Importing the Dependencies

In [32]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
credit_card_data = pd.read_csv(r'C:\Users\Manas Ranjan Kar\Downloads\creditcard.csv\creditcard.csv')
```

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

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	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	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
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	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
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	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
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	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

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

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
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	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	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	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
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	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
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	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

5 rows × 31 columns

In [5]:
```
# 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
```

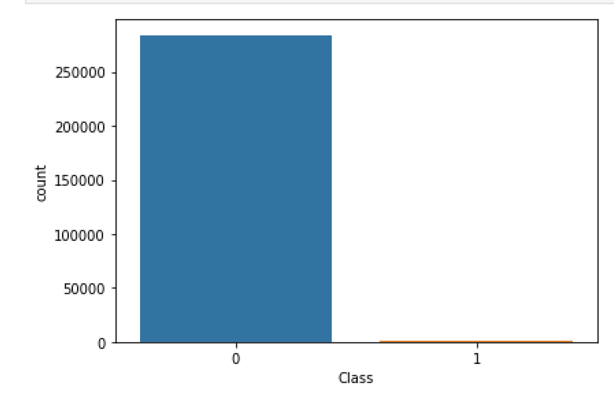
In [6]:
```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[6]:
```
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 [7]:
```
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

Out[7]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [34]:
```
fig, ax = plt.subplots(figsize = (6,4))
ax = sns.countplot(x= 'Class', data = credit_card_data)
plt.tight_layout()
```



Dataset is highly unblanced

In [8]:
```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [9]:
```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

In [10]:
```
legit.Amount.describe()
```

Out[10]:
```
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
```

In [11]:
```
fraud.Amount.describe()
```

Out[11]:
```
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
```

In [12]:
```
credit_card_data.groupby('Class').mean()
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amo
Class																					
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	0.000070	0.000182	-0.000072	-0.000089	-0.000295	-0.000131	88.291
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575	0.075667	122.211

2 rows × 30 columns

Under-Sampling

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

Number of Fraudulent Transactions --> 492

In [13]:
```
legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

In [14]:
```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

In [15]:
```
new_dataset.head()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
238819	149836.0	2.094767	-0.641661	-1.579913	-0.421893	-0.172272	-0.610978	-0.135759	-0.170210	-0.625563	...	-0.407511	-0.677551	0.240255	0.702415	-0.003108	-0.431915	-0.019188	-0.054319	29.77	0
263306	160884.0	-0.851774	0.183403	0.768546	-1.048717	1.345062	-1.006637	0.677157	0.012604	-0.448166	...	-0.146588	-0.659110	-0.048541	-0.358702	-0.183530	0.099422	0.010924	0.122436	1.98	0
283793	171879.0	-2.535260	3.078355	-2.779635	-1.710399	0.641377	-1.175867	0.863469	0.592606	1.007590	...	-0.023107	0.544271	-0.113326	0.037426	0.227732	0.086414	0.520846	0.117049	0.77	0
91535	63511.0	-1.259089	1.062665	2.001524	1.071766	-0.769207	0.421672	-1.076257	-2.428427	-0.072034	...	-1.103739	1.159688	-0.095516	0.403683	-0.053440	-0.190208	0.444482	0.207686	59.35	0
13984	24847.0	1.100239	0.191058	0.456758	1.239196	-0.219356	-0.272331	-0.130156	0.045394	1.234513	...	-0.034779	0.058715	-0.046182	0.132083	0.470556	-0.364423	-0.015183	0.003027	27.95	0

5 rows × 31 columns

In [16]:
```
new_dataset.tail()
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0.292680	0.147968	390.00	1
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.145640	-0.081049	0.521875	0.739467	0.389152	0.186637	0.76	1
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0.385107	0.194361	77.89	1
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.456108	-0.183659	-0.328168	0.606116	0.884876	-0.253700	245.00	1
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.072173	-0.450261	0.313267	-0.289617	0.002988	-0.015309	42.53	1

5 rows × 31 columns

In [17]:
```
new_dataset['Class'].value_counts()
```

Out[17]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [18]:
```
new_dataset.groupby('Class').mean()
```

Out[18]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amoun
Class																					
0	95010.556911	-0.130465	-0.030630	-0.029698	0.075940	0.041759	-0.076566	0.089686	0.017891	-0.036455	...	0.017307	-0.004261	-0.028226	0.030663	0.011203	0.004100	-0.003210	-0.032834	-0.007245	99.782825
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575	0.075667	122.211321

2 rows × 30 columns

Splitting the data into Features & Targets

In [19]:
```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

In [20]:
```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
238819  149836.0  2.094767 -0.641661 -1.579913 -0.421893 -0.172272 -0.610978   
263306  160884.0 -0.851774  0.183403  0.768546 -1.048717  1.345062 -1.006637   
283793  171879.0 -2.535260  3.078355 -2.779635 -1.710399  0.641377 -1.175867   
91535    63511.0 -1.259089  1.062665  2.001524  1.071766 -0.769207  0.421672   
13984    24847.0  1.100239  0.191058  0.456758  1.239196 -0.219356 -0.272331   
...          ...       ...       ...       ...       ...       ...       ...   
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  \
238819 -0.135759 -0.170210 -0.625563  ... -0.617201 -0.407511 -0.677551   
263306  0.677157  0.012604 -0.448166  ... -0.056549 -0.146588 -0.659110   
283793  0.863469  0.592606  1.007590  ...  0.803245 -0.023107  0.544271   
91535  -1.076257 -2.428427 -0.072034  ...  0.763939 -1.103739  1.159688   
13984  -0.130156  0.045394  1.234513  ... -0.271009 -0.034779  0.058715   
...          ...       ...       ...  ...       ...       ...       ...   
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  
238819  0.240255  0.702415 -0.003108 -0.431915 -0.019188 -0.054319   29.77  
263306 -0.048541 -0.358702 -0.183530  0.099422  0.010924  0.122436    1.98  
283793 -0.113326  0.037426  0.227732  0.086414  0.520846  0.117049    0.77  
91535  -0.095516  0.403683 -0.053440 -0.190208  0.444482  0.207686   59.35  
13984  -0.046182  0.132083  0.470556 -0.364423 -0.015183  0.003027   27.95  
...          ...       ...       ...       ...       ...       ...     ...  
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]
```

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

```
238819    0
263306    0
283793    0
91535     0
13984     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

In [22]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

In [23]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

Model Training

Logistic Regression

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

In [25]:
```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

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

Model Evaluation

Accuracy Score

In [26]:
```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [27]:
```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9479034307496823
```

In [28]:
```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [29]:
```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9187817258883249
```

Random Forest

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=10, criterion="entropy", random_state=0)
classifier.fit(X_train, Y_train)
```

Out[74]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [75]:
```
y_pred = classifier.predict(X_test)
```

In [76]:
```
test_data_accuracy = accuracy_score(y_pred, Y_test)
```

In [77]:
```
test_data_accuracy
```

Out[77]:
```
0.9187817258883249
```

In [78]:
```
Y_pred = classifier.predict(X_train)
```

In [79]:
```
test_data_accuracy = accuracy_score(Y_pred, Y_train)
```

In [80]:
```
test_data_accuracy
```

Out[80]:
```
0.9936467598475223
```

Decision Tree

In [82]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [86]:
```
DT = DecisionTreeClassifier(max_depth = 8, criterion = 'entropy')
DT.fit(X_train, Y_train)
dt_yhat = DT.predict(X_test)
```

In [87]:
```
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(Y_test, dt_yhat)))
```

```
Accuracy score of the Decision Tree model is 0.8883248730964467
```

In [88]:
```
dt_xhat = DT.predict(X_train)
```

In [90]:
```
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(Y_train, dt_xhat)))
```

```
Accuracy score of the Decision Tree model is 0.98856416772554
```

In [ ]: