NINJA DATA

PYTHON 3, PANDAS 2

Python Cheat Sheet Wojownik

Analiza danych w pandas: wczytujesz, czyścisz, liczysz i zapisujesz. Z odpowiednikami w SQL.

ŚCIĄGA | NOWICJUSZ | WOJOWNIK | MISTRZ

01 START — Wczytanie danych

- **import**: standardowe skróty, tak pisze cały świat.

```
import pandas as pd
import numpy as np
```

- **read_csv, read_excel**: plik do tabeli (DataFrame).

```
df = pd.read_csv("sprzedaz.csv", sep=";",
                 decimal=",", encoding="utf-8")
df = pd.read_excel("raport.xlsx",
                   sheet_name="Dane")
```

- **Pierwszy rzut oka**: wymiary, typy, braki, statystyki.

```
df.shape          # (wiersze, kolumny)
df.head(10)       # pierwsze 10 wierszy
df.info()         # typy i niepuste wartości
df.describe()     # średnia, min, max, kwartyle
df["Kategoria"].value_counts()  # ile razy
```

02 WYBIERANIE — Kolumny i wiersze

- **Kolumny**: jedna to Series, lista to DataFrame. SQL: SELECT.

```
df["Cena"]
df[["Nazwa", "Cena"]]
```

- **loc, iloc**: po nazwach albo po numerach.

```
df.loc[df["Cena"] > 50, ["Nazwa", "Cena"]]
df.iloc[:5, :3]   # 5 wierszy, 3 kolumny
```

- **Filtr**: warunek w nawiasach, & to AND, | to OR. SQL: WHERE.

```
df[(df["Cena"] > 50) & (df["Kraj"] == "PL")]
```

- **isin, between, str.contains**: IN, BETWEEN i LIKE.

```
df[df["Kraj"].isin(["PL", "DE"])]
df[df["Cena"].between(50, 100)]
df[df["Nazwa"].str.contains("Pro", na=False)]
```

- **query**: filtr zapisany jak tekst, krócej przy wielu warunkach.

```
df.query("Cena > 50 and Kraj == 'PL'")
```

- **sort_values, head**: ORDER BY i TOP.

```
df.sort_values("Cena", ascending=False).head(10)
```

03 CZYSZCZENIE — Braki, duplikaty, typy

- **isna, fillna, dropna**: brakujące wartości (NaN).

```
df.isna().sum()   # braki w kolumnach
df["Rabat"] = df["Rabat"].fillna(0)
df = df.dropna(subset=["KlientID"])
```

- **drop_duplicates**: usuwasz powtórzone wiersze.

```
df = df.drop_duplicates(subset=["Email"],
                        keep="last")
```

- **astype, to_numeric**: zmiana typu, błędne wartości jako NaN.

```
df["Ilosc"] = df["Ilosc"].astype(int)
df["Kwota"] = pd.to_numeric(df["Kwota"],
                            errors="coerce")
```

- **rename, drop**: nowe nazwy kolumn, usuwanie kolumn.

```
df = df.rename(columns={"cena_pln": "Cena"})
df = df.drop(columns=["Uwagi"])
```

- **Tekst**: metody .str działają na całej kolumnie.

```
df["Miasto"] = (df["Miasto"].str.strip()
                            .str.title())
```

04 NOWE KOLUMNY — Obliczenia

- **Działania na kolumnach**: liczy się wiersz po wierszu, bez pętli.

```
df["Wartosc"] = df["Cena"] * df["Ilosc"]
```

- **np.where**: if na kolumnie. SQL: CASE WHEN.

```
df["Segment"] = np.where(df["Cena"] >= 500,
                         "Premium", "Standard")
```

- **pd.cut**: przedziały liczb jako kategorie.

```
df["Grupa"] = pd.cut(df["Wiek"],
    bins=[0, 25, 40, 120],
    labels=["do 25", "26-40", "41+"])
```

- **apply**: własna funkcja, gdy nie ma gotowej metody (wolniej).

```
df["Kod"] = df["Nazwa"].apply(
    lambda x: x[:3].upper())
```

05 DATY
Kolumny z datą

- **to_datetime**: tekst na datę. Bez tego daty sortują się jak tekst.

```
df["Data"] = pd.to_datetime(df["Data"],
                            dayfirst=True)
```

- **.dt**: rok, miesiąc, dzień tygodnia i okres.

```
df["Rok"] = df["Data"].dt.year
df["Miesiac"] = df["Data"].dt.to_period("M")
df["Dzien"] = df["Data"].dt.day_name()
```

- Różnica dat i przesunięcie.

```
df["Dni"] = (df["Dostawa"] - df["Data"]).dt.days
df["Za7Dni"] = df["Data"] + pd.Timedelta(days=7)
```

- **resample**: suma po tygodniach albo miesiącach (data jako indeks).

```
(df.set_index("Data")["Wartosc"]
   .resample("MS").sum())
```

06 GRUPOWANIE
groupby i tabele przestawne

- **groupby + agg**: GROUP BY z nazwanymi wynikami.

```
wynik = (df.groupby("Kategoria", as_index=False)
           .agg(Sprzedaz=("Wartosc", "sum"),
                Zamowienia=("ID", "count"),
                Klienci=("KlientID", "nunique")))
```

- Kilka kolumn grupujących i sortowanie wyniku.

```
(df.groupby(["Rok", "Kraj"])["Wartosc"].sum()
   .sort_values(ascending=False))
```

- **transform**: wynik grupy obok każdego wiersza (jak okno w SQL).

```
suma = (df.groupby("Kategoria")["Wartosc"]
          .transform("sum"))
df["UdzialKat"] = df["Wartosc"] / suma
```

- **pivot_table**: tabela przestawna jak w Excelu.

```
pd.pivot_table(df, index="Produkt",
    columns="Kraj", values="Wartosc",
    aggfunc="sum", fill_value=0)
```

07 ŁĄCZENIE
merge i concat

- **merge**: JOIN dwóch tabel. how: inner, left, right, outer.

```
df = zam.merge(klienci, how="left",
               left_on="KlientID", right_on="ID")
```

- **indicator**: sprawdzasz, co się nie dopasowało.

```
m = zam.merge(klienci, on="KlientID", how="left",
              indicator=True)
m[m["_merge"] == "left_only"]
```

- **concat**: tabele jedna pod drugą. SQL: UNION ALL.

```
razem = pd.concat([sty, lut, mar],
                  ignore_index=True)
```

WSKAZÓWKA Po merge porównaj `len(df)` przed i po. Więcej wierszy znaczy, że klucz po drugiej stronie się powtarza.

08 SQL A PANDAS
Słownik

- **SELECT a, b**: `df[["a", "b"]]`
- **WHERE**: `df[df["a"] > 5]` albo `df.query("a > 5")`
- **ORDER BY a DESC**: `df.sort_values("a", ascending=False)`
- **TOP 10**: `df.head(10)`
- **GROUP BY**: `df.groupby("a").agg(...)`
- **JOIN**: `df.merge(df2, on="id", how="left")`
- **UNION ALL**: `pd.concat([df1, df2])`
- **DISTINCT**: `df.drop_duplicates()`
- **COUNT(DISTINCT a)**: `df["a"].nunique()`

09 ZAPIS
Wynik do pliku

- **to_csv, to_excel**: bez indeksu, polskie znaki zostają.

```
wynik.to_csv("wynik.csv", index=False,
             sep=";", encoding="utf-8-sig")
wynik.to_excel("wynik.xlsx", index=False)
```

Uczysz się Pythona? Odhaczaj tematy dzień po dniu. **Tracker 100 dni Python** sklep.ninjadata.pl/tracker-100-dni-nauki-python/