

Fraud Detection Project

```
In [1]: import numpy as np
import pandas as pd
```

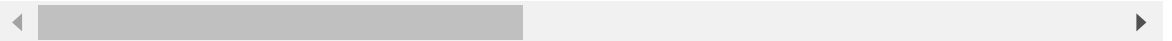
```
In [2]: Fraud_data=pd.read_csv("Fraud_detect.csv")
```

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

```
Out[3]:
```

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

5 rows × 31 columns

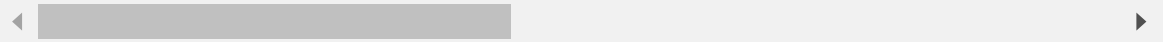


```
In [4]: Fraud_data.tail()
```

```
Out[4]:
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 |
|--------|-------|-----------|----------|-----------|-----------|-----------|-----------|-----------|---------|
| 153753 | 99980 | -0.460128 | 0.203036 | 0.469998 | -0.877349 | -0.333565 | 1.198391 | 1.088143 | -0.0... |
| 153754 | 99984 | 1.930509 | 0.589877 | -0.573178 | 4.011452 | 0.646176 | 0.406016 | 0.003458 | -0.0... |
| 153755 | 99995 | -0.080563 | 0.840885 | -0.085326 | -0.606702 | 0.879790 | -0.493156 | 0.879652 | -0.0... |
| 153756 | 99998 | -1.730665 | 1.302833 | 0.397864 | -0.445631 | -0.773382 | 0.223966 | -0.921886 | 1.0... |
| 153757 | 99999 | -0.309641 | 0.771215 | 0.679051 | -1.069673 | 1.414042 | -0.379638 | 1.315839 | -0.0... |

5 rows × 31 columns



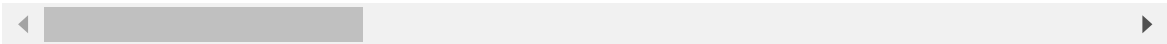
In [5]:

Fraud_data.describe()

Out[5]:

| | Time | V1 | V2 | V3 | V4 | |
|-------|---------------|---------------|---------------|---------------|---------------|----------|
| count | 153758.000000 | 153758.000000 | 153758.000000 | 153758.000000 | 153758.000000 | 153758.0 |
| mean | 55219.166333 | -0.234187 | 0.045504 | 0.604016 | 0.134835 | -0.2 |
| std | 22658.988461 | 1.836892 | 1.620692 | 1.327202 | 1.352793 | 1.3 |
| min | 0.000000 | -56.407510 | -72.715728 | -33.680984 | -5.519697 | -42.1 |
| 25% | 39394.000000 | -1.021436 | -0.539477 | 0.096945 | -0.723555 | -0.8 |
| 50% | 56784.000000 | -0.258642 | 0.119851 | 0.704860 | 0.154723 | -0.2 |
| 75% | 73829.000000 | 1.164805 | 0.805848 | 1.337165 | 0.975383 | 0.2 |
| max | 99999.000000 | 2.401777 | 18.902453 | 9.382558 | 16.875344 | 34.8 |

8 rows × 31 columns



In [6]:

Fraud_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 [8]: class_name={0: 'Not Fraud', 1: 'Fraud'}  
print(Fraud_data.Class.value_counts().rename(index=class_name))
```

```
Not Fraud    153428  
Fraud         330  
Name: Class, dtype: int64
```

```
In [13]: #important function  
#train=2/3  
#stratify = split a data  
  
#importing the library  
from sklearn.model_selection import train_test_split  
  
#output  
y=Fraud_data["Class"]  
  
#input  
X=Fraud_data.loc[:, Fraud_data.columns != 'Class']  
  
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=1/3, random_sta
```

Logistic Regression

```
In [26]: #classification algorithms (fraud or no Fraud)  
  
#import Library for accuracy Score  
from sklearn.metrics import accuracy_score  
  
#import Library for Logistic Regression  
from sklearn.linear_model import LogisticRegression
```

```
In [27]: #initialize the Logistic Regression classifier  
  
logisreg=LogisticRegression()
```

```
In [28]: #Train the model using Training dataset
logisreg.fit(X_train,y_train)
```

```
#Prediction using test data
y_pred=logisreg.predict(X_test)
```

C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html> (<https://scikit-learn.org/stable/modules/preprocessing.html>)

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
In [29]: #calculating Model accuracy by comparing y_test and y_pred
```

```
acc_logisreg=round( accuracy_score(y_test,y_pred)*100,2)
```

```
In [30]: print('Accuracy of Logistic Regression: ',acc_logisreg)
```

Accuracy of Logistic Regression: 99.86

Gaussian Naive Bayes

```
In [25]: #import Library for Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
```

```
#initialize the Gaussian Naive Bayes classifier
model = GaussianNB()
```

```
#Train the model using Training dataset
model.fit(X_train,y_train)
```

```
#Prediction using test data
y_pred =model.predict(X_test)
```

```
#calculating Model accuracy by comparing y_test and y_pred
acc_ganb=round( accuracy_score(y_test,y_pred)*100,2)
```

```
print('Accuracy of Gaussian Naive Bayes: ',acc_ganb)
```

Accuracy of Gaussian Naive Bayes: 98.54

Decision Tree (CART)

```
In [35]: # import Library for Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

#initialize the Decision Tree classifier
model=DecisionTreeClassifier()

#Train the model using Training dataset
model.fit(X_train,y_train)

#Prediction using test data
y_pred =model.predict(X_test)

#calculating Model accuracy by comparing y_test and y_pred
acc_dtree=round( accuracy_score(y_test,y_pred)*100,2)

print('Accuracy of Decision Tree: ',acc_dtree)
```

Accuracy of Decision Tree: 99.9

Random Forest

```
In [40]: # import Library for Random Forest
from sklearn.ensemble import RandomForestClassifier

#initialize the Random Forest classifier
model=RandomForestClassifier()

#Train the model using Training dataset
model.fit(X_train,y_train)

#Prediction using test data
y_pred =model.predict(X_test)

#calculating Model accuracy by comparing y_test and y_pred
acc_rf=round( accuracy_score(y_test,y_pred)*100,2)

print('Accuracy of Random Forest: ',acc_rf)
```

Accuracy of Random Forest: 99.94

K Nearest Neighbour Classifier

```
In [42]: # import Library for K Nearest Neighbour Model
from sklearn.neighbors import KNeighborsClassifier

#initialize the K Nearest Neighbour Model with Default value of K=5
model=KNeighborsClassifier()

#Train the model using Training dataset
model.fit(X_train,y_train)

#Prediction using test data
y_pred =model.predict(X_test)

#calculating Model accuracy by comparing y_test and y_pred
acc_knn=round( accuracy_score(y_test,y_pred)*100,2)

print('Accuracy of KNN Classifier: ',acc_knn)
```

Accuracy of KNN Classifier: 99.8

Model Selection

```
In [43]: models=pd.DataFrame({
    'Model':['Logistic Regression','Naive Bayes','Decision Tree','Random Fo
    'Score':[acc_logisreg,acc_ganb,acc_dtree,acc_rf,acc_knn]
})

models.sort_values(by='Score',ascending=False)
```

Out[43]:

| | Model | Score |
|---|---------------------|-------|
| 3 | Random Forest | 99.94 |
| 2 | Decision Tree | 99.90 |
| 0 | Logistic Regression | 99.86 |
| 4 | K-Nearest Neighbors | 99.80 |
| 1 | Naive Bayes | 98.54 |

In []: