Tecnológico de Monterrey

# Evidence 4

Luis Gabriel Delfín Paulín

A01701482

23/05/2024

## Description

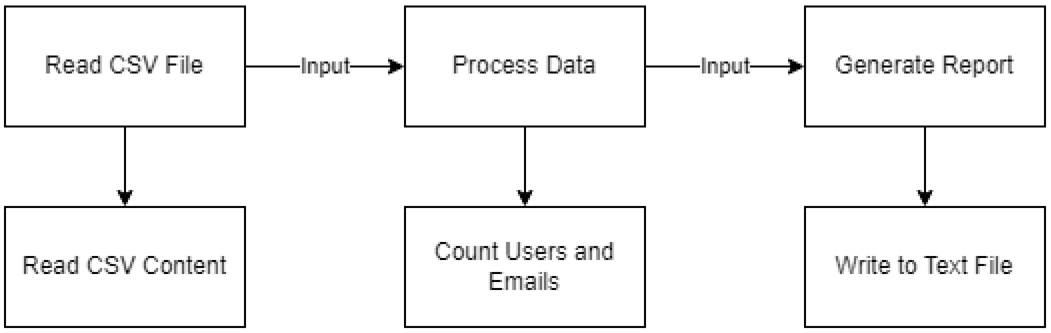
Many organizations store user data in CSV files. This program allows them to quickly extract relevant information from these files to perform a basic analysis, such as counting users and analyzing email domains (this could improve depending on the data the orgnization want).

It is useful for:

- Extraction of data from CSV files.
- Data processing by counting the number of users and the distribution of email domains.
- Generated reports
- Automation of data extraction and analysis

## Models

This is the model we used for the programming paradigm.



- Read CSV File: The program starts by reading the content of a CSV file containing user information and parsing it by using "read_and_extract_data()".
- Process Data: The program counts the number of users and emails domains by using "process_data()".
- Generate Report: The program generates a report summarizing the information by using "generate_report()".

## 2 Codes Usage

For this evidence, we will use 2 different codes. Both have the same objective of generating a report and analyzing the information from the CSV files, but in one “ProgrammingParadigm1” we use test cases inside the code just to make it easier to check if it works, so we don’t need additional files to run this code. The other one “ProgrammingParadigm2” needs a file to run, which is more realistic, so if you have a big file with usernames and emails, you can use it.

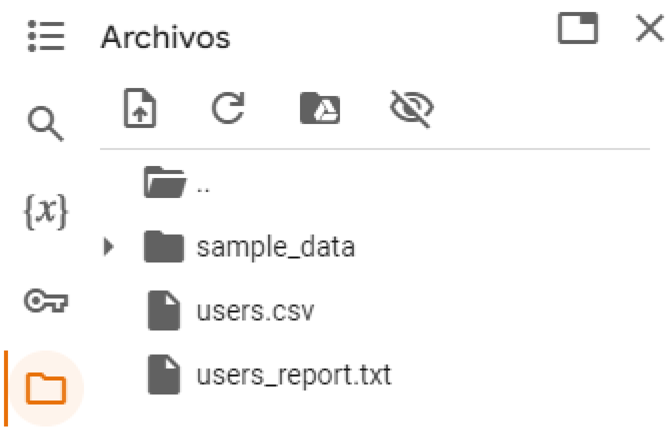
## Download

If you wish to download and run the code you will need the following:

Google Colab: Make sure you have a connection to use Google Colab and go to the following link https://colab.research.google.com/ . You must have an account to enter.

Running the code: In order to run the code, you need to run each cell one by one, by pushing the play button, and in order from top to bottom.

*Additional code: In order to run the code in which you need an additional file, you need to first upload the file to "Files" section.



When you run the code, it then will ask you just to insert the name of the file.

## First Code Explanation

**Library**

```python
import pandas as pd
```

The library we will be using is pandas for reading CSV files.

**Test Cases**

```python
# Helper function to create CSV files for testing
def create_csv(content, file_name):
    with open(file_name, 'w') as file:
        file.write(content)

# Test cases
test_cases = {
    # Regular test case
    'users.csv': """username,email
    GabrielDelfin,gabodelfin@tec.com
    Benji,benjamin@teachertec.com
    LuisPaulin,luigi@tec.com""",

    # Test case for an empty document
    'empty_users.csv': """username,email""",

    # Test case for multiple domains
    'multiple_domains_users.csv': """username,email
    luis,luis@tecqro.com
    gabriel,gabriel2@tecmty.com
    delfin,delfin@tecgdl.com
    paulin,paulin@tecqro.com
    benji,benji@tecgdl.com"""
}

# Create CSV files for each test case
for file_name, content in test_cases.items():
    create_csv(content, file_name)
```

In here we define our test cases, which will be our CSV files. We also define a function “create_csv” to create a CSV file and then create a CSV file for each of the test cases we defined.

**Read and Extract CSV**

```
# Function to read CSV and extract data
def read_and_extract_data(file_path):
    # Read CSV file
    df = pd.read_csv(file_path)

    # Extract relevant columns
    usernames = df['username']
    emails = df['email']

    return usernames, emails

# Function to process data
def process_data(usernames, emails):
    # Count number of users
    user_count = len(usernames)

    # Analyze email domains
    domain_counts = emails.str.split('@').str[1].value_counts()

    return user_count, domain_counts
```

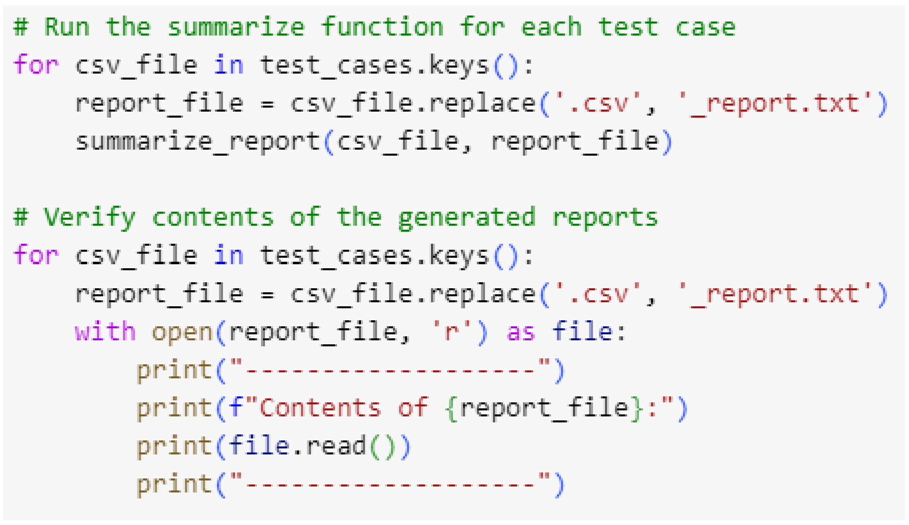
In here we create a function "read_and_extract_data" which helps us to read and extract the usernames and emails from the CSV file. The function "process_data" helps us by counting the number of users and the number of email domains.

**Generate Report**

```
# Function to generate report
def generate_report(user_count, domain_counts, report_path):
    with open(report_path, 'w') as file:
        file.write(f"Total number of users: {user_count}\n")
        file.write("\nEmail domain distribution:\n")
        for domain, count in domain_counts.items():
            file.write(f"{domain}: {count}\n")

# Function to summarize the information
def summarize_report(input_csv, report_path):
    usernames, emails = read_and_extract_data(input_csv)
    user_count, domain_counts = process_data(usernames, emails)
    generate_report(user_count, domain_counts, report_path)
    print(f"Report for {input_csv} generated successfully!")
```

In here we define a function “generate_report” which generates a report using the number of users counted in our previous function “process_data” and the domain count. Then we define a function “summarize_function” to get all our information together for our report text file.

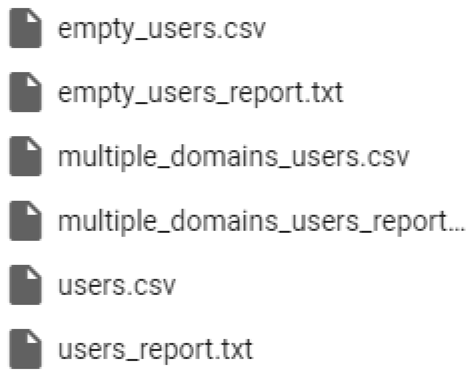
**Compiling**



```
# Run the summarize function for each test case
for csv_file in test_cases.keys():
    report_file = csv_file.replace('.csv', '_report.txt')
    summarize_report(csv_file, report_file)

# Verify contents of the generated reports
for csv_file in test_cases.keys():
    report_file = csv_file.replace('.csv', '_report.txt')
    with open(report_file, 'r') as file:
        print("-------------------")
        print(f"Contents of {report_file}:")
        print(file.read())
        print("-------------------")
```

Finally we call our functions “report_file” and “summarize_report” to generate each of our reports and prints the results to verify that everything is correct.

**Execution**



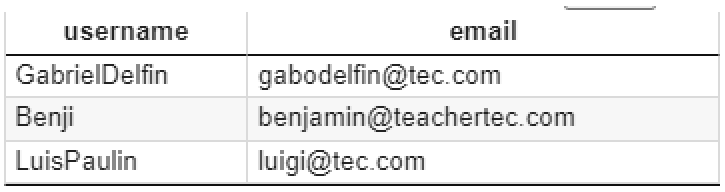
```
empty_users.csv
empty_users_report.txt
multiple_domains_users.csv
multiple_domains_users_report...
users.csv
users_report.txt
```

As we can see, we end with 6 documents, 3 of them are our test cases and the other 3 are the reports for each of our test cases.

```
Report for users.csv generated successfully!
Report for empty_users.csv generated successfully!
Report for multiple_domains_users.csv generated successfully!
-------------------
Contents of users_report.txt:
Total number of users: 3

Email domain distribution:
tec.com: 2
teachertec.com: 1

-------------------
-------------------
Contents of empty_users_report.txt:
Total number of users: 0

Email domain distribution:

-------------------
-------------------
Contents of multiple_domains_users_report.txt:
Total number of users: 5

Email domain distribution:
tecqro.com: 2
tecgdl.com: 2
tecmty.com: 1

-------------------
```

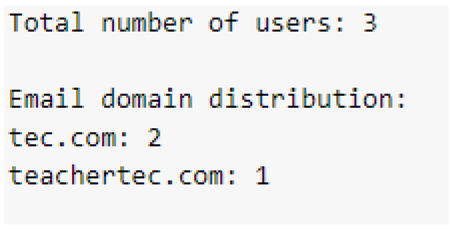
Then we have our results all together just to confirm that the information is correct. We could also see this information in our .txt files, which are the reports generated.

.csv file



| username | email |
|---|---|
| GabrielDelfin | gabodelfin@tec.com |
| Benji | benjamin@teachertec.com |
| LuisPaulin | luigi@tec.com |

.txt file



```
Total number of users: 3

Email domain distribution:
tec.com: 2
teachertec.com: 1
```

## Second Code Explanation

**Library**

```python
import pandas as pd
```

The library we will be using is pandas for reading CSV files.

**Read and Extract CSV**

```python
# Function to read CSV and extract data
def read_and_extract_data(file_path):
    # Read CSV file
    df = pd.read_csv(file_path)

    # Extract relevant columns
    usernames = df['username']
    emails = df['email']

    return usernames, emails

# Function to process data
def process_data(usernames, emails):
    # Count number of users
    user_count = len(usernames)

    # Analyze email domains
    domain_counts = emails.str.split('@').str[1].value_counts()

    return user_count, domain_counts
```

In here we create a function “read_and_extract_data” which helps us to read and extract the usernames and emails from the CSV file. The function “process_data” helps us by counting the number of users and the number of email domains.

**Generate Report**

```python
# Function to generate report
def generate_report(user_count, domain_counts, report_path):
    with open(report_path, 'w') as file:
        file.write(f"Total number of users: {user_count}\n")
        file.write("\nEmail domain distribution:\n")
        for domain, count in domain_counts.items():
            file.write(f"{domain}: {count}\n")

# Function to summarize the information
def summarize_report(input_csv, report_path):
    usernames, emails = read_and_extract_data(input_csv)
    user_count, domain_counts = process_data(usernames, emails)
    generate_report(user_count, domain_counts, report_path)
    print(f"Report for {input_csv} generated successfully!")
```

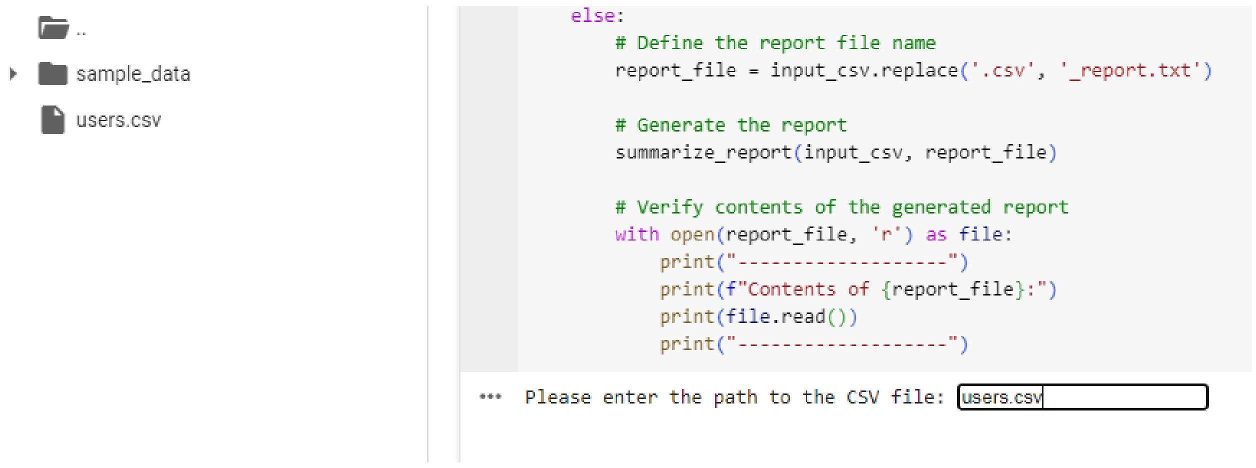
In here we define a function “generate_report” which generates a report using the number of users counted in our previous function “process_data” and the domain count. Then we define a function “summarize_function” to get all our information together for our report text file.

**Input CSV and Compiling**

```python
# Main execution
if __name__ == "__main__":
    import os

    # Prompt the user to input the path of the CSV file
    input_csv = input("Please enter the path to the CSV file: ")

    # Check if the file exists
    if not os.path.isfile(input_csv):
        print("The file does not exist. Please check the file path and try again.")
    else:
        # Define the report file name
        report_file = input_csv.replace('.csv', '_report.txt')

        # Generate the report
        summarize_report(input_csv, report_file)

        # Verify contents of the generated report
        with open(report_file, 'r') as file:
            print("-------------------")
            print(f"Contents of {report_file}:")
            print(file.read())
            print("-------------------")
```

Finally we ask the user to input the CSV file and then we call our functions “report_file” and “summarize_report” to generate each of our reports and prints the results to verify that everything is correct.

**Execution**



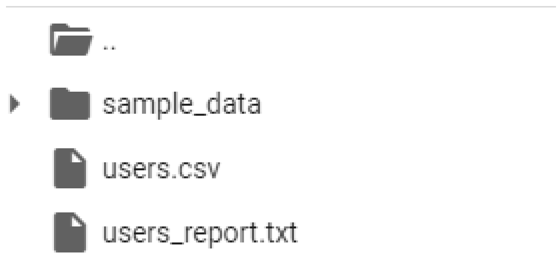
```
    else:
        # Define the report file name
        report_file = input_csv.replace('.csv', '_report.txt')

        # Generate the report
        summarize_report(input_csv, report_file)

        # Verify contents of the generated report
        with open(report_file, 'r') as file:
            print("-------------------")
            print(f"Contents of {report_file}:")
            print(file.read())
            print("-------------------")
```

```
Please enter the path to the CSV file: users.csv
```

Remember to upload your .csv file in the Files section and then when the program asks you input the file, write the exact name of the uploaded file.

```
Please enter the path to the CSV file: users.csv
Report for users.csv generated successfully!
-------------------
Contents of users_report.txt:
Total number of users: 50

Email domain distribution:
singledomain.com: 10
sample.com: 6
example.com: 4
domain1.com: 4
domain2.com: 3
domain3.com: 3
actor.com: 2
company2.org: 1
company9.com: 1
company8.edu: 1
company7.net: 1
company6.org: 1
company5.com: 1
company4.edu: 1
company3.net: 1
inventor.com: 1
company1.com: 1
singer.com: 1
tvshow.com: 1
producer.com: 1
musician.com: 1
writer.com: 1
detective.com: 1
fictional.com: 1
company10.org: 1
```

As we can see, we have even more examples because the .csv that I uploaded is bigger than the test cases from the first program.



It will take longer to generate a report, so probably if we have even a bigger document it will take even more.