CREDIT CARD FRAUD DETECTION USING DATA SCIENCE

Project Title: Credit Card Fraud Detection using Data Science

Objective: The primary aim of this project is to develop a machine learning model that can effectively identify and prevent fraudulent credit card transactions, ultimately improving the security of financial transactions.

Key Steps in the Project:

1. Data Collection:
- Gather historical credit card transaction data, which includes both legitimate and fraudulent transactions.
- The dataset should include features such as transaction amount, time, and various other transaction-related attributes.

2. Data Preprocessing:
- Handle missing values and outliers.
- Normalize or scale numerical features.
- Encode categorical features if applicable.

3. Exploratory Data Analysis (EDA):
- Visualize the data to understand patterns and relationships.
- Explore the distribution of legitimate and fraudulent transactions.
- Identify any potential correlations between variables.

4. Feature Selection and Engineering:
- Select relevant features that contribute to fraud detection.
- Create new features if they provide valuable information, e.g., transaction frequency or transaction velocity.

5. Model Selection:
- Choose appropriate machine learning algorithms for classification, such as logistic regression, random forests, or support vector machines.
- Consider deep learning models like neural networks for more complex patterns.

6. Data Splitting:
- Divide the dataset into training and testing sets to evaluate model performance.
- Consider using techniques like stratified sampling to maintain the fraud-to-non-fraud class balance.

7. Model Training:
- Train the selected models on the training dataset.
- Tune hyperparameters to optimize model performance.

8. Model Evaluation:
- Evaluate model performance using metrics such as accuracy, precision, recall, F1-score, and ROC AUC.
- Assess the model's ability to detect fraudulent transactions while minimizing false positives.

9. Deployment:
- Deploy the trained model to monitor real-time credit card transactions.
- Implement an alert system that triggers when potential fraud is detected.

10. Continuous Monitoring and Improvement:
- Continuously monitor the model's performance.
- Retrain the model periodically to adapt to evolving fraud patterns.

Transaction data simulator

This section presents a transaction data simulator of legitimate and fraudulent transactions. This simulator will be used throughout the rest of this book to motivate and assess the efficiency of different fraud detection techniques in a reproducible way.

A simulation is necessarily an approximation of reality. Compared to the complexity of the dynamics underlying real-world payment card transaction data, the data simulator that we present below follows a simple design.

This simple design is a choice. First, having simple rules to generate transactions and fraudulent behaviors will help in interpreting the kind of patterns that different fraud detection techniques can identify. Second, while simple in its design, the data simulator will generate datasets that are challenging to deal with.

The simulated datasets will highlight most of the issues that practitioners of fraud detection face using real-world data. In particular, they will include class imbalance (less than 1% of fraudulent transactions), a mix of numerical and categorical features (with categorical features involving a very large number of values), non-trivial relationships between features, and time-dependent fraud scenarios.

Design choices

Transaction features

Our focus will be on the most essential features of a transaction. In essence, a payment card transaction consists of any amount paid to a merchant by a customer at a certain time. The six main features that summarise a transaction therefore are:

1. The transaction ID: A unique identifier for the transaction
2. The date and time: Date and time at which the transaction occurs
3. The customer ID: The identifier for the customer. Each customer has a unique identifier
4. The terminal ID: The identifier for the merchant (or more precisely the terminal). Each terminal has a unique identifier
5. The transaction amount: The amount of the transaction.

6. The fraud label: A binary variable, with the value 0 for a legitimate transaction, or the value 1 for a fraudulent transaction.

These features will be referred to as TRANSACTION_ID, TX_DATETIME, CUSTOMER_ID, TERMINAL_ID, TX_AMOUNT, and TX_FRAUD.

The goal of the transaction data simulator will be to generate a table of transactions with these features. This table will be referred toas the labeled transactions table. Such a table is illustrated in Fig. 1.

TRANSACTION_ID	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_FRAUD
0	2018-04-01 00:00:31	596	3156	57.16	0
1	2018-04-01 00:02:10	4961	3412	81.51	0
2	2018-04-01 00:07:56	2	1365	146.00	0
...	...	...	...	...	...

Fig. 1. Example of labeled transaction table. Each transaction is represented as a row in the table, together with its label (TX_FRAUD variable, 0 for legitimate, and 1 for fraudulent transactions).

Contents

Customer profiles generation

Each customer will be defined by the following properties:

- CUSTOMER_ID: The customer unique ID
- (x_customer_id,y_customer_id): A pair of real coordinates (x_customer_id,y_customer_id) in a 100 * 100 grid, that defines the geographical location of the customer
- (mean_amount, std_amount): The mean and standard deviation of the transaction amounts for the customer, assuming that the transaction amounts follow a normal distribution. The mean_amount will be drawn from a uniform distribution (5,100) and the std_amount will be set as the mean_amount divided by two.
- mean_nb_tx_per_day: The average number of transactions per day for the customer, assuming that the number of transactions per day follows a Poisson distribution. This number will be drawn from a uniform distribution (0,4).

The generate_customer_profiles_table function provides an implementation for generating a table of customer profiles. It takes as input the number of customers for which to generate a profile and a random state for reproducibility. It returns a DataFrame containing the properties for each customer.

Transaction generation process

The simulation will consist of five main steps:

1. Generation of customer profiles: Every customer is different in their spending habits. This will be simulated by defining some properties for each customer. The main properties will be their geographical location, their spending frequency, and their spending amounts. The customer properties will be represented as a table, referred to as the customer profile table.
2. Generation of terminal profiles: Terminal properties will simply consist of a geographical location. The terminal properties will be represented as a table, referred to as the terminal profile table.
3. Association of customer profiles to terminals: We will assume that customers only make transactions on terminals that are within a radius of r of their geographical locations. This makes the simple assumption that a customer only makes transactions on terminals that are geographically close to their location. This step will consist of adding a feature 'list_terminals' to each customer profile, that contains the set of terminals that a customer can use.
4. Generation of transactions: The simulator will loop over the set of customer profiles, and generate transactions according to their properties (spending frequencies and amounts, and available terminals). This will result in a table of transactions.
5. Generation of fraud scenarios: This last step will label the transactions as legitimate or genuine. This will be done by followingthree different fraud scenarios.

The transaction generation process is illustrated below.

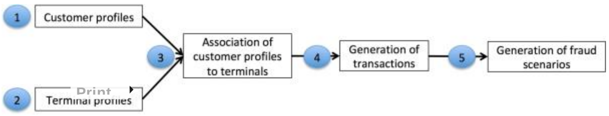


Fig. 2. Transaction generation process. The customer and terminal profiles are used to generate a set of transactions. The final step, which generates fraud scenarios, provides the labeled transactions table.

The following sections detail the implementation for each of these steps.

As an example, let us generate a customer profile table for five customers:

```
n_customers = 5
customer_profiles_table =
```

	CUSTOMER_ID	x_customer_id	y_customer_id	mean_amount	std_amount	mean_nb_tx_per_day
0	0	54.881350	71.518937	62.262521	31.131260	2.179533
1	1	42.365480	64.589411	46.570785	23.285393	3.567092
2	2	96.366276	38.344152	80.213879	40.106939	2.115580
3	3	56.804456	92.559664	11.748426	5.874213	0.348517
4	4	2.021840	83.261985	78.924891	39.462446	3.480049

Terminal profiles generation

Each terminal will be defined by the following properties:

- TERMINAL_ID: The terminal ID
- (x_terminal_id,y_terminal_id): A pair of real coordinates (x_terminal_id,y_terminal_id) in a 100 * 100 grid, that defines thegeographical location of the terminal

The generate_terminal_profiles_table function provides an implementation for generating a table of terminal profiles. It takes as input the number of terminals for which to generate a profile and a random state for reproducibility. It returns a DataFrame containingthe properties for each terminal.

As an example, let us generate a customer terminal table for five terminals:

```
n_terminals = 5
terminal_profiles_table =
```

	TERMINAL_ID	x_terminal_id	y_terminal_id
0	0	54.881350	71.518937
1	1	60.276338	54.488318
2	2	42.365480	64.589411
3	3	43.758721	89.177300
4	4	96.366276	38.344152

Association of customer profiles to terminals

Let us now associate terminals with the customer profiles. In our design, customers can only perform transactions on terminals thatare within a radius of r of their geographical locations.

Let us first write a function, called **get_list_terminals_within_radius**, which finds these terminals for a customer profile. The function will take as input a customer profile (any row in the customer profiles table), an array that contains the geographical locationof all terminals, and the radius **r**. It will return the list of terminals within a radius of **r** for that customer.

As an example, let us get the list of terminals that are within a radius $r = 50$ of the last customer:

```
# We first get the geographical locations of all terminals as a
numpy array
x_y_terminals =
```

The list contains the third and fourth terminals, which are indeed the only ones within a radius of 50 of the last customer.

```
[2, 3]
terminal_profiles_table
```

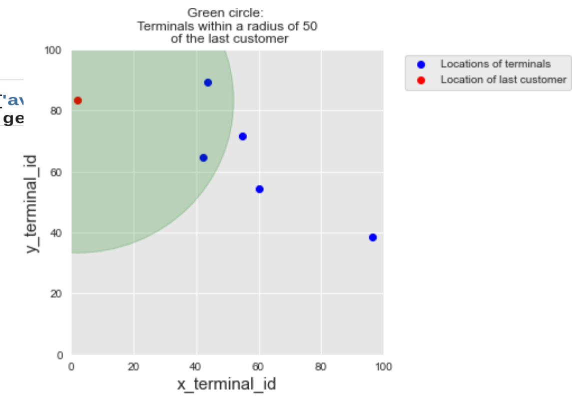
	TERMINAL_ID	x_terminal_id	y_terminal_id
0	0	54.881350	71.518937
1	1	60.276338	54.488318
2	2	42.365480	64.589411
3	3	43.758721	89.177300
4	4	96.366276	38.344152

For better visualization, let us plot

- The locations of all terminals (in red)
- The location of the last customer (in blue)
- The region within radius of 50 of the first customer (in green)

```
terminals_available_to_customer_fig
```

```
customer_profiles_table['av
s_table.apply(lambda x : ge
```



Computing the list of available terminals for each customer is then straightforward, using the panda **apply** function. We store theresults as a new column **available_terminals** in the customer profiles table.

	CUSTOMER_ID	x_customer_id	y_customer_id	mean_amount	std_amount	mean_nb_tx_per_day	available_terminals
0	0	54.881350	71.518937	62.262521	31.131260	2.179533	[0, 1, 2, 3]
1	1	42.365480	64.589411	46.570785	23.285393	3.567092	[0, 1, 2, 3]
2	2	96.366276	38.344152	80.213879	40.106939	2.115580	[1, 4]
3	3	56.804456	92.559664	11.748426	5.874213	0.348517	[0, 1, 2, 3]
4	4	2.021840	83.261985	78.924891	39.462446	3.480049	[2, 3]

It is worth noting that the radius r controls the number of terminals that will be on average available for each customer. As the numberof terminals is increased, this radius should be adapted to match the average number of available terminals per customer that isdesired in a simulation.

Generation of transactions

The customer profiles now contain all the information that we require to generate transactions. The transaction generation will be done by a function **generate_transactions_table** that takes as input a customer profile, a starting date, and a number of days for which to generate transactions. It will return a table of transactions, which follows the format presented above (without the transaction label, which will be added in [fraud scenarios generation]).

Let us for example generate transactions for the first customer, for five days, starting at the date 2018-04-01:

```
transaction_table_customer_0=generate_transactions_table(custom
er_profiles_table.iloc[0],
                                                    start_date
```

	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_TIME_SECONDS	TX_TIME_DAYS
0	2018-04-01 07:19:05	0	3	123.59	26345	0
1	2018-04-01 19:02:02	0	3	46.51	68522	0
2	2018-04-01 18:00:16	0	0	77.34	64816	0
3	2018-04-02 15:13:02	0	2	32.35	141182	1
4	2018-04-02 14:05:38	0	3	63.30	137138	1
5	2018-04-02 15:46:51	0	3	13.59	143211	1
6	2018-04-02 08:51:06	0	2	54.72	118266	1
7	2018-04-02 20:24:47	0	3	51.89	159887	1
8	2018-04-03 12:15:47	0	2	117.91	216947	2
9	2018-04-03 08:50:09	0	1	67.72	204609	2
10	2018-04-03 09:25:49	0	1	28.46	206749	2
11	2018-04-03 15:33:14	0	2	50.25	228794	2
12	2018-04-03 07:41:24	0	1	93.26	200484	2
13	2018-04-04 01:15:35	0	0	46.40	263735	3
14	2018-04-04 09:33:58	0	2	23.26	293638	3
15	2018-04-05 16:19:09	0	1	71.96	404349	4
16	2018-04-05 07:41:19	0	2	52.69	373279	4

We can make a quick check that the generated transactions follow the customer profile properties:

- The terminal IDs are indeed those in the list of available terminals (0, 1, 2 and 3)
- The transaction amounts seem to follow the amount parameters of the customer (mean_amount=62.26 and std_amount=31.13)
- The number of transactions per day varies according to the transaction frequency parameters of the customer (mean_nb_tx_per_day=2.18).

Let us now generate the transactions for all customers. This is straightforward using the pandas groupby and apply methods:

```
transactions_df=customer_profiles_table.groupby('CUSTOMER_ID').
apply(lambda x : generate_transactions_table(x.iloc[0],
```

	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_TIME_SECONDS	TX_TIME_DAYS
0	2018-04-01 07:19:05	0	3	123.59	26345	0
1	2018-04-01 19:02:02	0	3	46.51	68522	0
2	2018-04-01 18:00:16	0	0	77.34	64816	0
3	2018-04-02 15:13:02	0	2	32.35	141182	1
4	2018-04-02 14:05:38	0	3	63.30	137138	1
...	...	...	...	...	...	...
60	2018-04-05 07:41:19	4	2	111.38	373279	4
61	2018-04-05 06:59:59	4	3	80.36	370799	4
62	2018-04-05 17:23:34	4	2	53.25	408214	4
63	2018-04-05 12:51:38	4	2	36.44	391898	4
64	2018-04-05 12:38:46	4	3	17.53	391126	4

65 rows × 6 columns

This gives us a set of 65 transactions, with 5 customers, 5 terminals, and 5 days.

Scaling up to a larger dataset

We now have all the building blocks to generate a larger dataset. Let us write a generate_dataset function, that will take care ofrunning all the previous steps. It will

- take as inputs the number of desired customers, terminals and days, as well as the starting date and the radius r
- return the generated customer and terminal profiles table, and the DataFrame of transactions.

> **Note**
>
> In order to speed up the computations, one can use the parallel_apply function of the pandarallel module. This functionreplaces the panda apply function, and allows the distribution of the computation on all the available CPUs.

Let us generate a dataset that features

- 5000 customers
- 10000 terminals
- 183 days of transactions (which corresponds to a simulated period from 2018/04/01 to 2018/09/30)

The starting date is arbitrarily fixed at 2018/04/01. The radius r is set to 5, which corresponds to around 100 available terminals foreach customer.

It takes around 3 minutes to generate this dataset on a standard laptop.

```
(customer_profiles_table,
     terminal_profiles_table,
     transactions_df)=\
     generate_dataset(n_customers =
```

A total of 1754155 transactions were generated.

```
transactions_df.shape
```

```
(1754155, 7)
```

Note that this number is low compared to real-world fraud detection systems, where millions of transactions may need to be processed every day. This will however be enough for the purpose of this book, in particular to keep reasonable executions times.

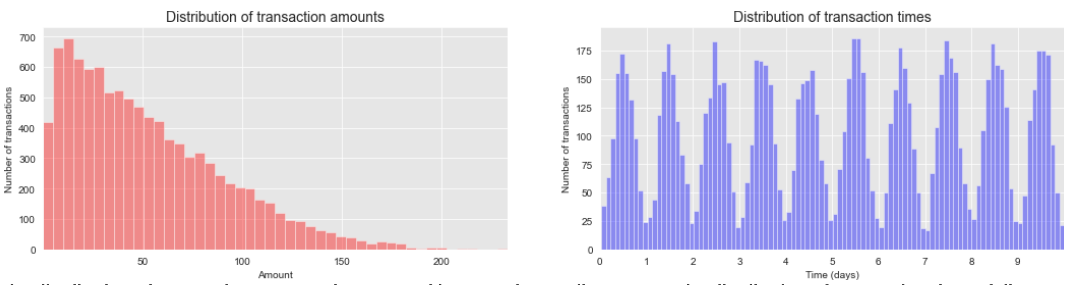
```
transactions_df
```

	TRANSACTION_ID	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_TIME_SECONDS	TX_TIME_DAYS
0	0	2018-04-01 00:00:31	596	3156	57.16	31	0
1	1	2018-04-01 00:02:10	4961	3412	81.51	130	0
2	2	2018-04-01 00:07:56	2	1365	146.00	476	0
3	3	2018-04-01 00:09:29	4128	8737	64.49	569	0
4	4	2018-04-01 00:10:34	927	9906	50.99	634	0
...	...	...	...	...	...	...	...
1754150	1754150	2018-09-30 23:56:36	161	655	54.24	15810996	182
1754151	1754151	2018-09-30 23:57:38	4342	6181	1.23	15811058	182
1754152	1754152	2018-09-30 23:58:21	618	1502	6.62	15811101	182
1754153	1754153	2018-09-30 23:59:52	4056	3067	55.40	15811192	182
1754154	1754154	2018-09-30 23:59:57	3542	9849	23.59	15811197	182

1754155 rows × 7 columns

As a sanity check, let us plot the distribution of transaction amounts and transaction times.

```
distribution_amount_times_fig
```



The distribution of transaction amounts has most of its mass for small amounts. The distribution of transaction times follows a gaussian distribution on a daily basis, centered around noon. These two distributions are in accordance with the simulation parameters used in the previous sections.

Fraud scenarios generation

This last step of the simulation adds fraudulent transactions to the dataset, using the following fraud scenarios:

- Scenario 1: Any transaction whose amount is more than 220 is a fraud. This scenario is not inspired by a real-world scenario. Rather, it will provide an obvious fraud pattern that should be detected by any baseline fraud detector. This will be useful tovalidate the implementation of a fraud detection technique.

- Scenario 2: Every day, a list of two terminals is drawn at random. All transactions on these terminals in the next 28 days will be marked as fraudulent. This scenario simulates a criminal use of a terminal, through phishing for example. Detecting this scenario will be possible by adding features that keep track of the number of fraudulent transactions on the terminal. Since the terminal is only compromised for 28 days, additional strategies that involve concept drift will need to be designed to efficiently deal with this scenario.
- Scenario 3: Every day, a list of 3 customers is drawn at random. In the next 14 days, 1/3 of their transactions have their amounts multiplied by 5 and marked as fraudulent. This scenario simulates a card-not-present fraud where the credentials of a customer have been leaked. The customer continues to make transactions, and transactions of higher values are made by the fraudster who tries to maximize their gains. Detecting this scenario will require adding features that keep track of the spending habits ofthe customer. As for scenario 2, since the card is only temporarily compromised, additional strategies that involve concept drift should also be designed.

Let us add fraudulent transactions using these scenarios:

```
%time  transactions_df  =  add_frauds(customer_profiles_table,
```

```
N
u
m
```

Percentage of fraudulent transactions:

```
transactions_df.TX_FRAUD.mean()
```

```
0.008369271814634397
```

Number of fraudulent transactions:

```
transactions_df.TX_FRAUD.sum()
```

```
14681
```

A total of 14681 transactions were marked as fraudulent. This amounts to 0.8% of the transactions. Note that the sum of the fraudsfor each scenario does not equal the total amount of fraudulent transactions. This is because the same transactions may have been marked as fraudulent by two or more fraud scenarios.

Our simulated transaction dataset is now complete, with a fraudulent label added to all transactions.

```
transactions_df.head()
```

	TRANSACTION_ID	TX_DATETIME	CUSTOMER_ID	TERMINAL_ID	TX_AMOUNT	TX_TIME_SECONDS	TX_TIME_DAYS	TX_FRAUD	TX_FRAUD_SCENARIO
0	0	2018-04-01 00:00:31	596	3156	57.16	31	0	0	0
1	1	2018-04-01 00:02:10	4961	3412	81.51	130	0	0	0
2	2	2018-04-01 00:07:56	2	1365	146.00	476	0	0	0
3	3	2018-04-01 00:09:29	4128	8737	64.49	569	0	0	0
4	4	2018-04-01 00:10:34	927	9906	50.99	634	0	0	0

```
transactions_df[transactions_df.TX_FRAUD_SCENARIO==1].shape
```

```
(973, 9)
```

```
transactions_df[transactions_df.TX_FRAUD_SCENARIO==2].shape
```

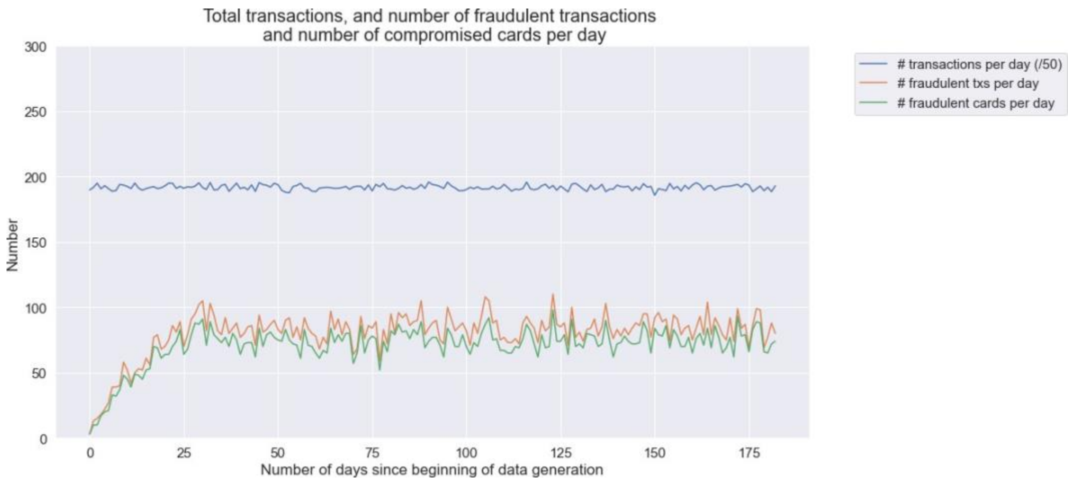
```
(9077, 9)
```

```
transactions_df[transactions_df.TX_FRAUD_SCENARIO==3].shape
```

```
(4631, 9)
```

Let us check how the number of transactions, the number of fraudulent transactions, and the number of compromised cards vary on adaily basis.

```
fraud_and_transactions_stats_fig
```



This simulation generated around 10000 transactions per day. The number of fraudulent transactions per day is around 85, and the number of fraudulent cards around 80. It is worth noting that the first month has a lower number of fraudulent transactions, which isdue to the fact that frauds from scenarios 2 and 3 span periods of 28 and 14 days, respectively.

The resulting dataset is interesting: It features class imbalance (less than 1% of fraudulent transactions), a mix of numerical and categorical features, non-trivial relationships between features, and time-dependent fraud scenarios.

Let us finally save the dataset for reuse in the rest of this book.

Saving of dataset

Instead of saving the whole transaction dataset, we split the dataset into daily batches. This will allow later the loading of specificperiods instead of the whole dataset. The pickle format is used, rather than CSV, to speed up the loading times. All files are saved inthe `DIR_OUTPUT` folder. The names of the files are the dates, with the `.pkl` extension.

```
DIR_OUTPUT = "./simulated-data-raw/"
if not
i
    f
for in
    n
    o
    t
    o
```