importing the dependencies

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
#load the dataset
df=pd.read_csv('creditcard.csv')
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

```
df.tail()
```

	Time	V1	V2	V3	V4	V5	V6	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.(
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.(
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

5 rows × 31 columns

```
#dataset information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
```

```
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
#checking for missing values
df.isnull().sum()
```

```
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
```

```
#distribution of legit transaction and fraudulent transaction
df['Class'].value_counts()
```

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

This dataset is highly unbalenced 0 -> Normal transaction 1 -> fraudulent transaction

```
#separating the data for analysis
legit= df[df.Class==0]
fraud=df[df.Class==1]
```

```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

```
legit['Amount'].describe()
```

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
fraud['Amount'].describe()
```

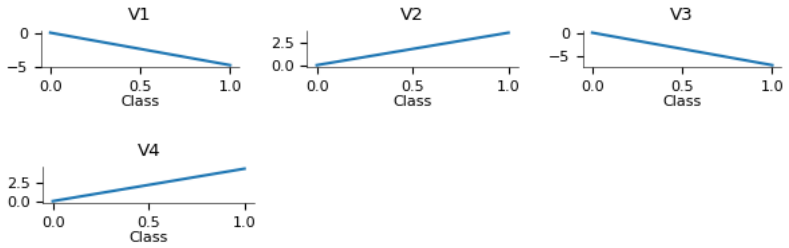
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

```
#compare the values of both transaction
df.groupby('Class').mean()
```

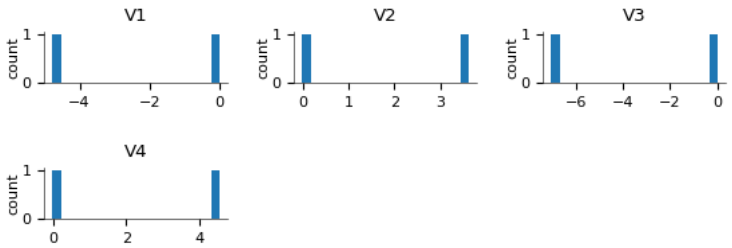
Class	Time	V1	V2	V3	V4	V5	V6	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-

2 rows × 30 columns

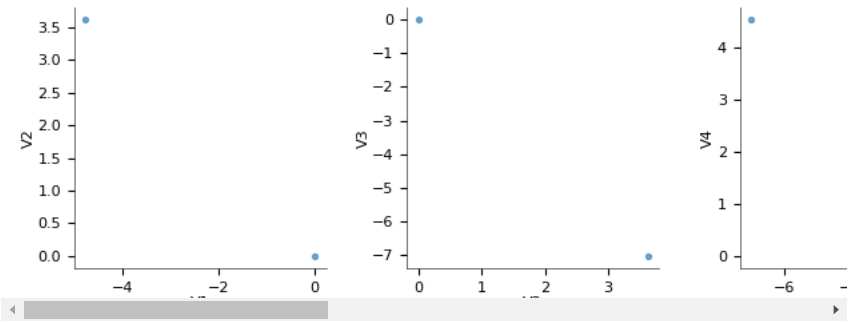
Values



Distributions



2-d distributions



under-sampling

build a sample dataset containing similar distribution of normal transaction and fraudulent transation

number of fraudulent transaction->> 492

```
legit_sample=legit.sample(n = 492)
```

concatenating two dataframes

```
new_df=pd.concat([legit_sample,fraud],axis=0)
```

```
new_df.head()
```

	Time	V1	V2	V3	V4	V5	V6	
201783	134035.0	1.815872	-0.635431	-0.231802	0.472039	-0.836760	-0.243136	-0.6
247002	153422.0	-0.133684	1.717586	0.364654	4.556414	0.487969	-0.033233	0.3
17244	28557.0	-0.016040	0.373693	0.116868	-1.819846	0.195248	-1.275842	0.7
200838	133603.0	1.176521	-1.500626	-0.766906	1.473935	-0.854656	0.064217	-0.1
47268	43153.0	1.163186	0.281816	0.356777	1.209631	-0.642557	-1.106850	-0.0

5 rows × 31 columns

```
new_df.tail()
```

	Time	V1	V2	V3	V4	V5	V6	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.88
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.41
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.23
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.20
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.22

5 rows × 31 columns

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

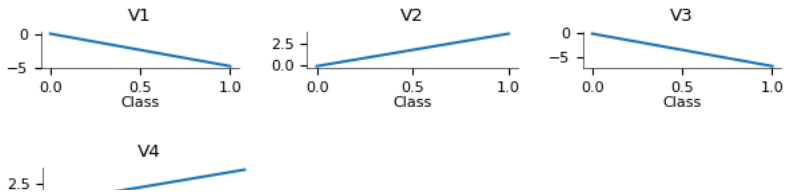
```
0    492
1    492
Name: Class, dtype: int64
```

```
new_df.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	
0	96321.032520	-0.005116	-0.038503	-0.039403	-0.040548	0.024332	0.046927	
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-

2 rows × 30 columns

Values



spliting the data into Feature and Target

```
X=new_df.drop(columns='Class',axis=1)
Y=new_df['Class']
```

```
print(X)
print(Y)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
201783  134035.0  1.815872 -0.635431 -0.231802  0.472039 -0.836760 -0.243136
247002  153422.0 -0.133684  1.717586  0.364654  4.556414  0.487969 -0.033233
17244    28557.0 -0.016040  0.373693  0.116868 -1.819846  0.195248 -1.275842
200838  133603.0  1.176521 -1.500626 -0.766906  1.473935 -0.854656  0.064217
47268    43153.0  1.163186  0.281816  0.356777  1.209631 -0.642557 -1.106850
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

                V7        V8        V9  ...       V20       V21       V22  \
201783 -0.648644  0.092788  1.206726  ... -0.117698  0.288055  0.937303
247002  0.362601  0.186318 -2.466715  ...  0.053153  0.322699  0.946255
17244   0.714206 -0.304416 -1.319121  ...  0.143866  0.301587  0.902397
200838 -0.114347  0.045600  0.748702  ...  0.519312  0.484340  0.584323
47268  -0.052257 -0.145071  0.523549  ... -0.079085 -0.143207 -0.162980
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

              V23       V24       V25       V26       V27       V28  Amount
201783  0.078674  0.044090 -0.175058 -0.238123  0.030902 -0.037171   62.24
247002 -0.003495  0.097051  0.119742  0.574433 -0.230451 -0.143841    1.00
17244  -0.087022 -0.088623 -0.563822 -0.393171  0.427991  0.273642   15.00
200838 -0.250906 -0.354208 -0.249364 -0.600547 -0.027085  0.023616  400.00
47268  -0.031861  0.862306  0.482774  0.377841  0.003390  0.048828   12.31
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
201783    0
247002    0
17244     0
200838    0
47268     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

spliting the data into training data and testing data

```
X_train, X_test, Y_train, Y_test=train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=2)
```

```
print(X_train.shape,X_test.shape,X.shape)
```

```
(787, 30) (197, 30) (984, 30)
```

```
print(Y_train.shape,Y_test.shape,Y.shape)
```

```
(787,) (197,) (984,)
```

Model Training

Logisticregression

```
model=LogisticRegression()
```

```
#training the data
model.fit(X_train,Y_train)
```

```
LogisticRegression
LogisticRegression()
```

model evaluation

Accuracy score

```
#accuracy on tarining data
x_train_prediction=model.predict(X_train)
training_data_accuracy=accuracy_score(x_train_prediction,Y_train)
```

```
print('accuracy on the training data:',training_data_accuracy)
```

```
accuracy on the training data: 0.9390088945362135
```

```
X_test_prediction=model.predict(X_test)
testing_data_accuracy=accuracy_score(X_test_prediction,Y_test)
```

```
print('accuracy score of testing data:',testing_data_accuracy)
```

```
accuracy score of testing data: 0.9289340101522843
```