Project: Fraud Detection Analysis

This document provides a detailed, beginner-friendly explanation of the

```
Fraud Detection analysis.ipynb
```

Jupyter Notebook. It breaks down each step of the machine learning pipeline, from data preparation to model deployment, to help you understand how the fraud detection system was built.

1. Project Overview

The main goal of this project is to build a machine learning model that can accurately predict if a financial transaction is fraudulent. This is a crucial task for banks and online payment companies to prevent financial losses. The project is a classic **binary classification** problem, which means the model has to classify each transaction into one of two categories:

```
isFraud
```

(1) or

```
isFraud
```

(0).

This notebook follows the standard data science workflow:

1. **Data Exploration:** Understanding the data to find patterns and problems.

2. **Data Preprocessing:** Cleaning and preparing the data for the machine learning model.
3. **Model Building:** Training a model to learn from the data.
4. **Evaluation:** Checking how well the model performs.
5. **Deployment:** Saving the model to use it in a live application (the Streamlit app).

2. Detailed Code Explanation

Section 1: Setup and Data Loading

Code:

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

Explanation: This is like gathering your tools before you start a project. You're importing four essential libraries for data analysis in Python:

- `pandas`

  : The most important tool for working with tabular data (like a spreadsheet). It allows you to load, manipulate, and analyze data easily.

- **`numpy`**

  : A powerful library for numerical operations. It is highly efficient for performing calculations on large datasets.
- **`matplotlib.pyplot`**

  and

  `seaborn`

  : These are your visualization libraries, your "art supplies." They help you create various charts and graphs to understand your data visually.

  `seaborn`

  is built on top of

  `matplotlib`

  , making it easier to create professional-looking plots.

Code:

```
import warnings
warnings.filterwarnings("ignore")
sns.set(style="whitegrid")
```

Explanation: This is a bit of digital housecleaning. The first two lines tell Python to ignore minor warning messages that don't affect the code's functionality, keeping your notebook tidy. The third line sets a clean, professional style for all the charts you'll create, with a white background and a grid.

Code:

```
            df = pd.read_csv("D:\\Rakesh ap\\Projects\\Fraud Detection Project\\AIML
Dataset.csv")
```

Explanation: This is the crucial step of loading the dataset. The

```
pd.read_csv()
```

command reads your

```
.csv
```

(Comma Separated Values) file from its location on your computer. It converts this data into a

```
DataFrame
```

, which is a table-like structure, and stores it in the variable

```
df
```

(a common shorthand for DataFrame).

Section 2: Exploratory Data Analysis (EDA)

Code:

```
df.head()
```

Explanation: This is your first look at the data. The

```
.head()
```

function displays the first five rows of your DataFrame. This gives you a quick snapshot of the columns, the types of values they contain, and the overall structure of your dataset.

Code:

```
df.info()
```

Explanation: This command provides a summary of the entire DataFrame. It tells you:

- The total number of rows (entries).
- The name of each column.
- The data type of each column (e.g.,

  ```
  int64
  ```

  for integers,

  ```
  float64
  ```

for decimals, and

```
object
```

for text).

- Whether there are any missing values (

```
Non-Null Count
```

).

Code:

```
df.isnull().sum().sum()
```

Explanation: This is a quick and effective way to check for any missing information. It calculates the total count of all missing values (represented as

```
NaN
```

). The output

```
0
```

means your dataset is perfectly clean and ready for analysis.

Code:

```
df["isFraud"].value_counts()
```

and

```
df["isFlaggedFraud"].value_counts()
```

Explanation: This code investigates the most important column for this project:

```
isFraud
```

. The

```
value_counts()
```

function counts how many times each unique value appears in the column. It shows you the number of fraudulent (1) and non-fraudulent (0) transactions. You will notice a huge difference in the counts, which is a common characteristic of fraud datasets—they are **highly imbalanced**.

Code:

```
round((df["isFraud"].value_counts()[1]/df.shape[0])*100,2)
```

Explanation: This line calculates the exact percentage of fraudulent transactions. You are taking the count of fraudulent transactions, dividing it by the total number of transactions, and then multiplying by 100. The result,

```
0.13
```

, highlights just how rare fraudulent cases are in this dataset, which is a key challenge for the model.

Section 3: Feature Engineering and Data Preprocessing

Code:

```
df['nameDest'] = df['nameDest'].str.slice(0, 1)
df['nameOrig'] = df['nameOrig'].str.slice(0, 1)
```

Explanation: This step simplifies the data by extracting just the first character from the

```
nameDest
```

and

```
nameOrig
```

columns. For example, 'C' might stand for a customer account and 'M' for a merchant. This could potentially give the model a useful feature to work with.

Code:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['type'] = le.fit_transform(df['type'])
df['nameDest'] = le.fit_transform(df['nameDest'])
df['nameOrig'] = le.fit_transform(df['nameOrig'])
```

Explanation: Machine learning models can only work with numbers, not text. This code uses

```
LabelEncoder
```

to convert the text values in the

```
type
```

,

```
nameDest
```

, and

```
nameOrig
```

columns into numerical representations. For example, 'PAYMENT' might become 0, 'CASH_OUT' might become 1, and so on.

Code:

```
            X = df.drop("isFraud", axis=1)
            y = df["isFraud"]
            from sklearn.model_selection import train_test_split
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Explanation: This is a critical step for preparing your model for a fair test.

- `X` contains your **features** (the input data, or all columns except `isFraud`).
- `y` contains your **target** (the output you want to predict, which is the `isFraud` column).
- `train_test_split()` then divides your data into a **training set** (80% of the data, used to teach the model) and a **testing set** (20%, used to see how well the model works on data it has never seen).

Section 4: Building and Evaluating the Model

Code:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
```

```
            # ... code to define the pipeline ...
            pipeline.fit(X_train, y_train)
```

Explanation: A

```
Pipeline
```

is a great way to combine all the data preprocessing steps and the model into a single, neat package. It ensures that the same transformations (like converting text to numbers and scaling the data) are applied consistently to both your training and testing data. The

```
pipeline.fit()
```

command trains the entire pipeline on your training data.

Code:

```
            from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score
            y_pred = pipeline.predict(X_test)
            # ... code to print metrics ...
```

Explanation: Once the model is trained, you need to check its performance.

-
  ```
  pipeline.predict(X_test)
  ```

tells the model to make predictions on the test data it has never seen before.
- The rest of the code calculates various **metrics** to evaluate the model's performance. For a fraud detection model, **precision** and **recall** are more important than accuracy because of the imbalanced data.

Section 5: Saving the Model for Deployment

Code:

```
            import joblib
            joblib.dump(pipeline, 'fraud_detection_pipeline.pkl')
```

Explanation: This is the final step where you save your trained model and all the preprocessing steps to a single file. The

```
joblib
```

library is a simple and efficient way to save (

```
dump
```

) the entire machine learning pipeline (

```
pipeline
```

) to a file named

```
fraud_detection_pipeline.pkl
```

. This

```
.pkl
```

file can then be loaded by your Streamlit application, so you don't have to retrain the model every time the app runs.

How to Use This App & My Key Learnings

How to Use This App

The live Streamlit application simplifies the model into a user-friendly interface. All the complex data preprocessing and prediction logic are handled behind the scenes.

1. **Enter Transaction Details:** On the app's home page, you will see several input fields. You need to enter the details of a transaction, such as the

   ```
   Transaction Type
   ```

   ,

   ```
   Amount
   ```

   , and the balances of the sender and receiver accounts.
2. **Click "Predict":** Once you've entered the information, click the "Predict" button.
3. **Get the Result:** The app will use the saved model (

   ```
   fraud_detection_pipeline.pkl
   ```

) to instantly analyze your input data and tell you if the transaction is likely to be fraudulent or not. The output will be a simple "1" for fraud or "0" for not fraud.

My Key Learnings

This project was a great learning experience. Here are some of the most important takeaways:

- **The Importance of EDA:** You can't just jump into building a model. The exploratory data analysis phase was crucial for discovering that the data was highly imbalanced. This understanding guided my choice of evaluation metrics (focusing on precision and recall instead of just accuracy).
- **The Power of Pipelines:** Using

  ```
  sklearn.pipeline
  ```

  was a game-changer. It taught me how to package all my data preprocessing steps and the model into a single, reliable object. This makes the code much cleaner and prevents common errors that occur when deploying models.
- **The Real-World Challenge of Deployment:** Building the model is only half the battle. This project showed me the challenges of taking a model from a notebook to a live application. I learned about the importance of using a

  ```
  requirements.txt
  ```

  file to ensure the live environment has the exact same library versions as the one used for training the model. This is a common and critical step for any data science project.