```
# Data Collection:
from google.colab import drive
import pandas as pd
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/creditcard.csv'
df = pd.read_csv(file_path)
```

```
Mounted at /content/drive
```

```
df = pd.read_csv(file_path, sep=',')
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
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
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
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
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
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

5 rows × 31 columns

```
import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

```
df.info()
```

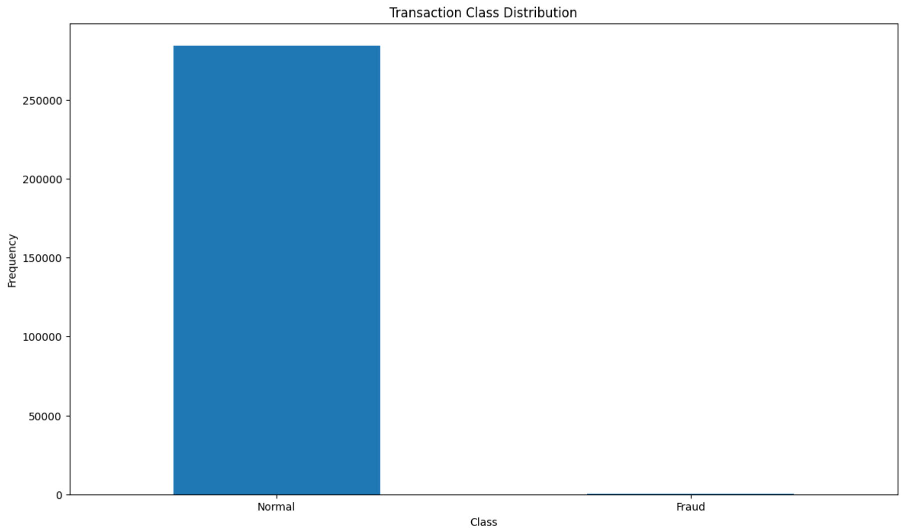
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 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
```

```
# Exploratory Data Analysis:
df.isnull().values.any()
```

```
False
```

```
count_classes = pd.value_counts(df['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

```
Text(0, 0.5, 'Frequency')
```




```
# Get the Fraud and the normal dataset
fraud = df[df['Class']==1]
normal = df[df['Class']==0]
print(fraud.shape,normal.shape)
```

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

```
# Transaction data analysis to get more info
# Different amount of money in different classes of transaction
fraud.Amount.describe()
```

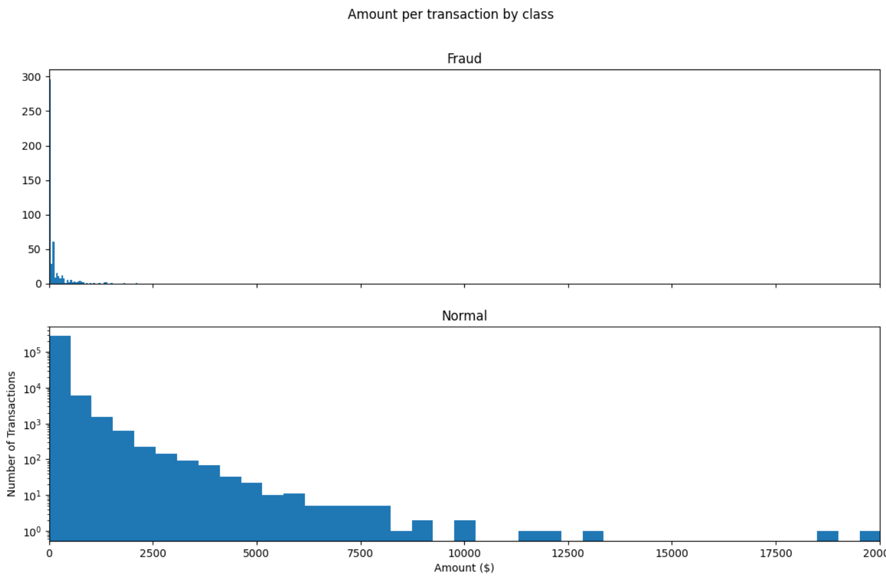
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

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

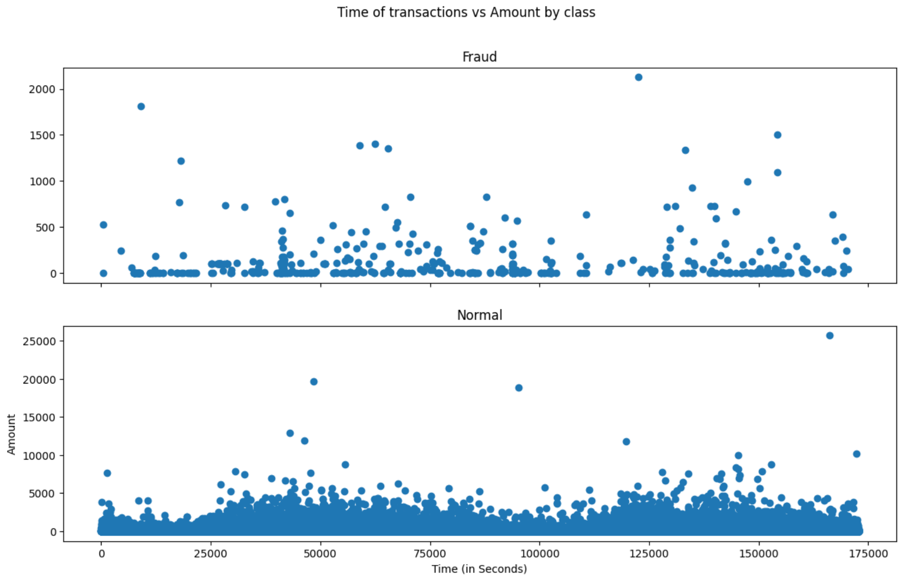
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
```

```
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
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()
```




```
# Fradulent transaction checking : Occurring different times
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transactions 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()
```




```
# Take some data sample
df1= df.sample(frac= 0.1, random_state=1)
df1.shape
```

```
(28481, 31)
```

```
df.shape
```

```
(284807, 31)
```

```
# Determination of fraud and valid number of transactions
Fraud = df1[df1['Class']==1]
Valid = df1[df1['Class']==0]
outlier_fraction = len(Fraud)/float(len(Valid))
```

```
print(outlier_fraction)
```

```
0.0017234102419808666
```

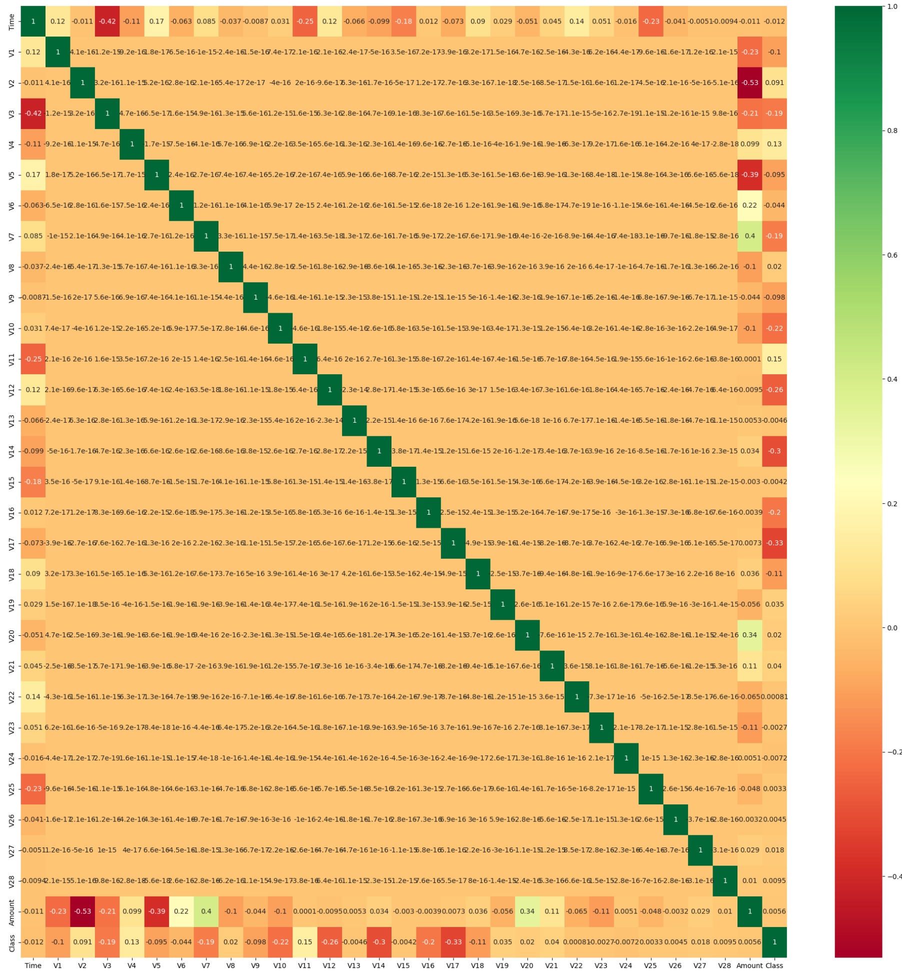
```
print("Fraud Cases : {}".format(len(Fraud)))
print("Valid Cases : {}".format(len(Valid)))
```

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

```
# Pearson Correlation
import seaborn as sns

# Get correlations of each features in dataset
cormat = df1.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(25,25))

# Plot Heat Map
g=sns.heatmap(df[top_corr_features].corr(), annot=True, cmap="RdYlGn")
```



```
# Create Independent and Dependent Features
columns = df1.columns.tolist()

# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]

# Store the variable of prediction
target = 'Class'

# Define a random state
state = np.random.RandomState(42)
X = df1[columns]
Y = df1[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))

# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(28481, 30)
(28481,)
```

```
# Model Prediction:
# Isolation Forest Algorithm : Local Outlier Factor Algorithm
# Define the outlier detection methods
classifiers = {
    "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(X),
                                       contamination=outlier_fraction,random_state=state, verbose=0),
    "Local Outlier Factor":LocalOutlierFactor(n_neighbors=20, algorithm='auto',
                                              leaf_size=30, metric='minkowski',
                                              p=2, metric_params=None, contamination=outlier_fraction),
    "Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.05,
                                         max_iter =+ 1)

}

type(classifiers)
n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    elif clf_name == "Support Vector Machine":
        clf.fit(X)
        y_pred = clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud transactions
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_errors = (y_pred != Y).sum()
    # Run Classification Metrics
    print("{}: {}".format(clf_name,n_errors))
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
    print("Classification Report :")
    print(classification_report(Y,y_pred))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but IsolationForest wa
  warnings.warn(
Isolation Forest: 75
Accuracy Score :
0.9973666654962958
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.24      0.24      0.24        49

    accuracy                           1.00     28481
   macro avg       0.62      0.62      0.62     28481
weighted avg       1.00      1.00      1.00     28481

Local Outlier Factor: 97
Accuracy Score :
0.9965942207085425
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.02      0.02      0.02        49
```

```
    accuracy                           1.00     28481
   macro avg       0.51      0.51      0.51     28481
weighted avg       1.00      1.00      1.00     28481

/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:299: ConvergenceWarning: Solver terminated early (max_iter=1).  Conside
  warnings.warn(
Support Vector Machine: 26995
Accuracy Score :
0.05217513430005969
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.05      0.10     28432
           1       0.00      0.94      0.00        49

    accuracy                           0.05     28481
   macro avg       0.50      0.49      0.05     28481
weighted avg       1.00      0.05      0.10     28481
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# checking the number of missing values in each column
df.isnull().sum()
# distribution of legit transactions & fraudulent transactions
df['Class'].value_counts()
# separating the data for analysis
legit = df[df.Class == 0]
fraud = df[df.Class == 1]
print(legit.shape)
print(fraud.shape)
# statistical measures of the data
legit.Amount.describe()
fraud.Amount.describe()
# compare the values for both transaction
df.groupby('Class').mean()
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	
Class									
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.

2 rows × 30 columns

```
# Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions
legit_sample = legit.sample(n=492)
new_dataset = pd.concat([legit_sample, fraud], axis=0)
new_dataset.head()
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.6972
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.24852
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.21015
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.05873
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.06838

5 rows × 31 columns

```
new_dataset['Class'].value_counts()
new_dataset.groupby('Class').mean()
# Splitting the data into Features & Targets
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
print(X)
print(Y)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
model = LogisticRegression()
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
# accuracy on training data
```

```
# accuracy on training data
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 test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
264811  161601.0  0.017938  0.687194  0.144892 -0.771315  0.518759 -0.570233
200276  133368.0 -1.952688  0.665167  1.716223  0.951378 -0.818139  1.805204
248600  154009.0  0.076369 -4.413245 -2.037587 -0.211303 -1.530600  0.745610
108097   70763.0  1.211666 -0.496228  0.018935 -0.437778 -0.874557 -0.999573
122611   76627.0 -0.304312  0.947578  1.482788  0.786227  0.339348 -0.059198
...          ...       ...       ...       ...       ...       ...       ...
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  \
264811  0.770330  0.078802 -0.037990  ... -0.131485 -0.244406 -0.627677
200276 -0.299761  0.916901  0.684711  ...  0.034366 -0.099803  0.165693
248600  0.420485 -0.133955 -0.240216  ...  2.217154  0.453885 -1.113855
108097 -0.361239 -0.137549 -0.872550  ...  0.163425  0.042623 -0.108184
122611  0.718600  0.040451 -0.337145  ... -0.122365 -0.028883  0.151610
...          ...       ...       ...  ...       ...       ...       ...
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
264811  0.037402 -0.446302 -0.546071  0.151802  0.236342  0.079725     0.89
200276 -0.374975  0.297583  0.608344 -0.391337  0.215877  0.133290   130.00
248600 -0.577173  0.057545 -0.807282 -0.697142 -0.199388  0.135135  1089.24
108097  0.031474  0.308438  0.259870 -0.336059  0.011798  0.047881    79.00
122611 -0.219457  0.101185  0.241547 -0.354059  0.070765  0.021626     1.00
...          ...       ...       ...       ...       ...       ...      ...
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]
264811    0
200276    0
248600    0
108097    0
122611    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
(984, 30) (787, 30) (197, 30)
Accuracy on Training data :  0.9479034307496823
Accuracy score on Test Data :  0.9137055837563451
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```