МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 8
«Візуалізація та обробка даних за допомогою
спеціалізованих бібліотек Python»
з дисципліни "Спеціалізовані мови програмування”

Виконала:
ст. гр. ІТ-32,
Троцько О. М.
Прийняв:
Щербак С. С.

ЛЬВІВ – 2023

Мета: розробка додатка для візуалізації CSV-наборів даних за допомогою Matplotlib та базових принципів ООП (наслідування, інкапсуляція, поліморфізм).

План роботи

Завдання 1: Вибір CSV-набору даних

Оберіть CSV-набір даних, який ви хочете візуалізувати. Переконайтеся, що він містить відповідні дані для створення змістовних візуалізацій.

Завдання 2: Завантаження даних з CSV

Напишіть код для завантаження даних з CSV-файлу в ваш додаток Python. Використовуйте бібліотеки, такі як Pandas, для спрощення обробки даних.

Завдання 3: Дослідження даних

Визначте екстремальні значення по стовцям

Завдання 4: Вибір типів візуалізацій

Визначте, які типи візуалізацій підходять для представлення вибраних наборів даних. Зазвичай це може бути лінійні графіки, стовпчикові діаграми, діаграми розсіювання, гістограми та секторні діаграми.

Завдання 5: Підготовка даних

Попередньо обробіть набір даних за необхідністю для візуалізації. Це може включати виправлення даних, фільтрацію, агрегацію або трансформацію.

Завдання 6: Базова візуалізація

Створіть базову візуалізацію набору даних, щоб переконатися, що ви можете відображати дані правильно за допомогою Matplotlib. Розпочніть з простої діаграми для візуалізації однієї змінної.

Завдання 7: Розширені візуалізації

Реалізуйте більш складні візуалізації, виходячи з характеристик набору. Поекспериментуйте з різними функціями Matplotlib та налаштуваннями.

Завдання 8: Декілька піддіаграм

Навчіться створювати кілька піддіаграм в межах одного малюнка для відображення декількох візуалізацій поруч для кращого порівняння.

Завдання 9: Експорт і обмін

Реалізуйте функціональність для експорту візуалізацій як зображень (наприклад, PNG, SVG) або інтерактивних веб-додатків (наприклад, HTML)

Код програми:

```
# diagrams.py
"""
Diagrams Module

A module that defines the Diagrams class for creating and visualizing diagrams based on data from a CSV file.
"""
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
from shared.settings import get_lab_settings

settings = get_lab_settings("lab8")
DIAGRAMS_DIR = settings["diagrams_dir"]
DEFAULT_YEAR = settings["default_year"]
DEFAULT_MONTH = settings["default_month"]


class Diagrams:
    """
    A class for creating and visualizing diagrams based on data from a CSV file.

    Attributes:
    - df (pandas.DataFrame): The DataFrame containing the loaded data from the CSV file.
    """

    def __init__(self, csv_file_path):
        """
        Initializes a Diagrams object.

        Parameters:
        - csv_file_path (str): The path to the CSV file containing the data.
```

```
    Returns:
    - None
    """
    self.df = self.load_csv(csv_file_path)

def load_csv(self, csv_file_path):
    """
    Loads a CSV file into a pandas DataFrame.

    Parameters:
    - csv_file_path (str): The path to the CSV file.

    Returns:
    - df (pandas.DataFrame): The loaded DataFrame.
    """
    try:
        df = pd.read_csv(csv_file_path)
        print("Data was loaded successfully.")
        return df
    except FileNotFoundError:
        print(f"File '{csv_file_path}' was not found.")
    except pd.errors.EmptyDataError:
        print(f"File '{csv_file_path}' is empty or doesn't contain data.")
    except pd.errors.ParserError:
        print(f"Cannot read the file '{csv_file_path}'.")

def print_min_values(self):
    """
    Prints the minimum values for each column in the DataFrame.

    Parameters:
    - None

    Returns:
    - None
    """
    min_values = self.df.min()
    print("\nMin values:")
    print(min_values)
```

```
def print_max_values(self):
    """
    Prints the maximum values for each column in the DataFrame.

    Parameters:
    - None

    Returns:
    - None
    """
    max_values = self.df.max()
    print("\nMax values:")
    print(max_values)

def get_age(self):
    """
    Calculates the age of each person in the DataFrame.

    Parameters:
    - None

    Returns:
    - age (pandas.Series): The calculated age of each person.
    """
    self.df['birthdate'] = pd.to_datetime(self.df['birthdate'])
    today = datetime.now()
    return (today - self.df['birthdate']).astype('<m8[Y]')

def visualize_histogram(self):
    """
    Visualizes a histogram of the number of people by country.

    Parameters:
    - None

    Returns:
    - None
    """
    num_people_by_country = self.df['country'].value_counts()
```

```
        plt.figure(figsize=(10, 6))
        plt.hist(self.df['country'], bins=len(num_people_by_country), color='pink',
edgecolor='black', alpha=0.7)
        plt.title('Number of People by Country')
        plt.xlabel('Country')
        plt.ylabel('Number of People')
        plt.grid(axis='y')
        fig = plt.gcf()
        plt.show()
        self.export_data(fig, "histogram")

    def visualize_column_diagram(self):
        """
        Visualizes a column diagram of the average age of people by country.

        Parameters:
        - None

        Returns:
        - None
        """
        self.df['age'] = self.get_age()
        avg_age_by_country = self.df.groupby('country')['age'].mean()
        plt.figure(figsize=(10, 6))
        avg_age_by_country.plot(kind='bar', color='orange')
        plt.title('Average Age of People by Country')
        plt.xlabel('Country')
        plt.ylabel('Average Age')
        plt.grid(axis='y')
        fig = plt.gcf()
        plt.show()
        self.export_data(fig, "column_diagram")

    def visualize_sector_diagram(self, year=DEFAULT_YEAR):
        """
        Visualizes a sector diagram of the percentage of people born after a given
year by country.

        Parameters:
```

```
          - year (int): The year after which to consider people for the diagram.
Default is DEFAULT_YEAR.

        Returns:
        - None
        """
        self.df['age'] = self.get_age()
        df_after_year = self.df[self.df['birthdate'].dt.year >= year]
                                                    percentage_after_year      =
df_after_year['country'].value_counts(normalize=True) * 100
        plt.figure(figsize=(8, 8))
                plt.pie(percentage_after_year,  labels=percentage_after_year.index,
autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
        plt.title(f'Percentage of People Born After {year} by Country')
        fig = plt.gcf()
        plt.show()
        self.export_data(fig, "sector_diagram")

    def visualize_line_plot_and_sector(self, month=DEFAULT_MONTH):
        """
          Visualizes a line plot of the average age of people born in a specific
month by country,
         along with a pie chart showing the distribution of countries used in the
line plot.

        Parameters:
         - month (int): The month for which to consider people for the line plot.
Default is DEFAULT_MONTH.

        Returns:
        - None
        """
         months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December']
        self.df['age'] = self.get_age()
        df_june = self.df[self.df['birthdate'].dt.month == month]
        avg_age_by_country = df_june.groupby('country')['age'].mean()
        countries_used = avg_age_by_country.index
        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))
```

```
        # Plot 1: Line plot for average age by country
        avg_age_by_country.plot(kind='line', marker='o', color='green', ax=axes[0])
           axes[0].set_title(f'Average Age of People Born in {months[month-1]} by
Country')
        axes[0].set_xlabel('Country')
        axes[0].set_ylabel('Average Age')
        axes[0].grid(True)

        # Plot 2: Pie chart for the distribution of countries used in the line plot
            country_counts_used = self.df[self.df['country'].isin(countries_used)]
['country'].value_counts()
                axes[1].pie(country_counts_used, labels=country_counts_used.index,
autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
        axes[1].set_title('Distribution of Countries in the Line Plot')
        axes[1].axis('equal')
        plt.tight_layout()
        fig = plt.gcf()
        plt.show()
        self.export_data(fig, "line_plot_and_sector")

    def export_data(self, fig, file_name='diagram'):
        """
        Exports the diagram to a PNG file.

        Parameters:
            - fig (matplotlib.figure.Figure): The figure object representing the
diagram.
         - file_name (str): The name of the file to save the diagram. Default is
'diagram'.

        Returns:
        - None
        """
        choice = input("Export to PNG? (y/n): ")
        if choice.capitalize() == 'Y':
            fig.savefig(f'{DIAGRAMS_DIR}{file_name}.png')
            print(f"Exported to {file_name}.png")
        fig.clf()
```

```python
# csv_menu.py
"""
Module: csv_menu

This module contains the CSVMenu class, which represents a CSV menu for displaying
extreme values and diagrams.
"""
from UI.menu import Menu
from UI.menu_item import Item
from classes.lab8.diagrams_saver.diagrams import Diagrams
from shared.settings import get_lab_settings

settings = get_lab_settings("lab8")
CSV_FILE_PATH = settings["csv_file_path"]

class CSVMenu:
    """
    A class representing a CSV menu.

    Attributes:
    - csv_file_path (str): The path to the CSV file.
    - diagrams (Diagrams): An instance of the Diagrams class.
    """

    def __init__(self) :
        self.csv_file_path = CSV_FILE_PATH
        self.diagrams = Diagrams(self.csv_file_path)

    def menu(self):
        """
        Displays the main menu.
        """
        main_menu = Menu("\nMenu (Lab 8)")
        main_menu.set_color("red")
        main_menu.add_item(Item("1", "Extreme values", self.values_menu))
        main_menu.add_item(Item("2", "Diagrams", self.diagram_menu))
        main_menu.add_item(Item("0", "Exit"))
        main_menu.run()

    def values_menu(self):
```

```
        """
        Displays the values menu.
        """
        values_menu = Menu("\nExtreme values")
        values_menu.set_color("red")
                                values_menu.add_item(Item("1",    "Min    values",
self.diagrams.print_min_values))
                                values_menu.add_item(Item("2",    "Max    values",
self.diagrams.print_max_values))
        values_menu.add_item(Item("0", "Back"))
        values_menu.run()

    def diagram_menu(self):
        """
        Displays the diagram menu.
        """
        diagram_menu = Menu("\nDiagrams")
        diagram_menu.set_color("red")
         diagram_menu.add_item(Item("1", "Visualize people by country (histogram)",
self.diagrams.visualize_histogram))
         diagram_menu.add_item(Item("2", "Visualize percentage of people born after
(some year) by country (pie chart)", self.visuzlize_pie_chart_with_input))
        diagram_menu.add_item(Item("3", "Visualize average age of people by country
(column diagram)", self.diagrams.visualize_column_diagram))
          diagram_menu.add_item(Item("4", "Visualize average age of people born in
(some    month)    by    country    (linear    diagram    and    sector    diagram)",
self.visuzlize_line_plot_and_sector_with_input))
        diagram_menu.add_item(Item("0", "Back"))
        diagram_menu.run()

    def visuzlize_pie_chart_with_input(self):
        """
        Visualizes a pie chart based on user input.
        """
        year = input("Enter year (1950-2005): ")
        if year.isdigit():
            year = int(year)
            if 1950 <= year <= 2005:
                self.diagrams.visualize_sector_diagram(year)
                return
```

```
        print("Invalid input.")
        self.diagrams.visualize_sector_diagram()

    def visuzlize_line_plot_and_sector_with_input(self):
        month = input("Enter month (1-12): ")
        if month.isdigit():
            month = int(month)
            if 1 <= month <= 12:
                self.diagrams.visualize_line_plot_and_sector(month)
                return

        print("Invalid input.")
        self.diagrams.visualize_line_plot_and_sector()
```

```
# runner.py
"""
Module: run_csv_menu

Module provides a simple script to run the CSV Menu for diagrams for Lab 8.
"""
from classes.lab8.diagrams_menu.csv_menu import CSVMenu

def run():
    """
    Initializes and runs the CSV Menu for diagrams.
    """
    csv_menu = CSVMenu()
    csv_menu.menu()
```

GitHub Repository: https://github.com/trolchiha/SPL-labs.git

Висновок: під час виконання лабораторної роботи навчилася створювати багатофункціональний додаток для візуалізації CSV-наборів даних за допомогою Matplotlib. Цей проект покращив мої навички візуалізації даних, дозволяючи досліджувати результати з різноманітними наборами даних.