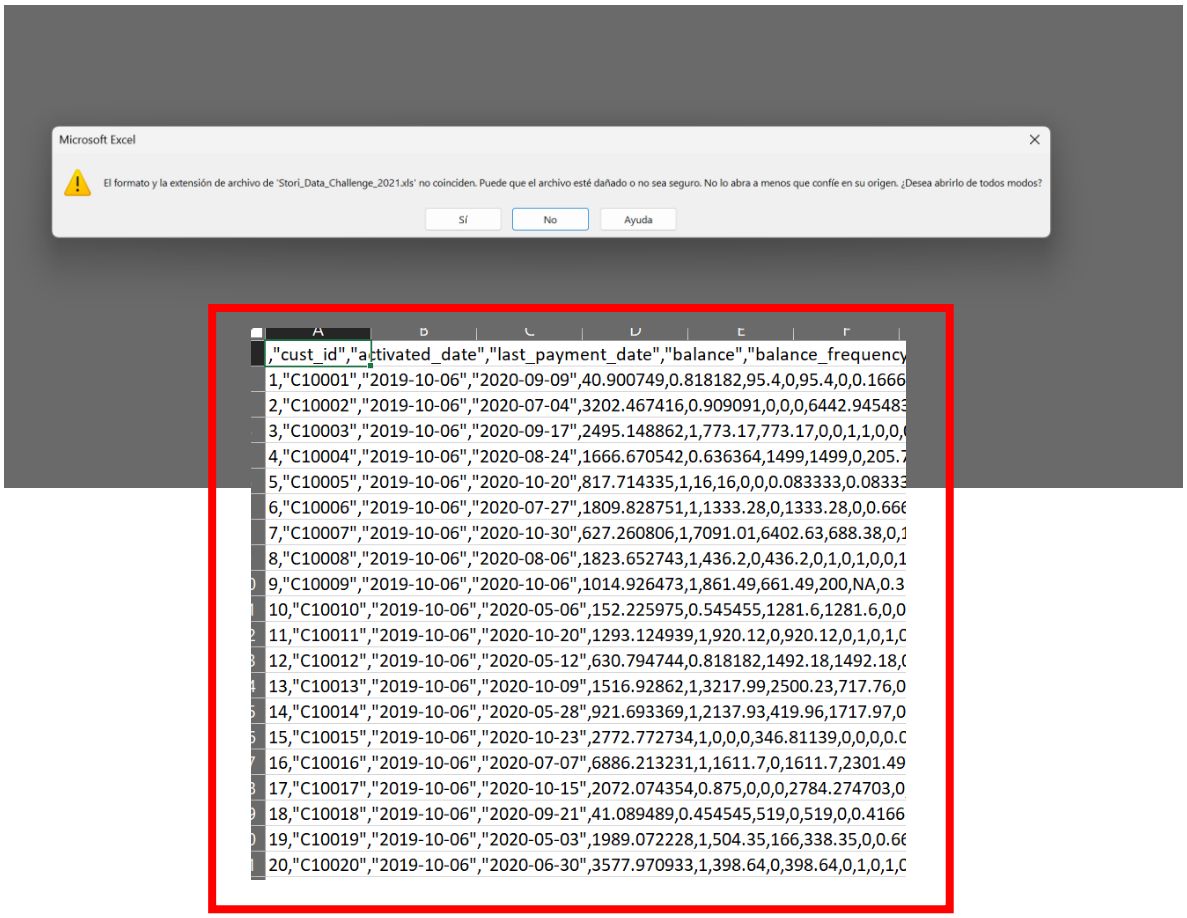
Desarrollo

This problem consists of several characteristics to address given the questions raised in the subsequent points. In the first instance, when evaluating the provided documents, it is possible to appreciate that the format of the file does not match its name. Upon opening the Excel file containing the information, it generates a data corruption error. However, upon observing the composition of the dataset to work with, it is clearly seen that the data is separated by commas (Illustration 2). Therefore, instead of working on a method to separate each data into different columns, it is only necessary to change the file extension and it will automatically initialize with the corresponding format.



El formato y la extensión de archivo de 'Stori_Data_Challenge_2021.xls' no coinciden. Puede que el archivo esté dañado o no sea seguro. No lo abra a menos que confíe en su origen. ¿Desea abrirlo de todos modos?

```
,"cust_id","activated_date","last_payment_date","balance","balance_frequency
1,"C10001","2019-10-06","2020-09-09",40.900749,0.818182,95.4,0,95.4,0,0.1666
2,"C10002","2019-10-06","2020-07-04",3202.467416,0.909091,0,0,0,6442.94548
3,"C10003","2019-10-06","2020-09-17",2495.148862,1,773.17,773.17,0,0,1,1,0,0,
4,"C10004","2019-10-06","2020-08-24",1666.670542,0.636364,1499,1499,0,205.
5,"C10005","2019-10-06","2020-10-20",817.714335,1,16,16,0,0,0.083333,0.0833
6,"C10006","2019-10-06","2020-07-27",1809.828751,1,1333.28,0,1333.28,0,0.66
7,"C10007","2019-10-06","2020-10-30",627.260806,1,7091.01,6402.63,688.38,0,
8,"C10008","2019-10-06","2020-08-06",1823.652743,1,436.2,0,436.2,0,1,0,1,0,0,
9,"C10009","2019-10-06","2020-10-06",1014.926473,1,861.49,661.49,200,NA,0.3
10,"C10010","2019-10-06","2020-05-06",152.225975,0.545455,1281.6,1281.6,0,0
11,"C10011","2019-10-06","2020-10-20",1293.124939,1,920.12,0,920.12,0,1,0,1,
12,"C10012","2019-10-06","2020-05-12",630.794744,0.818182,1492.18,1492.18,
13,"C10013","2019-10-06","2020-10-09",1516.92862,1,3217.99,2500.23,717.76,0
14,"C10014","2019-10-06","2020-05-28",921.693369,1,2137.93,419.96,1717.97,0
15,"C10015","2019-10-06","2020-10-23",2772.772734,1,0,0,0,346.81139,0,0,0,0.
16,"C10016","2019-10-06","2020-07-07",6886.213231,1,1611.7,0,1611.7,2301.49
17,"C10017","2019-10-06","2020-10-15",2072.074354,0.875,0,0,0,2784.274703,0
18,"C10018","2019-10-06","2020-09-21",41.089489,0.454545,519,0,519,0,0.4166
19,"C10019","2019-10-06","2020-05-03",1989.072228,1,504.35,166,338.35,0,0.66
20,"C10020","2019-10-06","2020-06-30",3577.970933,1,398.64,0,398.64,0,1,0,1,
```

Ilustración 1 - Corrupted file

For this case study, Python was used for the processing and analysis of the data contained in the df.csv file using the Pandas and Matplotlib libraries.

In this case, Pandas offers higher-level access to its data in the form of hash-based tabular objects. Unlike an array, you can access the data directly by passing a column name and an index.

In short: if you want to perform efficient mathematical operations using arrays, Numpy would be a better option. For tables and general data analysis, we chose Pandas.

Pandas automates tasks that are time-consuming and repetitive. Some uses of Pandas include:

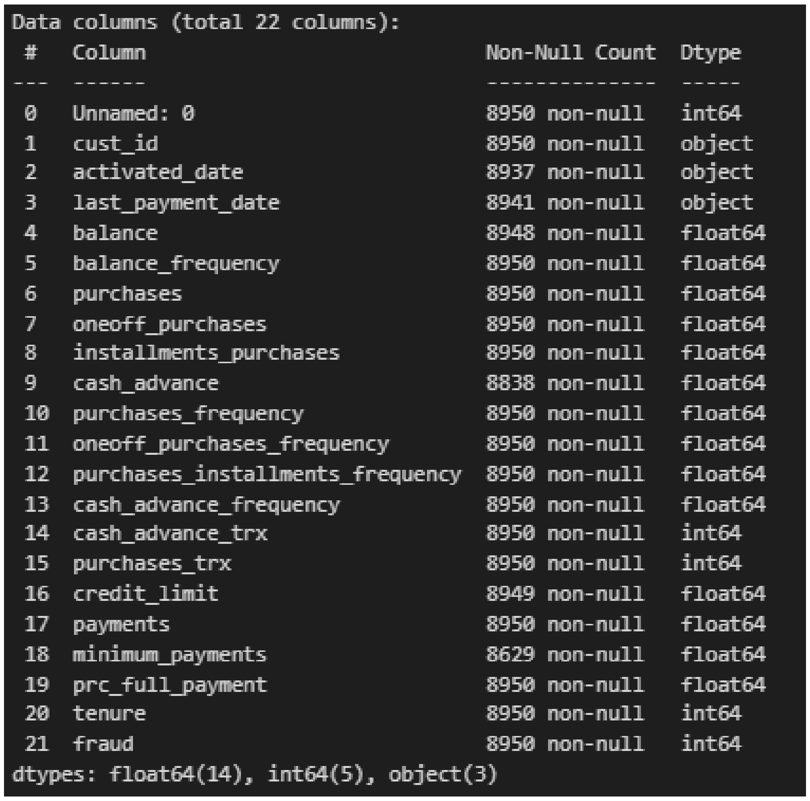
1. Data cleaning: systematically clean dirty data
2. Loading and saving data: easily import data from an external source and export it to your local computer
3. Data imputation: systematically impute missing data

4. Statistical analysis: perform statistical analysis on data sets easily

As a first step, we can use Pandas to clean our data and prepare it for analysis. However, there are different types of problems commonly encountered in any data set:

1. Missing data
2. Duplicate data
3. Incorrect data

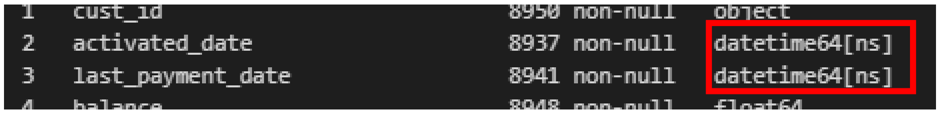
To address these data problems, we can use built-in Pandas functions to efficiently clean them. Before starting the cleaning process, we can use the `info()` function to give us a snapshot of the data types in our DataFrame (Illustration 3) and systematically change the data to the appropriate data types.



```
Data columns (total 22 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Unnamed: 0                        8950 non-null   int64  
 1   cust_id                           8950 non-null   object 
 2   activated_date                    8937 non-null   object 
 3   last_payment_date                 8941 non-null   object 
 4   balance                           8948 non-null   float64
 5   balance_frequency                 8950 non-null   float64
 6   purchases                         8950 non-null   float64
 7   oneoff_purchases                  8950 non-null   float64
 8   installments_purchases            8950 non-null   float64
 9   cash_advance                      8838 non-null   float64
 10  purchases_frequency               8950 non-null   float64
 11  oneoff_purchases_frequency        8950 non-null   float64
 12  purchases_installments_frequency  8950 non-null   float64
 13  cash_advance_frequency            8950 non-null   float64
 14  cash_advance_trx                  8950 non-null   int64  
 15  purchases_trx                     8950 non-null   int64  
 16  credit_limit                      8949 non-null   float64
 17  payments                          8950 non-null   float64
 18  minimum_payments                  8629 non-null   float64
 19  prc_full_payment                  8950 non-null   float64
 20  tenure                            8950 non-null   int64  
 21  fraud                             8950 non-null   int64  
dtypes: float64(14), int64(5), object(3)
```

Ilustración 2 - Command "Info()"

First, it's worth noting that the columns 'activated_date' and 'last_paid_date' are currently labeled as type 'object', which is not optimal for representing dates. It would be appropriate to use the 'datetime' data type for these columns. To achieve this correction, we can apply the 'to_datetime()' function to the DataFrame. After this modification, upon rechecking the data type of the columns, we should notice the change reflected clearly and accurately.

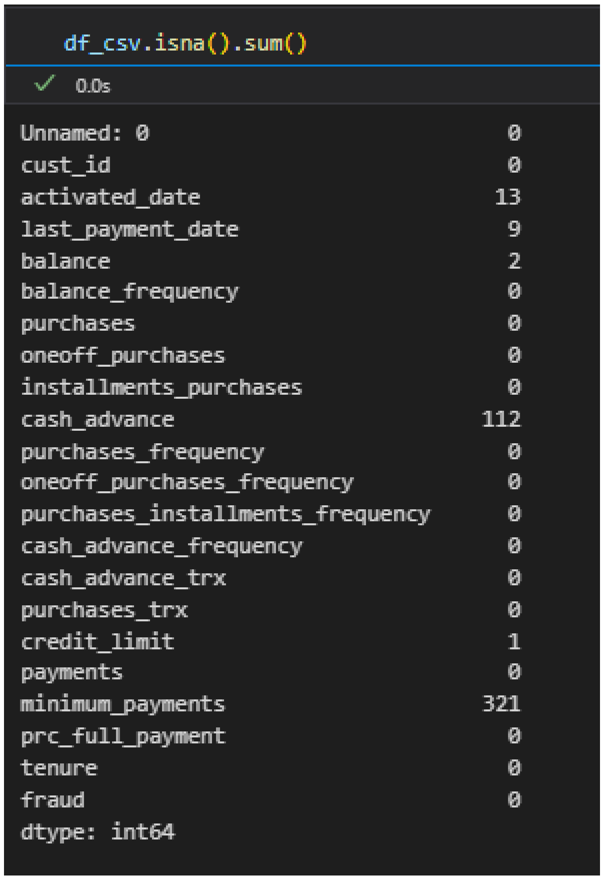


```
 1   cust_id                           8950 non-null   object
 2   activated_date                    8937 non-null   datetime64[ns]
 3   last_payment_date                 8941 non-null   datetime64[ns]
 4   balance                           8948 non-null   float64
```

Ilustración 3 - Command "to_datetime()"

Missing Data.

In the analysis conducted through the `info()` function, the presence of null values in our data has been identified, as shown in Illustration 3 when observing the "Non-Null Count" column. It is important to note that some columns have a different non-null count, indicating variability in the quantity of null values among them. To gain a more detailed insight into which columns and rows contain null values, we can employ the `isna()` function, although it is important to consider that the direct interpretation of these results can be complex. One strategy to simplify this information is to combine `isna()` with the `sum()` function, allowing us to more clearly visualize the number of null values for each column.



```
df_csv.isna().sum()
```

```
✓ 0.0s
```

```
Unnamed: 0                            0
cust_id                               0
activated_date                       13
last_payment_date                     9
balance                               2
balance_frequency                     0
purchases                             0
oneoff_purchases                      0
installments_purchases                0
cash_advance                        112
purchases_frequency                   0
oneoff_purchases_frequency            0
purchases_installments_frequency      0
cash_advance_frequency                0
cash_advance_trx                      0
purchases_trx                         0
credit_limit                          1
payments                              0
minimum_payments                    321
prc_full_payment                      0
tenure                                0
fraud                                 0
dtype: int64
```

Ilustración 4 - Command "isna().sum() "

In this type of scenario, we can handle missing data through two commonly used methods:

1. Removing null values from rows and/or columns of the DataFrame.
2. Filling null values with a constant or other values from the DataFrame.

However, completely eliminating rows and columns with null values is often an ineffective method for data cleaning, as it can result in the loss of significant data. A more common approach to clean these null values is to fill them using the `fillna()` function. Given that the missing data for the particular case of this dataset does not represent a significant portion, we will use `fillna(0)` to fill the missing values with the value of zero. By filling missing values with zeros, you maintain consistency in your dataset. This can be important, especially if you are performing numerical operations later on, as it will prevent errors or unexpected results due to missing values.

Histograms

This type of graph shows the frequency of occurrence of a particular value in our dataset. It represents the frequency of each value, allowing us to visualize the distribution of those values. In question 1.1, we will assess the balances of all clients.

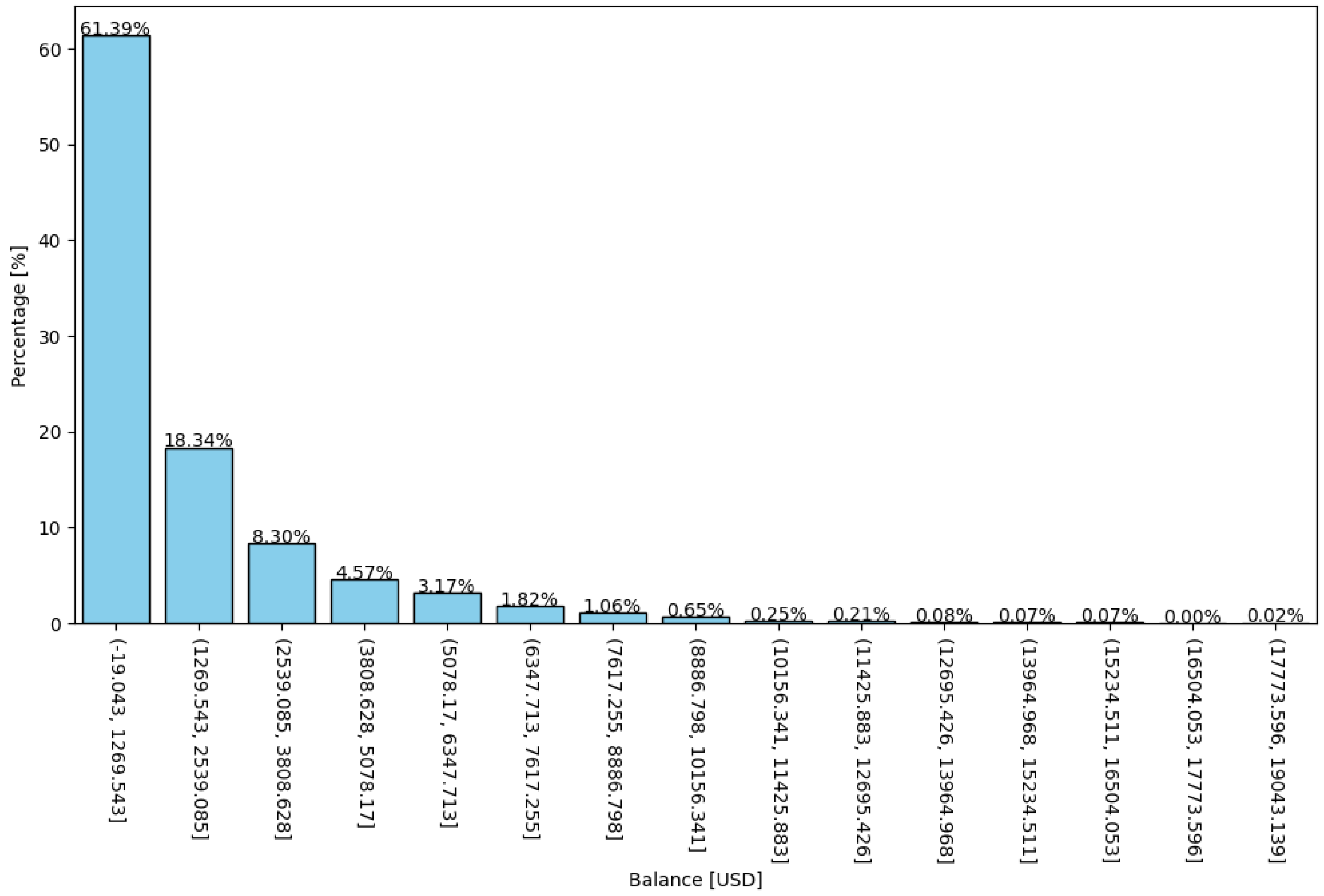


Ilustración 5 - Balance amount for all the customers (Percentage)

Distribution of client balances:

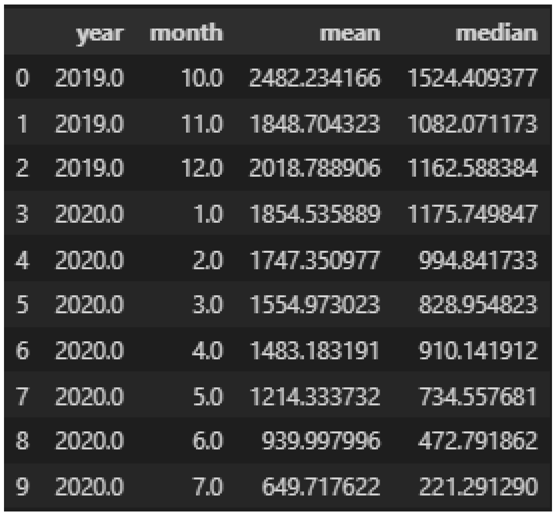
1. The majority of clients (61.39%) maintain balances between $0 and $1269.543. This range suggests a favorable financial situation for most, indicating a considerable spending capacity. In financial terms, these users exhibit a high ability to make additional purchases.

2. On the other hand, there is a smaller proportion of clients with lower balances, indicating a minority with more limited financial resources. Lower balances imply reduced availability of funds in their accounts, potentially limiting their capacity to make larger purchases or transactions.

In summary, the distribution of balances suggests that most clients enjoy a solid economic situation, although a smaller fraction has lower balances, indicating a more restricted spending capacity for this group.

Mean and median balance report, grouped by year and month of activation date.

To carry out this procedure, we create two new columns: Year and Month, allowing us to perform the appropriate data grouping. Since we previously converted the data type to its corresponding format, there should be no inconvenience in creating the columns and performing the grouping according to the requested characteristics. Thus, upon executing the code, we obtain the following results (Illustration 7). It is worth noting that the treatment of null data has already been performed, thus posing no issue for analysis.



	year	month	mean	median
0	2019.0	10.0	2482.234166	1524.409377
1	2019.0	11.0	1848.704323	1082.071173
2	2019.0	12.0	2018.788906	1162.588384
3	2020.0	1.0	1854.535889	1175.749847
4	2020.0	2.0	1747.350977	994.841733
5	2020.0	3.0	1554.973023	828.954823
6	2020.0	4.0	1483.183191	910.141912
7	2020.0	5.0	1214.333732	734.557681
8	2020.0	6.0	939.997996	472.791862
9	2020.0	7.0	649.717622	221.291290

Ilustración 6 - Mean and median balance, grouped by year and month of activated date.

According to the information, the following analysis can be conducted:

1. General Trend:
   - Both the mean and median balances show a decreasing trend over time, from October 2019 to July 2020.
2. 2019 Analysis:
   - In October 2019, the highest values are observed, with a mean balance of 2482.23 and a median balance of 1524.41.
   - In November 2019, both values decrease, with a mean balance of 1848.70 and a median balance of 1082.07.
   - In December 2019, the mean balance slightly increases to 2018.79, but the median balance decreases to 1162.59.
3. 2020 Analysis:
   - In January 2020, the mean balance decreases to 1854.54, but the median balance increases to 1175.75.
   - From February 2020 onwards, both the mean and median balances decrease steadily month by month.
   - In July 2020, the lowest values are reached, with a mean balance of 649.72 and a median balance of 221.29.
4. Possible Causes and Actions:
   - The steady decrease in mean and median balances could indicate a slowdown in economic activity or a reduced influx of funds into the analyzed business or sector.
   - It is advisable to investigate the underlying causes of this decreasing trend, such as changes in economic conditions, competition, marketing strategies, among other factors.
   - Depending on the identified causes, corrective actions can be taken, such as adjustments in sales strategies, cost optimization, improvements in offered services or products, among other actions, to reverse the negative trend.

Report on Activated Clients and Last Payment in 2020.

In this section, a detailed table is presented with information about customers who activated their accounts and made their last payment during the year 2020. The table includes the following fields:

1. Cust ID: Customer identification (excluding letters).
2. Activation date: Date when the account was activated (format YYYY-MM).
3. Last payment date: Date of the customer's last payment (format YYYY-MM-DD).
4. Cash advance: Amount of cash advance.
5. Credit limit: Credit limit granted to the customer.
6. Cash advance (%): Percentage of cash advance relative to the credit limit.

The approach taken to address this report using Python was through the following code:

1. First, we created a filter for all customers who activated their account and made their last payment during 2020.
   - `filtered_df = df_csv[(df_csv['activated_date'].dt.year == 2020) & (df_csv['last_payment_date'].dt.year == 2020)].copy()`

2. To obtain only the numbers from the 'cust_id' column, we use the `str.extract('(\\d+)')` function on the 'filtered_df' DataFrame. This function searches for matches of one or more digits (\d+) in each value of the 'cust_id' column using a regular expression. The parentheses around \d+ indicate that only the digits found in each value will be extracted. Double slashes \\ are used to escape the backslash, as it is part of the regular expression syntax in Python.

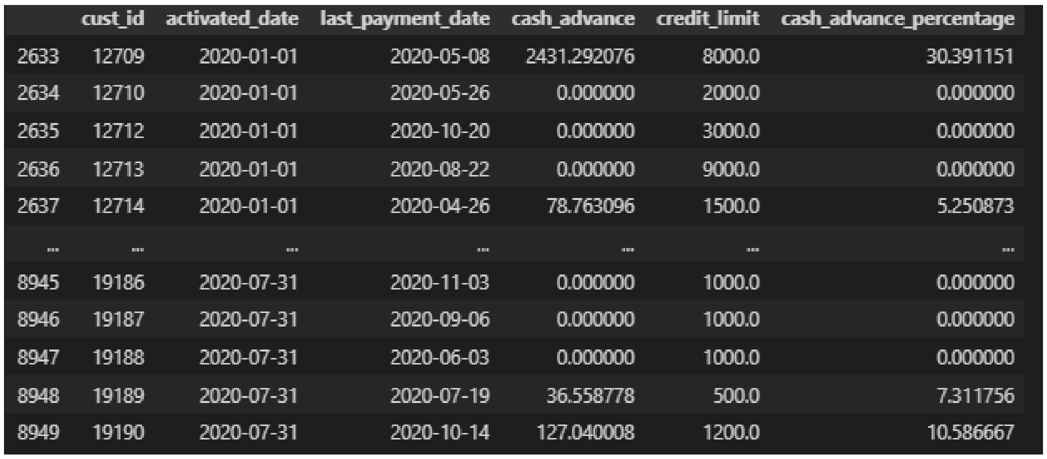
   - `filtered_df['cust_id'] = filtered_df['cust_id'].str.extract('(\\d+)')`

3. We then calculate Cash Advance as a percentage of the credit limit:

   - `filtered_df['cash_advance_percentage'] = (filtered_df['cash_advance'] / filtered_df['credit_limit']) * 100`

4. Finally, we select the required columns and generate a new report with the filtered information.

   - `report_df = filtered_df[['cust_id', 'activated_date', 'last_payment_date', 'cash_advance', 'credit_limit', 'cash_advance_percentage']]`
   - `report_df.to_excel('report_customers_2020.xlsx', index=False)`



```
      cust_id activated_date last_payment_date cash_advance credit_limit cash_advance_percentage
2633   12709     2020-01-01        2020-05-08  2431.292076       8000.0               30.391151
2634   12710     2020-01-01        2020-05-26     0.000000       2000.0                0.000000
2635   12712     2020-01-01        2020-10-20     0.000000       3000.0                0.000000
2636   12713     2020-01-01        2020-08-22     0.000000       9000.0                0.000000
2637   12714     2020-01-01        2020-04-26    78.763096       1500.0                5.250873
 ...     ...            ...               ...          ...          ...                     ...
8945   19186     2020-07-31        2020-11-03     0.000000       1000.0                0.000000
8946   19187     2020-07-31        2020-09-06     0.000000       1000.0                0.000000
8947   19188     2020-07-31        2020-06-03     0.000000       1000.0                0.000000
8948   19189     2020-07-31        2020-07-19    36.558778        500.0                7.311756
8949   19190     2020-07-31        2020-10-14   127.040008       1200.0               10.586667
```

Ilustración 7 - Report customers who activated their account and made their last payment during 2020

Development of a Fraud Predictive Model

In the field of fraud analysis, Python offers several tools for building predictive models. Among these tools, the Scikit-learn library stands out as a reliable and widely used option for implementing supervised and unsupervised learning algorithms.

Scikit-learn provides a variety of effective algorithms, such as Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM), which can be employed for fraud detection. The choice of the most suitable algorithm depends on factors such as the nature and size of the dataset, as well as the complexity of the fraud detection problem.

This report examines the process of implementing a fraud predictive model using Scikit-learn, exploring the different algorithm options available and considering the specific characteristics of the data and the problem to be solved.

Classification Models for Predictive Analysis:

1. **Logistic Regression**: It serves as a solid starting point due to its simplicity and interpretability. It performs effectively when there is a linear relationship between features and the target variable.

2. **Decision Trees**: They stand out for their intuitiveness and ability to handle non-linear features effortlessly. Moreover, they provide valuable insights into the decision-making process in classification.

3. **Random Forests**: They represent an evolution of decision trees by combining multiple trees to enhance accuracy. They are particularly robust against overfitting and are well-suited for extensive datasets.

4. **Support Vector Machines (SVM)**: Recommended when there is a clear separation between data classes. They are particularly useful in high-dimensional datasets.

5. **Anomaly Detection**: Alongside conventional classification algorithms, anomaly detection methods such as Isolation Forest or One-Class SVM can be considered. These are especially effective when the dataset is dominated by normal instances with a few anomalies, such as fraud.

Model Comparison for Prediction through Confusion Matrix.

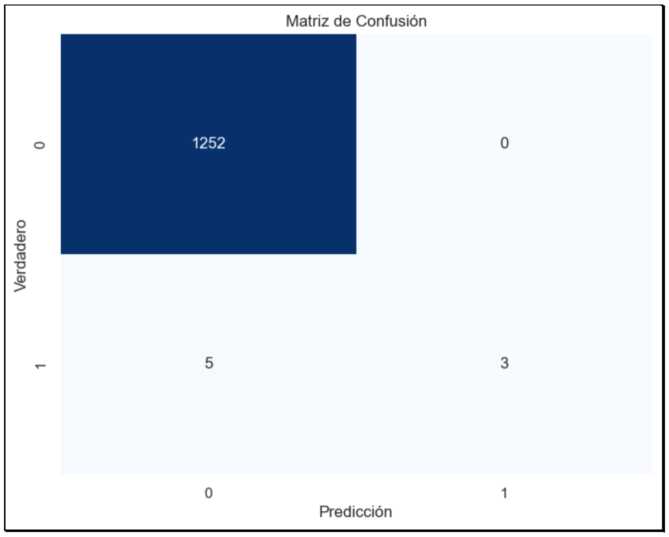


Ilustración 8 - Logistic regression - Accuracy: 0.9960

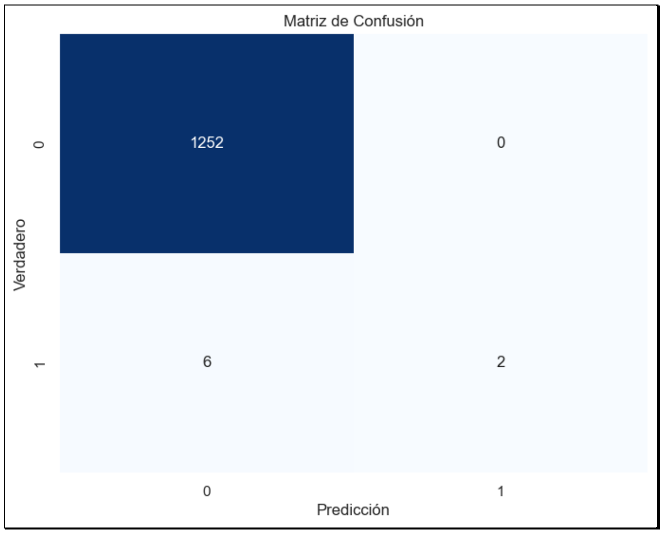


Ilustración 10 - Random Forests - Accuracy: 0.9952

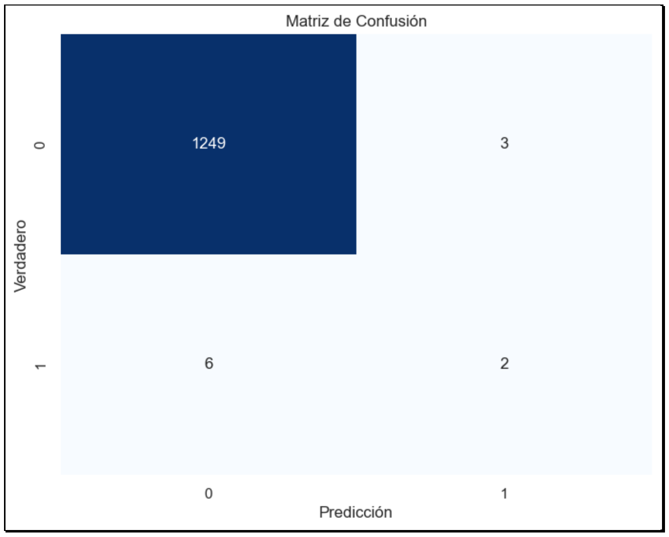


Ilustración 9 - Decision Trees - Accuracy: 0.9928

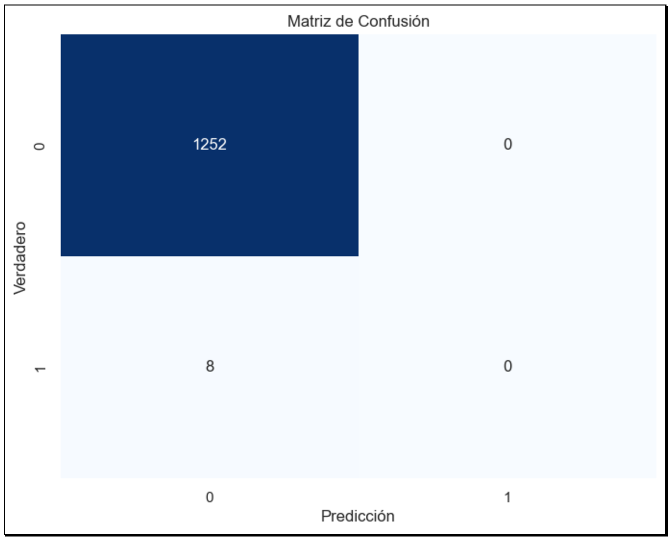


Ilustración 11 - Support Vector Machines (SVM) - Accuracy: 0.9936

The precision, defined as the proportion of transactions correctly classified by the model, is exceptionally high in all cases, exceeding 99%. This figure reflects the models' ability to accurately identify both fraudulent and legitimate transactions, providing robust confidence in their predictive and detection capability.

Modelo	Precision	Recall	F1-score
Regresión Logística	1.00	0.38	0.55
Árboles de Decisión	1.00	0.25	0.31
Bosques Aleatorios	1.00	0.25	0.40
SVM	0.99	1.00	1.00

Clarifications

- **Accuracy (Precision)**: Represents the proportion of transactions marked as fraud that were actually fraudulent.
- **Recall (Sensitivity)**: Represents the proportion of actual fraudulent transactions that were identified by the model.
- **F1-score**: It is a harmonic mean between precision and recall, serving as an indicator of the overall performance of the model.

Analysis by Model

1. **Logistic Regression**: Achieves a perfect precision (1.00) in correctly identifying all legitimate transactions. However, its performance in detecting fraudulent transactions is poor (recall of 0.38), meaning it lets through a significant amount of fraud.

2. **Decision Trees and Random Forests**: Exhibit similar behavior to logistic regression, with perfect precision but low recall (0.25) in fraud detection.

3. **SVM**: Attains the highest precision in detecting fraudulent transactions (recall of 1.00), identifying all fraudulent transactions. However, its slightly lower precision (0.99) compared to logistic regression in classifying legitimate transactions is notable.

While all models show high precision in identifying legitimate transactions, their performance in fraud detection is unsatisfactory. The first three models (logistic regression, decision trees, and random forests) overlook a significant number of fraudulent transactions, resulting in a high false negative rate.

In this context, the SVM model stands out as the best option, as it correctly identifies all fraudulent transactions (maximum recall). However, its inability to identify legitimate transactions makes it ineffective for its primary purpose of fraud prevention.