Importing the Dependencies

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")
```

```
# Loading the dataset
credit_card_data = pd.read_csv('/content/creditcard.csv')
```

```
# Displaying first 5 rows of the dataset
credit_card_data.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	0.098
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270

5 rows × 31 columns

```
# Dataset informations
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
---  ------  --------------   -----
 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 the number of missing values
credit_card_data.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
```

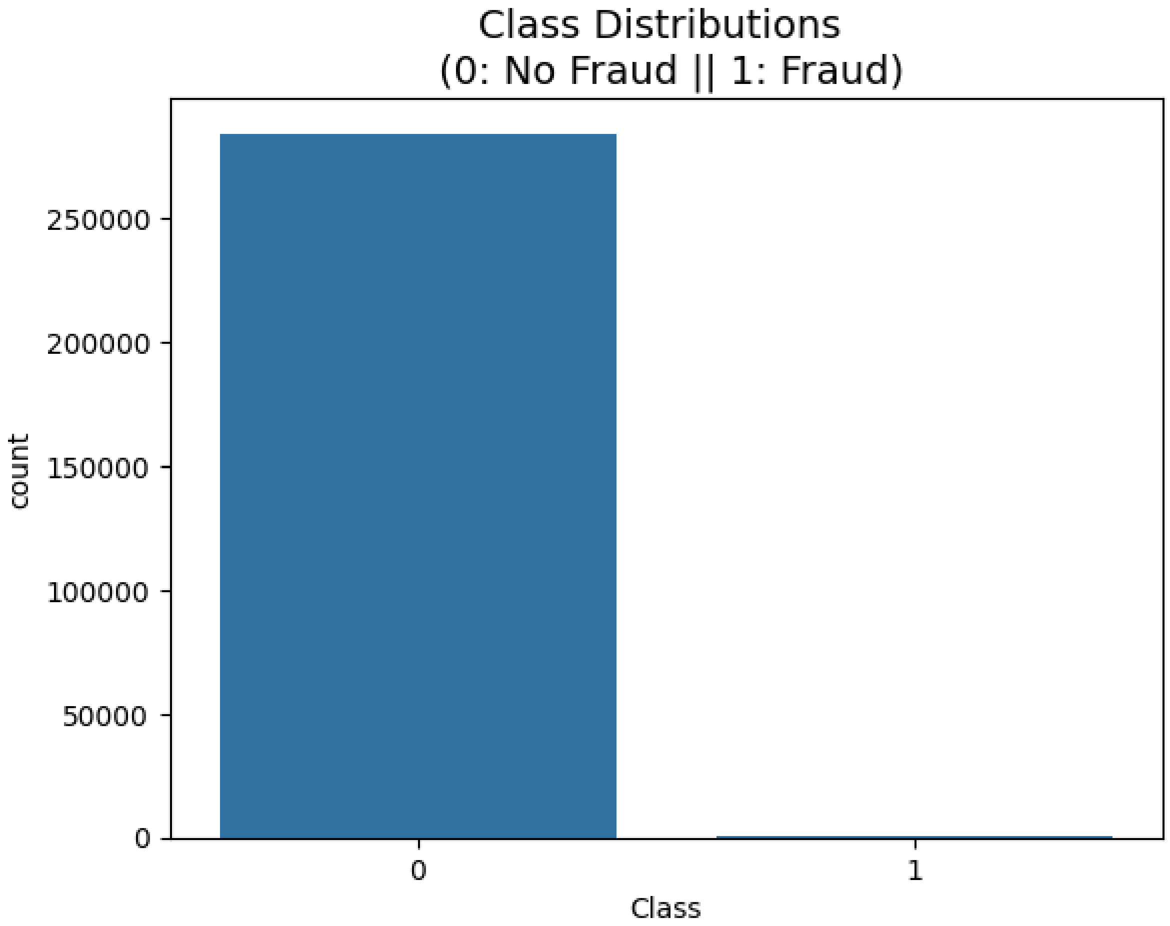
```
# Distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

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

This Dataset is highly unblanced

```
sb.countplot(data = credit_card_data, x = 'Class')
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize = 14)
```

```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```



0 --> Normal Transaction

1 --> fraudulent transaction

```
# Separating the data
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.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
```

```
# Comparing the values for both transactions
credit_card_data.groupby('Class').mean()
```

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5(

2 rows × 30 columns

Under-Sampling

Building a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions --> 492

1. List item
2. List item

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

Concatenating two DataFrames

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

```
new_dataset.head()
```

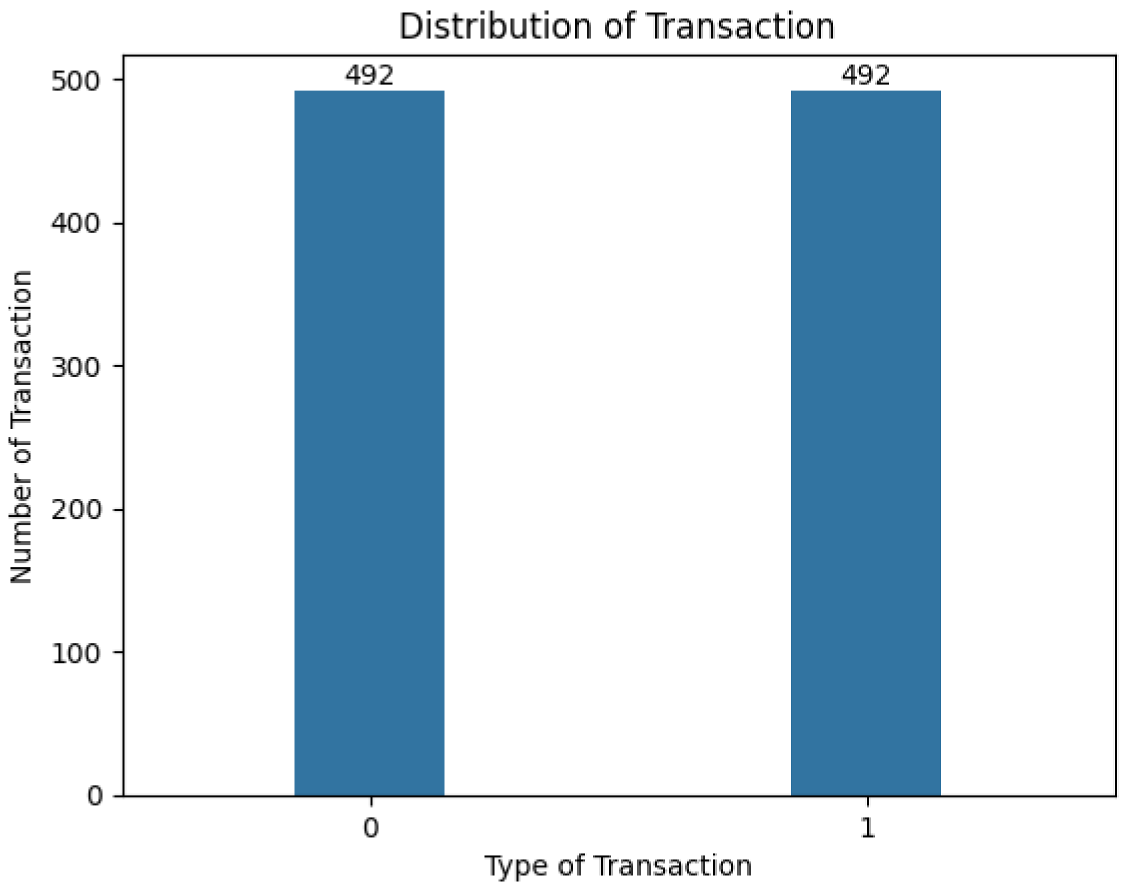
	Time	V1	V2	V3	V4	V5	V6	V
202612	134415.0	1.848186	-1.477551	0.896678	1.200279	-2.055843	0.804574	-1.8919
282560	170990.0	2.054361	-0.122642	-1.245717	0.189567	0.132497	-0.620765	0.0595
263618	161038.0	0.079717	1.052143	-1.718368	-0.870788	2.580570	-0.006619	1.4720
29800	35634.0	-1.903809	-0.753779	1.207583	0.334182	1.174934	-0.602482	1.0195
102601	68276.0	1.315431	-1.775045	1.496520	-1.002987	-2.341170	0.685621	-2.0597

5 rows × 31 columns

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

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

```
ax = sb.countplot(data = new_dataset, x = "Class", width = 0.3)
ax.set_title("Distribution of Transaction")
plt.xlabel("Type of Transaction")
plt.ylabel("Number of Transaction")
for i in ax.containers:
    ax.bar_label(i)
plt.show()
```




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

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	96698.682927	-0.051099	-0.160891	0.084828	0.054956	0.026079	-0.018868	0.01
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.56

2 rows × 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        V6  \
202612  134415.0  1.848186 -1.477551  0.896678  1.200279 -2.055843  0.804574
282560  170990.0  2.054361 -0.122642 -1.245717  0.189567  0.132497 -0.620765
263618  161038.0  0.079717  1.052143 -1.718368 -0.870788  2.580570 -0.006619
29800    35634.0 -1.903809 -0.753779  1.207583  0.334182  1.174934 -0.602482
102601   68276.0  1.315431 -1.775045  1.496520 -1.002987 -2.341170  0.685621
...          ...       ...       ...       ...       ...       ...       ...
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  \
202612 -1.891948  0.444322  1.536617  ... -0.662056 -0.226271  0.248479
282560  0.059581 -0.148058  0.338940  ... -0.209856 -0.271204 -0.687048
263618  1.472063 -0.168286 -0.821654  ...  0.084802  0.223777  0.647916
29800   1.019538 -0.034677 -0.670700  ...  0.763076  0.301158  0.047469
102601 -2.059707  0.395977 -0.494474  ... -0.301429 -0.172451  0.170433
...          ...       ...       ...  ...       ...       ...       ...
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
202612  0.172092 -0.129436 -0.302338 -0.466334  0.147297 -0.007813   58.42
282560  0.271569 -0.497120 -0.270115  0.208619 -0.076075 -0.075428    0.99
263618  0.020418  3.515344 -0.556051  0.332389  0.148672  0.294834   16.00
29800   0.253887 -0.607248  0.972496 -0.313825 -0.092791  0.095467  250.00
102601 -0.093914  0.028818  0.347013 -0.044191  0.095635  0.025081   49.00
...          ...       ...       ...       ...       ...       ...     ...
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]
```

```
print(Y)
```

```
202612    0
282560    0
263618    0
29800     0
102601    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, rand
```

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

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

Model Training

Logistic Regression

```
model = LogisticRegression()
```

```
# Training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

Accuracy Score

```
# Accuracy on training data
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 * 100)
```

```
Accuracy on Training data :  93.39263024142312
```

```
# 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 * 100)
```

```
Accuracy score on Test Data :  90.35532994923858
```

Model's Accuracy: 90.35%