```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
LABELS=["Normal",'Fraud']
```

```
data=pd.read_csv(r"C:\Users\DELL\Downloads\creditcard.csv.zip")
data
```

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2395
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0788
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7914
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2376
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5929
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9182
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0243
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2968
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.6861
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5770

284807 rows × 31 columns

```
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
```

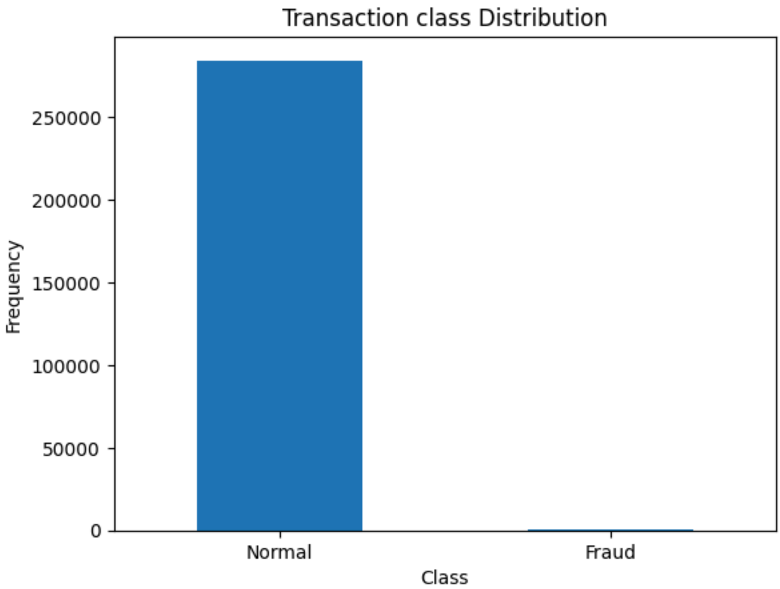
```
data.isnull().sum()
```

```
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
```

```
count_class=pd.value_counts(data['Class'],sort=True)
count_class.plot(kind='bar',rot=0)
plt.title("Transaction class Distribution")
```

```
plt.xticks(range(2),LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_19068\2651216606.py:1: FutureWarning: panda
  count_class=pd.value_counts(data['Class'],sort=True)
Text(0, 0.5, 'Frequency')
```




```
fraud=data[data['Class']==1]
normal=data[data['Class']==0]
print(fraud.shape,normal.shape)
```

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

```
fraud.Amount.describe()
```

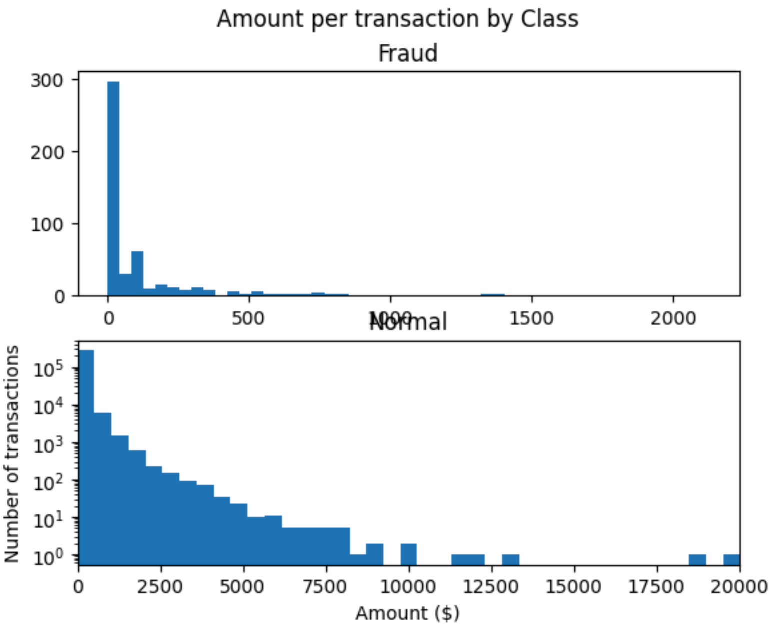
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

```
normal.Amount.describe()
```

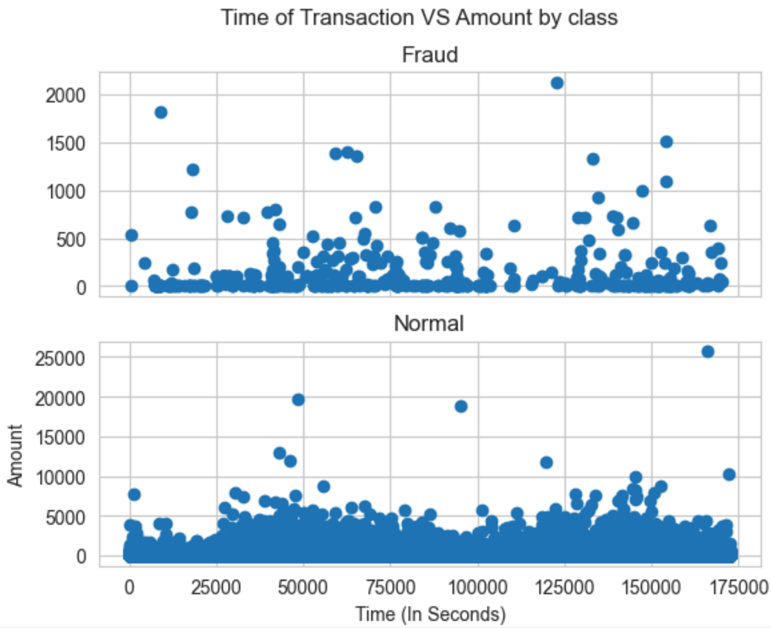
```
count    284315.000000
mean         88.291022
std         250.105092
```

```
min            0.000000
25%            5.650000
50%           22.000000
75%           77.050000
max        25691.160000
Name: Amount, dtype: float64
```

```
f,(ax1,ax2)=plt.subplots(2,1)
f.suptitle("Amount per transaction by Class")
bins=50
ax1.hist(fraud.Amount,bins=bins)
ax1.set_title("Fraud")
ax2.hist(normal.Amount,bins=bins)
ax2.set_title("Normal")
plt.xlabel("Amount ($)")
plt.ylabel("Number of transactions")
plt.xlim(0,20000)
plt.yscale('log')
plt.show()
```




```
f,(ax1,ax2)=plt.subplots(2,1,sharex=True)
f.suptitle("Time of Transaction VS Amount by class")
ax1.scatter(fraud.Time,fraud.Amount)
ax1.set_title("Fraud")
ax2.scatter(normal.Time,normal.Amount)
ax2.set_title("Normal")
plt.xlabel("Time (In Seconds)")
plt.ylabel("Amount")
plt.show()
```



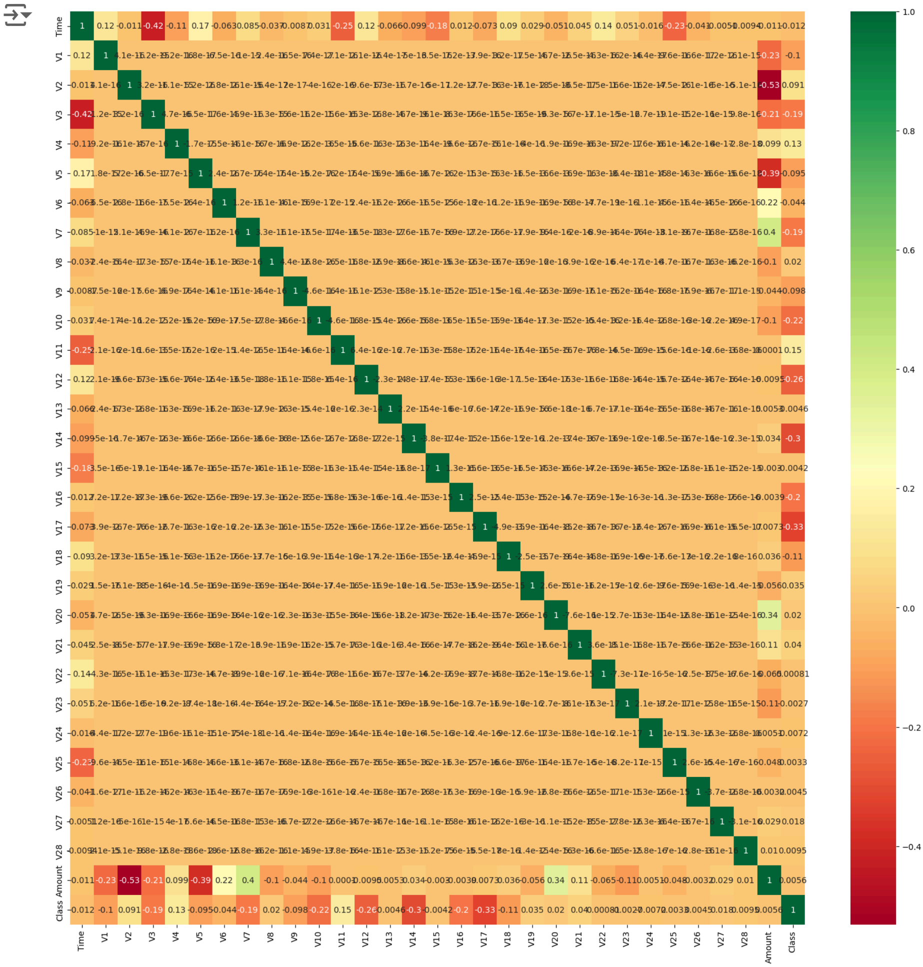

```
data1=data.sample(frac=0.1,random_state=1)
data1.shape
```

```
(28481, 31)
```

```
#To detemine the no.of fraud and valid transactions
fraud_Cases=data1[data1['Class']==1]
valid_Cases=data1[data1['Class']==0]
print("Fraud Cases:",len(fraud_Cases))
print("Valid Cases:",len(valid_Cases))
```

```
Fraud Cases: 49
Valid Cases: 28432
```

```
#Correlation with respective to class variable
correat=data1.corr()
corr_features=correat.index
plt.figure(figsize=(20,20))
s=sns.heatmap(data[corr_features].corr(),annot=True,cmap="RdYlGn")
```



```
legit= data1[data1["Class"] == 0]
fraud =data1[data1["Class"] == 1]

print("legit: ", legit.shape)
print("fraud: ", fraud.shape)
```

```
legit:  (28432, 31)
fraud:  (49, 31)
```

```
print(legit["Amount"].describe())
```

```
count    28432.000000
mean        89.813898
std        270.636594
min          0.000000
25%          5.990000
50%         22.380000
75%         78.820000
max      19656.530000
Name: Amount, dtype: float64
```

```
print(fraud["Amount"].describe())
```

```
count      49.000000
mean      173.505306
std       387.996569
min         0.000000
25%         1.000000
50%         4.900000
75%       122.680000
max      2125.870000
Name: Amount, dtype: float64
```

```
data1.groupby("Class").mean()
```

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	94715.083849	0.005467	-0.023228	0.010909	-0.006879	-0.011407	0.005578	-0.00
1	88874.367347	-3.836270	2.846880	-5.868173	4.194921	-2.487054	-1.124580	-4.00

2 rows × 30 columns

```
legit_sample = legit.sample(n=492)
```

```
new_dataset = pd.concat([legit_sample,fraud],axis=0)
new_dataset.head(5)
```

	Time	V1	V2	V3	V4	V5	V6	V
282557	170985.0	2.126078	-0.712775	-2.333687	-0.959377	0.169801	-1.316296	0.37096
53850	46179.0	0.987681	-1.556111	1.947043	-0.051896	-2.186619	1.163731	-1.82660
121911	76363.0	0.955729	-0.379007	0.817511	1.631598	-0.722025	0.289415	-0.34254
144773	86386.0	-2.096857	1.415074	0.339063	-1.839924	0.478760	1.317507	0.03234
26746	34232.0	-0.152899	-2.013286	-2.362095	0.824978	1.669968	3.479095	0.55413

5 rows × 31 columns

```
new_dataset.tail(5)
```

	Time	V1	V2	V3	V4	V5	V6	
6882	8808.0	-4.617217	1.695694	-3.114372	4.328199	-1.873257	-0.989908	-4.5772
150647	93824.0	-3.632809	5.437263	-9.136521	10.307226	-5.421830	-2.864815	-10.6340
88876	62330.0	1.140865	1.221317	-1.452955	2.067575	0.854742	-0.981223	0.3257
153885	100501.0	-6.985267	5.151094	-4.599338	4.534479	0.849054	-0.210701	-4.4252
245556	152802.0	1.322724	-0.843911	-2.096888	0.759759	-0.196377	-1.166353	0.4825

5 rows × 31 columns

```
new_dataset["Class"].value_counts()
```

```
Class
0    492
1     49
Name: count, dtype: int64
```

```
new_dataset.groupby("Class").mean()
```

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	93025.428862	-0.019906	-0.112196	0.023096	-0.022500	-0.052871	-0.029908	0.00
1	88874.367347	-3.836270	2.846880	-5.868173	4.194921	-2.487054	-1.124580	-4.00

2 rows × 30 columns

```
X = new_dataset.drop(columns = "Class",axis= 1)
Y= new_dataset["Class"]
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, randc
```

```
print(X.shape,X_train.shape,X_test.shape)
```

```
(541, 30) (432, 30) (109, 30)
```

Model Training
-->Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Create a logistic regression model
model = LogisticRegression()
model = LogisticRegression(max_iter=10000)
```

Traning Logistic Regression with training data

```
model.fit(X_train,Y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=10000)
```

Model Evaluation
-->Accuracy on traning data

```
from sklearn.metrics import accuracy_score
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Accuracy on training data", training_data_accuracy)
```

```
Accuracy on training data 0.9930555555555556
```

```
Accuracy On test Data
```

```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Accuracy on test data", test_data_accuracy)
```

```
Accuracy on test data 0.9541284403669725
```

```
We investigated the data, checking for data unbalancing, visualizing the features and unde
We then investigated two predictive models. The data was split in 3 parts, a train set anc
we only used the train and test set.
```

```
We Find the Accuracy Scores on fro both test and train are quite similar.
Accuracy is above 94%.
```

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
# 3. Set Up Random Forest and RandomizedSearchCV
# Random forest classifier
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [100, 200, 500],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Randomized search with cross-validation
random_search = RandomizedSearchCV(estimator=rf,
                                   param_distributions=param_dist,
                                   n_iter=100,  # Number of parameter combinations to try
                                   scoring='roc_auc',
                                   cv=3,  # 3-fold cross-validation
                                   verbose=2,
                                   random_state=42,
                                   n_jobs=-1)

# 4. Fit the model using RandomizedSearchCV
random_search.fit(X_train, Y_train)

best_rf = random_search.best_estimator_
Y_pred = best_rf.predict(X_test)
Y_prob = best_rf.predict_proba(X_test)[:, 1]
print("Classification Report:\n", classification_report(Y_test, Y_pred))

# Print confusion matrix
```

```
print("Confusion Matrix:\n", confusion_matrix(Y_test, Y_pred))
roc_auc = roc_auc_score(Y_test, Y_prob)
print(f"AUC-ROC Score: {roc_auc}")
print("Best Hyperparameters:\n", random_search.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        99
           1       0.88      0.70      0.78        10

    accuracy                           0.96       109
   macro avg       0.92      0.84      0.88       109
weighted avg       0.96      0.96      0.96       109

Confusion Matrix:
 [[98  1]
 [ 3  7]]
AUC-ROC Score: 0.9787878787878788
Best Hyperparameters:
 {'n_estimators': 100, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_features':
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Random forest classifier
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [100, 200, 500],
    'max_features': ['sqrt', 'log2'],  # Removed 'auto'
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Randomized search with cross-validation
random_search = RandomizedSearchCV(estimator=rf,
                                   param_distributions=param_dist,
                                   n_iter=100,  # Number of parameter combinations to try
                                   scoring='roc_auc',
                                   cv=3,  # 3-fold cross-validation
                                   verbose=2,  # <-- This prints detailed fitting informat
                                   random_state=42,
                                   n_jobs=-1)

# Fit the model using RandomizedSearchCV
random_search.fit(X_train, Y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
▸ RandomizedSearchCV
▸ best_estimator_: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
knn = KNeighborsClassifier()
param_dist = {
    'n_neighbors': list(range(1, 31)),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
random_search = RandomizedSearchCV(estimator=knn,
                                   param_distributions=param_dist,
                                   n_iter=100,
                                   scoring='roc_auc',
                                   cv=3,
                                   verbose=2,
                                   random_state=42,
                                   n_jobs=-1)
random_search.fit(X_train, Y_train)
best_knn = random_search.best_estimator_
Y_pred = best_knn.predict(X_test)
Y_prob = best_knn.predict_proba(X_test)[:, 1]
print("Classification Report:\n", classification_report(Y_test, Y_pred))
print("Confusion Matrix:\n", confusion_matrix(Y_test, Y_pred))
roc_auc = roc_auc_score(Y_test, Y_prob)
print(f"AUC-ROC Score: {roc_auc}")
print("Best Hyperparameters:\n", random_search.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Classification Report:
               precision    recall  f1-score   support

           0       0.93      1.00      0.96        99
           1       1.00      0.20      0.33        10

    accuracy                           0.93       109
   macro avg       0.96      0.60      0.65       109
weighted avg       0.93      0.93      0.90       109

Confusion Matrix:
 [[99  0]
 [ 8  2]]
AUC-ROC Score: 0.6646464646464647
```

```
Best Hyperparameters:
 {'weights': 'distance', 'n_neighbors': 17, 'metric': 'manhattan'}
```