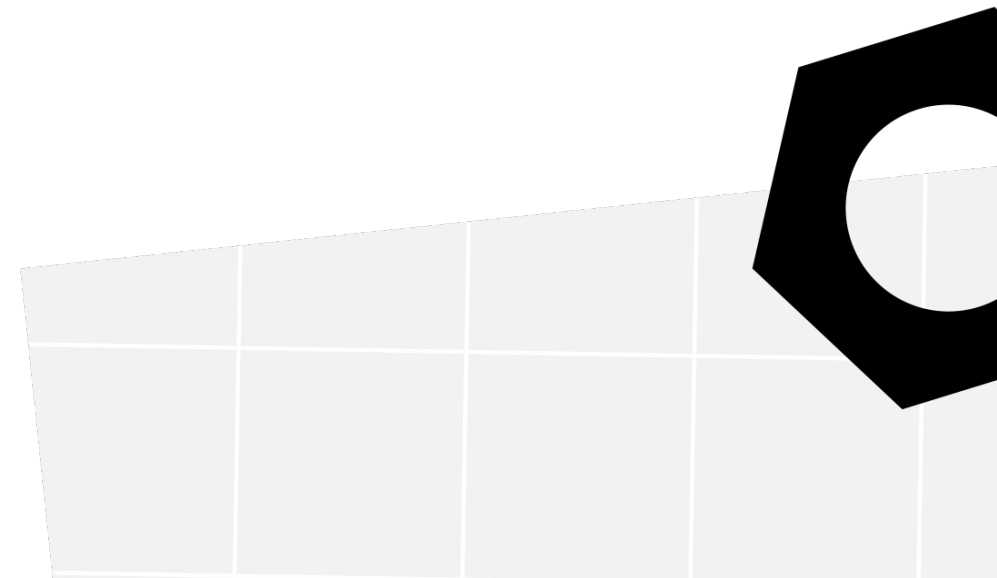




# Przetwarzanie zbiorów danych

**Kurs Data Science**

**Część I**





# Zadania praktyczne podczas zajęć

W prezentacji do bloku **Przetwarzanie zbiorów danych** zamieszczone zostały dodatkowe zadania praktyczne.

Rozwiązania do nich możesz pobrać z sekcji  
"Dodatkowe materiały do bloku" [tutaj](#).

# Zadania praktyczne podczas zajęć

Uwaga: W razie problemów z instalacją pandas-profiling na Google Colaboratory, polecamy zainstalować starszą wersję stosując poniższe komendy:

```
import sys
!pip install -U pip
!{sys.executable} -m pip install -U pandas-profiling[notebook]
!pip install pandas-profiling==2.7.1
!jupyter nbextension enable --py widgetsnbextension
#po tym uruchomić środowisko ponownie ctrl+M
```



# Plan zajęć:

- 1) 5 godzin pracy z trenerem + 2 godziny pracy samodzielnej:
  - ramki danych – biblioteka pandas
  - praca z danymi, podstawowe operacje
  - feature engineering
- 1) 5 godzin pracy z trenerem + 2 godziny pracy samodzielnej:
  - język SQL i bazy danych
  - bazy danych programowanie
- 1) 5 godzin pracy z trenerem + 2 godziny pracy samodzielnej:
  - sql\_alchemy
  - praca z plikami – serializacja
  - pandas profiling
  - pobieranie danych z API

**uwaga: do zadań wykonywanych podczas zajęć będziemy wykorzystywać notebook jupyterowy z zadaniami z Pandasa – możesz go pobrać z sekcji “Dodatkowe materiały do bloku” [tutaj](#)**

# Biblioteca Pandas





# Pandas – co to w ogóle jest?

Pandas to kolejna biblioteka niezbędna do analizy danych w Pythonie. Dostarcza wydajne struktury danych, dzięki którym praca z danymi tabelarycznymi staje się prosta i intuicyjna. Celem twórców jest utrzymanie statusu biblioteki niezbędnej do codziennych analiz oraz zdobycie fotela lidera w kategorii najpotężniejszego narzędzia open-source do analizy danych w jakimkolwiek języku programowania. Obecnie, projekt wciąż prężnie się rozwija i jego znajomość jest niezbędna dla każdego data scientista.



# Pandas – kiedy stosować?

Pandas będzie dobrym wyborem do następujących zastosowań:

- Dane tabelaryczne (kolumny jak w SQLu lub Excelu)
- Dane reprezentujące szeregi czasowe
- Macierze
- Wyniki pomiarów i statystyk



# Pandas - mocne strony



Mocnymi stronami Pandas są między innymi:

- Prosta obsługa brakujących wartości (NaN),
- Możliwość modyfikowania rozmiaru DataFrame'a - możemy dodawać i usuwać kolumny i wiersze,
- Automatyczne wyrównywanie danych w obliczeniach (jak w NumPy),
- Metoda groupby działająca analogicznie jak w SQLu,
- Łatwo stworzyć DataFrame na podstawie innego obiektu,
- Cięcie, indeksowanie i tworzenie podzbiorów,
- Łączenie (join i merge) zbiorów.



# Pandas - struktury danych

**Series** to jednowymiarowa struktura danych (jednowymiarowa macierz numpy), która oprócz danych przechowuje też unikalny indeks.

Tworzenie - *pd.Series(zawartosc)*

```
pd.Series(np.random.random(10))
```

```
0    0.661750
1    0.072319
2    0.758071
3    0.035651
4    0.992312
5    0.488344
6    0.083446
7    0.319852
8    0.419058
9    0.310146
dtype: float64
```

# Pandas – struktury danych

Drugą strukturą w pandas jest **DataFrame (ramki danych)** – czyli dwu lub więcej wymiarowa struktura danych, najczęściej w formie tabeli z wierszami i kolumnami. Kolumny mają nazwy, a wiersze mają indeksy.

Tworzenie – `pd.DataFrame(zawartosc)`

Date	Open	High	Low	Close	Volume	Adj Close
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
...	...	...	...	...	...	...

# Pandas – wczytywanie danych

Pierwszy krok jest zwykle ten sam. Dane są przechowywane w plikach csv, tsv, bazach danych, plikach excel itd. Wczytać je można np. z użyciem funkcji `pd.read_csv`

[https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

Najważniejsze argumenty (warto przejrzeć wszystkie, to chyba najważniejsza funkcja pandas, odpowiednie wczytanie pliku ułatwia pracę na nim):

- ścieżka do pliku
- separator kolumn (domyślnie przecinek)
- nagłówek
- kolumna indeksu
- nazwy kolumn

```
# plik tsv - rozdzielony tabulatorami  
chipotle = pd.read_csv('ML-datasets/chipotle.tsv', sep='\t')
```



# Pandas – wyświetlanie danych



Po załadowaniu danych kolejnym krokiem jest ich wyświetlenie, można to robić na wiele sposobów:

- `df.head(n)` – wyświetla pierwsze `n` rekordów danych
- `df.tail(n)` – wyświetla końcowe `n` rekordów danych
- `df.sample(n)` – wyświetla losowe rekordy danych w liczbie `n`
- `df["kolumna"]` lub `df.kolumna` – wyświetlenie danej kolumny (jako Series)
- `df[["kolumna"]]` – wyświetlenie danej kolumny (jako DataFrame)
- `df[["kolumna1", "kolumna2"]]` – wyświetlenie kilku kolumn
- `df.index` – wyświetlanie nazw indeksów (wierszy)
- `df.columns` – wyświetlanie nazw kolumn
- `df.info()` – ogólne informacje o zbiorze



# Pandas - wyświetlanie danych - loc i iloc



Dla dataframe'a występują dwie funkcje do pobierania określonych danych:

- loc szuka po nazwach kolumn i indeksów  
[[wiersze],[kolumny]]
- iloc po ich numerach porządkowych  
[[numer wiersza],[numer kolumny]]

# Pandas – wyświetlanie danych – warunki

Można również  
w nawiasach  
podawać  
warunek  
logiczny.

```
print(read.how == 'SEO') # Series z wartościami True/False według wierszy  
display(read[read.how == 'SEO']) # Wyświetla wszystkie wiersze dla których było True
```

```
time  
2018-01-01 00:01:01    True  
2018-01-01 00:03:20    True  
2018-01-01 00:04:01   False  
2018-01-01 00:04:02   False  
2018-01-01 00:05:03   False  
...  
2018-01-01 23:57:14   False  
2018-01-01 23:58:33    True  
2018-01-01 23:59:36   False  
2018-01-01 23:59:36   False  
2018-01-01 23:59:38   False  
Name: how, Length: 1795, dtype: bool
```

	status	country	identifier	how	continent
time					
2018-01-01 00:01:01	read	country_7	2458151261	SEO	North America
2018-01-01 00:03:20	read	country_7	2458151262	SEO	South America
2018-01-01 00:08:57	read	country_7	2458151272	SEO	Australia
2018-01-01 00:11:22	read	country_7	2458151276	SEO	North America
2018-01-01 00:13:05	read	country_8	2458151277	SEO	North America



# Pandas – nadpisywanie danych



Za pomocą komendy przypisania (=) można też nadpisywać lub dołączać dane:

- `df["kolumna"] = seria_danych`
- `df[["kolumna"]] = dataframe`
- `df[["kolumna1", "kolumna2"]] = dataframe`
- `df.loc[kolumna:wiersz] = dana`
- `df.iloc[numer_kolumny:numer_wiersza] = dana`



# Pandas – operacje na danych



Wszystkie poniższe operacje dla dataframe'u zwrócą wartości dla każdej z kolumn, możemy wybierać kolumny dla których chcemy je wykonać:

- `df.count()` – zliczanie liczby elementów (nie NaNów)
- `df.kolumna.value_counts()` – zliczanie liczby unikalnych elementów
- `df.sum()` – suma wszystkich elementów
- `df.min()` – element minimalny
- `df.max()` – element maksymalny
- `df.mean()` – średnia zbioru
- `df.median()` – mediana zbioru





# Zadanie 1

Stwórz dataframe z dziesięcioma imionami uczniów i uczennic oraz liczbą punktów, jakie uzyskali z egzaminu. Następnie sprawdź, jaka była średnia i mediana wyników.



# Zadanie 2

Bazując na ramce danych utworzonej w poprzednim zadaniu, wyświetl czwarty wiersz. Następnie zapisz do osobnej ramki tych uczniów, którzy uzyskali wynik powyżej średniej.

# Pandas – operacje na danych

Wszystkie poniższe operacje dla dataframe'u zwrócą wartości dla każdej z kolumn, możemy wybierać kolumny dla których chcemy je wykonać:

- `df.apply(funkcja)` – aplikacja funkcji dla każdej komórki zbioru

```
print(df.apply(lambda x: x.max() - x.min()))
```

```
A    0.863928  
B    0.697781  
C    1.431593  
D    0.000000  
F    4.000000  
dtype: float64
```



# Zadanie 3

Dodaj do ramki trzecią kolumnę, w której znajdzie się procentowy wynik uczniów i uczennic z tego egzaminu (to znaczy, że każda wartość liczbowa musi zostać podzielona przez maksimum punktów, jakie można było na tym egzaminie uzyskać).

Następnie, chcemy zanonimizować kolumnę z imionami: chcemy, by została tylko pierwsza i ostatnia litera imienia. Np. dla imienia "Marcelina" chcemy zachować "M...a"



# Pandas – operacje na danych – grupowanie



Od czasu do czasu trzeba wykonać segmentację bazy danych. Oprócz wyznaczania statystyk dla wszystkich wartości, czasem można te wartości pogrupować. W pandasie służy do tego metoda *groupby*.

- `df.groupby("kolumna").operacja().kolumna`

```
display(cars.groupby('cylinders').mean().horsepower)
```

```
cylinders
3      99.250000
4      78.281407
5      82.333333
6     101.506024
8     158.300971
Name: horsepower, dtype: float64
```

# Pandas – usuwanie danych

Do usuwania danych służy nam komenda `df.drop()` – jako parametr dajemy spis indexów lub kolumn do usunięcia oraz `axis` – czy ma być usunięty index czy kolumna.

Możemy również usuwać dane niepełne (NaNy) komendą `df.dropna()` – parametr `axis` usuwa kolumny lub wiersze.

```
wine = wine.drop(wine.columns[[0,3,6,8,11,12,13]], axis = 1)
wine.head()
```

	Malic acid	Ash	Magnesium	Total phenols	Nonflavanoid phenols	Color intensity	Hue
0	1.71	2.43	127	2.80	0.28	5.64	1.04
1	1.78	2.14	100	2.65	0.26	4.38	1.05
2	2.36	2.67	101	2.80	0.30	5.68	1.03
3	1.95	2.50	113	3.85	0.24	7.80	0.86
4	2.59	2.87	118	2.80	0.39	4.32	1.04



# Pandas – łączenie danych



W rzeczywistości często zamiast korzystać z jednej dużej bazy, lecz łączymy wiele mniejszych (łatwiej jest nimi zarządzać, unikać redundancji, dodatkowo oszczędzamy miejsce na dysku i osiągamy większą szybkość). Dane w bibliotece możemy łączyć na dwa sposoby:

- metoda `df.append(object)` dodaje nowe wiersze na końcu istniejącego dataframe'a
- metoda `df.merge()`, która w swoich założeniach jest bardzo podobna do SQL-owego JOINa – można wybrać metodę łączenia – inner (część wspólna), outer (suma), left, right. Parametr `on` – nazwa kolumny, która ma być łącznikiem.

Dodatkowo, możemy sami podać nową nazwę kolumny i w ten sposób dodać dane do istniejących danych: `df["nowa kolumna"] = dane`



# Pandas – sortowanie danych



- metoda `df.sort_values(by="nazwa kolumny")`
- parametr `ascending` – czy rosnąco czy malejąco
- często po sortowaniu chcemy zresetować indexy – służy do tego metoda `df.reset_index(drop=True)` – parametr ten usuwa stary index





# Zadanie 4

Posortuj ramkę danych względem liczby punktów otrzymanych z egzaminu, od najwyższego do najniższego. W przypadku takiej samej liczby punktów, osoby mają zostać wyświetlone w kolejności alfabetycznej.



# Zadanie 5

Połącz ze sobą dwie poniższe ramki.

```
student_data1 = pd.DataFrame({  
    'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'],  
    'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin']  
})
```


```
student_data2 = pd.DataFrame({  
    'student_id': ['S6', 'S7', 'S8', 'S9', 'S10'],  
    'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha Preston']  
})
```



# Zadanie 6

Do ramki utworzonej w poprzednim zadaniu dodaj informacje o rezultatach studentów z ramki poniżej.

```
exam_results = pd.DataFrame({  
    'student_id': ['S2', 'S10', 'S3', 'S1', 'S7', 'S9', 'S5', 'S4', 'S8', 'S6'],  
    'marks': [200, 210, 190, 222, 199, 201, 200, 198, 219, 201]  
})
```



# Pandas – NaNy

- metoda `df.fillna(wartość)` – wypełnia dane określoną wartością
- metoda `df.dropna()` – usuwa wiersze z pustymi danymi z tabeli

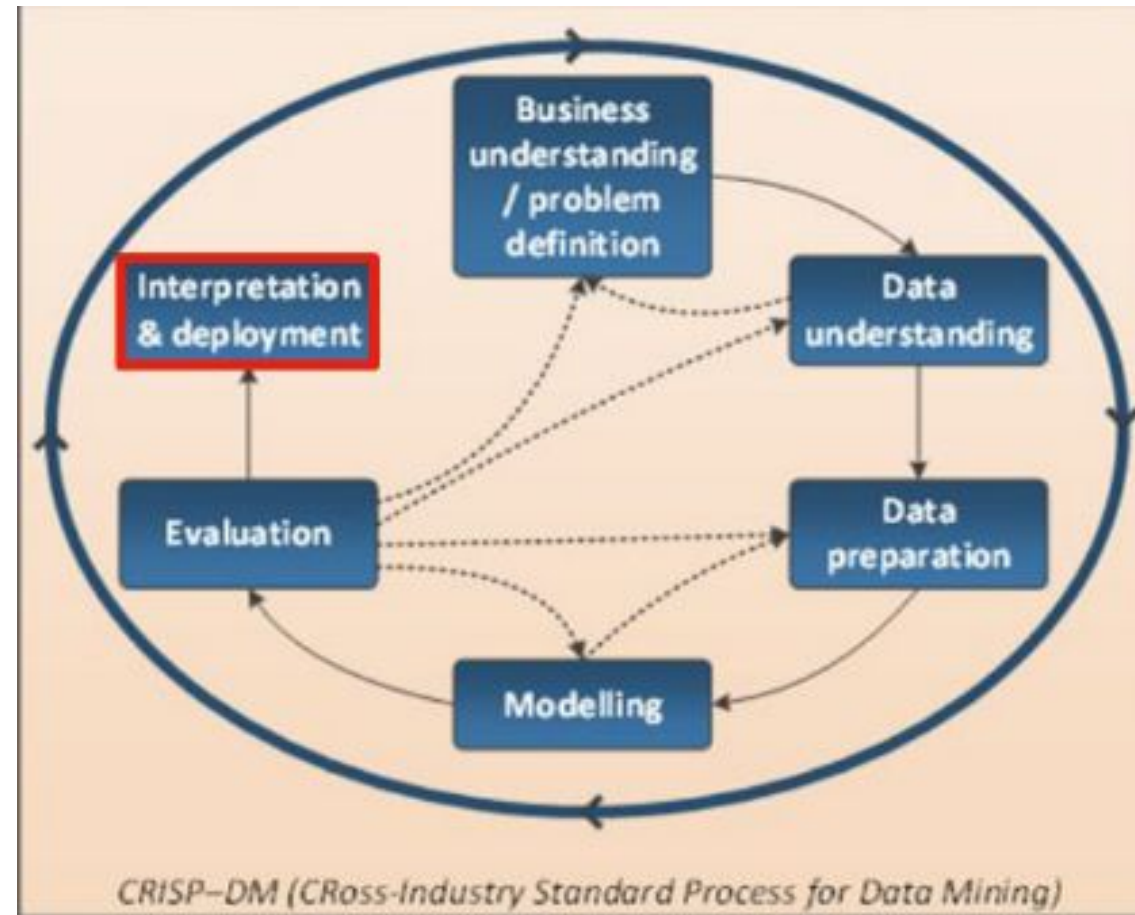


# Proces analizy danych

# Proces analizy danych



# Schemat





# Proces analizy danych

Data understanding – zebranie określonych danych potrzebnych do rozwiązania problemu, opis danych (czasem również etykietyzacja), eksploracja i weryfikacja jakości.

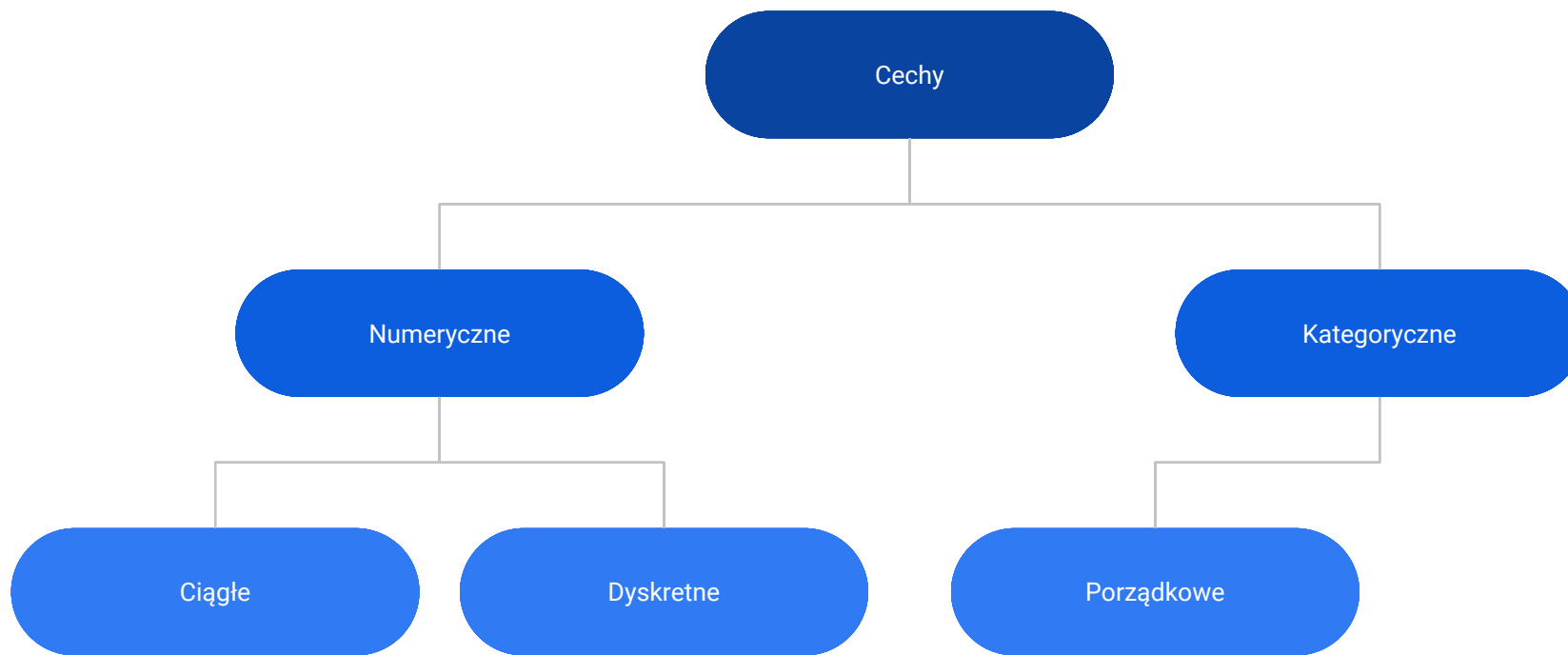
Typy danych – arkusze CSV (na takich będziemy dziś pracować), tekst, obraz, dźwięk, wideo, dane pogodowe, giełdowe, archiwa publiczne itd.

Najważniejsze zadanie – **EKSTRAKCJA CECH (feature engineering)**.

Bardzo dużo zależy od wielkości zbioru i liczby cech – w zależności od tych parametrów podejmiemy różne kroki w procesie analizy danych.



# Cechy – podział





# Metody agregacji danych

Numerycznych:

- średnia
- suma,
- max, min, odchylenie standardowe

Kategorycznych:

- liczenie wystąpień
- szukanie najczęstszych
- wyniki procentowe np. 23% populacji to blondyni



# Przygotowanie danych – wartości niekompletne



Wartości niekompletne – jak sobie z tym poradzić:

- ignorować
- wstawić wartość “nieznana”
- ręcznie uzupełniać na podstawie zgromadzonej wiedzy
- usuwać rekordy z niekompletnymi danymi
- uzupełniać algorytmicznie – poszukiwać najbliższego sąsiada, wyciągać średnią, wstawiać wartości losowe, zadania klasyfikacji



# Przygotowanie danych – wartości niekompletne



```
threshold = 0.7

#Dropping columns with missing value rate higher than threshold
data = data[data.columns[data.isnull().mean() < threshold]]

#Dropping rows with missing value rate higher than threshold
data = data.loc[data.isnull().mean(axis=1) < threshold]
```



# Przygotowanie danych – wartości niekompletne

Wprowadzenie nowych wartości w miejsce NaNów

```
#Filling all missing values with 0  
data = data.fillna(0)
```

```
#Filling missing values with medians of the columns  
data = data.fillna(data.median())
```



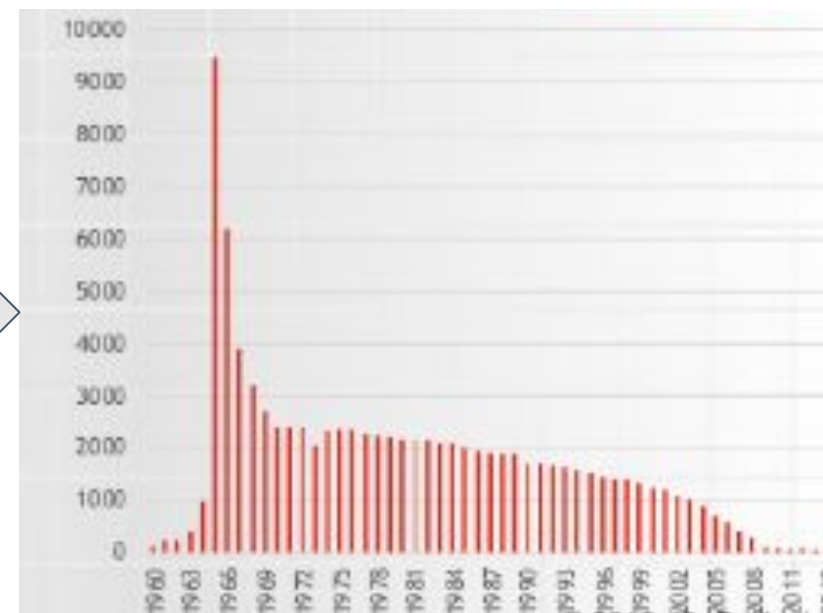
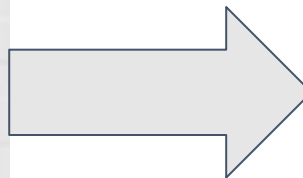
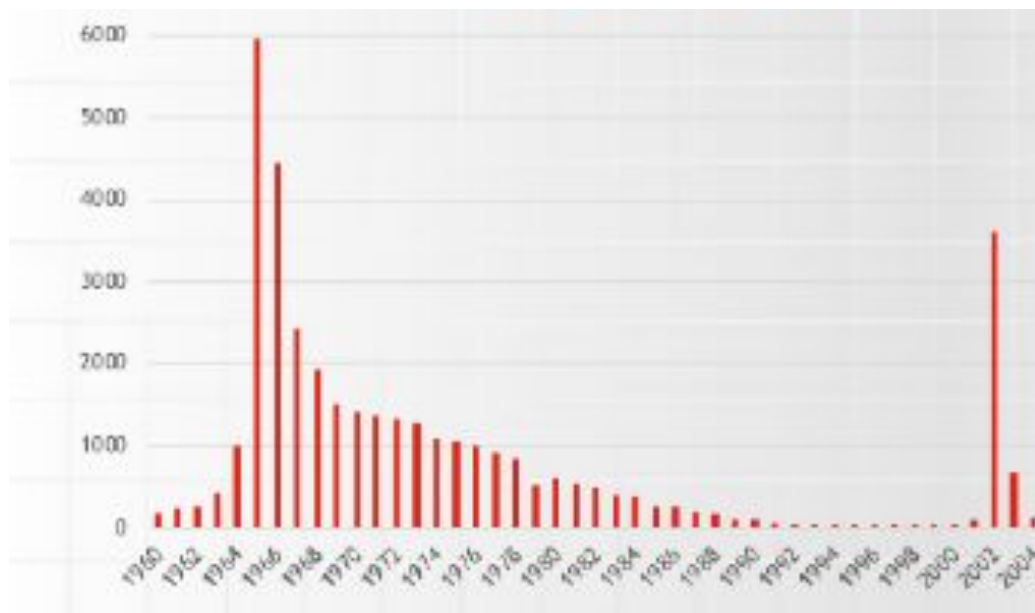
# Przygotowanie danych – wartości niekompletne

Wprowadzenie nowych wartości w miejsce NaNów – cechy katégoryczne

```
#Max fill function for categorical columns  
data['column_name'].fillna(data['column_name'].value_counts()  
.idxmax(), inplace=True)
```

# Przygotowanie danych – czyszczenie (usuwanie szumów)

Czyszczenie danych – usuwanie błędów, szumów, wartości odstających





# Przygotowanie danych - czyszczenie (usuwanie szumów)



Usuwanie wartości odstających za pomocą odchylenia standardowego

```
#Dropping the outlier rows with standard deviation
factor = 3
upper_lim = data['column'].mean () + data['column'].std () *
factor
lower_lim = data['column'].mean () - data['column'].std () *
factor

data = data[(data['column'] < upper_lim) & (data['column'] >
lower_lim)]
```





# Przygotowanie danych - czyszczenie (usuwanie szumów)



Usuwanie wartości odstających za pomocą percentyli

```
#Dropping the outlier rows with Percentiles
upper_lim = data['column'].quantile(.95)
lower_lim = data['column'].quantile(.05)

data = data[(data['column'] < upper_lim) & (data['column'] >
lower_lim)]
```



# Przygotowanie danych – czyszczenie (usuwanie szumów)



Ograniczanie wartości odstających zamiast usuwania – małe zbiory danych  
(dla dużych usuwanie zbędnych informacji nie jest problemem)

```
#Capping the outlier rows with Percentiles
upper_lim = data['column'].quantile(.95)
lower_lim = data['column'].quantile(.05)

data.loc[(df[column] > upper_lim), column] = upper_lim
data.loc[(df[column] < lower_lim), column] = lower_lim
```



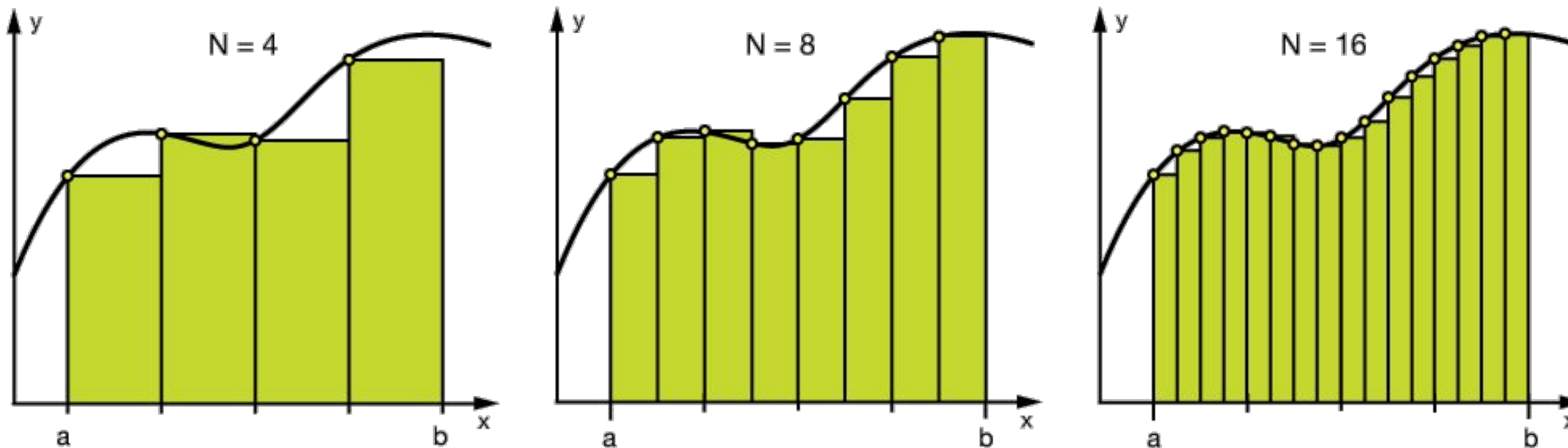
# Przygotowanie danych - standaryzacja

Standaryzacja polega na ujednoliceniu różnych formatów danych w zbiorze:

- styl kodowania języka: UTF-8, ANSI itp.
- daty: 2019-09-12, 12 wrz 2019, 120919 itp.
- Zielona Góra, ZG, Z. Góra
- AGH, Akademia Górniczo Hutnicza
- Nerozpoznawalne znaki

# Przygotowanie danych - dyskretyzacja

Ciągła dystrybucja danych jest niedobra dla większości algorytmów uczenia maszynowego





# Przygotowanie danych - dyskretyzacja



## #Numerical Binning Example

Value	Bin
0-30 ->	Low
31-70 ->	Mid
71-100 ->	High

## #Categorical Binning Example

Value	Bin
Spain ->	Europe
Italy ->	Europe
Chile ->	South America
Brazil ->	South America



# Przygotowanie danych – transformacja logarytmiczna



- pomaga poradzić sobie z wypaczonymi danymi
- porządkuje wielkość danych
- zmniejsza wpływ wartości odstających
- zwiększa niezawodność modelu
- trzeba zająć się wartościami ujemnymi

# Przygotowanie danych - transformacja logarytmiczna

```
#Log Transform Example
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})

data['log+1'] = (data['value']+1).transform(np.log)

#Negative Values Handling
#Note that the values are different
data['log'] = (data['value']-data['value'].min()+1)
               .transform(np.log)
```

	value	log(x+1)	log(x-min(x)+1)
0	2	1.09861	3.25810
1	45	3.82864	4.23411
2	-23	nan	0.00000
3	85	4.45435	4.69135
4	28	3.36730	3.95124
5	2	1.09861	3.25810
6	35	3.58352	4.07754
7	-12	nan	2.48491

# Przygotowanie danych - One - hot encoding

Kodowanie 1 z n

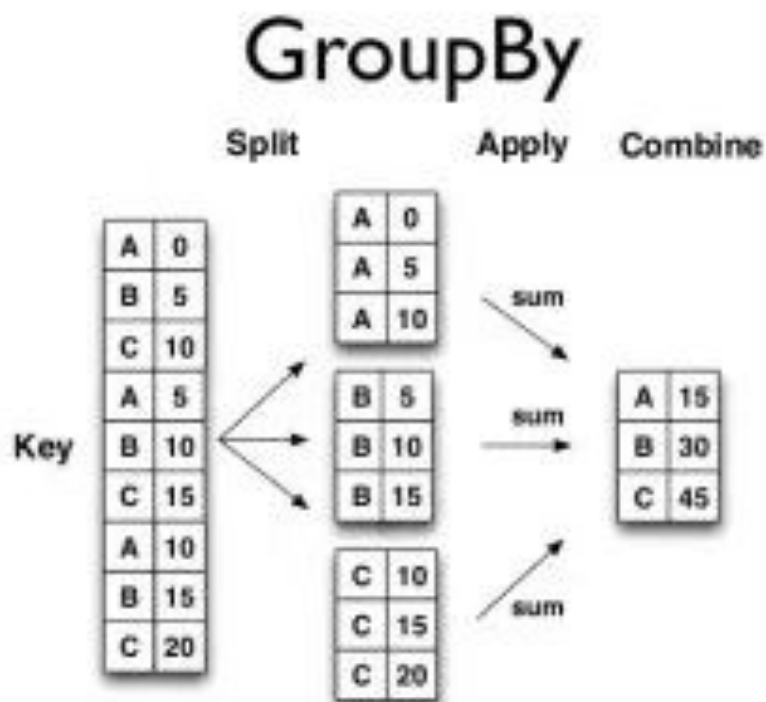
User	City
1	Roma
2	Madrid
1	Madrid
3	Istanbul
2	Istanbul
1	Istanbul
1	Roma



User	Istanbul	Madrid
1	0	0
2	0	1
1	0	1
3	1	0
2	1	0
1	1	0
1	0	0



# Przygotowanie danych – grupowanie



```
display(cars.groupby('cylinders').mean().horsepower)
display(cars.groupby('modelyear').max()[['acceleration']])
```

```
cylinders
3      99.250000
4      78.281407
5      82.333333
6     101.506024
8     158.300971
Name: horsepower, dtype: float64
```

**acceleration**

```
modelyear
70      20.5
71      20.5
72      23.5
73      21.0
74      21.0
```

# Przygotowanie danych – rozbijanie danych (dzielenie)

Przydatne, gdy mamy wiele cech w jednej kolumnie

```
data.name
0  Luther N. Gonzalez
1  Charles M. Young
2  Terry Lawson
3  Kristen White
4  Thomas Logsdon

#Extracting first names
data.name.str.split(" ").map(lambda x: x[0])
0  Luther
1  Charles
2  Terry
3  Kristen
4  Thomas

#Extracting last names
data.name.str.split(" ").map(lambda x: x[-1])
0  Gonzalez
1  Young
2  Lawson
3  White
4  Logsdon
```



# Przygotowanie danych – skalowanie

W większości przypadków cechy numeryczne zestawu danych nie mają określonego zakresu i różnią się od siebie. W rzeczywistości nie ma sensu oczekiwać, że kolumny wieku i dochodów będą miały ten sam zakres. Ale z punktu widzenia uczenia maszynowego, ten sam zakres pomaga usprawnić model.

Skalowanie rozwiązuje ten problem. Funkcje ciągłe stają się identyczne pod względem zakresu po procesie skalowania. Ten proces nie jest obowiązkowy w przypadku wielu algorytmów, ale nadal warto go zastosować. Jednak algorytmy oparte na obliczeniach odległości, takich jak k-NN lub k-Means, muszą mieć skalowane ciągłe funkcje jako dane wejściowe modelu.

Zasadniczo istnieją dwa popularne sposoby skalowania: **normalizacja i standaryzacja**.

# Przygotowanie danych - normalizacja

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})  
  
data['normalized'] = (data['value'] - data['value'].min()) /  
                    (data['value'].max() - data['value'].min())
```

	value	normalized
0	2	0.23
1	45	0.63
2	-23	0.00
3	85	1.00
4	28	0.47
5	2	0.23
6	35	0.54
7	-12	0.10

# Przygotowanie danych - standaryzacja

$$z = \frac{x - \mu}{\sigma}$$

```
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]}))
```

```
data['standardized'] = (data['value'] - data['value'].mean()) /  
data['value'].std()
```

	value	standardized
0	2	-0.52
1	45	0.70
2	-23	-1.23
3	85	1.84
4	28	0.22
5	2	-0.52
6	35	0.42
7	-12	-0.92

# Przygotowanie danych - wyodrębnianie daty

```
from datetime import date

data = pd.DataFrame({'date':
['01-01-2017',
'04-12-2008',
'23-06-1988',
'25-08-1999',
'20-02-1993',
]})

#Transform string to date
data['date'] = pd.to_datetime(data.date, format="%d-%m-%Y")

#Extracting Year
data['year'] = data['date'].dt.year

#Extracting Month
data['month'] = data['date'].dt.month

#Extracting passed years since the date
data['passed_years'] = date.today().year - data['date'].dt.year

#Extracting passed months since the date
data['passed_months'] = (date.today().year - data['date'].dt.year)
* 12 + date.today().month - data['date'].dt.month

#Extracting the weekday name of the date
data['day_name'] = data['date'].dt.day_name()
```

	date	year	month	passed_years	passed_months	day_name
0	2017-01-01	2017	1	2	26	Sunday
1	2008-12-04	2008	12	11	123	Thursday
2	1988-06-23	1988	6	31	369	Thursday
3	1999-08-25	1999	8	20	235	Wednesday
4	1993-02-20	1993	2	26	313	Saturday



# Feature engineering – przykład



Jak ocenić czy dana osoba na facebooku nie jest botem?

- liczba znajomych
- liczba opublikowanych postów
- liczba zdjęć
- liczba polubień
- polubienia pod postami
- długość postów
- pochodzenie numeru telefonu
- domena adresu e-mail

A ocena atrakcyjności filmu na YouTube?



# Feature engineering – realne zadanie



Firma wynajmująca samochody ARRA chce ocenić niezawodność i koszty utrzymania różnych modeli samochodów dostępnych we flocie. Posiada następujące dane:

- tabelę z pojedynczymi aktywnościami wszystkich samochodów we flocie: data, miejsce wypożyczenia, status (w jeździe, awaria), dane wypożyczającego (wiek, płeć, narodowość), ilość przejechanych kilometrów, ilość spalonego paliwa, numer identyfikacyjny auta
- tabelę dotyczącą każdego auta: rok produkcji, silnik, wyposażenie, marka, model itp.

Jakie cechy możemy tutaj znaleźć?





# Ciekawostka

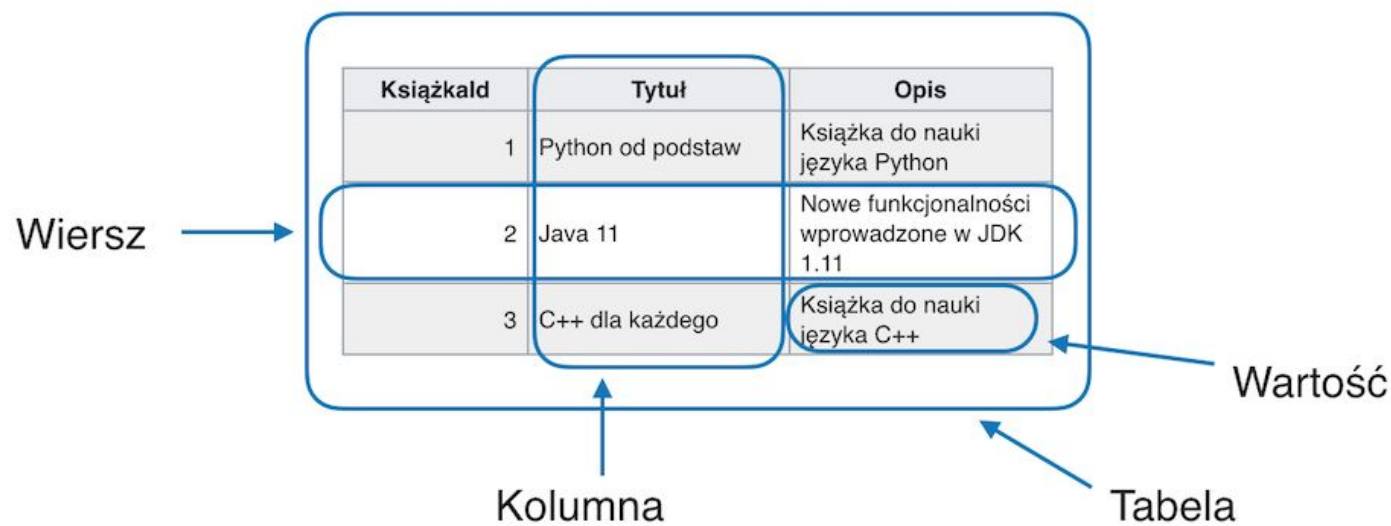
Proces pozyskania, preprocessingu danych i wydobywania z nich cech zajmuje nawet **80% czasu** pracy inżyniera danych.



# KONCEPCJA RELACYJNYCH BAZ DANYCH

# TABELA

- Zgodnie z relacyjną koncepcją baz danych jest to miejsce do przechowywania danych.
- Tabela składa się z wierszy oraz rekordów.
- Każdy wiersz opisuje jeden rekord (pojedyncza informacja/zestaw danych).
- Kolumna natomiast odpowiedzialna jest za opisanie właściwości/struktury danego rekordu.



# KLUCZ GŁÓWNY

W celu jednoznacznej identyfikacji rekordów na potrzeby tabel bazodanowych, wprowadza się identyfikator, który jest określany jako tzw. **klucz główny/podstawowy** (z ang. **primary key**).

Imię	Nazwisko	Data urodzenia
Jan	Kowalski	31-03-1989
Wacław	Nowak	12-08-1968
Ryszard	Lubicz	13-07-1955
Wacław	Nowak	12-08-1968

Id	Imię	Nazwisko	Data urodzenia
1	Jan	Kowalski	31-03-1989
2	Wacław	Nowak	12-08-1968
3	Ryszard	Lubicz	13-07-1955
4	Wacław	Nowak	12-08-1968

# KLUCZ GŁÓWNY

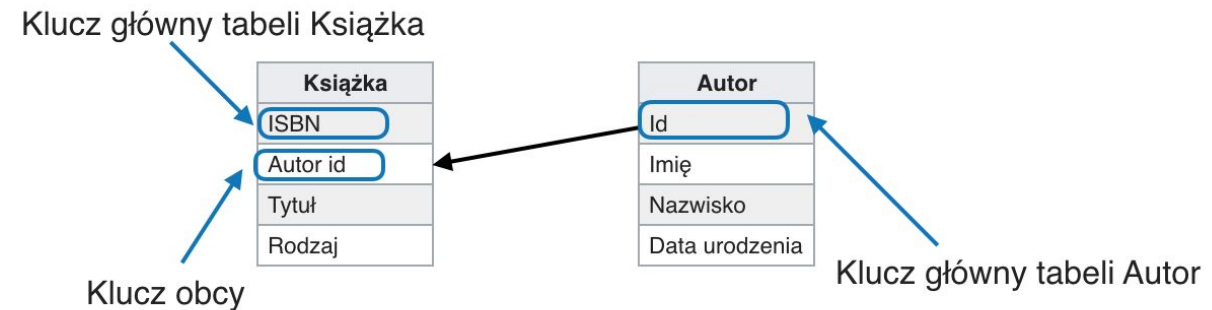
Może być to rzeczywisty (lub tzw. naturalny) identyfikator jak np. numer PESEL. Wielokrotnie jednak w bazach danych implementowany jest abstrakcyjny identyfikator, który nie ma nic wspólnego z rzeczywistą strukturą danych, którą baza ma przechowywać.

Książkald	Tytuł	Opis
1	Python od podstaw	Książka do nauki języka Python
2	Java 11	Nowe funkcjonalności wprowadzone w JDK 1.11
3	C++ dla każdego	Książka do nauki języka C++

↑  
Klucz główny

# RELACJE

W relacyjnych systemach bazodanowych tabele łączone są ze sobą w tzw. relacje, określające rodzaj zależności pomiędzy dwoma nośnikami danych, np. książka i autor. Wiązanie takie realizowane jest za pomocą **klucza obcego (z ang. foreign key)**.





# RODZAJE RELACJI

Zgodnie z teoretycznym relacyjnym modelem baz danych wyróżniamy trzy rodzaje relacji:

- jeden do jednego
- jeden do wielu
- wiele do wielu

# RELACJA 1:1

Rodzaj relacji, który zakłada, że jeden rekord tabeli A może być powiązany tylko z jednym rekordem tabeli B.

Na poniższym przykładzie mamy do czynienia z sytuacją, gdzie jednemu rekordowi z tabeli Osoba odpowiada jeden rekord z tabeli Pesel.

Id	Student id	Wydział	Nazwa
1	2234	Informatyka	Inżynieria oprogramowania
2	3489	Biologia	Mikrobiologia



Numer indeksu	Imię	Nazwisko	Data urodzenia
2234	Tomasz	Nowak	12-09-1987
2256	Wacław	Kowalski	31-04-1988
3489	Tomasz	Nowak	12-09-1987



# RELACJA 1:N

Rodzaj relacji, który zakłada, że jeden rekord tabeli A może być powiązany z wieloma rekordami tabeli B.

W powyższym przykładzie mamy do czynienia z sytuacją, w której jeden klient może dokonać wielu różnych zamówień, natomiast jedno zamówienie może być przypisane tylko i wyłącznie do jednego klienta.

Id	Klient id	Nazwa zamówienia	Data zamówienia
1	1	Macbook Pro	12-03-2020
2	1	Iphone 11	14-03-2020
3	2	Dell XPS 15	15-03-2020



Id	Imię	Nazwisko
1	Jan	Kowalski
2	Wacław	Kowalski
3	Tomasz	Nowak

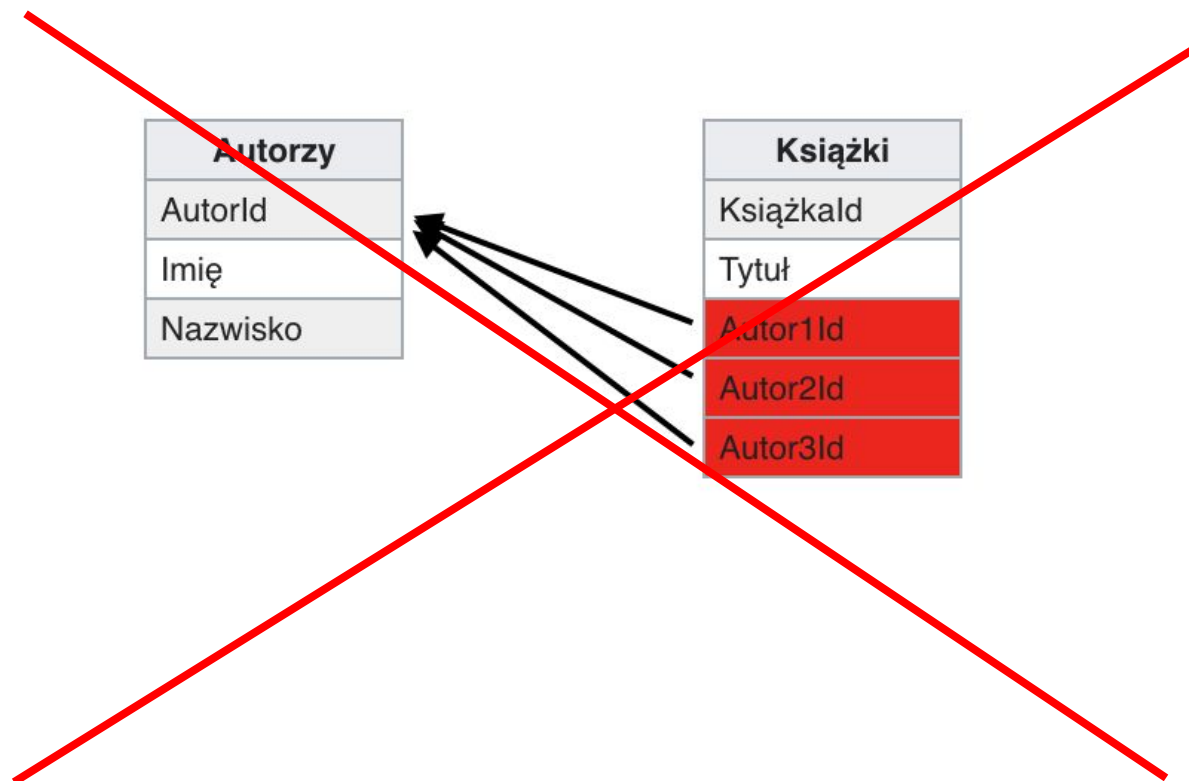
# RELACJA N:M

Rodzaj relacji, który zakłada, że jeden rekord tabeli A może być powiązany z wieloma rekordami tabeli B.

**Błędna** realizacją tej relacji jest przykład

→

Rozwiązanie to przy spełnieniu kilku warunków może działać poprawnie, jednak w przypadku, gdybyśmy chcieli uwzględnić więcej niż trzech autorów książki, to powyższy przykład nie zrealizowałby docelowej funkcjonalności.

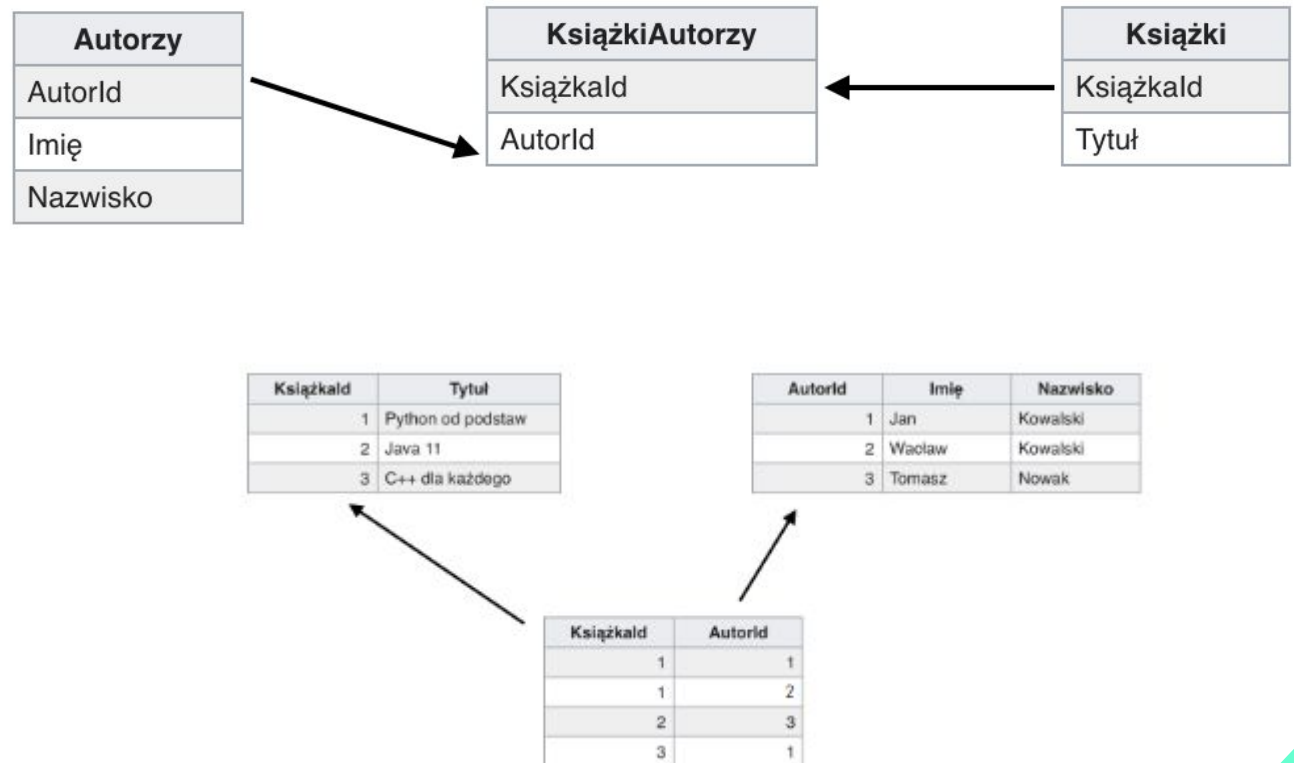


# RELACJA N:M

W praktyce relacje wiele do wielu realizuje się za pomocą dodatkowej tabeli, tzw. tabeli pomocniczej.

W powyższym przykładzie została powołana do życia nowa tabela Książki Autorzy (jej nazwa jest zupełnie dowolna), która jest odpowiedzialna za powiązanie w relację tabeli Autorzy oraz tabeli Książki.

W tym przypadku kolumna KsiążkaId wskazuje na klucz główny tabeli Książki, natomiast kolumna AutorId, wskazuje na kolumnę AutorId.





# **ZASADY PROJEKTOWANIA TABEL**



# Określenie celu

Strukturę projektowanej bazy danych należy dobrać do specyfiki konkretnego systemu. Inaczej będzie modelowany system, w którym dane osobowe są jego istotnym punktem (przygotowanie osobnych tabel dla osób oraz adresów wraz z relacją), a inaczej będzie wizualizowany system, w którym dane osobowe pełnią drugorzędną funkcję i nie są istotnym punktem systemu (wspólna tabela łącząca dane osobowe oraz adresy).

# Unikanie duplikowania danych

Dla przykładu część informacji, jak nazwa producenta oraz adres producenta, powielają się w bazie wielokrotnie. Jest to po pierwsze marnotrawstwem pamięci, a dodatkowo jakakolwiek pomyłka spowoduje brak spójności danych.

W takim przypadku dużo efektywniej jest rozbić daną tabelę na dwie i wprowadzić konkretny rodzaj relacji, np.:

**Należy bezwzględnie unikać sytuacji, w której w kolejnych wierszach znajdują się powielenia tych samych danych.**

Id	Nazwa	Nazwa producenta	Adres producenta
1	Macbook Pro 16	Apple	Cupertino, CA 95014, United States
2	Iphone 11 Pro	Apple	Cupertino, CA 95014, United States
3	Dell XPS 15	Dell	The Boulevard, Cain Road, Bracknell, Berkshire, RG12 1LF

Wydawnictwo Id	Nazwa producenta	Adres producenta
1	Apple	Cupertino, CA 95014, United States
3	Dell	The Boulevard, Cain Road, Bracknell, Berkshire, RG12 1LF

Id	Nazwa	WydawnictwoId
1	Macbook Pro 16	1
2	Iphone 11 Pro	1
3	Dell XPS 15	3



# Atomowość informacji

Każde pole bazy danych powinno zawierać pojedyncze informacje (tzn. atomowe). Atomowość jest zależna od konkretnego systemu, natomiast w skrócie, polega ona na minimalizowaniu ilości przechowywanych informacji w danym polu.

Dla powyższego przykładu adres producenta zawiera dużą ilość informacji i np. wyszukiwanie konkretnego produktu po adresie może być mocno utrudnione. W tym celu należałoby rozbić adres producenta na takie informacje jak Kraj, czy Region.

Wydawnictwo Id	Nazwa producenta	Adres producenta
1	Apple	Cupertino, CA 95014, United States
3	Dell	The Boulevard, Cain Road, Bracknell, Berkshire, RG12 1LF

Id	Nazwa	WydawnictwoId
1	Macbook Pro 16	1
2	Iphone 11 Pro	1
3	Dell XPS 15	3

# Multiplikowanie odwołań

Mając np. sytuację, w której klient wypożycza wiele sprzętu komputerowego, może zrealizować te założenia za pomocą tabeli ->

KlientId	ProduktId	Data
1	3, 6, 90	14-03-2020
2	5	16-03-2020
3	4, 11, 15	17-08-2020

W powyższym przykładzie pole ProduktId zawiera informację o wielu produktach. Jednak forma ta jest **niepoprawna** ze względu na dużą trudność uzyskania wszystkich niezbędnych danych statystycznych, np. wyodrębnienia informacji, ile razy dany produkt został wypożyczony. Prawidłowa struktura tabeli powinna wyglądać jak obok ->

KlientId	ProduktId	Data
1	3	14-03-2020
1	6	14-03-2020
1	90	14-03-2020
2	5	16-03-2020
3	4	17-08-2020
3	11	17-08-2020
3	15	17-08-2020



# Unikanie pustych pól

W tabelach należy unikać pozostawiania pustych pól (niezawierających danych). W pewnych określonych sytuacjach puste pola mogą być jednak nieuniknione. ->

W powyższym przypadku 75% rekordów posiada pustą zawartość pola Uwagi. Dla powyższego przypadku zdecydowanie lepiej jest wprowadzić dodatkową tabelę, która będzie odpowiedzialna za przechowywanie informacji dodatkowych oraz redukcję pustych pól.

Zamówienield	KlientId	ProduktId	Data	Uwagi
1	1	3	14-03-2020	
2	1	6	14-03-2020	odbiór osobisty dopiero po 29.03.2020
3	1	90	14-03-2020	
4	2	5	16-03-2020	

Zamówienield	KlientId	ProduktId	Data
1	1	3	14-03-2020
2	1	6	14-03-2020
3	1	90	14-03-2020
4	2	5	16-03-2020



Zamówienield	Uwagi
2	odbiór osobisty dopiero po 29.03.2020



# Unikalne identyfikatory rekordów

Każdy rekord tabeli powinien być **jednoznacznie identyfikowany**. W przeciwnym przypadku nie będzie istniała możliwość ich rozróżniania. Klucz główny musi być dobrze zidentyfikowany. Czasami jego abstrakcyjna forma może zostać zastąpiona rzeczywistym identyfikatorem, np. ISBN dla książek, PESEL dla obywateli Polski. Wszystko zależy przede wszystkim od kontekstu modelowanego systemu.



# Zadanie 7

1. Jakie tabele będą nam potrzebne, aby stworzyć szkielet bazy danych do zarządzania wypożyczalnią samochodów.
2. Jakie relacje powinny być między tabelami samochodów, klientów i rezerwacji?
3. Co może być kluczem głównym w poszczególnych tabelach?

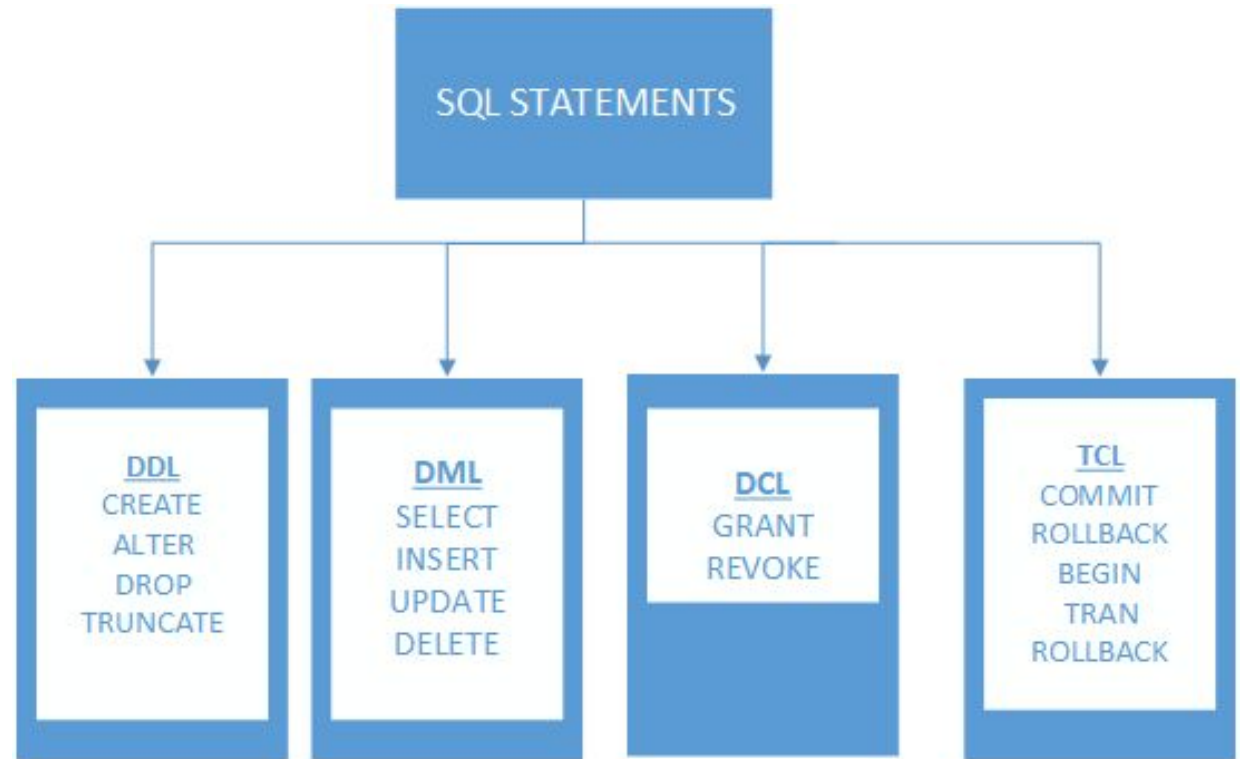
# SQL

SQL to strukturalny język zapytań, który umożliwia wykonywanie operacji na relacyjnych bazach danych. Jest to język uniwersalny wykorzystywany m.in. w MySQL, PostgreSQL, Microsoft Server SQL i wielu innych.



# SQL – Rodzaje zadań

- DDL (Data Definition Language) – język definiowania danych; realizuje definiowanie struktury danych
- DML (Data Manipulation Language) – język manipulacji danych; realizuje pobieranie i modyfikowanie danych
- DCL (Data Control Language) – język kontroli dostępu do danych
- TCL (Transaction Control Language) – język kontroli transakcji; umożliwia zarządzanie grupami zapytań DML.





# TYPY DANYCH

Każda kolumna tabeli bazodanowej ma zdefiniowany typ, który określa rodzaj danych, które można przechowywać w danej komórce.

Wyróżniamy:

- **typy liczbowe: INT, FLOAT, DOUBLE**
- **typy łańcuchowe: VARCHAR, TEXT**
- **data i czas: DATE, TIME**



# **DDL – DATA DEFINITION LANGUAGE**



# CREATE

Polecenie SQL umożliwiające tworzenie tabel ->

Nazwy tabel w wersji podstawowej, mogą zawierać:

- litery
- cyfry
- znaki \$, \_
- znaki o kodach: U+0080-U+FFFF
- małe, jak i wielkie litery - niemniej jednak to, czy ich interpretacja będzie: case sensitive, czy incase sensitive zależy od serwera i konfiguracji bazy danych.

Nazwy tabel nie mogą jednak:

- składać się z samych cyfr
- składać się z samych słów kluczowych języka SQL (chyba, że znajdują się pomiędzy apostrofami)
- kończyć się znakiem spacji.

```
CREATE TABLE nazwa_tabeli(  
    nazwa_kolumny_1 typ_kolumny_1 [atrybuty],  
    nazwa_kolumny_2 typ_kolumny_2 [atrybuty],  
    nazwa_kolumny_3 typ_kolumny_3 [atrybuty]  
    ...  
);
```

```
CREATE TABLE Produkt(  
    id INTEGER,  
    Nazwa VARCHAR(20),  
    Producent VARCHAR(25)  
);
```



# ATRYBUTY KOLUMN

Tabele tworzone za pomocą polecenia SQL, mogą deklarować różne atrybuty dla definiowanych kolumn.

- PRIMARY KEY
- NOT NULL
- AUTO\_INCREMENT
- DEFAULT
- INDEX
- UNIQUE



# SHOW



Za pomocą polecenia SHOW istnieje możliwość zwracania informacji o strukturze danej tabeli:

Zamiast korzystać z SHOW COLUMNS, możemy wykorzystać polecenie DESCRIBE, które posiada jednak mniej opcji wybierania szczegółów.

```
SHOW COLUMNS FROM tabela  
[FROM baza_danych]  
[LIKE `wzorzec_tabeli`];
```

```
SHOW COLUMNS FROM Produkt.sda;
```

```
SHOW COLUMNS FROM Produkt.sda  
LIKE 'nazwa%';
```



# ALTER

Za pomocą polecenia ALTER istnieje możliwość zmiany struktury istniejącej tabeli:

Instrukcja ta umożliwia:

- dodawanie i usuwanie kolumn
- modyfikowanie typów, nazw
- dodawanie indeksów
- usuwanie indeksów

Słowa kluczowe FIRST i AFTER pozwalają umiejscowić kolumnę w określonej strukturze tabeli, np.:

```
ALTER TABLE nazwa_tabeli zmiana1[, zmiana2[, ...]];
```

```
ALTER TABLE nazwa_tabeli  
ADD [COLUMN] definicja_kolumny  
[FIRST | AFTER nazwa_kolumny];
```

```
ALTER TABLE Produkt  
ADD opis_produkta VARCHAR(255)  
AFTER cena_produkta;
```



# DROP

Usuwanie tabeli realizowane jest za pomocą instrukcji DROP.

Istnieje możliwość usunięcia jednej tabeli bądź też całego zestawu tabel, pod warunkiem, że tabela rzeczywiście istnieje.

```
DROP TABLE [IF EXISTS] tabela1, tabela2, ..., tabelaN;
```

```
DROP TABLE Produkt;
```

# KLUCZE OBCE

# TWORZENIE KLUCZA OBCEGO

W celu złączenia tabel w relacje, należy określić w łączonych tabelach klucz obcy. Może to być realizowane podczas tworzenia tabeli:

Klucz obcy można też wprowadzić w istniejącej tabeli:

```
CREATE TABLE Produkt
(
    Id INTEGER PRIMARY KEY,
    Nazwa VARCHAR(20),
    ProducentId INTEGER,
    CONSTRAINT producentId_fk FOREIGN KEY (ProducentId)
REFERENCES Producent(Id)
);
```

```
ALTER TABLE Produkt ADD CONSTRAINT producent_fk FOREIGN
KEY (ProducentId) REFERENCES Producent(Id);
```



# USUWANIE KLUCZA OBCEGO

Klucz obcy można usunąć za pomocą poniższego polecenia:

```
ALTER TABLE Produkt DROP FOREIGN KEY producent_fk;
```





# Zadanie 8



1. Utwórz bazę danych car\_rental i upewnij się, że aktualnie będzie ona bazą domyślną do wykonywania operacji. (USE car\_rental)
2. Dodaj odpowiednie tabele zawierające:
  - a. cars – car\_id, producer, model, year, horse\_power, price\_per\_day
  - b. clients – client\_id, name, surname, address, city
  - c. bookings – booking\_id, client\_id, car\_id, start\_date, end\_date, total\_amount
3. \*Dodaj autoinkrementację dla kluczy.
4. \*Zaktualizuj tabelę bookings o dwa klucze obce, które posiada.

# Odpowiedź

1)

```
CREATE DATABASE car_rental;  
USE car_rental;
```

2)

```
CREATE TABLE cars  
(  
    car_id INTEGER PRIMARY KEY,  
    producer VARCHAR(30),  
    model VARCHAR(30),  
    year INTEGER,  
    horse_power INTEGER,  
    price_per_day INTEGER  
);
```

```
CREATE TABLE clients
```

```
(  
    client_id INTEGER PRIMARY KEY,  
    name VARCHAR(30),  
    surname VARCHAR(30),  
    address TEXT,  
    city VARCHAR(30)  
);
```

```
CREATE TABLE bookings
```

```
(  
    booking_id INTEGER PRIMARY KEY,  
    client_id INTEGER,  
    car_id INTEGER,  
    start_date DATE,  
    end_date DATE,  
    total_amount INTEGER  
);
```

3)

```
ALTER TABLE clients MODIFY COLUMN client_id INTEGER  
AUTO_INCREMENT;  
ALTER TABLE cars MODIFY COLUMN car_id INTEGER  
AUTO_INCREMENT;  
ALTER TABLE bookings MODIFY COLUMN booking_id INTEGER  
AUTO_INCREMENT;
```

4)

```
ALTER TABLE bookings  
ADD CONSTRAINT client_id_fk FOREIGN KEY (client_id)  
REFERENCES clients (client_id),  
ADD CONSTRAINT car_id_fk FOREIGN KEY (car_id)  
REFERENCES cars (car_id);
```



# Dziękujemy!

