**Лабораторна робота 1**

**Агрегація, обробка пропусків та візуалізація даних пакетами Python**

**1. Мета роботи**

Метою лабораторної роботи є отримання практичних навичок у роботі з raw data, використовуючи пакети jupyter, pandas, seaborn.

**2. Контекст**

У дата сеті знаходяться 31 набір даних з іменами nyt1.csv, nyt2.csv, ..., nyt31.csv.

Кожен із них демонструє один (симульований) день показів оголошень та переходів по них, записаних на головній сторінці газети The New York Times у травні 2012 року. Кожен рядок представляє одного користувача. Існує п'ять стовпців: вік, стать (0 = жінка, 1 = чоловік), кількість показів, кількість переходів та статус авторизації.

**3. Завдання до виконання**

* Створіть нову змінну age\_group, яка агрегує користувачів як <18, 18–24, 25–34, 35–44, 45–54, 55–64 та 65+.
* Зафіксуйте на діаграмі кількість показів та показник переходів (CTR = #clicks/#inmpressions) для цих шести вікових категорій.
* Вивчіть дані та проведіть візуальні та кількісні порівняння між сегментами користувачів/демографічними групами (наприклад, чоловіки старше 18 років у порівнянні з жінками старше 18 років або авторизовані та неавторизовані користувачі).
* Створіть метрики/вимірювання/статистику, які підсумовують дані. Приклади можливих метрик включають CTR, квантил, середнє значення, медіану, дисперсію та максимальне значення. Ці показники потрібно розрахувати за різними сегментами користувачів. Подумайте про елементи, які важливо відстежувати з часом - що стискає дані, але, як і раніше, захоплює поведінку користувача.
* Результати статистичного дослідження подати у вигляді результуючого ДатаФрейма (одного), дивлячись на який можна зрозуміти і порівнювати дані за віковими підкатегоріями.
* Опишіть та інтерпретуйте будь-які закономірності, які знайдете.
* Завантажити файл ipynb з виконаними завданнями на git в окрему папку з відповідною назвою лабораторної роботи

**4. Необхідні пакети**

pandas, seaborn, jupyter

**4. Супровідні матеріали**

На відміну від модуля *time*, що призначений в першу чергу для визначення поточного часу, існує також модуль *datetime*, який призначений для виконання певних маніпуляцій, логічних і арифметичні операції, порівнянь, обробки та ін. з даними, що безпосередньо зберігають значення часу та дати. Модуль datetime в Python має 5 основних підмодулів, кожен з яких реалізує відповідний клас об’єктів:

* + - *date* – для роботи з об’єктами-датами;
    - *time* – для маніпулювання об’єктами які представляють час;
    - *datetime* – це комбінація дати та часу;
    - *timedelta* – дозволяє працювати з поняттям «тривалість часу», або

«інтервал»;

* + - *tzinfo* – дозволяє працювати поняттям «часовий пояс».

З переліку видно – але ми звернемо на це спеціальну увагу – що дані для представлення інформації щодо дати та часу можуть мати різну семантику. Одні з них вказують певну дату та/або час, причому таке представлення може використовуватися у різних форматах, наприклад *«July 4» ,*

*«March 8, 2023 22:00», «31 December 2023, 23:59:59», «07/23/2023»,*

*«23/07/2023»* тощо. Такі дані ще називають «часовою міткою» або «міткою часу». Інші дані представляють величину проміжку між деякими моментами часу, наприклад *«1 year 3 month 5 days», «24 hours, 17 minutes, 56 seconds, 268 miliseconds», «3 weaks, 3 days, 3 hours, 3 minutes»* тощо. Про такі дані говорять як про «інтервали». При обробці інформації можна, наприклад, до певної дати (мітки часу) додати «інтервал» і отримати іншу дату («мітку часу»). Можна додати один інтервал до іншого і отримати новий (більший за протяжністю) інтервал. А от скласти дві мітки часу семантично сенсу не має. Отже, в одному виразі можуть бути присутні дані обох типів, а реалізація модулю *datetime* як раз і забезпечує їх коректну обробку.

Імпортуємо модуль *datetime* і створимо об’єкт дати (екземпляр об’єкту класу *date*):

import datetime as dt st\_date=dt.date(year=2023, month=11, day=10) print(st\_date)

>>>>

2023-11-10

або скорочено, притримуючись порядку опису параметрів:

st\_date=dt.date(2023,11,10)

Зазначимо, що отримане значення представлене в стандартному для модулю datetime форматі, а також, що це незмінюваний об’єкт (термінах опису об’єктів в *Python*). Спроба ввести некоректну дату, наприклад *dt.date(2023, 26, 3)* призведе до ситуації виключення типу *ValueError*.

Стровремо об’єкт класу time (Зауваження – не слід путати об’єкти класу time модулю datetime і об’єкти, з якими працюють за допомогою функцій модулю time).

st\_time= dt.time(hour=12, minute=45, second=16, microsecond=257) print(st\_time)

>>>>

12:45:16.000257

Цікаво зазирнути в менеджер змінних IDE і подивитися, на скільки сильно різняться представлення створених вище об’єктів *st\_date* та *st\_time*.

В разі, якщо виникає потреба представити мітку часу з точністю до дати, години, хвилини, секунди та мілісекунди, використовують об’єкти окремого класу *datetime* модулю *datetime*, який по суті є класом-наступником від класів *date* та *time*.

st\_moment=dt.datetime(year=2023, month=11, day=10, hour=13,second=55) print(st\_moment)

>>>>

2023-11-10 13:00:55

Якщо при створенні об’єкту класу *datetime* ми задамо не всі параметри, то об’єкт все рівно буде створений.

st\_moment1=dt.datetime(year=2023, month=11, day=10) print(st\_moment1)

>>>>

2023-11-10 00:00:00

Зрозуміло, що виключати параметри з опису можна тільки «послідовно», починаючи з останнього. Тобто – не можна пропустити тільки параметр *year*, або, наприклад, параметр *month*. Такі спроби призведуть до виникнення виключень типу «*TypeError*».

Окрема функція модулю *datetime*, а саме *datetime.combine()* створює об’єкт

* «комбінацію» з часток, що відповідають даті та часу:

st\_moment2=dt.datetime.combine(st\_date, st\_time)

>>>>

2023-11-10 12:45:16.000257

Звісно, можна і навпаки:

st\_date2=st\_moment2.date() #відокремлюємо дату.

st\_time2=st\_moment2.time() #відокремлюємо час. print(st\_date2,’\n’,st\_time2)

>>>>

2023-11-10

12:45:16.000257

Коректність подальшої роботи з утвореними змінними забезпечується тим, що об’єкт *st\_date2* створюється як об’єкт класу *date*, а об’єкт *st\_time2* – як об’єкт класу *time*.

Група методів дозволяє виділяти окремі компоненти мітки часу:

print(‘tuple:’, st\_moment1.timetuple()) print(‘Рік:’, st\_moment1.year) print(‘Місяць :’, st\_moment1.month) print(‘День :’, st\_moment1.day)

>>>>

tuple: time.struct\_time(tm\_year=2023, tm\_mon=11, tm\_mday=10, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=3, tm\_yday=314, tm\_isdst=-1)

Рік: 2023

Місяць : 11

День : 10

Отримати описові характеристики даних, що є часовими мітками, можна за допомогою методів:

print(‘Номер дня в році:’, st\_moment1.()) print(‘Номер дня в тижні:’,st\_moment1.weekday()) print(‘Номер дня в тижні:’,st\_date.weekday())

>>>>

Номер дня в році: 738469 Номер дня в тижні: 3 Номер дня в тижні: 3

(При визначенні номеру для в тижні приймається, що понеділку відповідає значення 0, вівторку – 1, тощо).

В разі, якщо потрібно отримати значення поточного часу, в *datetime*

використовується функції *now()* або *today()*.

now = dt.datetime.now() print(now)

>>>>

2023-11-19 18:47:19.518040

Відмінність між *datetime.now()* та *datetime.today()* полягає в тому, що перша з них в разі потреби дає можливість працювати не тільки з локальним часом, але використовувати так званий «час за Грінвічем».

Відомий нам метод *strftime()* має реалізацію і для об’єктів модулю *datetime*:

tod = dt.datetime.today() tod.strftime("%d.%m.%Y")

>>>> ‘19.11.2023’

tod.strftime("%H:%M:%S")

>>>> ‘19:42:45’

Для зміни певних компонент опису дат та часу, представлених в форматі

*datetime*, використовується метод *datetime.replace()*. Розглянемо приклад:

time\_mk = dt.datetime(2015, 3, 21) print(time\_mk)

time\_mk2 = time\_mk.replace(year = 2023) print(time\_mk2) time\_mk2=time\_mk2.replace(month=time\_mk2.month+5) print(time\_mk2) time\_mk2=time\_mk2.replace(hour=3,second=54) print(time\_mk2)

>>>>

2015-03-21 00:00:00

2023-03-21 00:00:00

2023-08-21 00:00:00

2023-08-21 03:00:54

В прикладі показано, як можна вирахувати нове значення параметру на основі старого його значення (*time\_mk2.month+5*), однак такий спосіб досить кропіткий і потребує уваги програміста. Оскільки діапазони атрибутів визначаються відповідними природніми кордонами (наприклад – значення параметра *month* має лежати в діапазоні від 1 до 12, *hour –* від 0 до 23, *minute* – від 0 до 59 тощо), відповідальність за коректність змін покладається на програміста.

Метод *timestamp()* дозволяє перейти від об’єкту *datetime* до його представлення у вигляді секунд з початку епохи.

st\_moment1=dt.datetime(year=2023, month=11, day=10, hour=13,second=55) print(st\_moment1)

sys\_time=st\_moment1.timestamp() print(sys\_time)

>>>>

2023-11-10 13:00:55

1668078055.0

Зворотня дія виконується за допомогою функції *fromtimestamp()*:

st\_moment2=dt.datetime.fromtimestamp(sys\_time) print(st\_moment2)

>>>>

2023-11-10 13:00:55

Одна з найбільш поширених задач при роботі з часом та даними – знаходження інтервалу між подіями, які відбувалися в певні моменти часу. Користуючись класом *datetime*, можна легко знаходити різницю між двома різними датами. Наступний приклад демонструє створення двох об’єктів. Виконуючи просту операцію знаходження різниці (віднімання) між об’єктами time\_mk1 та time\_mk2, створюється новий об’єкт time\_int.

time\_mk1 = dt.datetime.now() time\_mk2 = dt.datetime(2015, 3, 21) time\_int = time\_mk1 – time\_mk2 print(time\_int)

>>>>

2800 days, 19:49:13.420862

Створений об’єкт *time\_int* є об’єктом класу *timedelta*, що реалізований засобами модулю *datetime*. Екземпляр об’єкту *datetime.timedelta* зберігає значення інтервалу як поєднання *days*, *seconds* і *microseconds*, а решта переданих параметрів конструктор конвертує в ці одиниці, а саме:

* + *millisecond* перетворюється на 1000 *microseconds*.
  + *minute* перетворюється на 60 *seconds*.
  + *hour* перетворюється на 3600 *seconds*.
  + *week* перетворюється на 7 *days*.

Пояснимо це на прикладі. Явним чином створимо об’єкт класу

*datetime.timedelta*:

delta = dt.timedelta( days=50, seconds=27, microseconds=10, milliseconds=29000, minutes=5, hours=8,

weeks=2

)

print(delta)

>>>>

64 days, 8:05:56.000010

При цьому слід по-перше, звернути увагу, що значення кількості днів в інтервалі були перераховані (50 днів+ 2 тижні=50+2\*7=64 дні). По-друге, незважаючи на те, що функція *print()* для таких об’єктів робить відповідний перерахунок, в дійсності об’єкт зберігає лише три атрибути:

datetime.timedelta(days=64, seconds=29156, microseconds=10)

Отримати значення цих атрибутів можна звичайним чином:

days\_int= delta.days seconds\_int= delta.seconds

microseconds\_int= delta.microseconds print(days\_int)

print(seconds\_int) print(microseconds\_int)

>>>> 64

29156

10

Отримані таким чином компоненти опису інтервалу часу представлені в вигляді змінних типу *int* і можуть використовуватися за звичайними правилами роботи з даними цього типу в мові програмування *Python*.

Також можна об’єкти-інтервали додавати один до одного, додавати об’єкти-інтервали до об’єктів часових міток (або віднімати від других перші), формуючи тим самим нові об’єкти. У наступному прикладі показано, як отримати нову дату маючи початкову дату та значення інтервалу до другої дати.

date1 = dt.datetime(2023, 12, 5)

delta1= dt.timedelta(weeks=8, days= 8, hours=2, minutes=5, seconds=17) print(date1)

date2 = date1 + delta1 print(date2)

>>>>

2023-12-05 00:00:00

202-02-07 02:05:17

Зверніть увагу, як коректно перераховані роки та місяці. Навіть якщо дані задані так, що прийдеться зважати на високосність року, відмінність кількості днів в місяці тощо, відповідальність за правильність перерахунків модуль *datetime* бере на себе.

Окрім вказаних, над об’єктами класу *timedelta* можуть виконуватися операції множення на ціле число (що збільшує інтервал в відповідну кількість разів) та навіть на дійсне число (що призводить до округлення результату), ділення інтервалу на інтервал, одномісна операція «мінус», тобто зміна знаків значень компонентів опису дати на протилежний), логічні операції порівняння тощо. До речі, якщо компоненти значень об’єкту від’ємні, це означає, що при додаванні такого інтервалу до певної дати, результуюча дата виявиться менша за першу, тобто – буде звернута «у минуле».

Нарешті метод *total\_seconds()* дозволяє отримати значення інтервалу в секундах:

delta1= dt.timedelta(weeks=8, days= 8, hours=2, minutes=5, seconds=17) delta\_sec=delta1.total\_seconds()

print(delta\_sec)

>>>> 5537117.0

На *Python* існує кілька бібліотек для читання і запису *JSОN*-даних. Ми розглянемо роботу вбудованими в *pandas* методами, що спрощують роботу з *JSON*-файлами при використанні *DataFrame* в якості відповідного об’єкту застосунку.

При роботі з *JSON* об’єкти, що використовуються програмою на мові *Python,* упаковуються у байтову послідовність, а сама операція такої упаковки носить назву *серіалізації*. А розпакування байтової послідовності в об’єкти мови *Python*, приведення послідовності байт назад до типів та структур *Python* — операцією *десеріалізації*. Саме послідовність байт забезпечує необхідну незалежність даних, що передаються, як від середовища передачі, так і від особливостей реалізації застосунків, що обмінюються інформацією за допомогою серелізованих потоків байтів.

# Імпорт бібліотек

import pandas as pd

import seaborn as sns

# Завантаження даних

df = pd.read\_csv("data/nyt1.csv")

# Обробка даних

df["age\_group"] = pd.cut(df["age"], [0, 18, 25, 35, 45, 55, 65, 99], labels=["<18", "18–24", "25–34", "35–44", "45–54", "55–64", "65+"])

# Візуалізація даних

sns.barplot(x="age\_group", y="impressions", data=df)

plt.title("Кількість показів за віковими категоріями")

plt.show()

# Імпорт бібліотек

import numpy as np

import matplotlib.pyplot as plt

# Генерація даних

x = np.linspace(0, 10, 100)

y = x\*\*2

# Візуалізація даних

plt.plot(x, y)

plt.title("Графік функції y=x^2")

plt.show()