Data Pre-Processing and visualisation:

```
[3] print("The data looks like :")
    print(X.head())
    print(X.columns)
```

```
The data looks like :
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -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    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
```

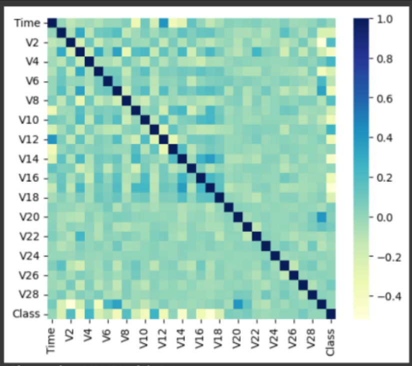
```
[4] print(X.describe())
```

```
               Time            V1            V2            V3            V4  \
count  45646.000000  45646.000000  45646.000000  45646.000000  45646.000000
mean   27545.441331     -0.237544      0.028942      0.696718      0.191505
std    12907.770469      1.886548      1.613070      1.530553      1.403936
min        0.000000    -56.407510    -72.715728    -32.965346     -5.172595
25%    19891.000000     -0.985248     -0.543868      0.222804     -0.714566
50%    32447.000000     -0.246259      0.088873      0.801638      0.191835
75%    37571.750000      1.157412      0.739723      1.434911      1.070924
max    42437.000000      1.960497     18.183626      4.101716     16.491217

                 V5            V6            V7            V8            V9  \
count  45646.000000  45645.000000  45645.000000  45645.000000  45645.000000
mean      -0.248407      0.098588     -0.117937      0.053331      0.157893
std        1.414608      1.308548      1.282818      1.210502      1.222268
min      -42.147898    -26.160506    -26.548144    -41.484823     -9.283925
25%       -0.853436     -0.638480     -0.600071     -0.148358     -0.589301
50%       -0.280620     -0.155353     -0.073143      0.054815      0.038334
75%        0.287608      0.487645      0.429944      0.324345      0.859738
max       34.801666     22.529298     36.677268     20.007208     10.392889

       ...           V21           V22           V23           V24  \
count  ...  45645.000000  45645.000000  45645.000000  45645.000000
mean   ...     -0.027098     -0.108684     -0.039060      0.009358
std    ...      0.733323      0.636670      0.572037      0.592176
min    ...    -20.262054     -8.593642    -26.751119     -2.836627
25%    ...     -0.232743     -0.529609     -0.179136     -0.322003
50%    ...     -0.070241     -0.083448     -0.051304      0.062230
75%    ...      0.105614      0.303459      0.077905      0.401392
max    ...     22.614889      5.805795     17.297845      4.014444
```

```
std         0.437824      0.502716      0.389241      0.338798    240.298594
min        -7.495741     -1.438650     -8.567638     -9.617915      0.000000
25%        -0.128065     -0.329709     -0.063670     -0.006837      7.580000
50%         0.175771     -0.067778      0.008425      0.021814     24.990000
75%         0.421857      0.302819      0.084017      0.076209     82.600000
max         5.525093      3.517346     11.135740     33.847808   7879.420000

              Class
count  45645.000000
mean       0.003111
std        0.055690
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000

[8 rows x 31 columns]
```

Correlations:

```python
# correlation heatmap
corrMatrix=X.corr()
sns.heatmap(data=corrMatrix, annot=False , cmap="YlGnBu")
plt.gcf().set_size_inches(6, 5)
plt.figure()
plt.show()
```



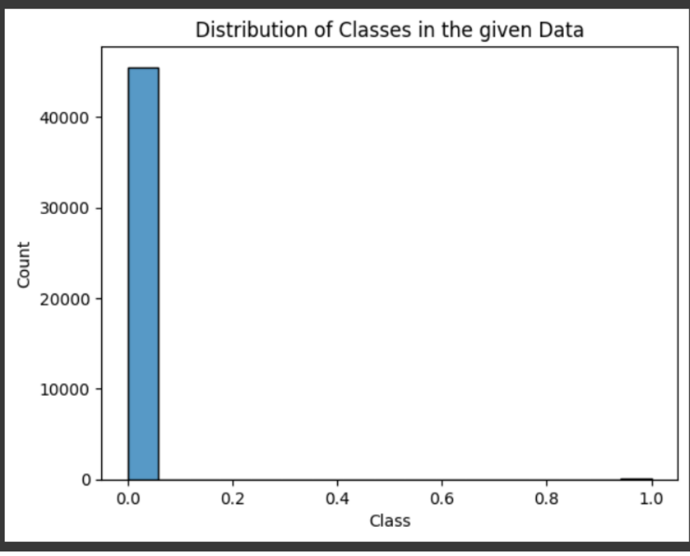

```
<Figure size 640x480 with 0 Axes>
```

The columns do not seem to have correlations with each other , and seem to have great correlation with the Class and time variables , hence being a great indicator that simple models would be helpful here.

Class Imbalance in dataset:

```python
sns.histplot(X['Class'])
plt.title("Distribution of Classes in the given Data")
print(X['Class'].value_counts())
print()
print()
plt.show()
# huge imbalance seen
```

```
Class
0.0    45503
1.0      142
Name: count, dtype: int64
```



To cure the imbalance, we can use the over sampling.

```
[28] # using SMOTE
     smote_1=SMOTE()
     X_train,y_train= smote_1.fit_resample(X_train,y_train)
```

Training and Testing dataset

```
[26] #dividing X and y
     y = X['Class']
     X.drop(['Class'],axis=1,inplace=True)
     X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=0)
```

```
[19] # using SMOTE
     smote_1=SMOTE()
     X_train,y_train= smote_1.fit_resample(X_train,y_train)
```

```
classifier = LogisticRegression(max_iter=150)
classifier.fit(X_train, y_train)
```

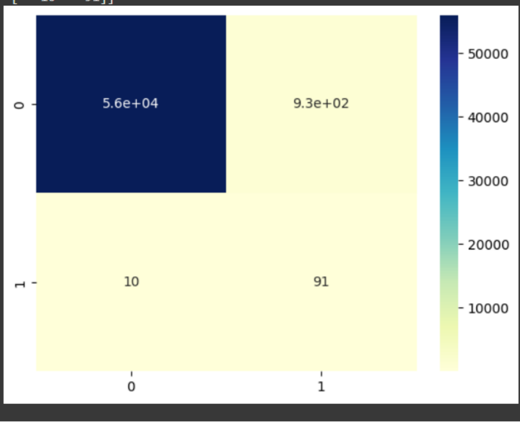
Test size is defined as 0.8 and 0.2, which implies that 80% of the data is used for training and the remaining 20% is used for testing. The data is divided into training and testing. As from the illustration above, our dataset was oversampled using SMOTE to counter the imbalance in the dataset.

RESULTS

```
from sklearn.metrics import classification_report,mean_absolute_error,mean_squared_error,r2_score
pred_train = classifier.predict(X_train)
report_train = classification_report(y_train,pred_train)
print(report_train)
pred_test = classifier.predict(X_test)
report_test = classification_report(y_test,pred_test)
print(report_test)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98    227454
           1       0.98      0.97      0.97    227454

    accuracy                           0.98    454908
   macro avg       0.98      0.98      0.98    454908
weighted avg       0.98      0.98      0.98    454908

              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56861
           1       0.09      0.90      0.16       101

    accuracy                           0.98     56962
   macro avg       0.54      0.94      0.58     56962
weighted avg       1.00      0.98      0.99     56962
```

```
Test result confusion matrix is :
[[55931   930]
 [   10    91]]
```




```
Pretty great , we only missed 11 frauds from detection out of 101 , 90% safety improvement here

Precision :0.9998212402352479
Recall :0.9836443256362005
f1_score_test : 0.991666814418184

The F1 score is very good hence my project is successful .
```

The F1 score came 0.99 meaning the Classifier is working great . It managed to catch 91 out of 101 frauds , thus preventing frauds 90% of the time