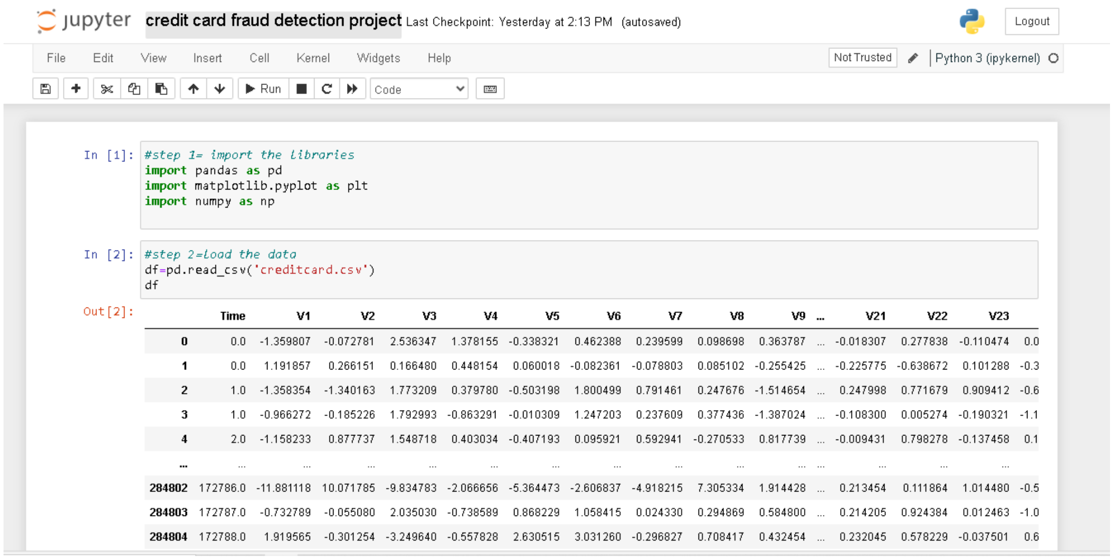
CREDIT CARD FRAUD DETECTION using machine learning algorithm
=

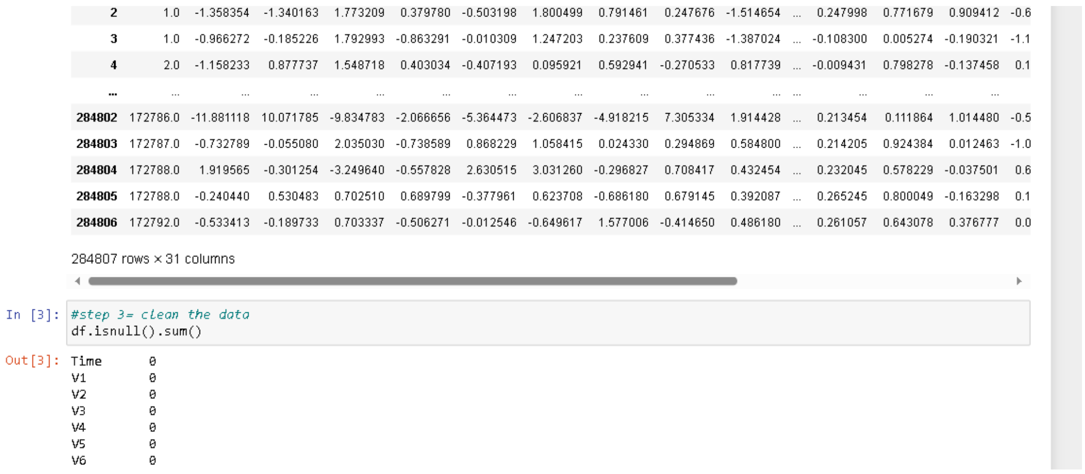
I used following steps to detect credit card fraud.

- Step 1= Import the libraries
- Step 2= load the data

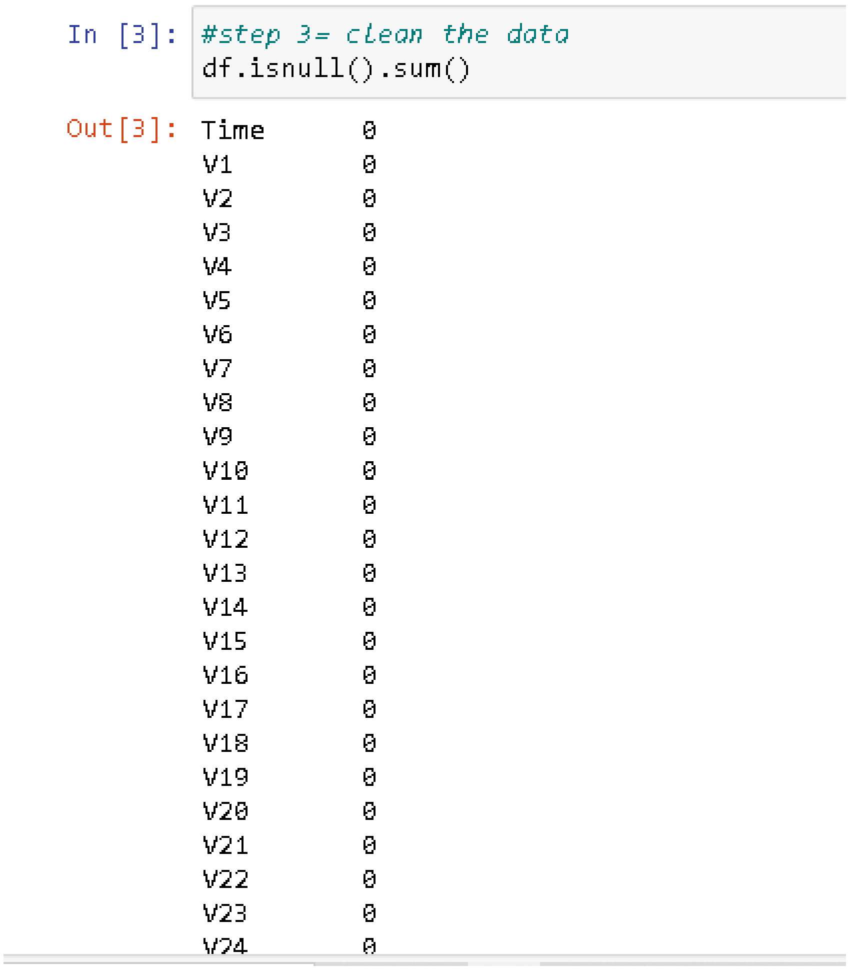

jupyter
credit card fraud detection project Last Checkpoint: Yesterday at 2:13 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

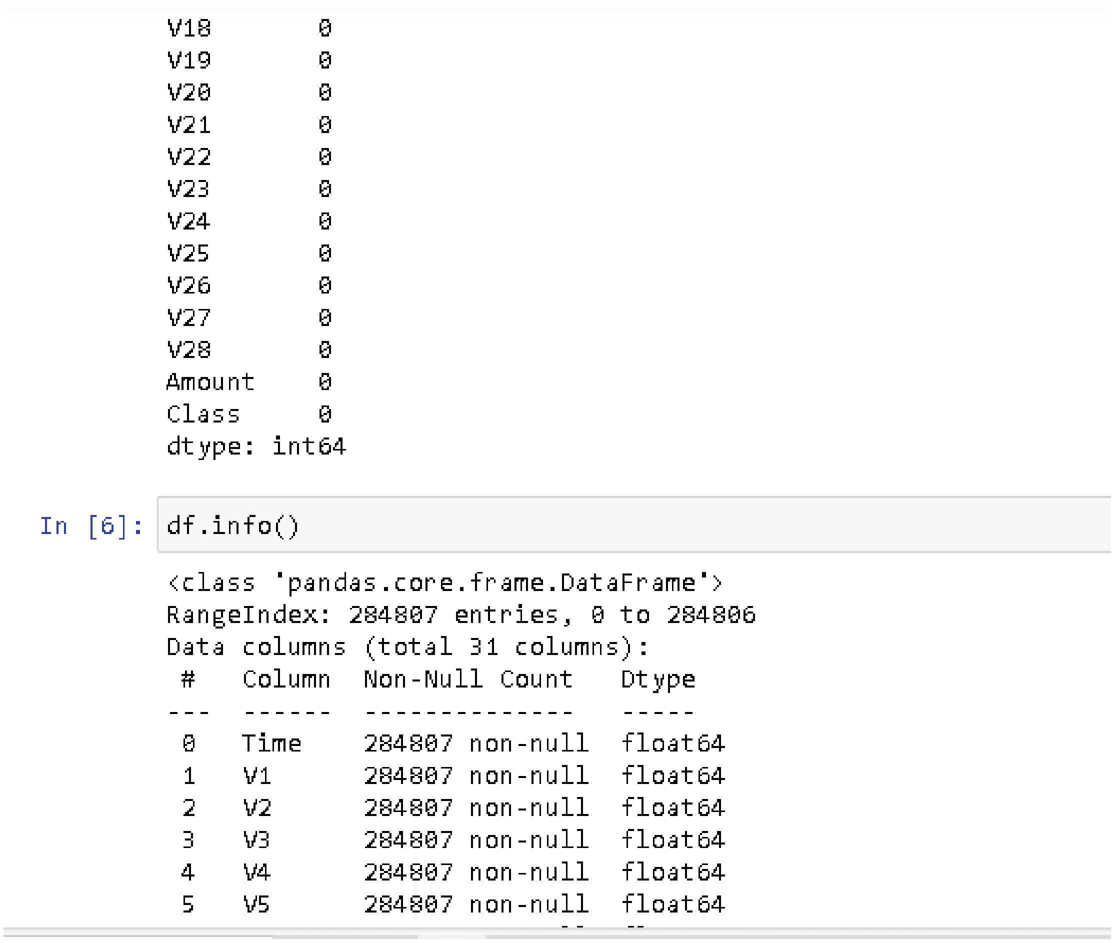
```
In [1]: #step 1= import the libraries
        import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np

In [2]: #step 2=load the data
        df=pd.read_csv('creditcard.csv')
        df

Out[2]:
              Time         V1         V2        V3        V4        V5        V6        V7        V8        V9  ...       V21       V22       V23
     0         0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.0
     1         0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.3
     2         1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.6
     3         1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.1
     4         2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.1
   ...         ...        ...        ...       ...       ...       ...       ...       ...       ...       ...  ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864  1.014480 -0.5
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384  0.012463 -1.0
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229 -0.037501  0.6
```

- Step 3= Clean the data



```
     2         1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.6
     3         1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.1
     4         2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.1
   ...         ...        ...        ...       ...       ...       ...       ...       ...       ...       ...  ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864  1.014480 -0.5
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384  0.012463 -1.0
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229 -0.037501  0.6
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049 -0.163298  0.1
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078  0.376777  0.0

284807 rows × 31 columns

In [3]: #step 3= clean the data
        df.isnull().sum()

Out[3]: Time      0
        V1        0
        V2        0
        V3        0
        V4        0
        V5        0
        V6        0
```

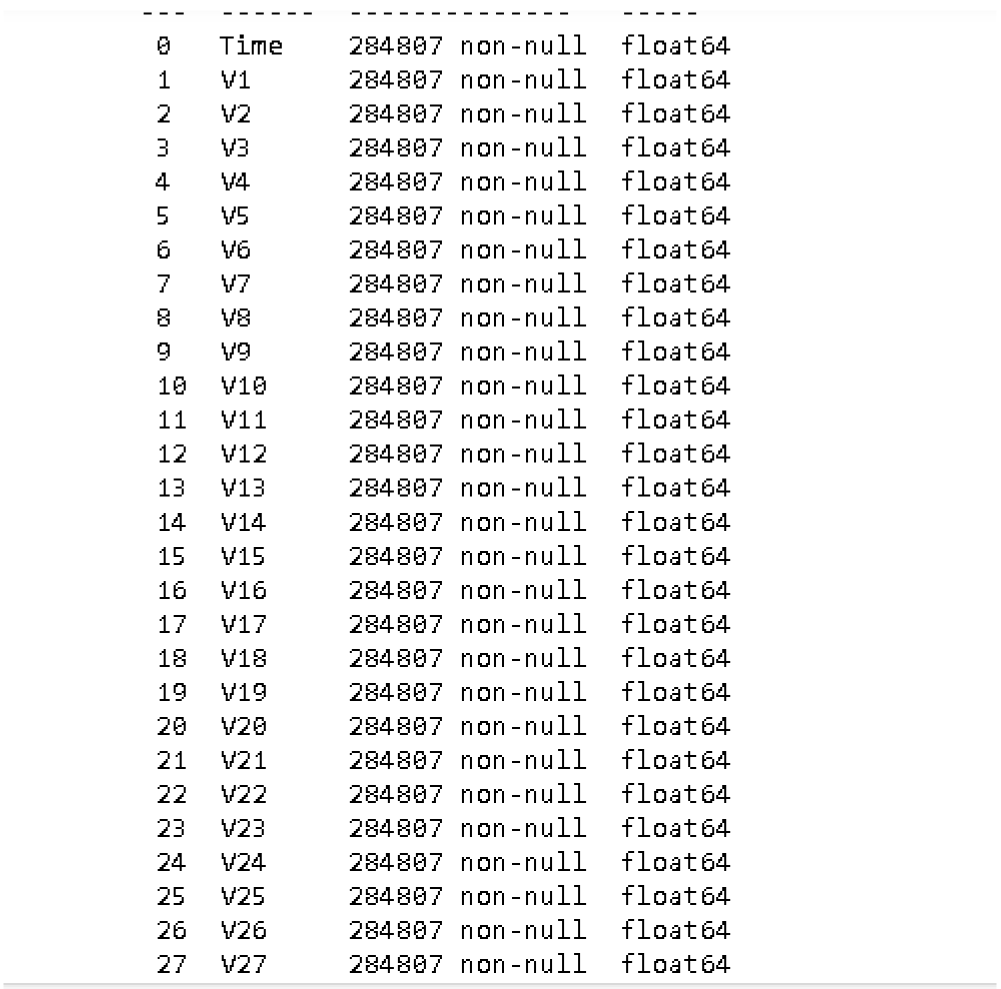

```
In [3]:  #step 3= clean the data
         df.isnull().sum()

Out[3]:  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
```



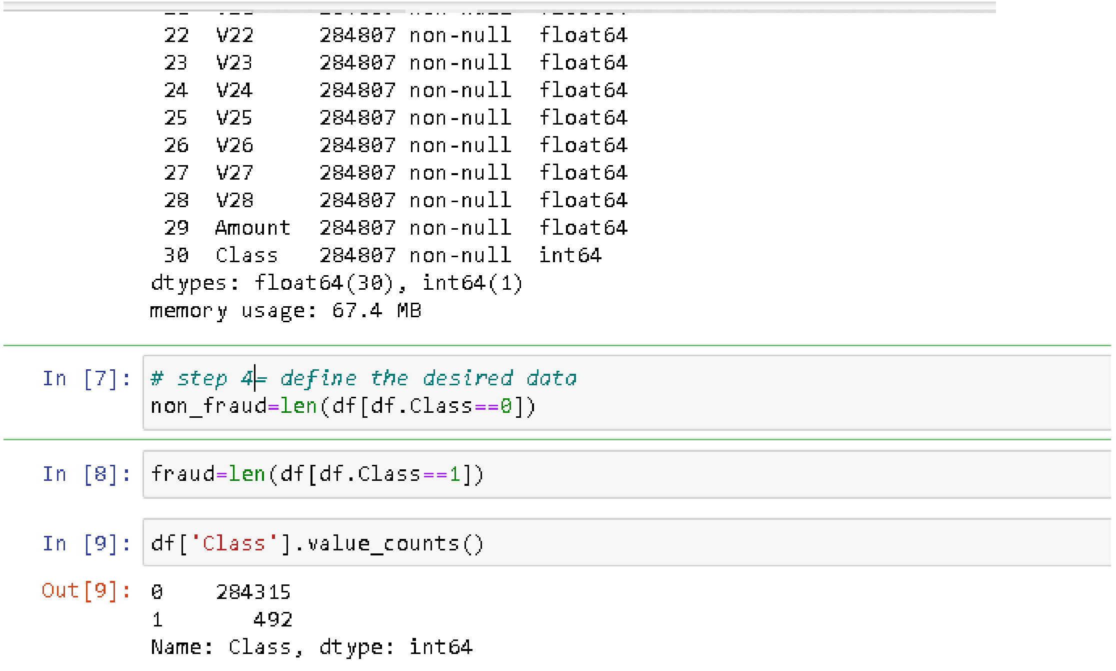
```
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [6]:
```
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
```



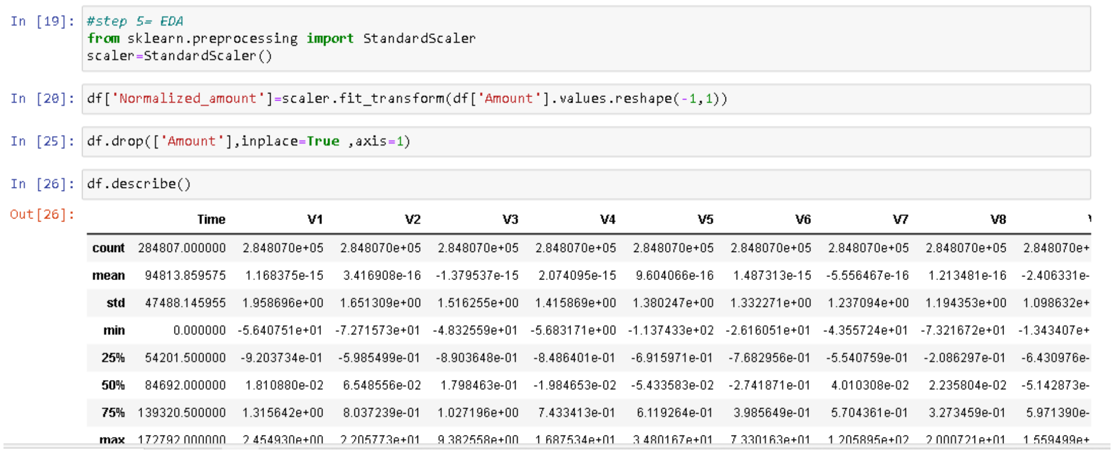
```
 ---  ------  --------------  -----
 0    Time    284807 non-null  float64
 1    V1      284807 non-null  float64
 2    V2      284807 non-null  float64
 3    V3      284807 non-null  float64
 4    V4      284807 non-null  float64
 5    V5      284807 non-null  float64
 6    V6      284807 non-null  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
```



```
 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
```

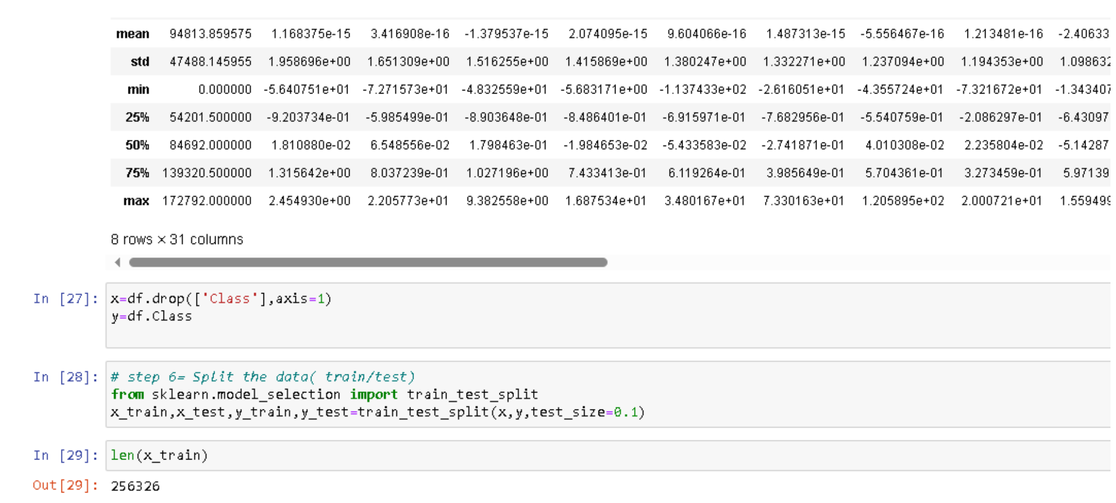
```
In [7]: # step 4= define the desired data
        non_fraud=len(df[df.Class==0])

In [8]: fraud=len(df[df.Class==1])

In [9]: df['Class'].value_counts()

Out[9]: 0    284315
        1       492
        Name: Class, dtype: int64
```

- Step 4= Define the desire data
- Step 5=EDA



```
In [19]: #step 5= EDA
         from sklearn.preprocessing import StandardScaler
         scaler=StandardScaler()

In [20]: df['Normalized_amount']=scaler.fit_transform(df['Amount'].values.reshape(-1,1))

In [25]: df.drop(['Amount'],inplace=True ,axis=1)

In [26]: df.describe()

Out[26]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+

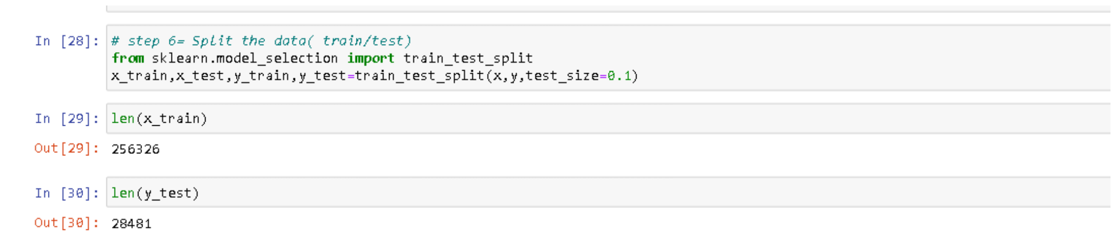


mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.40633
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.09863
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.43097
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.14287
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.97139
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.55949

8 rows × 31 columns

```
In [27]: x=df.drop(['Class'],axis=1)
         y=df.Class

In [28]: # step 6= Split the data( train/test)
         from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1)

In [29]: len(x_train)

Out[29]: 256326
```

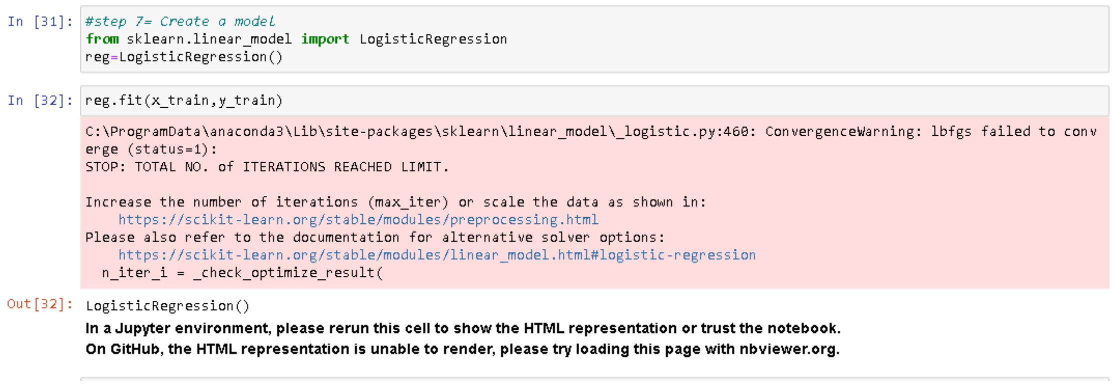
- Step 6= Split the dataset(train/test)



```
In [28]: # step 6= Split the data( train/test)
         from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1)

In [29]: len(x_train)

Out[29]: 256326
```

```
In [30]: len(y_test)

Out[30]: 28481
```

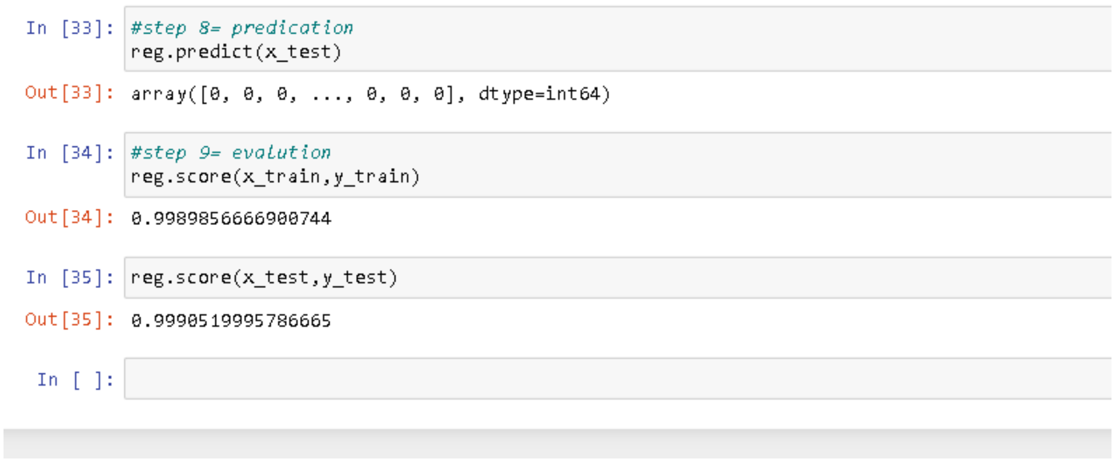
- Step 7= Create a Model



```
In [31]: #step 7= Create a model
         from sklearn.linear_model import LogisticRegression
         reg=LogisticRegression()

In [32]: reg.fit(x_train,y_train)

         C:\ProgramData\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to conv
         erge (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
           n_iter_i = _check_optimize_result(

Out[32]: LogisticRegression()
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

- Step 8= prediction

- Step 9= Evalution



```
In [33]: #step 8= predication
         reg.predict(x_test)

Out[33]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

In [34]: #step 9= evalution
         reg.score(x_train,y_train)

Out[34]: 0.9989856666900744

In [35]: reg.score(x_test,y_test)

Out[35]: 0.9990519995786665

In [ ]:
```