In [1]:

```
import numpy as np
import pandas as pd
```

The dataset contains transactions made by credit cards in September 2013 by European cardholders.

In [2]:

```
# import the library we will use
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

load the csv file

In [3]:

```
data = pd.read_csv('./creditcard.csv')
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81

5 rows × 31 columns

In [4]:

```
data.shape
```

Out[4]:

```
(284807, 31)
```

In [5]:

```
data.describe()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7

8 rows × 31 columns

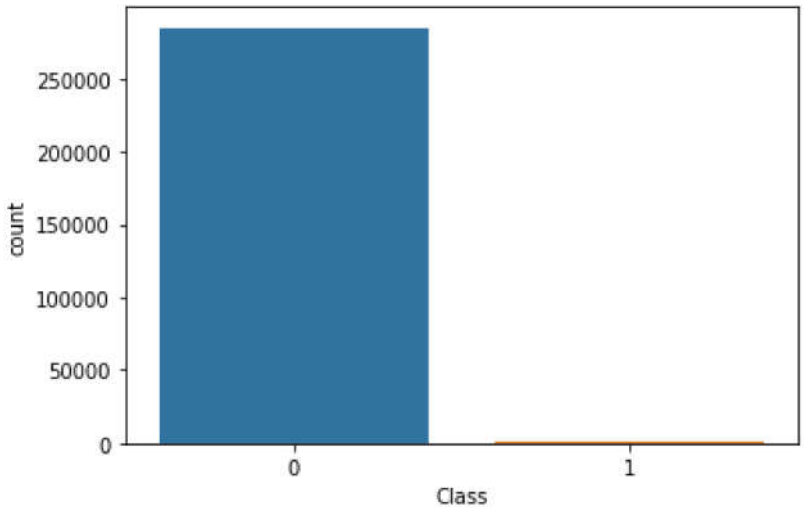
In [6]:

```
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 [7]:

```
fig, ax=plt.subplots(figsize=(6,4))
ax=sns.countplot(x='Class',data=data)
```



In [8]:

```
print('Total # transactions :',data.Class.count())
print('Total # genuine transactions :',data[data['Class'] == 0].shape[0])
print('Total # fraud transactions :',data[data['Class' ]== 1].shape[0])
```

```
Total # transactions : 284807
Total # genuine transactions : 284315
Total # fraud transactions : 492
```

we can see from this countplot, the classes 0 & 1 are highly imbalanced. where 0 --> genuine transaction 1 --> fraud transaction

In [9]:

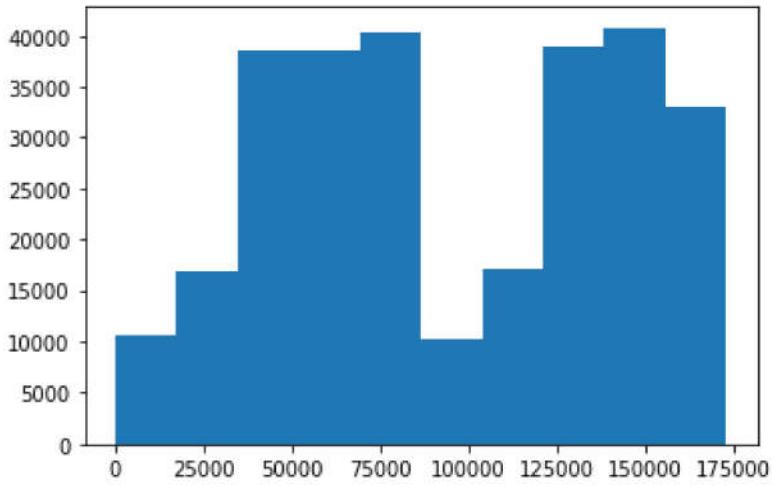
```
data['Time'].describe()
```

Out[9]:

```
count    284807.000000
mean      94813.859575
std       47488.145955
min           0.000000
25%       54201.500000
50%       84692.000000
75%      139320.500000
max      172792.000000
Name: Time, dtype: float64
```

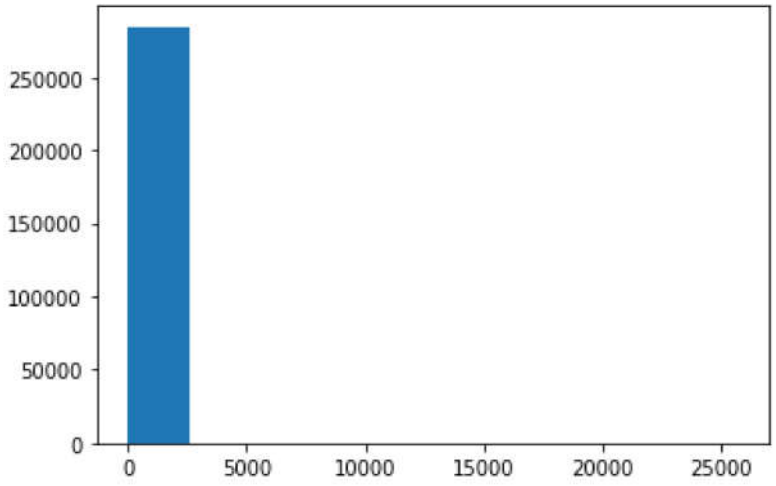
In [10]:

```
plt.hist(x='Time',data=data)
plt.show()
```



In [11]:

```
plt.hist(x='Amount',data=data)
plt.show()
```



In [12]:

```
data.Amount.describe().round(2)
```

Out[12]:

```
count    284807.00
mean         88.35
std         250.12
min           0.00
25%           5.60
50%          22.00
75%          77.16
max       25691.16
Name: Amount, dtype: float64
```

In [13]:

```
print('Maximum Amount : ' , data.Amount.max())
print('Minimum Amount : ', data.Amount.min())
```

```
Maximum Amount :  25691.16
Minimum Amount :  0.0
```

Data cleaning(remove the unwanted features)

In [14]:

```
data.drop(['Time'], axis=1, inplace=True)
```

In [15]:

```
data.drop_duplicates(inplace=True)
```

In [16]:

```
data.shape
```

Out[16]:

```
(275663, 30)
```

In [17]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
amount = data['Amount'].values
data['Amount'] = sc.fit_transform(amount.reshape(-1, 1))
```

Train & Test Split

In [18]:

```
X = data.drop('Class', axis = 1).values
y = data['Class'].values
```

In [19]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state =
```

Model Building

In [20]:

```
from sklearn.tree import DecisionTreeClassifier
```

Training data

In [21]:

```
dtc=DecisionTreeClassifier().fit(X_train, y_train)
dtc_pred=dtc.predict(X_test)
```

In [22]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

print('Accuracy : {}'.format(accuracy_score(y_test, dtc_pred)))
```

```
Accuracy : 0.9990689125624251
```

print result set

In [23]:

```
dtc_pred
```

Out[23]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [24]:

```
data.head()
```

Out[24]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 30 columns