```
import os
os.makedirs("/root/.kaggle", exist_ok=True)

from google.colab import files
files.upload()

!cp /content/kaggle.json /root/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json
```

Choose files kaggle.json
kaggle.json(application/json) - 72 bytes, last modified: 23/09/2025 - 100% done

```
Saving kaggle.json to kaggle.json
```

```
!kaggle datasets download -d mlg-ulb/creditcardfraud
!unzip creditcardfraud.zip
```

```
Dataset URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
License(s): DbCL-1.0
Downloading creditcardfraud.zip to /content
  0% 0.00/66.0M [00:00<?, ?B/s]
100% 66.0M/66.0M [00:00<00:00, 1.39GB/s]
Archive:  creditcardfraud.zip
  inflating: creditcard.csv
```

Project Explanation – Credit Card Fraud Detection (Data Exploration)

In this project, I started by performing an exploratory data analysis (EDA) on the Credit Card Fraud Detection dataset, which contains anonymized transaction data. The dataset includes numerical features (V1–V28), the transaction amount, the transaction time, and a target column Class (0 = legitimate transaction, 1 = fraudulent transaction).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
from imblearn.over_sampling import SMOTE
```

I used the Pandas library to load the dataset into a DataFrame for further analysis. Displayed the first five rows of the dataset to understand the structure and check column names.

Verified that the target column is labeled as Class. The dataset contains 284,807 rows and 31 columns.

This shows that we are working with a large-scale dataset, which is important for building reliable machine learning models.

Provided a statistical overview of the numerical columns (mean, min, max, standard deviation, etc.).

This helped to identify the ranges and distributions of different features.

```
import pandas as pd

csv_path = "/content/creditcard.csv"
df = pd.read_csv(csv_path)

print(df.head())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -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]
```

```
!ls /content
```

```
creditcard.csv  creditcardfraud.zip  kaggle.json  sample_data
```

```
import pandas as pd

df = pd.read_csv("/content/creditcard.csv")

print(df.head())

print("Shape:", df.shape)

print(df.describe())

print(df['Class'].value_counts())
```

```
[5 rows x 31 columns]
Shape: (284807, 31)
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00
```

```
[8 rows x 31 columns]
Class
0    284315
1       492
Name: count, dtype: int64
```

```
print("Shape of dataset:", df.shape)
print(df['Class'].value_counts())
```

```
Shape of dataset: (284807, 31)
Class
0    284315
1       492
Name: count, dtype: int64
```

```
X = df.drop("Class", axis=1)
y = df["Class"]
```

The dataset is highly imbalanced:

Most transactions are legitimate (Class 0).

A very small fraction are fraudulent (Class 1).

This imbalance is a critical challenge in fraud detection and requires special handling (e.g., resampling, anomaly detection, or cost-sensitive learning).

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
```

```
print("Train size:", X_train.shape, " Test size:", X_test.shape)
```

```
Train size: (227845, 30)  Test size: (56962, 30)
```

```
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

```
print("After SMOTE:", y_res.value_counts())
```

```
After SMOTE: Class
0    227451
1    227451
Name: count, dtype: int64
```

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_res, y_res)
y_pred_lr = log_reg.predict(X_test)
```

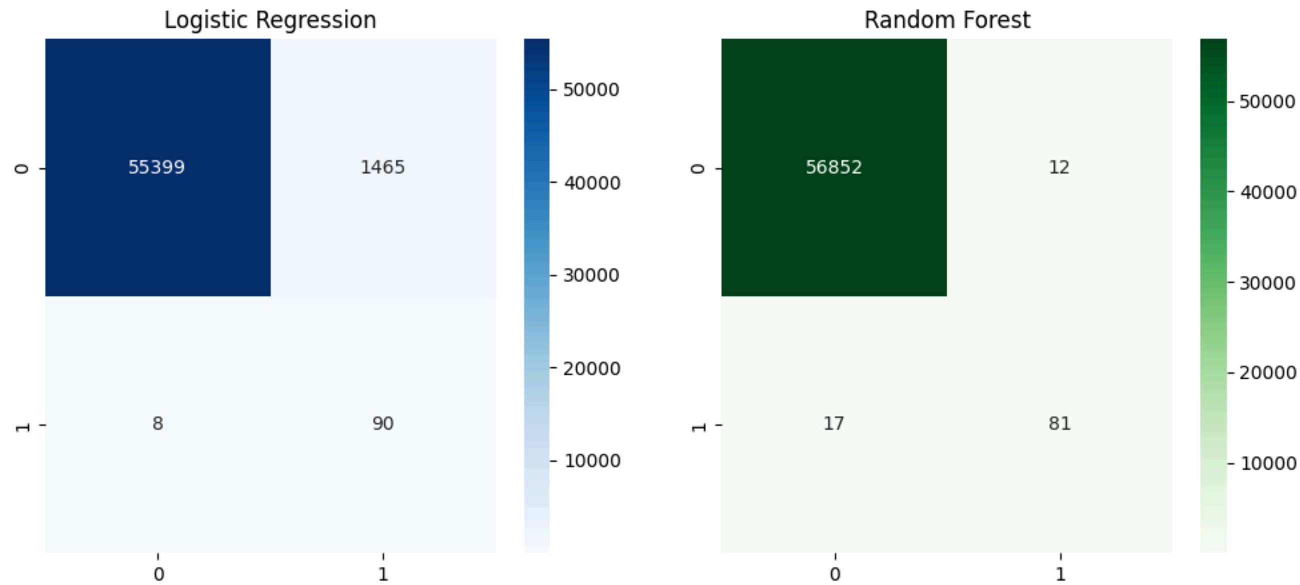
```
print("---- Logistic Regression ----")
print(classification_report(y_test, y_pred_lr))
print("ROC-AUC:", roc_auc_score(y_test, log_reg.predict_proba(X_test)[:,1]))
```

```
---- Logistic Regression ----
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.97     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.97      0.99     56962

ROC-AUC: 0.970987885165321
```

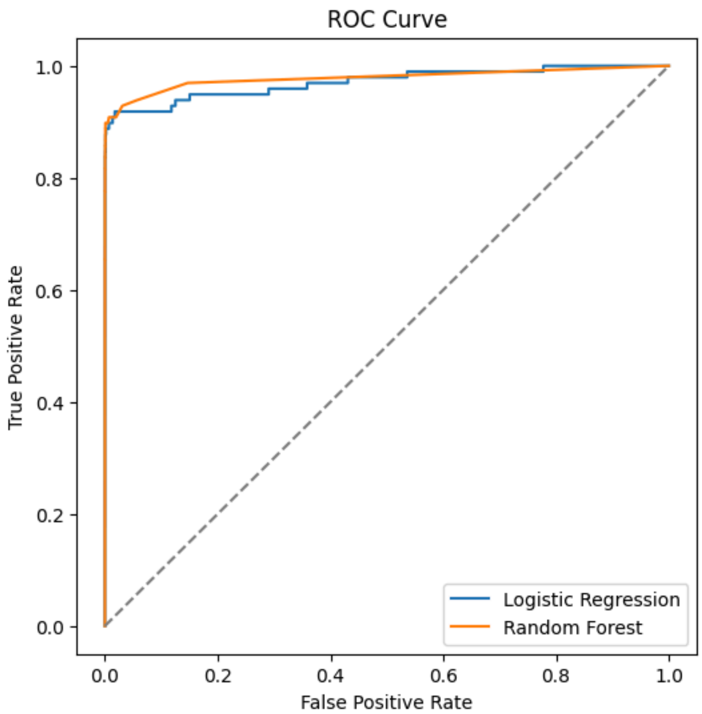
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_res, y_res)
y_pred_rf = rf.predict(X_test)
```

```
print("---- Random Forest ----")
print(classification_report(y_test, y_pred_rf))
print("ROC-AUC:", roc_auc_score(y_test, rf.predict_proba(X_test)[:,1]))
```

```
fig, ax = plt.subplots(1, 2, figsize=(12,5))

sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt="d", cmap="Blues", ax=ax[0])
ax[0].set_title("Logistic Regression")

sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt="d", cmap="Greens", ax=ax[1])
ax[1].set_title("Random Forest")

plt.show()
```




```
fpr1, tpr1, _ = roc_curve(y_test, log_reg.predict_proba(X_test)[:,1])
fpr2, tpr2, _ = roc_curve(y_test, rf.predict_proba(X_test)[:,1])

plt.figure(figsize=(6,6))
plt.plot(fpr1, tpr1, label="Logistic Regression")
plt.plot(fpr2, tpr2, label="Random Forest")
plt.plot([0,1],[0,1],"--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



Double-click (or enter) to edit

```
Conclusion:

At this stage, I successfully:

Loaded and inspected the dataset.

Understood the feature set and target variable.

Identified class imbalance as a key issue.
```

Conclusion:

At this stage, I successfully:

Loaded and inspected the dataset.

Understood the feature set and target variable.

Identified class imbalance as a key issue.