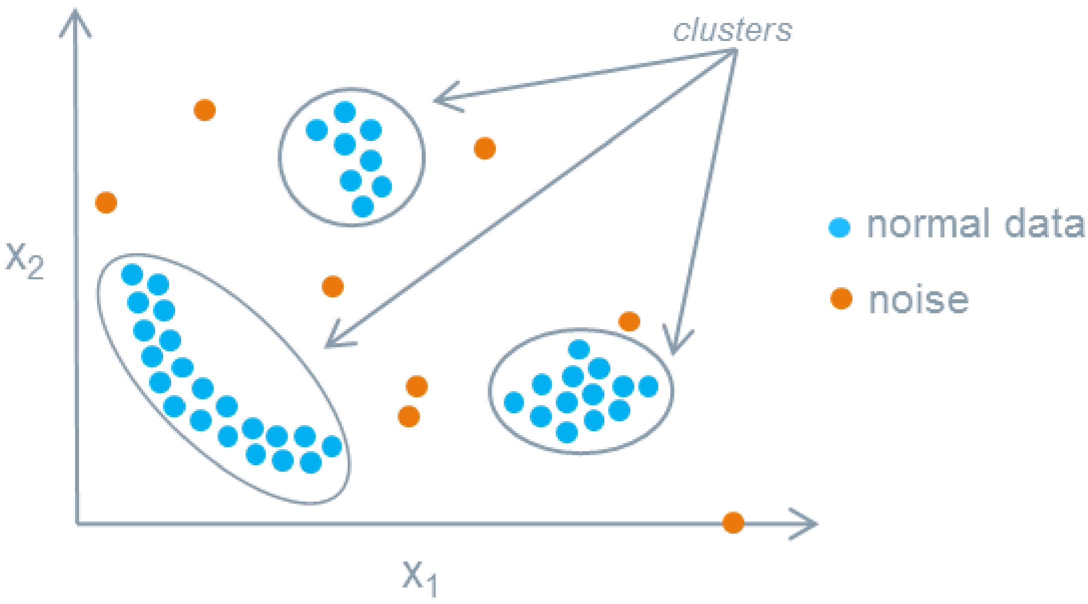
ANAMOLY DETECTION

Anamoly Detection is also known as outlier detection. Let us understand the Outlier in the Laymen language. For instance, you are asked to remove the rotten tomatoes from bucket because if not separated it will also spoil the other good tomatoes.

Similarly, there are variable/features/data points which are of no use or making no difference but could be responsible for greater loss. Thus we need to find the Outliers and remove them for better accuracy.



The noise are the data points which are detected as the outliers.

Anamoly Detection is categorized into three broad categories -

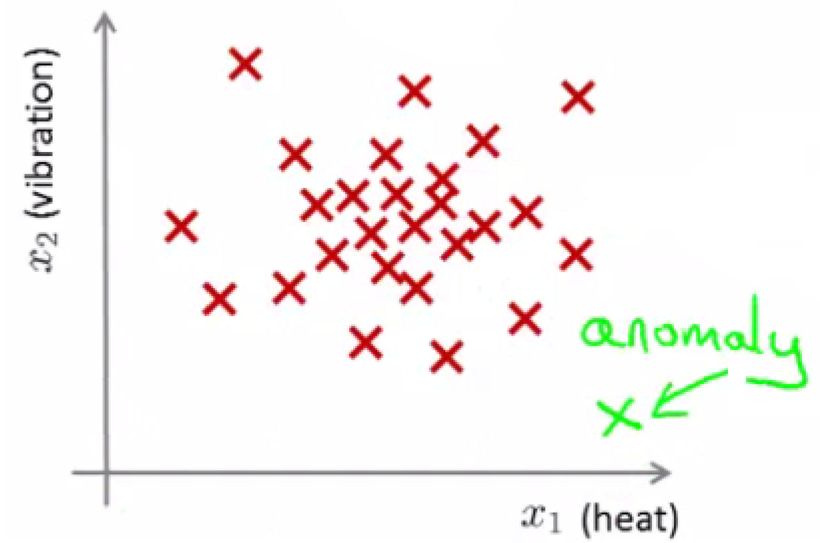
1. **Supervised Anamoly Detection** In Supervised Detection, there is a classifier which classifies whether the data pints is Normal or Abnormal.
2. **Unsupervised Anamoly Detection** It detects the anomalies in the given dataset by assuming that the testing dataset contains the least fit to the remainder of the data set.
3. **Semi-Supervised Anamoly Detection** The training data set to construct the normal behaviour to the model and it checks the test data for the likelihood by the experience the model generated.

Anamolies and is classifications

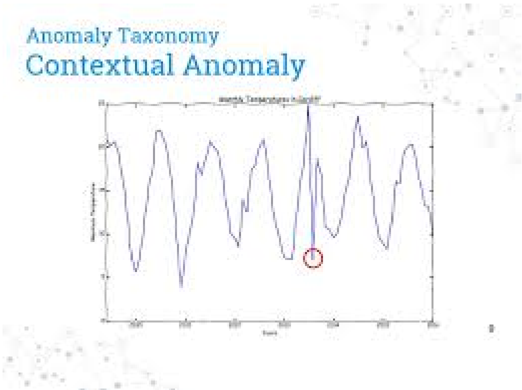
Anamoly is use to identify the rare items, suspcious items, events and outcomes which can raise a harm to the model.

The anamolies have several classifications -

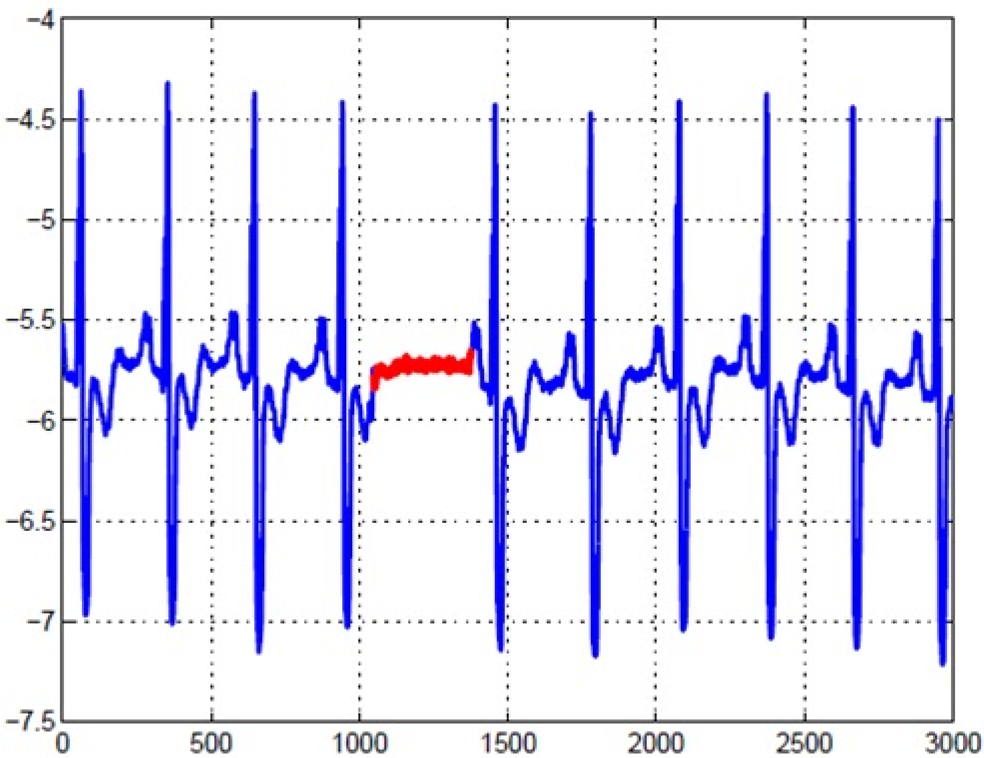
1. **Point anamolies** When a single data point is too far from the rest data points which makes it merely impossible to make the cluster or map it to the data points or cluster then we simply remove such data points. This is called the point anamolies.



2. **Contextual anamolies** If the abnormality is context specific, For instance investing 1000 rupee everyday on buying shoe since you play football is normal, but odd anyway.



3. **Collective anamolies** A set of data instance is responsible to track this anamoly. If someone is remotely using a machine and extracting the information to the local host. It gives the sign of the cyber attack.



Anamoly Detection is similar to Novelty detection but not completly similar. Novelty Detection is mainly concerned of identifying the unobseverd pattern in the observations.

Anamoly Detection Techniques

Simple Statistical Methods - The simple methods to find the irregularities in data points that deviate from common statistical properties of distribution including mean, median, standard deviation, etc.

Anamoly detection techniques

Isolation Forest Anomaly Detection Algorithm

Density-Based Anomaly Detection (Local Outlier Factor)Algorithm

Support Vector Machine Anomaly Detection Algorithm

Applications of Anamoly detection

1. Intrusion Detection
2. Fraud Detection
3. Fault Detection
4. System Health Monitoring
5. Event Detection in networks
6. Detecting Natural disturbances.

Maths used in Anamoly Detection

Anomaly detection algorithm

1. Choose n features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

3. Given new example x , compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

12

Problem Statement

Credit Card Fraud Detection

Data Preprocessing and EDA

Why Data processing and visualization is important ?

It is very important to clean the data(preprocess) before using it to fit the model. The method helps in removing the outliers and make the data standardized.

To understand the data more easily and widely we visualize the data

Now, let us preprocess the data, visualize the data and fit the data into the model.

In [ ]:

```
# importing the libraries

import pandas as pd
import numpy as np
```

[Recaller -]

1. Pandas - It is an open-source library which we can use to manipulate, create or wrangle the data.
2. Numpy - NumPy stands for 'Numerical Python'. It is a python package used to perform scientific computations like performing linear algebra, arranging the data, dropping the data, etc.

In [ ]:

```
#Importing the dataset so that we can use it for the further proceedings

db = pd.read_csv('creditcard.csv', sep=',')
```

Now let us define the basic information to the dataset

In [ ]:

```
# Describing the data which includes the data count, mean, min, max, standard deviatior

db.describe()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	
count	27819.000000	27818.000000	27818.000000	27818.000000	27818.000000	27818.000000	27
mean	20434.634315	-0.217255	0.149360	0.723559	0.221251	-0.199312	
std	11866.057310	1.866645	1.545773	1.648474	1.425213	1.431480	
min	0.000000	-30.552380	-40.978852	-31.103685	-5.172595	-42.147898	
25%	9037.500000	-0.951060	-0.424408	0.271315	-0.690871	-0.788013	
50%	24675.000000	-0.259642	0.163461	0.855090	0.202149	-0.230110	
75%	31319.000000	1.166130	0.803933	1.483404	1.102574	0.316960	
max	34712.000000	1.960497	16.713389	4.101716	13.143668	34.099309	

The description of the dataset with extracting the mean, mode, min and max of all the columns to show the importance of the dataset.

In [ ]:

```
# Getting the information of the dataframe.
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27819 entries, 0 to 27818
Data columns (total 31 columns):
Time      27819 non-null int64
V1        27818 non-null float64
V2        27818 non-null float64
V3        27818 non-null float64
V4        27818 non-null float64
V5        27818 non-null float64
V6        27818 non-null float64
V7        27818 non-null float64
V8        27818 non-null float64
V9        27818 non-null float64
V10       27818 non-null float64
V11       27818 non-null float64
V12       27818 non-null float64
V13       27818 non-null float64
V14       27818 non-null float64
V15       27818 non-null float64
V16       27818 non-null float64
V17       27818 non-null float64
V18       27818 non-null float64
V19       27818 non-null float64
V20       27818 non-null float64
V21       27818 non-null float64
V22       27818 non-null float64
V23       27818 non-null float64
V24       27818 non-null float64
V25       27818 non-null float64
V26       27818 non-null float64
V27       27818 non-null float64
V28       27818 non-null float64
Amount    27818 non-null float64
Class     27818 non-null float64
dtypes: float64(30), int64(1)
memory usage: 6.6 MB
```

Why to handle the missing values ?

- If the missing value is not handled, the programmer would end up with the interpretation of the inaccurate results and thus the model would not fit.

There are several ways of handling the missing values in the data :

1. Remove rows with missing values
2. Set some value for missing values.
3. You can set the median or mean for missing values.

There are several methods to check the missing values

In [ ]:

```
# Checking out the missing values for the dataset so that we can remove it and work fur
total = db.isnull().sum().sort_values(ascending=False)
percent = (db.isnull().sum()/db.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data
```

Out[9]:

	Total	Percent
Class	1	0.000036
V14	1	0.000036
V1	1	0.000036
V2	1	0.000036
V3	1	0.000036
V4	1	0.000036
V5	1	0.000036
V6	1	0.000036
V7	1	0.000036
V8	1	0.000036
V9	1	0.000036
V10	1	0.000036
V11	1	0.000036
V12	1	0.000036
V13	1	0.000036
V15	1	0.000036
Amount	1	0.000036
V16	1	0.000036
V17	1	0.000036
V18	1	0.000036
V19	1	0.000036
V20	1	0.000036
V21	1	0.000036
V22	1	0.000036
V23	1	0.000036
V24	1	0.000036
V25	1	0.000036
V26	1	0.000036
V27	1	0.000036
V28	1	0.000036
Time	0	0.000000

After clearly analyzing the missing value we can remove the last column from the dataset as only one column is given that is time and everything is empty

In [ ]:

```
df = db.drop(db.index[[27818]])
```

Now again checking the missing value

In [ ]:

```
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data
```

Out[16]:

	Total	Percent
Class	0	0.0
V14	0	0.0
V1	0	0.0
V2	0	0.0
V3	0	0.0
V4	0	0.0
V5	0	0.0
V6	0	0.0
V7	0	0.0
V8	0	0.0
V9	0	0.0
V10	0	0.0
V11	0	0.0
V12	0	0.0
V13	0	0.0
V15	0	0.0
Amount	0	0.0
V16	0	0.0
V17	0	0.0
V18	0	0.0
V19	0	0.0
V20	0	0.0
V21	0	0.0
V22	0	0.0
V23	0	0.0
V24	0	0.0
V25	0	0.0
V26	0	0.0
V27	0	0.0
V28	0	0.0
Time	0	0.0

In [ ]:

```
# FInding the data correlation
traindata_corr = df.corr()[:-1]
traindata_corr

```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7
Time	1.000000	0.017843	-0.085133	-0.074388	-0.027062	-0.077892	-0.033042	-0.020945
V1	0.017843	1.000000	-0.194719	0.345856	-0.114341	0.129202	0.117884	0.220005
V2	-0.085133	-0.194719	1.000000	-0.307192	0.130604	-0.180550	-0.024093	-0.086011
V3	-0.074388	0.345856	-0.307192	1.000000	-0.171269	0.346188	0.026216	0.396023
V4	-0.027062	-0.114341	0.130604	-0.171269	1.000000	-0.093218	-0.047014	-0.136110
V5	-0.077892	0.129202	-0.180550	0.346188	-0.093218	1.000000	0.098720	0.103534
V6	-0.033042	0.117884	-0.024093	0.026216	-0.047014	0.098720	1.000000	0.115448
V7	-0.020945	0.220005	-0.086011	0.396023	-0.136110	0.103534	0.115448	1.000000
V8	0.044383	-0.141597	0.075406	-0.336094	0.109543	-0.157343	-0.086550	-0.153243
V9	-0.293857	-0.022197	-0.041766	0.178833	-0.059679	0.042272	0.052875	0.055992
V10	0.095036	0.040906	-0.024396	0.228420	-0.097926	0.172361	0.059299	0.214319
V11	-0.161147	-0.047651	0.110534	-0.149577	0.064598	-0.069125	-0.101456	-0.140587
V12	0.316616	0.068996	-0.127333	0.142124	-0.122874	0.053368	0.003583	0.194259
V13	-0.298307	0.012947	0.049822	0.001660	0.053404	0.044115	0.021916	-0.021938
V14	-0.225374	0.168348	-0.113223	0.268323	-0.091373	0.103103	0.090459	0.113874
V15	0.151200	0.049909	0.051819	-0.165047	-0.120000	0.072669	-0.112564	0.074967
V16	0.035102	0.144262	-0.070947	0.053057	-0.169155	0.134775	0.023914	0.149784
V17	-0.091594	0.119973	-0.095156	0.198082	-0.002422	0.075076	0.038902	0.170686
V18	-0.048760	0.001890	-0.012105	0.051179	-0.031070	0.099324	0.054958	0.107083
V19	0.025286	0.016048	-0.015703	-0.034994	-0.027488	-0.005221	0.095867	-0.047653
V20	0.016103	-0.132026	-0.071731	-0.109565	0.026720	0.006977	-0.023939	-0.031464
V21	0.024056	-0.103010	0.033487	-0.019200	0.005097	-0.049911	0.042070	-0.108226
V22	0.044396	0.028874	-0.115479	0.244642	-0.019553	-0.069435	0.014627	0.030463
V23	-0.010600	-0.041757	-0.001000	0.054753	-0.013118	0.027013	-0.004506	0.059080
V24	-0.012599	-0.001799	-0.027067	0.037405	-0.022450	-0.004478	0.021981	0.007174
V25	0.056241	0.169636	-0.090531	-0.189051	-0.019392	-0.067720	0.060828	-0.126596
V26	-0.039900	0.026456	-0.060862	0.065718	0.036497	-0.048299	0.012117	-0.040418
V27	-0.000972	-0.133281	0.075478	-0.181176	0.059052	-0.131250	-0.022211	-0.141613
V28	0.000907	0.139417	0.024509	0.039110	-0.018672	0.000174	-0.029312	-0.106850
Amount	0.056877	-0.211082	-0.480456	-0.154408	0.106500	-0.364685	0.216729	0.318986

What is Correlation ?

Correlation is used to check how strongly the variable is depended on the another variable. There are three typer of correlation.

1. Negative Correlation - When the varibles change in different directions
2. Positive Correlation - when the variables chane in the same direction.
3. Neutral Correlation - when there is no relationship between the variables.

There are several methods to check the correlation. Pearson's Correlation, Spearman's Correlation, etc.

Hence showing the correlation of the data with other data points

In [ ]:

```
#Understand the distribution of the data
df.tail()

```

Out[30]:

	Time	V1	V2	V3	V4	V5	V6	V7	
27813	34710	1.087354	0.043296	0.252652	1.225238	0.029356	0.340272	-0.011683	0.15:
27814	34711	1.443955	-1.052462	-0.141721	-1.564017	-0.966274	-0.333886	-0.777060	0.02:
27815	34711	-0.263364	0.931818	1.193111	-0.507924	0.862019	0.249381	0.815449	-0.09(
27816	34712	0.976345	-1.024867	0.978714	0.639442	-1.413711	0.311635	-0.909035	0.23:
27817	34712	1.464604	-0.437919	-0.018869	-1.057177	-0.154243	0.251215	-0.584866	-0.02!

In [ ]:

```
df.skew() # It tells the degree of distortion from the normal distribution which is use

```

Out[27]:

```
Time      -0.430981
V1        -4.270079
V2        -3.022931
V3        -6.827191
V4         0.567489
V5        -2.374845
V6         1.140171
V7        -2.438038
V8        -5.666808
V9         0.453450
V10       -0.187264
V11        0.899125
V12       -1.707235
V13        0.084497
V14       -3.349011
V15       -0.612727
V16       -2.050530
V17       -4.684985
V18       -0.547035
V19       -0.150282
V20        1.823668
V21        7.244862
V22       -0.760379
V23       -8.982165
V24       -0.616100
V25       -0.671586
V26        0.648109
V27       -1.735438
V28       -5.816776
Amount    12.337668
Class     17.209113
dtype: float64
```

In [ ]:

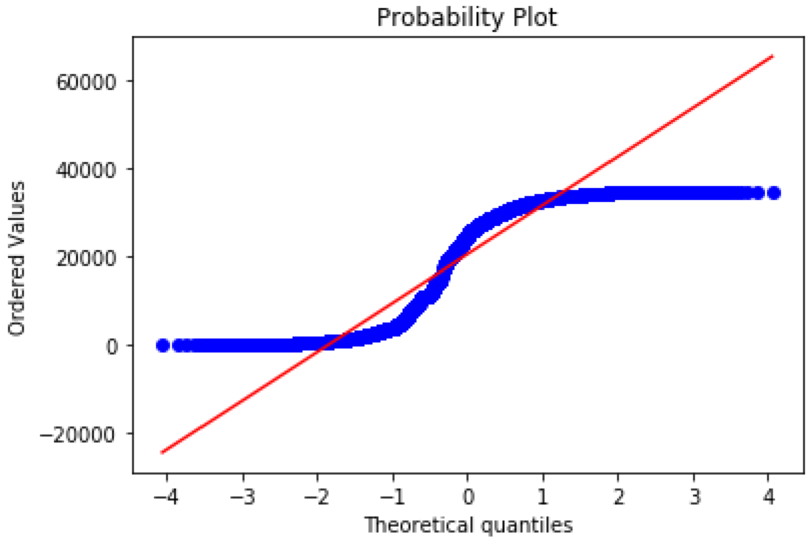
```
df.kurtosis()
```

Out[28]:

```
Time        -1.371055
V1          40.304061
V2          75.252743
V3          86.471913
V4           3.099847
V5          93.086784
V6          17.825994
V7         109.554986
V8         162.092937
V9           2.267243
V10         29.834539
V11          5.838710
V12         11.453112
V13         -0.382548
V14         34.665659
V15          0.447983
V16         17.785951
V17         71.392805
V18          3.598523
V19          0.956319
V20         95.667081
V21        232.926775
V22          8.821084
V23        437.572489
V24          0.721293
V25          6.298231
V26          0.102549
V27        108.508209
V28        204.621027
Amount     269.016632
Class      294.174722
dtype: float64
```

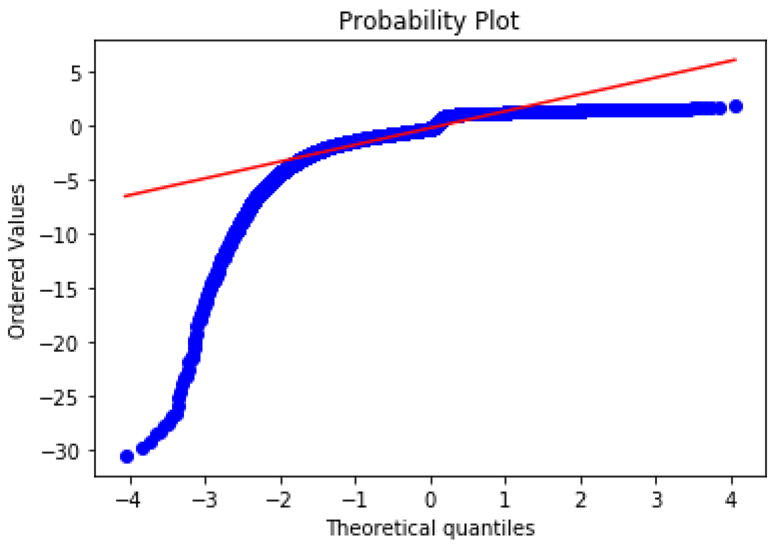
In [ ]:

```
# Understanding the probability distribution with the help of matplotlib with all the f

from scipy import stats
import matplotlib.pyplot as plt
res = stats.probplot(df['Time'], plot=plt)

```



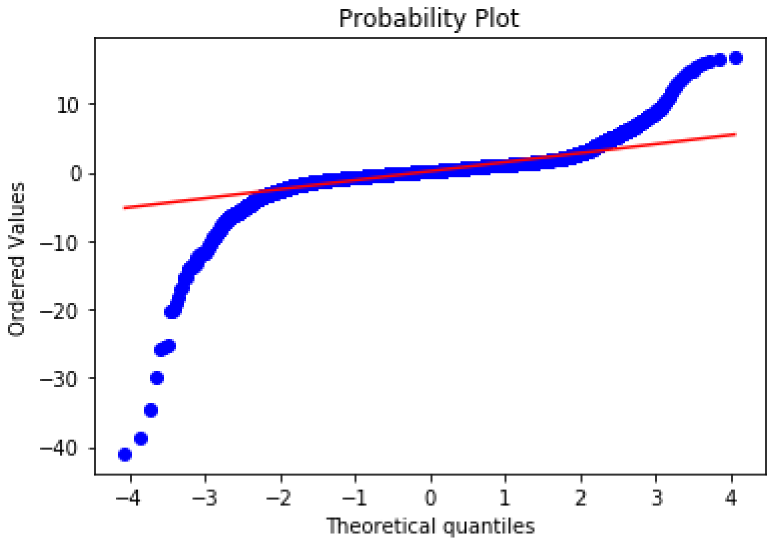
In [ ]:

```
res = stats.probplot(df['V1'], plot=plt)
```



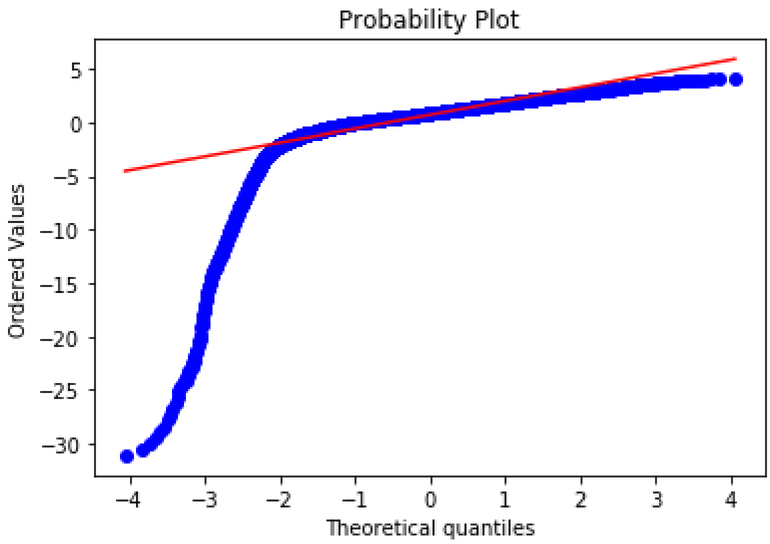
In [ ]:

```
res = stats.probplot(df['V2'], plot=plt)
```



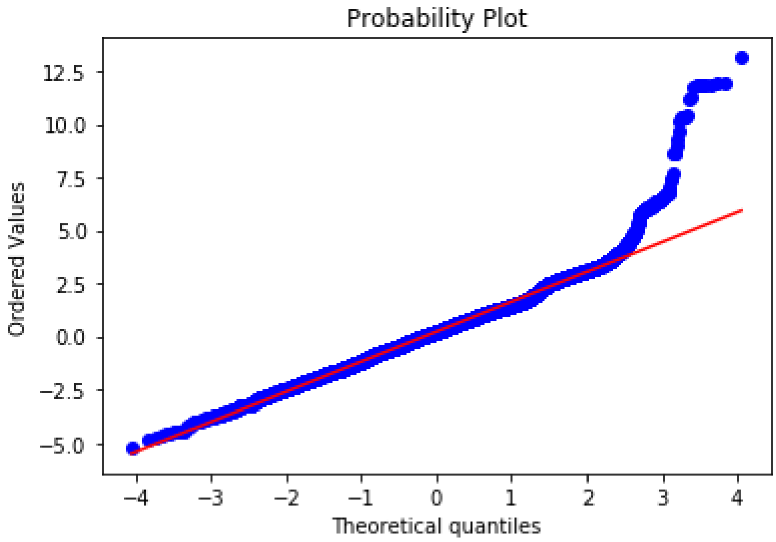
In [ ]:

```
res = stats.probplot(df['V3'], plot=plt)
```



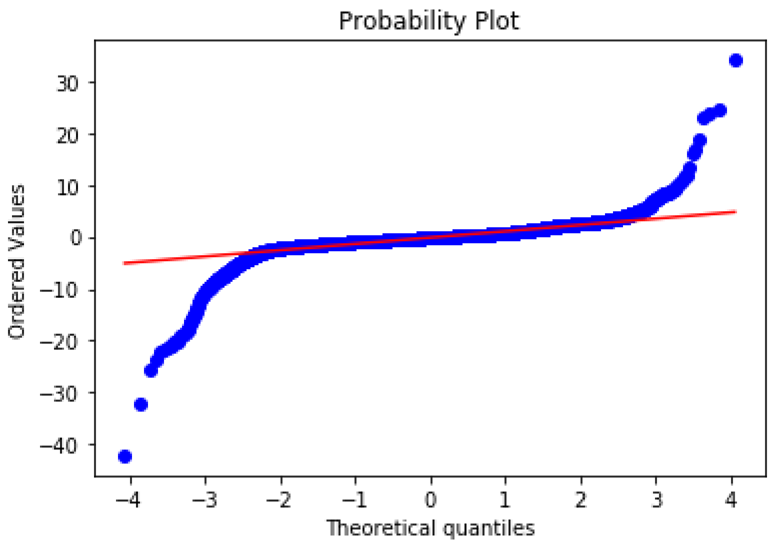
In [ ]:

```
res = stats.probplot(df['V4'], plot=plt)
```



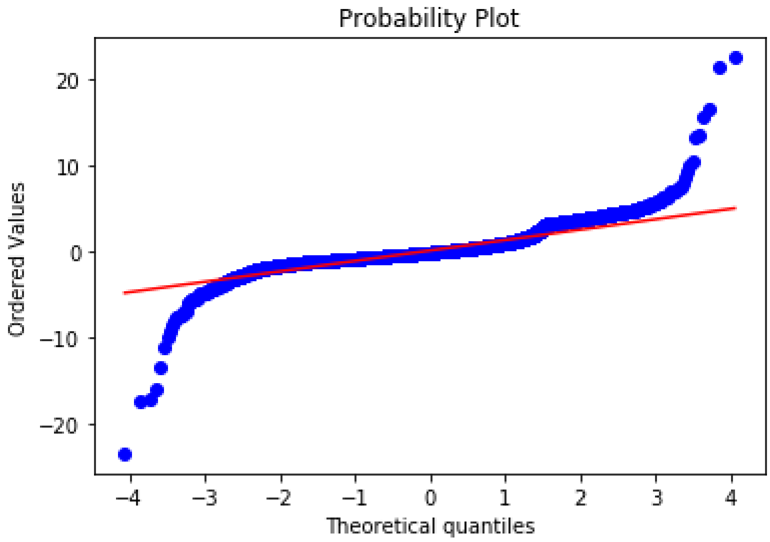
In [ ]:

```
res = stats.probplot(df['V5'], plot=plt)
```



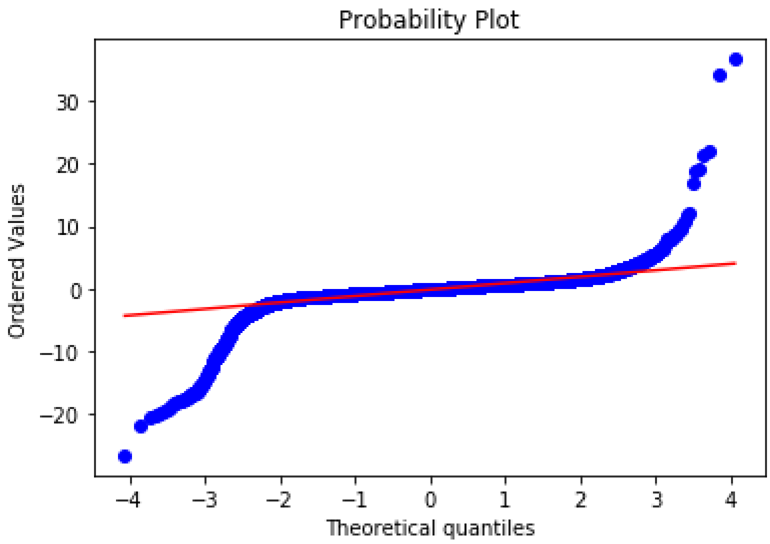
In [ ]:

```
res = stats.probplot(df['V6'], plot=plt)
```



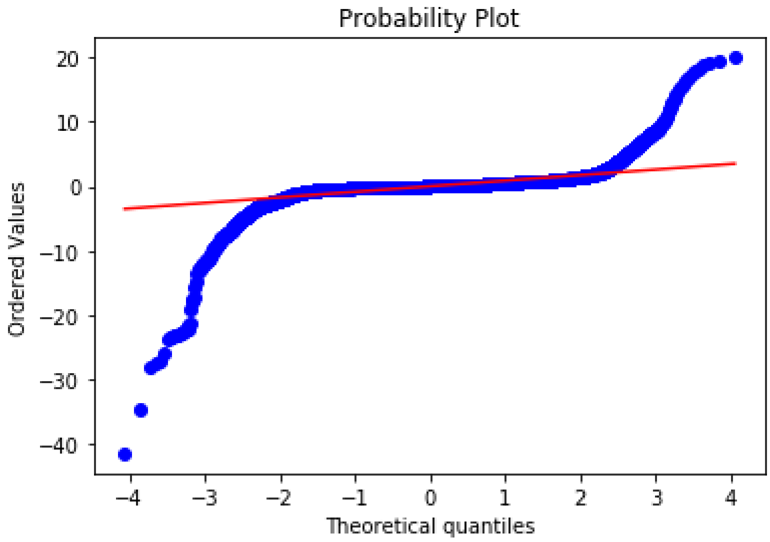
In [ ]:

```
res = stats.probplot(df['V7'], plot=plt)
```



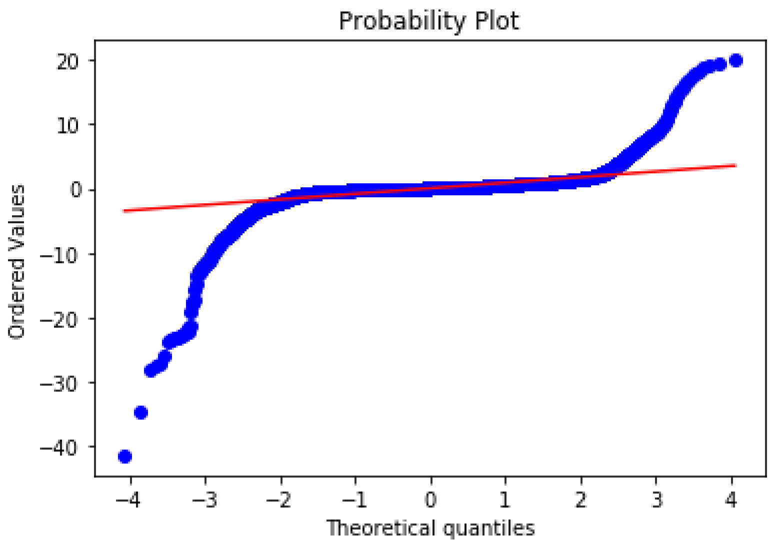
In [ ]:

```
res = stats.probplot(df['V8'], plot=plt)
```



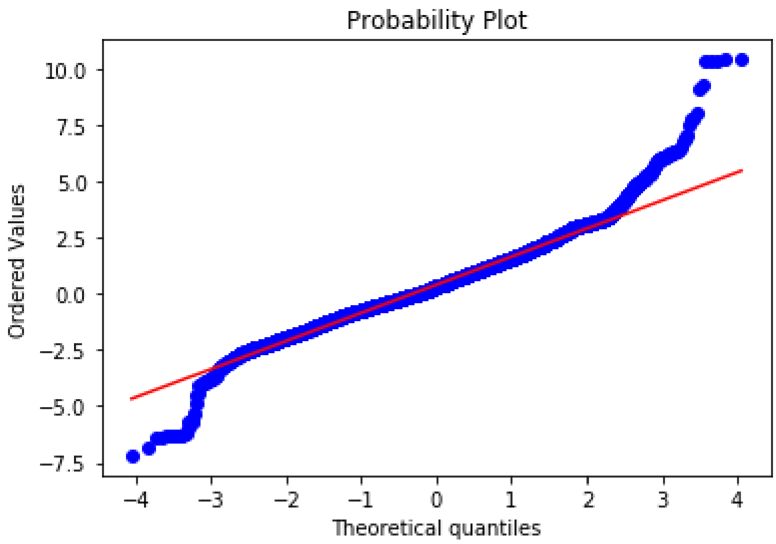
In [ ]:

```
res = stats.probplot(df['V8'], plot=plt)
```



In [ ]:

```
res = stats.probplot(df['V9'], plot=plt)
```



In [ ]:

```
# Plotting the Bar chart for showing the comparision between all the features
import matplotlib.pyplot as plt
df.plot(kind='bar')
plt.show()
```

In [ ]:

```
# To check the number of anamoly(1) and normal(0) in the class variable
# Normal variable are all the values of class with value 0. It is the normal data point

df['Amount'] = np.log(df['Amount'] + 1)
df['Time'] = np.log(df['Time'] + 1)
normal = df[df['Class'] == 0]
anomaly = df[df['Class'] == 1]

# Understanding the shape of the normal and anamoly data

print(normal.shape)
print(anomaly.shape)
```

```
(27725, 31)
(93, 31)
```

In [ ]:

```
# Making a class for defining the model fitting  and making the function for predictior

class hist_model(object):

    def __init__(self, bins=50):
        self.bins = bins

    def fit(self, X):

        bin_hight, bin_edge = [], []

        for var in X.T:
            # get bins hight and interval
            bh, bedge = np.histogram(var, bins=self.bins)
            bin_hight.append(bh)
            bin_edge.append(bedge)

        self.bin_hight = np.array(bin_hight)
        self.bin_edge = np.array(bin_edge)


    def predict(self, X):

        scores = []
        for obs in X:
            obs_score = []
            for i, var in enumerate(obs):
                # find wich bin obs is in
                bin_num = (var > self.bin_edge[i]).argmin()-1
                obs_score.append(self.bin_hight[i, bin_num]) # find bin hitght

            scores.append(np.mean(obs_score))

        return np.array(scores)

#fitting the model

model = hist_model()
model.fit(df.drop('Class', axis=1).values)

```

In [ ]:

```
from scipy.stats import multivariate_normal


mu = df.drop('Class', axis=1).mean(axis=0).values
sigma = df.drop('Class', axis=1).cov().values
model = multivariate_normal(cov=sigma, mean=mu, allow_singular=True)
```

In [ ]:

```
# Applying Gaussian Mixture Algorithm for model fitting
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, n_init=4, random_state=42)
gmm.fit(df.drop('Class', axis=1).values)
print(gmm.score(df[df['Class'] == 0].drop('Class', axis=1).values))
print(gmm.score(df[df['Class'] == 1].drop('Class', axis=1).values))
```

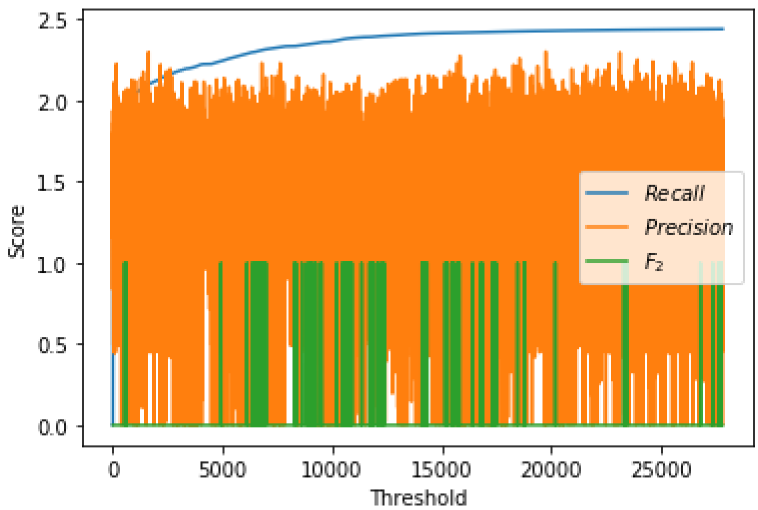
```
3.6145221342636265
-110.75461281593208
```

Data Visualization

Data visualization is a compatible way of understandin the behaviour of the feature so that we can fit the model accurately

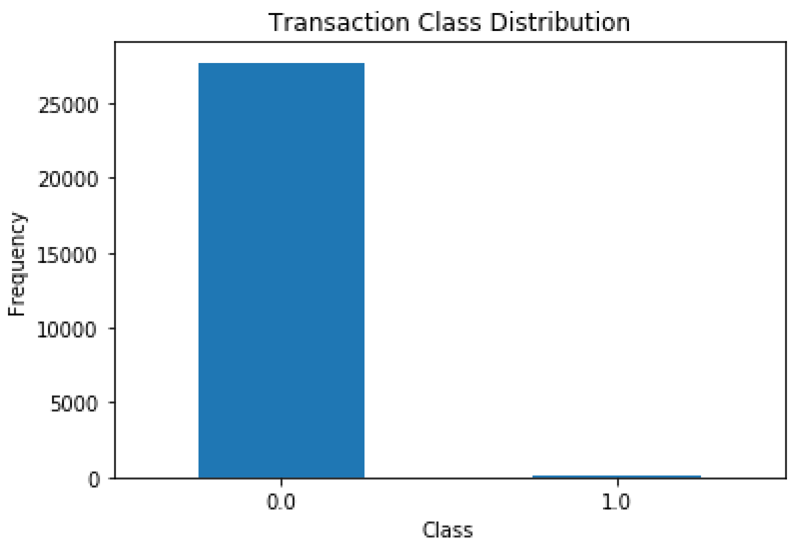
In [ ]:

```
# We need to check at what time the fraud is occuring with the class and how much amour
import matplotlib.pyplot as plt


plt.plot(df['Time'], label='$Recall$')
plt.plot(df['Amount'], label='$Precision$')
plt.plot(df['Class'], label='$F_2$')
plt.ylabel('Score')
# plt.xticks(np.logspace(-10, -200, 3))
plt.xlabel('Threshold')
plt.legend(loc='best')
plt.show()

```



In [ ]:

```
# Checking the Transaction distribution with the Class [0 = Normal, 1 = Fraud]
count_classes = pd.value_counts(db['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2) )
plt.xlabel("Class")
plt.ylabel("Frequency");
```



In [ ]:

```
# Making two variables Fraud and the normal data. Fraud has a value of one in the class

Fraud = db[db['Class']==1]

Normal = db[db['Class']==0]
```

In [ ]:

```
# Checking the fraud shape
Fraud.shape
```

Out[79]:

```
(93, 31)
```

In [ ]:

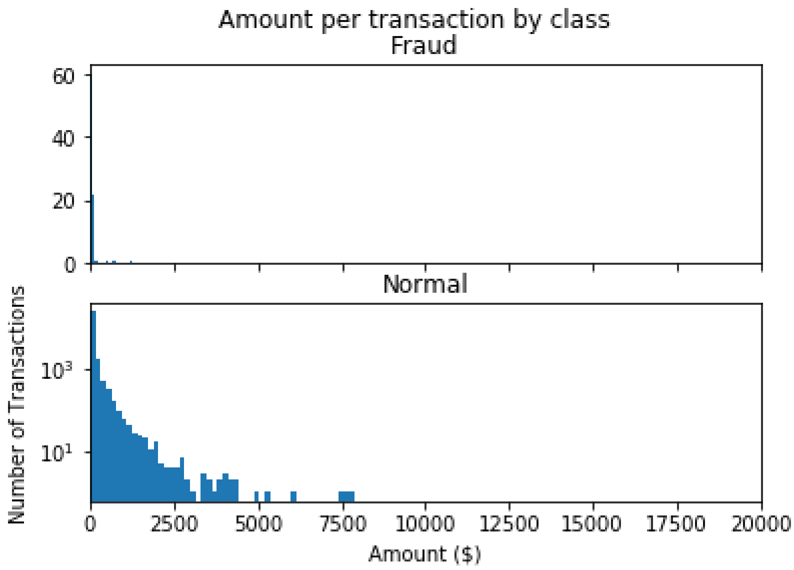
```
# Checking the Normal shape
Normal.shape
```

Out[81]:

```
(27725, 31)
```

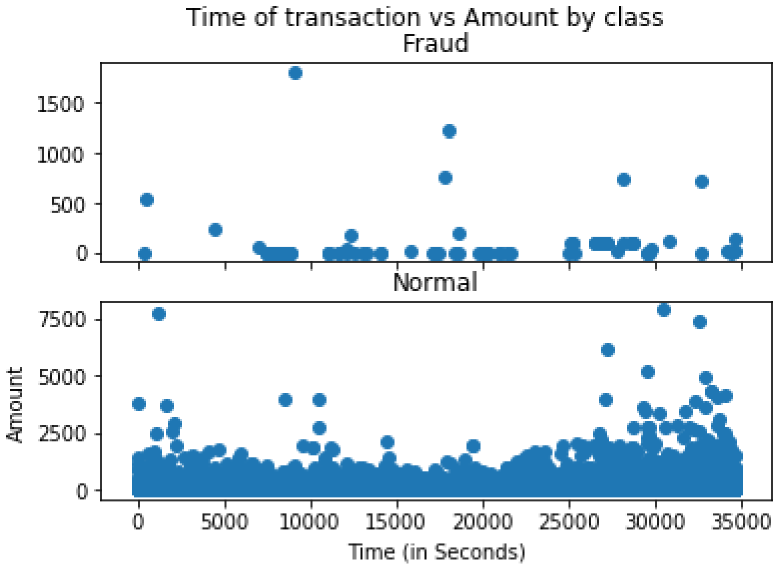
In [ ]:

```
# PLotting the graph separately for the amount of transaction of the clasws on the basi
 
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(Fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(Normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```



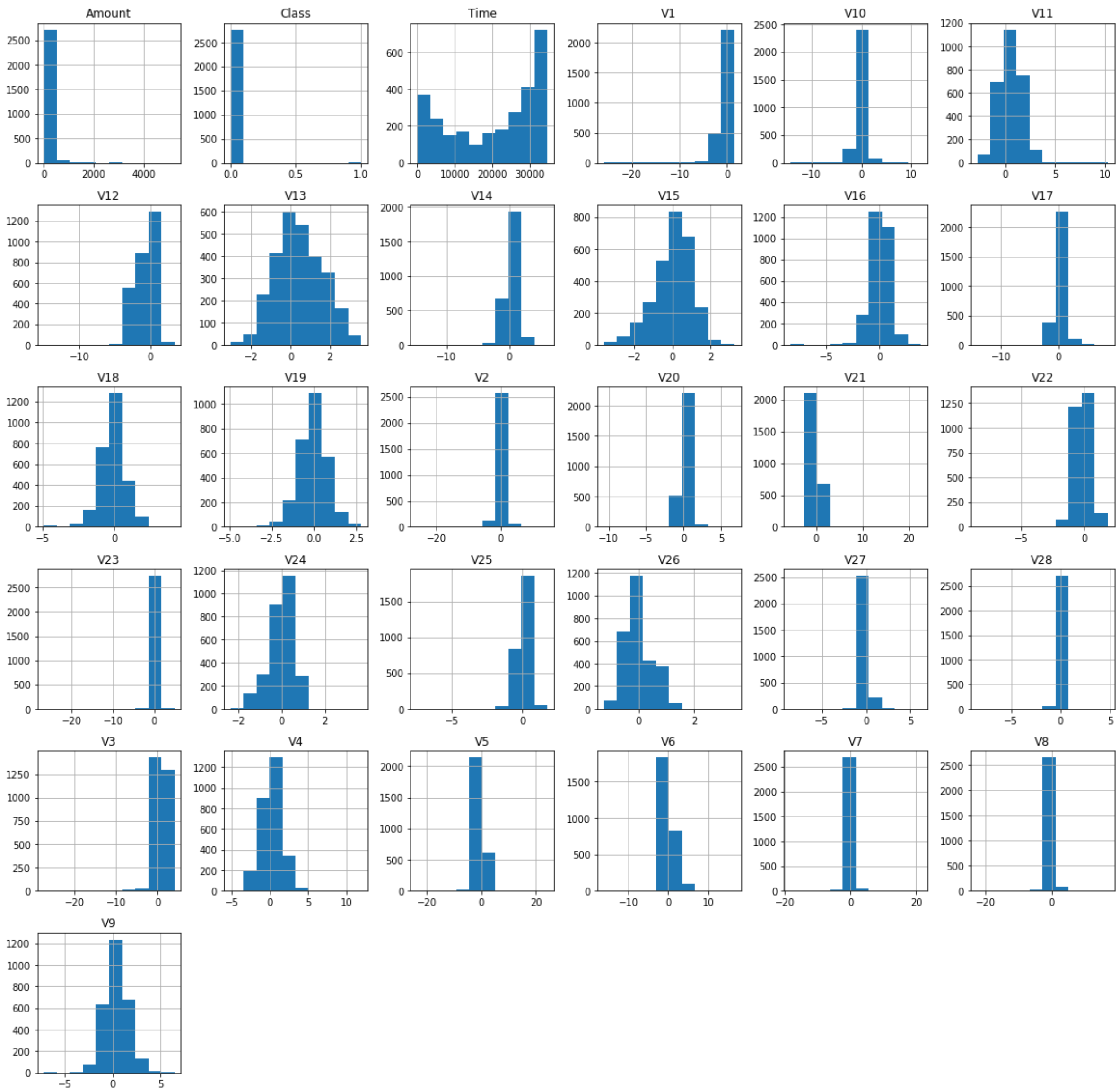
In [ ]:

```
# Plotting the scatter plot for the fraud and Normal detection with the help of time tr

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(Fraud.Time, Fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()


```



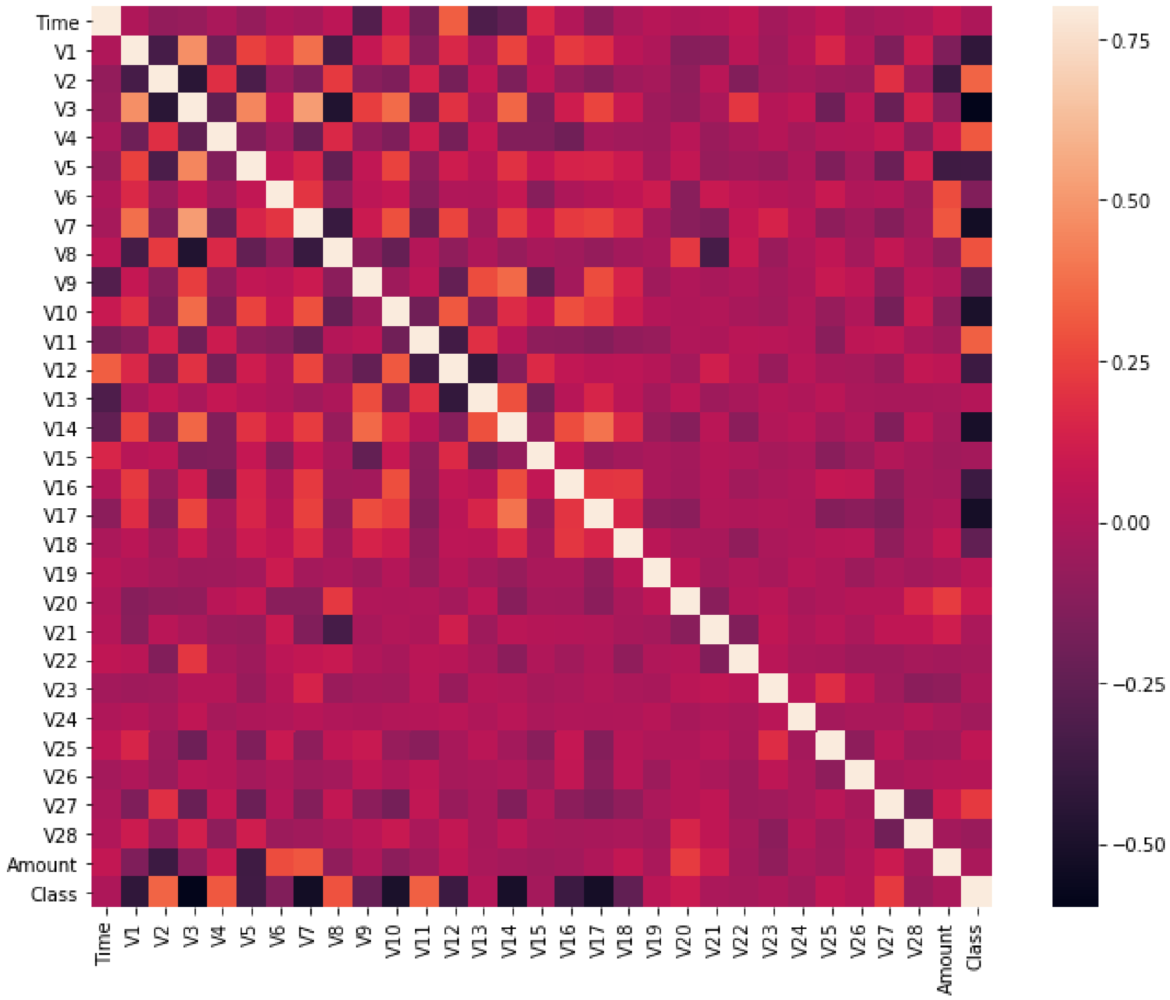
In [ ]:

```
# Plotting the individual Data to understand it more clearly
import matplotlib.pyplot as plt
data1= db.sample(frac = 0.1,random_state=1)

data1.shape
data1.hist(figsize=(20,20))
plt.show()
```



In [ ]:

```
# FInding the Correlation between the data points
import seaborn as sns

correlation_matrix = data1.corr()
fig = plt.figure(figsize=(12,9))
sns.heatmap(correlation_matrix,vmax=0.8,square = True)
plt.show()


```



In [ ]:

```
# Finding the outlier fraction by divind the fraud and valid variable count
Fraud = data1[data1['Class']==1]
Valid = data1[data1['Class']==0]
outlier_fraction = len(Fraud)/float(len(Valid))
print(outlier_fraction)
```

```
0.004332129963898917
```

In [ ]:

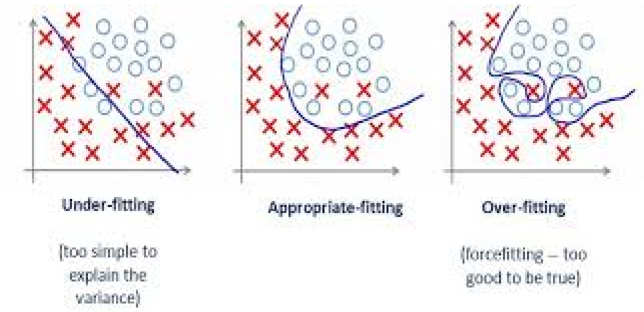
```
# Importing the libraries from  sklearn package
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
```

Model Fitting

Why it is important to fit the model ?

If your model is not fitting the data accurately, the outcome it will produce would be inefficient for taking the decision in the practical use.

There are three ways the model can get fit. It is shown by the simple figure.



Let us take a definition to underfitting, overfitting and appropriate fitting.

Overfitting - The performance on the data is good but the accuracy generates overfits(poor) the data.

Underfitting - The performance is poor and the accuracy generated by the model is also poor

Appropriate fitting - The performance and the generalization are both good

In [ ]:

```
# Making a classifier using oneClassSVM algorithm.

from sklearn.svm import OneClassSVM

classifiers = {
    "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(db),
                                       contamination=outlier_fraction,random_state = 0,
    "Local Outlier Factor":LocalOutlierFactor(n_neighbors=20, algorithm='auto',
                                              leaf_size=30, metric='minkowski',
                                              p=2, metric_params=None, contamination=ou
    "Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.05,
                                         max_iter=-1)

}


```

In [ ]:

```
columns = data1.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]
# Store the variable we are predicting
target = "Class"

X = data1[columns]
Y = data1[target]
n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    elif clf_name == "Support Vector Machine":
        clf.fit(X)
        y_pred = clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud transactic
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_errors = (y_pred != Y).sum()
    # Run Classification Metrics
    print("{}: {}".format(clf_name,n_errors))
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
    print("Classification Report :")
    print(classification_report(Y,y_pred))

```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/_iforest.py:281: Use
rWarning: max_samples (27819) is greater than the total number of samples (2
782). max_samples will be set to n_samples for estimation.
  % (self.max_samples, n_samples))


Isolation Forest: 9
Accuracy Score :
0.996764917325665
Classification Report :
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2770
         1.0       0.62      0.67      0.64        12

    accuracy                           1.00      2782
   macro avg       0.81      0.83      0.82      2782
weighted avg       1.00      1.00      1.00      2782

Local Outlier Factor: 25
Accuracy Score :
0.9910136592379583
Classification Report :
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2770
```

```
         1.0       0.00      0.00      0.00        12

    accuracy                           0.99      2782
   macro avg       0.50      0.50      0.50      2782
weighted avg       0.99      0.99      0.99      2782

Support Vector Machine: 1659
Accuracy Score :
0.403666427030913
Classification Report :
              precision    recall  f1-score   support

         0.0       1.00      0.40      0.57      2770
         1.0       0.00      0.58      0.01        12

    accuracy                           0.40      2782
   macro avg       0.50      0.49      0.29      2782
weighted avg       0.99      0.40      0.57      2782
```

We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

Thus we can use autoencoder.

Advantages of Anamoly Detection

1. It can Monitor the data source, networks, users, etc.
2. It can identify the Security threads.
3. It can find the trend of the unusual behavior of the data set and handles the security and safity.
4. It can identify key outliers.

Disadvantages

The biggest Disadvantage is that it cannot identify the novelty attacks and the various existing attacks.