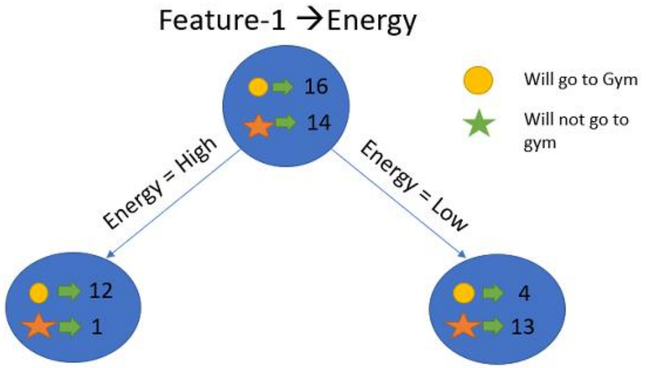
$$Information\ Gain\ =\ E(Y)\ -\ E(Y|X)$$

Information Gain Example

suppose our entire population has a total of 30 instances. The dataset is to predict whether the person will go to the gym or not. Let's say 16 people go to the gym and 14 people don't.



Information Gain Example …



$$E(Parent) = -\left(\frac{16}{30}\right)\log_2\left(\frac{16}{30}\right) - \left(\frac{14}{30}\right)\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Parent|Energy = \text{"high"}) = -\left(\frac{12}{13}\right)\log_2\left(\frac{12}{13}\right) - \left(\frac{1}{13}\right)\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Parent|Energy = \text{"low"}) = -\left(\frac{4}{17}\right)\log_2\left(\frac{4}{17}\right) - \left(\frac{13}{17}\right)\log_2\left(\frac{13}{17}\right) \approx 0.79$$

Information Gain Example ...

To see the weighted average of entropy of each node we will do as follows:

$$E(Parent|Energy) = \frac{13}{30} * 0.39 + \frac{17}{30} * 0.79 = 0.62$$

Information Gain Example ...

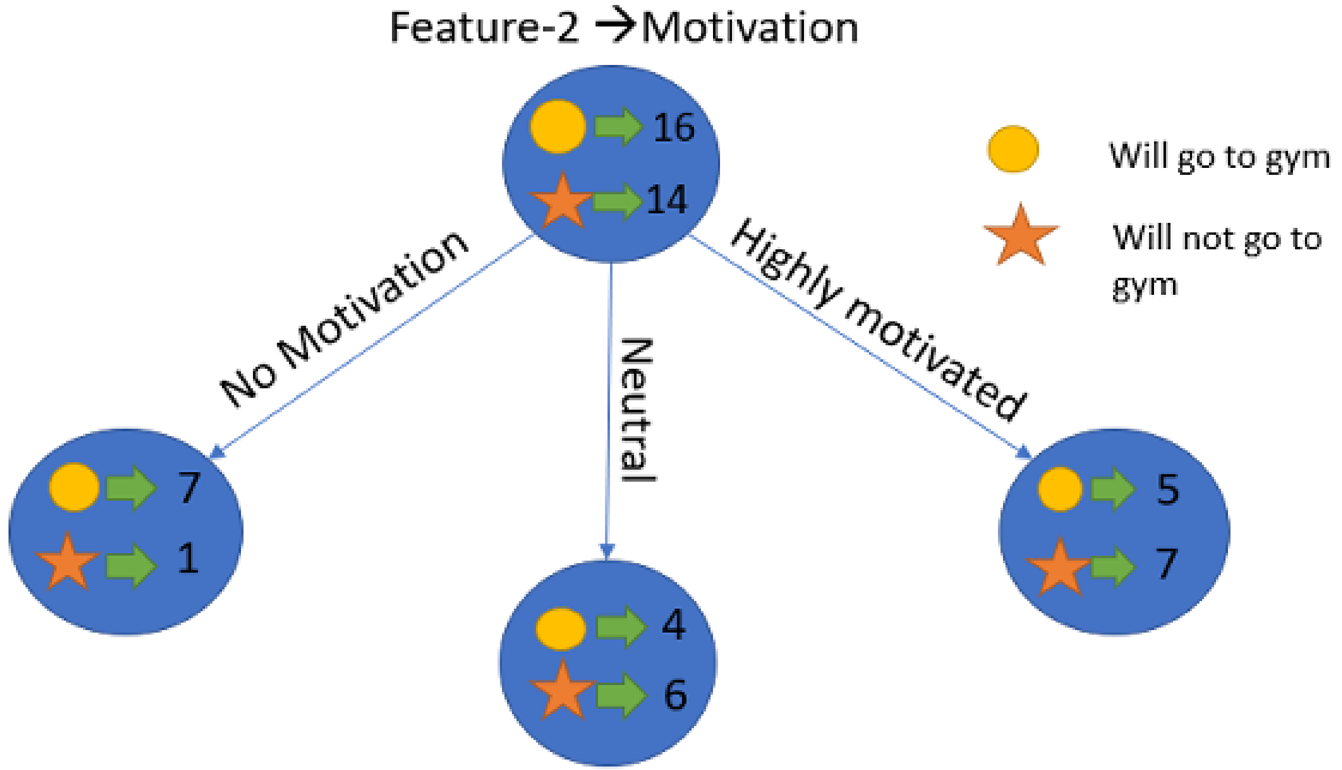
Now we have the value of E(Parent) and E(Parent|Energy), information gain will be:

$$
\begin{aligned}
Information\ Gain &= E(parent) - E(parent|energy) \\
&= 0.99 - 0.62 \\
&= 0.37
\end{aligned}
$$

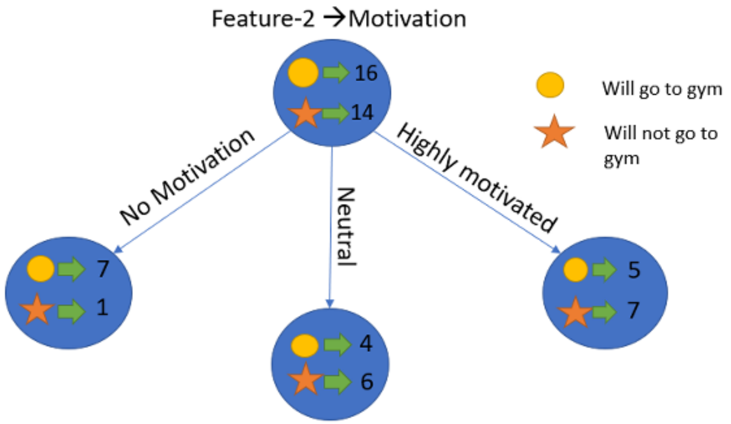
Our parent entropy was near 0.99 and after looking at this value of information gain, we can say that the entropy of the dataset will decrease by 0.37 if we make "Energy" as our root node.

Information Gain
Example-2

Similarly, we will do this with the other feature "Motivation" and calculate its information gain.



Information Gain Example 2...



$$E(Parent) = 0.99$$

$$E\left(Parent|Motivation = "No\ motivation"\right) = -\left(\frac{7}{8}\right)log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) = 0.54$$

$$E(Parent|Motivation = "Neutral") = -\left(\frac{4}{10}\right)log_2\left(\frac{4}{10}\right) - (\frac{6}{10})\log_2\left(\frac{6}{10}\right) = 0.97$$

$$E(Parent|Motivation = "Highly\ motivated") = -\left(\frac{5}{12}\right)log_2\left(\frac{5}{12}\right) - (\frac{7}{12})\log_2\left(\frac{7}{12}\right) = 0.98$$

Information Gain Example 2...

To see the weighted average of entropy of each node we will do as follows:

$$E(Parent|Motivation) = \frac{8}{30} * 0.54 + \frac{10}{30} * 0.97 + \frac{12}{30} * 0.98 = 0.86$$

Information Gain
Example 2…

Now we have the value of E(Parent) and E(Parent|Energy), information gain will be:

$$
\begin{aligned}
Information\ Gain &= E(Parent) - E(Parent|Motivation) \\
&= 0.99 - 0.86 \\
&= 0.13
\end{aligned}
$$

Our parent entropy was near 0.99 and after looking at this value of information gain, we can say that the entropy of the dataset will decrease by 0.37 if we make “Energy” as our root node.

Conclusion based on Information Gain in gym Example

We now see that the “Energy” feature gives more reduction which is 0.37 than the “Motivation” feature. Hence we will select the feature which has the highest information gain and then split the node based on that feature.

In this example “Energy” will be our root node and we’ll do the same for sub-nodes. Here we can see that when the energy is “high” the entropy is low and hence we can say a person will definitely go to the gym if he has high energy, but what if the energy is low? We will again split the node based on the new feature which is “Motivation”.

6. Logistic Regression
7. Decision Tree
8. Support Vector Machine (SVM)
9. Data Preparation & Metric Trap
10. Training & Evaluation

6. Logistic Regression
7. Decision Tree
8. Support Vector Machine (SVM)
9. Data Preparation & Metric Trap
10. Training & Evaluation

6. Logistic Regression
7. Decision Tree
8. Support Vector Machine (SVM)
9. Data Preparation & Metric Trap
10. Training & Evaluation

Course References

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2021.

[2] T. Ghosh and S. K. B. Math, *Practical Mathematics for AI and Deep Learning: A Concise yet In-Depth Guide on Fundamentals of Computer Vision, NLP, Complex Deep Neural Networks and Machine Learning (English Edition)*. BPB Publications, 2022.

[3] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[4] T. V. Geetha and S. Sendhilkumar, *Machine Learning: Concepts, Techniques and Applications*. CRC Press LLC, 2023.

[5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2023.

[6] O. Theobald, *Machine Learning for Absolute Beginners: A Plain English Introduction (Third Edition)*. Scatterplot Press, 2021.

Accessing Course Resource

linkedin.com/in/Samanipour

t.me/SamaniGroup

github.com/Samanipour