

```

import numpy as np
import pandas as pd
import sklearn as sl
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

df = pd.read_csv("creditcard.csv")
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

```

```
df.head()
```

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

	V8	V9	...	V21	V22	V23	V24
V25 \							
0	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
0.128539							
1	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
0.167170							
2	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
0.327642							
3	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
0.647376							
4	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267
0.206010							

	V26	V27	V28	Amount	Class
0	-0.189115	0.133558	-0.021053	149.62	0
1	0.125895	-0.008983	0.014724	2.69	0
2	-0.139097	-0.055353	-0.059752	378.66	0
3	-0.221929	0.062723	0.061458	123.50	0
4	0.502292	0.219422	0.215153	69.99	0

```
[5 rows x 31 columns]
```

```
df.tail()
```

	Time	V1	V2	V3	V4
V5 \					
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828
284805	172788.0	-0.240440	0.530483	0.702510	0.689799
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271

	V6	V7	V8	V9	...	V21
V22 \						
284802	-2.606837	-4.918215	7.305334	1.914428	...	0.213454
0.111864						
284803	1.058415	0.024330	0.294869	0.584800	...	0.214205
0.924384						
284804	3.031260	-0.296827	0.708417	0.432454	...	0.232045
0.578229						
284805	0.623708	-0.686180	0.679145	0.392087	...	0.265245
0.800049						
284806	-0.649617	1.577006	-0.414650	0.486180	...	0.261057
0.643078						

	V23	V24	V25	V26	V27	V28
Amount \						
284802	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731
0.77						
284803	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527
24.79						
284804	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561
67.88						
284805	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533
10.00						
284806	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649
217.00						

	Class
284802	0
284803	0
284804	0
284805	0
284806	0

[5 rows x 31 columns]

#Cheking the number of Missing values in each column

```
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 transactions & fraudulent transactions

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

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

#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)
```

#Statistical measures of the data

```
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 for both transaction

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

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

	V6	V7	V8	V9	...	V20	V21
0	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235
1	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588

	V22	V23	V24	V25	V26	V27
0	-0.000024	0.000070	0.000182	-0.000072	-0.000089	-0.000295
1	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575

Amount

Class

0 88.291022

1 122.211321

[2 rows x 30 columns]

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

#Concatenating two Dataframes

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

```
new_dataset.head()
```

	Time	V1	V2	V3	V4	V5
V6 \						
1334	1042.0	-1.434116	0.052869	-0.159703	-2.385246	0.837335
0.864149						
138969	82942.0	-2.075623	0.760682	-0.390566	-0.984905	1.559705
3.963283						
27237	34454.0	-0.213402	0.536697	1.998777	0.118768	-0.199312
0.158182						
91636	63565.0	-0.932697	0.472401	2.518028	-1.199550	-0.529448
0.099109						
71567	54344.0	-0.746070	0.708378	1.128185	2.507941	-0.346745
0.851121						

	V7	V8	V9	...	V21	V22
V23 \						
1334	0.058739	0.443637	0.382264	...	0.128405	0.308762
0.416396						
138969	-0.428276	1.449244	0.328528	...	0.178604	0.364477
0.475819						
27237	0.493994	-0.102112	-0.433024	...	0.092834	0.404125
0.014461						
91636	0.040862	0.364532	0.589005	...	-0.046406	-0.011569
0.200882						
71567	0.250762	0.730630	-1.203883	...	0.050566	-0.105359
0.649769						

	V24	V25	V26	V27	V28	Amount
Class						
1334	-0.740970	-0.116688	-0.087127	0.194358	-0.020598	1.00
0						
138969	1.043961	0.190651	-0.516660	-0.831666	-0.267418	89.00
0						
27237	0.054632	-0.759193	0.319154	-0.026853	-0.062298	53.45
0						
91636	0.098508	0.065467	0.946610	0.226912	0.112529	6.77

```

0
71567 -0.030724 -1.446214 -0.406540 0.123396 0.168303 134.19
0

[5 rows x 31 columns]
new_dataset.tail()

```

	Time	V1	V2	V3	V4	V5
V6 \						
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	...	V21	V22
V23 \						
279863	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189
0.639419						
280143	-1.413170	0.248525	-1.127396	...	0.370612	0.028234
0.145640						
280149	-2.234739	1.210158	-0.652250	...	0.751826	0.834108
0.190944						
281144	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209
0.456108						
281674	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135
0.072173						

	V24	V25	V26	V27	V28	Amount
Class						
279863	-0.294885	0.537503	0.788395	0.292680	0.147968	390.00
1						
280143	-0.081049	0.521875	0.739467	0.389152	0.186637	0.76
1						
280149	0.032070	-0.739695	0.471111	0.385107	0.194361	77.89
1						
281144	-0.183659	-0.328168	0.606116	0.884876	-0.253700	245.00
1						
281674	-0.450261	0.313267	-0.289617	0.002988	-0.015309	42.53
1						

```

[5 rows x 31 columns]
new_dataset['Class'].value_counts()

```

```

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

new_dataset.groupby('Class').mean()

```

	Time	V1	V2	V3	V4	V5
0	94209.672764	-0.076244	0.016551	0.021377	0.030525	-0.025204
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225

	V6	V7	V8	V9	...	V20	V21
0	-0.059966	0.040193	-0.032970	0.026679	...	-0.016484	0.038301
1	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588

	V22	V23	V24	V25	V26	V27
0	0.004885	0.052448	0.012779	0.005824	-0.008914	0.011870
1	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575

	Amount
0	91.274614
1	122.211321


```

[2 rows x 30 columns]

```

#Splitting the data into Features & Targets

```

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

print(X)

```

	Time	V1	V2	V3	V4	V5
1334	1042.0	-1.434116	0.052869	-0.159703	-2.385246	0.837335

0.864149							
138969	82942.0	-2.075623	0.760682	-0.390566	-0.984905	1.559705	
3.963283							
27237	34454.0	-0.213402	0.536697	1.998777	0.118768	-0.199312	
0.158182							
91636	63565.0	-0.932697	0.472401	2.518028	-1.199550	-0.529448	-
0.099109							
71567	54344.0	-0.746070	0.708378	1.128185	2.507941	-0.346745	
0.851121							
...	
...							
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 \							
1334	0.058739	0.443637	0.382264	...	-0.035200	0.128405	
0.308762							
138969	-0.428276	1.449244	0.328528	...	-0.504098	0.178604	
0.364477							
27237	0.493994	-0.102112	-0.433024	...	0.306857	0.092834	
0.404125							
91636	0.040862	0.364532	0.589005	...	-0.028023	-0.046406	-
0.011569							
71567	0.250762	0.730630	-1.203883	...	0.134160	0.050566	-
0.105359							
...
.							
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							

```

1334    -0.416396 -0.740970 -0.116688 -0.087127  0.194358 -0.020598
1.00
138969 -0.475819  1.043961  0.190651 -0.516660 -0.831666 -0.267418
89.00
27237    0.014461  0.054632 -0.759193  0.319154 -0.026853 -0.062298
53.45
91636   -0.200882  0.098508  0.065467  0.946610  0.226912  0.112529
6.77
71567    0.649769 -0.030724 -1.446214 -0.406540  0.123396  0.168303
134.19
...      ...      ...      ...      ...      ...      ...
...
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]

```
print(Y)
```

```

1334      0
138969    0
27237     0
91636     0
71567     0
..
279863    1
280143    1
280149    1
281144    1
281674    1

```

Name: Class, Length: 984, dtype: int64

#Split the data into Training data & 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.shape, X_train.shape, X_test.shape)
```

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

#Training the logistic regression model with training data

```
model = LogisticRegression()
model.fit(X_train_scaled, Y_train)
LogisticRegression()
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training Data : ' , training_data_accuracy)
Accuracy on Training Data :  0.9504447268106735
```

#accuracy on test data

```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data : ' , test_data_accuracy)
Accuracy score on Test Data :  0.9187817258883249
model = LogisticRegression()
```