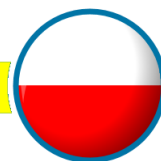




# **Data Science: Wprowadzenie do statystyki z Pythonem**

**Podręcznik kursowy**

Mobilo © 2023



W tym miejscu zwykle pojawia się informacja o tym kto i jak może posługiwać się tym podręcznikiem. Bez regułek prawnych odwołam się po prostu do kilku prostych zasad, które oddają sens kto, kiedy i jak może wg zamysłu autora korzystać z tego materiału:

- Ten podręcznik jest integralnym elementem kursu online.
- Możesz z niego korzystać będąc uczestnikiem tego kursu. Podręcznik jest dla Ciebie i korzystaj z niego do woli – drukuj, wypełniaj, uzupełniaj, przeglądaj, póki Twoim celem jest samodzielne opanowanie tematu.
- Proszę nie umieszczaj go w publicznie dostępnych miejscach, jak blogi, repozytoria git hub, chomik itp.
- Nie wykorzystuj go w innych celach, np. organizacji własnych szkoleń, gdzie występujesz np. jako instruktor.
- Jeśli nie posłuchasz moich próśb, to jako autor zapiszę się do ZAKS-u i następnym razem kupując smartfona zapłacisz za niego kilkaset złotych więcej 😊, więc może lepiej po prostu przestrzegaj praw autorskich 😊

Z góry dziękuję!

Rafał Mobilo © 2023

Zapraszam do odwiedzenia strony:

<http://www.kursyonline24.eu/>

[Review 2023-05-13](#)

## Spis treści

Wprowadzenie – o kursie .....	5
Jak się uczyć? .....	6
Konfiguracja środowiska .....	7
Statystyka, populacja, próba.....	8
Python – populacja i próba .....	10
Typy danych .....	12
Wykresy dla typów danych kategoriycznych .....	14
Wykresy dla typów numerycznych - histogram.....	17
Wykres dla dwóch kategorii – side by side bar i scatter plot.....	20
Średnia, mediana, dominanta, skośność .....	23
Średnia ważona i ucinana .....	25
Wariancja .....	27
Odchylenie standardowe i współczynnik zmienności.....	29
Kwartyle, wykresy pudełkowe i usuwanie outlierów .....	31
Detekcja outlierów i nieco uczenia maszynowego .....	34
Kowariancja i korelacja .....	39
Rozkład danych i nieco hazardu.....	43
Rozkład normalny (Gaussowski, Bell-Curve).....	45
Centralne twierdzenie graniczne (Central Limit Theorem).....	47
Przedział pewności i współczynnik ufności – rozkład normalny.....	50
Przedział pewności i współczynnik ufności – rozkład T-studenta.....	53
Przedział pewności w wyznaczeniu proporcji w populacji .....	56
Przedział pewności dla 2 prób zależnych .....	58
Przedział pewności dla dwóch zbiorów niezależnych.....	61
Testowanie hipotez.....	64
Błędy pierwszego i drugiego typu. P-value .....	67
Testowanie hipotezy dla zbiorów zależnych.....	70
Testowanie hipotezy dla zbiorów niezależnych.....	73
Spróbuj też!.....	78

Ta strona celowo jest pusta

## Wprowadzenie – o kursie

Bez statystyki nie byłoby sztucznej inteligencji, ale nie tylko. Statystyka jest wykorzystywana w procesie badań naukowych, podejmowania decyzji w przedsiębiorstwach, w określaniu polityki współczesnych państw. Nie bez powodu pierwsze spisy powszechne były przeprowadzane już w starożytnym Egipcie i Chinach, a i my pewnie teraz interesujemy się średnimi zarobkami, efektem cieplarnianym, czy wzrostem PKB. Tam wszędzie jest statystyka.

Ale wracając do tych nowocześniejszych zastosowań. Jeśli na poważnie chcesz podejść do zrozumienia działania algorytmów uczenia maszynowego, czy data science, to przygotuj się na to, że trzeba będzie znać wiele statystycznych pojęć, jak chociażby: wariancja, odchylenie standardowe, korelacja, rozkład danych, istotność statystyczna i inne. Na dodatek programując, czy analizując te tematy z pewnością skorzystasz z dokumentacji, która będzie powoływać się na zależności statystyczne i to może jeszcze po angielsku.

Mając to na uwadze, w tym kursie znajdziesz

- Kurs statystyki od podstaw, zaczynający się od wyjaśnienia głównych pojęć, terminologii, omawiający najpopularniejsze miary statystyczne i przedstawiający dość zaawansowane tematy jak przedziały pewności i testowanie hipotez
- Kod w języku Python, który pokazuje, jak korzystać z licznych bibliotek pozwalających pracować z danymi, rysować wykresy, wyliczać wartości statystyk
- Podręcznik ze streszczeniem lekcji, zadaniami do samodzielnego rozwiązania i propozycjami rozwiązań tych zadań

Kończąc ten kurs, będziesz

- rozumieć najważniejsze pojęcia statystyczne,
- stosować pythonowe funkcje i metody pozwalające na wyznaczanie statystyk,
- znać najpopularniejsze symbole i wzory statystyczne, które czasami nieźle potrafią zamieszać w głowie,
- zrozumiesz również „na intuicję” znaczenie terminologii ze świata statystyki.

Statystyka nie jest prosta, dlatego ta intuicja jest ważna. Nie stosuję tu wyrafinowanego naukowego języka (choć oczywiście trochę go używać trzeba) ale staram się mówić językiem dość potocznym, podawać przykłady z życia codziennego. Biorąc też pod uwagę przewagę języka angielskiego w świecie nauki, terminologię przedstawiam i po polsku i po angielsku.

W data science lepiej radzą sobie ci, którzy znają statystykę, dlatego nie czekaj. Przejrzyj wymagania kursowe, zapoznaj się ze spisem treści, obejrzyj przykładowe lekcje i w zwiększ swoje szanse na powodzenie w świecie data science!

Statystycznie rzecz ujmując – lepiej znać statystykę!

Zapraszam na kurs! Miłej nauki!

Rafał

## Jak się uczyć?

Skoro tutaj zaglądasz, to znaczy, że planujesz poznać statystykę i to z Pythonem. To świetnie!

Naukę oczywiście zorganizujesz sobie po swojemu, ale pozwól, że zaproponuję kilka sposobów nauki, a Ty sam/a wybierzesz, co z tego Ci się podoba, a co wolisz zrobić po swojemu

- 1) **Co za dużo to niezdrowo** – nie rób na raz za dużo materiału. Nie od razu Kraków zbudowano. Jedna lub dwie lekcje na dzień powinny wystarczyć.
- 2) **Liczy się regularność** – niekoniecznie uczyć trzeba się codziennie, ale jeśli postanowisz przerabiać lekcje we wtorki, czwartki i soboty to już coś!
- 3) **Wykonuj zadania praktyczne**. Od samego oglądania filmów się nie nauczysz. Nie martw się jeśli nie wiesz, o co chodzi autorowi w zadaniu. Zajrzyj do propozycji rozwiązań, na początku tylko przeanalizuj kod. Za jakiś czas wróć i spróbuj zaimplementować go samodzielnie.
- 4) **Zmieniaj treść poleceń na własną rękę**. Wykonaj podobny przykład na innych danych. Im więcej kreatywności podczas nauki, tym więcej się zapamiętuje
- 5) Uczę się uczyć, a do głowy nie wchodzi – wszyscy tak mamy i to pewnie dlatego nauka w szkole trwa aż tyle lat! **Od czasu do czasu zrób sobie powtórkę**. Przecież nikt Cię nie goni i nie rozlicza z postępów. Wracaj do zadań i rozwiązuj je wielokrotnie.
- 6) Jeśli zadania sprawiają problem, wróć do notatki lub lekcji – zobaczysz, że słuchając drugi raz tego samego, materiał nie będzie już taki trudny
- 7) Notatki w podręczniku są dla Twojej wygody. Niestety wygoda leży blisko lenistwa. Nie bądź leniem. Przygotuj sobie zeszyt lub kilka luźnych kartek i **zapisuj to czego się uczysz**. To co wejdzie oczami lub uszami, będzie wychodzić rękami i... nie ma wyjścia – po drodze zahaczy o mózg 😊
- 8) Jeśli możesz – wydrukuj sobie podręcznik, dopisuj do niego własne notatki, uwagi itp.
- 9) Kiedy osiągniesz jakiś „kamień milowy”, ukończysz sekcję kursu, a może nawet cały kurs – **daj sobie nagrodę** – to niesamowicie zwiększa motywację!
- 10) **Nie bój się korzystać z innych materiałów**: książek, blogów, forów itp. Część z nich na początku może być nieco za trudna, ale nic nie stoi na przeszkodzie, żeby na początku nic nie mówić, tylko słuchać 😊.
- 11) Kiedy już ukończysz kurs – **zaktualizuj CV na LinkedIn**, pochwal się swoim certyfikatem, daj się odnaleźć rekruterom, zaproś mnie do znajomych (link w profilu). Chętnie potwierdzę Twoją nową umiejętność!

Powodzenia!

Rafał

## Konfiguracja środowiska

Kurs możesz przejść korzystając z dowolnych środowisk programistycznych. Dobrze wiemy, że skrypty można pisać nawet w notatniku, tylko to mało wygodne. Potraktuj więc poniższe uwagi tylko jako propozycje:

- Visual Studio Code to środowisko programistyczne firmowane przez Microsoft. Słynie z wielu rozszerzeń, między innymi do Pythona
- Warto utworzyć środowisko wirtualne. Każdy projekt może mieć przypisane własne środowisko wirtualne, dzięki czemu nie dochodzi do konfliktów między projektami

```
python -m venv venv
```

Środowisko wirtualne trzeba aktywować:

```
.\venv\scripts\activate.bat
```

W systemie Linux użyj:

```
source venv/bin/activate
```

Aby w środowisku zainstalować nowe moduły korzysta się z pip

```
pip install pandas numpy matplotlib
```

Do udokumentowania zainstalowanych w środowisku modułów posłuż się poleceniem

```
pip freeze > requirements.txt
```

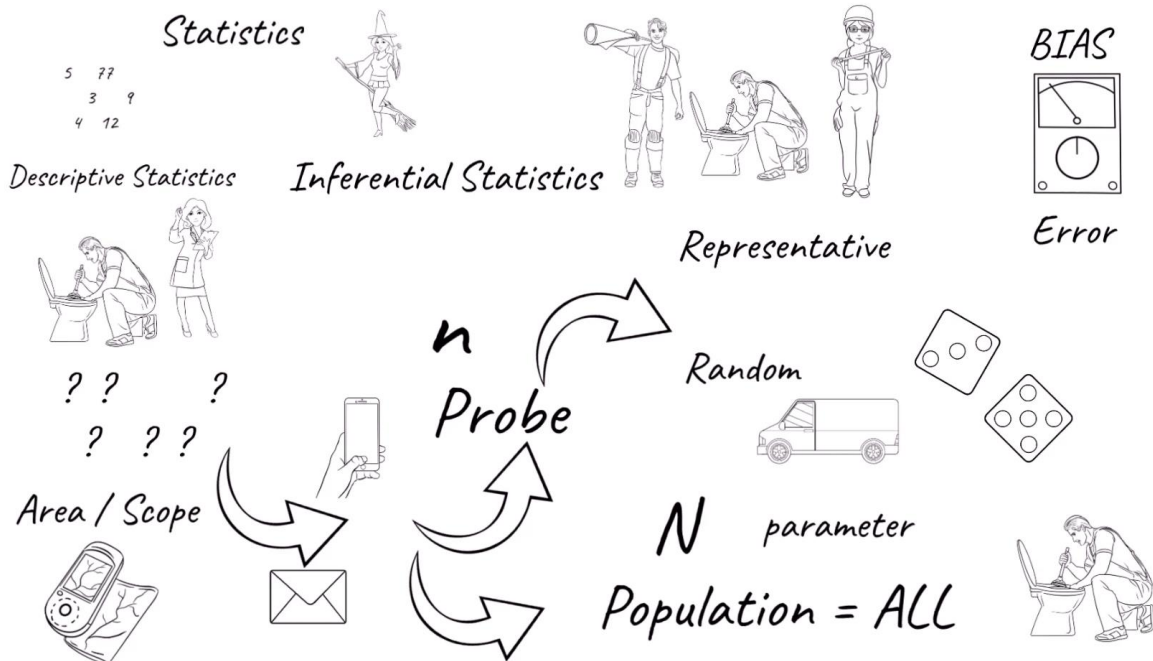
Aby odtworzyć takie środowisko np. na innym komputerze użyj

```
pip -r requirements.txt
```

# Statystyka, populacja, próba

## Notatka

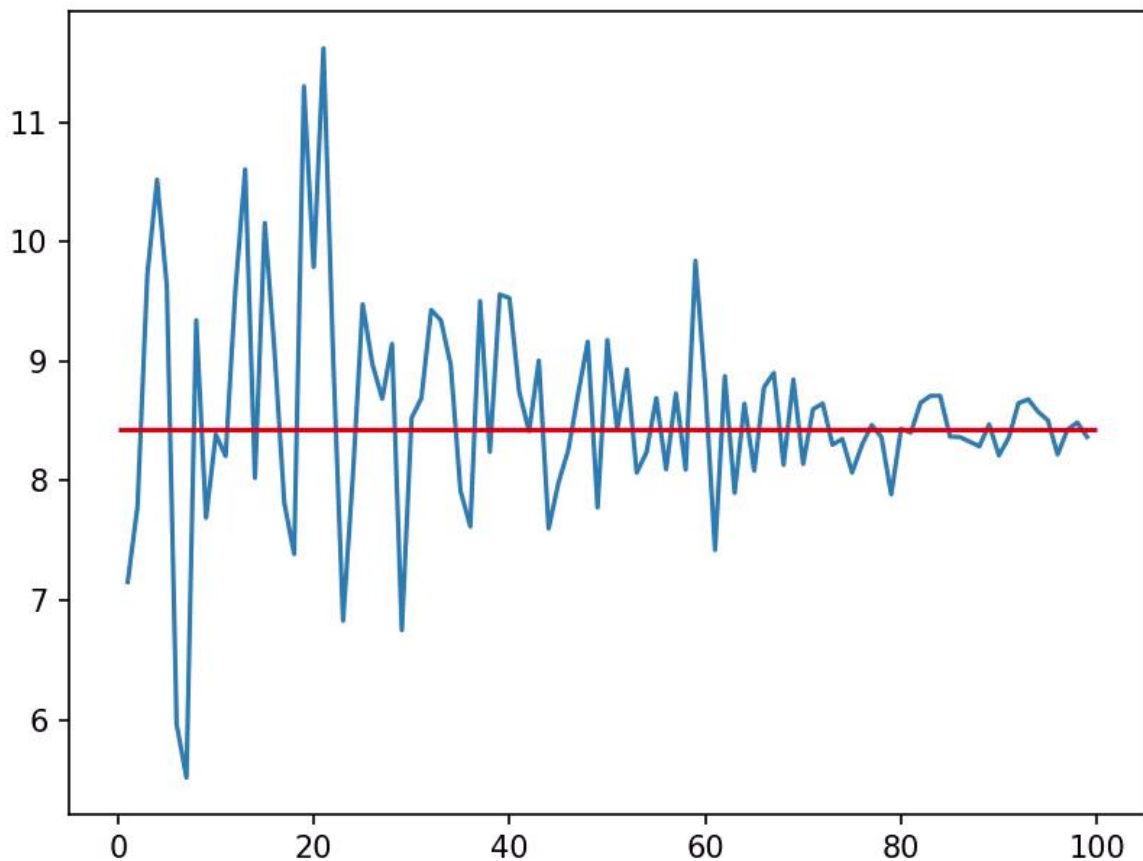
- Statystyka opisuje zbiory przy pomocy wartości numerycznych (descriptive statistics) oraz wnioskowaniem statystycznym (inferential statistics)
- Problemy z analizowaniem zbiorów:
  - Trudny do ustalenia zakres
  - Problemy z dotarciem do wszystkich badanych osób/obiektów
  - Koszt i czas z tym związany
- **Populacja** – zbiór wszystkich badanych osób/obiektów,
- **Próba** – wylosowany podzbiór populacji
- **Parametr** – wartość liczbową opisującą cechę populacji,
- **Statystyka** – wartość liczbową opisującą charakterystykę próby.
- Dobra próba spełnia warunki:
  - Jest wyznaczana losowo – zwykle można to osiągnąć poprzez całkowicie przypadkowy wybór obiektów
  - Jest reprezentatywna – zwykle osiągamy to przez budowanie stosunkowo dużej próby
- Jeśli populacja jest mało liczna, to uzyskane wyniki mogą nadal nie być reprezentatywne
- Zły wybór próby może powodować obciążenie badań błędem zwanym BIAS.
- Kiedy we wzorach jest mowa o populacji, to stosujemy symbol  $N$ , a kiedy o próbie to symbol  $n$





- 1) Zapoznaj się z artykułem dostępnym pod adresem:  
<https://www.cio.com/article/190888/5-famous-analytics-and-ai-disasters.html>  
Znajdziesz w nim opis wpadek kilku dużych firm, które zbudowały modele data science obciążone BIAS, względnie utraciły dane wskutek przyjęcia błędnych założeń. W efekcie do pracy w Amazon byli przyjmowani tylko mężczyźni, chat bot Microsoftu popisał się rasizmem na Tweeterze, a służba zdrowia zaniedbywała profilaktykę osób innych ras. Nie musisz tego artykułu czytać słowo po słowie, ale zobacz, że dla profesjonalnych firm, problem unikania BIAS jest bardzo istotny. Przeglądając artykuł odpowiedz sobie na pytanie, czy w ogóle byłeś/byłaś świadomy/świadoma takich uprzedzeń?
- 2) Zapoznaj się z artykułem przedstawiającym kilka wniosków dotyczących analizy efektywności wypożyczalni samochodów dostępnej pod adresem: <https://www.pgs-soft.com/pl/blog/data-science-w-praktyce-analiza-aktywnosci-wypożyczalni-samochodow-elektrycznych-vozilla/>  
Przeglądając artykuł zauważ, jakie wnioski udaje się wysnuwać z danych.

- Pandas – moduł do analizy danych
  - `read_parquet(...)` – wczytuje plik w formacie parquet
  - DataFrame:
    - ♦ `count()` – liczba znanych wartości w kolumnach
    - ♦ `head()` – nagłówek data frame
    - ♦ `info()` – kolumny i ich typy
    - ♦ `describe()` – podstawowe parametry każdej kolumny
    - ♦ `sample(...)` – losowanie próby
    - ♦ `df_probe["trip_distance"].mean()` – średnia dla kolumny
- Matplotlib – moduł do wizualizacji danych
  - `plt.plot(means)` – ilustracja liniowa dla wartości z listy means
  - `plt.hlines(...)` – pozioma linia na wykresie
- **Populacja** – cały zakres danych do analizy, oznaczany przez N
- **Próba** – podzbiór populacji, oznaczany przez n
- Próba jest łatwiejsza do zebrania niż populacja, ale jest obarczona błędem. Aby próba była dobra powinna być losowa i reprezentatywna



W lekcji skorzystaliśmy z pandasowej funkcji `sample()`. A gdyby tak, taką funkcjonalność zbudować samodzielnie? Napiszmy taki program! Ze zbioru taxi chcemy wylosować próbę zawierającą `probe_size` wierszy. Masz prawo nie znać PANDAS, dlatego program jest już w dużej mierze napisany. Zapoznaj się najpierw z dokumentacją kilku funkcji, a następnie uzupełnij kod. Jeśli zechcesz, w drugim podejściu spróbuj napisać podobny kod zupełnie samodzielnie. Wszelkie własne modyfikacje mile widziane!

- 1) Zapoznaj się z opisem funkcji
  - a) `random.sample` – losowanie liczb z określonego zakresu
  - b) `pd.DataFrame` – tworzenie nowego, pustego obiektu `DataFrame`
  - c) `DataFrame.loc[i]` function – wybranie z `DataFrame` wskazanych wierszy
- 1) Uzupełnij luki w kodzie:

```
# Import module for data analysis in Python
import ..... as pd
import random

# Import data from file yellow_tripdata_2021-05.parquet
taxi = pd.....('./datasets/yellow_tripdata_2021-05.parquet',
               engine='auto', columns=None, storage_options=None, use_nullable_dtypes=False)

probe_size=100

# Generate random numbers 0 - len of taxi. The result should have probe_size numbers
sample_indexes = random.sample(range(len(taxi.index)), probe_size)

# Create copy of taxi data frame
taxi_probe = pd.DataFrame(data=None, columns=taxi.columns)
# Copy data frame rows one by one
for i in range(probe_size):
    taxi_probe.loc[i] = taxi.loc[sample_indexes[i]]

# Print a calculated mean for the column trip_distance
print(taxi_probe['.....']......())
```

### Propozycja rozwiązania

```
import pandas as pd
import random

taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet',
                      engine='auto', columns=None, storage_options=None, use_nullable_dtypes=False)

probe_size=100

sample_indexes = random.sample(range(len(taxi.index)), probe_size)

taxi_probe = pd.DataFrame(data=None, columns=taxi.columns)
for i in range(probe_size):
    taxi_probe.loc[i] = taxi.loc[sample_indexes[i]]

print(taxi_probe['trip_distance'].mean())
```

# Typy danych

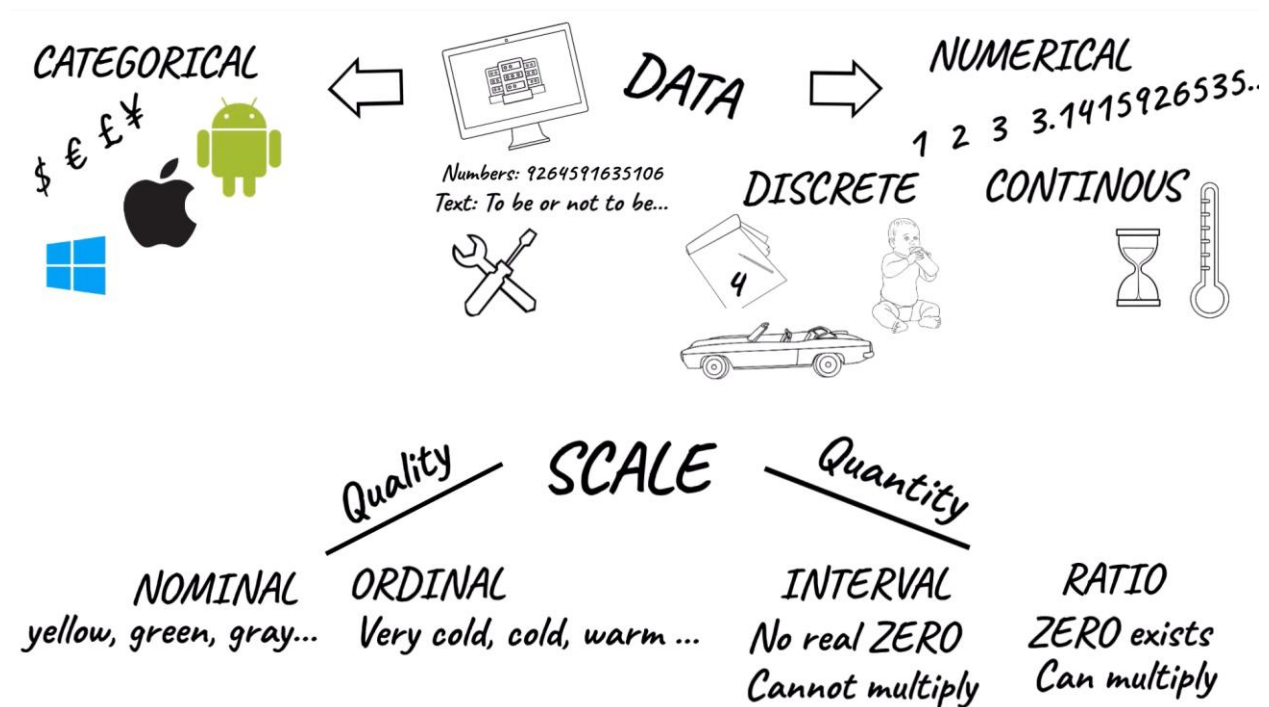
## Notatka

Typy danych można podzielić na

- **Kategoryczne** (categorical): opisują kategorię np. waluta, kolor, marka samochodu
- **Numeryczne** – opisywane liczbami
  - **Dyskretne** – wskazują na konkretną liczbę: liczba dzieci, liczba lat, ilość pieniędzy
  - **Ciągłe** (continuous) – trudne do zmierzenia, mierzone z różną dokładnością: temperatura, czas (jako ciągłe continuum), waga mierzona z „nieskończoną dokładnością”

Wartościom można też przypisywać wartości na skalach:

- **Jakościowe** (quality)
  - **Porządkowe** (ordinal) – jakość jest opisana poprzez przypisanie do nienumerycznej kategorii, ale tę kategorię można w sensowny sposób uporządkować, np. nigdy, rzadko, często, zawsze
  - **Nominalne** – jakość jest opisana przez przypisanie do nienumerycznej kategorii, ale te kategorie nie mają naturalnej kolejności, np. krajowy/zagraniczny, kobieta/mężczyzna
- **Ilościowe** (quantity)
  - **Interwałowe** – nie mają „rzeczywistego zera” (w znaczeniu nic), miary można dodawać i odejmować, ale nie można mnożyć, np. temperatura w stopniach Celsjusza
  - **Ilorazowe** (ratio) – mają rzeczywiste zero (w znaczeniu nic), można dodawać, mnożyć odejmować, np. pieniądze, punkty w grze



## Lab

- 2) Do jakich typów danych zaliczysz:
- a) Języki obce, którymi mówisz, np. polski, angielski, hiszpański
  - b) Ukończony poziom edukacji
  - c) Masa cukierków na wagę
  - d) Liczba owoców Kiwi sprzedawanych na sztuki
  - e) Czas trwania zajęć w minutach

## Propozycja rozwiązania

Języki – kategoriyczny / nominalny np. Włoski, Angielski

Edukacja – kategoriyczny / porządkowy, np. Szkoła podstawowa, liceum, studia licencjackie

Cukierki – numeryczny / ilorazowy (ratio) / jeśli ważony z dokładnością np. do gramów, to dyskretny, jeśli mierzony z „nieskończoną dokładnością” to ciągły

Kiwi – numeryczny / dyskretny / ilorazowy (ratio)

Czas zajęć – numeryczny / dyskretny / ilorazowy (ratio)

## Wykresy dla typów danych kategorycznych

### Notatka

- Dbaj o pobieranie tylko tych danych, które są potrzebne
  - Wymieniaj kolumny podczas wczytywania danych np. w `pd.read_parquet`
  - Jeśli dane są kategoryczne to stosuj `pandas.api.types.CategoricalDtype`

```
cat_weekdays = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
cat_type = CategoricalDtype(categories=cat_weekdays, ordered=True)
taxi['dayofweek'] = taxi['tpep_pickup_datetime'].dt.day_name().astype(cat_type)
```

- Do budowy frequency table użyj grupowania. W razie potrzeby zmień nazwy wierszy, kolumn, definiuj indeks:

```
taxi = taxi.groupby(['dayofweek'], as_index=False).count()
taxi.rename(columns={'tpep_pickup_datetime': 'count'}, inplace=True)
taxi.set_index('dayofweek', inplace=True)
```

- Wykresy dobre dla danych kategorycznych:
  - Wykres słupkowy – bar chart

```
taxi.plot.bar(y='count', legend=False)
```

- Wykres kołowy – pie

```
taxi.plot.pie(y='count', legend=False, counterclock=False, autopct='%1.1f%%')
```

- Wykres pareto

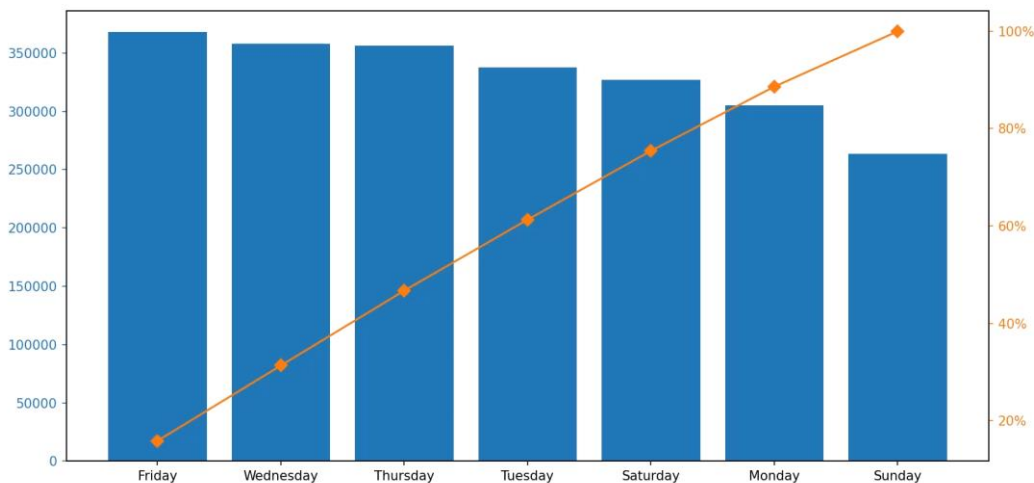
Kilka poleceń z tej lekcji:

- `sum()`, `csum()` – suma i suma narastająca

```
taxi["cumulative_percent"] = taxi["count"].cumsum()/taxi["count"].sum()*100
```

- sortowanie data frame wg wybranej kolumny:

```
taxi.sort_values(by='count', ascending=False, inplace=True)
```



Klienci taksówek mogą płacić za przejazd

- kartą kredytową,
- gotówką,
- mogą mieć przejazd opłacony z góry,
- mogą nie chcieć zapłacić, bo czują się oszukani przez „złotówę”
- albo nie odnotowano, jak zapłacili.

Zakładając, że wartości w kolumnie „payment\_type” oznaczają:

0: 'Credit card', 1: 'Cash', 2: 'No charge', 3: 'Dispute', 4: 'Unknown'

spróbuj zilustrować jakim powodzeniem cieszyły się te formy płatności w przejazdach w maju 2021. Skorzystaj z kodu zaprezentowanego na lekcji, podmieniając tylko kilka fragmentów. Oto zmiany:

- 1) Pracujemy na kolumnie payment\_type
- 2) Ta kolumna już ma wartość numeryczną i nie ma naturalnej kolejności, dlatego rezygnujemy z zabawy w przekształcanie danych do typu kategoriycznego
- 3) Po grupowaniu danych zmień nazwy wierszy, tak by odpowiadały wymienionym wyżej nazwom. Zrobisz to takim oto poleceniem:

```
taxi.rename(index={0: 'Credit card', 1: 'Cash', 2: 'No charge',
                  3: 'Dispute', 4: 'Unknown'}, inplace=True)
```

- 4) Ponieważ taksówkarze to w gruncie rzeczy dobrzy ludzie i z nikim nigdy się nie kłócą, to połącz wartości z wiersza Dispute i Unknown w nowy wiersz Other. Potem usuń wiersz Dispute i Unknown. Zrobisz to poleceniami:

```
taxi.loc['Other'] = taxi.loc['Dispute'] + taxi.loc['Unknown']
taxi.drop(['Dispute', 'Unknown'], inplace=True)
```

- 5) Być może zauważyłeś drobne oszustwo w przykładzie z lekcji – skala na osi po prawej stronie wykresu pareto nie zaczynała się od zera. Aby to poprawić dodaj poniższą instrukcję po instrukcji plot rysującej skalę z prawej strony:

```
ax2.set_ylim(bottom=0)
```

I jak, wiesz już w jaki sposób płaci się najczęściej za przejazdy taksówką w New York?

## Propozycja rozwiązania

```
import pandas as pd
from pandas.api.types import CategoricalDtype
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.ticker import PercentFormatter

taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet',
                      engine='auto', columns=['tpep_pickup_datetime', 'payment_type'],
                      storage_options=None, use_nullable_dtypes=False)

taxi = taxi.query("tpep_pickup_datetime >= '2021-05-19' and tpep_pickup_datetime <
'2021-05-20'")

taxi = taxi.groupby(['payment_type'], as_index=True).count()
print(taxi)

taxi.rename(columns={'tpep_pickup_datetime': 'count'}, inplace=True)
print(taxi)

taxi.rename(index={0: 'Credit card', 1: 'Cash', 2: 'No charge', 3: 'Dispute', 4:
'Unknown'}, inplace=True)
taxi.loc['Other'] = taxi.loc['Dispute'] + taxi.loc['Unknown']
taxi.drop(['Dispute', 'Unknown'], inplace=True)
print(taxi)

taxi.plot.bar(y='count', legend=False)
plt.show()

taxi.plot.pie(y='count', legend=False, counterclock=False, autopct='%1.1f%%')
plt.show()

# PARETO by weekday

taxi.sort_values(by='count', ascending=False, inplace=True)
taxi["cumulative_percent"] = taxi["count"].cumsum()/taxi["count"].sum()*100
print(taxi)

fig, ax = plt.subplots()
ax.bar(taxi.index, taxi["count"], color="C0")
ax2 = ax.twinx()
ax2.plot(taxi.index, taxi["cumulative_percent"], color="C1", marker="D", ms=7)
ax2.set_ylim(bottom=0)
ax2.yaxis.set_major_formatter(PercentFormatter())

ax.tick_params(axis="y", colors="C0")
ax2.tick_params(axis="y", colors="C1")
plt.show()
```



## Wykresy dla typów numerycznych - histogram

### Notatka

Popularny wykres dla danych numerycznych to histogram. Powstaje przez podział wartości na pewne przedziały, a następnie zliczenie liczby obserwacji, które ze względu na wartość można przypisać do jednego z tych przedziałów. Zazwyczaj staramy się, aby te przedziały były równe. W takim przypadku najważniejsze jest ustalenie, ile takich przedziałów należy stworzyć. Liczba przedziałów zależy od danych, jakie mamy do zaprezentowania, ale można zaczynać od małych wartości np. 5 lub 6. Pierwszy przedział powinien zaczynać się od minimalnej wartości do zaprezentowania, a ostatni powinien się kończyć na wartości maksymalnej. W Pandas można je wyznaczyć korzystając z funkcji min i max:

```
taxi["trip_distance"].min()
taxi["trip_distance"].max()
```

Długość każdego przedziału można wyznaczyć dzieląc różnicę między max i min przez liczbę przedziałów:

```
interval_width = (taxi['trip_distance'].max() - taxi["trip_distance"].min())
                  /number_of_bins
```

Liczbę wartości przypadających na każdy przedział oraz punkty krańcowe przedziałów można wygenerować funkcją np. histogram:

```
abs_frequency, intervals = np.histogram(taxi["trip_distance"], bins = 50)
```

Wyznaczenie przedziałów może też być zrealizowane poleceniem interval\_range:

```
interval_range = pd.interval_range(start=0, freq=1, end=50, closed='right')
```

Mając taki zbiór przedziałów można następnie utworzyć w analizowanym zbiorze nową kolumnę zawierającą informacje o przedziale, do którego zakwalifikował się rekord:

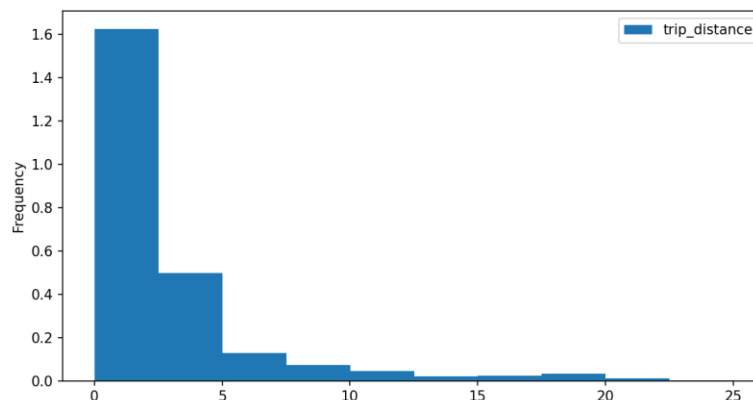
```
taxi['bin'] = pd.cut(taxi['trip_distance'], bins=interval_range)
```

Zliczenie, ile rekordów przypadło na każdy przedział można następnie wykonać poleceniem crosstab:

```
freq_table = pd.crosstab(index=taxi["bin"], columns="trip_distance")
```

W najprostszej postaci rysowanie histogramu wykona polecenie

```
taxi.plot.hist(column="trip_distance", bins=50)
plt.show()
```



W podobny sposób jak na lekcji pobaw się danymi ze zbioru taxi ilustrując wysokość napiwku zostawianego przez pasażerów kierowcom taksówki. Przy pierwszym podejściu posiłkuj się rozwiązaniem lub materiałem lekcji, przy kolejnych próbach staraj się co nieco zrobić samodzielnie.

- 1) Dane znajdziesz w kolumnie trip\_amount
- 2) Odfiltruj tylko próby w których trip\_amount jest większe od 0 i mniejsze równe 20
- 3) Wyznacz wartość minimalną, maksymalną dla tip\_amount, zgadnij liczbę przedziałów histogramu (możesz zacząć np. od wartości 10)
- 4) Wyznacz „dobrą długość przedziału”
- 5) Wyznaczanie przedziałów – podejście nr 1:
  - a) Korzystając z funkcji np.histogram wyznacz frequency table i punkty krańcowe przedziałów
- 6) Wyznaczanie przedziałów – podejście nr 2
  - a) Korzystając z funkcji pd.interval\_range wyznacz zbiór przedziałów
  - b) Korzystając z funkcji pd.cut przypisz do każdej próby pasujący przedział i zapisz go w nowej kolumnie bin
  - c) Korzystając z funkcji pd.crosstab wylicz liczbę prób przypadających na każdy przedział
- 7) Narysuj histogram

## Propozycja rozwiązania

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet',
                      engine='auto', columns=['tip_amount'], storage_options=None,
                      use_nullable_dtypes=False)

taxi.query('tip_amount > 0 and tip_amount <= 20', inplace=True)

min_tip = taxi['tip_amount'].min()
max_tip = taxi['tip_amount'].max()
number_of_bins = 10

# Manual calculation of good interval width
interval_width = (max_tip - min_tip) / number_of_bins
print(f'Good interval width for {number_of_bins} bins is {interval_width}')

# Automatic calculation of intervals
abs_frequency, intervals = np.histogram(taxi["tip_amount"], bins = number_of_bins)
print('Automatically calculated intervals', abs_frequency, intervals,
      len(abs_frequency), len(intervals))

# create borders of intervals
interval_range = pd.interval_range(start=0, freq=interval_width, end=max_tip,
                                   closed='right')
# assign each row to a bin
taxi['bin'] = pd.cut(taxi['tip_amount'], bins=interval_range)
print(taxi.head())
# calculate counts for each bin
freq_table = pd.crosstab(index=taxi["bin"], columns="tip_amount")
print(freq_table)

# # illustrate as bar chart
# freq_table.plot.bar(y='tip_amount', legend=False)
# plt.show()

# illustrate as histogram
taxi.plot.hist(column="tip_amount", bins=number_of_bins)
# change labels on x-axis
plt.xticks([ i*2 for i in range(number_of_bins)], [ i*2 for i in
range(number_of_bins)])
plt.show()
```

## Wykres dla dwóch kategorii – side by side bar i scatter plot

### Notatka

Wykres słupkowy grupowany (side by side bar) nadaje się do ilustrowania dwóch kategorii i wartości numerycznej cechującej te kategorie. Budowanie takiego wykresu rozpoczyna się od przygotowania tabeli przestawnej, gdzie w wierszach znajduje się jedna kategoria, w kolumnach druga, a na przecięciu wiersza i kolumny – wartość numeryczna.

```
taxi_s.plot.bar()
```

Wykres punktowy (scatter plot) nadaje się do ilustrowania zależności między dwiema wartościami numerycznymi.

```
taxi.plot.scatter(x='fare_amount', y='tip_amount')
```

Kiedy w Pandas chcesz zastosować funkcję, do wybranej kolumny użyj metody apply (trip\_comment przekazywane jako argument w metodzie apply to nazwa funkcji):

```
taxi['trip_comment'] = taxi['trip_distance'].apply(trip_comment)
```

Jeśli funkcja ma mieć przekazany jako argument typ datetime, to można „wyłuskać” z kolumny wartość skonwertowaną do daty/czasu (time\_of\_day to nazwa funkcji):

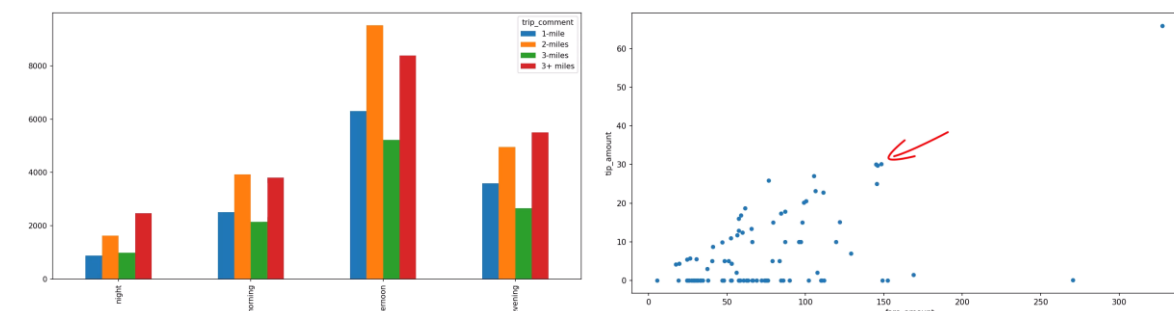
```
taxi['time_of_day'] = taxi['tpep_pickup_datetime'].dt.hour.apply(time_of_day)
```

Budowanie tabeli przestawnej można przeprowadzić korzystając z funkcji pd.crosstab:

```
taxi_s = pd.crosstab(taxi["time_of_day"], taxi['trip_comment'])
```

Jeśli chcesz zmienić kolejność kolumn i wierszy to możesz użyć dla wierszy polecenia reindex, a dla kolumn, przepisać kolumny w zdefiniowanej przez siebie kolejności:

```
taxi_s = taxi_s.reindex(['night', 'morning', 'afternoon', 'evening'])  
taxi_s = taxi_s[['1-mile', '2-miles', '3-miles', '3+ miles']]
```



## Lab

W tym laboratorium zastanowimy się, ile pieniędzy wydaje się na przejazdy taksówkami w zależności od pory dnia i długości przejazdu (dwie kategorie i jedna wartość) oraz zobaczymy jaka jest relacja między długością trasy i opłatą za przejazd.

- 1) Zaimportuj z pliku z przejazdami taksówek kolumny `trip_distance`, `tpcp_pickup_datetime`, `fare_amount`, `RatecodeID`.
- 2) Odfiltruj ze zbioru tylko dane dotyczące 9 maja 2021
- 3) Korzystając z poniższych funkcji utwórz kolumnę (serie danych) zawierającą opis pory dnia/nocy oraz opis długości dystansu przejazdu

```
def time_of_day(hour):  
    if hour < 6:  
        return 'night'  
    elif hour < 12:  
        return 'morning'  
    elif hour < 18:  
        return 'afternoon'  
    else:  
        return 'evening'
```

```
def trip_comment(trip_distance):  
    if trip_distance < 1:  
        return '1-mile'  
    elif trip_distance < 2:  
        return '2-miles'  
    elif trip_distance < 3:  
        return '3-miles'  
    else:  
        return '3+ miles'
```

- 4) Zbuduj tabelę przestawną zawierającą w wierszach porę dnia, a w kolumnach długość przejazdu. Wartościami ma być suma `fare_amount`.  
Wskazówka: Do argumentów funkcji `crosstab` dodaj: `values=taxi['fare_amount'], aggfunc="sum"`
- 5) Zadbaj o to aby wiersze i kolumny były ułożone w sensownej kolejności.
- 6) Narysuj wykres `side by side bar`
- 7) Odfiltruj ze zbioru danych tylko przejazdy, gdzie `fare_amount > 0` i `RatecodeID == 4`
- 8) Narysuj wykres punktowy ilustrujący zależność między długością trasy, a opłatą. Czy widzisz jakiś kierowców, którzy „naciągają” klientów? Czy widzisz jakieś przejazdy, w których kwota do zapłaty była zastanawiająco niska? Czy widzisz korelację między długością przejazdu a opłatą?

## Propozycja rozwiązania

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

def time_of_day(hour):
    if hour < 6:
        return 'night'
    elif hour < 12:
        return 'morning'
    elif hour < 18:
        return 'afternoon'
    else:
        return 'evening'

def trip_comment(trip_distance):
    if trip_distance < 1:
        return '1-mile'
    elif trip_distance < 2:
        return '2-miles'
    elif trip_distance < 3:
        return '3-miles'
    else:
        return '3+ miles'

taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet', engine='auto',
                      columns=['trip_distance', 'tpep_pickup_datetime', 'fare_amount', 'RatecodeID'],
                      storage_options=None, use_nullable_dtypes=False)

taxi.query("tpep_pickup_datetime>='2021-05-09' and tpep_pickup_datetime<'2021-05-10'",
          inplace=True)

taxi['time_of_day'] = taxi['tpep_pickup_datetime'].dt.hour.apply(time_of_day)
taxi['trip_comment'] = taxi['trip_distance'].apply(trip_comment)

taxi_s = pd.crosstab(taxi["time_of_day"], taxi['trip_comment'],
                    values=taxi['fare_amount'], aggfunc="sum")
print('--- initial cross table ---')
print(taxi_s)

taxi_s = taxi_s.reindex(['night', 'morning', 'afternoon', 'evening'])
taxi_s = taxi_s[['1-mile', '2-miles', '3-miles', '3+ miles']]
print('--- after setting up order ---')
print(taxi_s)

taxi_s.plot.bar()
plt.show()

taxi.query("fare_amount>0 and RatecodeID==4", inplace=True)

print(taxi.head())
print(taxi.count())

taxi.plot.scatter(x='trip_distance', y='fare_amount')
plt.show()
print(task)
```

## Średnia, mediana, dominanta, skośność

### Notatka

Średnia, mediana, dominanta i współczynnik skośności danych pozwalają na opisanie „środka danych”. Wiele wartości wykorzystywanych w statystyce jest zwracanych przez funkcję describe:

```
taxi.describe()
```

Średnią wyznaczysz dzieląc sumę wartości próby przez liczbę elementów w tej próbie:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \dots + x_n}{n}$$

Średnią można też wyznaczyć dla populacji i wtedy oznaczamy ją symbolem  $\mu$

W Pandas średnią można wyznaczyć przy pomocy funkcji mean:

```
taxi.mean()
```

Mediana to wartość środkowa. Wartości z próby należałoby uporządkować w kolejności rosnącej, a następnie wybrać element znajdujący się w środku uporządkowanego zbioru. Jeśli liczba obserwacji jest parzysta, to wyznacza się średnią arytmetyczną dwóch najbardziej środkowych elementów.

Mediana jest miarą odporniejszą (mniej wrażliwą na wartości odstające) niż średnia.

W Pandas medianę można wyznaczyć przy pomocy funkcji median:

```
taxi.median()
```

Dominanta (mode) to wartość, która w zbiorze pojawia się najczęściej. Jeśli kilka wartości miałoby się w zbiorze powtarzać taką samą liczbę razy, to dominanta będzie zbiorem liczb.

W Pandas dominantę można wyznaczyć przy pomocy funkcji mode:

```
taxi.mode()
```

Na histogram można nałożyć linie odpowiadające za wyznaczane wartości środkowe:

```
taxi.plot.hist(column="trip_distance", bins=50)
plt.axvline(taxi_mean, color='pink', linestyle='dotted')
plt.axvline(taxi_median, color='red', linestyle='dotted')
plt.axvline(taxi_mode, color='black', linestyle='dotted')
plt.show()
```

Współczynnik skośności pozwala opisać histogram – z której strony histogramu będzie „górką”, a z której strony „ogon”.

- Jeśli ogon jest z prawej, to mówimy o skośności prawej. Współczynnik jest wtedy dodatni.
- Jeśli ogon jest z lewej, to mówimy o skośności lewej. Współczynnik jest wtedy ujemny
- Jeśli histogram jest raczej symetryczny, to skośność jest centralna i współczynnik też bliski zeru.

W Pandas współczynnik skośności można wyznaczyć przy pomocy funkcji skew:

```
taxi.skew()
```

## Lab

W tym labie, zastanowimy się jakie napiwki dostają taksówkarze w Nowym Jorku.

- 1) Zaimportuj dane z kolumny tip\_amount
- 2) Odfiltruj tylko te wiersze, gdzie tip\_amount jest >0 i <=20
- 3) Wyznacz wartość średnią, medianę, dominantę i współczynnik skośności.
- 4) W oparciu o współczynnik skośności, spróbuj sobie wyobrazić, jak może wyglądać histogram.
- 5) Narysuj histogram i nanieś na wykres pionowe linie odpowiadające za średnią, medianę i dominantę.
- 6) Jaki jest najczęstszy napiwek? Ile wynosi napiwek dla większości kierowców? Ile wynosi średni napiwek dla kierowców z uwzględnieniem szczęściarzy, którzy dostawali nieco lepsze napiwki?

## Propozycja rozwiązania

```
taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet',
                      engine='auto', columns=['tip_amount'], storage_options=None,
                      use_nullable_dtypes=False)

taxi.query('tip_amount > 0 and tip_amount <= 20', inplace=True)

print(taxi.describe())

taxi_mean = taxi.mean().iloc[0]
taxi_median = taxi.median().iloc[0] # values[0]
taxi_mode = taxi.mode().iloc[0,0]
taxi_skew = taxi.skew()
print('Mean:', taxi_mean)
print('Median:', taxi_median)
print('Mode:', taxi_mode)
print('Skew:', taxi_skew)

taxi.plot.hist(column="tip_amount", bins=50)
plt.axvline(taxi_mean, color='pink', linestyle='dotted')
plt.axvline(taxi_median, color='red', linestyle='dotted')
plt.axvline(taxi_mode, color='black', linestyle='dotted')
plt.show() print(q.get())
```



## Średnia ważona i ucinana

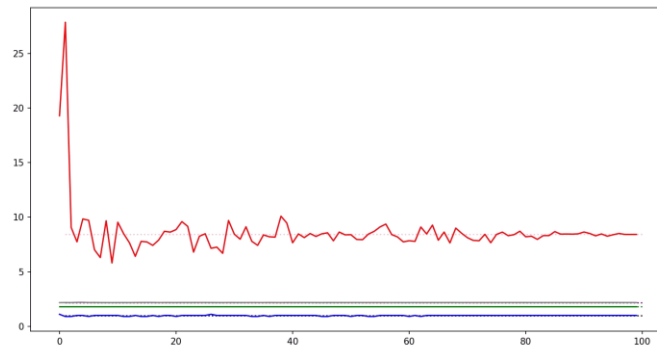
### Notatka

Celem średniej ucinanej jest zwiększenie odporności średniej. Miarę określamy jako odporną, jeśli nie jest ona zaburzana przez wartości odstające. Odporność jest zwiększana poprzez odrzucenie kilku procent wartości najmniejszych i największych. Z takiego zbioru wyznacza się następnie wartość średnią.

$$\bar{x} = \frac{\sum_{i=k}^{n-k} x_i}{n - 2k}$$

Średnią ucinaną można wyznaczyć funkcją `stats.trim_mean` z modułu `scipy`:

```
stats.trim_mean(taxi["trip_distance"], 0.1)
```



Od góry: średnia, średnia ważona, mediana i dominanta

Wyznaczając średnią ważoną należy sumować wartości mnożone przez wektor wag i dzielić uzyskaną sumę przez sumę wag. Pozwala to na przypisanie obserwacjom większej lub mniejszej ważności.

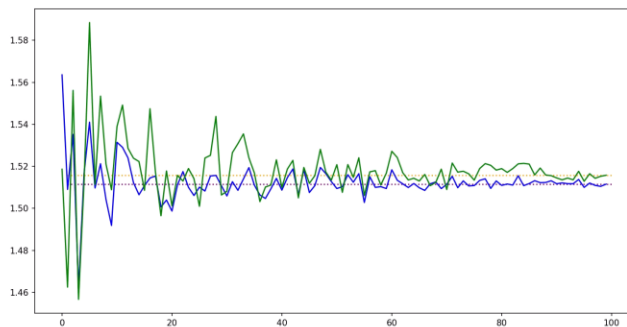
$$\bar{x} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}$$

Przyjmując, że `trip_distance` to wagi, to średnią ważoną dla `passenger_count` wyznaczysz wzorem:

```
sum(taxi["trip_distance"] * taxi["passenger_count"]) / taxi["trip_distance"].sum()
```

Istnieją też gotowe funkcje, które można wykorzystać, np. w module `NumPy`:

```
np.average(a=taxi['passenger_count'], weights=taxi['trip_distance'])
```



Zielony: średnia ważona liczby pasażerów z wagą długość przejazdu; niebieski: średnia liczby pasażerów

## Lab

W tym lab rozbudujemy rozwiązanie z poprzedniego LAB-a. Dodaj do rozwiązania obliczenia średniej ucinanej i ważonej. Porównaj ich wartości z wyznaczonymi do tej pory.

- 1) Zainstaluj scipy i zaimportuj z niego stats.
- 2) Dodaj polecenie wyznaczające średnią ucinaną obcinając 5% danych.
- 3) Do wykresu histogramu dodaj pionową linię ilustrującą średnią ucinaną.
- 4) Zauważ, że średnia ucinana znajduje się między średnią, a medianą. Możesz przeprowadzić więcej eksperymentów, ucinając ze zbioru coraz więcej obserwacji. Linia odpowiedzialna za średnią ucinaną powinna zbliżać się do linii ilustrującej medianę.
- 5) Do polecenia importującego dane dodaj kolumnę passenger\_count. W wyrażeniu filtrującym dodaj warunek, że passenger\_count ma być większe od 0.
- 6) Wyznacz średnią ważoną dla zmiennej tip\_amount ważoną przez passenger\_count
- 7) Do wykresu histogramu dodaj pionową linię ilustrującą średnią ważoną.
- 8) Linia prawdopodobnie będzie bardzo blisko średniej arytmetycznej, co oznacza, że dodanie wagi niewiele zmieniło. Może to oznaczać, że liczba pasażerów taksówki nie wpływa na wysokość napiwku, albo że nowojorczyści podróżują w większości w pojedynkę.

## Propozycja rozwiązania

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy import stats

taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet',
                      engine='auto', columns=['tip_amount', 'passenger_count'], storage_options=None,
                      use_nullable_dtypes=False)

taxi.query('tip_amount > 0 and tip_amount <= 20 and passenger_count > 0',
          inplace=True)

print(taxi.describe())

taxi_mean = taxi.mean().iloc[0]
taxi_median = taxi.median().iloc[0] # values[0]
taxi_mode = taxi.mode().iloc[0,0]
taxi_skew = taxi.skew()
taxi_trim_mean = stats.trim_mean(taxi["tip_amount"], 0.05)
taxi_weight_mean = np.average(taxi["tip_amount"], weights=taxi["passenger_count"])
print('Mean:', taxi_mean)
print('Median:', taxi_median)
print('Mode:', taxi_mode)
print('Skew:', taxi_skew)
print('Trim weight:', taxi_trim_mean)
print('Weight mean', taxi_weight_mean)

taxi.plot.hist(column="tip_amount", bins=50)
plt.axvline(taxi_mean, color='pink', linestyle='dotted')
plt.axvline(taxi_median, color='red', linestyle='dotted')
plt.axvline(taxi_mode, color='black', linestyle='dotted')
plt.axvline(taxi_trim_mean, color='yellow', linestyle='dotted')
plt.axvline(taxi_weight_mean, color='blue', linestyle='solid')
plt.show()
```

## Wariancja

### Notatka

Wariancja to miara rozproszenia w zbiorze. Jeśli zbiór jest skupiony w okolicach wartości średniej, to odległości poszczególnych wartości od średniej są małe. Jeśli zbiór jest rozproszony (punkty leżą daleko od punktu wyznaczonego przez średnią), to odległości tych punktów od wartości średniej są duże.

Wzór na wariancję populacji i wariancję próby jest określony wzorami:

$$\sigma^2 = \frac{\sum_{i=1}^N (x_i - \mu)^2}{N} \qquad s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$$

W liczniku od wartości obserwacji odejmuje się wartość średniej. Operacja podniesienia do kwadratu niweluje wartości ujemne. Ta wartość odpowiada za obliczenie odległości punktu od wartości średniej. Sumę tych wartości dzieli się przez N dla populacji lub n-1 dla próby. Uzyskana wartość ma jednostkę podniesioną do kwadratu, np. cm<sup>2</sup>, kg<sup>2</sup>, zł<sup>2</sup>.

Wzór rzeczywiście opisuje rozrzut danych w zbiorze i robi to możliwie dokładnie.

W module statistics dostępna jest funkcja pvariance wyznaczająca wariancję populacji. Z kolei funkcja variance wyznacza wartość wariancji dla próby:

```
import random as r
import statistics as stat

my_list = [r.randint(0,100) for i in range(200)]
print(stat.pvariance(my_list))
```

### Lab

- 1) Wygeneruj listę 200 liczb losowych z przedziału 0-100. Zapisz wynik w zmiennej my\_list.
- 2) Utwórz kolejne listy, które zawierają wartości wyliczone w oparciu o my\_list w następujący sposób:
  - a) Lista my\_list\_1 – każdy element jest wyliczany poprzez dodanie liczby 2 do wartości z listy my\_list
  - b) Lista my\_list\_2 – każdy element jest wyliczany poprzez pomnożenie liczby 2 z wartością z listy my\_list
  - c) Lista my\_list\_3 – każdy element listy jest wyliczany przez pomnożenie liczby 2 z wartością z listy my\_list i dodanie liczby 2
  - d) Lista my\_list\_4 – każdy element listy jest wyliczany jako pierwiastek kwadratowy z liczby z my\_list
- 3) Teraz na chwilę się zatrzymaj i spróbuj powiedzieć, jakie będą relacje między wariancjami wyznaczonymi dla wszystkich zdefiniowanych list. Oczywiście nie zgaduj liczby, tylko pomyśl, które wartości będą sobie równe, które większe, a które mniejsze.
- 4) Wyznacz wariancję populacji dla każdej z list i zobacz, czy jest to zgodne z pomysłami jakie powstały w poprzednim punkcie.
- 5) Utwórz próby losowe po 20 elementów z każdej z list i nazwij je my\_sample, my\_sample\_1 itp.

- 6) Teraz zatrzymaj się na chwilę i spróbuj powiedzieć, jakie będą relacje między wariancjami wyznaczanymi dla kolejnych prób. Możesz próbować estymować nie tylko relacje, ale nawet przybliżone wartości. Jeśli uda Ci się trafić, koniecznie obstaw dzisiaj totolotka.
- 7) Oblicz wariancję prób i sprawdź czy masz dobrą intuicję.
- 8) Zachowaj skrypt i wyniki do kolejnego LAB-a

### Propozycja rozwiązania

```
import random as r
import statistics as stat
import math

my_list = [r.randint(0,100) for i in range(200)]

my_list_1 = [i + 2 for i in my_list]
my_list_2 = [i * 2 for i in my_list]
my_list_3 = [i * 2 + 2 for i in my_list]
my_list_4 = [math.sqrt(i) for i in my_list]

print('Population variance - random numbers', stat.pvariance(my_list))
print('Population variance - plus 2', stat.pvariance(my_list_1))
print('Population variance - multi 2', stat.pvariance(my_list_2))
print('Population variance - multi 2 plus 2', stat.pvariance(my_list_3))
print('Population variance - root square', stat.pvariance(my_list_4))

my_sample = r.sample(my_list, 20)
my_sample_1 = r.sample(my_list_1, 20)
my_sample_2 = r.sample(my_list_2, 20)
my_sample_3 = r.sample(my_list_3, 20)
my_sample_4 = r.sample(my_list_4, 20)

print('Probe variance - random numbers', stat.pvariance(my_sample))
print('Probe variance - plus 2', stat.pvariance(my_sample_1))
print('Probe variance - multi 2', stat.pvariance(my_sample_2))
print('Probe variance - multi 2 plus 2', stat.pvariance(my_sample_3))
print('Probe variance - root square', stat.pvariance(my_sample_4))
```

## Odchylenie standardowe i współczynnik zmienności

### Notatka

Odchylenie standardowe (standard deviation) to pierwiastek z wariancji. Zaletą tej wartości jest taka, że jest ona wyrażona w tej samej jednostce, co wartości w populacji lub w próbie. Znaczenie i interpretacja wartości pozostają takie same, to znaczy określa ona o ile średnio rzecz ujmując są oddalone wartości zbioru od wartości średniej. Wzory dla populacji i próby wyglądają następująco:

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum_{i=1}^N (x_i - \mu)^2}{N}} \quad s = \sqrt{s^2} = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$

Funkcje pozwalające na wyliczenie wartości odchylenia standardowego znajdują się w module statistics i nazywają się pstdev (dla populacji) i stdev (dla próby).

Odchylenie standardowe jest specyficzne dla zbioru, trudno jest więc porównywać ze sobą dwa zbiory danych. Dlatego powstało pojęcie współczynnika zmienności (coefficient of variation). Wyznaczamy je dzieląc odchylenie standardowe przez średnią:

$$v = \frac{\sigma}{\mu}, \quad \sigma \neq 0 \quad V = \frac{s}{\bar{x}}, \quad \bar{x} \neq 0$$

W standardowych modułach Python nie ma gotowej funkcji do wyznaczeni współczynnika zmienności, ale łatwo go wyliczysz dzieląc odchylenie przez średnią.

Jeśli chcesz ręcznie dodawać elementy do DataFrame i potraktować go po prostu jako „wirtualny arkusz danych”, możesz to zrobić w następujący sposób:

```
import pandas as pd

df = pd.DataFrame(columns=["Function", "KG", "LBS"])
df.loc[len(df)] = ['Mean', mean(packages_kg), mean(packages_lbs)]
df.loc[len(df)] = ['Variance', variance(packages_kg), variance(packages_lbs)]
df.loc[len(df)] = ['St Dev', stdev(packages_kg), stdev(packages_lbs)]
df.loc[len(df)] = ['Coef var', stdev(packages_kg)/mean(packages_kg),
                  stdev(packages_lbs)/mean(packages_lbs)]

print(df)
```

Chociaż tutaj skupiliśmy się na średniej jako wartości środkowej, to można analizować te same problemy względem innej wybranej wartości, np. względem mediany.

### Lab

- 1) W tym labie możesz skorzystać z wyników z poprzedniego laba.
- 2) Masz już 5 list. Wyznacz dla każdej z nich wariancję, odchylenie standardowe i współczynnik zmienności.
- 3) Spróbuj umieścić te wartości w data frame. Wyniki dla kolejnych list umieść w wierszach, a wyliczane wartości w kolumnach.

## Propozycja rozwiązania

```
import random as r
import statistics as stat
import math
import pandas as pd

my_list = [r.randint(0,100) for i in range(200)]

my_list_1 = [i + 2 for i in my_list]
my_list_2 = [i * 2 for i in my_list]
my_list_3 = [i * 2 + 2 for i in my_list]
my_list_4 = [math.sqrt(i) for i in my_list]

df = pd.DataFrame(columns=["List", "Variance", "Stdev", "Coefficient of variation"])
df.loc[len(df)] = ['my_list', stat.pvariance(my_list),
                  stat.stdev(my_list), stat.stdev(my_list)/stat.mean(my_list)]
df.loc[len(df)] = ['my_list + 2', stat.pvariance(my_list_1),
                  stat.stdev(my_list_1), stat.stdev(my_list_1)/stat.mean(my_list_1)]
df.loc[len(df)] = ['my_list * 2', stat.pvariance(my_list_2), stat.stdev(my_list_2),
                  stat.stdev(my_list_2)/stat.mean(my_list_2)]
df.loc[len(df)] = ['my_list * 2 + 2', stat.pvariance(my_list_3),
                  stat.stdev(my_list_3), stat.stdev(my_list_3)/stat.mean(my_list_3)]
df.loc[len(df)] = ['sqrt my_list', stat.pvariance(my_list_4),
                  stat.stdev(my_list_4), stat.stdev(my_list_4)/stat.mean(my_list_4)]
print(df)
```

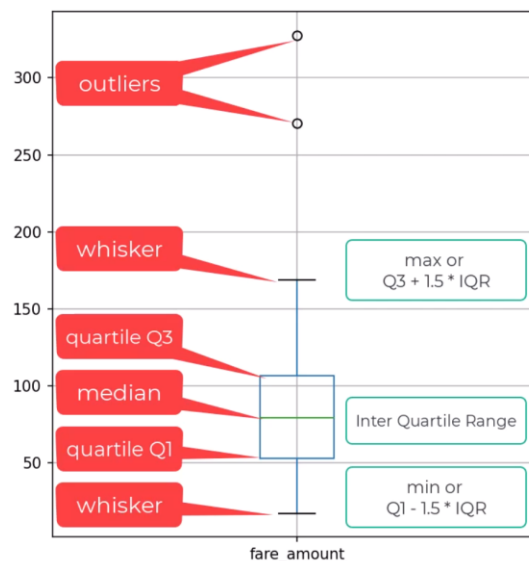
## Kwartyle, wykresy pudełkowe i usuwanie outlierów

### Notatka

Outlier (wartość odstająca) to wartość, która z jakiegoś powodu nie pasuje do zbioru. Może to wynikać z błędu pomiaru, błędu przetwarzania danych, ale czasami wartość odstająca jest poprawna i daje się logicznie uzasadnić. Statystyce często decydują się na odrzucanie wartości odstających.

Odnajdowanie outlierów rozpoczyna się często od rzutu oka na wykres. Wartości odstające są reprezentowane przez punkty znajdujące się dalej od „typowego skupiska” punktów. O ile łatwo jest zaobserwować outlierów, znacznie trudniej jest określić wzory, przy pomocy których można by takie obserwacje wykluczyć.

Gdy trzeba znaleźć wartości odstające w jednym wymiarze, można posłużyć się wykresem pudełkowym:



```
import pandas as pd
from matplotlib import pyplot as plt

taxi = pd.read_parquet('./datasets/yellow_tripdata_2021-05.parquet', engine='auto',
                      columns=['trip_distance', 'tpep_pickup_datetime', 'tip_amount',
                              'fare_amount', 'RatecodeID'], storage_options=None, use_nullable_dtypes=False)

taxi.query("tpep_pickup_datetime>='2021-05-09' and tpep_pickup_datetime<'2021-05-10'",
          inplace=True)
taxi.query("tip_amount<=100 and fare_amount>0 and RatecodeID==4 and tip_amount>0",
          inplace=True)

fig, axes = plt.subplots(nrows=1,ncols=2)
taxi.boxplot('fare_amount', ax = axes[0], meanline=True)
taxi.boxplot('tip_amount', ax = axes[1], meanline=True)
plt.show()
```

- **Kwartyl** – dzieli zbiór na 4 równe części podobnie, jak mediana dzieli zbiór na 2 części.
- **Percentyl** – dzieli zbiór na 100 równych części
- **Rozstęp ćwiartkowy** - (Inter Quartile Range – IRG) – różnica wartości wyznaczonych przez kwartyl 3 (Q3) a kwartyl 1 (Q1) :  $Q3 - Q1$ .

- **Wąs dolny** sięga do wartości najmniejszej lub  $Q1 - 1.5 * IRQ$
- **Wąs górny** sięga do wartości największej lub do  $Q3 + 1.5 * IRQ$

Q1 wyznacysz za pomocą metody `quantile` z argumentem 0.25. Podobnie z pozostałymi kwartylami

```
fare_amount_q1 = taxi['fare_amount'].quantile(0.25)
```

Przyjmując zasadę, że wartości “normalne” to te, które mieszczą się na wykresie pudełkowym między wąsami, należałoby jako outlierów potraktować te wartości, które:

- Są mniejsze niż  $Q1 - 1.5 * IQR$  lub
- Są większe niż  $Q3 + 1.5 * IQR$

Metoda `describe` wywoływana na rzecz obiektu data frame zwraca podstawowe miary dla zbioru danych, a wśród nich liczbę obserwacji, średnią, odchylenie standardowe, wartość minimalną i maksymalną oraz kwartyle: Q1, Q2 (mediana) i Q3.

```
print(taxi.describe())
```

## Lab

- 1) W tym lab zajmiemy się usunięciem outlierów w zbiorze Iris. Może go już znasz, bo jest on typowym przykładem pokazywanym w procesie uczenia maszynowego. Jeśli jednak mam to szczęście opowiedzieć o nim jako pierwszy, to w dużym skrócie: biolog Edgar Anderson zajmował się klasyfikacją gatunków kwiatów Irys (setosa, irginica i versicolor). Tego samego dnia, na tej samej polanie, jedna i ta sama osoba, posługując się tym samym przyrządem zebrała i zmierzyła 50 kwiatów. Mierzono długość i szerokość działek (sepal) i płatków (petal). Okazuje się, że algorytmy uczenia maszynowego uczące się na takiej 50-cio elementowej próbie są w stanie szybko nauczyć się odgadywać gatunek kwiatu w oparciu o jego rozmiary.
- 2) Uruchom poniższy kod, aby załadować zbiór danych, pobrać z niego tylko dwie pierwsze kolumny, dodać informację o gatunku i wyświetlić scatter plot przedstawiający każdy gatunek kwiatu w innym kolorze:

```
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd

# import data
iris = datasets.load_iris()
x = iris.data[:, :2] # take only the first two columns
y = iris.target
df = pd.DataFrame(x, columns=['sepal_l', 'sepal_w'])
df['species'] = iris.target
print(df)

# scatter plot
plt.figure(2, figsize=(8, 6))
plt.scatter(df['sepal_l'], df['sepal_w'], c=df['species'],
            cmap=plt.cm.get_cmap('gist_rainbow'))
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.show()
```

- 3) Odfiltruj dane w `df`, tak aby zawierał tylko kwiaty sklasyfikowane jako `species == 0` (to gatunek setosa, dodaję na wypadek, gdyby Twoja dziewczyna/chłopak zajmował się data science i zechcesz podarować jej/mu oryginalny bukiet)



- 4) Korzystając z metody describe wyświetl informacje o medianie i percentylach dla kolumny sepal\_l i sepal\_w.
- 5) Narysuj wykres boxplot i odczytaj z wykresu, w której kolumnie znajdują się outlierzy
- 6) Wyznacz dla tej kolumny Q1, Q3 oraz IQR.
- 7) Dowolną metodą odfiltruj zbiór iris usuwając z niego outlierów
- 8) Wyświetl wykres boxplot jeszcze raz. Zauważ, że outlierzy zniknęli

### Propozycja rozwiązania

```
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd

# import data
iris = datasets.load_iris()
X = iris.data[:, :2] # take only the first two columns
y = iris.target
df = pd.DataFrame(X, columns=['sepal_l', 'sepal_w'])
df['species'] = iris.target

# scatter plot
plt.figure(2, figsize=(8, 6))
plt.scatter(df['sepal_l'], df['sepal_w'], c=df['species'],
            cmap=plt.cm.get_cmap('gist_rainbow'))
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.show()

df = df[df['species']==0]
print(df)

print(df.describe())

fig, axes = plt.subplots(nrows=1,ncols=2)
df.boxplot('sepal_l', ax = axes[0], meanline=True)
df.boxplot('sepal_w', ax = axes[1], meanline=True)
plt.show()

Q1 = df['sepal_w'].quantile(0.25)
Q3 = df['sepal_w'].quantile(0.75)
IQR = Q3 - Q1
df = df[(Q1 - 1.5*IQR < df['sepal_w']) & (df['sepal_w'] < Q3 + 1.5*IQR)]

fig, axes = plt.subplots(nrows=1,ncols=2)
df.boxplot('sepal_l', ax = axes[0], meanline=True)
df.boxplot('sepal_w', ax = axes[1], meanline=True)
plt.show()
```

## Detekcja outlierów i nieco uczenia maszynowego

### Notatka

Jeśli rozkład danych jest normalny (brak dłuższego ogona na wykresie), to sensowne wydaje się założenie, że wartości próby powinny znajdować się stosunkowo niedaleko wartości średniej. Często przyjmuje się, że wartości nie są odstające, jeśli mieszczą się w odległości nie większej niż 3 odchylenia standardowe.

Sprawdzając, czy wartość jest outlierem, można sprawdzać, czy znajduje się poza przedziałem:

$$[\mu - 3 * \sigma, \quad \mu + 3 * \sigma]$$

W Pythonie można by to zrobić tak:

```
fare_amount_mean = taxi['fare_amount'].mean()
fare_amount_stdev = taxi['fare_amount'].std()

tip_amount_mean = taxi['tip_amount'].mean()
tip_amount_stdev = taxi['tip_amount'].std()

fare_amount_lower = fare_amount_mean - 3 * fare_amount_stdev
fare_amount_upper = fare_amount_mean + 3 * fare_amount_stdev

tip_amount_lower = tip_amount_mean - 3 * tip_amount_stdev
tip_amount_upper = tip_amount_mean + 3 * tip_amount_stdev

outliers = taxi.query(f"(fare_amount<{fare_amount_lower} or
fare_amount>{fare_amount_upper}) or"+
                      f"(tip_amount<{tip_amount_lower} or
tip_amount>{tip_amount_upper})")
```

Zaprezentowane metody są dobre dla zbiorów jednowymiarowych. W przypadku zbiorów dwuwymiarowych należałoby analizować jednocześnie dwie wartości.

Jeśli podejrzewamy, że dane są ułożone liniowo (scatter plot rysuje dużo punktów wzdłuż prostej), to można zastosować algorytm RANSAC. Bez wchodzenia w szczegóły, jak ten algorytm działa (więcej o nim na kursie uczenia maszynowego), można wykorzystać gotowe funkcje wchodzące w skład modułu `sklearn.linear_model`. Aby zainstalować ten pakiet uruchom:

```
pip install scikit-learn
```

Zbudowanie modelu uczenia maszynowego, z wykorzystaniem `RANSACRegressor` można zaimplementować tak:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import RANSACRegressor, LinearRegression

taxi = pd.read_parquet('./yellow_tripdata_2021-05.parquet', engine='auto',
                      columns=['trip_distance', 'tpep_pickup_datetime',
                              "tip_amount", "fare_amount", "RatecodeID"],
                      storage_options=None, use_nullable_dtypes=False)

taxi.query("tpep_pickup_datetime>='2021-05-09' and tpep_pickup_datetime<'2021-05-10'",
          inplace=True)

taxi.query("tip_amount<=100 and fare_amount>0 and RatecodeID==4 and tip_amount>0",
          inplace=True)
```

```

ransac = RANSACRegressor(estimator=LinearRegression(),
                          #min_samples=50,
                          max_trials=100,
                          loss='squared_error', random_state=42,
                          residual_threshold=10)

ransac.fit(taxi['fare_amount'].to_numpy().reshape(-1,1),
           taxi['tip_amount'].to_numpy().reshape(-1,1))

inliers_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inliers_mask)

plt.scatter(taxi['fare_amount'][inliers_mask], taxi['tip_amount'][inliers_mask],
            c='blue', edgecolor='white', marker='o', label='Inliers')

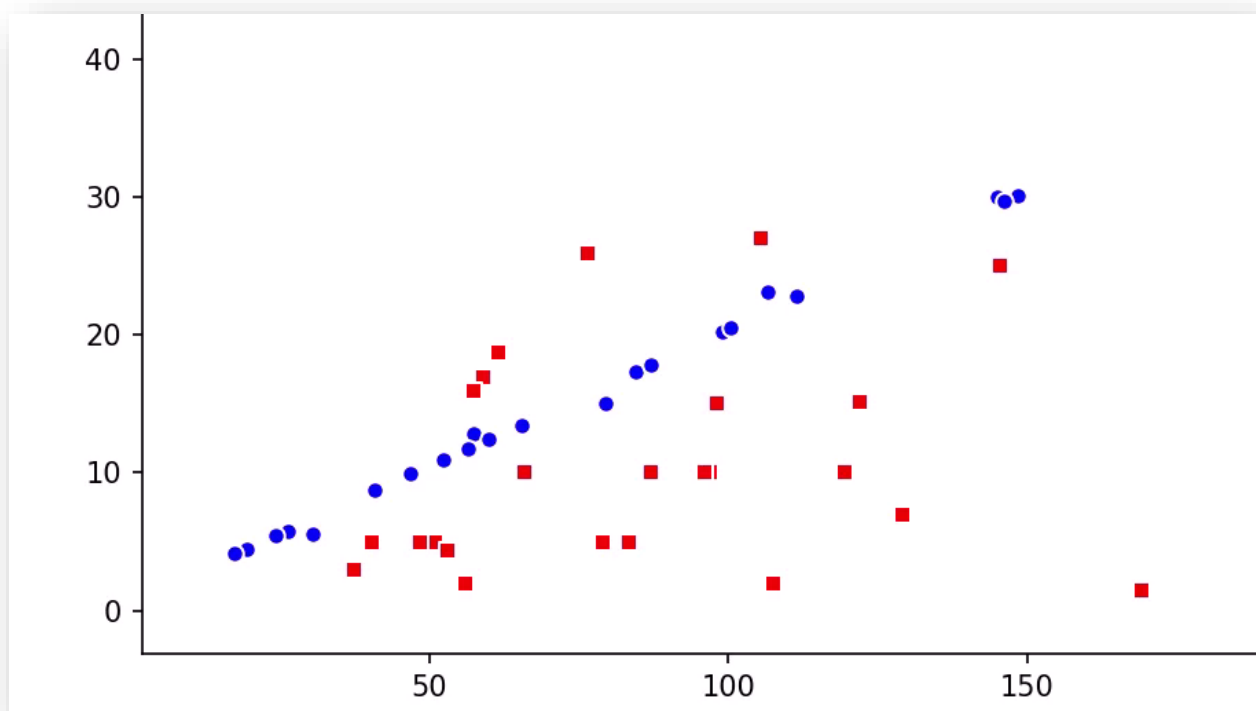
plt.scatter(taxi['fare_amount'][outlier_mask], taxi['tip_amount'][outlier_mask],
            c='red', edgecolor='white', marker='s', label='Outliers')

plt.show()

```

Algorytm uczenia maszynowego zwykle pracują w dwóch fazach:

- Nauka w oparciu o dane. Proces polega na właściwym doborze parametrów pewnego wewnętrznego równania, tak aby uzyskiwany błąd był jak najmniejszy.
- Uruchomienie algorytmu na nowych danych, ewentualnie odczytanie wartości wyznaczonych na etapie uczenia.



- 1) W tym laboratorium zobaczymy jak działa omówiony tu algorytm eliminacji outlierów dla zbioru iris. Korzystając z poniższego kodu zaimportuj dane iris i odfiltruj go zostawiając tylko gatunek setosa (`species == 0`)

```
...
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd

# import data
iris = datasets.load_iris()
X = iris.data[:, :2] # take only the first two columns
y = iris.target
df = pd.DataFrame(X, columns=['sepal_l', 'sepal_w'])
df['species'] = iris.target

# scatter plot
plt.figure(2, figsize=(8, 6))
plt.scatter(df['sepal_l'], df['sepal_w'], c=df['species'],
            cmap=plt.cm.get_cmap('gist_rainbow'), label=df['species'])
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
#plt.colorbar()
plt.show()

df = df[df['species']==0]
print(df)...
```

- 2) Wyznacz średnią i odchylenie standardowe dla kolumn `sepal_l` i `sepal_w`
- 3) Wyznacz najniższą i najwyższą wartość, którą będziemy traktować jako dopuszczalną dla `sepal_l` i `sepal_w`. W obliczeniach przyjmij „za normalne” te wartości, które są oddalone od średniej o maksymalnie 2 odchylenia standardowe.
- 4) Narysuj wykres box plot i zaobserwuj wartości odstające
- 5) Odfiltruj wartości odstające i narysuj wykres box plot ponownie. Zauważ, że po zmianach inna wartość jest teraz rozpoznawana jako odstająca. Jest to całkiem możliwy scenariusz w pracy z danymi, bo po eliminacji wartości odstających zmieniła się również średnia.
- 6) Narysuj scatter plot (wykres punktowy) ilustrujący wartości inlier-ów i Outlier-ów w różnych kolorach.
- 7) Uruchom poniższy skrypt, który korzystając z regresora RANSAC ustali wartości odstające przy założeniu że istnieje relacja liniowa między `sepal_l` i `sepal_w`:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import RANSACRegressor, LinearRegression
from sklearn import datasets

# import data
iris = datasets.load_iris()
X = iris.data[:, :2] # take only the first two columns
y = iris.target

df = pd.DataFrame(X, columns=['sepal_l', 'sepal_w'])
df['species'] = iris.target
df = df[df['species']==0]
```

```

ransac = RANSACRegressor(estimator=LinearRegression(),
                          max_trials=100,
                          loss='squared_error', random_state=42,
                          residual_threshold=0.1)

ransac.fit(df['sepal_l'].to_numpy().reshape(-1,1), df['sepal_w'].to_numpy().reshape(-1,1))

inliers_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inliers_mask)

plt.scatter(df['sepal_l'][inliers_mask], df['sepal_w'][inliers_mask],
            c='blue', edgecolor='white', marker='o', label='Inliers')

plt.scatter(df['sepal_l'][outlier_mask], df['sepal_w'][outlier_mask],
            c='red', edgecolor='white', marker='s', label='Outliers')

plt.show()

```

### Propozycja rozwiązania

```

import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd

# import data
iris = datasets.load_iris()
X = iris.data[:, :2] # take only the first two columns
y = iris.target
df = pd.DataFrame(X, columns=['sepal_l', 'sepal_w'])
df['species'] = iris.target

# scatter plot
plt.figure(2, figsize=(8, 6))
plt.scatter(df['sepal_l'], df['sepal_w'], c=df['species'],
            cmap=plt.cm.get_cmap('gist_rainbow'), label=df['species'])
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
#plt.colorbar()
plt.show()

df = df[df['species']==0]
print(df)

sepal_length_mean = df['sepal_l'].mean()
sepal_length_stddev = df['sepal_l'].std()
sepal_width_mean = df['sepal_w'].mean()
sepal_width_stddev = df['sepal_w'].std()

sepal_length_lower = sepal_length_mean - 2 * sepal_length_stddev
sepal_length_upper = sepal_length_mean + 2 * sepal_length_stddev
sepal_width_lower = sepal_width_mean - 2 * sepal_width_stddev
sepal_width_upper = sepal_width_mean + 2 * sepal_width_stddev

fig, axes = plt.subplots(nrows=1, ncols=2)
df.boxplot('sepal_l', ax = axes[0], meanline=True)
df.boxplot('sepal_w', ax = axes[1], meanline=True)
plt.show()

outliers = df.query(f"(sepal_l<{sepal_length_lower} or sepal_l>{sepal_length_upper}) or "+
                    f"(sepal_w<{sepal_width_lower} or sepal_w>{sepal_width_upper})")
df.query(f"(sepal_l>={sepal_length_lower} and sepal_l<={sepal_length_upper}) and "+
        f"(sepal_w>={sepal_width_lower} and sepal_w<={sepal_width_upper})", inplace=True)

```

```
print(outliers)

fig, axes = plt.subplots(nrows=1,ncols=2)
df.boxplot('sepal_l', ax = axes[0], meanline=True)
df.boxplot('sepal_w', ax = axes[1], meanline=True)
plt.show()

plt.scatter(df['sepal_l'], df['sepal_w'], color='blue')
plt.scatter(outliers['sepal_l'], outliers['sepal_w'], color='red')
plt.show()
```

## Kowariancja i korelacja

### Notatka

Czasami w danych jesteśmy w stanie zaobserwować zależności między różnymi cechami badanych obiektów, np. zależność między występowaniem słodyczy w diecie, a liczbą zębów zaatakowanych przez próchnicę.

W statystyce mówimy o:

- **Kowariancji/korelacji ujemnej** – gdy większym wartościom zmiennej X odpowiadają mniejsze wartości zmiennej Y
- **Kowariancji/korelacji dodatniej** – gdy większym wartościom zmiennej X odpowiadają większe wartości zmiennej Y

Kowariancję można wyznaczać dla próby i dla populacji przy pomocy wzorów:

$$s_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x}) * (y_i - \bar{y})}{n - 1}$$

$$\sigma_{xy} = \frac{\sum_{i=1}^N (x_i - \mu_x) * (y_i - \mu_y)}{N}$$

$s_{xy}$  – kowariancja między zmiennymi x i y w próbie

$\sigma_{xy}$  – kowariancja między zmiennymi x i y w populacji

$\bar{x}, \bar{y}$  – średnia wartość zmiennej x i y w próbie

$\mu_x, \mu_y$  – średnia wartość zmiennej x i y w populacji

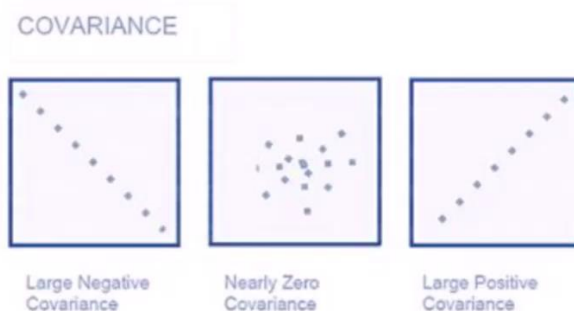
$n$  – liczebność próby

$N$  – liczebność populacji

Wynikiem powyższych obliczeń może być:

- Liczba dodatnia – co oznacza kowariancję dodatnią
- Liczbę ujemną – co oznacza kowariancję ujemną

Im większa wartość bezwzględna tak wyznaczonej wartości tym kowariancja jest silniejsza, a jeśli wartość bezwzględna zbliża się do zera, to wartości są słabo skorelowane lub wcale nie są skorelowane.



W Pandas kowariancję można wyznaczyć przy pomocy metody cov wywoływanej na rzecz serii danych:

```
taxi['fare_amount'].cov(taxi['tip_amount'])
```

Ponieważ podczas badania danych możemy nie wiedzieć, które zmienne są ze sobą powiązane, to można też uruchomić funkcję `cov` na rzecz obiektu Data Frame lub tylko jego wybranych kolumn:

```
taxi[['trip_distance', 'fare_amount', 'tip_amount', 'RatecodeID']].corr()
```

Możliwy wynik to tabela, w której na przecięciu wiersza i kolumny można odczytać kowariancję:

	trip_distance	fare_amount	tip_amount	RatecodeID
trip_distance	192.366561	715.442695	70.140695	0.0
fare_amount	715.442695	3385.828014	386.660754	0.0
tip_amount	70.140695	386.660754	128.459962	0.0
RatecodeID	0.000000	0.000000	0.000000	0.0

Ponieważ trudno jest zinterpretować uzyskane wyniki, to wprowadzono pojęcie korelacji lub inaczej współczynnika korelacji). Korelacja jest wartością ustandaryzowaną z zakresu -1 do 1. Porównanie korelacji wyznaczanych dla różnych zmiennych pozwala odpowiedzieć na pytanie, które zmienne są ze sobą skorelowane najmocniej. Korelację wyznaczamy wzorami:

$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{\sum_{i=1}^n (x_i - \bar{x}) * (y_i - \bar{y})}{(n - 1) s_x s_y}$$

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^N (x_i - \mu_x) * (y_i - \mu_y)}{N \sigma_x \sigma_y}$$

W Pandas są gotowe funkcje wyznaczające korelację:

```
taxi['fare_amount'].corr(taxi['tip_amount'])
taxi[['trip_distance', 'fare_amount', 'tip_amount', 'RatecodeID']].corr()
```

Wynik to ponownie tabela. Ale tym razem wartości można już łatwiej zinterpretować:

	trip_distance	fare_amount	tip_amount	RatecodeID
trip_distance	1.000000	0.886498	0.446192	NaN
fare_amount	0.886498	1.000000	0.586292	NaN
tip_amount	0.446192	0.586292	1.000000	NaN
RatecodeID	NaN	NaN	NaN	NaN

Moduł `seaborn` zawiera kilka przydatnych funkcji pozwalających „zobaczyć korelację”. Moduł zainstalujesz poleceniem:

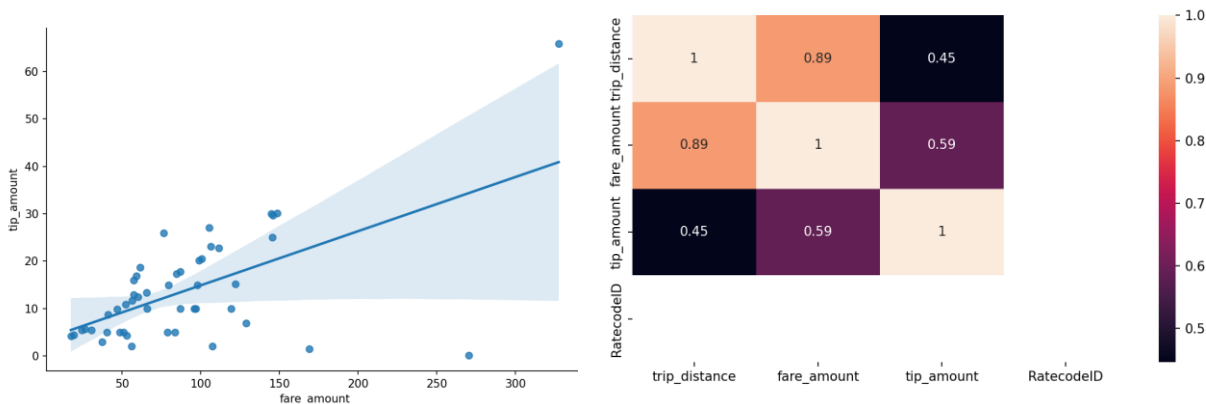
```
pip install seaborn
```

Poniższe dwa fragmenty kodu pozwolą obserwować liniową zależność między dwiema zmiennymi, albo wyświetlić „mapę ciepłą” pokazującą, które zmienne najbardziej od siebie zależą:

```
import seaborn as sns
sns.lmplot(x="fare_amount", y="tip_amount", data=taxi[['fare_amount', 'tip_amount']]);
plt.show()
```

```
import seaborn as sns
ax=sns.heatmap(taxi[['trip_distance', 'fare_amount', 'tip_amount',
'RatecodeID']].corr(), annot=True)
plt.show()
```





## Lab

- 1) Korzystając z poniższego kodu załaduj zbiór danych housing. Znajdują się w nim dane dotyczące cen nieruchomości oraz informacje o nieruchomości, np. powierzchnia, czy liczba łazienek:

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

houses = pd.read_csv("../datasets/housing/Housing.csv")
```

- 2) Zapoznaj się z danymi. Wyświetl:
    - a) Kilka początkowych wierszy (metoda head obiektu DataFrame)
    - b) Nazwy kolumn (właściwość columns dla obiektu DataFrame)
    - c) Liczba obserwacji w zbiorze (zwykła funkcja len)
  - 3) Wyznacz:
    - a) Kowariancję między liczbą łazienek a ceną
    - b) Korelację między liczbą łazienek a cenę
  - 4) Zbuduj i wyświetl:
    - a) Tablicę prezentującą kowariancję między wszystkimi kolumnami numerycznymi
    - b) Tablicę prezentującą korelację między wszystkimi kolumnami numerycznymi
- Wskazówka:** metody cov i corr można wywołać bezpośrednio dla obiektu DataFrame przekazując parameter numeric\_only=True
- 5) Narysuj „mapę cieplną” dla tablicy korelacji:
    - a) Użyj argumentu cmap określającego kolorystykę równą „Spectral”
    - b) Użyj argumentu annot=True, który spowoduje umieszczenie dodatkowo na mapie cieplnej wartości numerycznych dla prezentowanych danych.
  - 6) Zapoznaj się z dokumentacją funkcji sns.heatmap i wartości cmap. Pobaw się programem zmieniając kolorystykę lub opcje wykresu:
    - a) <https://seaborn.pydata.org/generated/seaborn.heatmap.html>
    - b) [https://seaborn.pydata.org/tutorial/color\\_palettes.html](https://seaborn.pydata.org/tutorial/color_palettes.html)

## Propozycja rozwiązania

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

houses = pd.read_csv("./datasets/housing/Housing.csv")

print(houses.head(5))
print(houses.columns)
print(len(houses))

print('Covariance between number of bathrooms and price')
print(houses["bathrooms"].cov(houses["price"]))

print('Correlation between number of bathrooms and price')
print(houses["bathrooms"].corr(houses["price"]))

covariance_tab = houses.cov(numeric_only=True)
print("Covariance table")
print(covariance_tab)

correlation_tab = houses.corr(numeric_only=True)
print('Correlation table')
print(correlation_tab)

# https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://seaborn.pydata.org/tutorial/color_palettes.html
sns.heatmap(correlation_tab, cmap="Spectral", annot=True)
plt.show()
```

Rozkład danych to funkcja, która każdej wartości ze zbioru przypisuje spodziewaną częstość wystąpienia tej wartości w populacji. Rysując wykres takiej funkcji uzyskujemy kształt podobny do histogramu, jednak precyzyjnie mówiąc, rozkład to nie wykres, ani nie histogram, tylko funkcja.

W niektórych przypadkach, jak np. rzut kostką, jesteśmy w stanie bardzo precyzyjnie opisać funkcję rozkładu, np. dla rzutu kostką można by taką funkcję przedstawić w taki sposób:

$$f(x) = \begin{cases} \frac{1}{6} & \text{dla } x \in \{1, 2, 3, 4, 5, 6\} \\ 0 & \text{dla } x \notin \{1, 2, 3, 4, 5, 6\} \end{cases}$$

Funkcję rozkładu można czasami wyznaczać w oparciu o reguły matematyczne, a w pozostałych przypadkach – funkcję trzeba odgadnąć lub ustalić w oparciu o eksperymenty np. w oparciu o wartości znajdowane w analizowanych próbach.

Poniższy program próbuje zbudować histogram dla sytuacji w której rzucamy 3 kostkami do gry i sumujemy liczbę oczek. Uzyskiwana wysokość słupków ilustruje ile razy udało się uzyskać daną sumę oczek. Oczywiście to tylko eksperyment i lepiej byłoby wykorzystać wiedzę matematyczną do zbudowania bardziej precyzyjnej funkcji rozkładu:

Wyznaczanie rozkładu dla danych wygenerowanych poprzez rzut 3 kostkami do gry i sumowaniu liczby oczek mogłoby wyglądać tak:

```
from random import randint
import matplotlib.pyplot as plt

dice = []
trials = 100

for i in range(trials):
    dice.append(randint(1,6) + randint(1,6)+ randint(1,6))

min_result = trials
max_result = 0

for i in range(3,19):
    result = dice.count(i)
    print(f'{i} - {result}')
    if min_result > result:
        min_result = result
    elif max_result < result:
        max_result = result

print(f'Difference: {max_result - min_result}')

plt.bar(list(range(3,19)), [dice.count(r) for r in range(3,19)], color='g')
plt.show()
```

## Lab

- 1) Przeprowadź symulację rzucania 5 kostkami do gry i sumowania liczby otrzymanych oczek. Zmieniaj liczbę wykonywanych rzutów i sprawdź czy uzyskiwany wykres dla takiego rozkładu przypomina „dzwon”.

## Propozycja rozwiązania

```
from random import randint
import matplotlib.pyplot as plt

#### 5 dices

dice = []
trials = 1000000

for i in range(trials):
    dice.append(randint(1,6) + randint(1,6)+ randint(1,6)+ randint(1,6)+ randint(1,6))

min_result = trials
max_result = 0

for i in range(5,31):
    result = dice.count(i)
    print(f"{i} - {result}")
    if min_result > result:
        min_result = result
    elif max_result < result:
        max_result = result

print(f'Difference: {max_result - min_result}')

plt.bar(list(range(5,31)), [dice.count(r) for r in range(5,31)], color='g')
plt.show()
```

## Rozkład normalny (Gaussowski, Bell-Curve)

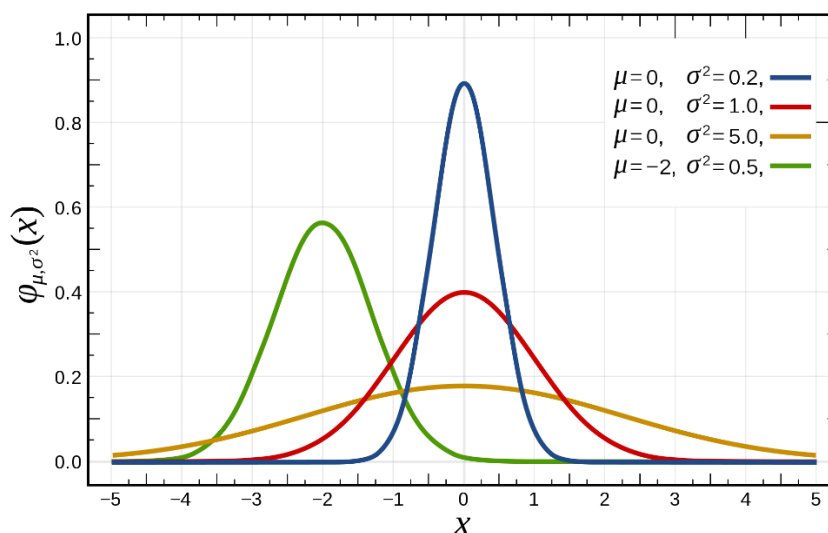
### Notatka

Zwykle specjalną uwagę poświęca się rozkładowi normalnemu, ponieważ często dobrze opisuje on zdarzenia obserwowane w naturze lub w biznesie. Oprócz rozkładu normalnego i T-Studenta (o którym opowiemy w kolejnej lekcji) jest jeszcze wiele innych typów rozkładów, które posiadają swoje specyficzne właściwości.

Odwołując się do rozkładu normalnego nie podajemy jego wzoru (jeśli są potrzebne wartości, to odczytujemy je ze specjalnej tabeli). Zamiast tego określamy, że rozkład normalny  $N$  jest definiowany przez wartość średnią populacji i odchylenie standardowe populacji. Zwróć uwagę na to odwoływanie się do populacji. W przypadku rozkładu T-Studenta już tak nie będzie:

$$N \sim (\mu, \sigma^2)$$

Położenie wykresu dla tak zdefiniowanego rozkładu może być przesunięte w prawo lub w lewo w zależności od wartości średniej i może być mniej lub bardziej rozłożyste zależnie od wartości odchylenia standardowego, co pokazuje ilustracja z Wikipedii poniżej:



Aby ustandaryzować dane można od każdej wartości odejmować wartość średnią i dzielić ją przez odchylenie standardowe:

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

Tak uzyskany rozkład będzie centralnie ułożony wokół zera, a większość danych będzie znajdować się w przedziale od -1 do 1, będzie go więc można określić wzorem:

$$N \sim (0, 1)$$

W Pythonie standaryzację możesz przeprowadzić w następujący sposób:

```
import pandas as pd
from matplotlib import pyplot as plt

blood = pd.read_csv('./datasets/blood_pressure.csv',
                    usecols=['Dayofwk', 'Meal_Time', 'Sys(mmHg)', 'Pulse(bpm)'])

blood_mean = blood['Sys(mmHg)'].mean()
blood_stdev = blood['Sys(mmHg)'].std()
blood["sys-z-score"] = (blood['Sys(mmHg)'] - blood_mean) / blood_stdev

plt.hist(blood['sys-z-score'], bins=9)
plt.show()
```

### Lab

- 1) Teraz przyjrzymy się temu, jak spało się Johnowi. Korzystając z poniższego kodu zaimportuj dane dotyczące snu:

```
import pandas as pd
from matplotlib import pyplot as plt

sleep = pd.read_csv('./datasets/sleep/sleep.csv',
                    usecols=['Date', 'Minutes.Asleep', 'Number.of.Awakenings'])
```

- 2) Narysuj histogramy
  - a) dla kolumny Minutes.Asleep (czas snu w minutach)
  - b) dla kolumny Number.of.Awakenings (liczba przebudzeni w czasie snu)
- 3) Żaden z wykresów idealnie nie pasuje do wykresu rozkładu normalnego, ale wraz ze zwiększeniem próby można by oczekiwać, że liczba przebudzeń przyjmie postać normalną
- 4) Ustandaryzuj dane z kolumny Numer.of.Awakenings i przedstaw ją ponownie na histogramie.

### Propozycja rozwiązania

```
# https://dataverse.harvard.edu/dataverse/r4r

import pandas as pd
from matplotlib import pyplot as plt

sleep = pd.read_csv('./datasets/sleep/sleep.csv',
                    usecols=['Date', 'Minutes.Asleep', 'Number.of.Awakenings'])

plt.hist(sleep['Minutes.Asleep'], bins=20)
plt.title('Minutes asleep')
plt.show()

sleep = sleep[sleep["Number.of.Awakenings"]>0]

plt.hist(sleep['Number.of.Awakenings'], bins=20)
plt.title('Number of awakenings')
plt.show()

sleep_mean = sleep['Number.of.Awakenings'].mean()
sleep_stdev = sleep['Number.of.Awakenings'].std()
sleep["sleep-z-score"] = (sleep['Number.of.Awakenings'] - sleep_mean) / sleep_stdev
plt.hist(sleep['sleep-z-score'], bins=20)
plt.show()
```

## Centralne twierdzenie graniczne (Central Limit Theorem)

### Notatka

To jedno z ważniejszych twierdzeń statystyki, które mówi, że jeśli z populacji będą losowane próby i dla tych prób będzie wyznaczana średnia, to zbiór średnich tych prób będzie miał średnią taką samą jak oryginalna populacja.

Utworzony nowy zbiór średnich ma również swój własny rozkład. Mówimy na niego sampling distribution, albo po polsku rozkład statystyki z próby, a dokładniej rozkład średniej z próby. Okazuje się, że nawet jeśli oryginalna populacja nie ma rozkładu normalnego, to rozkład średniej z próby jest rozkładem normalnym.

Nazwa „rozkład statystyki z próby” może być trochę dziwna. Dlatego przypomnij sobie, że kiedy wyznaczamy pewną charakterystykę dla podzbioru populacji to mówimy na nią „statystyka”, w odróżnieniu od tej samej charakterystyki wyznaczonej dla całej populacji, która jest nazywana parametrem. Wobec tego statystyka z prób to nazwa dla statystyki wyliczanej na zbiorze prób. Ponieważ wyznaczana przez nas statystyka to średnia, to możemy mówić o średniej z prób. Ten zbiór średnich z prób ma swój rozkład i stąd właśnie nazwa „rozkład statystyki z próby”.

Takie przybliżenie wartości średniej populacji odbywa się z pewnym błędem. Ten błąd wyrażamy poprzez wariancję zbioru średnich z prób. Błąd ten nazywamy błędem standardowym (standard error) i wyznaczamy go wzorem:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

$n$  to wielkość losowanych prób. Jeśli wielkość losowanej próby jest duża, to przybliżenie wartości średniej będzie dokładniejsze. W statystyce często przyjmuje się, że 30 to wystarczająco duża liczba.

Skoro znana jest wariancja, to i znane jest odchylenie standardowe. Dlatego wiedząc, że średnia próby ma rozkład normalny, możemy go opisać następująco:

$$N \sim \left(\mu, \frac{\sigma^2}{n}\right)$$

Sprawdzając w praktyce działanie twierdzenia, skorzystaliśmy z danych dotyczących opóźnień autobusów szkolnych w USA. Dane, przed przetwarzaniem trzeba było oczyścić i przetworzyć:

```
import pandas as pd
import statistics as stat
from matplotlib import pyplot as plt
import seaborn as sns
import math

delays = pd.read_csv('./datasets/bus-breakdown-and-delays.csv',
usecols=['How_Long_Delayed'])
delays['Delay'] = delays['How_Long_Delayed'].str.extract('(\d+)')
delays['is_canceled'] = delays['Delay'].isnull()
delays = delays[~ delays['is_canceled']]['Delay']
delays['Delay'] = pd.to_numeric(delays['Delay'])
# print(delays.head(10))
```

W celu sprawdzenia, na ile dobrze sprawuje się twierdzenie wyznaczaliśmy wartość średnią i odchylenie standardowe populacji

```
target_mean = delays['Delay'].mean()
target_stdev = stat.stdev(delays['Delay'])
```

Następnie przeprowadziliśmy test losując po 5, 10, 20, 30, 50 i 100 elementów z populacji. Każdorazowo losowaliśmy 100 prób. Po wylosowaniu próby, wyznaczaliśmy wartość średnią i zapisywaliśmy ją w nowym zbiorze "means". Oprócz wyświetlenia wartości liczbowych służących do zapoznania się z dokładnością wyników, generowaliśmy też wykresy przedstawiające rozkład danych w statystyce z próby.

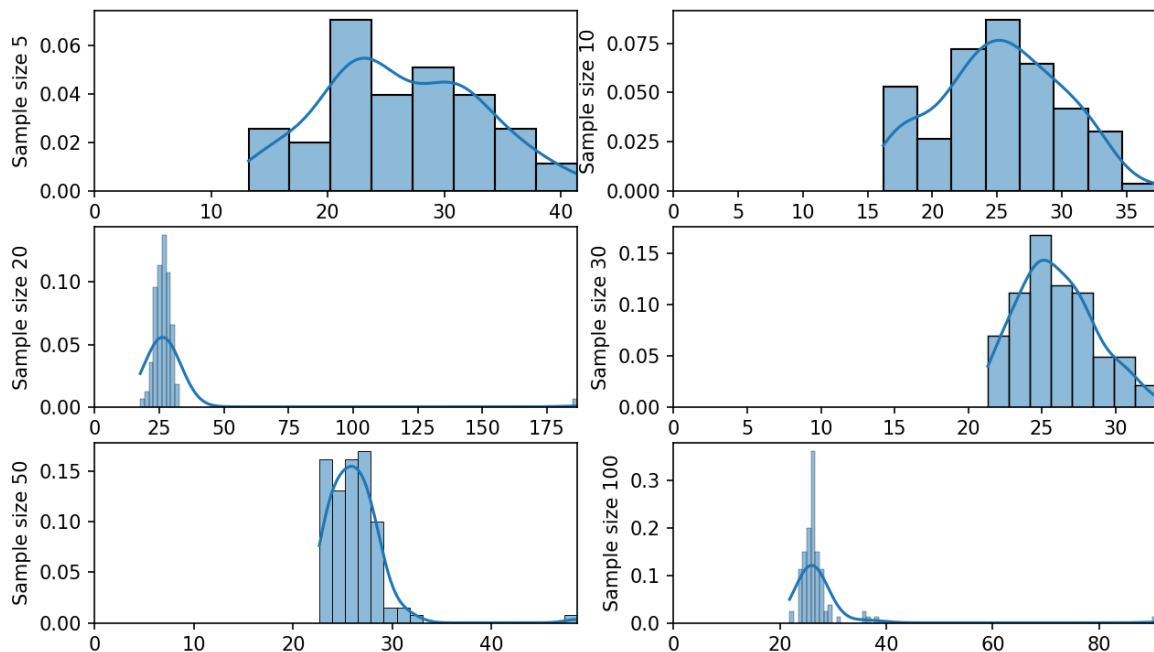
```
sample_sizes = [5, 10, 20, 30, 50, 100]
sample_number = 100
idx = 1

for ss in sample_sizes:
    means = []
    for i in range(sample_number):
        sample = delays.sample(ss)
        sample_mean = sample['Delay'].mean()
        means.append(sample_mean)

    print('Sample size:', ss)
    print('Target mean:', target_mean)
    print('Mean of means:', stat.mean(means), 'Standard deviation/Standard error:',
          target_stdev/math.sqrt(ss) )

    plt.subplot(math.ceil(len(sample_sizes)/2), 2, idx)
    plt.xlim(0, max(means))
    plt.ylabel('Sample size '+str(ss))
    sns.histplot(means, stat="density", kde=True)
    idx+=1

plt.show()
```





## Lab

- 1) Sprawdźmy, czy w oparciu o losowanie prób uda się wyznaczyć średni czas snu Johna oraz średnią liczbę przebudzeń. Korzystając z poniższego kodu zaimportuj dane:

```
import pandas as pd
import statistics as stat
from matplotlib import pyplot as plt
import seaborn as sns

sleep = pd.read_csv('./datasets/sleep/sleep.csv',
                    usecols=['Date', 'Minutes.Asleep', 'Number.of.Awakenings'])

sample_size = 30
sample_number = 20

means_sleep = []
means_awaken = []
```

- 2) W celu porównania wyników, wyznacz średni czas snu i średnią liczbę przebudzeni na skalę całej populacji.
- 3) Losu sample\_number razy próby o rozmiarze sample\_size z DataFrame sleep. Każdorazowo wyznacz średni czas snu i średnią liczbę przebudzeni i dodaj te wartości do listy means\_sleep i means\_awaken
- 4) Narysuj wykres sns.histplot, który pokaże rozkład danych dla sleep\_mean i awaken\_mean
- 5) Wyznacz średnie w listach sleep\_mean i awaken\_mean i porównaj wynik z wyznaczonymi początkowo wartościami średnich na całą populację.
- 6) Jeśli chcesz uruchom program kilka razy zmieniając sample\_size i sample\_number

## Propozycja rozwiązania

```
# code importing data like above

sleep_mean = sleep["Minutes.Asleep"].mean()
awaken_mean = sleep["Number.of.Awakenings"].mean()

for i in range(sample_number):
    sample = sleep.sample(sample_size)
    means_sleep.append(sample["Minutes.Asleep"].mean())
    means_awaken.append(sample["Number.of.Awakenings"].mean())

plt.subplot(2, 1, 1)
plt.xlim(0, max(means_sleep))
sns.histplot(means_sleep, stat="density", kde=True)

plt.subplot(2, 1, 2)
plt.xlim(0, max(means_awaken))
sns.histplot(means_awaken, stat="density", kde=True)

print(f"Sleep time: Real mean = {sleep_mean}, estimated mean = {stat.mean(means_sleep)}")
print(f"Awakenings: Real mean = {awaken_mean}, estimated mean = {stat.mean(means_awaken)}")

plt.show()
```

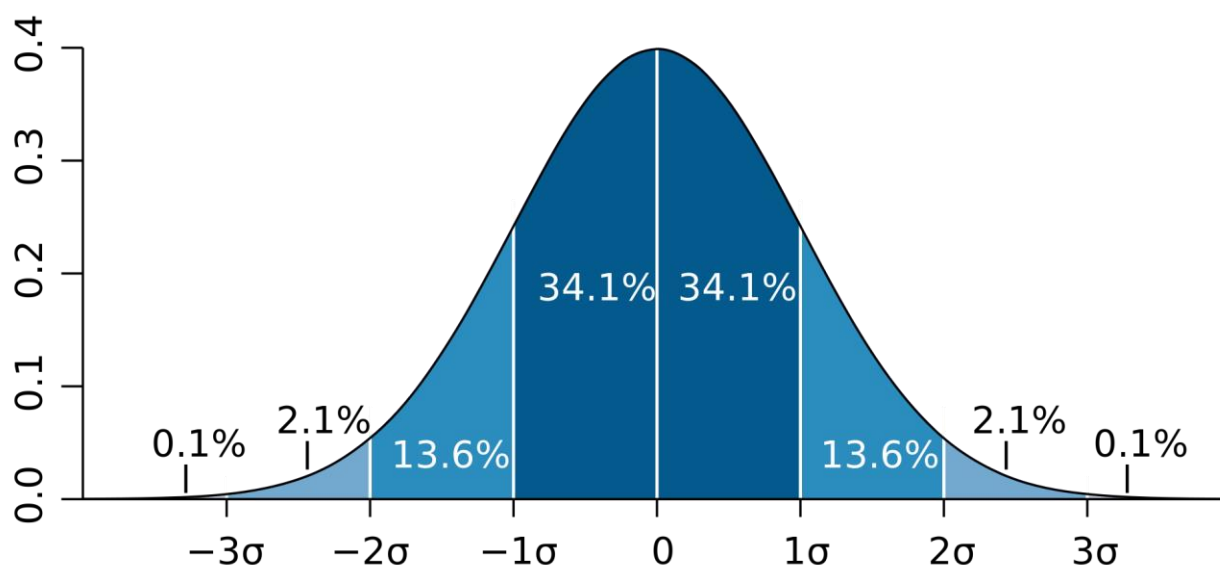
## Przedział pewności i współczynnik ufności – rozkład normalny

### Notatka

Podawanie konkretnej liczby jako estymacji parametru populacji jest ryzykowne i nie daje poczucia pewności tej odpowiedzi. Zamiast tego można próbować przybliżać poszukiwaną wartość poprzez przedział określający zakres od-do. Ten przedział to właśnie przedział pewności (confidence interval).

Taki przedział będzie szerszy lub węższy, zależnie od tego jak bardzo ma być prawdopodobne to, że poszukiwana wartość ma się w nim znajdować. Ta „pewność” to to tzw. współczynnik ufności (confidence level) i jest on liczbą od 0-100% i definiujemy go wzorem  $1 - \alpha$ .

Poszukując przedziału pewności w danych z rozkładem normalnym, należy „odciąć” najmniej prawdopodobne wartości z ogonów rozkładu:



Przykładowe dane o rozkładzie normalnym można wygenerować dedykowaną do tego funkcją rvs:

```
import scipy.stats as stats
population = stats.norm.rvs(size=100000, loc=45, scale=5)
```

Rozkład normalny można też odczytywać z tabeli rozkładu Z:

[https://en.wikipedia.org/wiki/Standard\\_normal\\_table#Reading\\_a\\_Z\\_table](https://en.wikipedia.org/wiki/Standard_normal_table#Reading_a_Z_table)

Wzór na przedział pewności to:

$$\left[ \bar{x} - z_{\alpha/2} \frac{\sigma}{\sqrt{n}}, \bar{x} + z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \right]$$

$\bar{x}$  – średnia dla próby

$z_{\alpha/2}$  – wartość krytyczna z odczytywana z tabelicy rozkładu normalnego Z

$\sigma$  – odchylenie standardowe populacji

$n$  – liczebność próby

Aby wyznaczyć wartość krytyczną  $z_{\alpha/2}$  odszukaj wartości  $1-\alpha/2$  w tablicy, a następnie dodaj do siebie wartości z nagłówka wiersza i kolumny. W Pythonie odpowiada za to funkcja:

```
z_critical = stats.norm.ppf(q=0.975)
```

Kiedy szukamy przedziału pewności ze współczynnikiem ufności 95%, to  $\alpha=5\%$  (patrz wzór  $1-\alpha$ ). Wartość 97,5% odpowiada za punkt, w którym „odcinamy ogon z prawej strony”.

W module `scipy.stats` jest dostępna dedykowana funkcja służąca do wyznaczania przedziału pewności:

```
import scipy.stats as stats  
interval = stats.norm.interval(confidence= 0.95,  
                               loc = sample_mean,  
                               scale = population_stdev/math.sqrt(sample_size))
```

### Lab

- 1) Pobawmy się w obliczanie przedziałów pewności i szukanie wartości  $z$ -critical. Poniższy kod wygeneruje dane o rozkładzie normalnym:

```
import scipy.stats as stats  
import math  
import numpy as np  
  
population = stats.norm.rvs(size=100000, loc=45, scale=5)  
population_mean = population.mean()  
population_stdev = population.std()
```

- 2) W oparciu o wylosowaną 30-elementową próbę, wyznacz przedział pewności współczynnika ufności:
  - a) 80% -  $\alpha=0.2$
  - b) 90% -  $\alpha=0.1$
  - c) 95% -  $\alpha=0.05$
- 3) Powinno udać się zaobserwować zależność, że im większa pewność, tym szerszy przedział pewności

## Propozycja rozwiązania

```
import scipy.stats as stats
import math
import numpy as np

population = stats.norm.rvs(size=100000, loc=45, scale=5)
population_mean = population.mean()
population_stdev = population.std()

sample_size = 50
sample = np.random.choice(a=population, size=sample_size)
sample_mean = sample.mean()
sample_stdev = sample.std()

# calculate confidence interval for 80% confidence level with Z-distribution
z_critical = stats.norm.ppf(q=0.90) #  $1-\alpha/2 = 1-0.2/2 = 1-0.1 = 0.9$ 
margin_of_error = z_critical * (population_stdev/math.sqrt(sample_size))
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
print(f"80%-Confidence interval Z (basing on one sample): {confidence_interval}")

# calculate confidence interval for 90% confidence level with Z-distribution
z_critical = stats.norm.ppf(q=0.95) #  $1-\alpha/2 = 1-0.1/2 = 1-0.05 = 0.95$ 
margin_of_error = z_critical * (population_stdev/math.sqrt(sample_size))
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
print(f"90%-Confidence interval Z (basing on one sample): {confidence_interval}")

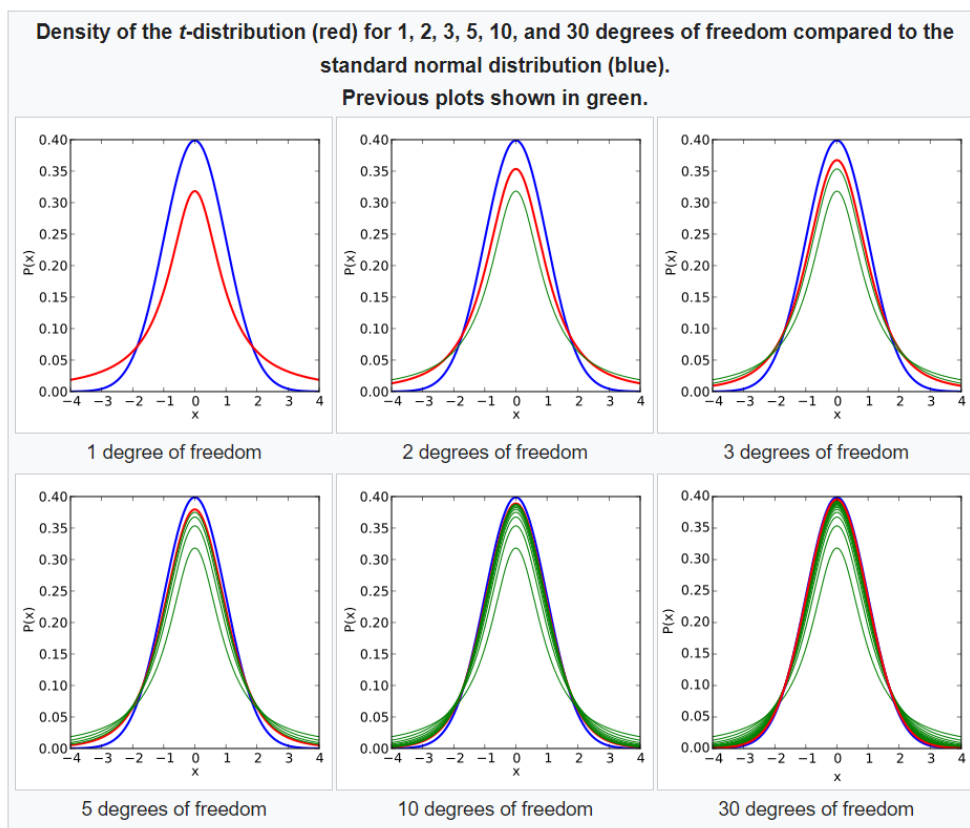
# calculate confidence interval for 95% confidence level with Z-distribution
z_critical = stats.norm.ppf(q=0.975) #  $1-\alpha/2 = 1-0.05/2 = 1-0.025 = 0.975$ 
margin_of_error = z_critical * (population_stdev/math.sqrt(sample_size))
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
print(f"95%-Confidence interval Z (basing on one sample): {confidence_interval}")
```

## Przedział pewności i współczynnik ufności – rozkład T-studenta

### Notatka

Rozkład T-Studenta jest dość podobny do rozkładu normalnego, ale „ogony” tego rozkładu są grubsze. Ten rozkład pozwala estymować parametry populacji w przypadku, gdy nie jest znane odchylenie standardowe populacji, lub gdy liczebność próby jest mała (<30).

Jednym z parametrów rozkładu T jest tzw. stopień swobody (degree of freedom) wpływający na kształt wykresu rozkładu:



Wartości funkcji charakterystycznej dla rozkładu T można odczytywać z tabeli:

<https://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf>

Wzór na przedział pewności wyznaczany metodą T-Studenta to:

$$\left[ \bar{x} - t_{n-1, \alpha/2} \frac{s}{\sqrt{n}}, \bar{x} + t_{n-1, \alpha/2} \frac{s}{\sqrt{n}} \right]$$

$\bar{x}$  – średnia dla próby

$t_{n-1, \alpha/2}$  – wartość krytyczna  $t$  odczytywana z tablicy rozkładu T-Studenta

$s$  – odchylenie standardowe próby

$n$  – liczebność próby

Wartość krytyczną  $t$  można odczytać z tablicy, lub posłużyć się funkcją:

```
import scipy.stats as stats
t_critical = stats.t.ppf(q=0.975, df=sample_size-1)
```

Istnieje gotowa funkcja wyznaczająca przedział pewności wg rozkładu T:

```
interval = stats.t.interval(confidence= 0.95,
                             df= sample_size - 1,
                             loc = sample_mean,
                             scale = sample_stdev/math.sqrt(sample_size))
print('T:', interval)
```

Gdy chcesz wygenerować testowy skośny zbiór danych możesz skorzystać z funkcji:

```
population = stats.skewnorm.rvs(a=10, size=100000, loc=25, scale=20)
```

Jeden ze scenariuszy użycia rozkładu T przy wyznaczaniu przedziałów ufności to

### Lab

- 1) Pobawmy się w obliczanie przedziałów pewności i szukanie wartości t-critical. Poniższy kod wygeneruje dane o rozkładzie normalnym:

```
import scipy.stats as stats
import math
import numpy as np

population = stats.t.rvs(size=100000, loc=45, scale=5)

sample_size = 30
sample = np.random.choice(a=population, size=sample_size)
sample_mean = sample.mean()
sample_stdev = sample.std()
```

- 2) Wyznacz przedział pewności dla współczynnika ufności:
  - a) 80% -  $\alpha=0.2$
  - b) 90% -  $\alpha=0.1$
  - c) 95% -  $\alpha=0.05$
- 3) Powinno udać się zaobserwować zależność, że im większa pewność, tym szerszy przedział pewności

## Propozycja rozwiązania

```
import scipy.stats as stats
import math
import numpy as np

population = stats.t.rvs(size=100000, loc=45, scale=5)

sample_size = 30
sample = np.random.choice(a=population, size=sample_size)
sample_mean = sample.mean()
sample_stdev = sample.std()

# calculate confidence interval for 80% confidence level with Z-distribution
t_critical = stats.t.ppf(q=0.90, df=sample_size-1) #  $1-\alpha/2 = 1-0.2/2 = 1-0.1 = 0.9$ 
margin_of_error = t_critical * (sample_stdev/math.sqrt(sample_size))
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
print(f"80%-Confidence interval T (basing on one sample): {confidence_interval}")

# calculate confidence interval for 90% confidence level with Z-distribution
t_critical = stats.t.ppf(q=0.95, df=sample_size-1) #  $1-\alpha/2 = 1-0.1/2 = 1-0.05 = 0.95$ 
margin_of_error = t_critical * (sample_stdev/math.sqrt(sample_size))
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
print(f"90%-Confidence interval T (basing on one sample): {confidence_interval}")

# calculate confidence interval for 95% confidence level with Z-distribution
t_critical = stats.norm.ppf(q=0.975, df=sample_size-1) #  $1-\alpha/2 = 1-0.05/2 = 1-0.025 = 0.975$ 
margin_of_error = t_critical * (sample_stdev/math.sqrt(sample_size))
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
print(f"95%-Confidence interval T (basing on one sample): {confidence_interval}")
print(f'Sorting duration for function quick_sort: {stop - start}')
```

## Przedział pewności w wyznaczeniu proporcji w populacji

### Notatka

Jedno z zadań statystyki to uogólnienie wyniku podziału próby na populację, np. liczba klientów sklepu, którzy odwiedzili ten sklep z powodu promocji vs liczba klientów, którzy odwiedzili sklep nie wiedząc o promocji.

Wzór pozwalający wyznaczyć przedział pewności, w którym znajdzie się taka proporcja w populacji w oparciu o proporcję próby to:

$$\left[ \hat{p} - z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}, \quad \hat{p} + z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} \right]$$

$\hat{p}$  – proporcja w próbie

$z_{\alpha/2}$  – wartość krytyczna z odczytywana z tablicy rozkładu Z

$n$  – liczebność próby

Gotowa funkcja pozwalająca na wykonanie obliczeń takiego przedziału to:

```
import scipy.stats as stats

p = 200/1000
n = 1000

confidence_interval = stats.norm.interval(
    confidence=0.95,          # Confidence level
    loc=p,                   # Point estimate of proportion
    scale=math.sqrt((p*(1-p))/n)) # Scaling factor
```

### Lab

- 1) W szkole, w której uczy się ok. 200 uczniów woźny zaobserwował, że ok.30 uczniów regularnie udaje się w ustronne miejsce za śmietnikami i pali tam papierosy. Woźny zastanawia się jak wielu uczniów w całym kraju oddaje się temu zgubnemu nałogowi. Pomóż mu to ustalić z 95% pewnością.



## Propozycja rozwiązania

```
import scipy.stats as stats
import math

# alfa = 0.05 = 5%
# alfa/2 = 0.025 = 2,5%
# confidence level = 1 - alfa = 0.95 = 95%
# z-score taken from Z-statistics table for value of 1 - alfa/2 = 1 - 0.025 = 0.975
# 1.96

z_critical = stats.norm.ppf(0.975)      # Record z-critical value
print(f'z_critical={z_critical}')

p = 30/200                             # Point estimate of proportion2
n = 200                                # Sample size

margin_of_error = z_critical * math.sqrt((p*(1-p))/n)
confidence_interval = (p - margin_of_error, p + margin_of_error)

print(f'we are 90% confident that the ratio of population falls into this interval:
{confidence_interval}')
```

```
confidence_interval = stats.norm.interval(
    confidence=0.95,                # Confidence level
    loc=p,                          # Point estimate of proportion
    scale=math.sqrt((p*(1-p))/n)) # Scaling factor

print(f'we are 90% confident that the ratio of population falls into this interval:
{confidence_interval}')
```

## Przedział pewności dla 2 prób zależnych

### Notatka

Jedno z zadań statystyka to porównywanie zbiorów danych. Zależnie od tego, czy porównywane dane są zależne od siebie, czy nie, należy korzystać z odpowiednich wzorów.

Przykład danych zależnych to np. porównanie wartości zebranych na początku eksperymentu z wynikami z końca eksperymentu, albo analiza wartości dla obiektów logicznie (a nawet uczuciowo) ze sobą powiązanych (np. relacje rodzinne).

Kiedy chcesz testowo wygenerować zbiór danych o rozkładzie normalnym możesz użyć polecenia:

```
before = stats.norm.rvs(size=size, loc=14, scale=1)
```

Jeśli chcesz jednocześnie przetwarzać dwie listy i w efekcie wyznaczyć wartość trzeciej listy, to skorzystaj z funkcji zip:

```
after = [x+y for (x,y) in zip(results_before, change)]
```

Funkcja zip przyda się też przy budowaniu Data Frame w oparciu o dwie listy:

```
df = pd.DataFrame(list(zip(before, after)), columns=['before', 'after'])
df['diff'] = df['after'] - df['before']
```

Wzór jakim można wyznaczać przedział pewności pracując z dwoma próbami zależnymi to:

$$\bar{d} \pm t_{n-1, \alpha/2} \frac{S_d}{\sqrt{n}}$$

$\bar{d}$  - średnia różnic między odpowiednimi elementami próby X i Y

$t_{n-1, \alpha/2}$  - wartość krytyczna rozkładu T, dla stopnia swobody n-1 i zadanego  $\alpha$

$S_d$  - odchylenie standardowe dla zbioru różnic wartości ze zbioru X i ze zbioru Y

$n$  - wielkość próby

Wartość  $t_{n-1, \alpha/2}$  można pobrać z tablicy rozkładu T, albo wykorzystać funkcję:

```
import scipy.stats as stats

alpha = 0.1
alpha_2 = alpha/2
sample_size = len(df)
confidence_level = 1-alpha

t_critical = stats.t.ppf(q=confidence_level+alpha_2, df=sample_size-1) # 1.699
print(f'T-critical = {t_critical}')
```

Ze względu na zachowanie rozkładu T, im większy jest współczynnik ufności (confidence level), tym większa wartość marginesu błędu (margin error) i tym większy przedział ufności (confidence interval).

- 1) Chyba niestety wiesz, co to inflacja? Zobaczmy co stało się z cenami w latach 2010-2020, ale żeby nie było nam zbyt smutno, zrobimy to z cenami w Singapurze. Dane tego laboratorium pochodzą ze strony <https://www.kaggle.com/datasets/subhamjain/prices-of-consumer-items-singapore>  
Odpowiednio przygotowany do tego zadania plik nazywa się prices-singapore.csv i jest dołączony do plików testowych.
- 2) Zaimportuj zawartość pliku do DataFrame (skorzystaj z metody pd.read\_csv)
- 3) Wyświetl średnią i odchylenie standardowe dla kolumny o nazwie 2010 i dla kolumny o nazwie 2020.
- 4) Zakładając, że w naszych danych znajduje się reprezentatywny koszyk produktów do oceny inflacji (założenie jest błędne – ale trudno znaleźć wiarygodne dane 😊) ustal przedział pewności, w którym znajdzie się średnia różnica cen produktów wyznaczona dla roku 2020 i 2010. Zrób to dla współczynnika ufności 90 i 95%.

```
import scipy.stats as stats
import statistics
import pandas as pd
import math
import random

# Two dependent values (regression: before=2010 and after=2020)
df = pd.read_csv('./datasets/prices-singapore.csv')
df['diff'] = df['2020'] - df['2010']

print('Before:', df['2010'].mean(), df['2010'].std())
print('After:', df['2020'].mean(), df['2020'].std())

# alpha = 0.1
# alpha/2 = 0.05
# confidence level = 1 - alpha = 0.9
# T-critical (degree of freedom = 52-1, alpha/2 = 0.05)

alpha = 0.1
alpha_2 = alpha/2
sample_size = len(df)
sample_mean = df['diff'].mean()
sample_stdev = df['diff'].std()
confidence_level = 1-alpha

t_critical = stats.t.ppf(q=confidence_level+alpha_2, df=sample_size-1)
print(f'T-critical = {t_critical}')

margin_of_error = t_critical * sample_stdev/math.sqrt(sample_size)
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)

print(f"We are {confidence_level} sure that the mean difference for products' price
falls between: {confidence_interval}")

# alpha = 0.05
# alpha/2 = 0.025
# confidence level = 1 - alpha = 0.95
# T-critical (degree of freedom = 29, alpha/2 = 0.025)

alpha = 0.05
alpha_2 = alpha/2
sample_size = len(df)
sample_mean = df['diff'].mean()
sample_stdev = df['diff'].std()
confidence_level = 1-alpha

t_critical = stats.t.ppf(q=confidence_level+alpha_2, df=sample_size-1)
print(f'T-critical = {t_critical}')
margin_of_error = t_critical * sample_stdev/math.sqrt(sample_size)
confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)

print(f"We are {confidence_level} sure that the mean difference for products' price
falls between: {confidence_interval}")
```

## Przedział pewności dla dwóch zbiorów niezależnych

### Notatka

Zbiory niezależne nie mogą być ze sobą w żaden sposób powiązane. To dwa zupełnie różne zbiory, które jednak z jakiegoś powodu chcemy opisać wykazując że pewne statystyki dla jednego zbioru są mniejsze/większe niż innego.

Wzór, z którego należy skorzystać, gdy oba te zbiory mają znane odchylenie standardowe to:

$$(\bar{x} - \bar{y}) \pm z_{\alpha/2} \sqrt{\frac{\sigma_x^2}{n_x} + \frac{\sigma_y^2}{n_y}}$$

$\bar{x}, \bar{y}$  - średnia próby dla zbioru X i Y

$z_{\alpha/2}$  - wartość krytyczna rozkładu Z dla zadanego  $\alpha$

$\sigma_x^2, \sigma_y^2$  - wariancja populacji dla zbioru X i Y

$n_x, n_y$  - liczba elementów w próbie ze zbioru X i Y

Im większy confidence level (współczynnik ufności) – tym szerszy confidence interval (przedział pewności).

Jeśli wariancja dla badanych populacji nie jest znana, to można bazować na wariancji próby, o ile tylko istnieją przesłanki, że oba porównywane zbiory mają tę nieznaną wariancję taką samą. W tym przypadku można skorzystać ze wzoru:

$$(\bar{x} - \bar{y}) \pm t_{n_x+n_y-2, \alpha/2} \sqrt{\frac{s_p^2}{n_x} + \frac{s_p^2}{n_y}}$$

$$s_p^2 = \frac{(n_x - 1)s_x^2 + (n_y - 1)s_y^2}{n_x + n_y - 2}$$

$t_{n_x+n_y-2, \alpha/2}$  - wartość krytyczna rozkładu T, dla stopnia swobody  $n_x + n_y - 2$  i zadanego  $\alpha$

$s_x^2, s_y^2$  - wariancja dla próby ze zbioru X i ze zbioru Y

W przypadku rozkładu T, jeśli stopień swobody jest duży, to można korzystać z wartości odpowiadających rozkładowi Z, bo oba rozkłady dla dużych prób się pokrywają.

- 1) Chcesz porównać cenę pizzy w Neapolu i Rzymie. Wygeneruj fikcyjne dane korzystając z:

```
pizza_Naples = stats.norm.rvs(size=20, loc=8, scale=1)
pizza_Rome = stats.norm.rvs(size=15, loc=12, scale=5)
```

- 2) Zakładając, że wariancja populacji jest znana i zgodna z wariancją tej próby wyznacz przedział pewności (ze współczynnikiem 90%) określający średnią różnicę ceny pizzy między tymi dwoma miastami. Możesz to najpierw zrobić na papierze, a potem w Pythonie – będzie ćwiczenie i ze statystyki i z programowania 😊. Zastanów się z jakiego wzoru skorzystać.
- 3) Zmieniamy delikatnie scenariusz. Teraz założmy, że nie znamy wariancji ceny pizzy dla populacji, ale podejrzewamy, że wariancja ceny pizzy w Rzymie jest taka sama, jak wariancja ceny w Neapolu. Odpowiednie dane wygenerujesz poleceniami:

```
pizza_Naples = stats.norm.rvs(size=20, loc=8, scale=1)
pizza_Rome = stats.norm.rvs(size=15, loc=12, scale=1)
```

- 4) Wyznacz przedział pewności (ze współczynnikiem 90%) określający średnią różnicę cen w Neapolu i Rzymie. Tutaj też możesz zacząć od papieru, a skończyć programem. Z którego wzoru skorzystać tym razem?

## Propozycja rozwiązania

```
import scipy.stats as stats
import statistics
import pandas as pd
import math

# Two independent values and variance is known (sometimes "well known")

pizza_Naples = stats.norm.rvs(size=20, loc=8, scale=1)
pizza_Rome = stats.norm.rvs(size=15, loc=12, scale=5)

mean_Naples = statistics.mean(pizza_Naples)
mean_Rome = statistics.mean(pizza_Rome)
st_dev_Naples = statistics.stdev(pizza_Naples)
st_dev_Rome = statistics.stdev(pizza_Rome)
count_Naples = len(pizza_Naples)
count_Rome = len(pizza_Rome)

alpha = 0.1
alpha_2 = alpha/2
confidence_level = 1-alpha
z_critical = stats.norm.ppf(q=1-alpha_2)
print(f'Z-critical = {z_critical}')
margin_of_error = z_critical * math.sqrt(st_dev_Naples**2/count_Naples +
st_dev_Rome**2/count_Rome)
confidence_interval = ((mean_Rome-mean_Naples) - margin_of_error, (mean_Rome-
mean_Naples) + margin_of_error)

print(f'We are {confidence_level} sure, that the pizza in Rome is more expensive\
then in Naples by: {confidence_interval}')
```

```
# Two independent values and variance is unknown and assumed to be equal

pizza_Naples = stats.norm.rvs(size=20, loc=8, scale=1)
pizza_Rome = stats.norm.rvs(size=15, loc=12, scale=1)

mean_Naples = statistics.mean(pizza_Naples)
mean_Rome = statistics.mean(pizza_Rome)
st_dev_Naples = statistics.stdev(pizza_Naples)
st_dev_Rome = statistics.stdev(pizza_Rome)
count_Naples = len(pizza_Naples)
count_Rome = len(pizza_Rome)

alpha = 0.1
alpha_2 = alpha/2
confidence_level = 1-alpha
t_critical = stats.t.ppf(q=1-alpha_2, df=count_Naples+count_Rome-2)
print(f'T-critical = {t_critical}')
var_p = ((count_Naples-1)*st_dev_Naples**2 + (count_Rome-
1)*st_dev_Rome**2)/(count_Naples+count_Rome-2)
margin_of_error = t_critical * math.sqrt(var_p/count_Naples + var_p/count_Rome)
confidence_interval = ((mean_Rome-mean_Naples) - margin_of_error, (mean_Rome-
mean_Naples) + margin_of_error)

print(f'We are {confidence_level} sure, that the pizza in Rome is more expensive\
then in Naples by: {confidence_interval}')
```

## Testowanie hipotez

### Notatka

Hipoteza, to pomysł, którą można przetestować. Zwykle udowodnić chcemy nową, przełomową hipotezę. Nazywamy ją hipotezą alternatywną (alternative hypothesis) i oznaczamy symbolem  $H_1$ . Przeciwieństwem hipotezy  $H_1$  jest stan obecny, status quo, który nazywamy hipotezą zerową (null hypothesis) i oznaczamy symbolem  $H_0$ . Gdyby takiej hipotezy nie było (odkrywamy coś naprawdę rewolucyjnego), to  $H_0$  należy na prędko zbudować jako zaprzeczenie  $H_1$ .

„Sztuczka” w testowaniu hipotez polega na tym, że zamiast przechodzić jakiś trudno definiowalny proces udowadniania hipotezy  $H_1$ , próbujemy obalić hipotezę  $H_0$ . Jeśli  $H_0$  uda się obalić, to ponieważ zaprzeczeniem  $H_0$  jest  $H_1$ , to przyjmujemy, że  $H_1$  jest prawdziwe.

Testowanie hipotez jest obarczone błędem wynikającym z tego, że pracujemy na próbie a nie na populacji. Błąd jest mniejszy, gdy próba jest większa. Badacz przed rozpoczęciem analizy powinien ustalić tzw. poziom istotności (significance level). Oznaczamy go symbolem  $\alpha$  i zazwyczaj przyjmuje on wartość 1%, 5% lub 10%. Możesz myśleć o tym parametrze jako o wadze, jaką przypisujemy wylosowanej próbie. W oparciu o poziom istotności będzie można oceniać, czy obserwowane zjawisko jest efektem istotnym statystycznie, czy raczej skutkiem błędu próby. Z poziomem istotności spotkaliśmy się już wcześniej wyznaczając przedział pewności (confidence level) wzorem  $1 - \alpha$ . W celu sprawdzenia czy hipoteza  $H_0$  jest prawdziwa czy nie wyznaczamy wartość z-score:

$$Z = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

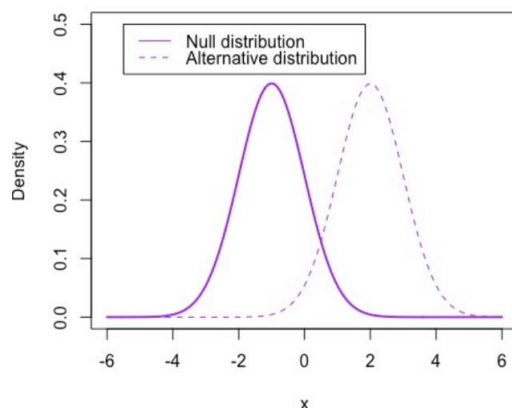
$\bar{x}$  – średnia w próbie

$\mu_0$  – średnia zakładana przez hipotezę  $H_0$

$s$  – odchylenie standardowe z próby


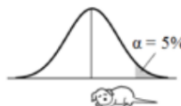
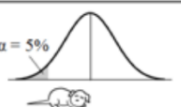
$n$  – liczebność próby

Zakładając, że zarówno  $H_0$ , jak i  $H_1$  dają się opisać rozkładem normalnym, to ilustrując ich wykresy razem w jednym układzie współrzędnych otrzymalibyśmy mniej lub bardziej oddalone od siebie wykresy rozkładu normalnego





Jeśli te dwa wykresy są blisko siebie (prawie się pokrywają), to hipoteza  $H_0$  zostanie zaakceptowana. Jeśli wykresy są daleko od siebie, to hipoteza  $H_0$  zostanie obalona. Podczas oceny „jak daleko od siebie znajdują się wykresy patrzymy na wartość z tablicy rozkładu Z (wartość krytyczna, z-critical) dla  $1-\alpha$  lub dla  $1-\alpha/2$  zależnie od tego, czy test jest jednostronny (1-tailed – jednoogonowy), czy dwustronny (2-tailed – dwuogonowy):

Comparison Operator		Tails of the Test	
$H_A$	$H_0$		
$\neq$	$=$	2-tailed	
$>$	$\leq$	1-tailed, right-tailed	
$<$	$\geq$	1-tailed, left-tailed	

Hipotezę  $H_0$  obalamy, jeśli z-score jest mniejszy od z-critical, a potwierdzamy w przeciwnym razie.

Obliczenia polegają na wykonaniu obliczeń wskazanych przez wzór – tutaj przykład dla 2-tail test:

```
# H0 - mean speed of typing is <= 40 wpm
# H1 - mean speed of typing is > 40 wpm

speed_h0 = 40

speed_test = stats.norm.rvs(size=100, loc=45, scale=15)
speed_test = speed_test.astype(int)
speed_test_len = len(speed_test)
speed_test_mean = speed_test.mean()
speed_test_stdev = speed_test.std()

z_score = (speed_test_mean - speed_h0)/(speed_test_stdev/math.sqrt(speed_test_len))

alpha = 0.05
confidence = 1 - alpha
z_critical = z_critical = stats.norm.ppf(q=confidence)

if z_score > z_critical:
    print('H0 should be rejected')
else:
    print('H0 should be accepted')
```

W praktyce, jeśli jest dostępnych więcej danych, to hipotezę należałoby potwierdzać w oparciu o coś więcej niż tylko jedna mała próba. Testowanie hipotez zostało zdefiniowane w czasach kiedy przeliczenie dużej ilości danych było niemożliwe.

Gdy próba jest mała (<30), to zamiast z rozkładu Z należałoby skorzystać z rozkładu T-Studenta.

## Lab

- 1) Jeśli chcesz, to przetestuj swoją prędkość pisania korzystając z <https://www.typingpal.com/en/typing-test>
- 2) Podobnie jak na lekcji spróbuj obalić/potwierdzić hipotezę  $H_0$  mówiącą, że piszemy na klawiaturze ze średnią prędkością 40 wpm (word per minute). Przykładowe dane próby wygeneruj następującym kodem:

```
# H0 - mean speed of typing is <= 40 wpm
# H1 - mean speed of typing is > 40 wpm

speed_h0 = 40

speed_test = stats.norm.rvs(size=20, loc=45, scale=15)
speed_test = speed_test.astype(int)
```

- 3) Ponieważ zbiór testowy jest mały (20 punktów), to zastosuj test T. Jedyna zmiana to inny sposób wyznaczenia wartości critical. Zamiast funkcji `stats.norm.ppf` należy użyć `stats.t.ppf` i dodatkowo przekazać jako parametr degree of freedom.

## Propozycja rozwiązania

```
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas as pd
import math

# H0 - mean speed of typing is <= 40 wpm
# H1 - mean speed of typing is > 40 wpm

speed_h0 = 40

speed_test = stats.norm.rvs(size=20, loc=45, scale=15)
speed_test = speed_test.astype(int)

speed_test_len = len(speed_test)
speed_test_mean = speed_test.mean()
speed_test_stdev = speed_test.std()

t_score = (speed_test_mean - speed_h0)/(speed_test_stdev/math.sqrt(speed_test_len))
print(f't-score={t_score}')

alpha = 0.05
confidence = 1 - alpha
t_critical = stats.t.ppf(q=confidence, df=speed_test_len-1)
print(f't-critical={t_critical}')

if t_score > t_critical:
    print('H0 should be rejected')
else:
    print('H0 should be accepted')
```

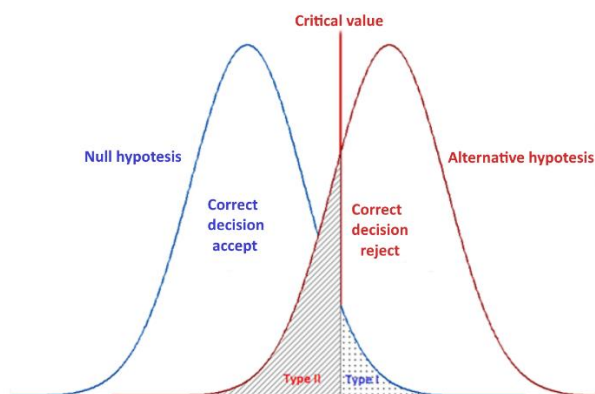
## Błędy pierwszego i drugiego typu. P-value

### Notatka

Błędy statystyczne związane z losowaniem próby są zupełnie normalne. W przypadku testowania hipotez rozróżniamy dwa rodzaje błędów podsumowane w poniższej tabelce:

Tabela typów błędów		Hipoteza zerowa ( $H_0$ ) jest	
		Prawdziwa	fałszywa
Decyzja podejmowana dla hipotezy zerowej ( $H_0$ )	Akceptacja	Wynik poprawny (true negative) (prawdopodobieństwo = $1-\alpha$ )	Błąd typu II (false negative) (prawdopodobieństwo = $\beta$ )
	Odrzucenie	Błąd typu I (false positive) (prawdopodobieństwo = $\alpha$ )	Wynik poprawny (true positive) (prawdopodobieństwo = $1-\beta$ )

Błąd typu I można ograniczać zmieniając wartość poziomu istotności  $\alpha$ . Zwiększając go odcinamy coraz to dłuższe „ogony”, przez co zwiększamy szansę na odrzucenie hipotezy, z kolei zmniejszając alfa, odcinamy coraz mniej doprowadzając do większego prawdopodobieństwa akceptacji.



Prawdopodobieństwo wystąpienia błędu typu I to  $\alpha$ , a prawdopodobieństwo wystąpienia błędu II typu to  $\beta$ . Moc testu to prawdopodobieństwo uniknięcia błędu drugiego rodzaju, wynosi ono  $1 - \beta$ .

P-value to najmniejsza wartość poziomu istotności  $\alpha$ , która powoduje odrzucenie hipotezy zerowej. Statystyk znając wartość p-value może akceptować/odrzucać hipotezę bez wyznaczania z-score i odgadywania wartości poziomu istotności. Jeśli okaże się, że p-value jest bardzo małe, to z dużym prawdopodobieństwem można odrzucić hipotezę zerową.

Wartość p-value można wyznaczyć funkcją ztest:

```
from statsmodels.stats.weightstats import ztest
ztest(speed_test, value=speed_h0)
```

- 1) Kontynuujemy temat z lekcji, jednak z drobną różnicą. Teraz hipoteza zerowa mówi, że „tempo pisanie na klawiaturze jest  $\leq 40$  wpm”, zaś hipoteza alternatywna mówi, że „tempo pisanie na klawiaturze jest większe niż 40 wpm”.
- 2) Zmodyfikuj kod z lekcji, tak aby przeprowadzić test jednostronny. Poniżej znajdziesz oryginalny kod do przeróbki. Zapoznaj się z dokumentacją funkcji `ztest`, a szczególnie przyjrzyj się argumentowi `alternative`, ponieważ dla testu jednostronnego trzeba użyć wartości parametru innego niż wartość domyślna. <https://www.statsmodels.org/dev/generated/statsmodels.stats.weightstats.ztest.html>

```
import matplotlib.pyplot as plt
import statistics
from statsmodels.stats.weightstats import ztest
import scipy.stats as stats
import pandas as pd
import math

speed_h0 = 40

speed_test = [41, 42, 34, 37, 40, 15, 46, 27, 35, 55, 52, 64, 51, 57, 49, 38, 76, 45,
65, 57, 68, 79, 44, 43, 23, 58, 53, 56, 15, 32, 37, 54, 43, 47, 55, 38, 50, 48, 55,
71, 78, 46, 54, 36, 44, 40, 31, 13, 55, 45, 40, 21, 27, 72, 54, 15, 30, 24, 20, 63,
18, 45, 48, 36, 42, 52, 55, 34, 47, 47, 11, 39, 47, 59, 52, 33, 36, 38, 63, 47, 40,
43, 20, 65, 43, 61, 29, 64, 64, 46, 53, 59, 34, 47, 41, 24, 51, 54, 60, 39]

speed_test_len = len(speed_test)
speed_test_mean = statistics.mean(speed_test)
speed_test_stdev = statistics.stdev(speed_test)

z_score = (speed_test_mean - speed_h0)/(speed_test_stdev/math.sqrt(speed_test_len))

for i in range(100, 1, -1):
    alpha = i/10000
    confidence = 1 - alpha/2
    z_critical = stats.norm.ppf(q=confidence)

    if z_score > z_critical:
        print(f'{alpha} - H0 should be rejected ({z_score} > {z_critical})')
    else:
        print(f'{alpha} - H0 should be accepted ({z_score} <= {z_critical})')

print(ztest(speed_test, value=speed_h0))
```

## Propozycja rozwiązania

```
import matplotlib.pyplot as plt
import statistics
from statsmodels.stats.weightstats import ztest
import scipy.stats as stats
import pandas as pd
import math

# H0 - mean speed of typing is <= 40 wpm
# H1 - mean speed of typing is > 40 wpm

speed_h0 = 40

speed_test = [41, 42, 34, 37, 40, 15, 46, 27, 35, 55, 52, 64, 51, 57, 49, 38, 76, 45,
65, 57, 68, 79, 44, 43, 23, 58, 53, 56, 15, 32, 37, 54, 43, 47, 55, 38, 50, 48, 55,
71, 78, 46, 54, 36, 44, 40, 31, 13, 55, 45, 40, 21, 27, 72, 54, 15, 30, 24, 20, 63,
18, 45, 48, 36, 42, 52, 55, 34, 47, 47, 11, 39, 47, 59, 52, 33, 36, 38, 63, 47, 40,
43, 20, 65, 43, 61, 29, 64, 64, 46, 53, 59, 34, 47, 41, 24, 51, 54, 60, 39]

speed_test_len = len(speed_test)
speed_test_mean = statistics.mean(speed_test)
speed_test_stdev = statistics.stdev(speed_test)

z_score = (speed_test_mean - speed_h0)/(speed_test_stdev/math.sqrt(speed_test_len))

for i in range(100, 1, -1):
    alpha = i/10000
    confidence = 1 - alpha
    z_critical = stats.norm.ppf(q=confidence)

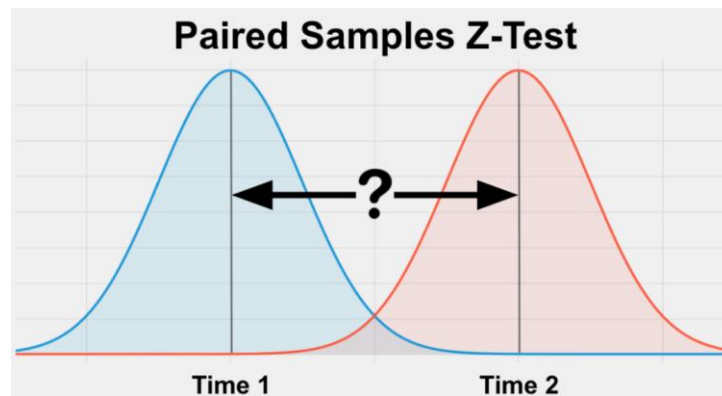
    if z_score > z_critical:
        print(f'{alpha} - H0 should be rejected ({z_score} > {z_critical})')
    else:
        print(f'{alpha} - H0 should be accepted ({z_score} <= {z_critical})')

print(ztest(speed_test, value=speed_h0, alternative='larger'))
```

## Testowanie hipotezy dla zbiorów zależnych

### Notatka

Zbiory zależne to zazwyczaj dwa takie same zbiory, dla których zebrano pewne dane w różnych punktach czasowych, np. parametr organizmu przed podjęciem leczenia i po jego zakończeniu nowym lekiem. Zakładając, że rozkład parametru dla populacji zarówno przed, jak i po jest rozkładem normalnym, można spodziewać się, że wartości średnie, jak i same wykresy rozkładu będą przesunięte:



Podobnie, jak w przypadku poprzednio omawianych przypadków, testowanie rozpoczynamy od zdefiniowania hipotezy  $H_1$  (to nowa przełomowa hipoteza, którą chcemy udowodnić) oraz  $H_0$  – status quo (zaprzeczenie tej nowej hipotezy).

Dzięki temu, że analizowany jest ten sam zbiór (lub zbiór zależny) można wyznaczyć różnice pomiarów dla poszczególnych obiektów i sprawdzać, czy tak powstała próba oscyluje w okolicach wartości podawanych w hipotezie. Wartość T-score można wyznaczyć wzorem:

$$T = \frac{\bar{d} - \mu_0}{s/\sqrt{n}}$$

$\bar{d}$  – średnia różnic między odpowiednimi elementami próby „przed” i „po”

$\mu_0$  – średnia zakładana przez hipotezę  $H_0$

$s$  – odchylenie standardowe z próby

$n$  – liczebność próby

Hipotezę  $H_0$  obalamy jeśli  $T\text{-score} > t\text{-critical}$ . Z rozkładu T-studenta korzystamy jeśli nie jest znana wariancja/odchylenie standardowe dla populacji i musimy korzystać z wariancji/odchylenia standardowego dla próby, oraz gdy próba jest mała (co w statystyce oznacza liczebność próby  $< 30$ ). Wartość t-critical lub z-critical. W zależności od tego, czy hipotezy sprawdzały warunki  $=$  i  $<$ , czy  $<$  i  $>=$  należy użyć wartości  $\alpha$  dla testu dwustronnego ( $\alpha$ ) lub jednostronnego ( $\alpha/2$ ).

Kod Python implementujący obliczenia dla zbiorów zależnych:

```
import scipy.stats as stats
import pandas as pd
import math

# H1: mean before < mean after
# H0: mean before >= mean after
```

```

size = 30
results_before = stats.norm.rvs(size=size, loc=14, scale=1)
change = stats.norm.rvs(size=size, loc=0.2, scale=0.5)
results_after = [x+y for (x,y) in zip(results_before, change)]

df = pd.DataFrame(list(zip(results_before, results_after)), columns=['before',
'after'])
df['diff'] = df['after'] - df['before']

alpha = 0.05
sample_size = len(df)
sample_mean = df['diff'].mean()
sample_stdev = df['diff'].std()
confidence_level = 1-alpha

t_critical = stats.t.ppf(q=confidence_level, df=sample_size-1) #1.699
t_score = (sample_mean - 0)/sample_stdev/math.sqrt(sample_size)
print(f't-critical = {t_critical}, T-score = {t_score}')

if t_critical < t_score:
    print('we reject H0')
else:
    print('we accept H0')

```

Potwierdzenie/obalenie hipotezy można również wykonać w oparciu o p-value. Jeśli wartość p-value jest większa lub równa poziomowi istotności, to hipotezę  $H_0$  odrzucamy, w przeciwnym razie przyjmujemy.

Wartość p-value można wyznaczyć gotową funkcją:

```

from statsmodels.stats.weightstats import ztest
test_stat, p_value = ztest(df["before"], df["after"], value=0, alternative='larger')
print(f'Z-test (p-value) = { p_value }')

if p_value < alpha:
    print('H0 is rejected')
else:
    print('H0 is accepted')

```

## Lab

- 1) Pewna linia lotnicza podsumowała sezon wakacyjny publikując zestawienie na temat zrealizowanych lotów, liczby pasażerów i opóźnień. Linia lotnicza nie miała zbyt dobrych opinii wśród pasażerów, dlatego wszystkich bardzo zdziwiła informacja, że średnie opóźnienie rok do roku wzrosło tylko o 5 minut. Jako statystyk, zbierasz informacje o 30 trasach, wyznaczasz średni czas lotów i zapisujesz dane w postaci list:

```

year_1 = [51.54147388, 66.37933735, 86.06956843, 61.21211192, 70.97453697, 67.818349
, 74.61029388, 64.15260126, 64.04404262, 73.20343147, 84.00388888, 71.76197586
, 81.92905628, 59.79383455, 69.76258392, 67.26433896, 76.15317597, 74.63963851
, 69.54745953, 81.37239488, 66.75573044, 65.68174176, 72.39491011, 94.49928345
, 50.02791987, 85.30256927, 73.19244005, 68.75115665, 70.17895383, 72.74099816]

year_2 = [87.10363669, 75.94465019, 63.60003755, 67.54621323, 78.9299618, 85.09730193
, 76.9517496, 75.36388527, 80.60139972, 80.61204067, 86.94199771, 65.38143752
, 84.08922726, 94.28436019, 77.85826007, 73.84883098, 69.77960042, 78.94087383
, 83.0521829, 56.88821237, 79.48768691, 82.4386409, 82.94123142, 89.38153559
, 65.55075576, 75.03079448, 78.38170544, 57.94891699, 82.68846853, 88.63010176]

```

- 2) Zdefiniuj hipotezę zerową i alternatywną jednostronną dotyczącą opóźnień > 5 minut.
- 3) Wykonaj test z poziomem istotności 10%, który potwierdzi lub obali informacje publikowane przez linię lotniczą. Możesz to zrobić samodzielnie lub posłużyć się następującymi krokami:

- a) Zapisz listy w DataFrame df w kolumnach y1 i y2 i wyznacz różnicę y2-y1 w kolumnie diff
  - b) Zdefiniuj  $\alpha=0.10$ , wyznacz rozmiar próby, średnią i odchylenie standardowe oraz confidence\_level wskazujący na położenie punktu krytycznego
  - c) Wyznacz punkt krytyczny wg rozkładu T oraz T-score
  - d) Wyświetl informacje o odrzuceniu hipotezy zerowej lub jej akceptacji.
- 4) Ponieważ hipoteza została zaakceptowana, chcesz sprawdzić przy jakim poziomie istotności można by obalać hipotezę  $H_0$ . Dlatego wyznacz p-value i dodaj komunikat mówiący o odrzucaniu/akceptacji hipotezy w zależności od aktualnie wybranego poziomu istotności  $\alpha$ .

### Propozycja odpowiedzi

```
import pandas as pd
import scipy.stats as stats
import math
from statsmodels.stats.weightstats import ztest

year_1 = [51.54147388, 66.37933735, 86.06956843, 61.21211192, 70.97453697, 67.818349
, 74.61029388, 64.15260126, 64.04404262, 73.20343147, 84.00388888, 71.76197586
, 81.92905628, 59.79383455, 69.76258392, 67.26433896, 76.15317597, 74.63963851
, 69.54745953, 81.37239488, 66.75573044, 65.68174176, 72.39491011, 94.49928345
, 50.02791987, 85.30256927, 73.19244005, 68.75115665, 70.17895383, 72.74099816]

year_2 = [87.10363669, 75.94465019, 63.60003755, 67.54621323, 78.9299618, 85.09730193
, 76.9517496, 75.36388527, 80.60139972, 80.61204067, 86.94199771, 65.38143752
, 84.08922726, 94.28436019, 77.85826007, 73.84883098, 69.77960042, 78.94087383
, 83.0521829, 56.88821237, 79.48768691, 82.4386409, 82.94123142, 89.38153559
, 65.55075576, 75.03079448, 78.38170544, 57.94891699, 82.68846853, 88.63010176]

# H0 - Mean delay is <= 5
# H1 - Mean delay is > 5

h0_difference = 5
df = pd.DataFrame(list(zip(year_1, year_2)), columns=['y1', 'y2'])
df['diff'] = df['y2'] - df['y1']

alpha = 0.10
sample_size = len(df)
sample_mean = df['diff'].mean()
sample_stdev = df['diff'].std()
confidence_level = 1-alpha

t_critical = stats.t.ppf(q=confidence_level, df=sample_size-1) #1.699
print(f'T-critical = {t_critical}')

t_score = (sample_mean - h0_difference)/(sample_stdev/math.sqrt(sample_size))
print(f'T-score = {t_score}')

if t_critical < t_score:
    print('H0 is rejected')
else:
    print('H0 is accepted')

test_stat, p_value = ztest(df["y2"], df["y1"], value=h0_difference,
alternative='larger')
print(f'Z-test (p-value) = { p_value }')

if p_value < alpha:
    print('H0 is rejected')
else:
    print('H0 is accepted')
```



## Testowanie hipotezy dla zbiorów niezależnych

### Notatka

Potwierdzanie lub odrzucanie hipotez można też wykonywać względem zbiorów niezależnych. Wzory służące do wyznaczenia odpowiednich wartości wykorzystują podobną logikę, jak wzory wykorzystywane przy wyznaczaniu przedziału pewności.

Pierwszy przypadek to hipotezy dotyczące dwóch zbiorów niezależnych, dla których jest znana wariancja:

$$Z = \frac{\bar{x} - \mu_0}{\sqrt{\frac{\sigma_x^2}{n_x} + \frac{\sigma_y^2}{n_y}}}$$

$\bar{x}$  – średnia różnic między wartościami średnimi obu zbiorów

$\mu_0$  – średnia zakładana przez hipotezę  $H_0$

$\sigma_x^2$   $\sigma_y^2$  – odchylenie standardowe z próby X i Y

$n_x$   $n_y$  – liczebność próby X i Y

Jak zwykle, jedną z istotniejszych rzeczy jest dobre zdefiniowanie hipotez. Jeśli w hipotezach testujemy warunki == lub <>, to odszukując wartości krytycznej powinniśmy od 1 odejmować  $\alpha/2$ , z kolei jeśli hipotezy pracują na nierównościach <, <=, >, >= to od 1 odejmujemy  $\alpha$ .

Kod pozwalający zweryfikować hipotezę w oparciu o próbę może wyglądać tak:

```
import scipy.stats as stats
import statistics
import pandas as pd
import math
import random

# Two independent values and variance is known (sometimes "well known")
# H0: mean therapists - mean doctors <= 0
# H1: mean therapist - mean doctors > 0

doctors = stats.norm.rvs(size=70, loc=65, scale=10)
therapists = stats.norm.rvs(size=40, loc=75, scale=10)

mean_doc = statistics.mean(doctors)
mean_ther = statistics.mean(therapists)
st_dev_doc = statistics.stdev(doctors)
st_dev_ther = statistics.stdev(therapists)
count_doc = len(doctors)
count_ther = len(therapists)

alpha = 0.05
confidence_level = 1-alpha
z_critical = stats.norm.ppf(q=1-alpha)
z_score = ((mean_ther - mean_doc) - 0) / math.sqrt(st_dev_ther**2/count_ther +
st_dev_doc**2/count_doc)

if z_critical < z_score :
    print('we reject H0')
else:
    print('we accept H0')
```

Czynność tę można też wykonać w oparciu o p-value:

```
from statsmodels.stats.weightstats import ztest
z_stat, p_value = ztest(therapists, doctors, alternative='larger')

if p_value < alpha:
    print('we reject H0')
else:
    print('we accept H0')
```

Drugi przypadek to weryfikacja hipotezy dla zbiorów niezależnych, dla których nie znamy wariancji, ale zakładamy, że jest ona taka sama. Do wyznaczenia wartości krytycznej należy korzystać z rozkładu T-Studenta. Wzór na T-score to:

$$T = \frac{\bar{x} - \mu_0}{\sqrt{\frac{\sigma_p^2}{n_x} + \frac{\sigma_p^2}{n_y}}}$$

gdzie:

$$s_p^2 = \frac{(n_x - 1)s_x^2 + (n_y - 1)s_y^2}{n_x + n_y - 2}$$

$\bar{x}$  - średnia różnic między wartościami średnimi obu zbiorów

$\mu_0$  - średnia zakładana przez hipotezę  $H_0$

$s_x^2$   $s_y^2$  - odchylenie standardowe z próby X i Y

$n_x$   $n_y$  - liczebność próby X i Y

Pythonowy skrypt potwierdzający lub odrzucający hipotezę może wyglądać tak:

```
# Two independent values and variance is unknown and assumed to be equal
# H0: mean therapists - mean doctors <= 0
# H1: mean therapist - mean doctors > 0

alpha = 0.05
confidence_level = 1-alpha
t_critical = stats.t.ppf(q=1-alpha, df=count_doc+count_ther-2)
print(f'T-critical = {t_critical}') #1.6590851435825054
var_p = ((count_doc-1)*st_dev_doc**2 + (count_ther-1)*st_dev_ther**2)/(count_doc+count_ther-2)
t_score = ((mean_ther - mean_doc) - 0) / math.sqrt(var_p/count_doc + var_p/count_ther)

if t_critical < t_score:
    print('we reject H0')
else:
    print('we accept H0')
```

Wersja skryptu korzystająca z p-value:

```
from statsmodels.stats.weightstats import ttest_ind
t_stat, p_value, df = ttest_ind(therapists, doctors, alternative='larger')

if p_value < alpha:
    print('we reject H0')
else:
    print('we accept H0')
```

## Lab

- 1) Restauratorzy z Rzymu i Neapolu chcą w końcu rozstrzygnąć spór o to, kto ma tańszą pizzę. Patrząc w dane z wylosowanej próby masz wrażenie, że pizza w Rzymie jest tańsza, dlatego formułujesz hipotezę  $H_1$  – „średnia cena pizzy w Rzymie jest wyższa niż w Neapolu”. Jaka będzie hipoteza  $H_0$ ?
- 2) Wygeneruj fikcyjne dane korzystając z:

```
import scipy.stats as stats  
  
pizza_Naples = stats.norm.rvs(size=20, loc=8, scale=1)  
pizza_Rome = stats.norm.rvs(size=15, loc=12, scale=5)
```

- 3) Zakładając, że wariancja populacji jest znana i równa wariancji próby, przetestuj hipotezę korzystając z porównania wartości krytycznej z oraz z z-score.
- 4) Przetestuj hipotezę korzystając z p-value.
- 5) Wykonaj powyższe testy przy założeniu, że wariancja nie jest znana, ale zakładamy, że dla Rzymu i Neapolu jest taka sama.

```
import scipy.stats as stats
import statistics
import pandas as pd
import math
import random

# Two independent values and variance is known (sometimes "well known")
# H0: mean pizza Rome - mean pizza Naples <= 0
# H1: mean pizza Rome - mean pizza Naples > 0

pizza_Naples = stats.norm.rvs(size=20, loc=8, scale=1)
pizza_Rome = stats.norm.rvs(size=15, loc=12, scale=5)

mean_Naples = statistics.mean(pizza_Naples)
mean_Rome = statistics.mean(pizza_Rome)
st_dev_Naples = statistics.stdev(pizza_Naples)
st_dev_Rome = statistics.stdev(pizza_Rome)
count_Naples = len(pizza_Naples)
count_Rome = len(pizza_Rome)

alpha = 0.05
confidence_level = 1-alpha
z_critical = stats.norm.ppf(q=1-alpha)
print(f'Z-critical = {z_critical}')
z_score = ((mean_Rome - mean_Naples) - 0) / math.sqrt(st_dev_Rome**2/count_Rome +
st_dev_Naples**2/count_Naples)
print(f'Z-score = {z_score}')

if z_critical < z_score :
    print('We reject H0')
else:
    print('We accept H0')

from statsmodels.stats.weightstats import ztest
z_stat, p_value = ztest(pizza_Rome, pizza_Naples, alternative='larger')
print(f'Z-test (p-value) = { p_value }')

if p_value < alpha:
    print('We reject H0')
else:
    print('We accept H0')

print('-'*60)
# Two independent values and variance is unknown and assumed to be equal
# H0: mean pizza Rome - mean pizza Naples <= 0
# H1: mean pizza Rome - mean pizza Naples > 0

alpha = 0.05
confidence_level = 1-alpha
t_critical = stats.t.ppf(q=1-alpha, df=count_Rome+count_Naples-2)
print(f'T-critical = {t_critical}')
var_p = ((count_Rome-1)*st_dev_Rome**2 +(count_Naples-
1)*st_dev_Naples**2)/(count_Rome+count_Naples-2)
t_score = ((mean_Rome - mean_Naples) - 0) / math.sqrt(var_p/count_Rome +
var_p/count_Naples)
print(f'T-score = {t_score}')

if t_critical < t_score:
    print('We reject H0')
else:
    print('We accept H0')
```

```
from statsmodels.stats.weightstats import ttest_ind
t_stat, p_value, df = ttest_ind(pizza_Rome, pizza_Naples, alternative='larger')
print(f'T-test (p-value) = { p_value }')

if p_value < alpha:
    print('We reject H0')
else:
    print('We accept H0')
```

Spróbuj też!

