

importing the libraries and functions we need (dependencies)

```
In [1]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
In [2]: ##Loading the dataset to Pandas Dataframe
credit_card_data = pd.read_csv('creditcard.csv')
```

```
In [3]: #printing the first five rows of the dataset
credit_card_data.head()
```

```
Out[3]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 31 columns

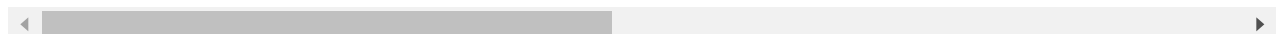


```
In [4]: credit_card_data.tail()
```

```
Out[4]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	

5 rows × 31 columns



```
In [5]: credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
#   Column      Non-Null Count  Dtype
---  -
#   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
```

In [9]:

#number of null values in each column
credit_card_data.isnull().sum()

```
Out[9]: 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 [10]: #disribution of Leget transation & fraudlent transantion
         credit_card_data['Class'].value_counts()
```

```
Out[10]: 0    284315
         1      492
         Name: Class, dtype: int64
```

The above data set is highly unbalanced so we need for data processing

The table '0' represents Legit Transation

The table '1' represents Fraudulent TRansation

```
In [11]: ##seperating the data for analysis
         legit = credit_card_data[credit_card_data.Class == 0]
         fraud = credit_card_data[credit_card_data.Class == 1]
```

```
In [12]: print(legit.shape)
         print(fraud.shape)
```

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

```
In [13]: #statstical measures of data
         legit.Amount.describe()
```

```
Out[13]: 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
```

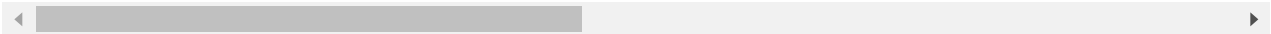
```
fraud.Amount.describe()
```

```
In [17]: #compare the values of the both Transations
         credit_card_data.groupby('Class').mean()
```

```
Out[17]:      Time      V1      V2      V3      V4      V5      V6      V7      V8
```

	Class	Time	V1	V2	V3	V4	V5	V6	V7	V8
Class										
	0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.00098
	1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.57063

2 rows × 30 columns



build a sample dataset containing similar of distribution of normal Transation and Fraud Transation

number of fraud transation is 492

In [18]:

legit_sample = legit.sample(n = 492)

concadianting 2 data frames

In [20]:

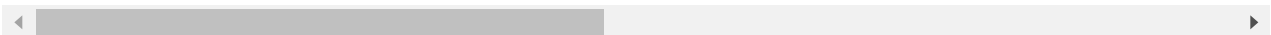
new_dataFrame = pd.concat([legit_sample,fraud] ,axis = 0)

In [21]:

new_dataFrame.head()

	Time	V1	V2	V3	V4	V5	V6	V7	V8
46562	42836.0	-0.040184	-2.945298	0.551378	0.136728	-2.112744	0.627532	-0.336272	0.119987
163003	115560.0	2.017699	-0.418599	-2.562017	-0.715467	2.371590	3.324169	-0.456447	0.787120
154258	100940.0	-0.156578	0.162522	2.104589	-0.047495	0.051366	0.453470	0.005426	-0.230959
222837	143138.0	-0.100226	0.654743	0.087549	-0.249101	0.747274	-0.309196	0.728941	0.058566
166461	118095.0	2.326970	-1.492014	-0.872100	-1.620743	-1.319074	-0.362659	-1.415057	-0.021397

5 rows × 31 columns



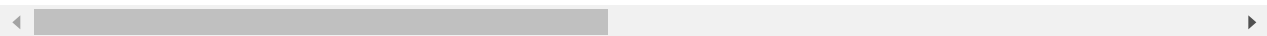
In [22]:

new_dataFrame.tail()

Out[22]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384

5 rows × 31 columns



```
In [25]: new_dataFrame['Class'].value_counts()
```

Out[25]:

0	492
1	492

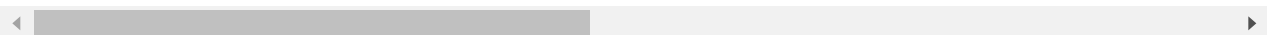
Name: Class, dtype: int64

```
In [27]: new_dataFrame.groupby('Class').mean()
```

Out[27]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
Class									
0	94082.711382	-0.128350	-0.044582	-0.008843	0.041949	0.033749	-0.018302	-0.016608	-0.081225
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636

2 rows × 30 columns



splitting the data into features and targets

```
In [33]: X = new_dataFrame.drop(columns = 'Class' , axis = 1)
         Y = new_dataFrame['Class']
```

```
In [34]: print(X)
```

	Time	V1	V2	V3	V4	V5	V6	\
46562	42836.0	-0.040184	-2.945298	0.551378	0.136728	-2.112744	0.627532	
163003	115560.0	2.017699	-0.418599	-2.562017	-0.715467	2.371590	3.324169	
154258	100940.0	-0.156578	0.162522	2.104589	-0.047495	0.051366	0.453470	
222837	143138.0	-0.100226	0.654743	0.087549	-0.249101	0.747274	-0.309196	
166461	118095.0	2.326970	-1.492014	-0.872100	-1.620743	-1.319074	-0.362659	
...	
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	\
46562	-0.336272	0.119987	-0.352596	...	0.844162	-0.102084	-1.023880	
163003	-0.456447	0.787120	0.401360	...	-0.120154	-0.283002	-0.843827	
154258	0.005426	-0.230959	1.897957	...	0.093824	-0.066492	0.472129	
222837	0.728941	0.058566	0.598702	...	-0.182464	-0.328871	-0.701658	
166461	-1.415057	-0.021397	-0.802626	...	-0.510608	-0.192358	-0.105021	
...	
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
46562	-0.374374	0.311548	-0.321080	0.840325	-0.112016	0.122923	659.20
163003	0.366668	0.699057	-0.271077	0.217451	-0.055362	-0.063963	17.99
154258	0.003764	-0.428468	-1.178449	0.481008	-0.285714	-0.256792	15.95
222837	0.078716	0.561749	-0.871132	0.115217	0.179840	0.243413	9.43
166461	0.283741	0.411888	-0.269307	-0.187400	0.013653	-0.045241	6.00
...
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]

In [35]:

```
print(Y)
```

```
46562      0
163003      0
154258      0
222837      0
166461      0
..
279863      1
280143      1
280149      1
281144      1
281674      1
```

Name: Class, Length: 984, dtype: int64

split the data into training data and testing data

In [36]:

```
X_train , X_test ,Y_train , Y_test = train_test_split(x ,y, test_size = 0.2 , stratify
```

In [37]:

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

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

In [39]:

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

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

model training

Linear regression model

In [40]:

```
model = LogisticRegression()
```

In [41]:

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

C:\Users\user\anaconda3\lib\site-packages\sklearn\linear_model_logistic.py:763: 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>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
Out[41]: LogisticRegression(  
    n_iter_i = _check_optimize_result(  
        n_iter_max=1000, tol=1e-05, verbose=0, warn_on_timeout=False)
```

Model Evaluation

Accuracy score

```
In [43]: #accuracy on training data  
X_train_prediction = model.predict(X_train)  
trainig_data_accuracy = accuracy_score(X_train_prediction , Y_train  
                                         )
```

```
In [44]: print('Accuracy on training data', trainig_data_accuracy)  
         #accuracy score above 70% is accepted
```

Accuracy on training data 0.9479034307496823

```
In [45]: #accuracy on test data  
X_test_prediction = model.predict(X_test)  
test_data_accuracy =accuracy_score(X_test_prediction , Y_test )
```

```
In [47]: print('Accuracy score of test data' , test_data_accuracy)
```

Accuracy score of test data 0.934010152284264

```
In [ ]:
```