Table of Contents

Author's Comments on Imbalanced Dataset

In this project I have dealt with various usual machine learning classifiers such as `Logistic Regression`, `Support Vector Classifier`, `Decision Tree Classifier`, `Random Forest Classifier`, `K-Nearest Neighbours Classifier` with and without resampling methods (upsampling SMOTE and downsampling) and with and without grid search (Grid Search and Randomized Search). All these methods works only when we have label column which tells whether the transaction is fraud or not. But in real life we have label column only for training data and we do not have label for the test set. Here we should note that the test set is highly imbalanced, there are only very few fraud cases and rest of them are non-fraud. We dont know which transactionsa are fraud or not and thus can not downsample or upsample. When I did all the classification modelling on imbalanced training and test on imbalanced test set, I got zero recall (basically model predicts everything as non-fraud). If I do modelling on resampled data and test on imbalanced it will also not work because we are simply violating the first principle of machine learning: Training and Test data must come from same distribution.

To deal with imbalanced dataset, we can not use usual machine learning techniques. However, there are some specialized machine learning techniques we can use for imbalanced dataset. In this project I have used two of these algorithms called `Local Outliers Factor (LOF)` and `Isolation Forest`. These algorithms does not give results as good as resampling methods but they have advantage that we can test these algorithms on imbalanced dataset. Out of 98 frauds in my test set, using `LOF` gave me 3 correct frauds and `Isolation Forest` gave me 25 correct frauds. In comparison, all of the usual machine learning methods gave me 0 correct frauds out of 98 frauds in imbalanced dataset.

Update After doing usual machine learning algorithms, I again did a follow up with gradient boosting methods. Scikiit learn has a simple boosting algorithm `GradientBoostingClassifier` and there are other specialized libraries just to do the boosting algorithms. Out of those libraries I used `xgboost`, `lightgbm`, and `catboost` classifiers.

Results For Imbalanced data

```
After train-test split, we have total frauds = 98
0 is non-fraud, 1 is fraud.
Predicting fraud as non-fraud is serious issue. (predicting 1 as 0 is bad
FN is bad)
------------------------------------------------------------------------
------------
Model                         FN (Frauds predicted not frauds)
Local Outlier Factor          95 (95 wrong out of 98)
Isolation Forest              73 (73 wrong out of 98)
XGBoost                       28 (30 wrong out of 98)
Linear Discriminant Analysis  27 (27 wrong out of 98)
LightGBM grid search          27 (19 wrong out of 98)
Catboost grid search          21 (21 wrong out of 98)
```

Background of the Data

- Data Source: https://www.kaggle.com/mlg-ulb/creditcardfraud

The datasets contains transactions made by credit cards in September 2013 by european cardholders.

This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

The dataset is highly imbalanced, the positive class (frauds) account for 0.172% of all transactions.

Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.

Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning.

Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Assumptions:

- Here we have data for just two days, but we assume the data is representative of the whole population of credit card transactions.
- We assume the 28 variables V1-V28 are obtained from correct method of PCA and scaled properly.

Business Problem

When we purchase something using credit card we do not want anybody else illegally buy stuffs from our account by fraud and want to detect the fraudulent activities.

From the given features (V1-V28, Amount, Time) we will build a model that will detect whether the transaction is fraudulent or not.

Here, Class is 1 means, fraud case and 0 means non-fraud case. Out of 1000 transactions, only 2 cases are fraud and other 998 cases are non-fraud. This means the dataset is highly imbalanced.

For the imbalanced dataset, we can not use the usual accuracy metric to assess the performance of the model. If we simply build a model to predict all the transactions as non-fraud we will get 99.8% accuracy, which looks extremely good but its totally useless here since it failed to detect a single fraud case.

We are interested in fraud case not in the accuracy of the model. So, we can use the RECALL as the metric of the model evaluation.

Recall is `Recall = TP / (TP + FN)`. Here, False Negative is detecting a fraud as non-fraud. This case is more important than detecting non-fraud as fraud so we use `Recall` instead of `Precision` where other case would be much important such as in spam email detection.

We can also use AUPRC (Area Under Precision-Recall Curve) to assess the performance of the model.

For the imbalanced dataset we can also use MCC (Matthews Correlation Coefficient)

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC is given by:

Accuracy and F1-score are given by following formula:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \qquad \text{F1 score} = \frac{2TP}{2TP + FP + FN}$$

Best accuracy is 1 worst is 0.
Best F1 score is 1 worst is 0.
Best MCC score is +1 and worst is -1.

Consider the following case:

```
TP = 95, FP = 5; TN = 0, FN = 0.

We get:
accuracy = 95%
F1 score = 97.44%
MCC = Undefined.
```

Another example:

```
TP = 90, FP = 4; TN = 1, FN = 5
accuracy = 91%
F1 score = 95.24%
MCC = 0.14
```

Also, we should note that F1-score depends on which class is defined as positive and which class is negative.

```
TP = 95, FP = 5; TN = 0, FN = 0 ==> F1 = 97.44
TP = 0, FP = 0; TN = 5, FN = 95 ==> F1 = 0
```

MCC does not depend on which class is defined as positive and which class is negative.

About the confusion matrix:

```
tn,fp,fn,fp = confusion_matrix(ytest,ypreds).ravel()
recall = tp / (tp+fn)


                         Precision = TP (TP/FP)   Precision has all three
P's
                         ^
                         |
              Predicted  |
Correct 0     0(TN)   1(FP)
        1     0(FN)   1(TP) ---> recall = TP / (TP + FN)   Recall has one
Negative
                                Out of all 10 last true anniversaries,
how many did I recall correctly?

0 is non-fraud, non-spam
1 is fraud    , spam

# We take 1 as fraud or spam cases.
FP ==> non-spam detected as spam      (but we want to send all non-spam
emails)
FN ==> fraud    detected as non-fraud (but we want to find all frauds)
```

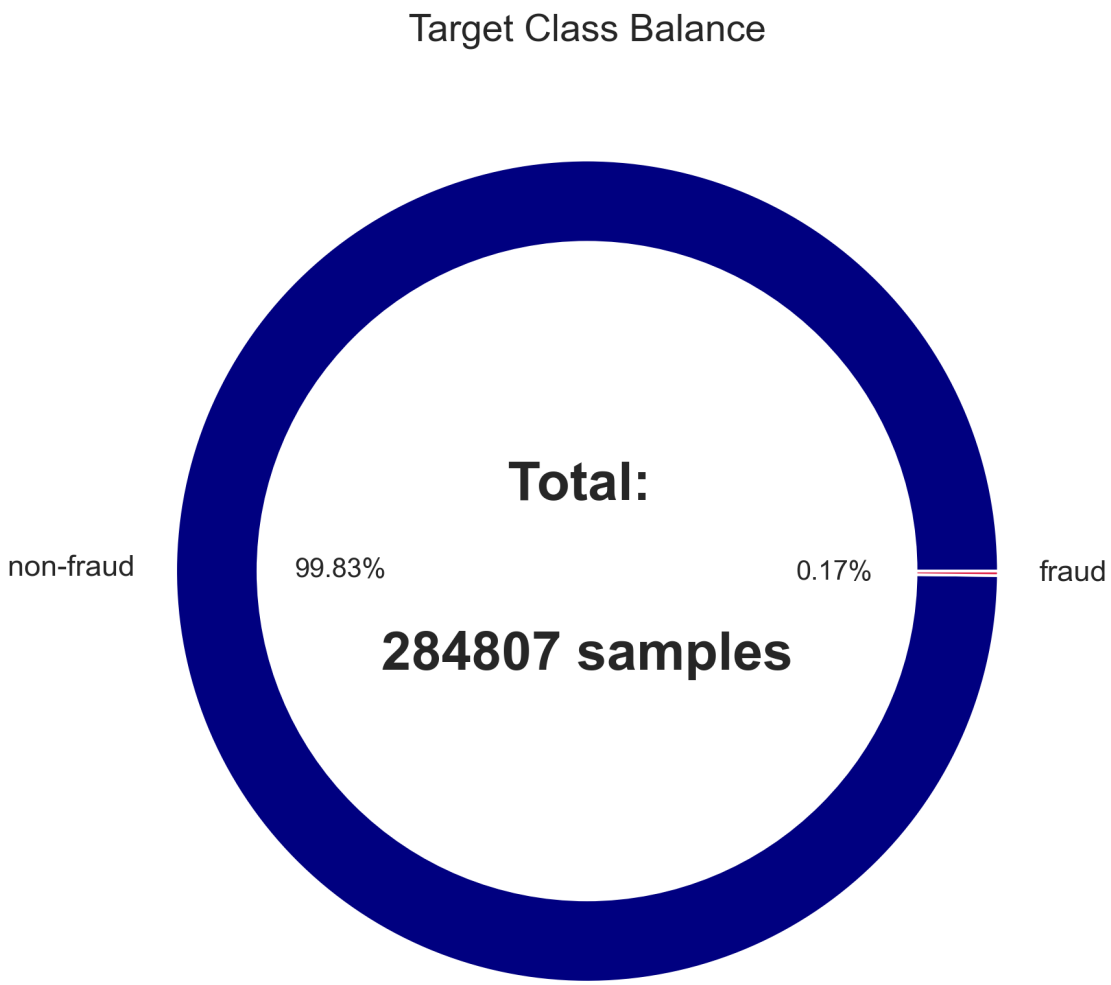
Confusion Matrix		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Here I am interested in the quantity False Negative, I want to make it as small as possible. (Ideally zero.) And True positive as high as possible.

Get to know with the data (EDA)

Class Balance

As we read in the description our dataset is heavily imbalanced. There are about 2 frauds in 1000 transactions.

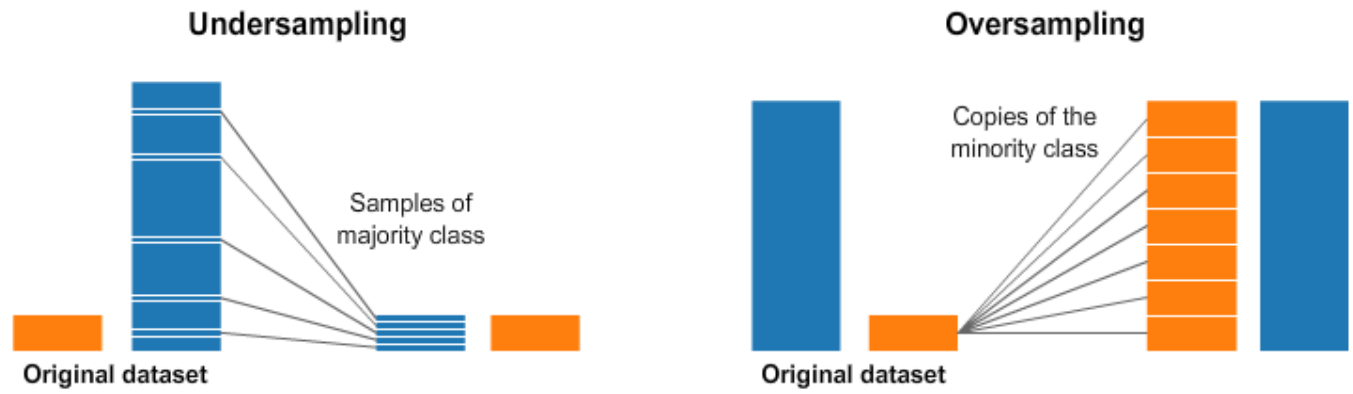


If we run classification models (e.g. Logistic regression, Random Forest Regressor) they will wrongly classify all the cases as non-frauds and get accuracy of 99% but will fail to classify correctly any of the fraud cases.

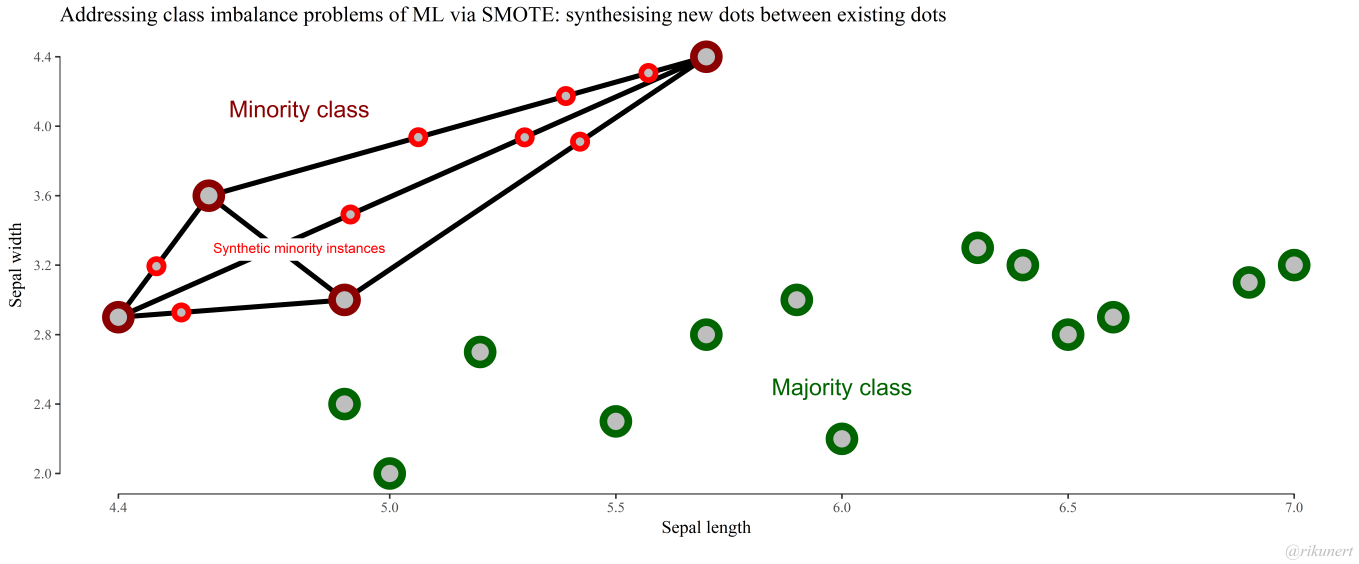
We have two choices here, either undersample the data or over-sample the data. Most common method is undersampling. It makes the model run fast but at the cost of losing lots of lots of data points. Here in this project, if we undersample the data, we will get 492 fraud cases. If we just take random sample of another 492 non-frauds our data will have roughly 1000 samples. Note that previously we had 285,000 samples.

Another way of dealing with imbalanced dataset is oversampling. One of the popular over-sampling method is called SMOTE (Synthetically Modified Oversampling Technique).

Resampling undersampling and oversampling can be described using this image



SMOTE Oversampling will synthesize minority classes and matches with the majority classes.

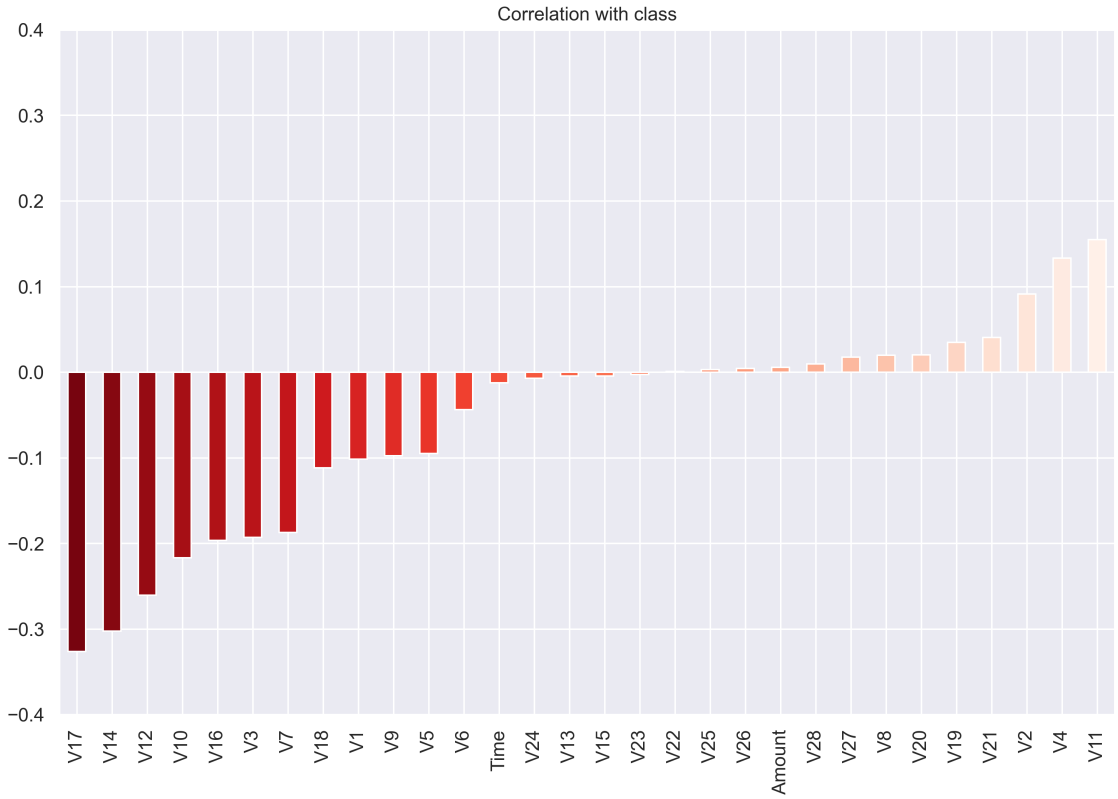


For SMOTE sampling in python we can use imbalanced-learn library which will oversample the minority classes and creates a large dataset. In our project we have 284k non-frauds majority class and 500 samples of fraud minority case. 284k samples of fraud cases will be synthesized using the SMOTE algorithm and finally we will have 568k samples.

Correlations

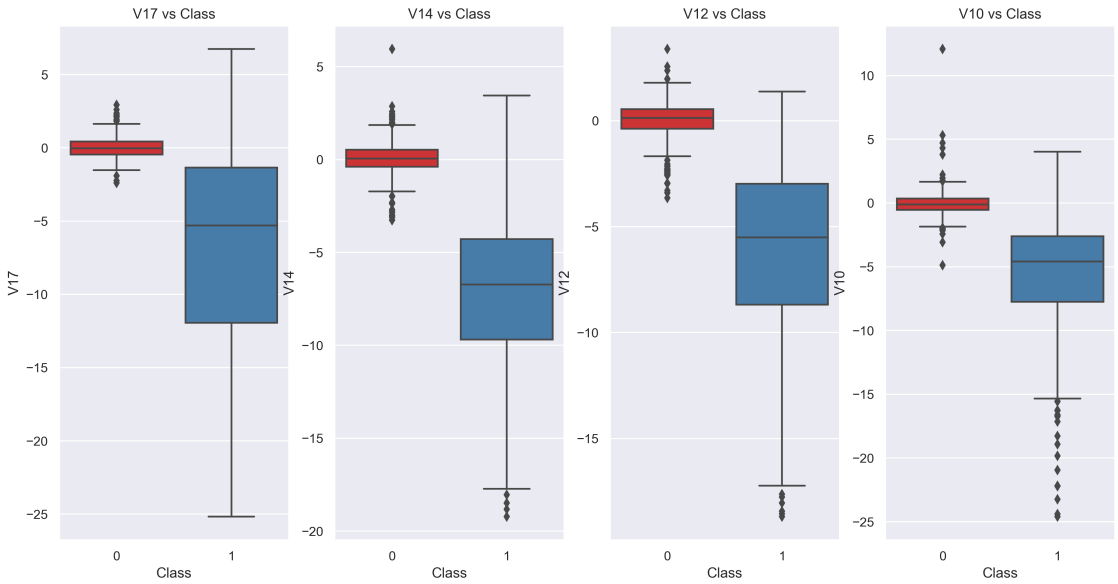
Here we look at the how the features are correlated with the target. I found that feature V17 and V14 is negatively correlated with Target and feature V11 and V4 are positively correlated with Target. This means if we

increase the value of feature V17 we will see less fraudulent cases and vice versa.



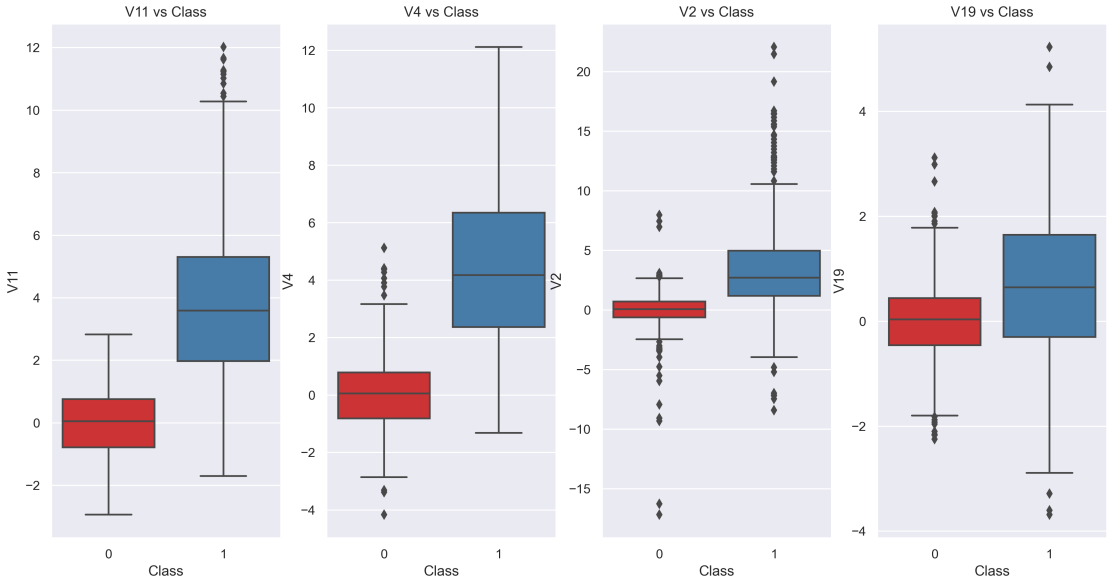
Negative Corelation:

For the Class 1 (Fraud case), we can see that feature V17 has median -5, which is less than zero.

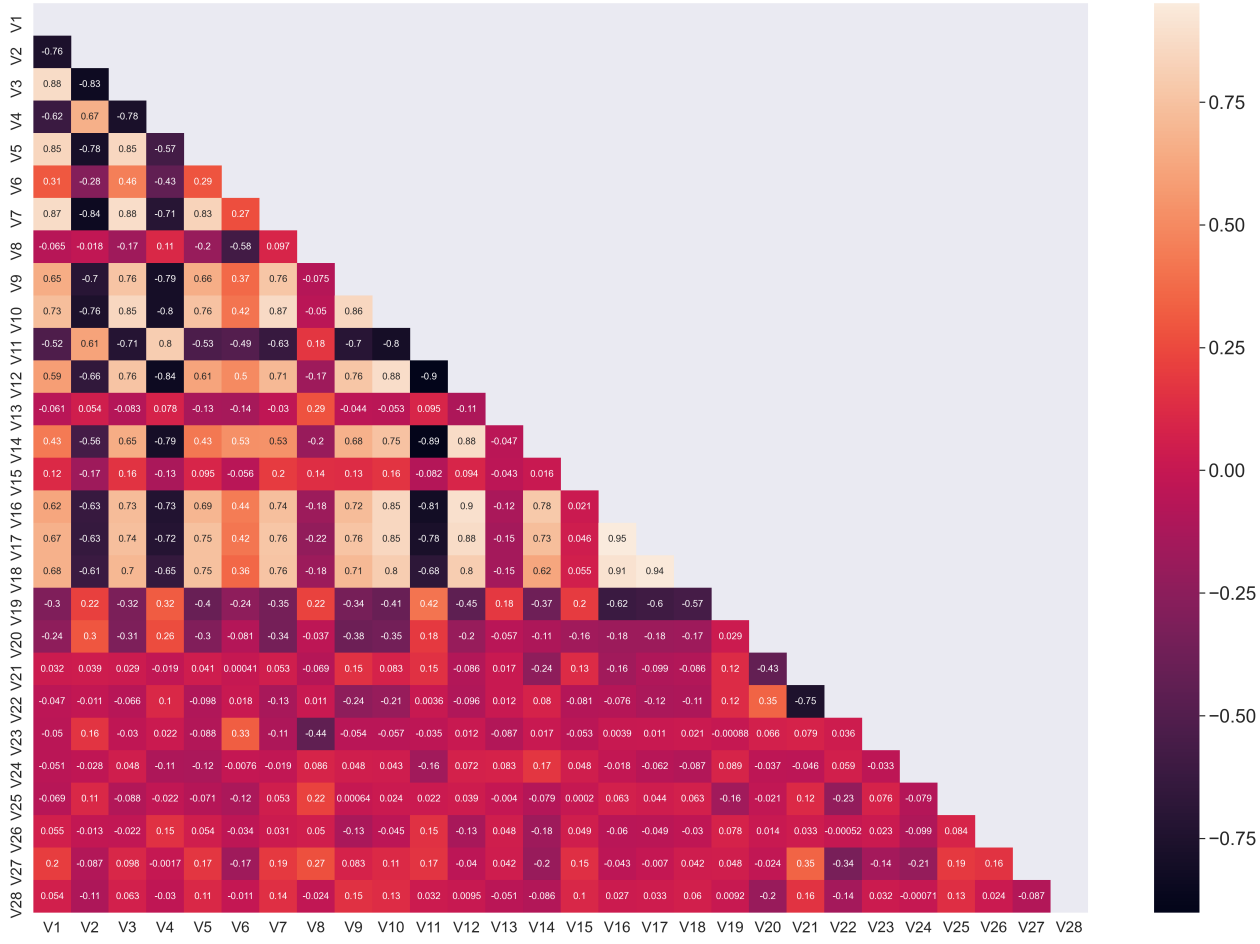


Positive Corelation:

For the Class 1 (Fraud case), we can see that feature V11 has median about +4, which is greater than zero.



Sometimes we see collinearity between two features. If two features are collinear, they have high correlation coefficient between. If one feature increases another feature also increases and model will have difficulty which feature feature causes the changes in Target variable. If any two features are highly correlated (eg. r > 0.9) one the feature may be dropped from the dataset. Then there is a question which feature to keep and which feature to drop. We look at the correlation both features with the Target and keep the feature having larger correlation. The correlation plot among the PCA transformed variables V1-V28 is given below. This shows there is not much collinearity among the features.



Histograms of Features

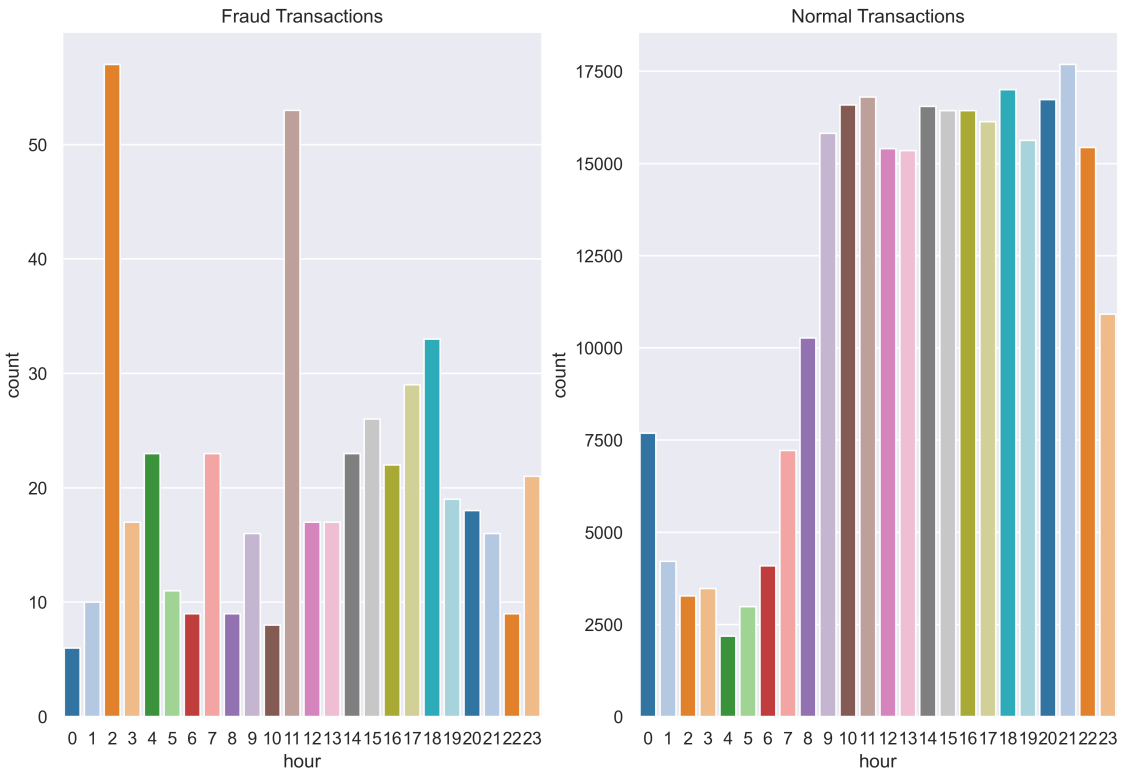
For the linear models we have an underlying assumption that the features are independent of each others and they are all distributed like Gaussian. Histogram is one of the way inspecting wheter the features look like Gaussian distributed or not.

Here, features V1-V28 are already PCA transformed, I will not change them. But the feature `Amount` can be log transformed to make it look like more Gaussian. Also the feature `Time` can be log transformed to make it look like more Gaussian.

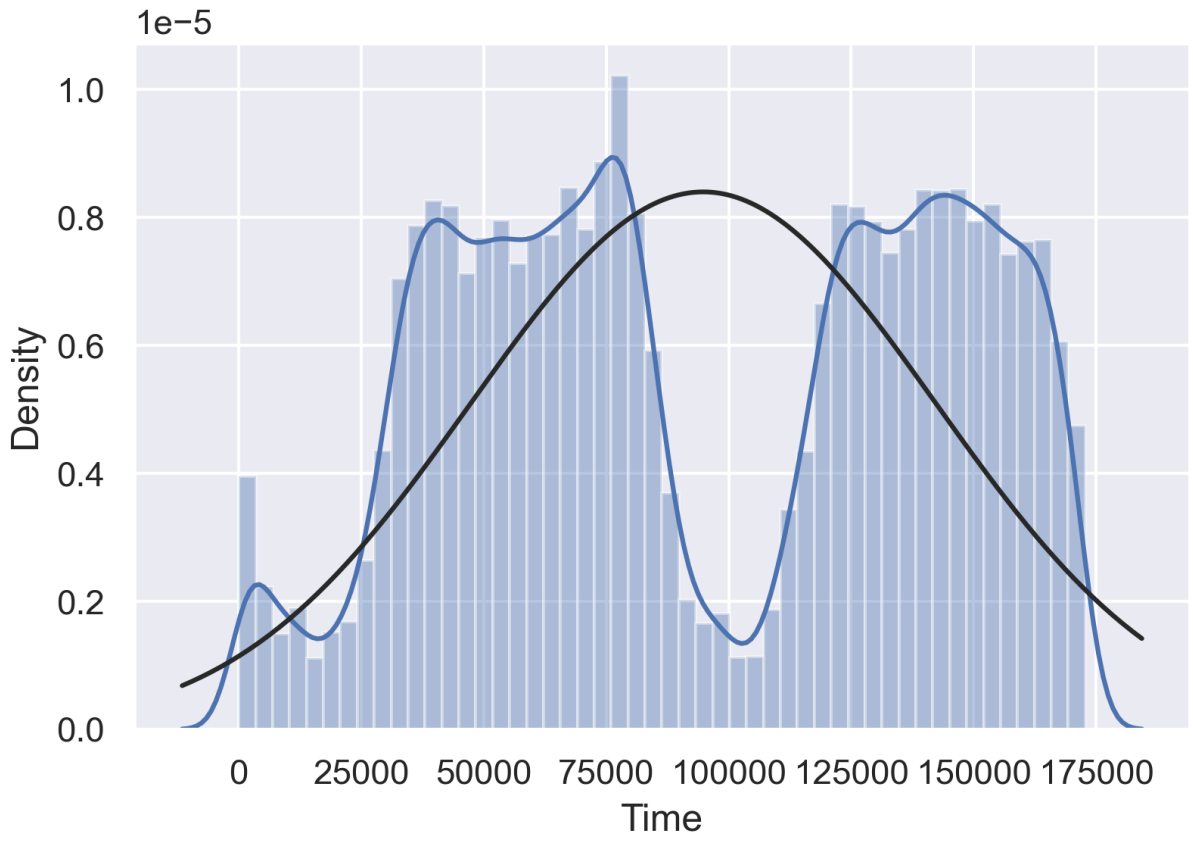
Temporal Feature Study

Here, in this dataset we have one temporal variable called `Time`. This gives the time elapsed in seconds after the first transaction occur. The dataset says the data is from September of 2013, but it does not say which day it is and what is the first transaction time. If I assume, the first transaction time is 7AM, then the peak of fraud

activity happened after two hours (9AM) and 11 hours (6PM).



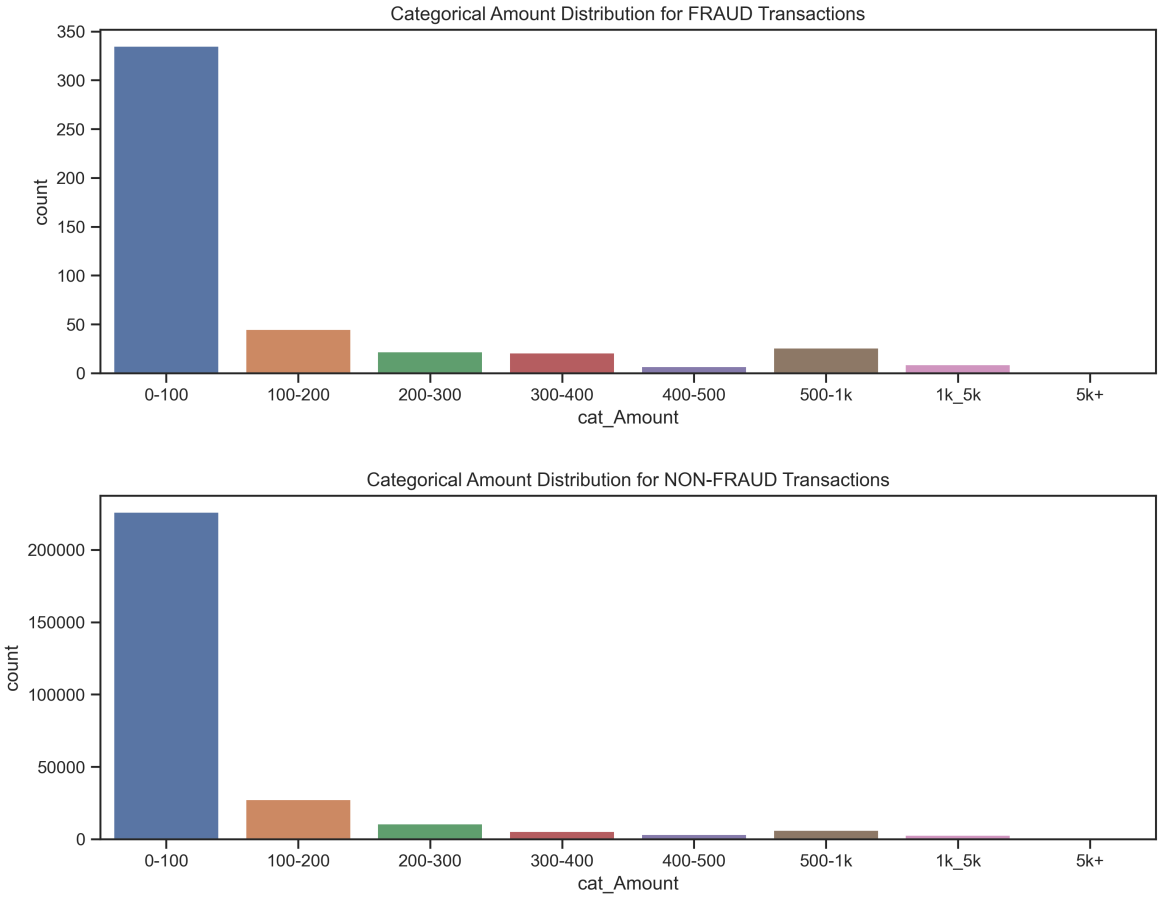
Distibution Plot of Time:



We can clearly see the bimodal distribution, this is because we have data of two days and distribution of transactions is somewhat similar in both of the days.

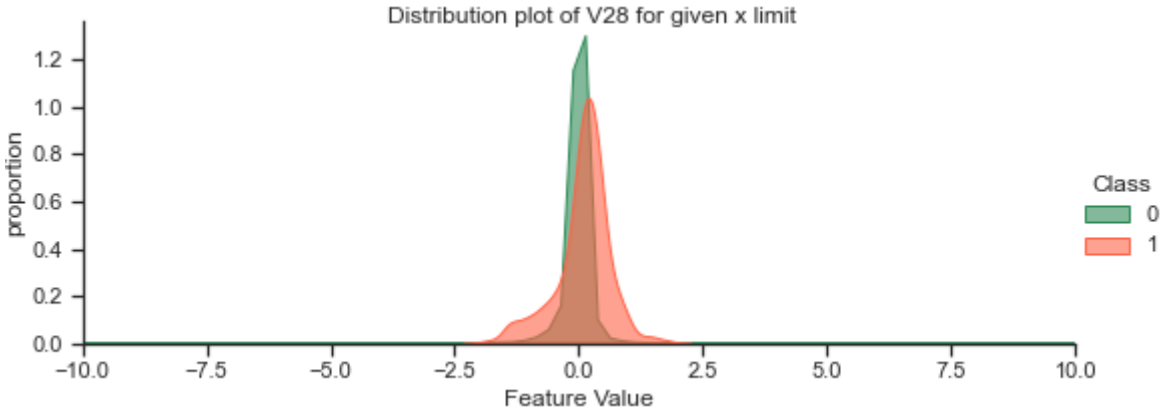
Amount vs Target

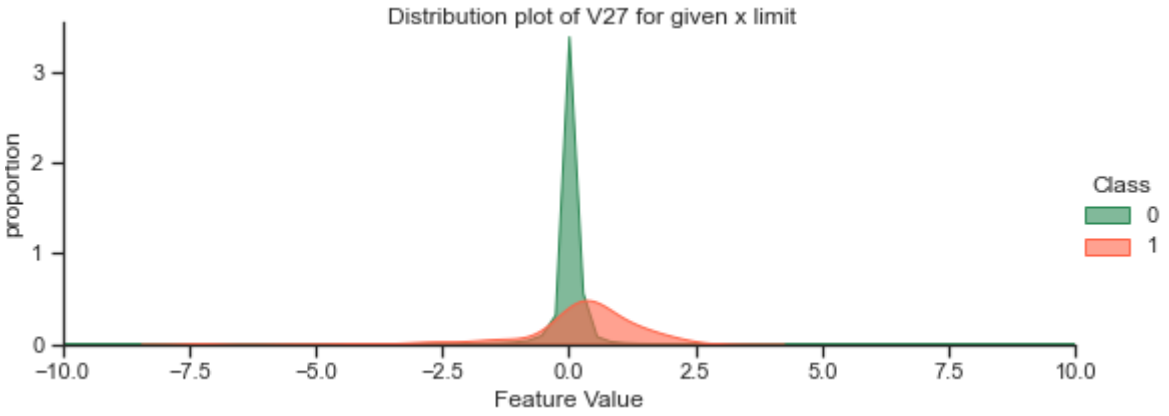
Here we study the distribution of feature `Amount` and its relation with Target. We see that most of transactions are below $100 for both fraud and non-fraud transactions.



Distribution Plots of Features

Our data set does not have too many features, it has 28 PCA transformed features and two usual features. We can look at the distribution plot of each features for the class distribution of fraud and non-fraud and if the distribution of these two classes are same we may want to drop this feature from the entire dataset.





For example, for the features V28 and V27, the class distribution are more or less same for fraud and non-fraud classes and we may drop these features. But, when I looked at correlation of these features with Target, they have considerable correlation and I am planning to keep them for the analysis.

Statistics

In statistics we sometimes are interested in the moments of the variables. First four degrees of moments of a random variable has names: `mean`, `variance`, `skewness`, `kurtosis` and other moments are simply called moment of order n.

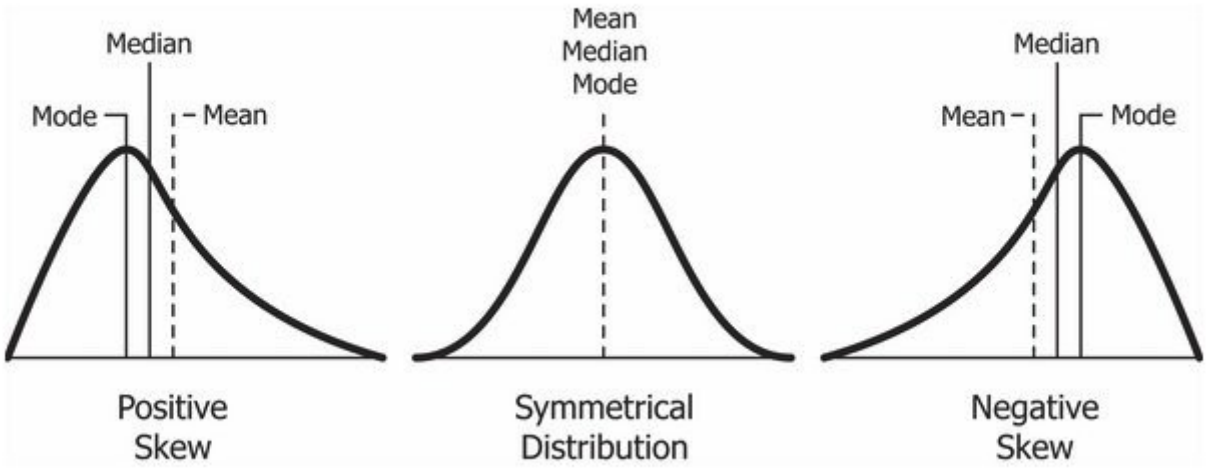
Moment number	Name	Measure of	Formula
1	Mean	Central tendency	$\bar{X} = \frac{\sum_{i=1}^{N} X_i}{N}$
2	Variance (Volatility)	Dispersion	$\sigma^2 = \frac{\sum_{i=1}^{N}(X_i - \bar{X})^2}{N}$
3	Skewness	Symmetry (Positive or Negative)	$Skew = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{(X_i - \bar{X})}{\sigma}\right]^3$
4	Kurtosis	Shape (Tall or flat)	$Kurt = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{(X_i - \bar{X})}{\sigma}\right]^4$

Where X is a random variable having N observations (i = 1,2,…,N).

The first order moment is called mean. For normal distribution mean is zero and variance is 1.

The third order moment is called skewness. Normal distribution has skewness of 0. If the skewness is less than 0, it is negatively skewed and has tail on left side. If the skewness is greater than 0, it is positively skewed and has tail on right side. In our dataset, some of the features are skewed. For example: feature V8 is negatively

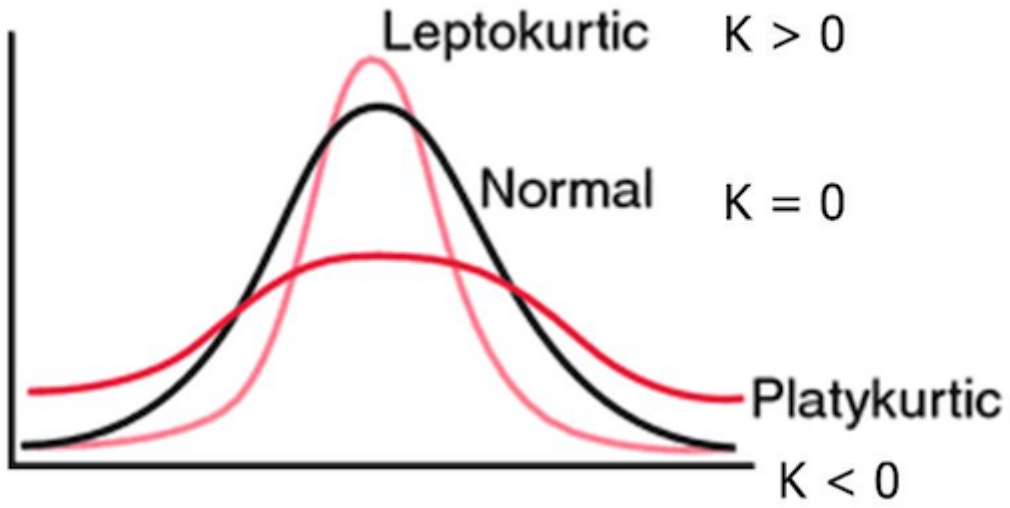
skewed and feature V28 is positively skewed.



If we have skewed features, then they do not follow the Gaussian distribution,and underlying assumptions of linear regression is violated. To reduce the skewness we can do log transformation (or sqrt transformation or boxcox transformation and other transformations). Here, the features are already from PCA transformation, so I am keeping them as it is.

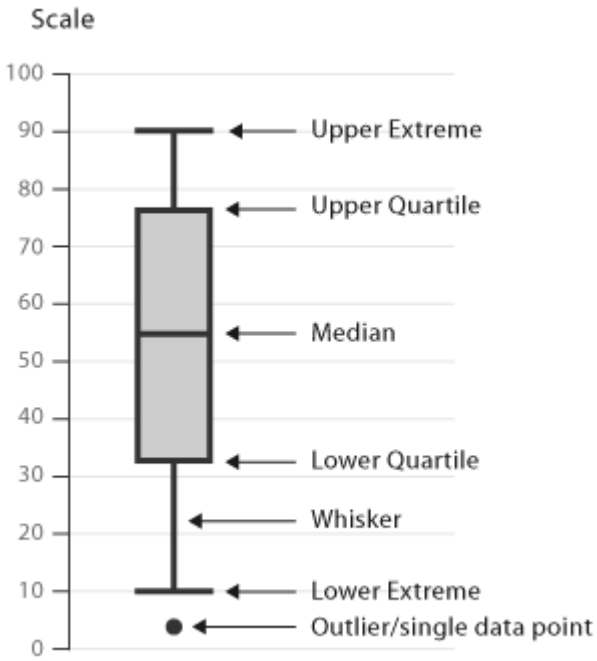
Kurtosis is related to the fourth moment. For Normal distribution, kurtosis has a value of 3. If kurtosis > 3 it is called lepto-kurtic and it is tall peaked and on the other hand if kurtosis is less than 3, it is called platy-kurtic and looks flat. Normal distribution has other name meso-kurtic.

In our dataset some of the features have high kurtosis values. This indicates that there might be some outliers in these features. For example feature V28 has very high kurtosis value and it also has high skewness value. It might have some outliers.

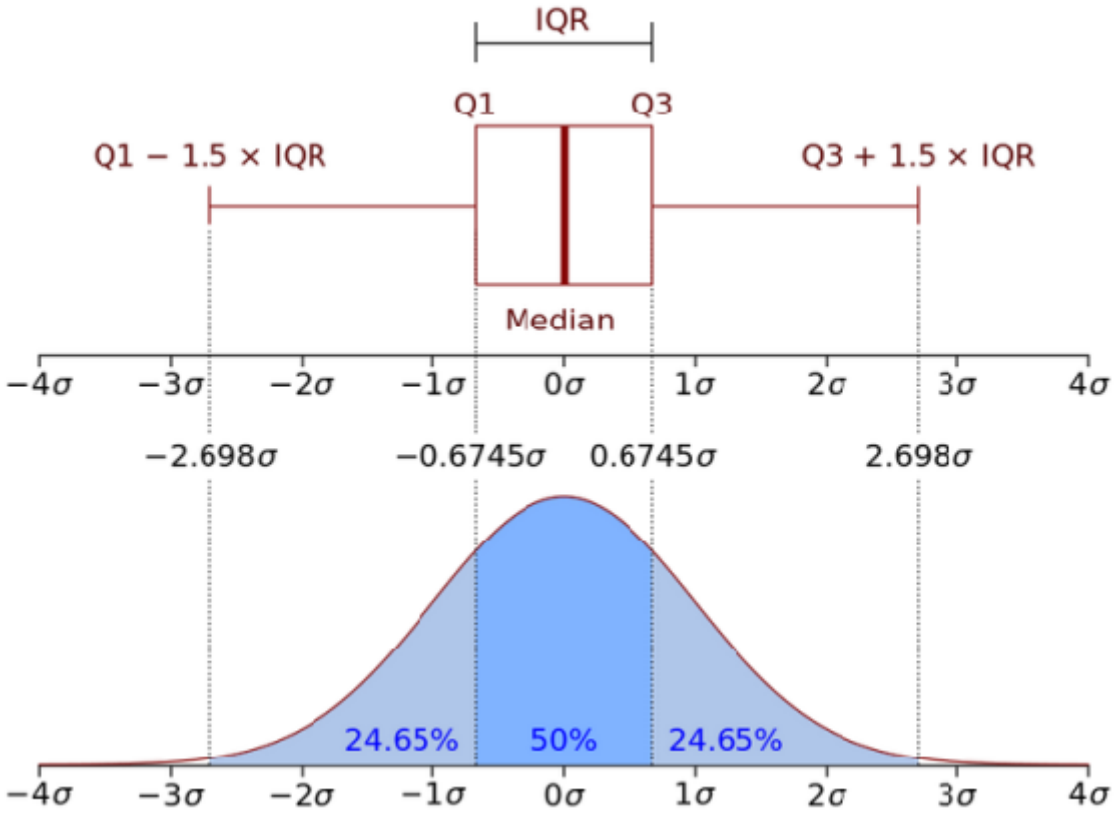


Look at the feature V28.

```
skewness = 11.2
kurtosis = 933.4
```



Lets look at the boxplot:



We can see that lots are possible outliers present in feature V28. Outliers distort the model's ability to fit the data properly and sometimes if we are sure these are the true outliers, we should remove these outliers.

Now, lets see the interquartile ranges of PCA features: We can see feature V1 has large IQR range.

Let's also see the standard deviation. IQR looks similar to standard deviations. We can say that there might not be too many outliers.

Finally, we can also look at all the features how their boxplots looks like:

Modelling Various Classifiers

I am mostly interested in designing a classifier that will give me the lowest False Negative values. Or, which will give me the highest Recall. Accuracy is not the concern here. Since our dataset is heavily imbalanced, first I undersampled the dataset and tested various machine learning classifier models.

False Negative Frauds Detection using Default Classifiers for Undersampled Data

	Model	Description	Total_Frauds	Incorrect_Frauds	Incorrect_Percent
0	Support Vector Classifier	Undersample	91	12	13.19 %
1	Random Forest Classifier	Undersample	91	17	18.68 %
2	Decision Tree Classifier	Undersample	91	19	20.88 %
3	Logistic Regression	Undersample	91	39	42.86 %
4	KNN	Undersample	91	61	67.03 %

Then I did grid search to find the best hyperparameters for all these models.

False Negative Frauds Detection using Classifiers with Grid Search for Undersampled Data

	Model	Description	Total_Frauds	Incorrect_Frauds	Incorrect_Percent
0	Logistic Regression	Undersample, grid search	91	5	5.49 %
1	Support Vector Classifier	Undersample, grid search	91	11	12.09 %
2	Random Forest Classifier	Undersample, grid search	91	13	14.29 %
3	KNN	Undersample, grid search	91	15	16.48 %
4	Decision Tree Classifier	Undersample, grid search	91	16	17.58 %

Recall for all Classifiers with Grid Search for Undersampled Data

	Model	Description	Accuracy	Precision	Recall	F1	Mathews Correlation Coefficient	Cohens Kappa	Area Under Precision Curve	Area Under ROC Curve
0	Logistic Regression	Undersample, Grid Search	0.741117	0.651515	0.945055	0.7713	0.541913	0.495303	0.95003	0.931474
1	Support Vector Classifier	Undersample, Grid Search	0.741117	0.651515	0.945055	0.7713	0.541913	0.495303	0.962546	0.94464
2	Random Forest Classifier	Undersample, Grid Search	0.913706	0.95122	0.857143	0.901734	0.828738	0.825181	0.981182	0.977918
3	KNN	Undersample, Grid Search	0.888325	0.915663	0.835165	0.873563	0.776569	0.773941	0.903222	0.93163
4	Decision Tree Classifier	Undersample, Grid Search	0.898477	0.949367	0.824176	0.882353	0.79999	0.793847	0.903238	0.931371

Classification Report for all Classifiers with Grid Search for Undersampled Data

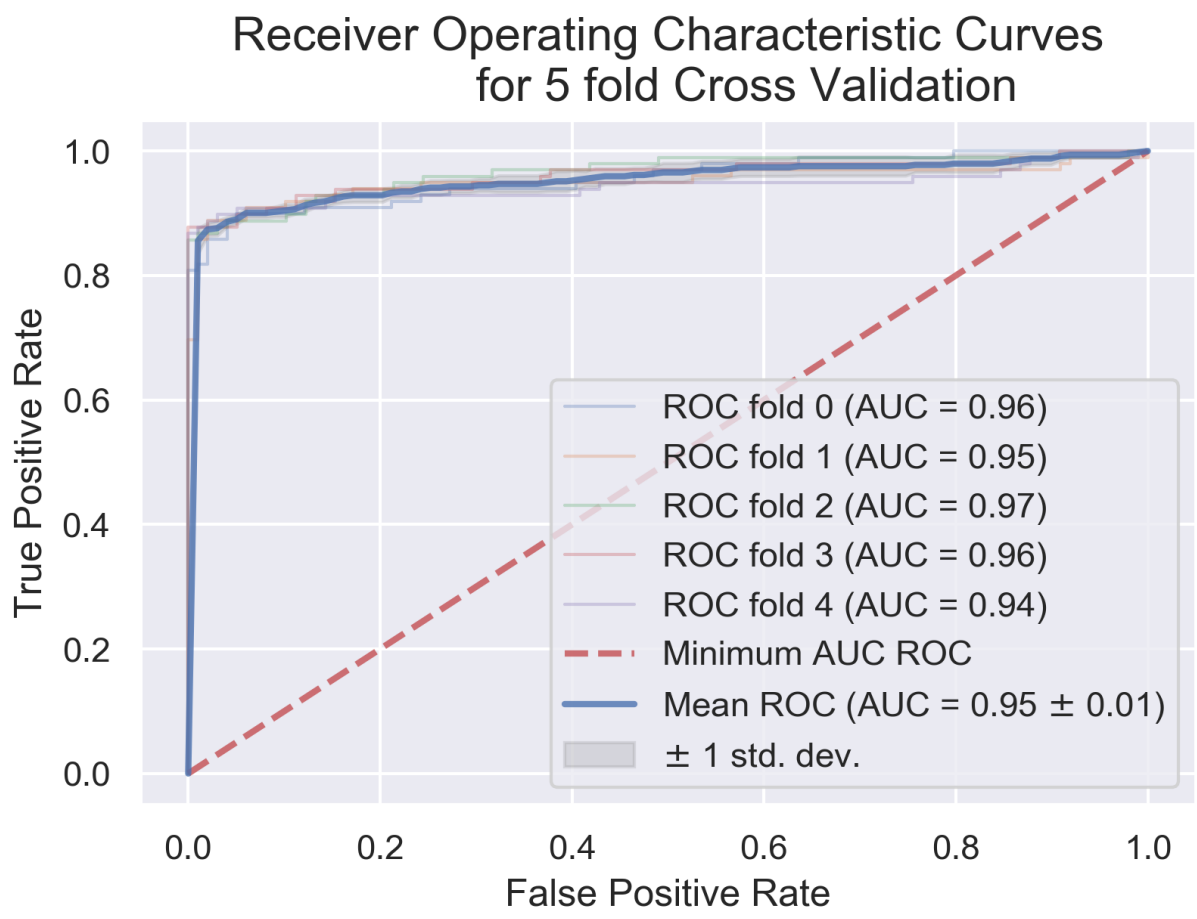
	Model	Desc	Precision_0	Precision_1	Recall_0	Recall_1	F1_Score_0	F1_Score_1	Support_0	Support_1
0	Logistic Regression	Undersample, Grid Search	0.923077	0.651515	0.566038	0.945055	0.701754	0.7713	106	91
1	Support Vector Classifier	Undersample, Grid Search	0.901786	0.941176	0.95283	0.879121	0.926606	0.909091	106	91
4	Random Forest Classifier	Undersample, Grid Search	0.886957	0.95122	0.962264	0.857143	0.923077	0.901734	106	91
2	KNN	Undersample, Grid Search	0.868421	0.915663	0.933962	0.835165	0.9	0.873563	106	91
3	Decision Tree Classifier	Undersample, Grid Search	0.864407	0.949367	0.962264	0.824176	0.910714	0.882353	106	91

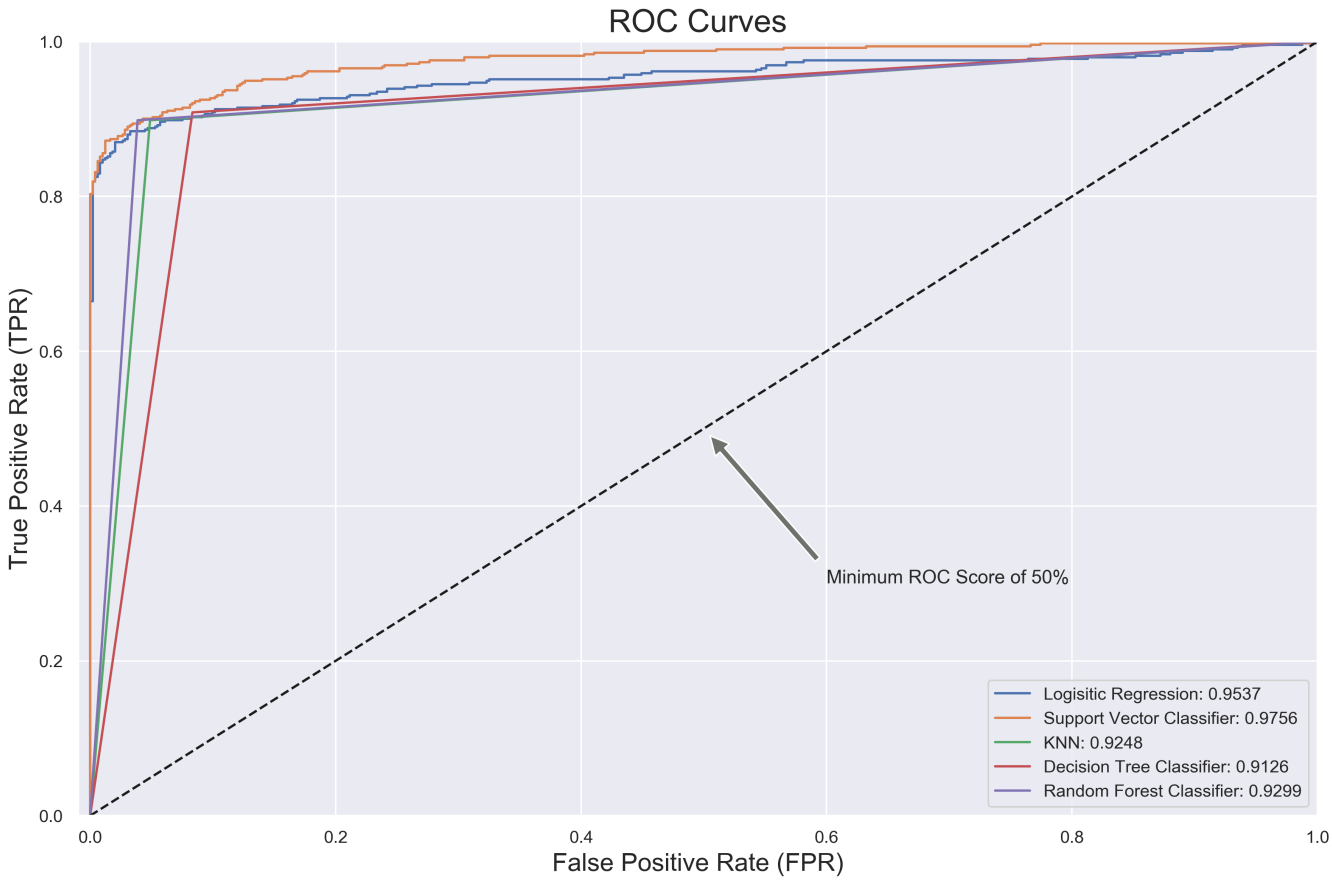
Best Model from Grid Search for Underampled Data

I found that logistic regression is the best model for classification for this dataset.

Logistic Regression, Undersample, Grid Search

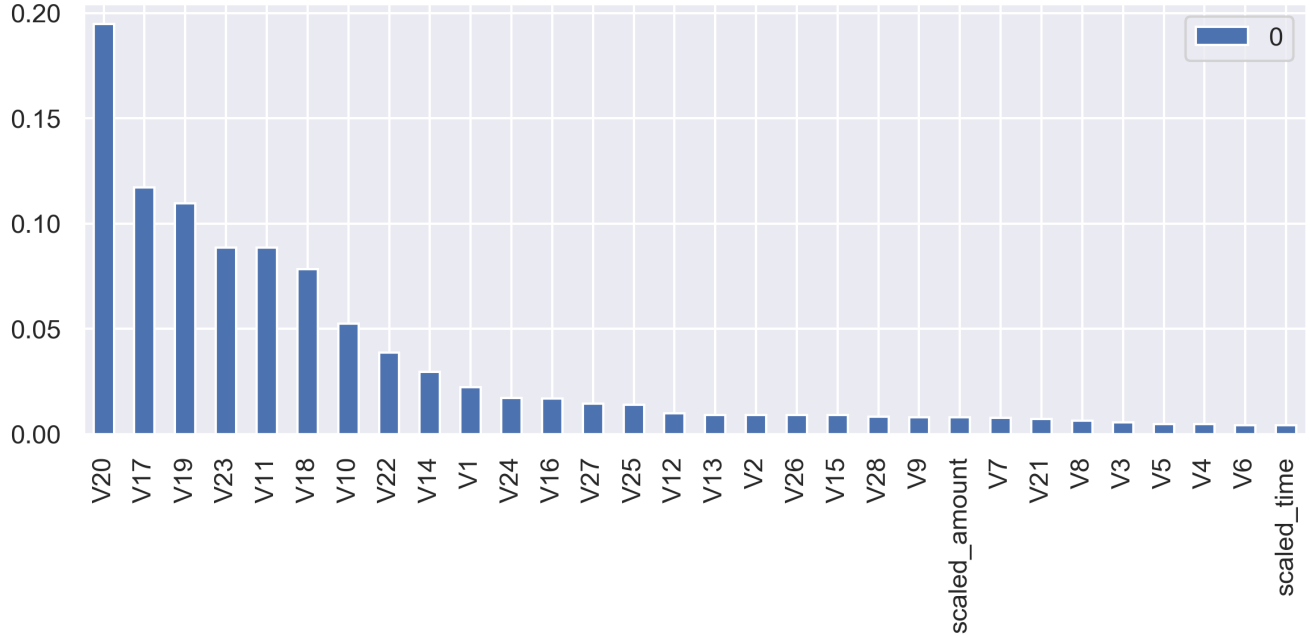
	Predicted_No_Fraud	Predicted_Fraud	Total_Frauds	Correct_Frauds	Incorrect_Frauds	Fraud_Detection
No_Fraud	60	46	91	86	5	94.51%
Fraud	5	86	91	86	5	94.51%





Random Forest Classifier Underampled Data

One nice property of Random Forest is that, it gives the most important features in the dataset. For Undersampled dataset, I found V20 was the most important feature.



Random Forest Classifier, Undersample, Grid Search

	Predicted_No_Fraud	Predicted_Fraud	Total_Frauds	Correct_Frauds	Incorrect_Frauds	Fraud_Detection
No_Fraud	102	4	91	78	13	85.71%
Fraud	13	78	91	78	13	85.71%

Detail Study of Logistic Regression with SMOTE Oversampling

Undersampling only gives results good for undersampled test set since in machine learning modelling train and test must come same distribution. Undersampling is a good way to get better results if we already know the class labels. But in fraud detection we only know the fitting parameters but we dont know whether the transactions is normal or fraudulent in advance. So our test set must be imbalanced.

When I tested the logistic regression fitted from underampled training data, it gives ZERO recall for imbalanced test set.

We need to oversample our dataset. One of the most popular resampling method to get oversampled data is SMOTE. Here, I used SMOTE algorithm and get the Recall of 0.42857 for the imbalanced test set.

Logistic Regression Model Evaluation Scalar Metrics

	Model	Description	Accuracy	Precision	Recall	F1	Mathews Correlation Coefficient	Cohens Kappa	Area Under Precision Curve	Area Under ROC Curve
0	Logistic Regression	Train Test Undersample, Grid Search	0.923858	0.910891	0.938776	0.924623	0.848129	0.847735	0.985611	0.97918
1	Logistic Regression	Train Test Undersample	0.944162	0.957895	0.928571	0.943005	0.888717	0.888305	0.991175	0.989281
2	Logistic Regression	Train Oversample SMOTE, Test Imbalanced	0.796831	0.00029432	0.428571	0.000588235	0.00661815	0.00030949	0.00142026	0.715063
3	Logistic Regression	Train Oversample SMOTE, Test Imbalanced, Grid Search from Undersample	0.875775	0.000481386	0.428571	0.000961693	0.0109009	0.000683173	0.00369781	0.730713
4	Logistic Regression Polynomial deg 2	Train Oversample SMOTE, Test Imbalanced, Grid Search from Undersample	0.875775	0.000481386	0.428571	0.000961693	0.0109009	0.000683173	0.00369781	0.730713
5	Logistic Regression	Train Test Imbalanced	0.99986	0	0	0	0	0	0.00305379	0.624277
6	Logistic Regression	Train Test Imbalanced, Grid Search	0.99986	0	0	0	0	0	0.000155879	0.466225
7	Logistic Regression	Train Undersample, Test Imbalanced	0.999482	0	0	0	-0.000229906	-0.000203943	0.000139512	0.492705

Logistic Regression Model Evaluation Classification Metrics

	Model	Description	Precision_0	Precision_1	Recall_0	Recall_1	F1_Score_0	F1_Score_1	Support_0	Support_1
0	Logistic Regression	Train Test Undersample, Grid Search	0.9375	0.910891	0.909091	0.938776	0.923077	0.924623	99	98
1	Logistic Regression	Train Test Undersample	0.931373	0.957895	0.959596	0.928571	0.945274	0.943005	99	98
2	Logistic Regression	Train Oversample SMOTE, Test Imbalanced	0.9999	0.00029432	0.796882	0.428571	0.886922	0.000588235	50168	7
3	Logistic Regression	Train Oversample SMOTE, Test Imbalanced, Grid Search from Undersample	0.999909	0.000481386	0.875837	0.428571	0.93377	0.000961693	50168	7
4	Logistic Regression Polynomial deg 2	Train Oversample SMOTE, Test Imbalanced, Grid Search from Undersample	0.999909	0.000481386	0.875837	0.428571	0.93377	0.000961693	50168	7
5	Logistic Regression		0.99986	0	1	0	0.99993	0	50168	7
6	Logistic Regression	Train Test Imbalanced, Grid Search	0.99986	0	1	0	0.99993	0	50168	7
7	Logistic Regression	Train Undersample, Test Imbalanced	0.99986	0	0.999621	0	0.999741	0	50168	7

Logistic Regression Confusion Matrix

Logistic Regression Oversampling SMOTE Grid Search from Undersampling

	Predicted_No_Fraud	Predicted_Fraud	Total_Frauds	Correct_Frauds	Incorrect_Frauds	Fraud_Detection
No_Fraud	43,939	6,229	7	3	4	42.86%
Fraud	4	3	7	3	4	42.86%

Outlier Detection Models: Isolation Forest and Local Outliers Factor (LOF)

Two of the most popular models to use in Outlier Detection are Isolation Forest and Local Outliers Factor. These models work directly on the imbalanced dataset and we do not have to undersample or oversample the training dataset.

One technical note for these two models is that they give class results +1 and -1 and we need to change them to 0 and 1.

```
ypreds[ypreds == 1] = 0
ypreds[ypreds == -1] = 1
ypreds_iso = ypreds
```

Isolation Forest gave me the better result than Local Outlier Factor.

```
Isolation Forest Results
------------------------
Total Frauds:  98
Incorrect Frauds:  73
Incorrect Percent:  74.49 %

Local Outliers Factor
-------------------------
Total Frauds:  98
Incorrect Frauds:  95
Incorrect Percent:  96.94 %
```

Modelling with LightGBM

For the underampled data I also tested the lightGBM model. It is fast to train and gives decent results.

```
lightGBM Results
------------------------
Total Frauds:  98
Incorrect Frauds:  62
Incorrect Percent:  63.27 %
```

After grid search, I got smaller number of errors:

```
LightGBM Grid Search Results
------------------------
Total Frauds:  98
Incorrect Frauds:  19
Incorrect Percent:  19.39 %
```

Deep Learning Methods

If we have large number of samples we can also use deep learning methods to train our model. I got the following results for keras.

```
Keras Imbalaned Data Results
------------------------
Total Frauds:  98
Incorrect Frauds:  94
```

```
Incorrect Percent:  95.92 %

Keras Undersampled Train Test Results
------------------------
Total Frauds:  98
Incorrect Frauds:  7
Incorrect Percent:  7.14 %
```

Big Data Analysis method using PySpark

For the imbalanced dataset I trained differet classifiers using the pyspark module. I got the best result for weighted Recall for random forest classifier after a grid search.

The results are shown below:

	model_name	desc	f1	weightedPrecision	weightedRecall	accuracy	areaUnderROC	areaUnderPR
0	Logistic Regression	log features	0.846066	0.730487	0.999300	0.999294	0.999337	0.999337
3	Decision Tree Classifier	log features	0.868053	0.765536	0.999398	0.999393	0.999424	0.999424
4	Random Forest Classifier	log features	0.890040	0.800147	0.999494	0.999490	0.999511	0.999511
6	Random Forest Classifier	log features, grid search	0.884580	0.836982	0.999541	0.999548	0.999564	0.999564