Ministerul Educației și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Raport

la disciplina **”Programarea Declarativa”**

Tema: **Manipularea datelor folosind limbajul Python**

Efectuat de: studentul/studenta gr.**TI-216 Vlasitchi Stefan**

Verificat de: asis.univ **Viorel Rusu**

Chișinău-2023

Exercitiul 1.

Scrieți secvența de cod care regrupează datele din cele 3 fișiere csv (obținute la laboratul precedent) după o caracteristică de bază (de ex: cursul valutar de la 3 bănci pentru aceeași zi)

```
from pathlib import Path
import pandas as pd


def filter_and_save(input_file, output_file=None, start_date='2023-10-24',
end_date='2023-10-31'):
    # Load data from the CSV file
    df = pd.read_csv(input_file)

    # Convert the 'Date' column to the datetime data type
    df['Date'] = pd.to_datetime(df['Date'])

    # Select only the data between the specified date range
    filtered_df = df[df['Date'].between(start_date, end_date)]

    # Extract the file name without extension using pathlib
    input_path = Path(input_file)
    file_name = input_path.stem
    file_extension = input_path.suffix

    # Construct the name of the new output file
    new_output_file = Path(output_file) if output_file else input_path.parent
/ f"new_{file_name}{file_extension}"

    # Save the sorted and filtered DataFrame to a new CSV file
    filtered_df.to_csv(new_output_file, index=False)

    print(
        f"Data from {input_file} has been sorted and filtered for the period
{start_date} - {end_date} and saved in {new_output_file}.")


# List of CSV file names
files = ['1_exchange_rates_october.csv', '2_exchange_rates_october.csv',
'3_exchange_rates_october.csv']

# Iterate through each file and apply the sorting function
for file in files:
    filter_and_save(file)
```

Figura 1. Codul exercitiului 1

	Date	Value
2	2023-10-31	19.155 MDL
3	2023-10-30	19.221 MDL
4	2023-10-29	19.113 MDL
5	2023-10-28	19.200 MDL
6	2023-10-27	19.201 MDL
7	2023-10-26	19.069 MDL
8	2023-10-25	19.069 MDL
9	2023-10-24	19.198 MDL

	Date	Value
2	2023-10-24	19.14 MDL
3	2023-10-25	18.97 MDL
4	2023-10-26	18.84 MDL
5	2023-10-27	18.82 MDL
6	2023-10-29	19.03 MDL
7	2023-10-30	18.83 MDL
8	2023-10-31	19.06 MDL

	Date	Value
2	2023-10-24	19.3376 MDL
3	2023-10-25	19.3432 MDL
4	2023-10-26	19.1997 MDL
5	2023-10-27	19.0875 MDL
6	2023-10-28	19.0875 MDL
7	2023-10-29	19.0875 MDL
8	2023-10-30	19.1200 MDL
9	2023-10-31	19.2190 MDL

Figura 2 . Rezultatul exercitiului 1

Exercitiul 2.

Determinați min, max, media pentru valorile regrupate pentru fiecare zi aparte și min, max media pentru toate datele stocate.

```
import pandas as pd
import os


def load_and_process_data(file_path):
    df = pd.read_csv(file_path)
    df['Date'] = pd.to_datetime(df['Date'])
    df['Value'] = pd.to_numeric(df['Value'].astype(str).str.replace(' MDL',
''), errors='coerce')
    df = df.dropna(subset=['Value'])
    return df


def calculate_daily_statistics(df):
    return df.groupby('Date')['Value'].agg(['min', 'max', 'mean'])


def display_statistics_for_date(date, stats_dict):
    print(f"\nComparare pentru data: {date} 00:00:00")
    for stat_type in ['min', 'max', 'mean']:
        values = [stats_dict[file_key].loc[date, stat_type] for file_key in
stats_dict]
        result = min(values) if stat_type == 'min' else max(values) if
stat_type == 'max' else sum(values) / len(values)
        print(f"{stat_type}: {result}")


def calculate_and_display_statistics(file_names, file_type):
    daily_statistics_dict = {}

    for file_name in file_names:
        df = load_and_process_data(file_name)
        daily_statistics = calculate_daily_statistics(df)
        file_key = os.path.splitext(os.path.basename(file_name))[0]
        daily_statistics_dict[file_key] = daily_statistics

    all_dates = set().union(*[daily_statistics_dict[file_key].index for
file_key in daily_statistics_dict])

    for date_to_check in sorted(all_dates):
        display_statistics_for_date(date_to_check, daily_statistics_dict)


def main():
    new_files = [f for f in os.listdir() if f.startswith("new")]
    specified_files = ['1_exchange_rates_october.csv', '2_exchange_rates_octo-
ber.csv', '3_exchange_rates_october.csv']

    print("\nApelul funcției pentru calculul statisticilor pentru fișierele
create anterior cu prefixul 'new'")
    calculate_and_display_statistics(new_files, "new")

    print("\n----------------------------------------------------------------
-----------")
```

```
    print("\nApelul funcției pentru calculul statisticilor pentru fișierele
specificate")
    calculate_and_display_statistics(specified_files, "specified")


if __name__ == "__main__":
    main()
```

Figura 3 . Codul exercitiului 2

```
Apelul funcției pentru calculul statisticilor pentru fișierele create anterior cu prefixul 'new'

Comparare pentru data: 2023-10-24 00:00:00 00:00:00
min: 19.14
max: 19.3376
mean: 19.2252
```

Figura 4.1. Rezultatul exercitiului 2

```
Apelul funcției pentru calculul statisticilor pentru fișierele specificate

Comparare pentru data: 2023-10-01 00:00:00 00:00:00
min: 19.1338
max: 19.295
mean: 19.192933333333333
```

Figura 4.2. Rezultatul exercitiului 2

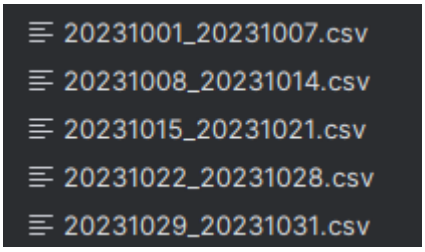
Exercitiul 3.

Scrieți secvența de cod care va împărți fișierul csv obținut după concatinare în N fișiere, unde fiecare fișier individual va corespunde unei săptămâini (7 rânduri). Fișierele sunt denumite după prima și ultima dată pe care le conțin (de ex 20231103_20231109.csv).

```
import pandas as pd

END_DATE = pd.to_datetime('2023-10-31')

files_to_concat = ['1_exchange_rates_october.csv', '2_exchange_rates_octo-
ber.csv', '3_exchange_rates_october.csv']

# List Comprehension for loading data
dataframes = [pd.read_csv(file) for file in files_to_concat]

concatenated_df = pd.concat(dataframes, ignore_index=True)
concatenated_df.to_csv('fisier_concatenat.csv', index=False)
print("Fișierul concatenat a fost creat cu succes.")
```

```
# Date parsing with error handling
concatenated_df['Date'] = pd.to_datetime(concatenated_df['Date'], format='%d
%B %Y', errors='coerce')

if concatenated_df['Date'].isna().any():
    print("Warning: Some dates could not be parsed.")

# Sorting by date
concatenated_df = concatenated_df.sort_values(by='Date')

# Using a fixed interval for weeks
distinct_weeks = pd.date_range(start=concatenated_df['Date'].min(), end=con-
catenated_df['Date'].max(), freq='7D')

for start_date in distinct_weeks:
    end_date = start_date + pd.DateOffset(days=6)
    end_date = min(end_date, END_DATE)  # Adjust end date to END_DATE constant

    week_df = concatenated_df[(concatenated_df['Date'] >= start_date) & (con-
catenated_df['Date'] <= end_date)]

    if not week_df.empty:
        weekly_output_file =
f"{start_date.strftime('%Y%m%d')}_{end_date.strftime('%Y%m%d')}.csv"
        week_df.to_csv(weekly_output_file, index=False)
        print(f"Created file: {weekly_output_file}")
    else:
        print("Could not create file: no valid data.")
```

Figura 5. Codul exercitiului 3



	Date	Value
2	2023-10-29	19.113 MDL
3	2023-10-29	19.0875 MDL
4	2023-10-29	19.03 MDL
5	2023-10-30	19.1200 MDL
6	2023-10-30	18.83 MDL
7	2023-10-30	19.221 MDL
8	2023-10-31	19.155 MDL
9	2023-10-31	19.06 MDL
10	2023-10-31	19.2190 MDL

Figura 6. Rezultatul exercitiului 3

Exercitiul 4.

Scrieți funcția ce are o dată de intrare tip datetime și returnează liniile din DataFrame (din fișierul cu date concatinate) pentru această dată sau None dacă nu există date pentru această dată.

```
import pandas as pd

# Încarcă fișierul CSV obținut după concatinare
try:
    concatenated_df = pd.read_csv("fisier_concatenat.csv")
except FileNotFoundError:
    print("Fișierul fisier_concatenat.csv nu a fost găsit.")
```

```
    concatenated_df = None


def filter_data_by_date(df, target_date):
    if df is not None and 'Date' in df.columns:
        try:
            formatted_target_date = target_date.strftime('%d %B %Y')
            filtered_rows = df[df['Date'] == formatted_target_date]

            if not filtered_rows.empty:
                return filtered_rows
            else:
                print(f'Nu există date pentru data: {target_date}')
                return None
        except ValueError as e:
            print(f"Eroare la filtrarea datelor: {e}")
            return None
    else:
        print("DataFrame invalid pentru filtrare.")
        return None


# Exemplu de utilizare:
try:
    day_input = input("Introdu ziua (DD): ")
    target_date = pd.to_datetime(f'2023-10-{day_input}', format='%Y-%m-%d')
except ValueError as e:
    print(f"Eroare: {e}")
    target_date = None

if target_date is not None:
    result = filter_data_by_date(concatenated_df, target_date)

    if result is not None:
        print(f"\nDatele pentru data {target_date} sunt:")
        print(result)
```

Figura 7. Codul exercitiului 4

```
Datele pentru data 2023-10-01 00:00:00 sunt:
                Date        Value
30  01 October 2023   19.295 MDL
31  01 October 2023    19.15 MDL
62  01 October 2023  19.1338 MDL
```

Figura 8. Rezultatul exercitiului 4

Exercitiul 5.

Scrieți secvența de cod care permite afișarea datelor pe secvențe de timp. De ex. pentru fiecare săptămână separat (funcția are o dată de intrare tip datetime) și returnează liniile din DataFrame pentru această perioadă sau None dacă nu există date pentru zilele indicate.

```
import pandas as pd

try:
    # Încarcă fișierul CSV obținut după concatinare
    concatenated_df = pd.read_csv("fisier_concatenat.csv")
except FileNotFoundError:
    print("Fișierul fisier_concatenat.csv nu a fost găsit.")
    concatenated_df = None

def get_data_for_time_interval(dataframe, start_date, end_date):
    try:
        # Asigură-te că coloana 'Date' este de tipul datetime
        dataframe['Date'] = pd.to_datetime(dataframe['Date'], errors='coerce')

        # Construiește datele de start și final cu luna și anul fixate la
2023-10
        start_date = pd.to_datetime(f'2023-10-{start_date:02}', format='%Y-%m-
%d')
        end_date = pd.to_datetime(f'2023-10-{end_date:02}', format='%Y-%m-%d')

        # Filtrarea dataframe-ului pentru intervalul specificat
        time_interval_data = dataframe[(dataframe['Date'] >= start_date) &
(dataframe['Date'] <= end_date)]

        # Verificare dacă există date pentru intervalul specificat
        if time_interval_data.empty:
            print(f"Nu există date pentru intervalul de la {start_date} la
{end_date}")
            return None
        else:
            return time_interval_data

    except ValueError as e:
        print(f"Eroare la conversia datelor sau filtrarea datelor: {e}")
        return None

# Exemplu de utilizare
try:
    start_date_input = int(input("Introdu data inițială (format: DD): "))
    end_date_input = int(input("Introdu data finală (format: DD): "))

    # Verifică dacă datele introduse sunt în formatul corect și sunt valide
    if 1 <= start_date_input <= 31 and 1 <= end_date_input <= 31:
        result = get_data_for_time_interval(concatenated_df, start_date_input,
end_date_input)

        if result is not None:
            print(result)
    else:
        print("Ați introdus o dată greșită.")

except ValueError:
    print("Datele introduse nu sunt valide.")
```

Figura 9. Codul exercitiului 5

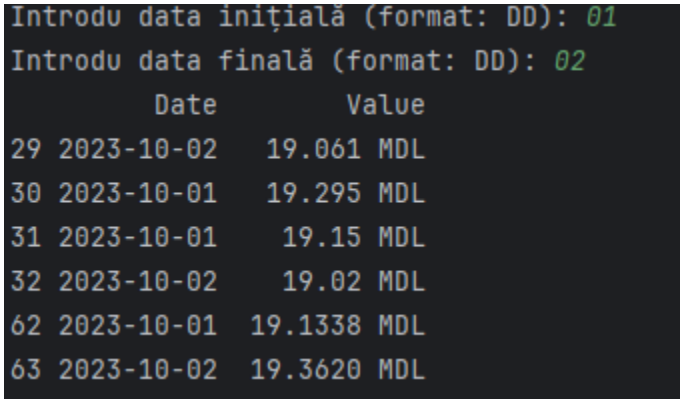


Figura 9. Rezultaul exercitiului 5

Concluzie

În cadrul laboratorului, s-au abordat diverse aspecte legate de manipularea datelor folosind limbajul Python. Am început prin concatenarea mai multor fișiere CSV într-un singur cadru de date, apoi am explorat diverse moduri de analiză și filtrare a acestora.

Pentru a face manipularea datelor mai interactivă și personalizată, am implementat funcții care permit utilizatorului să introducă date inițiale și finale pentru a obține intervalul dorit de informații din cadru. În aceste funcții, am folosit librăria Pandas pentru manipularea datelor și am integrat aspecte practice precum validarea intrărilor utilizatorului și gestionarea corectă a formatului datei.

În plus, am explorat moduri de sortare și exportare a datelor, creând fișiere CSV separate pentru săptămâni și afișând rezultatele într-un mod coerent și ușor de înțeles.