In [1]:
```
import numpy as np
```

In [2]:
```
import pandas as pd
#data cleaning
```

In [3]:
```
Fraud_data = pd.read_csv('fraud_detect.csv')
```

In [4]:
```
Fraud_data
```

Out[4]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098 |
| **1** | 0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085 |
| **2** | 1 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247 |
| **3** | 1 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377 |
| **4** | 2 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **153753** | 99980 | -0.460128 | 0.203036 | 0.469998 | -0.877349 | -0.333565 | 1.198391 | 1.088143 | -0.303 |
| **153754** | 99984 | 1.930509 | 0.589877 | -0.573178 | 4.011452 | 0.646176 | 0.406016 | 0.003458 | -0.104 |
| **153755** | 99995 | -0.080563 | 0.840885 | -0.085326 | -0.606702 | 0.879790 | -0.493156 | 0.879652 | -0.217 |
| **153756** | 99998 | -1.730665 | 1.302833 | 0.397864 | -0.445631 | -0.773382 | 0.223966 | -0.921886 | 1.329 |
| **153757** | 99999 | -0.309641 | 0.771215 | 0.679051 | -1.069673 | 1.414042 | -0.379638 | 1.315839 | -0.449 |

153758 rows × 31 columns

In [5]:
```
Fraud_data.head()
```

Out[5]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0 |
| **1** | 0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0 |
| **2** | 1 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1 |
| **3** | 1 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1 |
| **4** | 2 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0 |

5 rows × 31 columns

In [6]:
```
Fraud_data.tail()
```

Out[6]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| **153753** | 99980 | -0.460128 | 0.203036 | 0.469998 | -0.877349 | -0.333565 | 1.198391 | 1.088143 | -0.3037 |
| **153754** | 99984 | 1.930509 | 0.589877 | -0.573178 | 4.011452 | 0.646176 | 0.406016 | 0.003458 | -0.1040 |
| **153755** | 99995 | -0.080563 | 0.840885 | -0.085326 | -0.606702 | 0.879790 | -0.493156 | 0.879652 | -0.2178 |
| **153756** | 99998 | -1.730665 | 1.302833 | 0.397864 | -0.445631 | -0.773382 | 0.223966 | -0.921886 | 1.3292 |
| **153757** | 99999 | -0.309641 | 0.771215 | 0.679051 | -1.069673 | 1.414042 | -0.379638 | 1.315839 | -0.4499 |

5 rows × 31 columns

In [8]:
```
Fraud_data.shape
```

Out[8]: (153758, 31)

In [9]:
```
Fraud_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153758 entries, 0 to 153757
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    153758 non-null  int64  
 1   V1      153758 non-null  float64
 2   V2      153758 non-null  float64
 3   V3      153758 non-null  float64
 4   V4      153758 non-null  float64
 5   V5      153758 non-null  float64
 6   V6      153758 non-null  float64
 7   V7      153758 non-null  float64
 8   V8      153758 non-null  float64
 9   V9      153758 non-null  float64
 10  V10     153758 non-null  float64
 11  V11     153758 non-null  float64
 12  V12     153758 non-null  float64
 13  V13     153758 non-null  float64
 14  V14     153758 non-null  float64
 15  V15     153758 non-null  float64
 16  V16     153758 non-null  float64
 17  V17     153758 non-null  float64
 18  V18     153758 non-null  float64
 19  V19     153758 non-null  float64
 20  V20     153758 non-null  float64
 21  V21     153758 non-null  float64
 22  V22     153758 non-null  float64
 23  V23     153758 non-null  float64
 24  V24     153758 non-null  float64
 25  V25     153758 non-null  float64
 26  V26     153758 non-null  float64
 27  V27     153758 non-null  float64
 28  V28     153758 non-null  float64
 29  Amount  153758 non-null  float64
 30  Class   153758 non-null  int64  
dtypes: float64(29), int64(2)
memory usage: 36.4 MB
```

In [10]:
```
Fraud_data.isnull().sum()
```

Out[10]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [11]:
```
Fraud_data.describe()
```

Out[11]:

| V9 | ... | V21 | V22 | V23 | V24 | V25 | V2 |
|---|---|---|---|---|---|---|---|
| 00 | ... | 153758.000000 | 153758.000000 | 153758.000000 | 153758.000000 | 153758.000000 | 153758.00000 |
| 71 | ... | -0.036877 | -0.109254 | -0.028362 | 0.011553 | 0.115682 | 0.02275 |
| 97 | ... | 0.742079 | 0.647129 | 0.585021 | 0.595752 | 0.448905 | 0.49098 |
| 73 | ... | -34.830382 | -10.933144 | -44.807735 | -2.836627 | -10.295397 | -2.60455 |
| 97 | ... | -0.229086 | -0.551681 | -0.170680 | -0.327445 | -0.159352 | -0.32581 |
| 85 | ... | -0.059646 | -0.085381 | -0.041720 | 0.065159 | 0.154804 | -0.05603 |
| 93 | ... | 0.115133 | 0.321049 | 0.088913 | 0.411360 | 0.413009 | 0.28699 |
| 95 | ... | 27.202839 | 10.503090 | 19.002942 | 4.022866 | 7.519589 | 3.51734 |

In [12]:
```
class_names = {0:'Not Fraud',1:'Fraud'}
print(Fraud_data.Class.value_counts().rename(index = class_names))
```

```
Not Fraud    153428
Fraud           330
Name: Class, dtype: int64
```

In [15]:
```
from sklearn.model_selection import train_test_split
y = Fraud_data['Class']
x = Fraud_data.loc[:,Fraud_data.columns!='Class']
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=1/3, random_s
```

In [16]:
```
x_train,x_test,y_train,y_test
```

Out[16]: (

```
         Time        V1        V2        V3        V4        V5        V6  \
32096   36636 -0.556476  0.763791  1.526293 -1.338977  0.055728 -0.601567
40165   40122 -2.280471 -0.326044  0.244870  1.118990  0.262526 -0.612519
20310   30943 -0.517172  0.384920  2.014336  1.247855 -0.306978  0.337996
103907  68847  1.275845 -0.890244 -1.740971 -1.759315  1.725460  2.937240
95791   65468 -0.280641  1.064322  1.677338  0.153278 -0.090096 -1.260371
...       ...       ...       ...       ...       ...       ...       ...
95679   65419 -0.144034  1.223659 -0.255421  1.314053  0.290211 -0.678083
93185   64285  0.997774 -0.618597  1.268644  1.401820 -0.592760  1.982795
33495   37256 -2.528657 -0.343713  2.552338  2.265170  5.936980 -4.013499
58280   48319  1.470151 -0.432102 -0.539163 -0.956843 -0.277923 -0.639283
74264   55495  1.268762  0.151264 -0.013182 -0.119725 -0.085258 -0.832852

              V7        V8        V9  ...       V20       V21       V22  \
32096   0.681059 -0.143321  0.366432  ...  0.203522 -0.156812 -0.102191
40165   1.051707  0.301572 -0.993480  ... -0.194594 -0.330055 -0.612499
20310   0.079046  0.296088  0.218632  ...  0.015205  0.258817  0.859568
103907 -0.491588  0.602610 -1.282300  ...  0.417372 -0.129422 -0.904907
95791   0.915463 -0.322169 -0.273264  ...  0.159370 -0.282315 -0.570511
...          ...       ...       ...  ...       ...       ...       ...
95679   0.544573  0.250660 -0.878896  ...  0.074420  0.225462  0.754134
93185  -1.162782  0.664184  1.502888  ... -0.051910 -0.036956  0.308709
33495  -6.140718 -0.721761  0.489934  ... -1.271693  0.581498 -2.065646
58280  -0.232350 -0.143694 -1.176314  ...  0.019397  0.171971  0.350472
74264   0.260397 -0.266134 -0.121395  ... -0.023431 -0.040880 -0.003052

             V23       V24       V25       V26       V27       V28  Amount
32096  -0.030903  0.124942 -0.286211  0.727246  0.141696 -0.097933    7.68
40165   1.475323 -0.099025  0.160107 -0.588166  0.387752 -0.079526  199.90
20310  -0.059669  0.071208 -0.300359 -0.157647  0.197853  0.162837   47.80
103907 -0.063096  0.992000  0.570146 -0.447815 -0.041373  0.023436  125.15
95791   0.008598  0.865049 -0.261254  0.010732  0.082948 -0.119656    1.98
...          ...       ...       ...       ...       ...       ...     ...
95679   0.167102  0.115438 -0.713587 -0.323169  0.292775  0.087152   22.24
93185  -0.264624 -1.089620  0.596984 -0.145649  0.112083  0.021608   54.89
33495  -9.587483  0.877383 -2.729109 -0.640681  0.280202  0.406317   18.75
58280  -0.261815 -0.486822  0.834439 -0.046598 -0.040733 -0.019838   10.00
74264  -0.018060  0.194447  0.392159  1.065891 -0.079687 -0.005022    8.34

[102505 rows x 30 columns],
         Time        V1        V2        V3        V4        V5        V6  \
10376   16667  1.009401 -1.256933  1.643618 -0.632346 -2.074077  0.020436
79189   57909  0.126818 -0.667921  1.258806 -1.611600 -0.944668  0.530279
110122  71687 -0.903598  1.121444  0.920401 -0.026369  0.284560  0.817971
47821   43375  1.136775  0.222545  0.445628  1.264572 -0.028812  0.157545
34678   37777 -0.795127  0.266019 -1.038463  0.273348  2.813373  3.196132
...       ...       ...       ...       ...       ...       ...       ...
123953  77127 -0.058448  0.855790  0.852111  1.379529 -0.412359  0.539611
24039   33041  0.315608 -1.762890 -0.500017 -0.655834 -1.013058 -0.711085
16470   27846  1.245815 -0.062004  0.428500  0.785105 -0.467353 -0.310081
73475   55151 -1.933383 -1.456943  2.510036 -1.444307  0.216649 -1.297811
103372  68621 -0.602578  0.706041  0.189768 -0.305701  2.862724  3.452883

              V7        V8        V9  ...       V20       V21       V22  \
10376  -1.395074  0.212397  4.608729  ... -0.036507 -0.030341  0.542996
79189  -0.732632  0.244336 -2.504447  ... -0.074377 -0.219242 -0.372177
110122  0.026253  0.959968 -0.525517  ... -0.155684 -0.080748 -0.188855
```

```
47821  -0.005505  0.101294 -0.055180  ... -0.157121 -0.048144  0.106745
34678  -0.406960  1.275713 -0.735916  ...  0.215295  0.182472  0.194915
...          ...       ...       ...  ...       ...       ...       ...
123953 -1.073512 -2.462365 -0.884261  ...  0.643877 -1.064280  0.898922
24039   0.544821 -0.197151  0.953012  ...  0.874338  0.266235 -0.319705
16470  -0.187974 -0.038862  0.732560  ... -0.121261 -0.142759 -0.196286
73475  -1.138839  0.174311 -1.077102  ...  0.025141  0.483315  1.147641
103372  0.134375  0.879967 -0.967459  ...  0.222306  0.049677 -0.053145

             V23       V24       V25       V26       V27       V28  Amount
10376  -0.236625  0.431423  0.496446  0.112434  0.057854  0.037415   97.86
79189   0.192087 -0.727997 -0.582447 -0.277710  0.094281  0.025517   55.65
110122  0.240184 -0.704559 -0.508444  0.184242  0.184807  0.024458   14.55
47821  -0.075170  0.054127  0.655752 -0.331239  0.041228  0.005787    1.00
34678  -0.087260  1.031853 -0.294174 -0.285613  0.109792  0.024704   16.32
...          ...       ...       ...       ...       ...       ...     ...
123953 -0.072814  0.220485  0.602676 -0.123488  0.128983  0.250603   33.60
24039  -0.531044  0.002387  0.428149 -0.784877 -0.053306  0.096450  502.50
16470  -0.131048 -0.068641  0.600748  0.442992 -0.018024  0.007212   12.36
73475  -0.385061  0.587006  0.379512 -0.168074 -0.003092  0.129105   49.99
103372 -0.381152  1.009525  0.681250 -0.179583  0.067184  0.072070    4.35

[51253 rows x 30 columns],
32096     0
40165     0
20310     0
103907    0
95791     0
         ..
95679     0
93185     0
33495     0
58280     0
74264     0
Name: Class, Length: 102505, dtype: int64,
10376     0
79189     0
110122    0
47821     0
34678     0
         ..
123953    0
24039     0
16470     0
73475     0
103372    0
Name: Class, Length: 51253, dtype: int64)
```

In [ ]:
```
#machine learning
#5 ml algos
```

In [17]:
```
#importing libraries
#import library for accuracy score
from sklearn.metrics import accuracy_score
#import library for logistic regression
from sklearn.linear_model import LogisticRegression
```

In [18]:
```
#logistic regression
logisreg = LogisticRegression()
logisreg.fit(x_train, y_train)
y_pred = logisreg.predict(x_test)
acc_logisreg=round(accuracy_score(y_test, y_pred)*100,2)
```

```
C:\Users\udani\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [19]:
```
acc_logisreg
```

Out[19]:
```
99.86
```

In [21]:
```
#naive bayesian
#import library for gaussian naive bayes
from sklearn.naive_bayes import GaussianNB
#initialize the gaussian naive bayes classifier
model = GaussianNB()
#train the model using training dataset
model.fit(x_train,y_train)
#prediction using test data
y_pred = model.predict(x_test)
#Calculate model accuracy by comparing y_test and y_pred
acc_ganb = round(accuracy_score(y_test,y_pred)*100,2)
print('Accuracy of Gaussian Naive Bayes: ',acc_ganb)
```

```
Accuracy of Gaussian Naive Bayes:  98.54
```

In [22]:
```
#decision tree(CART)
#import library for decision tree classifier
from sklearn.tree import DecisionTreeClassifier
#initialize the decision tree classifier
model = DecisionTreeClassifier()
#train the model using training dataset
model.fit(x_train,y_train)
#prediction using test data
y_pred = model.predict(x_test)
#Calculate model accuracy by comparing y_test and y_pred
acc_dtree = round(accuracy_score(y_test,y_pred)*100,2)
print('Accuracy of Decision Tree Classifier: ',acc_dtree)
```

```
Accuracy of Decision Tree Classifier:  99.9
```

In [25]:
```
#Random Forest
#import library for Random Forest
from sklearn.ensemble import RandomForestClassifier
#initialize the Random Forest
model = RandomForestClassifier()
#train the model using training dataset
model.fit(x_train,y_train)
#prediction using test data
y_pred = model.predict(x_test)
#Calculate model accuracy by comparing y_test and y_pred
acc_rf = round(accuracy_score(y_test,y_pred)*100,2)
print('Accuracy of Random Forest: ',acc_rf)
```

```
Accuracy of Random Forest:  99.95
```

In [27]:
```
#K Nearest Neighbour Classifier
#import library for K Nearest Neighbour Classifier
from sklearn.neighbors import KNeighborsClassifier
#initialize the K Nearest Neighbour Model with Default value of K=5
model = KNeighborsClassifier()
#train the model using training dataset
model.fit(x_train,y_train)
#prediction using test data
y_pred = model.predict(x_test)
#Calculate model accuracy by comparing y_test and y_pred
acc_knn = round(accuracy_score(y_test,y_pred)*100,2)
print('Accuracy of KNN: ',acc_knn)
```

```
C:\Users\udani\anaconda3\lib\site-packages\sklearn\neighbors\_classification.
py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtos
is`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdim
s` will become False, the `axis` over which the statistic is taken will be el
iminated, and the value None will no longer be accepted. Set `keepdims` to Tr
ue or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

Accuracy of KNN:  99.8
```

In [28]:
```
models=pd.DataFrame({
    'Model':['Logistic Regression','Naive Bayes','Decision Tree','Random Fores
    'Score':[acc_logisreg,acc_ganb,acc_dtree,acc_rf,acc_knn]
})

models.sort_values(by='Score',ascending=False)
```

Out[28]:

| | Model | Score |
|---|---|---|
| **3** | Random Forest | 99.95 |
| **2** | Decision Tree | 99.90 |
| **0** | Logistic Regression | 99.86 |
| **4** | K-Nearest Neighbors | 99.80 |
| **1** | Naive Bayes | 98.54 |

In [ ]: