fraud-detect

August 3, 2025

[ ]:

1 Task

Build a classification model to detect fraudulent transactions using the data from "/content/creditcard.csv". The project should address the imbalanced nature of the dataset and use anomaly detection techniques.

1.1 Load the data

1.1.1 Subtask:

Load the credit card transaction data from the CSV file into a pandas DataFrame.

Reasoning: I will import the pandas library and load the data from the specified CSV file into a DataFrame. Then I will display the first few rows to verify the data has been loaded correctly.

[1]:
```python
import pandas as pd

df = pd.read_csv('/content/creditcard.csv')
display(df.head())
```

```
   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    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
```

```
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
```

1.2 Explore the data

1.2.1 Subtask:

Analyze the dataset to understand its structure, features, and the distribution of fraudulent vs. non-fraudulent transactions.

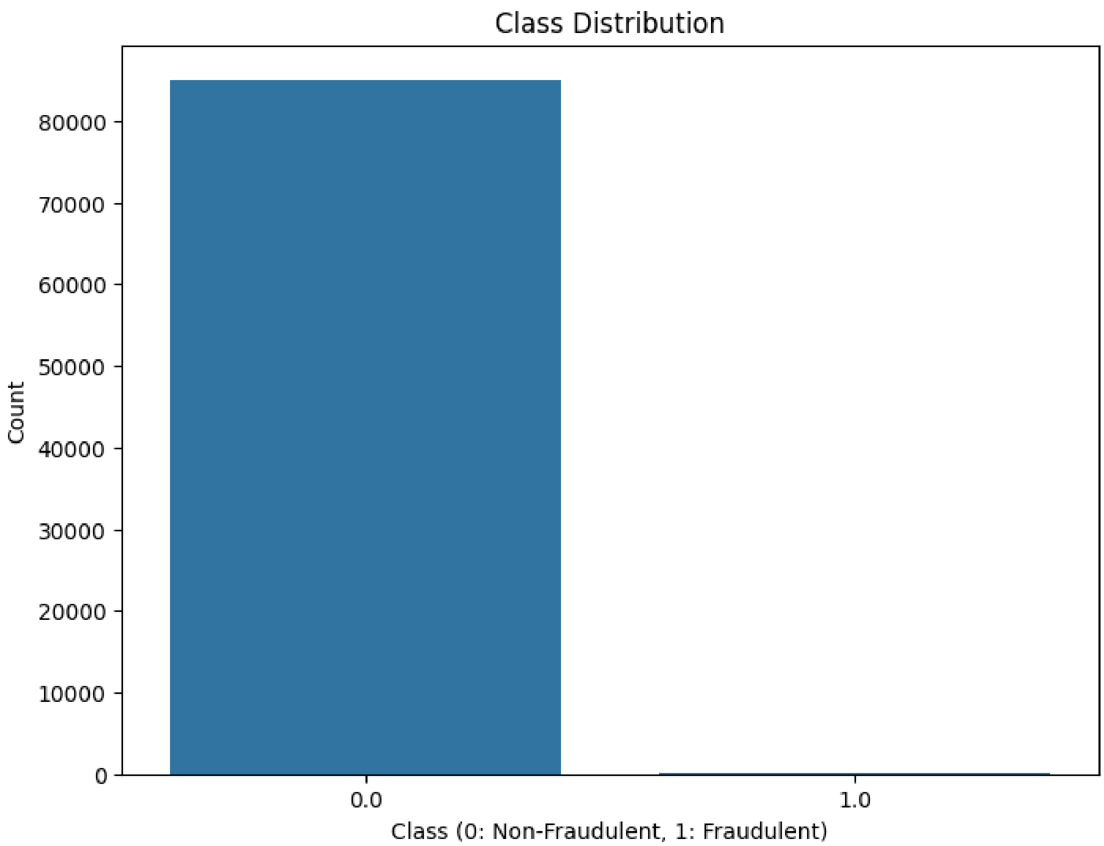
Reasoning: To analyze the dataset, I will first get a concise summary of the DataFrame using `.info()` to understand the data types and null values. Then, I will generate descriptive statistics for numerical columns with `.describe()`. Next, I will count the fraudulent and non-fraudulent transactions in the 'Class' column. Finally, I will visualize the class distribution using a seaborn count plot to better understand the imbalance in the data. All these steps are part of the initial data analysis requested by the user.

[2]:
```
import seaborn as sns
import matplotlib.pyplot as plt

print("Dataset Info:")
df.info()

print("\nDescriptive Statistics:")
display(df.describe())

print("\nClass Distribution:")
class_counts = df['Class'].value_counts()
print(class_counts)

plt.figure(figsize=(8, 6))
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.xlabel('Class (0: Non-Fraudulent, 1: Fraudulent)')
plt.ylabel('Count')
plt.show()
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85259 entries, 0 to 85258
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    85259 non-null  int64
 1   V1      85259 non-null  float64
 2   V2      85259 non-null  float64
```

```
 3   V3      85259 non-null  float64
 4   V4      85259 non-null  float64
 5   V5      85259 non-null  float64
 6   V6      85259 non-null  float64
 7   V7      85259 non-null  float64
 8   V8      85259 non-null  float64
 9   V9      85259 non-null  float64
 10  V10     85259 non-null  float64
 11  V11     85259 non-null  float64
 12  V12     85259 non-null  float64
 13  V13     85259 non-null  float64
 14  V14     85259 non-null  float64
 15  V15     85259 non-null  float64
 16  V16     85259 non-null  float64
 17  V17     85259 non-null  float64
 18  V18     85259 non-null  float64
 19  V19     85259 non-null  float64
 20  V20     85259 non-null  float64
 21  V21     85259 non-null  float64
 22  V22     85259 non-null  float64
 23  V23     85259 non-null  float64
 24  V24     85259 non-null  float64
 25  V25     85259 non-null  float64
 26  V26     85259 non-null  float64
 27  V27     85258 non-null  float64
 28  V28     85258 non-null  float64
 29  Amount  85258 non-null  float64
 30  Class   85258 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 20.2 MB

Descriptive Statistics:
               Time            V1            V2            V3            V4  \
count  85259.000000  85259.000000  85259.000000  85259.000000  85259.000000
mean   38698.541691     -0.262585     -0.039207      0.679054      0.163611
std    15668.300002      1.878484      1.670189      1.366683      1.363280
min        0.000000    -56.407510    -72.715728    -33.680984     -5.172595
25%    31595.000000     -1.025883     -0.602941      0.184642     -0.722311
50%    41180.000000     -0.258057      0.069615      0.762733      0.186825
75%    50942.000000      1.153099      0.724933      1.390659      1.042062
max    60684.000000      1.960497     18.902453      4.226108     16.715537

                 V5            V6            V7            V8            V9  \
count  85259.000000  85259.000000  85259.000000  85259.000000  85259.000000
mean      -0.277616      0.094861     -0.111651      0.054145     -0.021205
std        1.375170      1.303646      1.237707      1.223035      1.135023
min      -42.147898    -26.160506    -31.764946    -73.216718     -9.283925
```

```
25%     -0.896999    -0.644375    -0.603378    -0.140721    -0.690902
50%     -0.311837    -0.155047    -0.072501     0.070103    -0.097017
75%      0.257707     0.488674     0.416990     0.352738     0.602616
max     34.801666    22.529298    36.677268    20.007208    10.392888

       …           V21          V22          V23          V24  \
count  …  85259.000000 85259.000000 85259.000000 85259.000000
mean   …     -0.029273    -0.105613    -0.037762     0.008986
std    …      0.731860     0.635127     0.626519     0.595040
min    …    -34.830382   -10.933144   -26.751119    -2.836627
25%    …     -0.223696    -0.524879    -0.178156    -0.322420
50%    …     -0.057764    -0.080822    -0.049921     0.064547
75%    …      0.118290     0.309640     0.080706     0.405794
max    …     22.614889    10.503090    18.946734     4.014444

               V25          V26          V27          V28       Amount  \
count 85259.000000 85259.000000 85258.000000 85258.000000 85258.000000
mean      0.133598     0.026270     0.001449     0.001934    98.374057
std       0.441017     0.497236     0.389486     0.328312   268.381509
min      -7.495741    -2.534330    -9.390980    -9.617915     0.000000
25%      -0.130610    -0.326403    -0.063467    -0.005899     7.680000
50%       0.171852    -0.072601     0.009302     0.022947    26.990000
75%       0.421184     0.306305     0.082480     0.076063    89.900000
max       5.525093     3.517346    12.152401    33.847808 19656.530000

             Class
count 85258.000000
mean      0.002393
std       0.048857
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000

[8 rows x 31 columns]

Class Distribution:
Class
0.0    85054
1.0      204
Name: count, dtype: int64
```



1.3 Preprocess the data

1.3.1 Subtask:

Scale the 'Amount' feature and drop the 'Time' feature as it is not relevant for the model.

Reasoning: Scale the 'Amount' feature using StandardScaler and drop the 'Time' feature as instructed.

[3]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])

df = df.drop('Time', axis=1)

display(df.head())
```

```
         V1        V2        V3        V4        V5        V6        V7  \
0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
```

```
3 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9       V10  …       V21       V22       V23       V24  \
0  0.098698  0.363787  0.090794  … -0.018307  0.277838 -0.110474  0.066928
1  0.085102 -0.255425 -0.166974  … -0.225775 -0.638672  0.101288 -0.339846
2  0.247676 -1.514654  0.207643  …  0.247998  0.771679  0.909412 -0.689281
3  0.377436 -1.387024 -0.054952  … -0.108300  0.005274 -0.190321 -1.175575
4 -0.270533  0.817739  0.753074  … -0.009431  0.798278 -0.137458  0.141267

        V25       V26       V27       V28    Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  0.190946    0.0
1  0.167170  0.125895 -0.008983  0.014724 -0.356525    0.0
2 -0.327642 -0.139097 -0.055353 -0.059752  1.044363    0.0
3  0.647376 -0.221929  0.062723  0.061458  0.093621    0.0
4 -0.206010  0.502292  0.219422  0.215153 -0.105761    0.0

[5 rows x 30 columns]
```

1.4 Split the data

1.4.1 Subtask:

Split the data into training and testing sets.

Reasoning: To prepare the data for model training, I will first handle missing values by dropping rows with `NaN` values. Then, I will separate the features (X) from the target variable (y), which is the 'Class' column. Finally, I will split the data into training and testing sets using `train_test_split`, ensuring a stratified split to maintain the class distribution, which is crucial due to the imbalanced nature of the dataset.

[4]:
```
from sklearn.model_selection import train_test_split

df.dropna(inplace=True)

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

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

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (68206, 29)
X_test shape: (17052, 29)
```

```
y_train shape: (68206,)
y_test shape: (17052,)
```

1.5 Train a model

1.5.1 Subtask:

Train a Logistic Regression model on the training data.

Reasoning: I need to train a Logistic Regression model. To do this, I will first import the `LogisticRegression` class, then create an instance of the model, and finally, train it on the `X_train` and `y_train` data.

[5]:
```python
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
```

[5]:
```
LogisticRegression()
```

1.6 Evaluate the model

1.6.1 Subtask:

Evaluate the model's performance on the test data using metrics like accuracy, precision, recall, and the F1-score.

Reasoning: I will now evaluate the trained logistic regression model on the test data. I will import the necessary metrics from `sklearn.metrics`, make predictions on the test set, and then calculate and print the accuracy, precision, recall, and F1-score to assess the model's performance.

[6]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

y_pred = lr_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

```
Accuracy: 0.9986511846117757
Precision: 0.7647058823529411
Recall: 0.6341463414634146
F1-score: 0.6933333333333334
```