Project Explanation: Online Payment Fraud Detection Model

This document explains the entire Python script used to build, train, and evaluate a machine learning model for detecting fraudulent online payments.

I. The Goal of the Code

The main goal is to create a highly accurate **Ensemble Model** (a "super-model" built from several smaller models) that can look at a new transaction and tell us if it is legitimate or **fraudulent** (the target we are trying to predict).

II. Understanding the Data Columns

Your model uses a dataset where each row is a transaction. Here are the most important columns and what they represent:

Column Name	Type	Purpose in Fraud Detection
`isFraud`	**Integer (0 or 1)**	**Target Variable.** This is what we want to predict: `1` means the transaction was fraud, and `0` means it was safe.
`step`	Integer	Represents the hour of the simulation. (e.g., `step 1` is hour 1, `step 743` is hour 743).
`type`	Text (Object)	The type of transaction, like `CASH_OUT`, `TRANSFER`, `PAYMENT`, or `DEBIT`.
`amount`	Decimal	The monetary amount of the transaction.
`oldbalanceOrg`	Decimal	The starting balance of the customer who made the transfer (`nameOrig`).
`newbalanceOrig`	Decimal	The balance of the customer after the transaction.
`oldbalanceDest`	Decimal	The starting balance of the recipient (`nameDest`).
`newbalanceDest`	Decimal	The balance of the recipient after the transaction.
`isFlaggedFraud`	**Integer (0 or 1)**	Whether the system flagged the transaction as suspicious based on simple internal rules (usually used only for analysis, not prediction).
`nameOrig`, `nameDest`	Text (Object)	Customer/Recipient IDs. These are typically dropped because they are identifiers that don't help the model generalize.

III. Step-by-Step Code Explanation

A. Setup and Data Loading

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

data = pd.read_csv('...')
data.head()
```

- **Imports:** We bring in libraries (collections of tools):
  - `numpy` and `pandas` (`pd`): Essential for handling and manipulating data in tables (DataFrames).
  - `matplotlib.pyplot` (`plt`) and `seaborn` (`sns`): Used for making charts and graphs (visualizations).
  - `%matplotlib inline` : A special command to make charts appear right inside the notebook.
- **Data Loading:** The script reads your dataset (`new_data.csv`) into a pandas DataFrame named `data` . `data.head()` shows you the first few rows.

B. Initial Data Exploration (Data Types)

```
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:", len(object_cols))
# ... (Similar code for 'int' and 'float' types)
```

- This section checks the type of data in each column (e.g., numbers, text, etc.).
- **'object'** type variables are usually text (like `type` , `nameOrig` , `nameDest`), which we call **Categorical Variables** (variables that represent categories, not measurable quantities). Machine learning models cannot directly process text, so they need special handling.

C. Visualizing Data Patterns (Exploratory Data Analysis)

```
sns.barplot(x='type', y='amount', data=data)
data['isFraud'].value_counts()
```

- **Bar Plot:** `sns.barplot(x='type', y='amount', data=data)` creates a bar chart comparing the average transaction `amount` across different transaction `type` categories. This helps see if certain transaction types involve higher amounts than others.
- **Value Counts:** `data['isFraud'].value_counts()` tells us exactly how many fraud (`1`) versus non-fraud (`0`) transactions are in the data. This is crucial for checking **Class Imbalance** (when one category is much rarer than others, which is very common in fraud data).

D. Correlation Heatmap (Feature Relationships)

```
sns.heatmap(data.apply(lambda x: pd.factorize(x)[0]).corr(),
    # ... plotting arguments
    annot=True)
```

- **Purpose:** This plot helps you quickly see how strongly every pair of columns relates to every other column.
- `pd.factorize(x)[0]` **:** This is a trick used to temporarily turn text columns (like `type`) into numbers so they can be included in the correlation calculation.
- **Heatmap:** A visual matrix where:
  - **Dark Color (Green/Teal):** Strong positive relationship (as one value goes up, the other tends to go up).
  - **Dark Color (Brown/Yellow):** Strong negative relationship (as one value goes up, the other tends to go down).
  - **Light Color (White):** Weak or no relationship.

E. Data Preparation for Modeling

```
type_new = pd.get_dummies(data['type'], drop_first=True) # ...
data_new = pd.concat([data, type_new], axis=1) # ...
X = data_new.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1) # ...
y = data_new['isFraud'] # ...
```

- **One-Hot Encoding (** `pd.get_dummies` **):** This is how we handle the categorical `type` column. We convert the single 'type' column into several new number columns (e.g., `CASH_OUT`, `TRANSFER`, etc.). If a transaction is a 'TRANSFER', the `TRANSFER` column gets a `1`, and all others get a `0`.
- **Feature Selection:**
  - `X` (**Features**): This is the input data used to train the model. We drop columns that are not useful or are the target itself: `isFraud`, the old `type` column, and the identifier columns (`nameOrig`, `nameDest`).
  - `y` (**Target**): This is the output we want to predict (`isFraud`).

IV. Training the Machine Learning Models

A. Data Splitting

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

- **Train-Test Split:** We divide our data into two parts:
  - **Training Data (70%):** Used to teach the model.

- **Test/Validation Data (30%):** Used to see how well the model works on data it has **never seen before**. This is the true test of performance.

- `random_state=42` **:** Ensures that every time you run the code, the split is the exact same, making your results **reproducible** (reliable and consistent).

B. Defining the Models

We define three different Machine Learning models, each specialized in finding patterns:

1. **Logistic Regression (** `log_clf` **):** A fundamental, simple model that finds a **linear relationship** between inputs and the probability of fraud.
2. **Random Forest Classifier (** `rf_clf` **):** An **ensemble model** (built from multiple simple decision trees) that is excellent at capturing complex, non-linear relationships.
   - `n_estimators=7` : We are using 7 individual decision trees.
3. **XGBoost Classifier (** `xgb_clf` **):** A highly efficient and advanced **ensemble model** that uses a technique called *Gradient Boosting*. It performed exceptionally well after fixing a critical issue:
   - `scale_pos_weight=773` **(CRITICAL FIX):** This tells the model to give the rare "Fraud" cases 773 times more importance than the common "Not Fraud" cases, directly solving the **Class Imbalance** problem.

C. Creating the Ensemble (The "Super-Model")

```
voting_clf = VotingClassifier(
    estimators=ensemble_models,
    voting='soft',
    n_jobs=-1
)
```

- **Voting Classifier:** This combines the predictions of the three models (`lr` , `xgb` , `rf`) to create a single, more robust prediction.
- `voting='soft'` **:** Means the final prediction is based on the **average probability** given by each model, which is best when evaluating models using the ROC AUC score.
- `n_jobs=-1` **:** Tells the script to use all available processing power (CPU cores) to train the models faster.

V. Training and Evaluation Metrics

A. Training the Final Model

```
voting_clf.fit(X_train, y_train)
```

- This is the moment of training! The `VotingClassifier` trains all three models simultaneously using the `X_train` data.

B. Final Evaluation Metrics

```
train_preds_ensemble = voting_clf.predict_proba(X_train)[:, 1]
print('Ensemble Training ROC AUC:', ras(y_train, train_preds_ensemble))
# ... (Validation ROC AUC is calculated similarly using X_test)
```