import the libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
```

load the data

In [2]:
```
data = pd.read_csv("/home/tamanna/Downloads/creditcard.csv")
data
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.

284807 rows × 31 columns

data preprocessing

In [3]:
```
data.head()
```

Out[3]:

	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	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2

5 rows × 31 columns

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

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.(
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.:
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.(
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.!

5 rows × 31 columns

In [5]:
```
# generating descriptive statistics
data.describe()
```

Out[5]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848(
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380:
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137.
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.91!
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.11(
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.48(

8 rows × 31 columns

In [6]:
```
data.shape
```

Out[6]: (284807, 31)

In [7]:
```
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
```

In [8]:
```
data.isnull().sum()
```

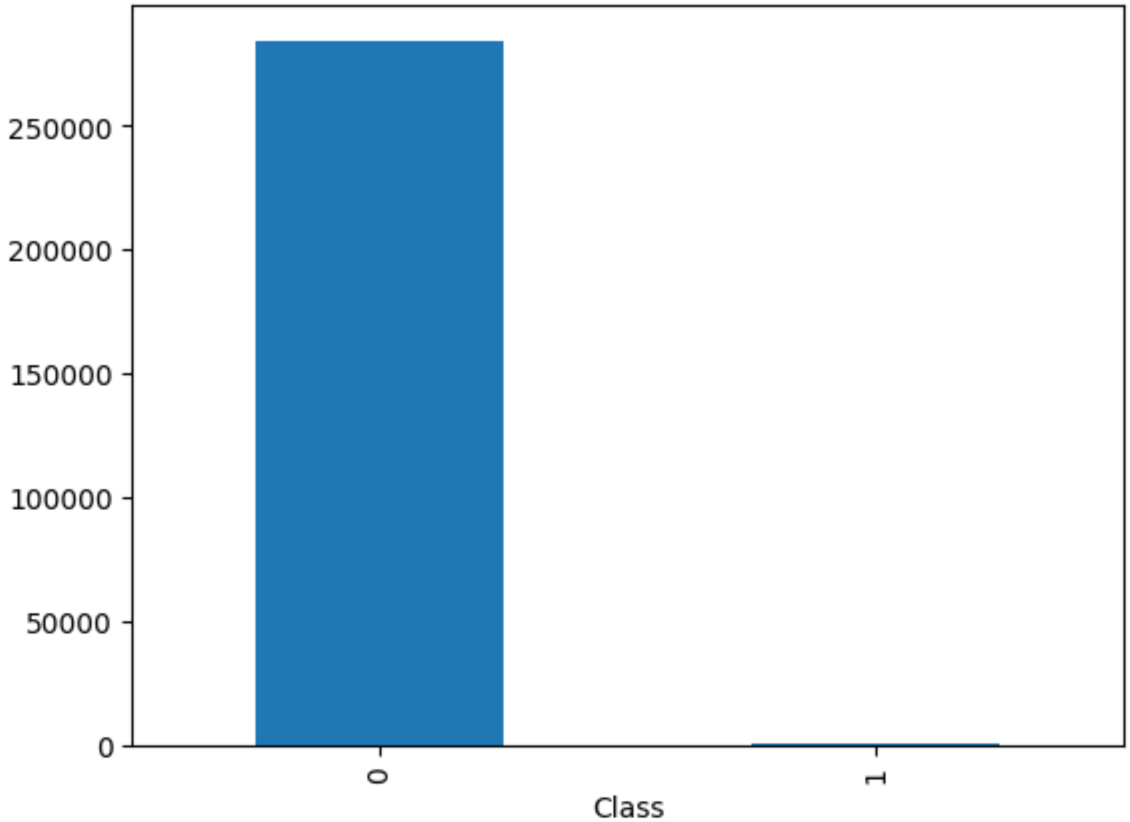
Out[8]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [9]:
```
data['Class'].value_counts()
```

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

In [10]:
```
data['Class'].value_counts().plot(kind = 'bar')
```

Out[10]:
```
<AxesSubplot: xlabel='Class'>
```



dataset is highly imbalanced.

In [11]:
```
non_fraud = data[data.Class == 0]
fraud = data[data.Class == 1]
```

In [12]:
```
non_fraud.shape
```

Out[12]: (284315, 31)

In [13]:
```
fraud.shape
```

Out[13]: (492, 31)

In [14]:
```
non_fraud_sample = non_fraud.sample(600)
```

In [15]:
```
non_fraud_sample
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	
79938	58251.0	-1.053308	0.588041	0.497243	0.514123	-0.243353	-1.143231	0.5
94158	64757.0	-4.283375	3.543912	-2.966651	-1.359581	0.587368	2.866529	-1.7
49211	43957.0	1.018221	0.085412	-0.353711	1.066409	0.340708	-0.150033	0.4
267573	162831.0	2.133450	0.005781	-2.323607	0.049250	0.971403	-0.498335	0.5
138673	82787.0	-0.935118	1.202229	1.257032	0.404653	0.133229	0.160176	0.1
...	...	...	...	...	...	...	...	
212291	138799.0	-0.878858	0.645186	-0.272966	-1.193022	3.076594	3.633350	0.2
27185	34430.0	1.209556	-0.166461	1.181014	0.565935	-0.981248	-0.130360	-0.7
93551	64483.0	1.105351	-1.457671	0.749924	-2.015344	-1.728027	-0.056765	-1.1
185371	126638.0	2.101841	0.625172	-2.668427	0.594459	0.963007	-1.364090	0.4
283626	171743.0	0.425363	-0.062341	-0.857997	0.326586	1.822954	-0.914518	0.3

600 rows × 31 columns

In [16]:
```
new_data = pd.concat([non_fraud_sample,fraud], axis =0)
```

In [17]:
```
new_data
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	
79938	58251.0	-1.053308	0.588041	0.497243	0.514123	-0.243353	-1.143231	0.51
94158	64757.0	-4.283375	3.543912	-2.966651	-1.359581	0.587368	2.866529	-1.74
49211	43957.0	1.018221	0.085412	-0.353711	1.066409	0.340708	-0.150033	0.48
267573	162831.0	2.133450	0.005781	-2.323607	0.049250	0.971403	-0.498335	0.51
138673	82787.0	-0.935118	1.202229	1.257032	0.404653	0.133229	0.160176	0.10
...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.88
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.41
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.23
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.20
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.22

1092 rows × 31 columns

values of x and y

In [18]:
```
x = new_data.drop(['Class'], axis = 1)
```

In [19]:
```
print(x)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
79938    58251.0 -1.053308  0.588041  0.497243  0.514123 -0.243353 -1.143231
94158    64757.0 -4.283375  3.543912 -2.966651 -1.359581  0.587368  2.866529
49211    43957.0  1.018221  0.085412 -0.353711  1.066409  0.340708 -0.150033
267573  162831.0  2.133450  0.005781 -2.323607  0.049250  0.971403 -0.498335
138673   82787.0 -0.935118  1.202229  1.257032  0.404653  0.133229  0.160176
...          ...       ...       ...       ...       ...       ...       ...
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  \
79938   0.516954  0.188210 -0.619711  ... -0.083019  0.128897  0.083167
94158  -1.748133  3.303362 -0.647340  ...  0.147722  0.057162 -0.549032
49211   0.484130 -0.056734 -0.563302  ...  0.021810  0.117429  0.155288
267573  0.518368 -0.259087 -0.017044  ... -0.206715  0.098535  0.411672
138673  0.103757  0.633826 -1.232453  ...  0.066806 -0.115048 -0.531792
...          ...       ...       ...  ...       ...       ...       ...
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
79938   0.014889  0.415131 -0.413715  0.457807 -0.151270  0.062739   72.98
94158   0.024283  1.006805  1.059677  0.419387 -0.541254  0.052017    6.76
49211  -0.250932 -0.289248  0.743356 -0.257872 -0.017277  0.010981  100.75
267573 -0.081135  0.165554  0.441014  0.700315 -0.124472 -0.095485    0.76
138673 -0.110064 -0.339523 -0.035581  0.308557 -0.068997 -0.007041    9.99
...          ...       ...       ...       ...       ...       ...     ...
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

[1092 rows x 30 columns]
```

In [20]:
```
y = new_data["Class"]
```

In [21]:
```
print(y)
```

```
79938     0
94158     0
49211     0
267573    0
138673    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 1092, dtype: int64
```

splitting of data into training and testing sets

In [22]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, random_state = 2, t
```

In [23]:
```
print(x.shape, x_train.shape, x_test.shape)
```

```
(1092, 30) (982, 30) (110, 30)
```

Logistic Regression

In [24]:
```
model = LogisticRegression()
```

In [25]:
```
model.fit(x_train, y_train)
```

Out[25]:
```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

In [26]:
```
x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
```

In [27]:
```
training_data_accuracy
```

Out[27]:
```
0.9480651731160896
```

In [28]:
```
x_test_prediction = model.predict(x_test)
testing_data_accuracy = accuracy_score(x_test_prediction, y_test)
```

In [29]:
```
testing_data_accuracy
```

Out[29]:
```
0.9454545454545454
```

svm classifier

In [30]:
```
model2 = svm.SVC(kernel = "linear")
```

In [31]:
```
model2.fit(x_train, y_train)
```

Out[31]:
```
▾ SVC
SVC(kernel='linear')
```

In [32]:
```
x_train_prediction =model2.predict(x_train)
```

Model Evaluation

In [33]:
```
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
```

In [34]:
```
training_data_accuracy
```

Out[34]:
```
0.8981670061099797
```

In [35]:
```
x_test_prediction = model2.predict(x_test)
```

In [36]:
```
testing_data_accuracy = accuracy_score(x_test_prediction, y_test)
testing_data_accuracy
```

Out[36]:
```
0.9181818181818182
```

Decision Tree Classifier

In [37]:
```
model3 = DecisionTreeClassifier(max_leaf_nodes = 10, random_state = 0)
model3.fit(x_train, y_train)
```

Out[37]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_leaf_nodes=10, random_state=0)
```

In [39]:
```
x_train_prediction = model3.predict(x_train)
```

Model Evaluation

In [40]:
```
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
training_data_accuracy
```

Out[40]:
```
0.955193482688391
```

In [41]:
```
x_test_prediction = model3.predict(x_test)
```

In [42]:
```
testing_data_accuracy = accuracy_score(x_test_prediction, y_test)
```

In [43]:
```
testing_data_accuracy
```

Out[43]:
```
0.9181818181818182
```

logistic regression has got the highest accuracy as compared to both decision tree classifier and svm