Credit Card Fraud Detection.ipynb

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

#NUMPY- NumPy can be used to perform a wide variety of mathematical operations on arrays

#Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

2.helps us to make data frame

#DATA FRAME- structured table

#train_test_split- allows us to split our data into training data and test data

#Logistic regression model- Logistic regression transforms the continuous output of a linear regression model into a categorical value (0 or 1) using a sigmoid function, which maps any input to a value between 0 and 1.

This is done to model binary classification problems.

#accuracy_score- tells you how often your model's predictions match the actual outcomes

```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('/content/credit_data.csv')
```

-load our dataset to a panda data frame

-v1 to v28 are vertical transactions (credit card details are sensitive details.)

- Dataset provider converted all the features through principal component analysis(PCA) and convert features into numerical value.

-we will use these numerical values for our analysis and prediction.

- CLASS means whether the transaction is LEGIT or FRAUDULENT transaction. (0 and 1 respectively.)

-currency is in DOLLAR and Time is in seconds

```
# dataset informations
credit_card_data.info()
```

-dataset informations

-tell entries , data type and how many values are present

```
credit_card_data.isnull().sum()
```

-checking the number of missing values in each column

```
credit_card_data['Class'].value_counts()
```

- distribution of legit transactions & fraudulent transactions
- 0--> Normal Transaction (284315)
- 1--> fraudulent transaction (492)

-This Dataset is highly unblanced

- We have two classes (two target variables) here , because in this case more than 99 percent data is in a particular class.

-We cannot feed this data to our machine learning model because if we train machine learning model with this data , then it cannot recognize fraudulent transactions because we have very less data points .

-then processing comes into play

```
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

separating the data for analysis

```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
```

(492, 31)

- 31 are columns.
- 492 are fraudulent transactions.
- 284315 are legit transactions.

legit.Amount.describe()

-statistical measures of the data

fraud.Amount.describe()

-mean is bigger in fraudulent.

credit_card_data.groupby('Class').mean()

-compare the values for both transactions

Legit_sample = legit.sample(n=492)

-Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions--> 492

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

-Concatenating two DataFrames

-axis =0 means row wise data

new_dataset.head()

new_dataset.tail()

new_dataset['Class'].value_counts()

new_dataset.groupby('Class').mean()

Splitting the data into Features & Targets

Splitting-> V1 , V2, and so on

Target -> 0 and 1

X = new_dataset.drop(columns='Class', axis=1)

Y = new_dataset['Class']

print(X)

print(Y)

Split the data into Training data & Testing Data

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)

-features of training data store in X_train (80 % data)

-All labels of corresponding data store in Y_train (20%)

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

(984, 30) (787, 30) (197, 30)

Model Training

Logistic Regression

model = LogisticRegression()

```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Model Evaluation

Accuracy Score

```
# 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 Training data :  0.9415501905972046
```

```
# 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)

Accuracy score on Test Data : 0.9390862944162437

NOTE:

Why we have predicted accuracy score of training data ?

- If accuracy score of training data is very different from test data, then it makes our model over fitted or under fitted.

-Let’s say if we get accuracy score of 95% in training data and only 50% in test data ,that means our model is over fitted with training data.

-It means model is overtrain on model data.

-In underfitting , we get very less training data accuracy.