[Skip to main content](https://colab.research.google.com/drive/1k7XtHFOKPOpwYG2FtNJEHFfHObG9H3Fe" \l "top-toolbar)

format\_list\_bulleted

search

vpn\_key

folder

code

terminal

addCodeInsert code cell below Ctrl+M B

addTextAdd text cell

ConnectConnect to a new runtime

arrow\_drop\_down

Gemini

people

settings

expand\_more

[ ]

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

arrow\_upward

arrow\_downward

spark

link

comment

settings

delete

more\_vert

[ ]

# loading the dataset to a Pandas DataFrame

credit\_card\_data = pd.read\_csv('/content/creditcard.csv')



[ ]

# first 5 rows of the dataset  
credit\_card\_data.head()

[ ]

credit\_card\_data.tail()

[ ]

# dataset informations  
credit\_card\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 78981 entries, 0 to 78980

Data columns (total 31 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 id 78981 non-null int64

1 V1 78981 non-null float64

2 V2 78981 non-null float64

3 V3 78981 non-null float64

4 V4 78981 non-null float64

5 V5 78981 non-null float64

6 V6 78981 non-null float64

7 V7 78981 non-null float64

8 V8 78981 non-null float64

9 V9 78981 non-null float64

10 V10 78981 non-null float64

11 V11 78981 non-null float64

12 V12 78981 non-null float64

13 V13 78981 non-null float64

14 V14 78981 non-null float64

15 V15 78981 non-null float64

16 V16 78981 non-null float64

17 V17 78981 non-null float64

18 V18 78981 non-null float64

19 V19 78981 non-null float64

20 V20 78981 non-null float64

21 V21 78981 non-null float64

22 V22 78981 non-null float64

23 V23 78981 non-null float64

24 V24 78981 non-null float64

25 V25 78981 non-null float64

26 V26 78980 non-null float64

27 V27 78980 non-null float64

28 V28 78980 non-null float64

29 Amount 78980 non-null float64

30 Class 78980 non-null float64

dtypes: float64(30), int64(1)

memory usage: 18.7 MB

[ ]

# checking the number of missing values in each column  
credit\_card\_data.isnull().sum()

[ ]

# distribution of legit transactions & fraudulent transactions

credit\_card\_data['Class'].value\_counts()



[ ]

# separating the data for analysis  
legit = credit\_card\_data[credit\_card\_data.Class == 0]  
fraud = credit\_card\_data[credit\_card\_data.Class == 1]

[ ]

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

(78789, 31)

(191, 31)

[ ]

# statistical measures of the data  
legit.Amount.describe()

[ ]

fraud.Amount.describe()

[ ]

# compare the values for both transactions  
credit\_card\_data.groupby('Class').mean()

Under-Sampling

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

Number of Fraudulent Transactions --> 191

[ ]

legit\_sample = legit.sample(n=800)

Concatenating two DataFrames

[ ]

new\_dataset = pd.concat([legit\_sample, fraud], axis=0)

[ ]

new\_dataset.head()

[ ]

new\_dataset.tail()

[ ]

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)

id V1 V2 V3 V4 V5 V6 \

45886 45886 0.601857 -0.690298 0.221114 -0.394787 0.322345 0.373684

74589 74589 0.584018 -0.852803 0.593744 -0.446028 -0.057823 0.418921

56831 56831 0.850714 -0.600245 1.129412 -0.119487 -0.040914 0.886598

77052 77052 -0.489535 0.236507 1.017009 0.080690 -0.278495 1.090081

16974 16974 0.193219 -0.545877 1.348309 -1.871407 -0.059119 0.122141

... ... ... ... ... ... ... ...

76929 76929 0.210765 0.263490 -0.541320 0.542008 0.034838 -0.318936

77099 77099 0.167844 0.103679 -0.255876 0.716690 -0.253954 -0.008106

77348 77348 -0.237414 0.279947 -0.310552 0.218637 -0.185348 -0.097516

77387 77387 -0.483189 0.216475 -0.405255 0.174345 -0.704477 0.653538

77682 77682 -1.481535 0.805291 -1.195057 1.054111 -1.174815 0.398891

V7 V8 V9 ... V20 V21 V22 \

45886 0.752291 -0.211711 0.318804 ... 0.432665 -0.187685 -1.131514

74589 0.423870 -0.120040 0.820562 ... 0.146213 -0.114038 -0.550726

56831 0.162138 -0.018494 1.231459 ... -0.456717 -0.112413 -0.122180

77052 -0.284830 -0.978705 0.227563 ... 0.848544 -1.183825 0.808605

16974 0.363823 -0.181808 -0.613808 ... -0.413741 -0.169511 -0.171675

... ... ... ... ... ... ... ...

76929 -0.294943 0.024809 -0.300752 ... 0.425540 0.009743 -0.717896

77099 -0.371112 0.090700 -0.642143 ... 0.178024 0.203761 0.210568

77348 -0.310118 0.127571 -0.571199 ... 0.396345 0.150547 0.032732

77387 -0.035613 0.199012 -0.455951 ... 0.113783 0.200453 0.236341

77682 -1.567181 -0.795795 -1.449913 ... 0.747471 -1.181048 1.658586

V23 V24 V25 V26 V27 V28 Amount

45886 -0.242001 -1.231819 0.308575 0.420842 -0.352610 0.032503 8197.92

74589 -0.147681 0.579081 -0.029114 1.786757 -0.369482 -0.020357 5230.26

56831 -0.038516 -0.651079 0.298605 -0.850267 -0.171923 -0.051331 12072.37

77052 0.128405 0.725304 0.260482 -0.144209 -0.702445 -0.040867 11795.69

16974 -0.154626 -0.246734 -0.099270 -0.500461 -0.151148 0.034419 8504.60

... ... ... ... ... ... ... ...

76929 -0.287690 -1.441470 1.515505 -0.508884 0.921081 1.032662 12602.28

77099 -0.167793 0.062873 0.981178 0.485519 0.772654 0.833450 23399.41

77348 -0.244640 -0.886682 0.284833 1.549896 0.571365 0.503470 4341.97

77387 0.294860 -1.008899 -0.280198 -1.213340 0.668832 0.399398 7152.63

77682 1.020410 -0.919196 -0.545193 -0.237177 0.164347 0.294841 8136.33

[382 rows x 30 columns]

[ ]

print(Y)

45886 0.0

74589 0.0

56831 0.0

77052 0.0

16974 0.0

...

76929 1.0

77099 1.0

77348 1.0

77387 1.0

77682 1.0

Name: Class, Length: 382, dtype: float64

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)

[ ]

print(X.shape, X\_train.shape, X\_test.shape)

(382, 30) (305, 30) (77, 30)

Model Training

Logistic Regression

[ ]

model = LogisticRegression()

[ ]

from sklearn.linear\_model import LogisticRegression  
  
# Initialize the LogisticRegression model with specific parameters  
model = LogisticRegression(max\_iter=1000)  
  
# Get and print all parameters of the model  
print(model.get\_params())

{'C': 1.0, 'class\_weight': None, 'dual': False, 'fit\_intercept': True, 'intercept\_scaling': 1, 'l1\_ratio': None, 'max\_iter': 1000, 'multi\_class': 'deprecated', '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 : 0.9704918032786886

[ ]

# 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 : 0.922077922077922

[ ]

from sklearn.ensemble import RandomForestClassifier  # Import the RandomForestClassifier  
  
# Initialize the RandomForestClassifier model with specific parameters  
model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Get and print all parameters of the model  
print(model.get\_params())

{'bootstrap': True, 'ccp\_alpha': 0.0, 'class\_weight': None, 'criterion': 'gini', 'max\_depth': None, 'max\_features': 'sqrt', 'max\_leaf\_nodes': None, 'max\_samples': None, 'min\_impurity\_decrease': 0.0, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'min\_weight\_fraction\_leaf': 0.0, 'monotonic\_cst': None, 'n\_estimators': 100, 'n\_jobs': None, 'oob\_score': False, 'random\_state': 42, 'verbose': 0, 'warm\_start': False}

[ ]

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
  
# Initialize the RandomForestClassifier model with specific parameters  
model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Fit the model on the training data  
model.fit(X\_train, Y\_train)  
  
# Make predictions on the training data  
X\_train\_prediction = model.predict(X\_train)  
  
# Calculate the accuracy on the training data  
training\_data\_accuracy = accuracy\_score(Y\_train, X\_train\_prediction)  
  
# Print the accuracy on the training data  
print("Accuracy on training data: ", training\_data\_accuracy)

Accuracy on training data: 1.0

[ ]

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
  
# Initialize and fit the RandomForestClassifier  
model\_rf = RandomForestClassifier(n\_estimators=100, random\_state=42)  
model\_rf.fit(X\_train, Y\_train)  # Fit the model on the training data  
  
# Make predictions on the testing data  
X\_test\_prediction\_rf = model\_rf.predict(X\_test)  
  
# Calculate accuracy on the testing data  
test\_data\_accuracy\_rf = accuracy\_score(Y\_test, X\_test\_prediction\_rf)  
  
# Print accuracy on the testing data  
print("Accuracy on testing data for Random Forest: ", test\_data\_accuracy\_rf)

Accuracy on testing data for Random Forest: 0.922077922077922

[ ]

import xgboost as xgb  
from sklearn.metrics import accuracy\_score  
  
# Initialize the XGBClassifier model with specific parameters  
model = xgb.XGBClassifier(n\_estimators=100, random\_state=42)  
  
# Fit the model on the training data  
model.fit(X\_train, Y\_train)  
  
# Make predictions on the training data  
X\_train\_prediction = model.predict(X\_train)  
  
# Calculate the accuracy on the training data  
training\_data\_accuracy = accuracy\_score(Y\_train, X\_train\_prediction)  
  
# Print the accuracy on the training data  
print("Accuracy on training data: ", training\_data\_accuracy)

Accuracy on training data: 1.0

[ ]

import xgboost as xgb  
from sklearn.metrics import accuracy\_score  
  
# Initialize and fit the XGBoost classifier  
model\_xgb = xgb.XGBClassifier(n\_estimators=100, random\_state=42)  
model\_xgb.fit(X\_train, Y\_train)  # Fit the model on the training data  
  
# Make predictions on the testing data  
X\_test\_prediction\_xgb = model\_xgb.predict(X\_test)  
  
# Calculate accuracy on the testing data  
test\_data\_accuracy\_xgb = accuracy\_score(Y\_test, X\_test\_prediction\_xgb)  
  
# Print accuracy on the testing data  
print("Accuracy on testing data for XGBoost: ", test\_data\_accuracy\_xgb)

Accuracy on testing data for XGBoost: 0.935064935064935

[ ]

from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy\_score  
  
# Initialize the DecisionTreeClassifier model  
model\_dt = DecisionTreeClassifier(random\_state=42)  
  
# Train the model on the training data  
model\_dt.fit(X\_train, Y\_train)  
  
# Make predictions on the training data  
X\_train\_prediction\_dt = model\_dt.predict(X\_train)  
  
# Calculate accuracy on the training data  
training\_data\_accuracy\_dt = accuracy\_score(Y\_train, X\_train\_prediction\_dt)  
  
# Print accuracy on the training data  
print("Accuracy on training data for Decision Tree: ", training\_data\_accuracy\_dt)

Accuracy on training data for Decision Tree: 1.0

[ ]

from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy\_score  
  
# Initialize and fit the DecisionTreeClassifier  
model\_dt = DecisionTreeClassifier(random\_state=42)  
model\_dt.fit(X\_train, Y\_train)  # Fit the model on the training data  
  
# Make predictions on the testing data  
X\_test\_prediction\_dt = model\_dt.predict(X\_test)  
  
# Calculate accuracy on the testing data  
test\_data\_accuracy\_dt = accuracy\_score(Y\_test, X\_test\_prediction\_dt)  
  
# Print accuracy on the testing data  
print("Accuracy on testing data for Decision Tree: ", test\_data\_accuracy\_dt)

Accuracy on testing data for Decision Tree: 0.8961038961038961

[ ]

# Importing necessary libraries  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.tree import DecisionTreeClassifier  
from xgboost import XGBClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from sklearn.datasets import make\_classification  
  
# Create an imbalanced dataset (For demonstration purposes, use sklearn's make\_classification)  
# In real-world cases, you can load your own dataset using pandas, e.g., pd.read\_csv('your\_file.csv')  
X, Y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, weights=[0.9, 0.1], flip\_y=0, random\_state=42)  
  
# Splitting the data into training and testing sets  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=42)  
  
# --------------------------------------------------------  
# Random Forest Classifier with Class Weights  
rf\_model = RandomForestClassifier(class\_weight='balanced', random\_state=42)  
rf\_model.fit(X\_train, Y\_train)  
Y\_pred\_rf = rf\_model.predict(X\_test)  
print("Random Forest - Classification Report:")  
print(classification\_report(Y\_test, Y\_pred\_rf))  
  
# --------------------------------------------------------  
# Decision Tree Classifier with Class Weights  
dt\_model = DecisionTreeClassifier(class\_weight='balanced', random\_state=42)  
dt\_model.fit(X\_train, Y\_train)  
Y\_pred\_dt = dt\_model.predict(X\_test)  
print("Decision Tree - Classification Report:")  
print(classification\_report(Y\_test, Y\_pred\_dt))  
  
# --------------------------------------------------------  
# XGBoost Classifier with scale\_pos\_weight (for imbalance)  
# Calculate the scale\_pos\_weight: ratio of negative class to positive class  
# For a dataset with class imbalance of 90%:10% (negative:positive), scale\_pos\_weight can be calculated  
# This is a basic calculation, in practice, you can compute the exact ratio from your dataset  
scale\_pos\_weight = (sum(Y == 0) / sum(Y == 1))  
  
xgb\_model = XGBClassifier(scale\_pos\_weight=scale\_pos\_weight, random\_state=42, use\_label\_encoder=False, eval\_metric='mlogloss')  
xgb\_model.fit(X\_train, Y\_train)  
Y\_pred\_xgb = xgb\_model.predict(X\_test)  
print("XGBoost - Classification Report:")  
print(classification\_report(Y\_test, Y\_pred\_xgb))

Random Forest - Classification Report:

precision recall f1-score support

0 0.97 0.98 0.97 275

1 0.76 0.64 0.70 25

accuracy 0.95 300

macro avg 0.86 0.81 0.84 300

weighted avg 0.95 0.95 0.95 300

Decision Tree - Classification Report:

precision recall f1-score support

0 0.95 0.95 0.95 275

1 0.44 0.44 0.44 25

accuracy 0.91 300

macro avg 0.69 0.69 0.69 300

weighted avg 0.91 0.91 0.91 300

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [07:23:40] WARNING: /workspace/src/learner.cc:740:

Parameters: { "use\_label\_encoder" } are not used.

warnings.warn(smsg, UserWarning)

XGBoost - Classification Report:

precision recall f1-score support

0 0.98 0.98 0.98 275

1 0.79 0.76 0.78 25

accuracy 0.96 300

macro avg 0.88 0.87 0.88 300

weighted avg 0.96 0.96 0.96 300

[ ]

# Import necessary libraries  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report, accuracy\_score  
from imblearn.over\_sampling import SMOTE  
from sklearn.datasets import make\_classification  
  
# Generate a toy imbalanced dataset  
X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, weights=[0.9, 0.1], flip\_y=0, random\_state=42)  
  
  
# Split the dataset into train and test  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
# Apply SMOTE for oversampling the minority class  
smote = SMOTE(sampling\_strategy='auto', random\_state=42)  
X\_train\_res, y\_train\_res = smote.fit\_resample(X\_train, y\_train)  
  
# Initialize the Logistic Regression model  
model = LogisticRegression(max\_iter=1000, random\_state=42)  
  
# Train the model on the resampled data  
model.fit(X\_train\_res, y\_train\_res)  
  
# Make predictions  
y\_pred = model.predict(X\_test)  
  
# Evaluate the model  
print("Accuracy:", accuracy\_score(y\_test, y\_pred))  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred, target\_names=["Class 0", "Class 1"]))

Accuracy: 0.87

Classification Report:

precision recall f1-score support

Class 0 0.97 0.88 0.93 275

Class 1 0.36 0.72 0.48 25

accuracy 0.87 300

macro avg 0.67 0.80 0.70 300

weighted avg 0.92 0.87 0.89 300

[Colab paid products](https://colab.research.google.com/signup?utm_source=footer&utm_medium=link&utm_campaign=footer_links)- [Cancel contracts here](https://colab.research.google.com/cancel-subscription)

fiber\_manual\_record

close