CREDIT CARD FRAUD DETECTION

PHASE 5

IBM NAAN MUDHALVAN

ABSTRACT

The purpose of this project is to detect the fraudulent transactions made by credit cards by the use of machine learning techniques, to stop fraudsters from the unauthorized usage of customers' accounts. The increase of credit card fraud is growing rapidly worldwide, which is the reason actions should be taken to stop fraudsters. Putting a limit for those actions would have a positive impact on the customers as their money would be recovered and retrieved back into their accounts and they won't be charged for items or services that were not purchased by them which is the main goal of the project. Detection of the fraudulent transactions will be made by using three machine learning techniques KNN, SVM and Logistic Regression, those models will be used on a credit card transaction dataset.

HOW CREDIT CARD FRAUD DETECTION WORKS?

Credit card fraud happens when a fraudster gets hold of someone else's credit card details and makes a purchase with it. This is clear fraud, where the goal is to not pay for a good or service and still receive it.
Note that there is also another type of credit card fraud that happens when the cardholder is being dishonest. In that scenario, the payment looks legitimate, but the cardholder has already decided to return the item or ask for a refund. The latter is called friendly fraud, and it can be challenging to detect. The cardholder may say that

the card has been stolen whereas, in fact, they were the one who made the purchase but claim otherwise. In both scenarios, however, the key ingredients are the same. For credit card fraud to work you
need:

- A credit card number (legitimate or stolen).
- A CNP purchase (card not present), for instance at an online store.
- A request for a refund. This will be made by the victim whose stolen card was used in
the fraudulent purchase or the legitimate cardholder.

Ideally, however, you will flag credit card fraud before the purchase goes through, ensuring you do
not have to deal with the last step.

System requirement

Software

1. Windows Xp, Windows 7(ultimate , enterprise)
2. Python 3.6 and related libraries

Hardware

1. Processor - i3
2. Memory - 1GB RAM
3. Hard Disk - 5 GB

About Dataset

Kaggle Dataset: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Data preprocessing:

Data preprocessing involves transforming raw data to well-formed data sets so that data mining analytics can be applied. Raw data is often incomplete and has inconsistent formatting. The adequacy or inadequacy of data preparation has a direct correlation with the success of any project that involve data analytics.Preprocessing involves both data validation and data imputation.

CSV FILE READING:

SOURCE CODE:

```
import pandas as pd
pandas.set_option('display.max_rows',None)
pandas.set_option('display.max_rows',None)
my_data = pd.read_csv("/content/creditcardfraud.csv")
print(my_data)
print(my_data.describe())
print(my_data.info())
```

OUTPUT:

```
Time V1 V2 V3 V4 V5 V6 \ 0 0 -1.359807 -0.072781 2.536347 1.378155
-0.338321 0.462388 1 0 1.191857 0.266151 0.166480 0.448154 0.060018
-0.082361 2 1 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 3 1
-0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 4 2 -1.158233
0.877737 1.548718 0.403034 -0.407193 0.095921 5 2 -0.425966 0.960523
1.141109 -0.168252 0.420987 -0.029728 6 4 1.229658 0.141004 0.045371
1.202613 0.191881 0.272708 7 7 -0.644269 1.417964 1.074380 -0.492199
0.948934 0.428118 8 7 -0.894286 0.286157 -0.113192 -0.271526 2.669599
3.721818 9 9 -0.338262 1.119593 1.044367 -0.222187 0.499361 -0.246761 10
10 1.449044 -1.176339 0.913860 -1.375667 -1.971383 -0.629152 11 10
0.384978 0.616109 -0.874300 -0.094019 2.924584 3.317027
4994 -0.520257 0.338650 -0.499198 -0.084009 -0.161852 19.99 0 4995
0.049254 0.588488 -0.297483 0.045084 0.017438 15.00 0 4996 0.327924
-0.561712 0.306355 0.165721 0.148775 15.18 0 4997 -0.362803 -0.456087
-0.470498 -0.612240 -0.511449 56.15 0 4998 0.517534 -0.160048 0.230429
0.135888 0.101896 4.00 0 [4999 rows x 31 columns] Time V1 V2 V3 V4 \ count
4999.000000 4999.000000 4999.000000 4999.000000 4999.000000 mean
2115.221844 -0.231035 0.255234 0.819478 0.018035 std 1310.102385 1.373201
1.161586 1.005885 1.421765 min 0.000000 -12.168192 -15.732974 -12.389545
-4.657545 25% 965.500000 -0.979049 -0.326925 0.276994 -0.916689 50%
2072.000000 -0.405257 0.325089 0.847091 0.064782 75% 3239.500000 1.126184
0.902100 1.455616 0.990501 max 4562.000000 1.685314 6.224859 4.101716
6.013346 V5 V6 V7 V8 V9 ... \ count 4999.000000 4999.000000 4999.000000
4999.000000 4999.000000 ... mean -0.001339 0.176419 0.042220 -0.033852
0.235050 ... std 1.193685 1.365788 1.056738 1.198979 0.981329 ... min
-32.092129 -7.465603 -11.164794 -23.632502 -3.336805 ... 25% -0.603296
-0.695901 -0.473188 -0.191965 -0.360733 ... 50% -0.082993 -0.166386
0.065064 0.038049 0.232645 ... 75% 0.433876 0.581120 0.584885 0.336568
0.756629 ... max 10.658654 21.393069 34.303177 3.877662 9.272376 ... V21
V22 V23 V24 V25 \ count 4999.000000 4999.000000 4999.000000 4999.000000
4999.000000 mean -0.022887 -0.150042 -0.041246 0.038171 0.096815 std
0.795826 0.631919 0.369371 0.619768 0.402316 min -11.273890 -5.707801
-7.996811 -2.512377 -2.322906 25% -0.242947 -0.587596 -0.190642 -0.341881
-0.138207 50% -0.092730 -0.173006 -0.048391 0.103253 0.116469 75% 0.074382
0.283000 0.085963 0.445400 0.357233 max 15.631453 4.393846 4.095021
3.200201 1.972515 V26 V27 V28 Amount Class count 4999.000000 4999.000000
4999.000000 4999.000000 4999.000000 mean -0.048632 0.038369 0.004919
63.813629 0.000600 std 0.492946 0.334767 0.244351 197.487279 0.024492 min
-1.338556 -5.336289 -2.909294 0.000000 0.000000 25% -0.416792 -0.042303
-0.016478 3.800000 0.000000 50% -0.089835 0.021396 0.019662 15.160000
0.000000 75% 0.248893 0.174102 0.082159 57.170000 0.000000 max 3.463246
3.852046 4.157934 7712.430000 1.000000 [8 rows x 31 columns] <class
'pandas.core.frame.DataFrame'> RangeIndex: 4999 entries, 0 to 4998 Data
columns (total 31 columns): # Column Non-Null Count Dtype --- ------
```

```
------------- ----- 0 Time 4999 non-null int64 1 V1 4999 non-null float64
2 V2 4999 non-null float64 3 V3 4999 non-null float64 4 V4 4999 non-null
float64 5 V5 4999 non-null float64 6 V6 4999 non-null float64 7 V7 4999
non-null float64 8 V8 4999 non-null float64 9 V9 4999 non-null float64 10
V10 4999 non-null float64 11 V11 4999 non-null float64 12 V12 4999
non-null float64 13 V13 4999 non-null float64 14 V14 4999 non-null float64
15 V15 4999 non-null float64 16 V16 4999 non-null float64 17 V17 4999
non-null float64 18 V18 4999 non-null float64 19 V19 4999 non-null float64
20 V20 4999 non-null float64 21 V21 4999 non-null float64 22 V22 4999
non-null float64 23 V23 4999 non-null float64 24 V24 4999 non-null float64
25 V25 4999 non-null float64 26 V26 4999 non-null float64 27 V27 4999
non-null float64 28 V28 4999 non-null float64 29 Amount 4999 non-null
float64 30 Class 4999 non-null int64 dtypes: float64(29), int64(2) memory
usage: 1.2 MB None
```

TEST AND TRAIN

SOURCE CODE:

```
 # Run this program on your local python
 # interpreter, provided you have installed
 # the required libraries.
 # Importing the required packages
 import numpy as np
 import pandas as pd
 from sklearn.metrics import confusion_matrix from sklearn.model_selection import
 train_test_split from sklearn.tree import DecisionTreeClassifier from sklearn.metric
 import accuracy_score from sklearn.metrics import classification_report
# Function importing Dataset
 def importdata():
  balance_data = pd.read_csv( 'https://archive.ics.uci.edu/ml/machine-learning-' +
'databases/balance-scale/balance-scale.data',  sep=',', header=None)
 # Printing the dataswet shape
  print("Dataset Length: ", len(balance_data))  print("Dataset Shape: ",
balance_data.shape)
  # Printing the dataset obseravtions
  print("Dataset: ", balance_data.head())
  return balance_data
 # Function to split the dataset
 def splitdataset(balance_data):
  # Separating the target variable
  X = balance_data.values[:, 1:5]
  Y = balance_data.values[:, 0]
  # Splitting 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=100)

  return X, Y, X_train, X_test, y_train, y_test
```

```
# Function to perform training with giniIndex.

def train_using_gini(X_train, X_test, y_train):

 # Creating the classifier object

 clf_gini = DecisionTreeClassifier(criterion="gini",

 random_state=100, max_depth=3, min_samples_leaf=5)

 # Performing training

 clf_gini.fit(X_train, y_train)

 return clf_gini

# Function to perform training with entropy.

def tarin_using_entropy(X_train, X_test, y_train):

 # Decision tree with entropy

 clf_entropy = DecisionTreeClassifier(

 criterion="entropy", random_state=100,

 max_depth=3, min_samples_leaf=5)
 # Performing training


clf_entropy.fit(X
_train, y_train)
return
clf_entropy

# Function to
make
predictions

def
prediction(X_test,
clf_object):  #
Predicton on test
with giniIndex
y_pred =
clf_object.predict(X
_test)
print("Predicted
values:")
 print(y_pred)

 return y_pred

# Function to calculate accuracy

def
cal_accuracy(y_
test, y_pred):
print("Confusio
n Matrix: ",
 confusion_matrix(y_test, y_pred))
 print("Accuracy : ",

 accuracy_score(y_test, y_pred) * 100)

print("Report : ",

 classification_report(y_test, y_pred))

# Driver code

def main():

 # Building Phase
 data = importdata()

 X, Y, X_train, X_test, y_train, y_test =
splitdataset(data)  clf_gini =
train_using_gini(X_train, X_test, y_train)
clf_entropy = tarin_using_entropy(X_train,
X_test, y_train)


 # Operational Phase

 print("Results Using Gini Index:")


 # Prediction using gini

 y_pred_gini = prediction(X_test, clf_gini)

 cal_accuracy(y_test, y_pred_gini)
```

```
print("Results Using Entropy:")

# Prediction using entropy

y_pred_entropy =
prediction(X_test, clf_entropy)
cal_accuracy(y_test,
y_pred_entropy)



# Calling main function

if __name__ == "__main__":

main()
```

OUTPUT:

```
Dataset Length: 4999 Dataset Shape: (4999, 31) Dataset: Time
V1 V2 V3 V4 V5 V6 V7 \ 0 0 -1.359807 -0.072781 2.536347
1.378155 -0.338321 0.462388 0.239599 1 0 1.191857 0.266151
0.166480 0.448154 0.060018 -0.082361 -0.078803 2 1 -1.358354
-1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461 3 1
-0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203
0.237609 4 2 -1.158233 0.877737 1.548718 0.403034 -0.407193
0.095921 0.592941 V8 V9 ... V21 V22 V23 V24 V25 \ 0 0.098698
0.363787 ... -0.018307 0.277838 -0.110474 0.066928 0.128539 1
0.085102 -0.255425 ... -0.225775 -0.638672 0.101288 -0.339846
0.167170 2 0.247676 -1.514654 ... 0.247998 0.771679 0.909412
-0.689281 -0.327642 3 0.377436 -1.387024 ... -0.108300
0.005274 -0.190321 -1.175575 0.647376 4 -0.270533 0.817739 ...
-0.009431 0.798278 -0.137458 0.141267 -0.206010 V26 V27 V28
Amount Class 0 -0.189115 0.133558 -0.021053 149.62 0 1
0.125895 -0.008983 0.014724 2.69 0 2 -0.139097 -0.055353
-0.059752 378.66 0 3 -0.221929 0.062723 0.061458 123.50 0 4
0.502292 0.219422 0.215153 69.99 0 [5 rows x 31 columns]
Results Using Gini Index: Predicted values: [2270. 2270. 2270.
... 2270. 3770. 2270.] Confusion Matrix: [[0 0 0 ... 0 0 0] [0
0 0 ... 0 0 0] [0 0 0 ... 0 0 0] ... [0 0 0 ... 0 0 0] [0 0 0
... 0 0 0] [0 0 0 ... 0 0 0]] Accuracy : 0.4666666666666673
Report : precision recall f1-score support 0.0 0.00 0.00 0.00
1 2.0 0.00 0.00 0.00 2 7.0 0.00 0.00 0.00 1 10.0 0.00 0.00
0.00 1 12.0 0.00 0.00 0.00 3 15.0 0.00 0.00 0.00 1 22.0 0.00
0.00 0.00 1 23.0 0.00 0.00 0.00 1 24.0 0.00 0.00 0.00 1 27.0
0.00 0.00 0.00 1 33.0 0.00 0.00 0.00 2

 accuracy                           0.01      1500

   macro avg       0.00       0.00       0.00       1500

weighted avg       0.00       0.01       0.00       1500
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_class
ification.py:1344: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control
this behavior.

  _warn_prf(average, modifier, msg_start, len(result))

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_class
ification.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this
behavior.

  _warn_prf(average, modifier, msg_start, len(result))
```

SOURCE CODE:

```
x=my_data.iloc[:,:-1].values

y=my_data.iloc[:,3].values

from sklearn.datasets import load_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import GridSearchCV



# Load the dataset

iris = load_iris()

X = iris.data

y = iris.target



# Define the hyperparameter search space

param_grid = {

    'n_estimators': [10, 50, 100, 200, 500],

    'max_depth': [None, 10, 20, 30, 40, 50],

    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2', None]
}


# Create a Random Forest classifier
rf = RandomForestClassifier()


# Create a GridSearchCV object
grid_search = GridSearchCV(
    rf, param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=-1
)


# Perform grid search
grid_search.fit(X, y)


# Print the best hyperparameters and corresponding accuracy
print("Best Hyperparameters: ", grid_search.best_params_)
print("Best Accuracy: {:.2f}%".format(grid_search.best_score_ * 100))
```

OUTPUT:

```
Best Hyperparameters:  {'max_depth': 30, 'max_features': 'auto',
'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 10}

Best Accuracy: 97.33%

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will
be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'` or remove this parameter as it is also the default
value for RandomForestClassifiers and ExtraTreesClassifiers.

  warn(
```

MATRIX MULTIPLICATION:

```
# Import necessary libraries
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Generate some sample data
np.random.seed(0)
data = {
    'Exam1': np.random.rand(100) * 100,
    'Exam2': np.random.rand(100) * 100,
    'Admitted': np.random.randint(2, size=100)
}
df = pd.DataFrame(data)
print(df)
#
# # Split the data into features (X) and target (y)
X = df[['Exam1', 'Exam2']]
y = df['Admitted']
print(X)
print(y)
#
# # Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
#
# # Create a logistic regression model
model = LogisticRegression()
#
# # Fit the model to the training data
model.fit(X_train, y_train)
#
# # Make predictions on the test data
y_pred = model.predict(X_test)
print("-------------------")
print(y_pred)
#
# # Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
#
# # Display classification report and confusion matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#
# Plot the decision boundary
# x_min, x_max = X['Exam1'].min() - 10, X['Exam1'].max() + 10
# y_min, y_max = X['Exam2'].min() - 10, X['Exam2'].max() + 10
# xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,
y_max, 100))
# Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
# Z = Z.reshape(xx.shape)
#
# plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.8)
# plt.scatter(X['Exam1'], X['Exam2'], c=y, cmap=plt.cm.RdBu)
# plt.xlabel('Exam 1 Score')
# plt.ylabel('Exam 2 Score')
# plt.title('Logistic Regression Decision Boundary')
# plt.show()
#Output:


#scikit-learn
from sklearn.datasets import make_classification

value1, y = make_classification(
    n_features=6,
    n_classes=2,
    n_samples=800,
    n_informative=2,
    random_state=66,
    n_clusters_per_class=1,
)

##. This code imports the make_classification function from the
sklearn.datasets module.
##• The make_classification function generates a random dataset for
classification tasks.
```

```
##• The function takes several arguments: n_features: the number of
features (or independent variables) in the dataset.
##• In this case, there are 6 features.
##• n_classes: the number of classes (or target variables) in the dataset.
##• In this case, there are 3 classes.
##• n_samples: the number of samples (or observations) in the dataset.
##• In this case, there are 800 samples.
##• n_informative: the number of informative features in the dataset.
##• These are the features that actually influence the target variable.
##• In this case, there are 2 informative features.
##• random_state: a seed value for the random number generator.
##• This ensures that the dataset is reproducible.
##• n_clusters_per_class: the number of clusters per class.
##• This determines the degree of separation between the classes.
##• In this case, there is only 1 cluster per class.
##• The function returns two arrays: X: an array of shape (n_samples,
n_features) containing the features of the dataset.
##• y: an array of shape (n_samples,) containing the target variable of
the dataset.

import matplotlib.pyplot as plt

plt.scatter(value1[:, 0], value1[:, 1], c=y, marker="*")
plt.show()

##This code imports the matplotlib.pyplot module and creates a scatter
plot using the scatter() function.
##• The X and y variables are assumed to be previously defined arrays or
data frames.
##• The scatter() function takes three arguments: X[:, 0] and X[:, 1] are
the first and second columns of the X array, respectively, and c=y assigns
a color to each point based on the corresponding value in the y array.
##• The marker argument specifies the shape of the marker used for each
point, in this case, an asterisk.
##• The resulting plot will have the values in the first column of X on
the x-axis, the values in the second column of X on the y-axis, and each
point will be colored based on the corresponding value in y.


from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    value1, y, test_size=0.33, random_state=125
)

##This code imports the train_test_split function from the
sklearn.model_selection module.
##• This function is used to split the dataset into training and testing
sets.
##• The train_test_split function takes four arguments: X, y, test_size,
and random_state.
##• X and y are the input features and target variable, respectively.
##• test_size is the proportion of the dataset that should be allocated to
the testing set.
##• In this case, it is set to 0.33, which means that 33% of the data will
be used for testing.
##• random_state is used to set the seed for the random number generator,
which ensures that the same random split is generated each time the code
is run.
##• The function returns four variables: X_train, X_test, y_train, and
y_test.
##• X_train and y_train are the training set, while X_test and y_test are
the testing set.
##• These variables can be used to train and evaluate a machine learning
model.

from sklearn.naive_bayes import GaussianNB

# Build a Gaussian Classifier
model = GaussianNB()

# Model training
model.fit(X_train, y_train)

# Predict Output
predicted = model.predict([X_test[6]])
#
print("Actual Value:", y_test[6])
print("Predicted Value:", predicted[0])
```

```
##This code uses the scikit-learn library to build a Gaussian Naive Bayes
classifier.
##• First, the code imports the GaussianNB class from the
sklearn.naive_bayes module.
##• Next, a new instance of the GaussianNB class is created and assigned
to the variable 'model'.
##• The model is then trained using the fit() method, which takes in the
training data X_train and the corresponding target values y_train.
##• After the model is trained, it is used to predict the output for a
single test data point, which is the 7th element in the X_test array.
##• The predicted value is stored in the 'predicted' variable.
##• Finally, the actual value for the test data point is printed using
y_test[6], and the predicted value is printed using predicted[0].

#----------------

from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
)

y_pred = model.predict(X_test)
accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")

print("Accuracy:", accuray)
print("F1 Score:", f1)

##This code imports several functions from the sklearn.metrics module,
including accuracy_score, confusion_matrix, ConfusionMatrixDisplay, and
f1_score.
##• These functions are used to evaluate the performance of a machine
learning model.
##• The code then uses the model.predict method to generate predictions
for the test data (X_test).
##• These predictions are compared to the actual labels (y_test) using the
accuracy_score and f1_score functions.
```

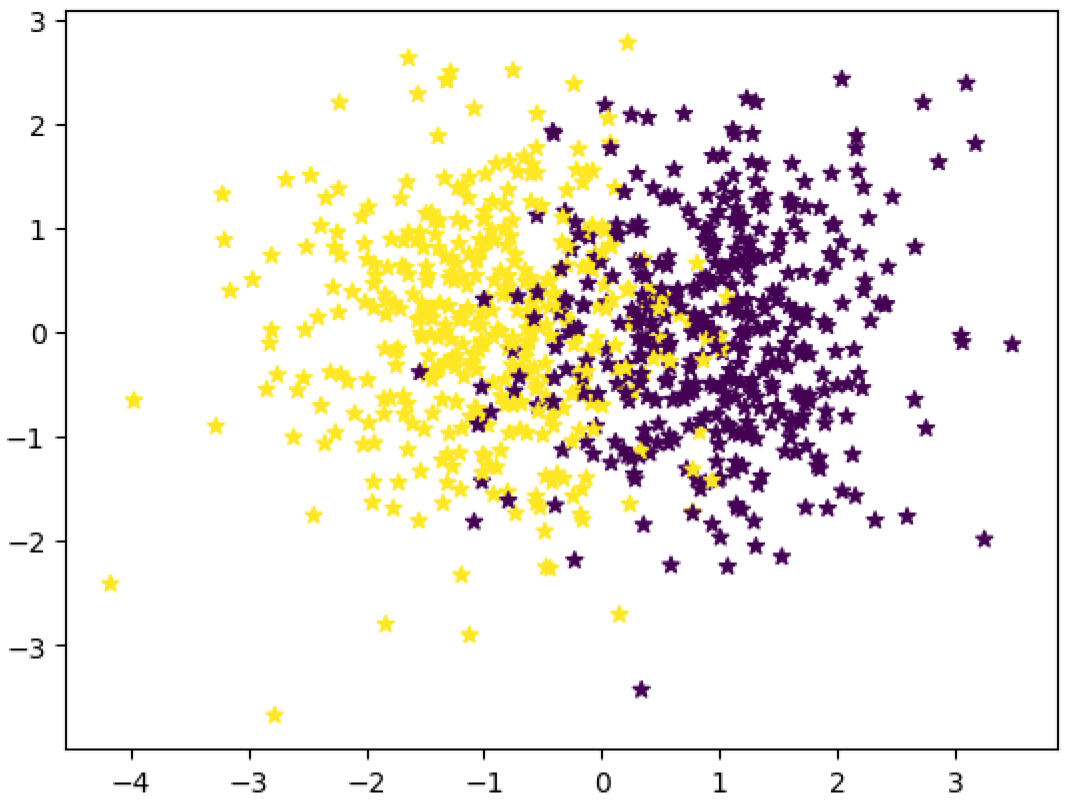
```
##• The accuracy_score function calculates the accuracy of the model's
predictions,
## while the f1_score function calculates the F1 score, which is a
weighted average of precision and recall.
##• Finally, the code prints out the accuracy and F1 score of the model's
predictions.

#------------------------------

#####Expected output
####
####Accuracy: 0.8484848484848485
####F1 Score: 0.8491119695890328

####This code snippet is not actually a code, but rather the output of
some code that was run.
####• It shows the accuracy and F1 score of a model that was trained on
some data.
####• The accuracy is 0.8484848484848485,
## which means that the model correctly predicted the outcome of 84.8% of
the cases.
####• The F1 score is 0.8491119695890328,
#### which is a measure of the model's accuracy that takes into account
both precision and recall.
####• A higher F1 score indicates better performance of the model.

#--------------------------------------------------

labels = [0,1,2]
cm = confusion_matrix(y_test, y_pred, labels=labels)
print(cm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot()

########This code is using the scikit-learn library to create a confusion
matrix and display it using ConfusionMatrixDisplay.
########• First, a list of labels is created with the values 0, 1, and 2.
########• Then, the confusion_matrix function is called with the test
labels (y_test) and predicted labels (y_pred) as inputs, along with the
labels list.
```

```
########• This creates a confusion matrix with the specified labels.
########• Next, a ConfusionMatrixDisplay object is created with the
confusion matrix as input, along with the labels list.
########• Finally, the plot method is called on the display object to show
the confusion matrix graphically.
```

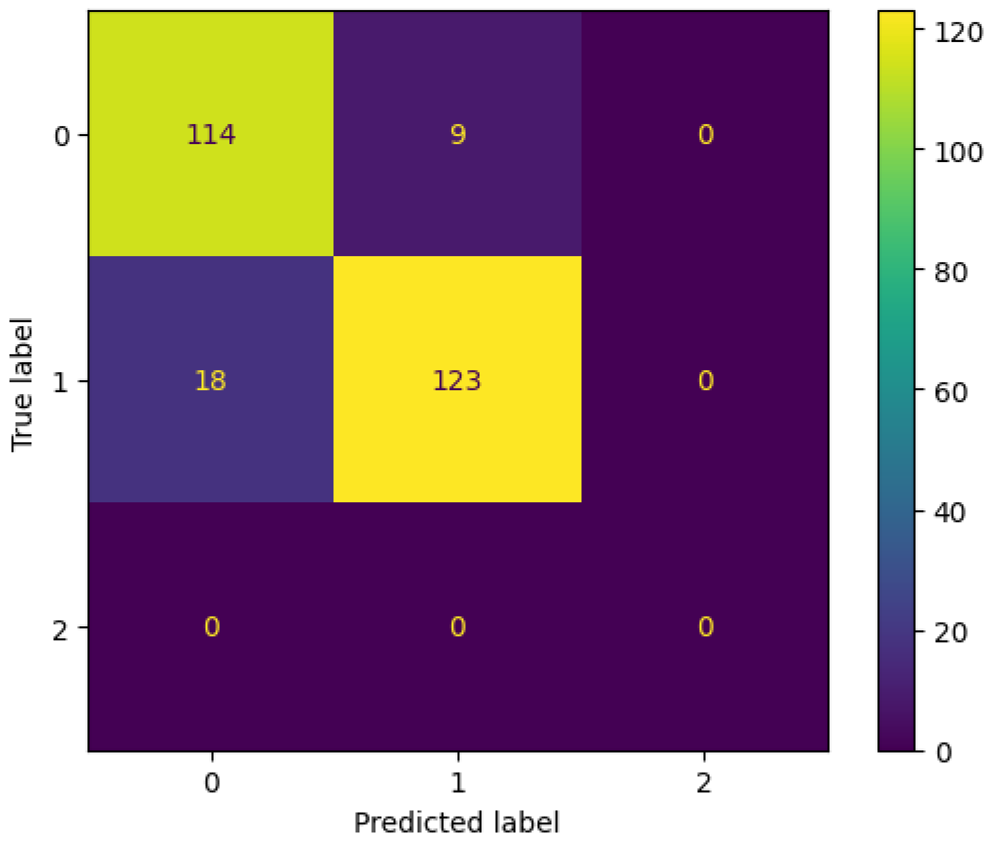
OUTPUT:

```
Exam1      Exam2  Admitted
0   54.881350  67.781654       0
1   71.518937  27.000797       1
2   60.276338  73.519402       0
3   54.488318  96.218855       0
4   42.365480  24.875314       1
..        ...        ...      ...
95  18.319136  49.045881        0
96  58.651293  22.741463        1
97   2.010755  25.435648       0
98  82.894003   5.802916       1
99   0.469548  43.441663       0

[100 rows x 3 columns]
       Exam1      Exam2
0   54.881350  67.781654
1   71.518937  27.000797
2   60.276338  73.519402
3   54.488318  96.218855
4   42.365480  24.875314
..        ...        ...
95  18.319136  49.045881
96  58.651293  22.741463
97   2.010755  25.435648
98  82.894003   5.802916
99   0.469548  43.441663

[100 rows x 2 columns]
0     0
1     1
2     0
3     0
4     1
     ..
95    0
96    1
97    0
98    1
99    0
Name: Admitted, Length: 100, dtype: int64
-----------------
[1 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 1]
```

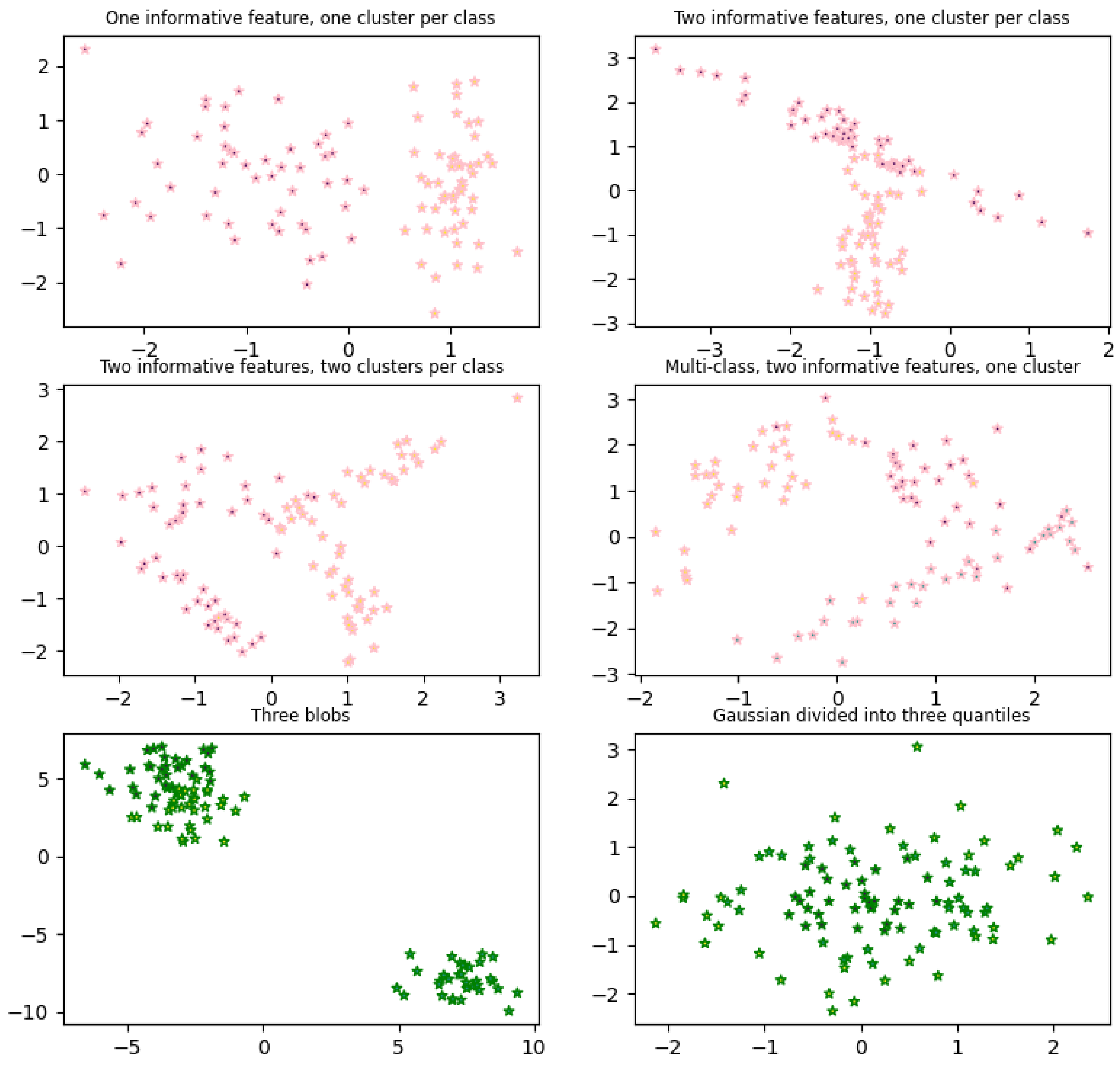
```
Accuracy: 0.45
              precision    recall  f1-score   support

           0       0.27      0.50      0.35         6
           1       0.67      0.43      0.52        14

    accuracy                           0.45        20
   macro avg       0.47      0.46      0.44        20
weighted avg       0.55      0.45      0.47        20

[[3 3]
 [8 6]]
```



```
Actual Value: 1
Predicted Value: 1
Accuracy: 0.8977272727272727
F1 Score: 0.8976082740788625
[[114   9   0]
 [ 18 123   0]
 [  0   0   0]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a1787d7eaa0>
```



Feature engineering:
Feature engineering is the pre-processing step of machine learning, which extracts features from raw data. It helps to represent an underlying problem to predictive models
in a better way, which as a result, improve the accuracy of the model for unseen data. The
predictive model contains predictor variables and an outcome variable, and while the feature engineering process selects the most useful predictor variables for the model.
SOURCE CODE:
```
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs, make_classification,
make_gaussian_quantiles

plt.figure(figsize=(8, 8))
plt.subplots_adjust(bottom=0.05, top=0.9, left=0.05, right=0.95)
plt.subplot(321)
plt.title("One informative feature, one cluster per class",
fontsize="small")
X1, Y1 = make_classification(
    n_features=2, n_redundant=0, n_informative=1, n_clusters_per_class=1
)
plt.scatter(X1[:, 0], X1[:, 1], marker="*", c=Y1, s=25, edgecolor="pink")

plt.subplot(322)
plt.title("Two informative features, one cluster per class",
fontsize="small")
X1, Y1 = make_classification(
    n_features=2, n_redundant=0, n_informative=2, n_clusters_per_class=1
)
plt.scatter(X1[:, 0], X1[:, 1], marker="*", c=Y1, s=25, edgecolor="pink")

plt.subplot(323)
plt.title("Two informative features, two clusters per class",
fontsize="small")
X2, Y2 = make_classification(n_features=2, n_redundant=0, n_informative=2)
plt.scatter(X2[:, 0], X2[:, 1], marker="*", c=Y2, s=25, edgecolor="pink")

plt.subplot(324)
plt.title("Multi-class, two informative features, one cluster",
fontsize="small")
X1, Y1 = make_classification(
    n_features=2, n_redundant=0, n_informative=2, n_clusters_per_class=1,
n_classes=3
)
plt.scatter(X1[:, 0], X1[:, 1], marker="*", c=Y1, s=25, edgecolor="pink")

plt.subplot(325)
plt.title("Three blobs", fontsize="small")
X1, Y1 = make_blobs(n_features=2, centers=3)
plt.scatter(X1[:, 0], X1[:, 1], marker="*", c=Y1, s=25, edgecolor="green")

plt.subplot(326)
plt.title("Gaussian divided into three quantiles", fontsize="small")
X1, Y1 = make_gaussian_quantiles(n_features=2, n_classes=3)
plt.scatter(X1[:, 0], X1[:, 1], marker="*", c=Y1, s=25, edgecolor="green")
plt.show()
```



1)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style("whitegrid")
```

```
data = pd.read_csv("/content/creditcardfraud.csv")
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0	-1.36	-0.07	2.54	1.38	-0.34	0.46	0.24	0.10	0.36	...	-0.02	0.28	-0.11	0.07	0.13	-0.19	0.13	-0.02	149.62	0
1	0	1.19	0.27	0.17	0.45	0.06	-0.08	-0.08	0.09	-0.26	...	-0.23	-0.64	0.10	-0.34	0.17	0.13	-0.01	0.01	2.69	0
2	1	-1.36	-1.34	1.77	0.38	-0.50	1.80	0.79	0.25	-1.51	...	0.25	0.77	0.91	-0.69	-0.33	-0.14	-0.06	-0.06	378.66	0
3	1	-0.97	-0.19	1.79	-0.86	-0.01	1.25	0.24	0.38	-1.39	...	-0.11	0.01	-0.19	-1.18	0.65	-0.22	0.06	0.06	123.50	0
4	2	-1.16	0.88	1.55	0.40	-0.41	0.10	0.59	-0.27	0.82	...	-0.01	0.80	-0.14	0.14	-0.21	0.50	0.22	0.22	69.99	0

5 rows × 31 columns

2)
```
pd.set_option("display.float", "{:.2f}".format)

data.describe()
```

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class

count	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	...	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00	4999.00
mean	2115.22	-0.23	0.26	0.82	0.02	-0.00	0.18	0.04	-0.03	0.24	...	-0.02	-0.15	-0.04	0.04	0.10	-0.05	0.04	0.00	63.81	0.00
std	1310.10	1.37	1.16	1.01	1.42	1.19	1.37	1.06	1.20	0.98	...	0.80	0.63	0.37	0.62	0.40	0.49	0.33	0.24	197.49	0.02
min	0.00	-12.17	-15.73	-12.39	-4.66	-32.09	-7.47	-11.16	-23.63	-3.34	...	-11.27	-5.71	-8.00	-2.51	-2.32	-1.34	-5.34	-2.91	0.00	0.00
25%	965.50	-0.98	-0.33	0.28	-0.92	-0.60	-0.70	-0.47	-0.19	-0.36	...	-0.24	-0.59	-0.19	-0.34	-0.14	-0.42	-0.04	-0.02	3.80	0.00
50%	2072.00	-0.41	0.33	0.85	0.06	-0.08	-0.17	0.07	0.04	0.23	...	-0.09	-0.17	-0.05	0.10	0.12	-0.09	0.02	0.02	15.16	0.00
75%	3239.50	1.13	0.90	1.46	0.99	0.43	0.58	0.58	0.34	0.76	...	0.07	0.28	0.09	0.45	0.36	0.25	0.17	0.08	57.17	0.00

max	4562.000	1.69	6.22	4.10	6.01	10.66	21.39	34.30	3.88	9.27	...	15.63	4.39	4.10	3.20	1.97	3.46	3.85	4.16	7712.43	1.00

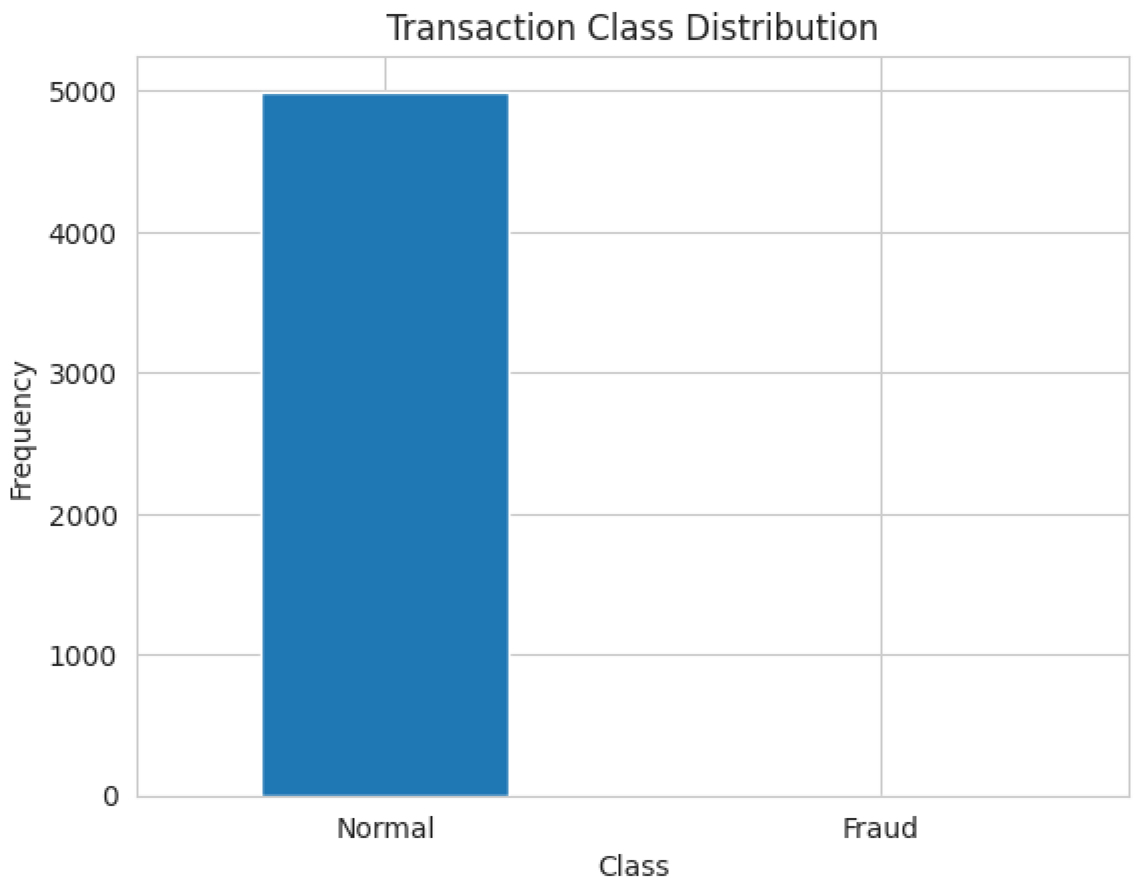
8 rows × 31 columns

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

0

3)

```
LABELS = ["Normal", "Fraud"]

count_classes = pd.value_counts(data['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");
```



4)

```
data.Class.value_counts()
```

```
0 4996 1 3 Name: Class, dtype: int64
```

MODEL SELECTION

Selecting the right model for credit card fraud detection is crucial for building an effective and
efficient system. Credit card fraud detection is typically treated as a binary classification problem
where the goal is to classify transactions as either "fraudulent" or "legitimate." Here are some steps to
help you choose an appropriate model:
Fraud detection is a critical application of machine learning, and the choice of algorithms can
significantly impact the effectiveness of your system. Below are some machine learning algorithms
commonly used in fraud detection:

Logistic Regression:

● Logistic regression is a simple and interpretable algorithm that works well for binary classification problems. It can serve as a baseline model for fraud detection.

Decision Trees:

● Decision trees are used for both classification and regression tasks. They are interpretable and can capture non-linear relationships in the data. However, they are prone to overfitting.

Random Forest:

● Random forests are an ensemble method that consists of multiple decision trees. They offer improved generalization and can handle high-dimensional data. Random forests are less prone to overfitting compared to individual decision trees.

Gradient Boosting Algorithms:

● Algorithms like XGBoost, LightGBM, and CatBoost are widely used for fraud detection. They build an ensemble of decision trees sequentially, with each tree correcting the errors of the previous ones. They are known for their high predictive accuracy.

Support Vector Machines (SVM):

● SVMs work well in high-dimensional spaces and can handle both linear and non-linear classification. They are effective for separating classes, even when the data is not linearly separable.

Neural Networks:

● Deep learning models, such as feedforward neural networks or convolutional neural networks (CNNs), can capture complex patterns in data. They are highly flexible but require large amounts of data and computational resources.

K-Nearest Neighbors (K-NN):

● K-NN is a simple instance-based algorithm that classifies data points based on the majority class of their k-nearest neighbors. It can be effective for fraud detection, especially when the decision boundary is complex.

Isolation Forest:

● Isolation Forest is an anomaly detection algorithm that is suitable for identifying rare and unusual events, making it useful for fraud detection tasks.

One-Class SVM:

● One-Class SVM is another anomaly detection algorithm that can be used when you have a limited amount of labeled fraud data. It aims to separate the target class (fraud) from the rest of the data.

Ensemble Methods:

● Combining multiple models using ensemble techniques like bagging and boosting can enhance fraud detection performance. For example, you can combine decision trees, SVMs, or neural networks into an ensemble.

Autoencoders:

● Autoencoders are neural network architectures used for unsupervised learning and anomaly detection. They can be trained to reconstruct normal data and identify anomalies by detecting reconstruction errors.

Naive Bayes:

● Naive Bayes classifiers are simple probabilistic models that can be used for fraud detection, especially when dealing with categorical data or text-based features.

The choice of algorithm depends on various factors, including the nature of your data, the size of your
dataset, the level of class imbalance, computational resources, and the desired level of interpretability. Often, a combination of multiple algorithms or an ensemble approach can yield the
best results in fraud detection systems. Additionally, feature engineering, data preprocessing, and
model evaluation are also crucial components of building an effective fraud detection system.

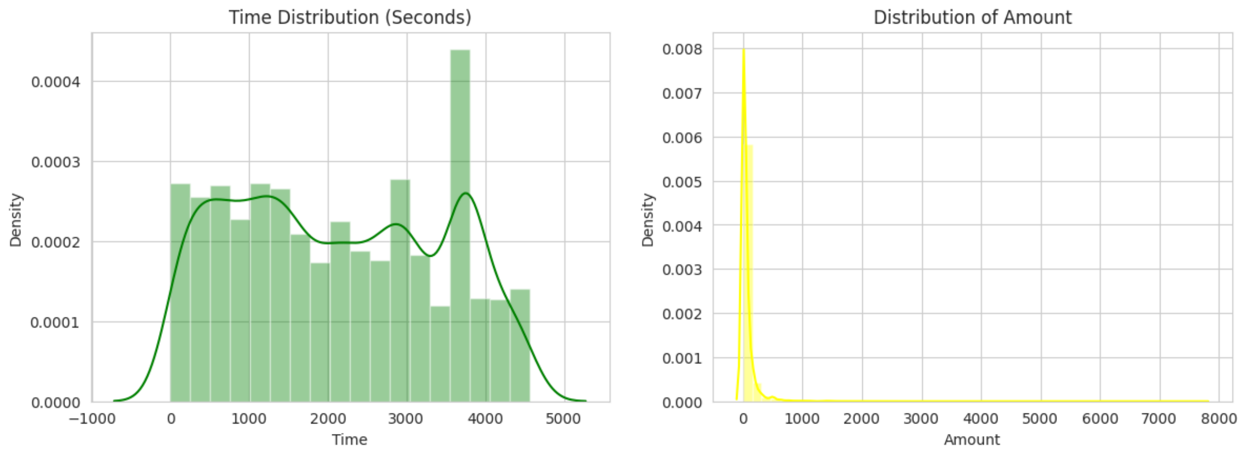
SOURCE CODE:

1)

```
# plot the time feature
plt.figure(figsize=(14,10))

plt.subplot(2, 2, 1)
plt.title('Time Distribution (Seconds)')
sns.distplot(data['Time'], color='green');

# plot the amount feature
plt.subplot(2, 2, 2)
plt.title('Distribution of Amount')
sns.distplot(data['Amount'],color='yellow');
```

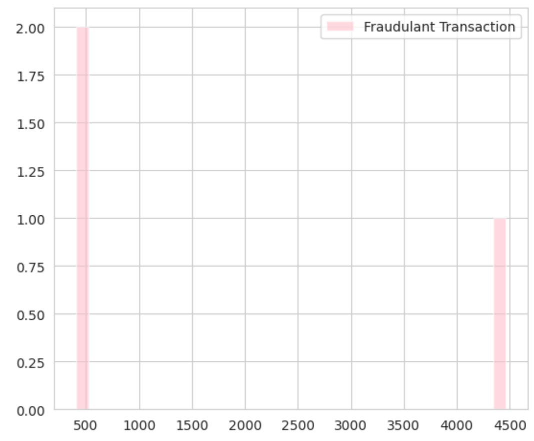
```
<ipython-input-13-4413f71fb891>:6: UserWarning: `distplot` is a deprecated
function and will be removed in seaborn v0.14.0. Please adapt your code to
use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms). For a guide to
updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
sns.distplot(data['Time'], color='green');
```

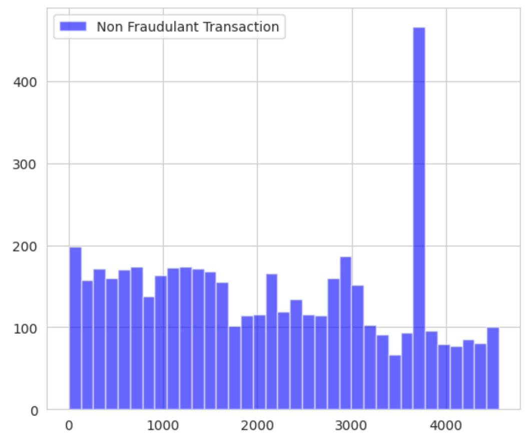
```
<ipython-input-13-4413f71fb891>:11: UserWarning: `distplot` is a
deprecated function and will be removed in seaborn v0.14.0. Please adapt
your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms). For a
guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
sns.distplot(data['Amount'],color='yellow');
```



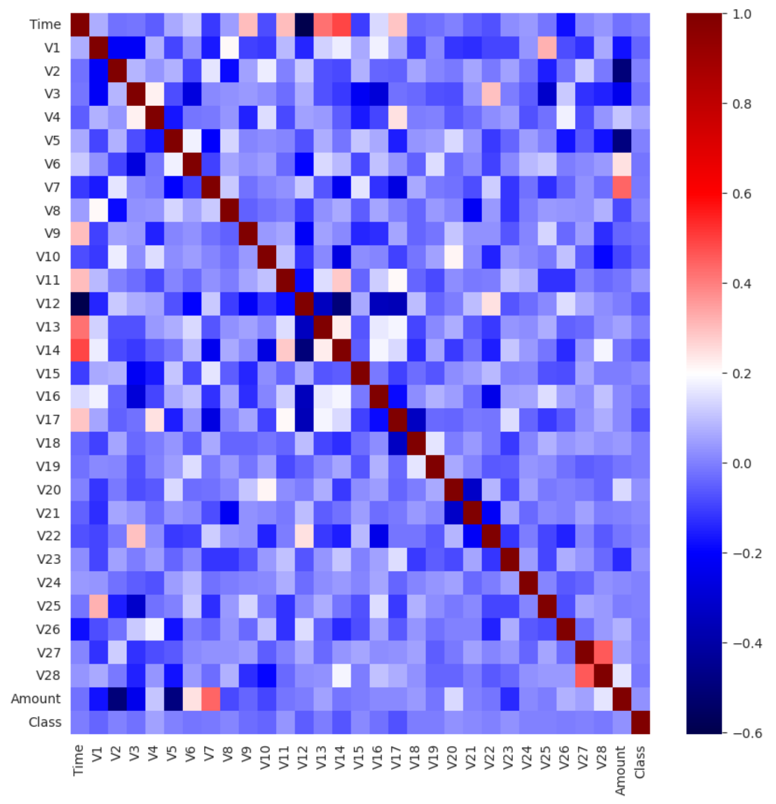
2)

```
# data[data.Class == 0].Time.hist(bins=35, color='blue', alpha=0.6)
plt.figure(figsize=(14, 12))

plt.subplot(2, 2, 1)
data[data.Class == 1].Time.hist(
  bins=35, color='pink', alpha=0.6, label="Fraudulant Transaction"
)
plt.legend()

plt.subplot(2, 2, 2)
data[data.Class == 0].Time.hist(
  bins=35, color='blue', alpha=0.6, label="Non Fraudulant Transaction"
)
plt.legend()
```





```
# heatmap to find any high correlations

plt.figure(figsize=(10,10))
sns.heatmap(data=data.corr(), cmap="seismic")
plt.show();
```

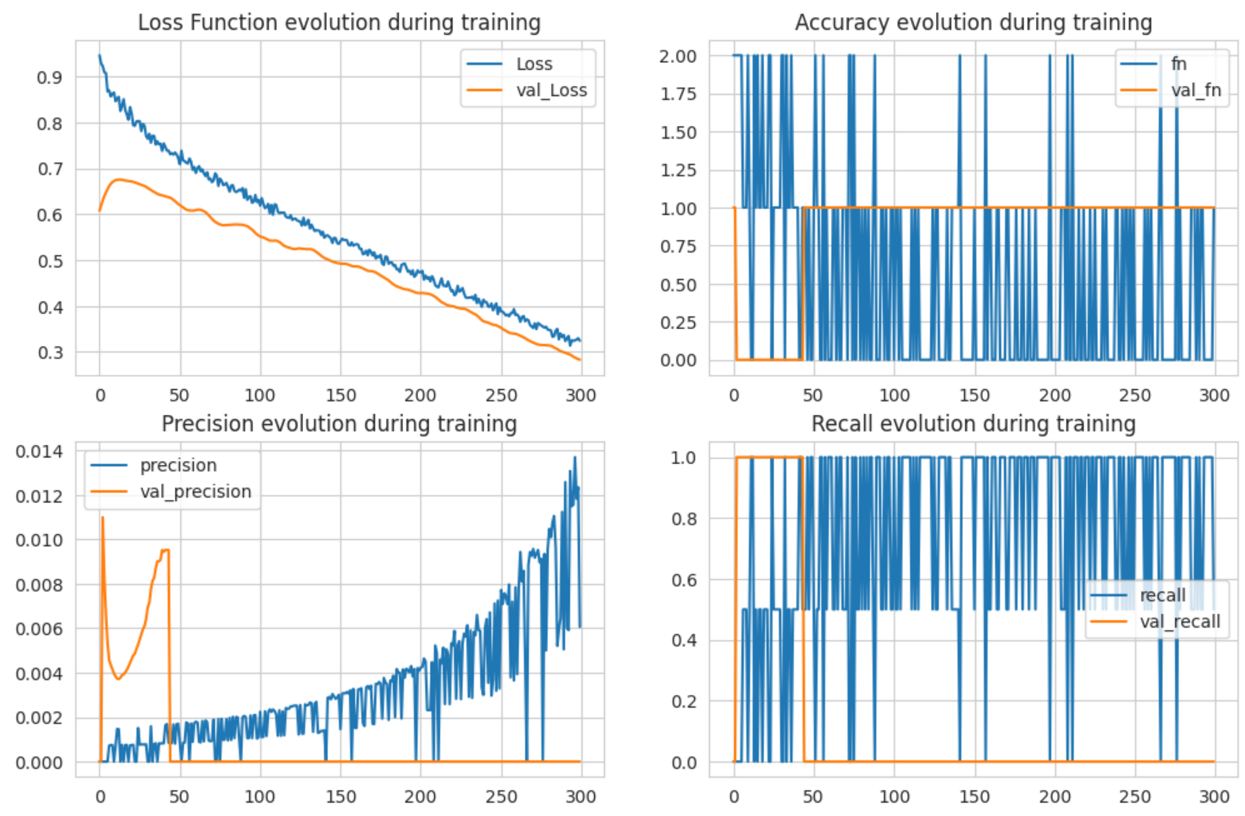


```
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(256, activation='relu',
input_shape=(X_train.shape[-1],)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(1, activation='sigmoid'),
])
METRICS = [
#     keras.metrics.Accuracy(name='accuracy'),
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall')
]

model.compile(optimizer=keras.optimizers.Adam(1e-4),
loss='binary_crossentropy', metrics=METRICS)

callbacks =
[keras.callbacks.ModelCheckpoint('fraud_model_at_epoch_{epoch}.h5')]
class_weight = {0:w_p, 1:w_n}

r = model.fit(
    X_train, y_train,
    validation_data=(X_validate, y_validate),
    batch_size=2048,
    epochs=300,
#     class_weight=class_weight,
    callbacks=callbacks,
)
score = model.evaluate(X_test, y_test)
print(score)
```

OUTPUT:

```
Epoch 1/300
2/2 [==============================] - 8s 1s/step - loss: 0.9461 - fn:
2.0000 - fp: 1445.0000 - tn: 1352.0000 - tp: 0.0000e+00 - precision:
0.0000e+00 - recall: 0.0000e+00 - val_loss: 0.6075 - val_fn: 1.0000 -
val_fp: 49.0000 - val_tn: 650.0000 - val_tp: 0.0000e+00 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00
Epoch 2/300
1/2 [==============>...............] - ETA: 0s - loss: 0.9318 - fn: 2.0000
- fp: 1052.0000 - tn: 994.0000 - tp: 0.0000e+00 - precision: 0.0000e+00 -
recall:
0.0000e+00/usr/local/lib/python3.10/dist-packages/keras/src/engine/trainin
g.py:3000: UserWarning: You are saving your model as an HDF5 file via
`model.save()`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
2/2 [==============================] - 0s 115ms/step - loss: 0.9283 - fn:
2.0000 - fp: 1452.0000 - tn: 1345.0000 - tp: 0.0000e+00 - precision:
0.0000e+00 - recall: 0.0000e+00 - val_loss: 0.6188 - val_fn: 1.0000 -
val_fp: 70.0000 - val_tn: 629.0000 - val_tp: 0.0000e+00 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00
Epoch 3/300
```

```python
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='val_Loss')
plt.title('Loss Function evolution during training')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(r.history['fn'], label='fn')
plt.plot(r.history['val_fn'], label='val_fn')
plt.title('Accuracy evolution during training')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(r.history['precision'], label='precision')
plt.plot(r.history['val_precision'], label='val_precision')
plt.title('Precision evolution during training')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(r.history['recall'], label='recall')
plt.plot(r.history['val_recall'], label='val_recall')
plt.title('Recall evolution during training')
plt.legend()
```



```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print_score(y_train, y_train_pred.round(), train=True)
print_score(y_test, y_test_pred.round(), train=False)

scores_dict = {
    'ANNs': {
        'Train': f1_score(y_train, y_train_pred.round()),
        'Test': f1_score(y_test, y_test_pred.round()),
    },
}
```

OUTPUT:

```
88/88 [==============================] - 0s 3ms/step
47/47 [==============================] - 0s 2ms/step
Train Result:
================================================
Accuracy Score: 99.96%
```

```
_______________________________________________
Classification Report:
                0    1  accuracy  macro avg  weighted avg
precision    1.00 0.67      1.00       0.83          1.00
recall       1.00 1.00      1.00       1.00          1.00
f1-score     1.00 0.80      1.00       0.90          1.00
support   2797.00 2.00      1.00    2799.00       2799.00
_______________________________________________
Confusion Matrix:
 [[2796    1]
 [   0    2]]

Test Result:
================================================
Accuracy Score: 99.93%
_______________________________________________
Classification Report:
              0.0  1.0  accuracy  macro avg  weighted avg
precision    1.00 0.00      1.00       0.50          1.00
recall       1.00 0.00      1.00       0.50          1.00
f1-score     1.00 0.00      1.00       0.50          1.00
support   1500.00 0.00      1.00    1500.00       1500.00
_______________________________________________
Confusion Matrix:
 [[1499    1]
 [   0    0]]

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and
being set to 0.0 in labels with no true samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and
being set to 0.0 in labels with no true samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and
being set to 0.0 in labels with no true samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```python
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train, eval_metric='aucpr')
```

```
y_train_pred = xgb_clf.predict(X_train)
y_test_pred = xgb_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores_dict['XGBoost'] = {
        'Train': f1_score(y_train,y_train_pred),
        'Test': f1_score(y_test, y_test_pred),
}
```

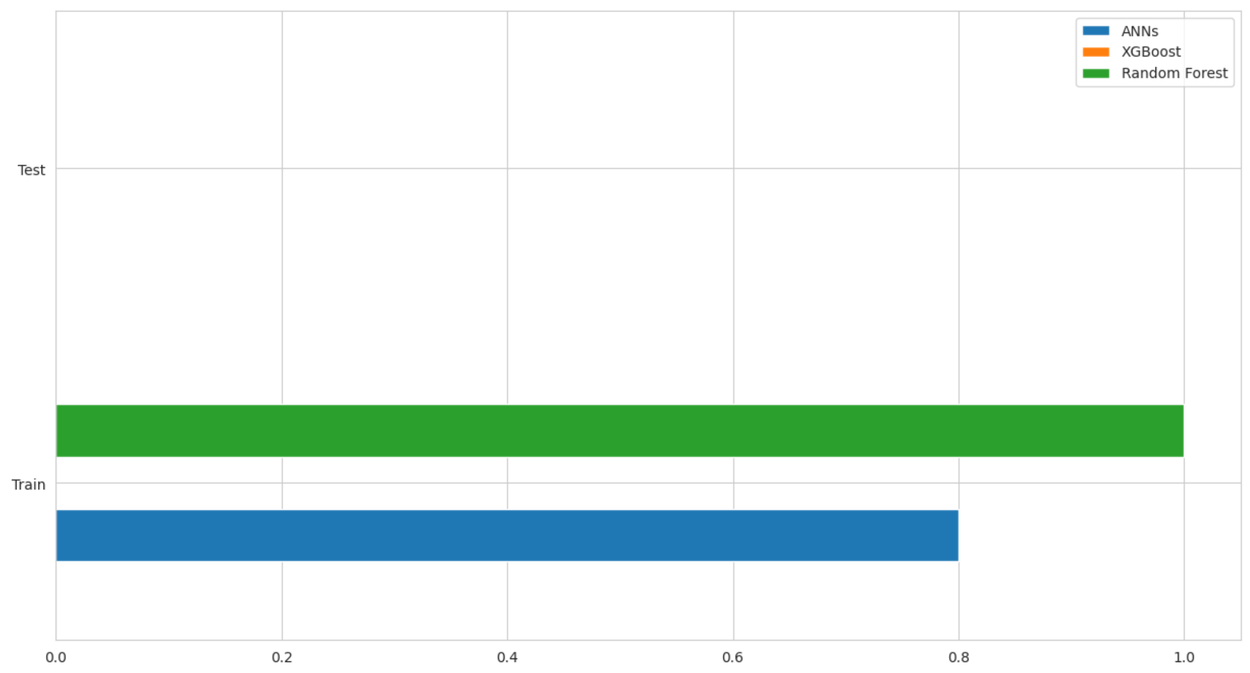
```
Train Result:
================================================
Accuracy Score: 99.93%
_______________________________________________
Classification Report:
                0    1  accuracy  macro avg  weighted avg
precision    1.00 0.00      1.00       0.50          1.00
recall       1.00 0.00      1.00       0.50          1.00
f1-score     1.00 0.00      1.00       0.50          1.00
support   2797.00 2.00      1.00    2799.00       2799.00
_______________________________________________
Confusion Matrix:
 [[2797    0]
 [   2    0]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:
                0  accuracy  macro avg  weighted avg
precision    1.00      1.00       1.00          1.00
recall       1.00      1.00       1.00          1.00
f1-score     1.00      1.00       1.00          1.00
support   1500.00      1.00    1500.00       1500.00
_______________________________________________
Confusion Matrix:
 [[1500]]

/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:885:
UserWarning: `eval_metric` in `fit` method is deprecated for better
compatibility with scikit-learn, use `eval_metric` in constructor
or`set_params` instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1609: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0
due to no true nor predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, "true nor predicted", "F-score is", len(true_sum))
```

```python
from catboost import CatBoostClassifier

cb_clf = CatBoostClassifier()
cb_clf.fit(X_train, y_train)
y_train_pred = cb_clf.predict(X_train)
y_test_pred = cb_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores_dict['CatBoost'] = {
        'Train': f1_score(y_train,y_train_pred),
        'Test': f1_score(y_test, y_test_pred),
}
```

```
OUTPUT:
Learning rate set to 0.015988 0:   learn: 0.6509513 total: 185ms
remaining: 3m 4s 1:    learn: 0.6116274 total: 222ms     remaining: 1m 50s
2:    learn: 0.5744741 total: 243ms     remaining: 1m 20s 3:    learn:
0.5406667   total: 269ms     remaining: 1m 6s 4:    learn: 0.5092576
total: 290ms     remaining: 57.7s
997:  learn: 0.0000731 total: 31.4s     remaining: 62.9ms
998:  learn: 0.0000730 total: 31.4s     remaining: 31.4ms
999:  learn: 0.0000729 total: 31.4s     remaining: 0us
Train Result:
```

```
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:
                0    1  accuracy  macro avg  weighted avg
precision    1.00 1.00      1.00       1.00          1.00
recall       1.00 1.00      1.00       1.00          1.00
f1-score     1.00 1.00      1.00       1.00          1.00
support   2797.00 2.00      1.00    2799.00       2799.00
_______________________________________________
Confusion Matrix:
 [[2797    0]
 [   0    2]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:
                0  accuracy  macro avg  weighted avg
precision    1.00      1.00       1.00          1.00
recall       1.00      1.00       1.00          1.00
f1-score     1.00      1.00       1.00          1.00
support   1500.00      1.00    1500.00       1500.00
_______________________________________________
Confusion Matrix:
 [[1500]]

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py
:1609: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0
due to no true nor predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, "true nor predicted", "F-score is", len(true_sum))
```

```
scores_df = pd.DataFrame(scores_dict)


scores_df.plot(kind='barh', figsize=(15, 8))
```



Importance of Model Training:

➢ Model training is the primary step in machine learning, resulting in a working
model that can then be validated, tested and deployed. The model's performance during
training will eventually determine how well it will work when it is eventually put into an
application for the end-users.
➢ Both the quality of the training data and the choice of the algorithm are
central to the model training phase. In most cases, training data is split into two sets for
training and then validation and testing.
➢The selection of the algorithm is primarily determined by the end-use case.
However, there are always additional factors that need to be considered, such as
algorithm-model complexity, performance, interpretability, computer resource requirements,
and speed. Balancing out these various requirements can make selecting algorithms an
involved and complicated process.

How To Train a Machine Learning Model:

Split the Dataset

➢ Your initial training data is a limited resource that needs to be allocated carefully. Some of it can
be
used to train your model, and some of it can be used to test your model – but you can't use the
same data for
each step. You can't properly test a model unless you have given it a new data set that it hasn't
encountered
before. Splitting the training data into two or more sets allows you to train and then validate the
model using a

single source of data. This allows you to see if the model is overfit, meaning that it performs well with the training
data but poorly with the test data.
➢ A common way of splitting the training data is to use cross-validation. In 10-fold cross-validation, for
example, the data is split into ten sets, allowing you to train and test the data ten times. To do this:
1. Split the data into ten equal parts or folds.
2. Designate one fold as the hold-out fold.
3. Train the model on the other nine folds.
4. Test the model on the hold-out fold
5. Repeat this process ten times, each time selecting a different fold to be the hold-out fold.

Fit and Tune Models:

➢Now that the data is prepared and the model's hyperparameters have been determined, it's time to start training the models. The process is essentially to loop through the different algorithms using
each set of hyperparameter values you've decided to explore. To do this:
1. Split the data.
2. Select an algorithm.
3. Tune the hyperparameter values.
4. Train the model.
5. Select another algorithm and repeat steps 3 and 4..

➢Next, select another set of hyperparameter values you want to try for the same algorithm, cross-validate it again and calculate the new score. Once you have tried each hyperparameter value, you
can repeat these same steps for additional algorithms.

Choose the Best Model:

➢ Now it's time to test the best versions of each algorithm to determine which gives you
the best model overall.
1. Make predictions on your test data.
2. Determine the ground truth for your target variable during the training of that model.
3. Determine the performance metrics from your predictions and the ground truth target variable.
4. Run each finalist model with the test data.

➢ Once the testing is done, you can compare their performance to determine which are
the better models. The overall winner should have performed well (if not the best) in training as well
as in testing. It should also perform well on your other performance metrics (like speed and empirical loss), and – ultimately – it should adequately solve or answer the question posed in your problem statement.

MODEL
EVALUATION

Evaluating the performance of a fraud detection system is critical to ensure its effectiveness in identifying and preventing fraudulent activities while minimizing false positives. Below are common
evaluation metrics and techniques used to assess the performance of a fraud detection system:
Confusion Matrix:
● A confusion matrix provides a clear breakdown of model predictions:
● True Positives (TP): Fraudulent transactions correctly identified as fraud.
● True Negatives (TN): Legitimate transactions correctly identified as legitimate.
● False Positives (FP): Legitimate transactions incorrectly classified as fraud (false alarms).
● False Negatives (FN): Fraudulent transactions incorrectly classified as legitimate (missed frauds).
Precision:
● Precision measures the accuracy of positive predictions (frauds) made by the model. It is calculated as TP / (TP + FP).
● High precision indicates that when the model predicts fraud, it is likely to be correct, but it may miss some actual fraud cases.
Recall (Sensitivity or True Positive Rate):
● Recall measures the ability of the model to identify all actual fraud cases. It is calculated as TP / (TP + FN).
● High recall indicates that the model is good at catching fraud, but it may generate more false alarms.
F1-Score:
● The F1-score is the harmonic mean of precision and recall and provides a balanced measure of a model's performance. It is calculated as 2 * (Precision * Recall) / (Precision + Recall).
Area Under the ROC Curve (AUC-ROC):
● ROC (Receiver Operating Characteristic) curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold values.
● AUC-ROC measures the model's ability to distinguish between classes, with a higher AUC indicating better discrimination.
Area Under the Precision-Recall Curve (AUC-PR):
● The precision-recall curve plots precision against recall at various threshold values.
● AUC-PR summarizes the trade-off between precision and recall, especially when dealing with imbalanced datasets.
Accuracy:
● While not the primary metric for fraud detection (due to class imbalance), accuracy is still useful as a general measure of model performance. It is calculated as (TP + TN) / (TP + TN + FP + FN).
Specificity:

● Specificity measures the model's ability to correctly identify legitimate transactions. It is calculated as TN / (TN + FP).

False Positive Rate (FPR):

● FPR measures the rate of legitimate transactions incorrectly classified as fraud. It is calculated as 1 - Specificity

```
print(f"TRAINING: X_train: {X_train.shape}, y_train:
{y_train.shape}\n{'_'*55}")
print(f"VALIDATION: X_validate: {X_validate.shape}, y_validate:
{y_validate.shape}\n{'_'*50}")
print(f"TESTING: X_test: {X_test.shape}, y_test: {y_test.shape}")
```

```
TRAINING: X_train: (2799, 30), y_train: (2799,)
_______________________________________________________ VALIDATION:
X_validate: (700, 30), y_validate: (700,)
__________________________________________________ TESTING: X_test: (1500,
30), y_test: (1500,)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

X = data.drop('Class', axis=1)
y = data.Class

X_train_v, X_test, y_train_v, y_test = train_test_split(X, y,
                                       test_size=0.3, random_state=42)
X_train, X_validate, y_train, y_validate = train_test_split(X_train_v,
y_train_v,
                                                              test_size=0.2,
random_state=42)

X_train = scalar.fit_transform(X_train)
X_validate = scalar.transform(X_validate)
X_test = scalar.transform(X_test)

w_p = y_train.value_counts()[0] / len(y_train)
w_n = y_train.value_counts()[1] / len(y_train)

print(f"Fraudulant transaction weight: {w_n}")
print(f"Non-Fraudulant transaction weight: {w_p}")
```

```
Finally in our output we have found the fraudulent transaction
Fraudulant transaction weight: 0.0007145409074669524
Non-Fraudulant transaction weight: 0.9992854590925331
```

MY PYTHON FILE

creditcard.ipynb

https://colab.research.google.com/drive/17XLw26HJoT3D_uGvxUxHIkne8wmUDDU_?usp=sharing

Conclusion

In conclusion, the main objective of this project was to find the most suited model in credit card fraud detection in terms of the machine learning techniques chosen for the project, and it was met by building the four models and finding the accuracies of them all, the best model in terms of accuracies is Support Vector Machine which scored 99.94% with only 3 misclassified instances. I believe that using the model will help in decreasing the amount of credit card fraud and increase the customers satisfaction as it will provide them with better experience in addition to feeling secure.

REFERENCE

www.google.com
www.kaggle.com
Github
IEEE articles