Credit card fraud detection

In [102]:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [2]:
```
credit_data = pd.read_csv('creditcard.csv')
```

In [3]:
```
credit_data.head(5)
```

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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

Data preprocessing

In [4]:
```
credit_data.shape
```

Out[4]:
```
(284807, 31)
```

In [5]:
```
credit_data.size
```

Out[5]:
```
8829017
```

In [6]:
```
credit_data.columns
```

Out[6]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [7]:
```
credit_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]:
```
credit_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]:
```
credit_data.tail(5)
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

5 rows × 31 columns

In [10]:
```
print('Number of rows', credit_data.shape[0])
print('Number of columns', credit_data.shape[1])
```

```
Number of rows 284807
Number of columns 31
```

In [11]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
sc=StandardScaler()
credit_data['Amount'] = sc.fit_transform(pd.DataFrame(credit_data['Amount']
```

In [13]:
```
credit_data.head()
```

Out[13]:

	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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [14]:
```
credit_data = credit_data.drop(['Time'], axis=1)
```

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

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817

5 rows × 30 columns

In [16]:
```
credit_data.duplicated().any()
```

Out[16]:
```
True
```

In [17]:
```
credit_data = credit_data.drop_duplicates()
```

In [18]:
```
credit_data.shape
```

Out[18]:
```
(275663, 30)
```

In [19]:
```
credit_data['Class'].value_counts()
```

Out[19]:
```
Class
0    275190
1       473
Name: count, dtype: int64
```

In [25]:
```
legit = credit_data[credit_data.Class==0]
fraud = credit_data[credit_data['Class']==1]
```

In [26]:
```
fraud['Class']
```

Out[26]:
```
541       1
623       1
4920      1
6108      1
6329      1
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 473, dtype: int64
```

In [27]:
```
legit.Amount.describe()
```

Out[27]:
```
count    275190.000000
mean          0.008682
std           1.012309
min          -0.353229
25%          -0.327682
50%          -0.258275
75%          -0.033782
max         102.362243
Name: Amount, dtype: float64
```

In [29]:
```
credit_data.groupby('Class').mean()
```

Out[29]:

Class	V1	V2	V3	V4	V5	V6	V7	V8
0	-0.029792	-0.008288	0.037131	-0.012054	-0.005596	-0.011768	0.017497	-0.007346
1	-4.498280	3.405965	-6.729599	4.472591	-2.957197	-1.432518	-5.175912	0.953255

2 rows × 29 columns

In [30]:
```
legit_sample = legit.sample(n=492)
```

In [31]:
```
new_data = pd.concat([legit_sample,fraud],axis=0)
```

In [33]:
```
new_data.head()
```

Out[33]:

	V1	V2	V3	V4	V5	V6	V7	V8
136754	1.301798	-1.170032	-0.244466	-1.515675	-0.870002	-0.295976	-0.535385	-0.100750
274538	2.040468	-1.453712	-1.852079	-1.907692	1.162493	3.697874	-1.662043	0.961862
18324	1.501540	-0.797578	-0.054047	-1.312796	-1.080745	-1.066227	-0.509636	-0.313412
271239	2.054002	0.523225	-3.196006	0.618924	1.170933	-1.302178	0.597304	-0.222190
19487	1.220089	0.576802	-0.365253	1.121952	0.208446	-0.692656	0.156710	-0.050598

5 rows × 30 columns

In [35]:
```
new_data['Class'].value_counts()
```

Out[35]:
```
Class
0    492
1    473
Name: count, dtype: int64
```

In [36]:
```
new_data.groupby('Class').mean()
```

Out[36]:

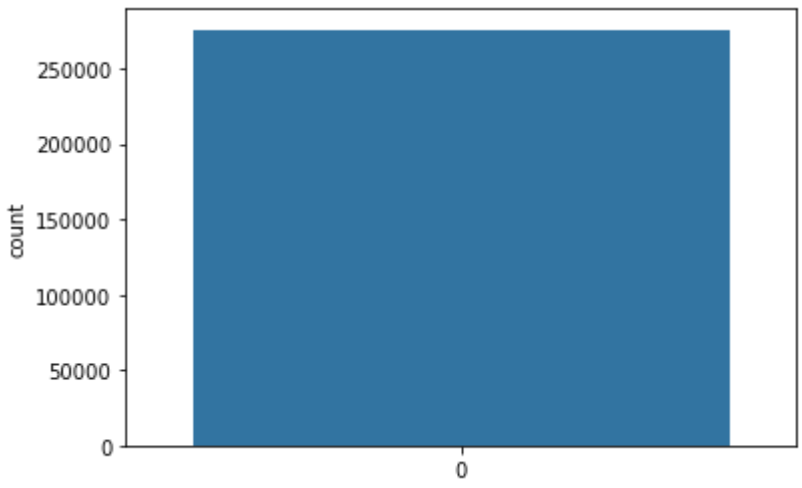
	V1	V2	V3	V4	V5	V6	V7	V8	-
Class									
0	0.144485	-0.005396	0.019267	-0.073610	-0.004070	0.003249	0.108556	0.063216	-
1	-4.498280	3.405965	-6.729599	4.472591	-2.957197	-1.432518	-5.175912	0.953255	-

2 rows × 29 columns

Data Visualization

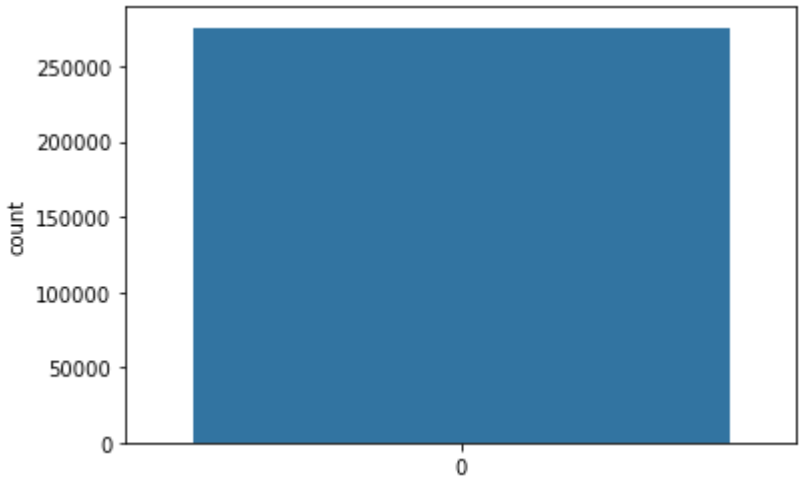
In [20]:
```
sns.countplot(credit_data['Class'])
```

Out[20]:
```
<AxesSubplot:ylabel='count'>
```



In [21]:
```
sns.countplot(credit_data['Amount'])
```

Out[21]: <AxesSubplot:ylabel='count'>



In [22]:
```
credit_data.describe()
```

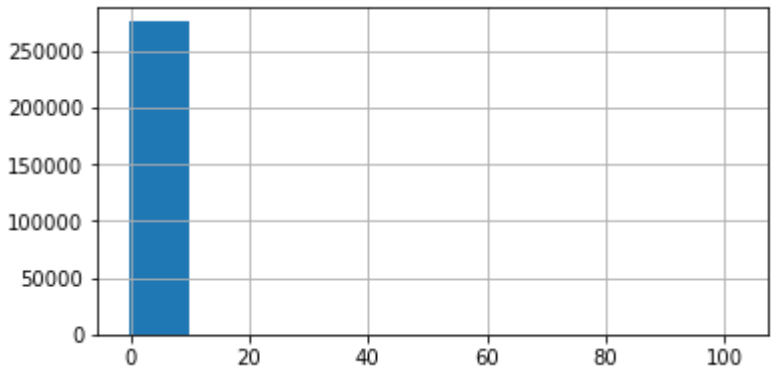
Out[22]:

	V1	V2	V3	V4	V5	
count	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.0
mean	-0.037460	-0.002430	0.025520	-0.004359	-0.010660	-0.0
std	1.952522	1.667260	1.507538	1.424323	1.378117	1.3
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.1
25%	-0.941105	-0.614040	-0.843168	-0.862847	-0.700192	-0.7
50%	-0.059659	0.070249	0.200736	-0.035098	-0.060556	-0.2
75%	1.294471	0.819067	1.048461	0.753943	0.604521	0.3
max	2.454930	22.057729	9.382558	16.875344	34.801666	73.3

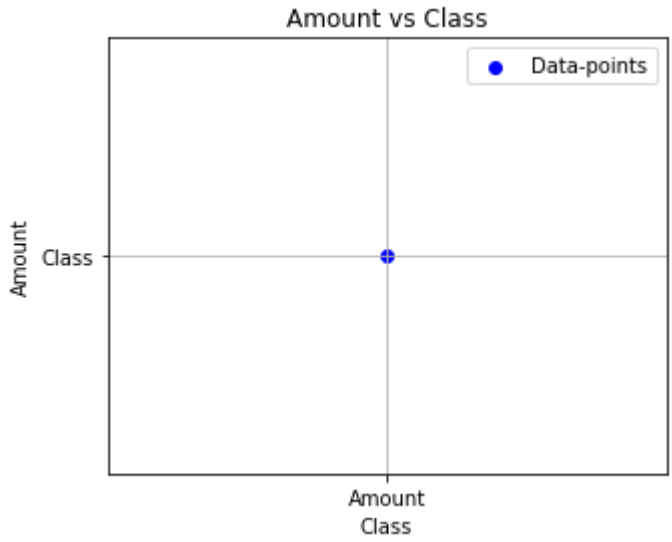
8 rows × 30 columns

In [23]:
```
plt.figure(figsize=(6,3))
#credit_data['Time'].hist()
credit_data['Amount'].hist()
credit_data['Class'].hist()
```

Out[23]: <AxesSubplot:>



In [24]:
```
# scatter plot
plt.figure(figsize=(5,4))
plt.scatter('Amount', 'Class', color='blue', marker='o', label='Data-points
plt.title('Amount vs Class')
plt.xlabel('Class')
plt.ylabel('Amount')
plt.grid(True)
plt.legend()
plt.show()
```



In [37]:
```
# Splitting the dataset into X and Y
x = new_data.drop(columns='Class', axis=1)
y = new_data['Class']
```

Model building

In [38]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, st
```

In [39]:
```
model = LogisticRegression()
```

In [73]:
```
# LogisticRegression
# training the model with logisticRegression with train data
model.fit(x_train, y_train)
#accuracy of trainining data
y_train_prediction = model.predict(x_train)
training_data_prediction = accuracy_score(y_train_prediction, y_train)
print("accuracy of training data: ", training_data_prediction)
```

```
accuracy of training data:  0.9572538860103627
```

In [74]:
```
# now accuracy of testing data
y_test_prediction = model.predict(x_test)
testing_data_prediction = accuracy_score(y_test_prediction, y_test)
print("accuracy of testing data: ", testing_data_prediction)
```

```
accuracy of testing data:  0.9481865284974094
```

In [75]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
```

In [76]:
```
precision_score(y_test, y_test_prediction)
```

Out[76]:
```
0.9885057471264368
```

In [77]:
```
recall_score(y_test, y_test_prediction)
```

Out[77]:
```
0.9052631578947369
```

In [78]:
```
f1_score(y_test, y_test_prediction)
```

Out[78]:
```
0.945054945054945
```

In [79]:
```
# DecisionTreeClassifier
```

In [80]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [81]:
```
model1 = DecisionTreeClassifier()
```

In [82]:
```
# on training data
model1.fit(x_train, y_train)
y_train_prediction1 = model1.predict(x_train)
training_data_prediction1 = accuracy_score(y_train_prediction1, y_train)
print("accuracy of training data: ", training_data_prediction1)
```

```
accuracy of training data:  1.0
```

In [83]:
```
# on testing data
y_test_prediction1 = model1.predict(x_test)
test_data_pred = accuracy_score(y_test_prediction1, y_test)
print("accuracy of testing data: ", test_data_pred)
```

```
accuracy of testing data:  0.9119170984455959
```

In [84]:
```
precision_score(y_test, y_test_prediction1)
```

Out[84]:
```
0.9333333333333333
```

In [85]:
```
recall_score(y_test, y_test_prediction1)
```

Out[85]:
```
0.8842105263157894
```

In [86]:
```
f1_score(y_test, y_test_prediction1)
```

Out[86]:
```
0.9081081081081082
```

In [87]:
```
# RandomForestClassifier
```

In [110]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [111]:
```
model2 = RandomForestClassifier
```

In [112]:
```
# training data

model2 = model.fit(x_train, y_train)
y_train_prediction2 = model1.predict(x_train)
training_data_prediction2 = accuracy_score(y_train_prediction2, y_train)
print("accuracy of training data: ", training_data_prediction2)
```

```
accuracy of training data:  1.0
```

In [113]:
```
# testing data
y_test_prediction2 = model2.predict(x_test)
test_data_pred2 = accuracy_score(y_test_prediction2, y_test)
print("accuracy of testing data: ", test_data_pred2)
```

```
accuracy of testing data:  0.9481865284974094
```

In [114]:
```
precision_score(y_test, y_test_prediction2)
```

Out[114]:
```
0.9885057471264368
```

In [115]:
```
recall_score(y_test, y_test_prediction2)
```

Out[115]:
```
0.9052631578947369
```

In [116]:
```
f1_score(y_test, y_test_prediction2)
```

Out[116]:
```
0.945054945054945
```

In [117]:
```
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "ACC":[accuracy_score(y_test,y_test_prediction)*100,
                     accuracy_score(y_test,y_test_prediction1)*100,
                     accuracy_score(y_test,y_test_prediction2)*100
                    ]})
```

In [118]:
```
final_data
```

Out[118]:

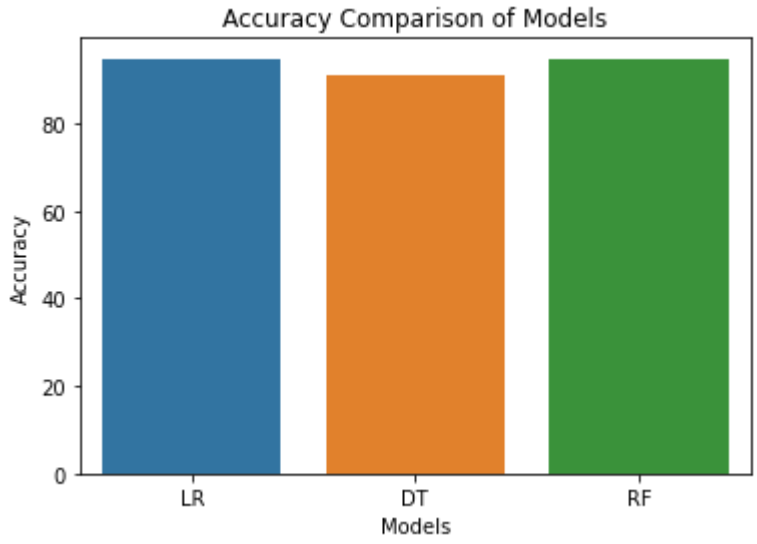
	Models	ACC
0	LR	94.818653
1	DT	91.191710
2	RF	94.818653

In [119]:
```
#  data
final_data =pd.DataFrame({'Models':['LR','DT','RF'],
                "ACC":[accuracy_score(y_test,y_test_prediction)*100,
                       accuracy_score(y_test,y_test_prediction1)*100,
                       accuracy_score(y_test,y_test_prediction2)*100
                      ]})

# Create a bar plot
sns.barplot(x='Models', y='ACC', data=final_data)

# Set labels and title
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Models')

# Display the plot
plt.show()
```



In [122]:
```
# saving the file
import pickle
model_data = pickle.dump(model2, open('credit_data_model.pkl', 'wb') )
```

Evaluating function

In [108]:
```
pred = model.predict([[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
```

In [109]:
```
if pred == 0:
    print("Normal Transcation")
else:
    print("Fraudulent Transcation")
```

```
Normal Transcation
```