МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №8

з дисципліни

«Спеціалізовані мови програмування»

на тему

«Візуалізація та обробка даних за допомогою спеціалізованих бібліотек Python»

Виконав:

ст. гр. РІ-32

Юліан МЕЛЬНИЧУК

Прийняв:

доц. каф. ІСМ

Сергій ЩЕРБАК

Львів - 2024

**Мета**

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

**Хід виконання роботи**

**Завдання 1: Вибір CSV-набору даних**

Було обрано CSV-набір даних, що містить релевантну інформацію для створення змістовних візуалізацій, а саме інформацію про сон та про активність. Набір даних містить стовпці з важливими характеристиками для подальшого аналізу та представлення.

**Завдання 2: Завантаження даних з CSV**

Код для завантаження даних із CSV-файлу було успішно написано з використанням бібліотеки Pandas. Це дозволило спростити обробку даних і забезпечити їх структурування у вигляді таблиці для подальшого аналізу.

**Завдання 3: Дослідження даних**

Було досліджено екстремальні значення по стовпцях набору даних. Цей процес допоміг виявити максимальні та мінімальні значення для кожного стовпця, що забезпечило кращу підготовку до візуалізації.

**Завдання 4: Вибір типів візуалізацій**

Було визначено найбільш відповідні типи візуалізацій для представлення обраних даних, включаючи лінійні графіки, стовпчикові діаграми, гістограми та діаграми розсіювання, з метою передати різні аспекти набору даних.

**Завдання 5: Підготовка даних**

Набір даних було попередньо оброблено, включаючи виправлення помилок, фільтрацію зайвих даних та виконання необхідних трансформацій. Це забезпечило коректну візуалізацію на наступних етапах.

**Завдання 6: Базова візуалізація**

Було створено базову візуалізацію для перевірки правильності відображення даних за допомогою Matplotlib. Початкова діаграма для візуалізації однієї змінної підтвердила, що дані відображаються коректно.

**Завдання 7: Розширені візуалізації**

Реалізовано більш складні візуалізації, які включали різноманітні можливості Matplotlib для відображення складних взаємозв’язків у даних. Було експериментовано з різними стилями графіків і параметрами візуалізацій.

**Завдання 8: Декілька піддіаграм**

Було навчено створювати кілька піддіаграм на одному малюнку, що дозволило розташувати декілька візуалізацій поруч для більш інформативного порівняння різних змінних або наборів даних.

**Завдання 9: Експорт і обмін**

Було реалізовано можливість експорту візуалізацій у різних форматах, таких як PNG і SVG, а також у вигляді інтерактивних веб-додатків у форматі HTML. Це дало змогу легко зберігати та обмінюватися результатами аналізу.

**Код програмного продукту**

import matplotlib.pyplot as plt

import seaborn as sns

from labs.lab8.bll.DataFrameUtils import \*

import logging

logger = logging.getLogger(\_\_name\_\_)

def plot\_steps\_vs\_calories(activity\_source):

"""

:param activity\_source: DataFrame containing activity data with columns 'date', 'steps', 'calories', and 'weekDay'

:return: Matplotlib plot of steps versus calories, colored by day of the week

"""

activity = activity\_source.copy()

split\_date\_column(activity, "date")

plt.figure(figsize=(10, 6))

sns.scatterplot(data=activity.reset\_index(), x="steps", y="calories", hue="weekDay",

palette="viridis")

return plt

def plot\_rem\_and\_bed\_time(sleep\_source):

"""

:param sleep\_source: DataFrame containing sleep data with columns such as 'start', 'deepSleepTime', 'shallowSleepTime', 'wakeTime', and 'REMTime'

:return: matplotlib.pyplot object with a scatter plot showing the relationship between REM time and bed time

"""

sleep = sleep\_source.copy()

split\_date\_column(sleep, "start", "start")

round\_all\_columns(sleep)

scale\_hour\_minute(sleep, "startHour", "startMinute", "startHourComplex",

True, 15)

sleep['totalSleepTime'] = sleep['deepSleepTime'] + sleep['shallowSleepTime'] + sleep['wakeTime'] + sleep['REMTime']

plt.figure(figsize=(10, 6))

sns.regplot(data=sleep, x="REMTime", y="startHourComplex", scatter=False, color="blue",

label="REM by at the time of falling asleep")

sns.scatterplot(data=sleep.reset\_index(), x="REMTime", y="startHourComplex",

hue="totalSleepTime", palette="viridis", hue\_order="Total Sleep Time")

plt.title("REM Time vs. Bed Time")

plt.xlabel("REM Time (minutes)")

plt.ylabel("Bed Time (in 2 days)")

plt.legend()

return plt

def plot\_sleep\_activity\_relationships (activity\_source, sleep\_source):

"""

:param activity\_source: DataFrame containing activity data with columns such as 'date', 'steps', etc.

:param sleep\_source: DataFrame containing sleep data with columns such as 'start', 'deepSleepTime', 'shallowSleepTime', 'REMTime', 'wakeTime', etc.

:return: A matplotlib pyplot object which shows the relationship between sleep time and daily steps with visual differentiation based on naps.

"""

activity = activity\_source.copy()

sleep = sleep\_source.copy()

split\_date\_column(activity, "date")

change\_naps(sleep, "naps")

split\_date\_column(sleep, "start", "start")

columns = [("year", "startYear"), ("month", "startMonth"), ("day", "startDay")]

activity\_sleep = combine\_dataframes\_on\_columns(activity, sleep, columns)

activity\_sleep['totalSleepTime'] = activity\_sleep['deepSleepTime'] + activity\_sleep['shallowSleepTime'] + \

activity\_sleep['wakeTime'] + activity\_sleep['REMTime']

activity\_sleep = remove\_rows\_with\_empty\_cells(activity\_sleep)

remove\_columns\_list = ['weekDay', 'date', 'stop', 'start', 'startWeekDay']

remove\_columns\_by\_names(activity\_sleep, remove\_columns\_list)

round\_all\_columns(activity\_sleep)

plt.figure(figsize=(10, 6))

sns.scatterplot(data=activity\_sleep.reset\_index(), x="totalSleepTime", y="steps", legend="full",

hue="naps", palette="viridis")

sns.regplot(data=activity\_sleep[activity\_sleep["naps"] == 0], x="totalSleepTime", y="steps",

scatter=False, color="blue", label="No naps", ci=None)

sns.regplot(data=activity\_sleep[activity\_sleep["naps"] == 1], x="totalSleepTime", y="steps",

scatter=False, color="green", label="With naps", ci=None)

plt.title("Sleep Time vs. Daily Steps")

plt.xlabel("Sleep Time")

plt.ylabel("Daily Steps")

plt.legend()

return plt

def plot\_steps\_by\_date(activity\_source):

"""

:param activity\_source: DataFrame containing activity data with at least 'date' and 'steps' columns

:return: A matplotlib plot showing a scatterplot of steps over time

"""

activity = activity\_source.copy()

plt.figure(figsize=(10, 6))

activity['date'] = pd.to\_datetime(activity['date'])

sns.scatterplot(data=activity.reset\_index(), x="date", y="steps", marker='o', s = 10)

plt.title("Steps taken")

plt.xlabel("Date")

plt.ylabel("Steps")

plt.xticks(rotation=45)

plt.gca().xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%Y-%m-%d'))

plt.gca().xaxis.set\_major\_locator(plt.matplotlib.dates.AutoDateLocator())

return plt

def plot\_steps\_by\_years(activity\_source):

"""

:param activity\_source: DataFrame containing activity data with at least 'date' and 'steps' columns.

:return: A Matplotlib figure object with subplots representing steps taken in each year.

"""

activity = activity\_source.copy()

activity['date'] = pd.to\_datetime(activity['date'])

activity['year'] = activity['date'].dt.year

years = activity['year'].unique()

num\_years = len(years)

fig, axs = plt.subplots(num\_years, 1, figsize=(10, 6 \* num\_years))

for i, year in enumerate(years):

ax = axs[i] if num\_years > 1 else axs

year\_data = activity[activity['year'] == year]

sns.scatterplot(data=year\_data.reset\_index(), x="date", y="steps", marker='o', s=10, ax=ax)

ax.set\_title(f"Steps taken in {year}")

ax.set\_xlabel("Date")

ax.set\_ylabel("Steps")

ax.tick\_params(axis='x', rotation=45)

ax.xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%Y-%m'))

ax.set\_xlim(pd.Timestamp(f'{year}-01-01'), pd.Timestamp(f'{year}-12-31'))

plt.tight\_layout()

return plt

def plot\_sleep\_duration(activity\_source, sleep\_source):

"""

:param activity\_source: DataFrame containing activity data with columns such as date, distance, and runDistance.

:param sleep\_source: DataFrame containing sleep data with columns such as start, stop, deepSleepTime, shallowSleepTime, wakeTime, and REMTime.

:return: Matplotlib pyplot object showing a plot of sleep duration by date with median inertia.

"""

activity = activity\_source.copy()

sleep = sleep\_source.copy()

split\_date\_column(activity, "date")

change\_naps(sleep, "naps")

split\_date\_column(sleep, "start", "start")

split\_date\_column(sleep, "stop", "stop")

remove\_column\_by\_name(sleep, "stopYear")

remove\_column\_by\_name(sleep, "stopMonth")

columns = [("year", "startYear"), ("month", "startMonth"), ("day", "startDay")]

activity\_sleep = combine\_dataframes\_on\_columns(activity, sleep, columns)

remove\_columns\_list = ['Hour', 'Minute', 'stopMonth', 'stopMonth', 'distance', 'runDistance']

remove\_columns\_by\_names(activity\_sleep, remove\_columns\_list)

activity\_sleep['totalSleepTime'] = activity\_sleep['deepSleepTime'] + activity\_sleep['shallowSleepTime'] + \

activity\_sleep['wakeTime'] + activity\_sleep['REMTime']

activity\_sleep = remove\_rows\_with\_empty\_cells(activity\_sleep)

round\_all\_columns(activity\_sleep)

scale\_hour\_minute(activity\_sleep, "startHour", "startMinute", "startHourComplex",

True, 15)

scale\_hour\_minute(activity\_sleep, "stopHour", "stopMinute", "stopHourComplex",

True, 15)

plt.figure(figsize=(10, 6))

activity\_sleep['date'] = pd.to\_datetime(activity\_sleep['date'])

start\_median\_inertia = activity\_sleep['startHourComplex'].rolling(window=7, min\_periods=1).median()

stop\_median\_inertia = activity\_sleep['stopHourComplex'].rolling(window=7, min\_periods=1).median()

plt.fill\_between(x=activity\_sleep['date'], y1=start\_median\_inertia, y2=stop\_median\_inertia,

color="skyblue", alpha=0.4, label="Sleep Duration Median (Inertia)")

plt.title("Sleep Duration by Date (with Median Inertia)")

plt.xlabel("Date")

plt.ylabel("Time (in 2 days combined)")

plt.xticks(rotation=45)

plt.legend()

plt.gca().xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%Y-%m-%d'))

plt.gca().xaxis.set\_major\_locator(plt.matplotlib.dates.AutoDateLocator())

explanation\_text = ("Hour Complex represents combined hours over approximately 48-hour windows.\n"

"The graph utilizes a rolling median with a window of 7 days to smooth out short-term fluctuations.")

plt.figtext(0.5, -0.1, explanation\_text, wrap=True, horizontalalignment='center', fontsize=10)

return plt

def plot\_correlation\_heatmap(activity\_source, sleep\_source):

"""

:param activity\_source: DataFrame that contains activity data with columns such as 'date', 'Hour', 'distance', etc.

:param sleep\_source: DataFrame that contains sleep data with columns such as 'start', 'stop', 'naps', 'deepSleepTime', etc.

:return: The plotted correlation heatmap as a Matplotlib object.

"""

activity = activity\_source.copy()

sleep = sleep\_source.copy()

split\_date\_column(activity, "date")

change\_naps(sleep, "naps")

split\_date\_column(sleep, "start", "start")

split\_date\_column(sleep, "stop", "stop")

columns = [("year", "startYear"), ("month", "startMonth"), ("day", "startDay")]

activity\_sleep = combine\_dataframes\_on\_columns(activity, sleep, columns)

remove\_columns\_list = ['naps','Hour',"stopYear",'stopMonth', 'Minute', 'stopMonth', 'stopMonth', 'distance', 'runDistance', "weekDay", "date", "stopWeekDay", "startWeekDay"]

remove\_columns\_by\_names(activity\_sleep, remove\_columns\_list)

activity\_sleep['totalSleepTime'] = activity\_sleep['deepSleepTime'] + activity\_sleep['shallowSleepTime'] + \

activity\_sleep['wakeTime'] + activity\_sleep['REMTime']

activity\_sleep = remove\_rows\_with\_empty\_cells(activity\_sleep)

round\_all\_columns(activity\_sleep)

scale\_hour\_minute(activity\_sleep, "startHour", "startMinute", "startHourComplex",

True, 15)

scale\_hour\_minute(activity\_sleep, "stopHour", "stopMinute", "stopHourComplex",

True, 15)

plt.figure(figsize=(12, 10))

heatmap = sns.heatmap(activity\_sleep.corr(), annot=True, fmt='.2f', xticklabels=True, yticklabels=True,

linewidth=.1, vmin=0, vmax=1, cmap='coolwarm')

heatmap.xaxis.tick\_top()

plt.title("Correlation Heatmap")

plt.xlabel("Features")

plt.ylabel("Features")

return plt

def plot\_sleep\_patterns\_by\_day(sleep\_source):

"""

:param sleep\_source: A pandas DataFrame containing sleep data with columns for 'date', 'deepSleepTime', 'shallowSleepTime', 'wakeTime', and 'REMTime'.

:return: A Matplotlib plot object showing a boxplot of total sleep time by day of the week.

"""

sleep = sleep\_source.copy()

split\_date\_column(sleep, "date")

sleep['totalSleepTime'] = sleep['deepSleepTime'] + sleep['shallowSleepTime'] + sleep['wakeTime'] + sleep['REMTime']

plt.figure(figsize=(10, 6))

sns.boxplot(x='weekDay', y='totalSleepTime', data=sleep)

plt.title('Sleep Patterns by Day of the Week')

plt.xlabel('Day of Week')

plt.ylabel('Total Sleep Time (minutes)')

return plt

def plot\_sleep\_phases\_distribution(sleep\_source):

"""

:param sleep\_source: DataFrame containing sleep data with columns 'deepSleepTime', 'shallowSleepTime', 'wakeTime', and 'REMTime'

:return: Matplotlib pie plot showing the average distribution of different sleep phases

"""

sleep = sleep\_source.copy()

sleep\_phases = sleep[['deepSleepTime', 'shallowSleepTime', 'wakeTime', 'REMTime']].mean()

plt.figure(figsize=(8, 8))

sleep\_phases.plot.pie(autopct='%1.1f%%', startangle=90)

plt.title('Average Sleep Phases Distribution')

plt.ylabel('')

return plt

def plot\_nap\_days\_per\_month(sleep\_source):

"""

:param sleep\_source: Pandas DataFrame containing sleep data, where each row represents sleep information, including start and stop times and nap details.

:return: A matplotlib plot object showing the count of days with and without naps for each month.

"""

sleep = sleep\_source.copy()

split\_date\_column(sleep, "start", "start")

remove\_column\_by\_name(sleep, "stopYear")

remove\_column\_by\_name(sleep, "stopMonth")

columns = [("year", "startYear"), ("month", "startMonth"), ("day", "startDay")]

sleep['napDays'] = sleep['naps'].apply(lambda x: "Yes" if x else "No")

activity\_sleep = remove\_rows\_with\_empty\_cells(sleep)

plt.figure(figsize=(10, 6))

ax = sns.countplot(x=activity\_sleep['startMonth'], hue=activity\_sleep['napDays'])

ax.set\_title("Days with naps")

ax.set(xlabel='Month', ylabel='Naps')

ax.legend\_.remove()

plt.gca().xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%M'))

plt.gca().xaxis.set\_major\_locator(plt.matplotlib.dates.AutoDateLocator())

return plt

def plot\_monthly\_activity\_summary(activity):

"""

:param activity: A pandas DataFrame containing at least 'date' and 'steps' columns, where 'date' represents the date of the activity and 'steps' represents the number of steps taken on that date.

:return: A matplotlib 'plt' object with a barplot representation of total steps taken each month.

"""

activity['date'] = pd.to\_datetime(activity['date'])

monthly\_steps = activity.groupby(activity['date'].dt.to\_period('M'))['steps'].sum()

monthly\_steps.index = monthly\_steps.index.astype(str)

plt.figure(figsize=(12, 6))

sns.barplot(x=monthly\_steps.index, y=monthly\_steps.values, palette="coolwarm")

plt.title('Total Steps Taken Each Month')

plt.xlabel('Month')

plt.ylabel('Total Steps')

plt.gca().xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%Y-%m-%d'))

plt.gca().xaxis.set\_major\_locator(plt.matplotlib.dates.AutoDateLocator())

plt.xticks(rotation=45)

return plt

def plot\_monthly\_sleep\_patterns(sleep):

"""

:param sleep: DataFrame containing sleep data with columns 'start', 'stop', 'deepSleepTime', 'shallowSleepTime', 'wakeTime', and 'REMTime'.

:return: The generated plot as a matplotlib figure object.

"""

sleep['start'] = pd.to\_datetime(sleep['start'])

sleep['stop'] = pd.to\_datetime(sleep['stop'])

sleep['month'] = sleep['start'].dt.month

sleep['totalSleepTime'] = sleep['deepSleepTime'] + sleep['shallowSleepTime'] + sleep['wakeTime'] + sleep['REMTime']

plt.figure(figsize=(12, 6))

sns.boxplot(x='month', y='totalSleepTime', data=sleep)

plt.title('Monthly Sleep Patterns')

plt.xlabel('Month')

plt.ylabel('Total Sleep Time (minutes)')

return plt

def plot\_rem\_sleep\_vs\_steps(activity, sleep):

"""

:param activity: DataFrame containing activity data, presumed to have columns 'date' and 'steps'.

:param sleep: DataFrame containing sleep data, presumed to have columns 'start', 'stop', and 'REMTime'.

:return: Matplotlib plot object displaying the correlation between REM sleep time and steps taken.

"""

sleep['start'] = pd.to\_datetime(sleep['start'])

sleep['stop'] = pd.to\_datetime(sleep['stop'])

sleep['date'] = sleep['start'].dt.date

activity['date'] = pd.to\_datetime(activity['date']).dt.date

combined\_df = pd.merge(sleep, activity, on='date', how='inner')

split\_date\_column(combined\_df, "date")

remove\_rows\_with\_empty\_cells(combined\_df)

plt.figure(figsize=(10, 6))

sns.scatterplot(data=combined\_df, x='REMTime', y='steps', hue='weekDay', palette='viridis')

sns.regplot(data=combined\_df, x='REMTime', y='steps', scatter=False, color='blue')

plt.title('Correlation Between REM Sleep Time and Steps Taken')

plt.xlabel('REM Sleep Time (minutes)')

plt.ylabel('Steps')

return plt

def plot\_calories\_vs\_sleep\_time(activity\_source, sleep\_source):

"""

:param activity\_source: DataFrame containing activity data.

:param sleep\_source: DataFrame containing sleep data.

:return: A matplotlib plot object showing a scatter plot of calories burned vs. total sleep time.

"""

activity = activity\_source.copy()

sleep = sleep\_source.copy()

split\_date\_column(activity, "date")

change\_naps(sleep, "naps")

split\_date\_column(sleep, "start", "start")

split\_date\_column(sleep, "stop", "stop")

remove\_column\_by\_name(sleep, "stopYear")

remove\_column\_by\_name(sleep, "stopMonth")

columns = [("year", "startYear"), ("month", "startMonth"), ("day", "startDay")]

activity\_sleep = combine\_dataframes\_on\_columns(activity, sleep, columns)

remove\_columns\_list = ['Hour', 'Minute', 'stopMonth', 'stopMonth', 'distance', 'runDistance']

remove\_columns\_by\_names(activity\_sleep, remove\_columns\_list)

activity\_sleep['totalSleepTime'] = activity\_sleep['deepSleepTime'] + activity\_sleep['shallowSleepTime'] + \

activity\_sleep['wakeTime'] + activity\_sleep['REMTime']

activity\_sleep = remove\_rows\_with\_empty\_cells(activity\_sleep)

round\_all\_columns(activity\_sleep)

scale\_hour\_minute(activity\_sleep, "startHour", "startMinute", "startHourComplex",

True, 15)

scale\_hour\_minute(activity\_sleep, "stopHour", "stopMinute", "stopHourComplex",

True, 15)

plt.figure(figsize=(10, 6))

sns.scatterplot(x='totalSleepTime', y='calories', data=activity\_sleep)

plt.title('Calories Burned vs. Total Sleep Time')

plt.xlabel('Total Sleep Time (minutes)')

plt.ylabel('Calories Burned')

plt.gca().xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%Y-%m-%d'))

plt.gca().xaxis.set\_major\_locator(plt.matplotlib.dates.AutoDateLocator())

return plt

import pandas as pd

from shared.services.relative\_to\_absolute\_path import absolute

from labs.lab8.dal.SettingsModel import SettingsModel

from config.settings\_paths import settings\_path\_lab8

from labs.lab8.bll.Controller import Controller

from labs.lab8.ui.UserInterface import UserInterface

import logging

def set\_up\_logging(file\_path):

"""

:param file\_path: The file path where the log file will be created.

:return: None

"""

# Create a file handler

file\_handler = logging.FileHandler(file\_path)

file\_handler.setLevel(logging.INFO)

# Create a stream handler

stream\_handler = logging.StreamHandler()

stream\_handler.setLevel(logging.WARNING)

# Set the logging format

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

file\_handler.setFormatter(formatter)

stream\_handler.setFormatter(formatter)

# Configure the root logger

logging.basicConfig(level=logging.DEBUG,

handlers=[file\_handler, stream\_handler])

def main():

"""

Loads settings, activity data, and sleep data from file paths, initializes the controller, sets up logging, and displays the user interface.

:return: None

"""

settings = SettingsModel(settings\_path\_lab8)

relative\_activity\_path = settings.get\_activity\_file\_path()

relative\_sleep\_path = settings.get\_sleep\_file\_path()

activity\_path = absolute(relative\_activity\_path)

sleep\_path = absolute(relative\_sleep\_path)

activity = pd.read\_csv(activity\_path)

sleep = pd.read\_csv(sleep\_path)

controller = Controller(settings, activity, sleep)

logger\_path = controller.get\_logger\_path()

set\_up\_logging(logger\_path)

user\_interface = UserInterface(controller)

user\_interface.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Висновки**

Завершення цих завдань дозволило створити багатофункціональний додаток для візуалізації CSV-наборів даних за допомогою Matplotlib. Проект надав цінний досвід в області візуалізації даних та обробки різноманітних наборів інформації.