МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №8

Візуалізація та обробка даних за допомогою спеціалізованих бібліотек Python

Виконала:

ст. гр. PI-32
Довгошия А.А

Балів	**Дата**

Прийняв:

асис. каф. ІСМ

Щербак С.С

Львів — 2024

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

Хід 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)

Data_explorer.py

```
"""
This module contains the DataExplorer class, which is responsible for exploring and analyzing data, such as finding extreme values in the dataset.
"""

class DataExplorer:
    """
    This class is responsible for exploring data, specifically to analyze
    extreme values (minimum and maximum) of numeric columns in a dataset.
    """
```

```
    def __init__(self, data):
        self.data = data

    def explore_data(self):
        """Explores extreme values in the dataset."""
        if self.data is not None:
            numeric_data = self.data.select_dtypes(include=['number'])
            print("Minimum values by numeric columns:")
            print(numeric_data.min())
            print("\nMaximum values by numeric columns:")
            print(numeric_data.max())

    def configure(self):
        """
        Placeholder method to avoid "too-few-public-methods" warning.
        This method can be extended if additional configuration is needed.
        """
        Pass
```

Data_handler.py

```
"""
This module contains the DataHandler class, which is responsible for
loading, exploring, and visualizing weather data. The class interacts
with the DataLoader, DataExplorer, and DataVisualizer to provide an
easy-to-use interface for data processing and analysis.
"""

from src.classes.lab8.data_loader import DataLoader
```

```
from src.classes.lab8.data_explorer import DataExplorer
from src.classes.lab8.data_visualizer import DataVisualizer

class DataHandler:
    """
    A class that handles loading, exploring, and visualizing weather data.

    Attributes:
        loader (DataLoader): Instance for loading data.
        data (DataFrame): The loaded data.
        explorer (DataExplorer): Instance for exploring the data.
        visualizer (DataVisualizer): Instance for visualizing the data.
    """

    def __init__(self, file_path):
        """
        Initializes the DataHandler with a file path for loading data.

        Args:
            file_path (str): The path to the data file.
        """
        self.loader = DataLoader(file_path)
        self.data = None
        self.explorer = None
        self.visualizer = None

    def load_and_process_data(self):
        """
```

```
    Loads the data and initializes the explorer and visualizer.

    This method loads the data, processes it, and initializes the
    `DataExplorer` and `DataVisualizer` if data is successfully loaded.
    """
    self.data = self.loader.load_data()
    if self.data is not None:
        self.explorer = DataExplorer(self.data)
        self.visualizer = DataVisualizer(self.data)

def explore_data(self):
    """
    Explores the data by checking for extreme values.

    Calls the `explore_data` method of the `DataExplorer` class.
    """
    if self.explorer:
        self.explorer.explore_data()

def visualize_data(self):
    """
    Visualizes the data using different plot types.

    Calls various visualization methods of the `DataVisualizer` class,
    including histogram, line chart, bar chart, scatter plot, pie chart,
    and subplots.
    """
    if self.visualizer:
```

```
        self.visualizer.plot_basic_histogram()
        self.visualizer.plot_line_chart()
        self.visualizer.plot_bar_chart()
        self.visualizer.plot_scatter_plot()
        self.visualizer.plot_pie_chart()
        self.visualizer.plot_multiple_subplots()
```

data_loader.py

```
import pandas as pd

class DataLoader:
    """
    A class that loads and preprocesses data from a CSV file.

    Attributes:
        file_path (str): The path to the CSV file.
        data (DataFrame): The loaded and processed data.

    Methods:
        load_data(): Loads and preprocesses the data from the CSV file.
        preprocess_data(): Handles missing values, date conversion,
                           and drops duplicates.
    """

    def __init__(self, file_path):
        """
        Initializes the DataLoader with the file path.
```

```
    Args:
        file_path (str): The path to the CSV file to load.
    """
    self.file_path = file_path
    self.data = None

def load_data(self):
    """Loads and preprocesses data from the CSV file."""
    try:
        self.data = pd.read_csv(self.file_path)
        self.preprocess_data()
        print("successful!")
    except FileNotFoundError:
        print(f"lost path to file: {self.file_path}")
    return self.data

def preprocess_data(self):
    """Preprocesses the data by handling missing values and dates."""
    if 'Date' in self.data.columns:
        self.data['Date'] = pd.to_datetime(self.data['Date'], errors='coerce')

    for column in self.data.select_dtypes(include=['float64', 'int64']).columns:
        self.data[column].fillna(self.data[column].mean(), inplace=True)

    for column in self.data.select_dtypes(include=['object']).columns:
        self.data[column].fillna(self.data[column].mode()[0], inplace=True)

    self.data.drop_duplicates(inplace=True)
```

```
        if 'Date' in self.data.columns:
            self.data['Year'] = self.data['Date'].dt.year
            self.data['Month'] = self.data['Date'].dt.month
            self.data['Day'] = self.data['Date'].dt.day
```

data_visualizer.py

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

"""This module contains the DataVisualizer class, which is used to visualize
weather data."""

class DataVisualizer:
    """A class for visualizing weather data using various types of plots."""

    def __init__(self, data):
        """Initialize the DataVisualizer with data."""
        self.data = data

    def plot_basic_histogram(self):
        """Basic histogram for a single variable (e.g., MaxTemp)."""
        if 'MaxTemp' in self.data.columns:
            plt.figure(figsize=(8, 6))
            self.data['MaxTemp'].plot(kind='hist', bins=30,
                                      color='green', edgecolor='black', alpha=0.7)
            plt.title('Distribution of Max Temperature')
```

```
        plt.xlabel('Max Temperature (°C)')
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.show()

    def plot_line_chart(self):
        """Line chart for MaxTemp and MinTemp over time (assuming there is a Date column)."""
        if 'Date' in self.data.columns:
            self.data['Date'] = pd.to_datetime(self.data['Date'])
            plt.figure(figsize=(10, 6))
            plt.plot(self.data['Date'], self.data['MaxTemp'],
                     label='Max Temperature', color='red')
            plt.plot(self.data['Date'], self.data['MinTemp'],
                     label='Min Temperature', color='yellow')
            plt.title('Max and Min Temperature Over Time')
            plt.xlabel('Date')
            plt.ylabel('Temperature (°C)')
            plt.legend()
            plt.grid(True)
            plt.xticks(rotation=45)
            plt.show()

    def plot_bar_chart(self):
        """Bar chart for RainToday (number of days with and without rain)."""
        rain_today_counts = self.data['RainToday'].value_counts()
        plt.figure(figsize=(8, 6))
        rain_today_counts.plot(kind='bar', color=['skyblue', 'orange'])
```

```
    plt.title('Rain Today (Yes vs No)')
    plt.xlabel('Rain Today')
    plt.ylabel('Number of Days')
    plt.xticks(rotation=0)
    plt.show()

def plot_scatter_plot(self):
    """Scatter plot for MaxTemp vs Rainfall."""
    plt.figure(figsize=(8, 6))
    sns.scatterplot(data=self.data, x='MaxTemp', y='Rainfall',
                    hue='RainToday', palette='coolwarm')
    plt.title('Max Temperature vs Rainfall')
    plt.xlabel('Max Temperature (°C)')
    plt.ylabel('Rainfall (mm)')
    plt.show()

def plot_pie_chart(self):
    """Pie chart for RainToday (Yes vs No)."""
    rain_today_counts = self.data['RainToday'].value_counts()
    plt.figure(figsize=(7, 7))
    rain_today_counts.plot(kind='pie', autopct='%1.1f%%',
                           colors=['lightcoral', 'lightgreen'])
    plt.title('Proportion of Rain Today (Yes vs No)')
    plt.ylabel('')
    plt.show()

def plot_multiple_subplots(self):
    """Create multiple subplots for different visualizations."""
```

```
_, axs = plt.subplots(2, 2, figsize=(12, 10))

if 'MaxTemp' in self.data.columns:
    axs[0, 0].hist(self.data['MaxTemp'], bins=30,
                   color='green', edgecolor='black', alpha=0.7)
    axs[0, 0].set_title('Distribution of Max Temperature')
    axs[0, 0].set_xlabel('Max Temperature (°C)')
    axs[0, 0].set_ylabel('Frequency')
    axs[0, 0].grid(True)

if ('Date' in self.data.columns and
    'MaxTemp' in self.data.columns and
    'MinTemp' in self.data.columns):
    self.data['Date'] = pd.to_datetime(self.data['Date'])
    axs[0, 1].plot(self.data['Date'], self.data['MaxTemp'],
                   label='Max Temperature', color='red')
    axs[0, 1].plot(self.data['Date'], self.data['MinTemp'],
                   label='Min Temperature', color='blue')
    axs[0, 1].set_title('Max and Min Temperature Over Time')
    axs[0, 1].set_xlabel('Date')
    axs[0, 1].set_ylabel('Temperature (°C)')
    axs[0, 1].legend()
    axs[0, 1].grid(True)

if 'RainToday' in self.data.columns:
    rain_today_counts = self.data['RainToday'].value_counts()
    axs[1, 0].bar(rain_today_counts.index, rain_today_counts.values,
                  color=['skyblue', 'orange'])
```

```
            axs[1, 0].set_title('Rain Today (Yes vs No)')
            axs[1, 0].set_xlabel('Rain Today')
            axs[1, 0].set_ylabel('Number of Days')

        if 'MaxTemp' in self.data.columns and 'Rainfall' in self.data.columns:
            sns.scatterplot(data=self.data, x='MaxTemp', y='Rainfall',
                            hue='RainToday', palette='coolwarm', ax=axs[1, 1])
            axs[1, 1].set_title('Max Temperature vs Rainfall')
            axs[1, 1].set_xlabel('Max Temperature (°C)')
            axs[1, 1].set_ylabel('Rainfall (mm)')

        plt.tight_layout()
        plt.show()
```

main.py

```
"""
This module loads, processes, and visualizes weather data.
It uses the DataHandler class to load the data, explore it,
and create different types of visualizations such as histograms,
line charts, and bar charts.
"""

from src.classes.lab8.data_handler import DataHandler
from src.const.path_constants import FOLDER_PATH_DATA


def main():
    """
```

```
    The main function to initialize the DataHandler, load the data,
    explore it, and generate visualizations.
    """
    file_path = FOLDER_PATH_DATA
    data_handler = DataHandler(file_path)

    data_handler.load_and_process_data()
    data_handler.explore_data()
    data_handler.visualize_data()

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

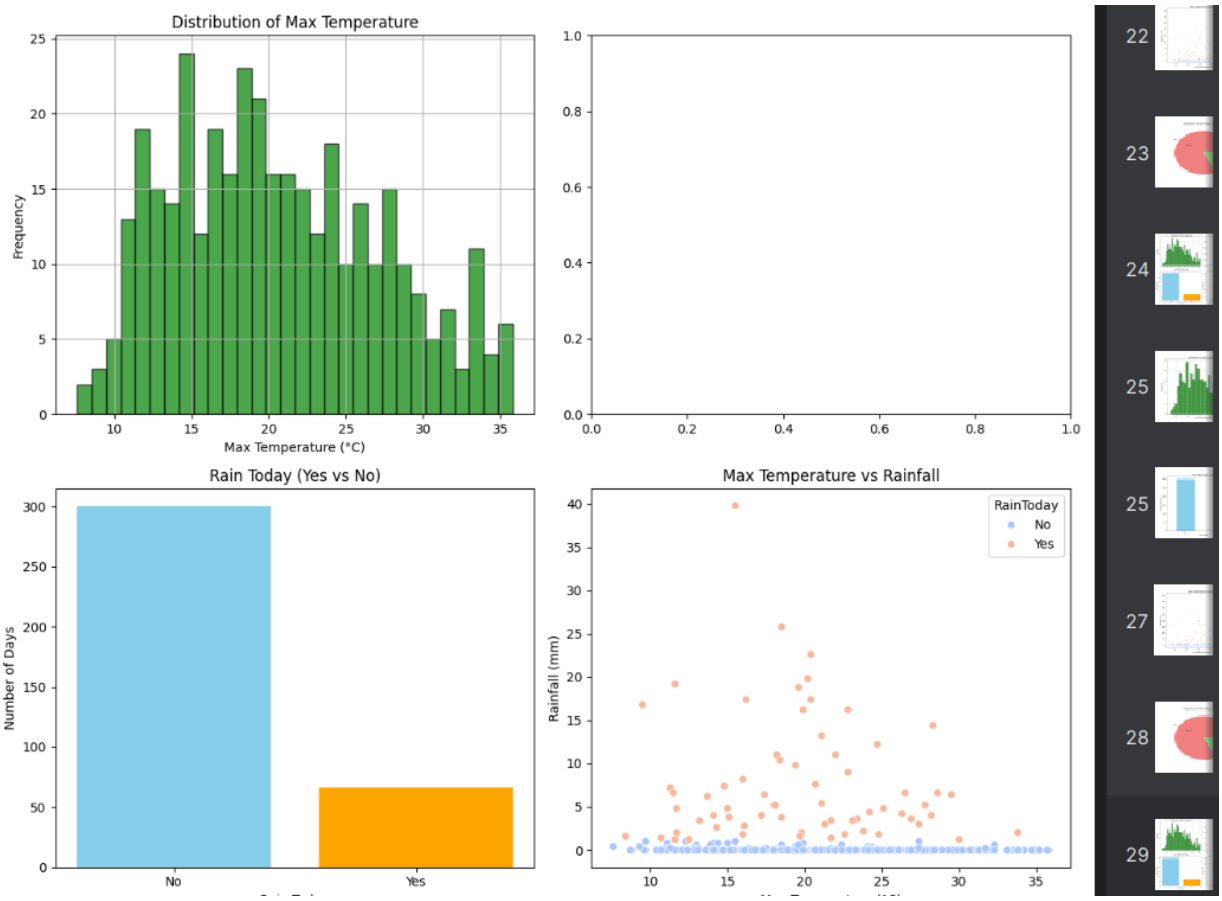

Рис.1. Результат програми.

Висновок: на цій лабораторній роботі я вивчила створення додатка для візуалізації CSV-наборів даних за допомогою Matplotlib та базових принципів ООП (наслідування, інкапсуляція, поліморфізм)