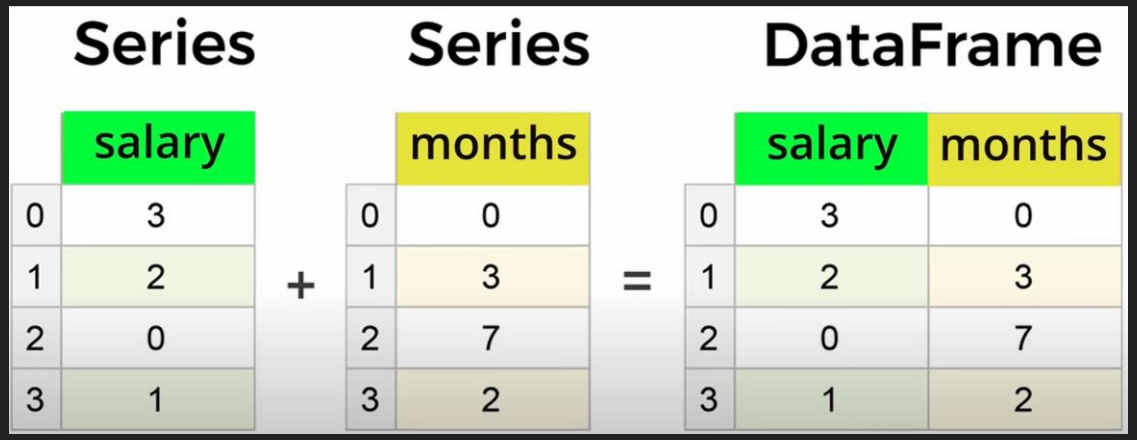
ММП 2025/2026

Викладач Канцедал Георгій Олегович

Дерево класів для організації своїх структур даних.

Бібліотека pandas у Python використовує дерево класів для організації своїх структур даних. Основними класами є:
pd.Series – це одномірний масив із мітками (індексами), схожий на стовпець у таблиці або на словник Python, де значення мають ключі (індекси).
pd.DataFrame – це двовимірна таблиця, що складається з кількох об'єктів Series.
pd.Index – клас, що використовується для управління індексами в Series і DataFrame.



Приклад

Створим DataFrame

Series є базовою одиницею DataFrame. Кожен стовпець DataFrame – це Series. Index є частиною Series та DataFrame, допомагаючи отримувати доступ до даних.

```python
import pandas as pd

# Створення Series
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(s, "\n")

# Створення DataFrame на основі Series
df = pd.DataFrame({"Колонка1": s, "Колонка2": [100, 200, 300]})
print(df)
```

[6] ✓ 0.0s

```
a    10
b    20
c    30
dtype: int64

   Колонка1  Колонка2
a        10       100
b        20       200
c        30       300
```

Розглянем властивості pd.Series та pd.DataFrame

Series – це одномірний об'єкт, схожий на список

Властивість	Опис
s.index	Повертає індекси Series
s.values	Масив значень Series
s.dtype	Тип даних у Series
s.size	Кількість елементів
s.shape	Розмірність ((n,) для Series)
s.name	Ім'я Series
s.isna()	Повертає True для пропущених значень
s.hasnans	True, якщо є NaN
s.nunique()	Кількість унікальних значень
s.empty	True, якщо Series порожній

```python
import pandas as pd

s = pd.Series([10, 20, 30, None], index=['a', 'b', 'c', 'd'])
print(s.index)   # Index(['a', 'b', 'c', 'd'])
print(s.dtype)   # float64 (через NaN)
print(s.hasnans) # True
```

✓ 0.4s

```
Index(['a', 'b', 'c', 'd'], dtype='object')
float64
True
```

Розглянем властивості pd.Series та pd.DataFrame

```python
df = pd.DataFrame({
    "Ім'я": ["Анна", "Іван", "Олег"],
    "Вік": [25, 30, 35],
    "Зарплата": [50000, 60000, None]
})

print(df.columns)  # Index(['Ім'я', 'Вік', 'Зарплата'])
print(df.dtypes)   # Дізнатися типи даних
print(df.shape)    # (3, 3)
```

[2] ✓ 0.0s

```
Index(['Ім'я', 'Вік', 'Зарплата'], dtype='object')
Ім'я         object
Вік           int64
Зарплата    float64
dtype: object
(3, 3)
```

DataFrame – це двовимірна таблиця (сукупність Series)

Властивість	Опис
df.index	Індекси рядків
df.columns	Імена стовпців
df.dtypes	Типи даних у кожному стовпці
df.values	Масив значень (numpy.ndarray)
df.shape	Розмірність (рядки, стовпці)
df.size	Загальна кількість елементів (рядки × стовпці)
df.T	Транспонування (поміняти місцями рядки та стовпці)
df.isna()	Повертає True для NaN
df.empty	True, якщо DataFrame порожній

Висновок

Series – це 1D-структура (один стовпець), DataFrame – 2D (таблиця).

Багато властивостей схожі (index, values, size).

У DataFrame є columns, а в Series – name.

Функції швидкого аналізу

Функція describe() в pandas для швидкого аналізу даних
Функція df.describe() дозволяє швидко отримати основні статистичні характеристики числових даних у DataFrame. Вона обчислює такі показники:

count – кількість значень (без NaN),

mean – середнє значення,

std – стандартне відхилення,

min – мінімальне значення,

25%, 50% (медіана), 75% – квартилі,

max – максимальне значення.

Приклад використання describe()

```python
import pandas as pd

# Створення DataFrame
df = pd.DataFrame({
    "Вік": [25, 30, 35, 40, 29],
    "Зарплата": [50000, 60000, 70000, 80000, None],  # Є NaN
    "Оцінка": [4.5, 3.8, 4.2, 4.9, 4.0]
})

# Використання describe()
print(df.describe())
```

✓ 0.0s

```
             Вік      Зарплата    Оцінка
count   5.00000      4.000000  5.000000
mean   31.80000  65000.000000  4.280000
std     5.80517  12909.944487  0.432435
min    25.00000  50000.000000  3.800000
25%    29.00000  57500.000000  4.000000
50%    30.00000  65000.000000  4.200000
75%    35.00000  72500.000000  4.500000
max    40.00000  80000.000000  4.900000
```

Особливості describe()

```
print(df["Вік"].describe())
```

✓ 0.0s

```
count     5.00000
mean     31.80000
std       5.80517
min      25.00000
25%      29.00000
50%      30.00000
75%      35.00000
max      40.00000
Name: Вік, dtype: float64
```

Особливості describe()
Працює лише з числовими даними (типи int і float).
Щоб включити об'єктні (str) або категорійні дані, використовуйте df.describe(include="all").

Огляд describe() для рядкових даних

```python
df2 = pd.DataFrame({
    "Ім'я": ["Анна", "Іван", "Анна", "Олег", "Іван", "Анна"]
})

print(df2.describe(include="object"))
```

✓ 0.0s

```
        Ім'я
count       6
unique      3
top      Анна
freq        3
```

Якщо DataFrame містить текстові (категорійні) дані, то describe() поверне:

count – кількість значень (без NaN),

unique – кількість унікальних значень,

top – найчастіше значення,

freq – частота цього значення.

Для швидкого аналізу датафрейму в Data Science використовуються такі функції:

Огляд структури даних

freq 3

```
df.info()  # Загальна інформація про датафрейм
df.head()  # Перші 5 рядків
df.tail()  # Останні 5 рядків
df.shape   # Кількість рядків і стовпців
df.columns # Назви стовпців
```

[ ]

Основні статистики

```
df.describe()  # Статистика для числових стовпців
df.describe(include='all')  # Статистика для всіх типів даних
df.nunique()  # Кількість унікальних значень
df.isnull().sum()  # Кількість пропущених значень
```

```
df.corr()  # Кореляція між числовими змінними
```

Фільтрація даних у pandas

У pandas є кілька потужних способів фільтрації даних у DataFrame:

Фільтрація через логічні умови (>, <, ==, !=, isin() тощо)

Метод filter() – використовується для вибору колонок або рядків за назвою

Метод query() – дозволяє застосовувати SQL-подібні запити

Фільтрація через логічні умови

Можна використовувати логічні оператори (>, <, ==, !=) для відбору рядків.
Приклад: Вибрати всі рядки, де Вік > 30

```python
import pandas as pd

df = pd.DataFrame({
    "Ім'я": ["Анна", "Іван", "Олег", "Марина"],
    "Вік": [25, 35, 40, 29],
    "Зарплата": [50000, 60000, 70000, 80000]
})

# Фільтр рядків, де Вік > 30
df_filtered = df[df["Вік"] > 30]
print(df_filtered)
```

✓ 0.0s

```
   Ім'я  Вік  Зарплата
1  Іван   35     60000
2  Олег   40     70000
```

Метод filter(like=…) – фільтрація за частиною назви

Метод filter() використовується для вибору колонок або рядків за збігом у назві.
Приклад: Вибрати всі колонки, які містять слово "Вік«
Приклад: Вибрати рядки за індексом

```
df_filtered = df.filter(like="Вік", axis=1)
print(df_filtered)
```

✓ 0.0s

```
   Вік
0   25
1   35
2   40
3   29
```

```
df.index = ["row1", "row2", "row3", "row4"]
df_filtered = df.filter(like="row", axis=0)
print(df_filtered)
```

✓ 0.0s

```
       Ім'я  Вік  Зарплата
row1   Анна   25     50000
row2   Іван   35     60000
row3   Олег   40     70000
row4  Марина   29     80000
```

Метод query() – SQL-подібна фільтрація

Метод query() дозволяє фільтрувати дані, використовуючи рядкові умови, схожі на SQL.

```
df_filtered = df.query("Вік == 35")
print(df_filtered)
```

✓ 0.0s

```
  Ім'я  Вік  Зарплата
1 Іван   35     60000
```

```
df_filtered = df.query("Зарплата > 50000 and Вік < 40")
print(df_filtered)
```

✓ 0.0s

```
    Ім'я  Вік  Зарплата
1   Іван   35     60000
3 Марина   29     80000
```

```
min_salary = 60000
df_filtered = df.query("Зарплата > @min_salary")
print(df_filtered)
```

✓ 0.0s

```
    Ім'я  Вік  Зарплата
2   Олег   40     70000
3 Марина   29     80000
```

Збереження та зчитування даних у pandas

У pandas є різні способи збереження та зчитування файлів, але іноді вони можуть спричиняти проблеми з пам’яттю, особливо при роботі з великими наборами даних. Давайте розглянемо це детальніше.

Збереження даних. Збереження у CSV

Збереження у CSV
index=False вимикає запис індексів у файл, що зменшує розмір файлу.

```
df.to_csv("data.csv", index=False)
```

Збереження у Excel
Працює повільніше за CSV, але підтримує кілька листів.

```
df.to_excel("data.xlsx", index=False)
```

Збереження у Parquet (економить пам’ять)
Формат Parquet значно ефективніший, ніж CSV, бо зберігає дані у стиснутому вигляді.

```
df.to_parquet("data.parquet", index=False)
```

Зчитування даних

Зчитування CSV
Може займати багато пам’яті, бо всі колонки читаються як object або float64.

Зчитування Excel
Використовує більше пам’яті та працює повільніше за CSV.

Зчитування Parquet (ефективно)
Найкращий варіант для великих файлів, бо займає менше RAM.

```
df = pd.read_csv("data.csv")



df = pd.read_excel("data.xlsx")



df = pd.read_parquet("data.parquet")
```

Оптимізація пам’яті при зчитуванні

Визначення типів даних (dtype)
Коли pandas зчитує CSV, він може автоматично використовувати float64, що займає багато пам’яті. Замість цього можна вказати dtype:

```
df = pd.read_csv("data.csv", dtype={"ID": "int32", "Категорія": "category"})
```

int32 замість int64 економить пам’ять.
category замість object пришвидшує обробку текстових даних.

Оптимізація пам’яті при зчитуванні

Використання usecols (завантажувати не всі колонки)
Якщо потрібні лише певні колонки, можна використовувати usecols:

```
df = pd.read_csv("data.csv", usecols=["ID", "Зарплата"])
```

Економить RAM, бо не завантажує непотрібні колонки.

Оптимізація пам’яті при зчитуванні

Використання chunksize (потокове зчитування)
Якщо файл великий, можна зчитувати його частинами:

```
chunks = pd.read_csv("data.csv", chunksize=10000)
for chunk in chunks:
    print(chunk.shape)  # Обробляємо кожну частину окремо
```

Це не перевантажує оперативну пам’ять.

Оптимізація пам’яті при зчитуванні

Використання low_memory=True (оптимізація RAM)
За замовчуванням pandas намагається вгадати типи даних, що може спричинити проблеми з пам’яттю.

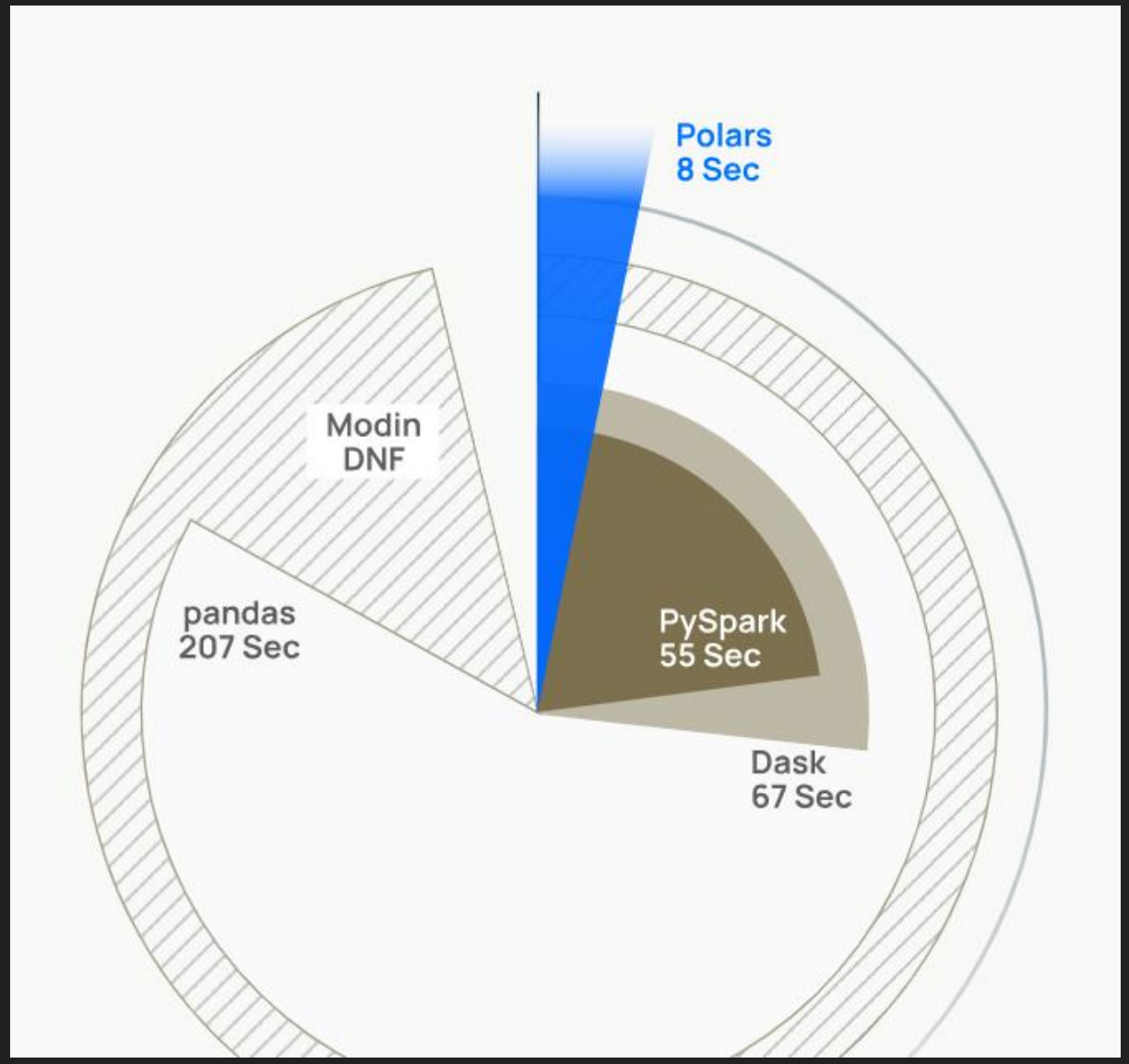
```
df = pd.read_csv("data.csv", low_memory=False)
```

Це дозволяє pandas завантажити дані одразу правильно.

Підсумовуючи

Якщо у вас великі файли, уникайте CSV та використовуйте Parquet або зчитуйте дані частинами. Оптимізація типів даних (dtype, category) значно зменшує RAM-навантаження.

Розглянем в порівнянні



Основні переваги Polars:

- Висока продуктивність
- Колончаста пам'ять
- Багатопотоковість

Основні переваги Polars:

- Зручний API
- Інтеграція з Python
- Потужні можливості обробки даних

Почнемо огляд та подальше порівняння (звертаємо увагу на масштаб датасетів)

```
import numpy as np
import polars as pl
```

```
pl.__version__
```

```
'0.20.30'
```

```
df = pl.read_csv("custom_1988_2020.csv")
```

```
df.shape
```

```
(113607321, 8)
```

```
df.head()
```

shape: (5, 8)

198801	1	103	100	000000190	0	35843	34353
i64	i64	i64	i64	i64	i64	i64	i64
198801	1	103	100	120991000	0	1590	4154
198801	1	103	100	210390900	0	4500	2565
198801	1	103	100	220890200	0	3000	757
198801	1	103	100	240220000	0	26000	40668
198801	1	103	100	250410000	0	5	8070

```
df.head().to_pandas()
```

	198801	1	103	100	000000190	0	35843	34353
0	198801	1	103	100	120991000	0	1590	4154
1	198801	1	103	100	210390900	0	4500	2565
2	198801	1	103	100	220890200	0	3000	757
3	198801	1	103	100	240220000	0	26000	40668
4	198801	1	103	100	250410000	0	5	8070

Завантажим ще

```
df2 = pl.read_csv('https://raw.githubusercontent.com/RandomFractals/chicago-crimes/main/data/crimes-2022.csv')
```

```
df2.head()
```

shape: (5, 22)

ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	Beat	District	Ward	Community Area	FBI Code
i64	str	str	str	str	str	str	str	bool	bool	i64	i64	i64	i64	str
12757446	"JF313117"	"07/08/2022 10:38:00 AM"	"087XX S WINCHESTER AVE"	"0820"	"THEFT"	"$500 AND UNDER"	"RESIDENCE"	false	true	2221	22	21	71	"06"
12755229	"JF310109"	"07/08/2022 03:21:00 AM"	"056XX N SPAULDING AVE"	"1154"	"DECEPTIVE PRACTICE"	"FINANCIAL IDENTITY THEFT $300 ...	null	false	false	1711	17	39	13	"11"
12763369	"JF320208"	"07/16/2022 10:55:00 PM"	"038XX N SHEFFIELD AVE"	"0330"	"ROBBERY"	"AGGRAVATED"	"SIDEWALK"	true	false	1923	19	46	6	"03"
12766036	"JF323691"	"07/19/2022 04:00:00 PM"	"113XX S PARNELL AVE"	"1792"	"KIDNAPPING"	"CHILD ABDUCTION / STRANGER"	"SIDEWALK"	false	false	2233	22	34	49	"26"
12758668	"JF314314"	"07/12/2022 06:30:00 AM"	"044XX W WALTON ST"	"0610"	"BURGLARY"	"FORCIBLE ENTRY"	"RESIDENCE"	false	true	1111	11	37	23	"05"

```
df2.shape
```

```
(215551, 22)
```

Попрацюємо

```
df2["District"].head(4)
```

shape: (4,)

District
i64
22
17
19
22

```
df2.get_column("District")
```

24
12
8

```
df2.select("District").to_series()
```

shape: (215_551,)

District
i64
22
17
19
22
11
...
4
1
24
12
8

Арифметичні операції

```
df2["Ward"] + 10
```

shape: (215_551,)

Ward
i64
31
49
56
44
47
...
20
52
59
37
26

```
df2["Ward"] * 2
```

shape: (215_551,)

Ward
i64
42
78
92
68
74

```
df2["Ward"].sum()
```

```
5031938
```

```
df2["Ward"].mean()
```

```
23.345618698994624
```

```
df2["Ward"] > 20
```

shape: (215_551,)

Ward
bool
true
true
true
true
true
...
false
true
true
true
false

```
df2["Ward"].gt(50)
```

shape: (215_551,)

Ward
bool
false
false
false
false
false
...
false
false
false
false
false

```
df2["Ward"].eq(30)
```

shape: (215_551,)

Ward
bool
false
false
false
false
false
...
false
false
false
false
false

```
df2["Ward"].is_between(20, 30)
```

shape: (215_551,)

Ward
bool
true
false
false
false
false
...
false
false
false
true
false

```
df2.with_columns(pl.lit(1).alias("new_column"))
```

ion ion	Arrest	Domestic	Beat	District	Ward	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Updated On	Latitude	Longitude	Location	new_column
str	bool	bool	i64	i64	i64	i64	str	i64	i64	i64	str	f64	f64	str	i32
CE"	false	true	2221	22	21	71	"06"	1165137	1846655	2022	"11/12/2022 03:46:21 PM"	41.734817	-87.670596	"(41.734817155, -87.670595647)"	1
ıull	false	false	1711	17	39	13	"11"	1153364	1937188	2022	"11/12/2022 03:46:21 PM"	41.983491	-87.711324	"(41.983490742, -87.711324421)"	1
.K"	true	false	1923	19	46	6	"03"	1168917	1925693	2022	"11/12/2022 03:46:21 PM"	41.951624	-87.654458	"(41.951623924, -87.654458486)"	1
.K"	false	false	2233	22	34	49	"26"	1174663	1829726	2022	"11/12/2022 03:46:21 PM"	41.688155	-87.636199	"(41.688154968, -87.636198645)"	1
CE"	false	true	1111	11	37	23	"05"	1146679	1905959	2022	"11/12/2022 03:46:21 PM"	41.897926	-87.73671	"(41.897926219, -87.736710223)"	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
CE"	false	false	431	4	10	51	"06"	1194763	1840111	2022	"11/12/2022 03:46:21 PM"	41.716183	-87.562275	"(41.71618255, -87.562274818)"	1
EL / EL"	false	false	123	1	42	32	"06"	1177377	1897431	2022	"11/12/2022 03:46:21 PM"	41.873884	-87.624219	"(41.873883785, -87.624218932)"	1

```
df2["Ward"].filter(df2["Ward"] % 2 == 0)
```

shape: (113_215,)

Ward
i64
46
34
28
26
2
…
44
42
10
42
16

```
s3 = df2["Ward"].filter(df2["Ward"] > 30)
```

shape: (67_360,)

Ward
i64
39
46
34
37
33
…
42
44
42
42
49

```
df2.describe()
```

shape: (9, 23)

statistic	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	Beat	District	
str	f64	str	str	str	str	str	str	str	f64	f64	f64	f64	
"count"	215551.0	"215551"	"215551"	"215551"	"215551"	"215551"	"215551"	"214940"	215551.0	215551.0	215551.0	215551.0	21
"null_count"	0.0	"0"	"0"	"0"	"0"	"0"	"0"	"611"	0.0	0.0	0.0	0.0	
"mean"	1.2712e7	null	null	null	null	null	null	null	0.11196	0.180755	1151.768087	11.287616	23.3
"std"	703357.550354	null	null	null	null	null	null	null	null	null	708.24493	7.078264	14.
"min"	26543.0	"HH186297"	"01/01/2022 01:00:00 AM"	"0000X E 100TH PL"	"110"	"ARSON"	"$500 AND UNDER"	"ABANDONED BUILDING"	0.0	0.0	111.0	1.0	
"25%"	1.266904e7	null	null	null	null	null	null	null	null	null	533.0	5.0	
"50%"	1.2751446e7	null	null	null	null	null	null	null	null	null	1032.0	10.0	
"75%"	1.2831264e7	null	null	null	null	null	null	null	null	null	1731.0	17.0	
"max"	1.2914904e7	"JP359726"	"11/30/2022 12:48:00 PM"	"137XX S LEYDEN AVE"	"5132"	"WEAPONS VIOLATION"	"VIOLENT OFFENDER - FAIL TO REG...	"YARD"	1.0	1.0	2535.0	31.0	

Загалом багато подібності між даними бібліотеками

Але давайте розглянем які переваги має Polars більше детально, а саме в порівнянні в часі

Увагу на час виконання

1.2 Basic operations

```
%%time
df_pandas.head(2)
```

```
CPU times: user 83 µs, sys: 116 µs, total: 199 µs
Wall time: 417 µs
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_lo
0	2014-04-26 12:56:25.0000001	12.5	2014-04-26 12:56:25 UTC	-73.980920	40.741840	-73
1	2015-04-27	8.0	2015-04-27	-73.994293	40.751015	-73

In [6]:

```
%%time
df.head(2)
```

```
CPU times: user 24 µs, sys: 10 µs, total: 34 µs
Wall time: 38.1 µs
```

Out[6]: shape: (2, 8)

key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
str	f64	str	f64	f64	f64	
"2014-		"2014-04-26				

Увагу на час виконання

```
# pandas
df_pandas[df_pandas['passenger_count'] > 5][:2]
```

```
CPU times: user 94.5 ms, sys: 403 ms, total: 497 ms
Wall time: 546 ms
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	drop
133	2014-09-21 21:28:00.000000103	6.0	2014-09-21 21:28:00 UTC	-74.000147	40.742907	
148	2013-03-25 11:44:00.00000085	7.5	2013-03-25 11:44:00 UTC	-73.980160	40.755210	

```
# polars
df.filter(pl.col("passenger_count") > 5).head(2)
```

```
CPU times: user 56.9 ms, sys: 408 ms, total: 465 ms
Wall time: 200 ms
```

Out[23]: shape: (2, 8)

key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
str	f64	str	f64	f64	f64	
"2014-09-21 21:...	6.0	"2014-09-21 21:...	-74.000147	40.742907	-73.996047	
"2013-03-25 11:...	7.5	"2013-03-25 11:...	-73.98016	40.75521	-73.99049	

Увагу на час виконання

```
1 %%time
2 df_pandas.describe()
```

```
CPU times: user 2.7 s, sys: 390 ms, total: 3.09 s
Wall time: 3.31 s
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
count	1.000000e+07	1.000000e+07	1.000000e+07	9.999921e+06	9.999921e+06	
mean	1.134640e+01	-7.250440e+01	3.991939e+01	-7.250769e+01	3.991822e+01	
std	2.603907e+01	1.281625e+01	9.411173e+00	1.284424e+01	9.486053e+00	
min	-1.125600e+02	-3.384693e+03	-3.492264e+03	-3.425208e+03	-3.547887e+03	
25%	6.000000e+00	-7.399207e+01	4.073493e+01	-7.399139e+01	4.073405e+01	
50%	8.500000e+00	-7.398180e+01	4.075267e+01	-7.398014e+01	4.075317e+01	
75%	1.250000e+01	-7.396706e+01	4.076715e+01	-7.396366e+01	4.076811e+01	

In [32]:

```
1 %%time
2 df.describe()
```

```
CPU times: user 3.05 s, sys: 929 ms, total: 3.98 s
Wall time: 1.34 s
```

Out[32]: shape: (9, 9)

statistic	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropo
str	str	f64	str	f64	f64	
"count"	"9964861"	9.964861e6	"9964861"	9.964861e6	9.964861e6	
"null_count"	"0"	0.0	"0"	0.0	0.0	
"mean"	null	11.354505	null	-72.503915	39.919053	
"std"	null	26.081635	null	12.825473	9.42218	
"min"	"2009-01-	-112.56	"2009-01-01	-3384.693027	-3492.263768	-3