Pozyskiwanie danych i ich procesowanie

Paweł Gliwny

Gromadzenie danych – fundament projektu analitycznego

- Gromadzenie danych (ang. *Data Collection*) to pozornie pierwszy etap każdego projektu analitycznego.
- W praktyce inżynierskiej praca analityka zaczyna się jednak znacznie wcześniej – na etapie **strategicznego planowania**.
- Zanim napiszesz choćby jedną linię kodu do pobrania danych, musisz precyzyjnie odpowiedzieć na kluczowe pytania:

Definicja Problemu: Jaki konkretny problem biznesowy, techniczny lub badawczy staramy się rozwiązać?

Definicja problemu biznesowego

Przykład Finanse

- Jak wykryć podejrzane transakcje?
- Kto nie spłaci kredytu?
- Kiedy sprzedać akcje?

Finanse - jakie dane potrzebujemy?

- **Wykrywanie podejrzanych transakcji:**
  - Historia transakcji (kwoty, częstotliwość, lokalizacje)
  - Dane demograficzne klienta
  - Relacje między kontami
- **Ocena ryzyka kredytowego:**
  - Historia kredytowa (BIK, credit score)
  - Dochody i zatrudnienie
  - Majątek i zobowiązania

Źródła danych - największe wyzwanie projektów

Dlaczego to najtrudniejszy etap:

- 80% czasu projektu = przygotowanie danych
- Dane często nie istnieją w potrzebnej formie
- Rozproszone w różnych systemach
- Różne formaty i standardy
- Problemy z jakością i kompletnością

Typowe źródła danych

Wewnętrzne systemy:

- Bazy danych transakcyjne
- CRM, ERP
- Logi aplikacji i serwerów
- Systemy IoT i sensory

CRM, ERP, systemy HR*

CRM (Customer Relationship Management):

- Zarządzanie relacjami z klientami
- Dane: kontakty, historia zakupów, interakcje
- Przykłady: Salesforce, HubSpot, Pipedrive
- Analiza: segmentacja klientów, przewidywanie churn

ERP (Enterprise Resource Planning):

- Zarządzanie zasobami przedsiębiorstwa
- Dane: finanse, magazyn, produkcja, zakupy
- Przykłady: SAP, Oracle, Microsoft Dynamics
- Analiza: optymalizacja kosztów, prognozowanie sprzedaży

Typowe źródła danych

Zewnętrzne źródła:

- API firm trzecich
- Dane publiczne (GUS, dane miast)
- Web scraping
- Zakup danych od brokerów

Wyzwania prawne i techniczne:

- RODO i prywatność danych
- Zgodności i licencje
- Limity API i rate limiting
- Zmieniające się struktury danych

Problemy integracji:

- Różne formaty (JSON, XML, CSV)
- Różne częstotliwości aktualizacji
- Problemy z encodingiem
- Duplikaty i niespójności

Rozwiązania

- Data lakes i warehouse
- ETL/ELT pipelines
- Standardy i governance
- Dokumentacja źródeł danych

O tym więcej na studiach II stopnia na zajęciach z akwizycji danych.

Źródła otwartych danych oraz platformy

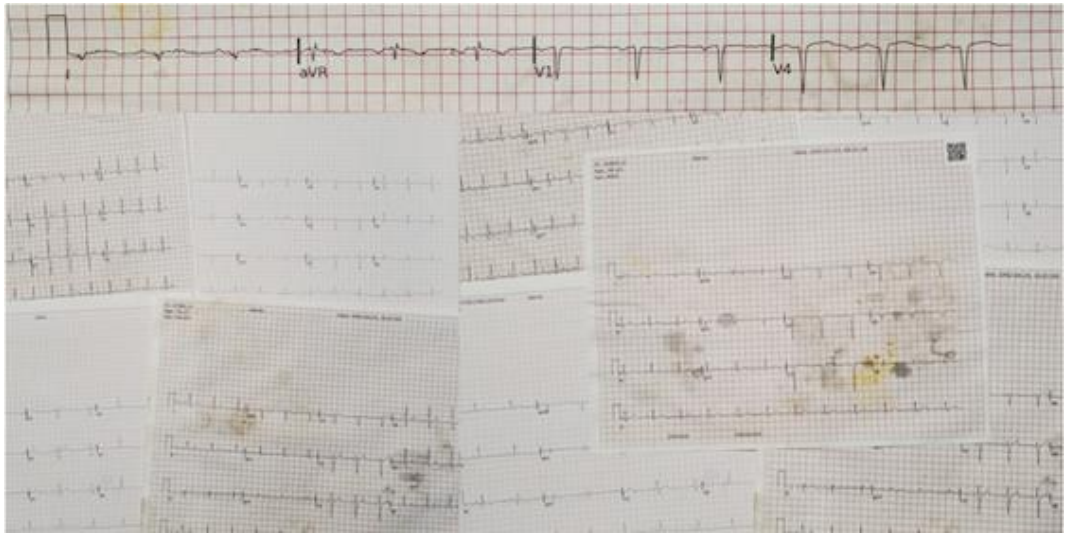
- Kaggle (kaggle.com)
- UCI Machine Learning Repository
- Hugging Face
- Our World in Data
- Dane.gov.pl

Kaggle (kaggle.com)

- Platforma konkursów data science
- Tysiące zbiorów danych z różnych dziedzin
- Gotowe kernels/notebooks z analizami
- Społeczność praktyków i naukowców

Przykład konkurs Kaggle

- Dane nieustrukturyzowane na ustrukturyzowane
- **PhysioNet - Digitization of ECG Images**
  - Extract the ECG time-series data from scans and photographs of paper printouts of the ECGs.



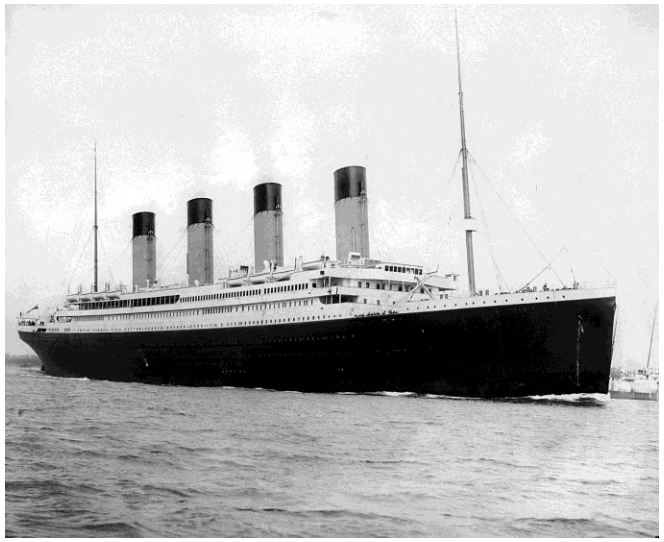
- link

UCI Machine Learning Repository

- Akademicka kolekcja zbiorów danych
- Klasyczne datasety do nauki ML
- Dobrze udokumentowane i oczyszczone
- Standardy w literaturze naukowej

Popularne zbiory danych

- **Klasyczne zbiory:**
  - **Iris** - kwiaty tęczówki (1936, Ronald Fisher)
  - **Titanic** - pasażerowie statku (1912, kaggle)
  - **Wine** - klasyfikacja win (UCI ML Repository)



Popularne zbiory danych

- Małe rozmiary - szybkie ładowanie i przetwarzanie
- Czyste dane - brak problemów z jakością
- Dobrze udokumentowane
- Dostępne w bibliotekach (**sklearn**, **seaborn**)
- Sprawdzone algorytmy i wyniki

Popularne zbiory danych

Minusy:

- Wszyscy je znają - brak oryginalności
- Zbyt proste - nie oddają rzeczywistych problemów
- Małe rozmiary - nie testują skalowalności
- Brak nowoczesnych wyzwań (big data, real-time)

Kiedy używać:

- Nauka podstaw
- Testowanie algorytmów
- Prototypowanie
- Demonstracje na zajęciach

Hugging Face (HF)

- Platforma open-source dla AI/ML
- Hub modeli, datasetsów i narzędzi
- Społeczność praktyków i badaczy
- Darmowe i płatne zasoby

- https://huggingface.co/

HF Datasets Hub

- **60,000+ zbiorów danych** z różnych dziedzin
- NLP, computer vision, audio, tabular data
- Gotowe do użycia w bibliotekach ML
- Dokumentacja i przykłady użycia

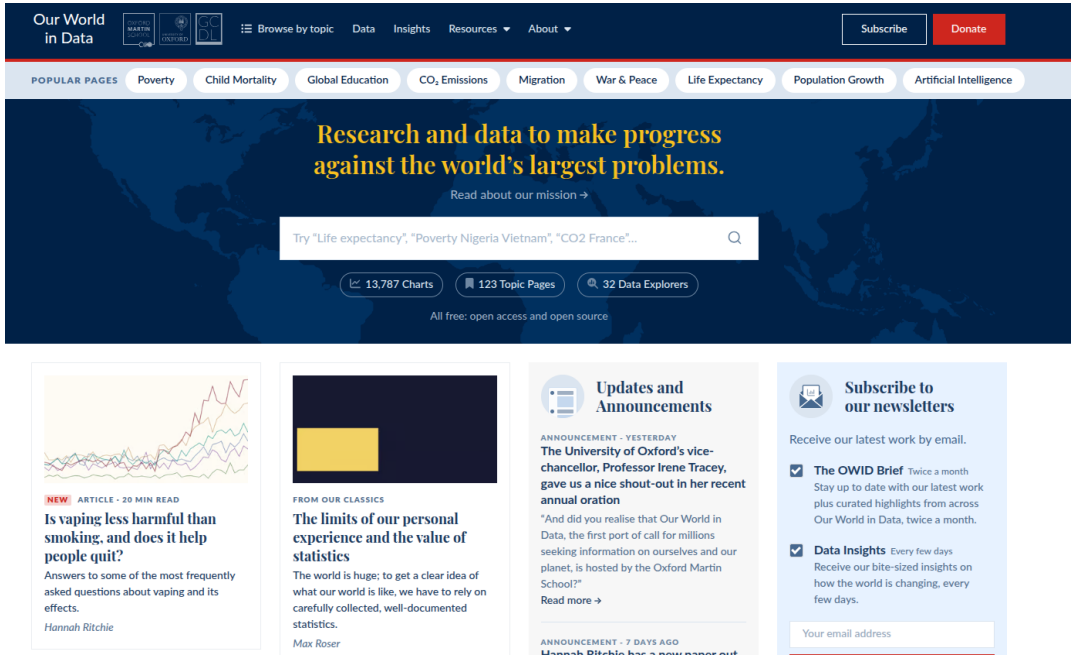
```
from datasets import load_dataset
dataset = load_dataset("imdb")
```

HF zastosowanie dla analizy danych

- **Text analysis** - sentiment, klasyfikacja, NER
- **Image processing** - klasyfikacja obrazów
- **Audio analysis** - rozpoznawanie mowy
- **Multimodal** - kombinacja tekst + obraz

Our World in Data

- Dane o globalnych trendach i problemach
- Demografia, ekonomia, środowisko, zdrowie
- Profesjonalne wizualizacje i analizy
- Rzetelne źródła i metodologia



Dane.gov.pl

- Oficjalne dane polskiej administracji
- Finanse publiczne, demografia, transport
- Format otwarty (CSV, JSON, XML)
- Aktualizowane przez urzędy

Portale Open Data polskich miast

- Platformy udostępniające dane publiczne w formatach umożliwiających ich pobieranie i analizę
- Dane o transporcie, środowisku, budżecie, infrastrukturze miejskiej
- Formaty: CSV, JSON, XML, API
- Dostęp bezpłatny i otwarty dla wszystkich
- **Link dla Łódź**: https://otwarte.miasto.lodz.pl/

Formaty danych i operacje

Operacje wejścia/wyjścia

- Odczytywanie plików tekstowych i innych bardziej wydajnych formatów na dysku,
- Ładowanie danych z baz danych
- Pozyskiwanie danych za pomocą interfejsu API

Rożne formaty danych

- **Formaty tekstowe:**
  - **CSV, TSV** - proste tabele, różne separatory
  - **JSON** - strukturalne dane z API
  - **XML** - hierarchiczne dane, metadane
  - **TXT, LOG** - niestrukturalne logi systemu
- **Formaty arkuszy:**
  - **Excel (.xlsx, .xls)** - arkusze z formatowaniem
  - **Google Sheets** - współdzielone arkusze online
  - **LibreOffice Calc** - alternatywa open source

Obsługa baz danych

Dane biznesowe często nie są przechowywane w Excel tylko w bazach danych.

- Zwykle są one umieszczone w relacyjnych bazach danych opartych na SQL-u (np. bazach SQL Server, PostgreSQL i MySQL).
- Popularne również inne alternatywne bazy danych.
- Wybór zależy od wydajności, spójności danych i potrzeby skalowalności aplikacji.

Bazy danych

- **SQL** - relacyjne (MySQL, PostgreSQL, Oracle)
- **NoSQL** - nierelacyjne (MongoDB, Cassandra)

- **Wektorowe bazy danych** - do przechowywania wektorów.
  - Tego typu dane są bardzo przydatne przy pracy z **przetwarzaniem języka naturalnego (NLP)**, modelami rekomendacyjnymi i przy semantycznym wyszukiwaniu.

Pliki tekstowe (CSV, TXT)

- Proste w użyciu, czytelne dla człowieka
- Duży rozmiar na dysku (brak kompresji)
- Wolne odczytywanie (parsowanie tekstu)
- Brak typów danych (wszystko jako tekst)

```
import pandas as pd
# Odczytywanie CSV - wolniejsze dla dużych zbiorów
df = pd.read_csv('dane.csv') # ~100 MB/s
```

Formaty binarne

- **Parquet** - kolumnowe, zoptymalizowane
- **HDF5** - duże dane naukowe
- **Pickle** - obiekty Python

Formaty binarne – dlaczego są bardziej wydajne?

1. **Parquet**
   - **Kompresja:** 80-90% mniejszy rozmiar niż CSV
   - **Kolumnowy format:** czyta tylko potrzebne kolumny
   - **Zachowuje typy danych:** nie trzeba konwertować
   - **Szybkość:** 5-10x szybsze odczytywanie

```
# Zapis i odczyt Parquet
df.to_parquet('dane.parquet',compression='snappy')


df = pd.read_parquet('dane.parquet')
# ~500 MB/s


# Czytanie tylko wybranych kolumn
df = pd.read_parquet('dane.parquet',
columns=['kolumna1', 'kolumna2'])
```

Hierarchiczny format danych: **HDF5**

- Obsługa ogromnych zbiorów danych
- Możliwość odczytu fragmentów
- Przechowywanie wielu tabeli w jednym pliku

```
# Zapis wielu tabel
df.to_hdf('dane.h5', key='sprzedaz', mode='w')
df2.to_hdf('dane.h5', key='klienci', mode='a')
 # Odczyt z filtrowaniem
df = pd.read_hdf('dane.h5', key='sprzedaz', where='cena
> 100')
```

Ładowanie danych z baz danych

Dlaczego bazy danych?

- Dane **za duże** dla pamięci RAM
- **Filtrowanie** po stronie serwera
- **Współdzielenie** danych między użytkownikami
- Spójność danych

Przykład SQLite

```
import sqlite3

conn = sqlite3.connect('baza.db')

df = pd.read_sql('SELECT * FROM klienci', conn)

query = """
        SELECT imie, nazwisko, zakupy
        FROM klienci
        WHERE miasto = 'Warszawa' AND wiek > 25 """

df = pd.read_sql(query, conn)
conn.close()
```

PostgreSQL (serwer bazodanowy)

```
from sqlalchemy import create_engine

Engine = create_engine('postgresql://user:password@localhost:5432/mydatabase')

df = pd.read_sql(
        'SELECT * FROM sprzedaz WHERE data > "2024-01-01"',
         engine)

# Zapis z powrotem do bazy
df.to_sql('wyniki', engine, if_exists='replace', index=False)
```

Chunking (dla dużych zbiorów)

```
# Odczyt w kawałkach, aby nie zapełnić pamięci

for chunk in pd.read_sql('SELECT * FROM duza_tabela', conn, chunksize=10000):
        przetworzone = chunk[chunk['wartosc'] > 100]
        przetworzone.to_csv('wynik.csv', mode='a', header=False)
```

Przygotowanie danych do analizy

Przygotowanie danych w analizie

80%+ czasu analityka = przygotowanie danych

Główne zadania w przygotowaniu danych:

- Ładowanie danych
- Czyszczenie
- Transformacja
- Reorganizacja

Problem: dane rzadko w odpowiednim formacie

Operacje na danych w praktyce

Rzeczywistość projektów:

- Dane rozproszone w wielu źródłach
- Różne formaty i struktury
- Nieprzygotowane do bezpośredniej analizy
- Wymagają transformacji przed użyciem

Kluczowe operacje: **łączenie danych (join)**

Łączenie danych (JOIN):

- **Inner join** - tylko wspólne rekordy
- **Left/Right join** - zachowanie wszystkich z jednej strony
- **Full outer join** - wszystkie rekordy z obu tabel
- **Cross join** - iloczyn kartezjański

Kluczowe operacje: **transformacje**

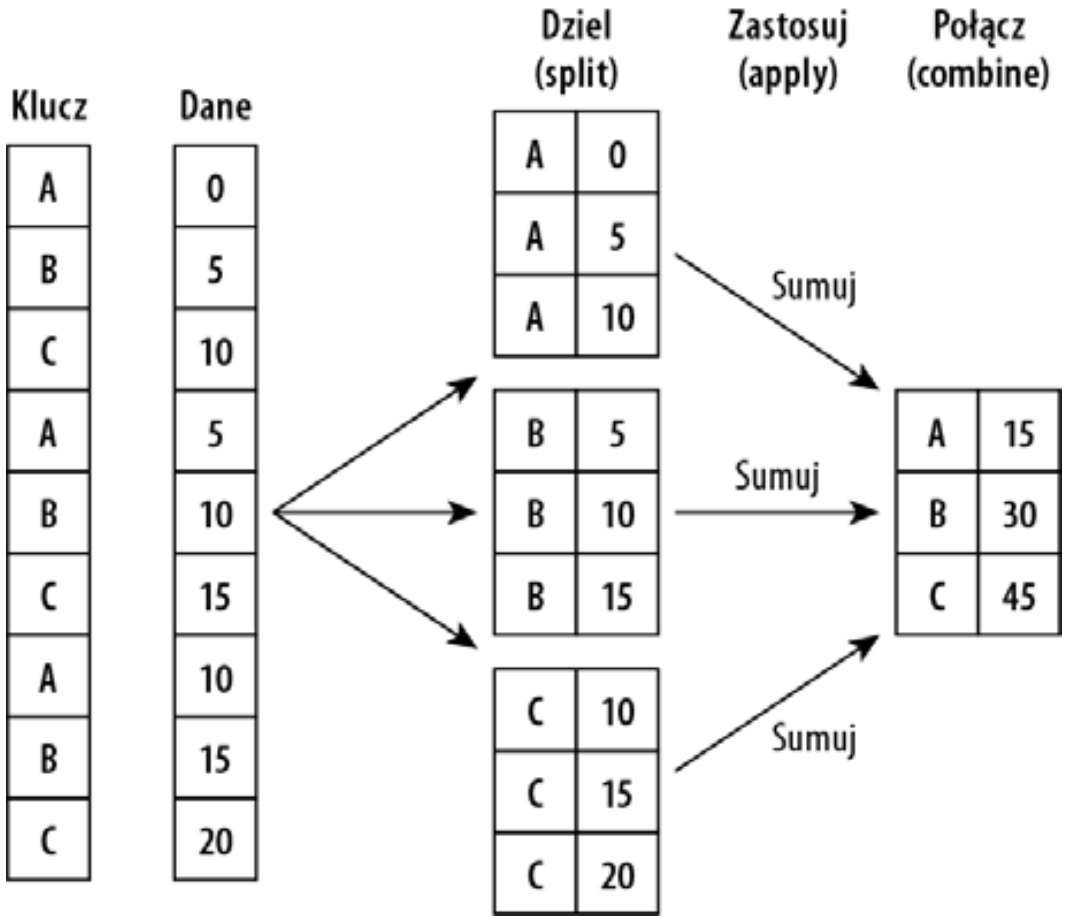
- **Pivot/Unpivot** - zmiana układu kolumn/wierszy
- **Filtering** - filtrowanie według warunków
- **Sorting** - sortowanie danych
- **Data type conversion** - zmiana typów danych

Kluczowe operacje: **agregacja danych**

- **GROUP BY** - grupowanie według kryteriów
- **SUM, AVG, COUNT** - funkcje agregujące
- **MIN, MAX** - wartości skrajne
- **DISTINCT** - unikalne wartości

GroupBy - co to jest?

- Mechanizm grupowania danych według wspólnych wartości
- Podstawowa operacja w analizie danych
- Pozwala analizować dane w kategoriach/segmentach
- Źródło: **Python w analizie danych, Wes McKinney**



GroupBy – po co?

- **Obliczenia statystyczne:**
  - Średnie wartości w grupach
  - Sumy według kategorii
  - Zliczanie wystąpień
- **Analiza segmentów:**
  - Sprzedaż według regionów
  - Wydajność według działów
  - Zachowania według grup wiekowych

Alternatywa bez GroupBy

- Ręczne filtrowanie każdej kategorii
- Wielokrotne obliczenia
- Podatność na błędy
- Nieefektywny kod
- **Z GroupBy = jedna linia kodu!**

```
# Grupowanie po jednej kolumnie
df.groupby('region').sales.mean() #

Grupowanie po wielu kolumnach
df.groupby(['region', 'product']).sum()


# Różne funkcje dla różnych kolumn
df.groupby('category').agg({
    'price': 'mean',
    'quantity': 'sum'
})
```

Narzędzia

- **SQL** - bazy danych
- **Pandas** - Python
- **dplyr** - R
- **Excel** - podstawowe operacje

Python i Excel

- Wiele biznesów opiera się o Excel :(
- **openpyxl** to biblioteka Pythona do pracy z plikami Excel (.xlsx) do automatyzacji pracy
- Pozwala:
  - **Tworzyć** nowe pliki Excel z wieloma arkuszami (sheets)
  - **Odczytywać** dane z istniejących plików
  - **Modyfikować** zawartość komórek, dodawać formuły i formatowanie
  - Wszystko programowo, bez potrzeby otwierania Excela

openpyxl i pandas

- **pandas** używa **openpyxl** jako "silnika" do odczytu i zapisu plików Excel.

Kiedy używać openpyxl

- **Wiele arkuszy** - kontrola nad każdym arkuszem
- **Formatowanie** - kolory, czcionki, obramowania
- **Formuły Excel** - `=SUMA(A1:A10)`
- **Precyzyjna kontrola** - konkretne komórki i struktura pliku

Najczęściej spotykane problemy w danych

- Elementy odstające
- Brakujące wartości

Elementy odstające

- Elementy odstające są punktami danych znajdującymi się w pewnej odległości od grup innych punktów.
- Powodują problemy podczas tworzenia modeli predykcyjnych:
  - długi czas uczenia modelu,
  - kiepska dokładność,
  - zwiększenie błędu wariancji,
  - zmniejszenie normalności danych

Wykrywanie i rozwiązywanie kwestii elementów odstających

- Wykres pudełkowy
- Wykres punktowy
- Wynik standaryzowany z (ang. z-score)
- Rozstęp ćwiartkowy
- Percentyl

Zobacz notebook

Brakujące wartości

Brakujące wartości to rekordy, które są nieobecne w zestawie danych.

Przyczyny występowania:

- Błąd człowieka podczas wprowadzania danych
- Ochrona prywatności (dane wrażliwe)
- Brak odpowiedzi w ankietach
- Problemy techniczne przy zbieraniu danych

Jest to jeden z **najczęściej występujących problemów** w analizie danych i kluczowy etap wstępnego przetwarzania (data preprocessing).

1. Sprawdzenie czy są brakujące wartości

```
>> df.isnull()
>> df.isna()
```

2. Liczba brakujących wartości w każdej kolumnie

```
>> df.isna() .sum()
```

3. Procent brakujących wartości

```
>> (df.isnull().sum() / len(df)) * 100
```

4. Całkowita liczba brakujących wartości

```
>> df.isnull().sum().sum()
```

Rozwiązanie kwestii brakujących danych

- **Usuwanie rekordów** zawierających brakujące wartości.
- **Ręczne uzupełnianie** brakujących wartości.
- Uzupełnianie **brakujących wartości wskaźnikami** tendencji centralnej, np.: średnią, medianą czy dominantą.
  - **Średniej** używamy w przypadku cech numerycznych,
  - **mediany** w cechach porządkowych,
  - natomiast **dominantę** (czyli najczęściej powtarzającą się wartość) umieszczamy w cechach kategorialnych.

Rozwiązanie kwestii brakujących danych Zaawansowane

- Uzupełnianie **najbardziej prawdopodobną wartością** przy użyciu modeli uczenia maszynowego
  - regresja,
  - drzewa decyzyjne
  - algorytm KNN.

Czym jest NaN i dlaczego jest wyjątkowy?

- **NaN** (Not a Number) – specjalna wartość reprezentująca brakujące dane
- Dane mogą zawierać różne oznaczenia braków:
  - Tekstowe: `"NA"`, `"N/A"`, `"brak"`, `"?"`, `"-"`
  - Numeryczne: `0`, `-999`, `-1`
  - Inne: puste stringi `""`

```
import numpy as np

np.nan == np.nan # False (!)

np.nan is np.nan # True

# Dlatego używamy specjalnych funkcji:

pd.isna(np.nan) # True ✓
pd.isnull(np.nan) # True ✓
```

Przykład z NaN

```
>> df = pd.DataFrame({
        'A': [1, np.nan, 3], # ✓ NaN - brakująca wartość
        'B': [4, None, 6], # ✓ None - też brakująca
        'C': [7, 0, 9], # ✗ 0 - NIE jest brakująca!
        'D': ['x', '', 'z'] # ✗ '' - NIE jest brakująca!
    })

>> df.isnull().sum()
# A 1 ← wykrywa NaN
# B 1 ← wykrywa None
# C 0 ← zero to wartość!
# D 0 ← pusty string to wartość!
```

Usuwanie brakujących wartości

Funkcja `dropna()` – usuwa wiersze/kolumny z brakującymi wartościami

```
# Usuń wiersze z DOWOLNĄ brakującą wartością (domyślnie)
df.dropna()
df.dropna(how='any')

# Usuń wiersze tylko gdy WSZYSTKIE wartości są brakujące
df.dropna(how='all')

# Usuń kolumny z brakującymi wartościami
df.dropna(axis=1)
```

Uzupełnianie brakującej wartości

- W Pythonie możemy uzupełnić brakujące wartości za pomocą funkcji **fillna()**.
  - Przyjmuje ona jedną wartość, która będzie wstawiana w pustych pozycjach lub zamiast wartości NaN.

```
# Czyta dane
>> data = pd.read_csv('employee.csv')
# Uzupełnia wszystkie brakujące wartości w kolumnie age średnią obliczoną z tejże kolumny
>> data['age'] = data.age.fillna(data.age.mean())
```

Uzupełnienie średnią lub modą

```
# Uzupełnia wszystkie brakujące wartości w kolumnie income
  medianą obliczoną z tejże kolumny
>>data['income']=data.income.fillna(data.income.median())

# Zastępuje wszystkie brakujące wartości w kolumnie gender
  (kolumna kategorii) dominantą wyliczoną z tejże kolumny
>>data['gender']=data['gender'].fillna(data['gender'].mode()[
0])
```