

Wykład 2. Wprowadzanie do NumPy i pandas. Wczytywanie i wstępna eksploracja danych

Michał Sochański

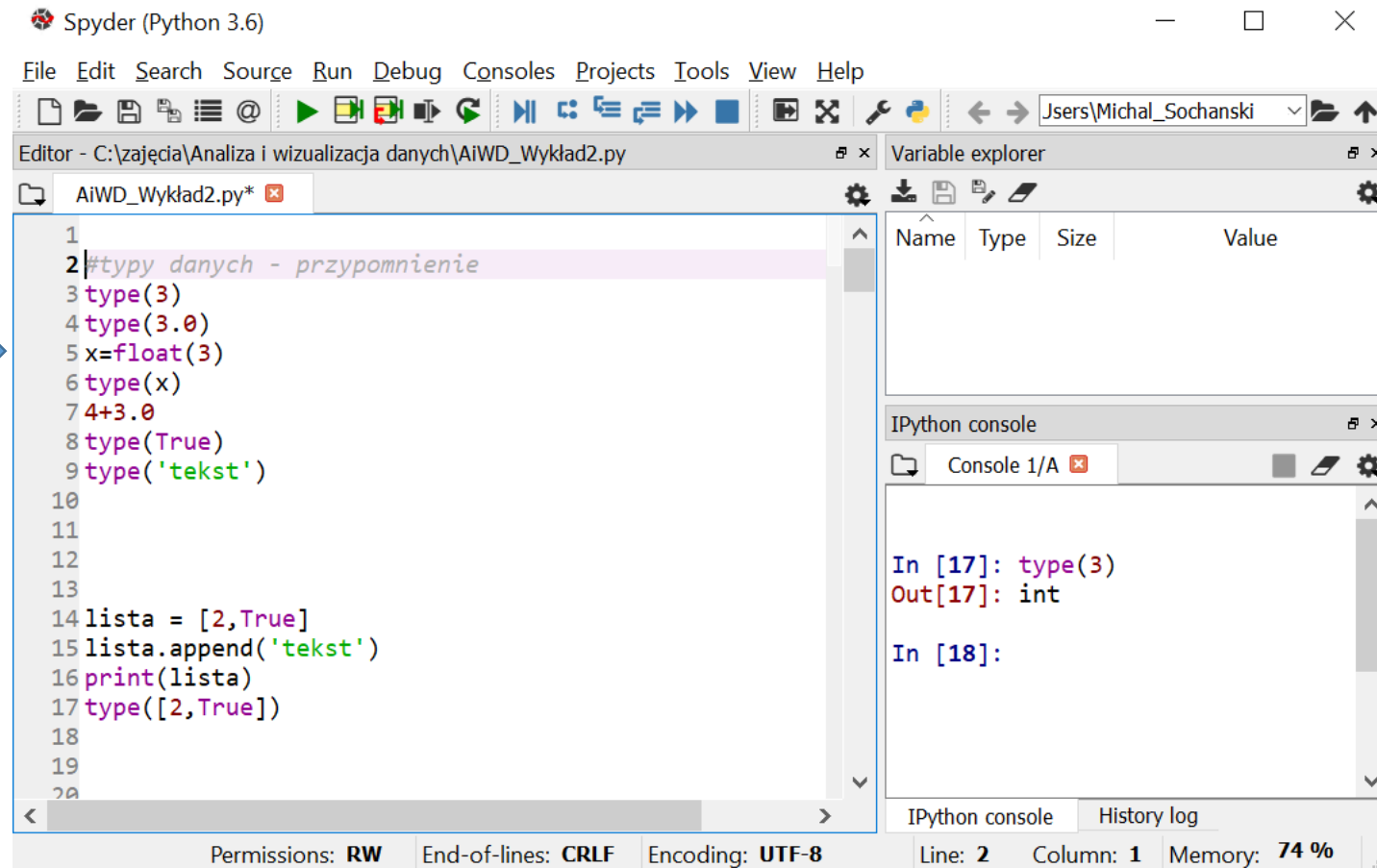
Plan wykładu

1. Wstępne uwagi o Spyderze, Pythonie i używanych pakietach
2. Przypomnienie – listy i typy zmiennych w Pythonie
3. Podstawowe informacje o pakiecie NumPy
4. Podstawowe informacje o pakiecie pandas
5. Wczytywanie danych jako obiektu DataFrame
6. Zapoznanie się ze strukturą zbioru danych
7. Podzbiory obiektu DataFrame

1. Wstępne uwagi o Spyderze, Pythonie i pakietach

- Będziemy korzystać z Pythona 3.XX
- www.python.org
- Anaconda – darmowa dystrybucja Pythona
 - Zawiera dużo zainstalowanych pakietów
- Spyder – IDE (Integrated Development Environment)
 - dostępny w ramach Anacondy
 - <https://www.spyder-ide.org/>

Spyder



Pakiety

- `import` pakiet `as` `pt`
- Niektóre pakiety składają się z modułów, albo „podpakietów”
- NumPy
- Pandas
- Scipy – zbiór pakietów zawierających narzędzia obliczeniowe używane w nauce (scientific computing)
 - `scipy.stats` – pakiet dedykowany statystyce
 - `scipy.linalg` – pakiet dedykowany algebrze liniowej
- Matplotlib – najbardziej popularna biblioteka do tworzenia wizualizacji
- <https://docs.python.org/3.6/tutorial/modules.html>
- `pt.__version__`

2. Przypomnienie – listy i typy zmiennych w Pythonie

- Podstawowe typy zmiennych w Pythonie:

- int
- float
- complex
- str
- bool
- list

} **liczbowe**

- Można przekonać się, jakiego typu jest dany obiekt za pomocą funkcji „type”:
 - type(3)
 - type(True)
 - type('tekst')
 - type([2,True])

Przypomnienie – listy i typy zmiennych w Pythonie

- Podstawowym typem danych w Pythonie jest lista. Może ona się składać z obiektów dowolnego typu, rozdzielanych przecinkami. Lista zamknięta jest w nawias kwadratowy.
- Przykłady list:
 - `[2,True]`
 - `['tekst', 3, 5, 'abc']`
 - `[['tekst1', 3], 5, [False, 2, 0]]`

Przypomnienie – listy i typy zmiennych w Pythonie

- Każdemu typowi obiektu odpowiadają konkretne „metody”, czyli operacje, które można na nim wykonywać. Na przykład dla list może to być „append”:

```
lista = [2,True]
lista.append('tekst')
print(lista)
[2, True, 'tekst']
```


NumPy – typy danych

- Typ „`numpy.ndarray`” różni się od list tym, że może zawierać dane **tylko jednego typu**.
- Każdy typ ma przypisane sobie specyficzne „metody”, czyli operacje, które można na nim wykonywać. Na obiektach typu *ndarray* można w szczególności wykonywać wiele operacji matematycznych, których nie można wykonywać na listach.
- NumPy wprowadza dużo nowych typów zmiennych, w szczególności – liczbowych. Oto niektóre z nich:
 - `int16`, `int32`, `int64`
 - `float16`, `float32`, `float64`

NumPy – tworzenie nowych obiektów

- W pakiecie NumPy wbudowane są przydatne funkcje do szybkiego tworzenia nowych obiektów o regularnych kształtach

```
np.zeros(5, dtype=int)
```

```
array([0, 0, 0, 0, 0])
```

```
np.ones(3)
```

```
array([1., 1., 1.])
```

```
np.linspace(0,1,5)
```

```
array([0. , 0.25, 0.5 , 0.75, 1.  ])
```

```
np.arange(10,20)
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19]) #nie wliczamy 20!
```

```
np.random.rand(5)
```

```
array([0.71672723, 0.24117782, 0.43315446, 0.85426802, 0.76590972])
```

```
#liczby losowe z przedziału (0,1)
```

NumPy – dalsze uwagi

- Ustalenie typu danych obiektu *ndarray*

- `np.array([1, 2, 3, 4], dtype='float32')`

- Odwołania do elementów *ndarray*

- `x=np.array([1,2,3])`

- `x[0]`

- 1

- Bardzo istotne jest to, że na obiektach *ndarray* (w przeciwieństwie do list) możemy wykonywać następujące operacje:

- `[1,2,3] + [3,5,9]`

- `[4,7,12]` #sumujemy elementy wektorów o odpowiednich indeksach

- `[4,17,3,2]*3`

- `[12,51,9,6]` #każdy element wektora mnożymy przez tą samą liczbę

NumPy – ogólne uwagi o macierzach

- Obiektami typu *ndarray* są też macierze. Można je tworzyć np. w następujący sposób:

```
matrix = np.array([[7,2],[4,6]])
```

- Odwoływanie się do konkretnego elementu macierzy:

```
matrix[0,1]  #pierwszy rząd, druga kolumna
```

- Możemy też przekształcić 1-wymiarowy *ndarray* (wektor) na 2-wymiarowy *ndarray* (macierz)

```
wektor = np.array([1,2,3,7,8,10])
```

```
macierz2 = wektor.reshape(2,3)
```

4. Podstawowe informacje o pakiecie pandas. Obiekt DataFrame

- Najważniejszym typem danych dostępnym w pakiecie pandas jest DataFrame. Jest to tabelaryczna struktura danych, w której w każdej kolumnie zawarte są dane jednego typu.
- Można o obiektach DataFrame myśleć jako o liście list odpowiadających kolumnom. Dokładniej, **każda kolumna jest listą typu `numpy.ndarray`** zawierającą dane jednego typu.
- Obiekty DataFrame w sposób naturalny reprezentują dane, w których wiersze odpowiadają obserwacjom a kolumny odpowiadają własnościom (zwanym też często zmiennymi).
- `import pandas as pd`

Obiekt DataFrame

- Własności obiektu DataFrame:
 - Każda kolumna musi mieć przypisany jeden typ danych.
 - Każda kolumna ma swoją etykietę.
 - Wiersze również muszą mieć etykiety, zwane indeksami.
 - Indeksy mogą z zasady być dowolnymi obiektami, domyślnie są jednak kolejnymi liczbami naturalnymi (zaczynając od 0)
 - DataFrame jako typ struktury danych jest „zbudowany” na strukturach NumPy: każda kolumna jest tak naprawdę obiektem typu `numpy.ndarray`
 - Każda kolumna jest obiektem typu **pd.Series** (można powiedzieć, że jest `ndarray`, który jest dodatkowo wyposażony w indeks)
 - Cały DataFrame jest z kolei obiektem typu **pd.DataFrame**

Pandas – wczytujemy pakiet i tworzymy pierwszy DataFrame

- Stwórzmy swój pierwszy DataFrame. Można to zrobić za pomocą tzw. słowników (*dictionaries*), oraz specjalnej funkcji z pakietu pandas – `pd.DataFrame` (zwróćmy uwagę, że poprzedzamy ją przez tekst „pd.” – zgodnie z aliasem, który uzgodniliśmy przy wczytywaniu pakietu).

```
df = pd.DataFrame({  
    'ColA': [1, 2, 3],  
    'ColB': [4, 5, 6],  
    'ColC': ['a', 'b', 'c']})
```

Uwaga - 'ColA' to nazwa, czy etykieta, kolumny. Indeksy są w tym przypadku automatycznie nadane jako liczby naturalne (licząc od 0).

Pandas – wczytujemy pakiet i tworzymy pierwszy DataFrame

- Tym razem stworzymy DataFrame z obiektów *ndarray*. „Index” określa etykiety wierszy.

```
df = pd.DataFrame(np.array([5,8,3]), index=[2,3,5])
```

```
df2 = pd.DataFrame(np.array([[1,2],[3,4]]), index=['x', 'y'])
```

```
print(df2)
```

```
In [165]: print(df2)
```

	0	1
x	1	2
y	3	4

5. Wczytywanie danych jako obiektu DataFrame. Wstępne uwagi

- Będziemy się uczyć wczytywać pliki tekstowe – o rozszerzeniu „.csv” lub „.txt”, a także pliki Excelowe – „.xlsx”
- Pierwszy krok to ustalenie katalogu roboczego. Możemy to zrobić za pomocą funkcji z pakietu „os”.

```
import os
```

```
os.getcwd()          #wyświetlamy aktualny katalog roboczy
```

```
os.listdir()         #sprawdzamy jego zawartość
```

```
os.chdir('C:\\Users\Folder') #zmieniamy katalog na docelowy
```

- Na koniec warto przypisać ścieżkę do zmiennej, np.:

```
plik = 'nazwa_pliku.txt'
```

Wczytywanie danych z plików tekstowych

- Pliki tekstowe – o rozszerzeniu „.csv” oraz „.txt” wczytujemy za pomocą następującej funkcji z pakietu „pandas”:

```
data = pd.read_csv(plik)
```

Dodatkowe opcje funkcji „pd.read_csv”:

1. Separator

Jeśli separatorem nie jest przecinek, dołączamy argument sep, np:

- sep=';'
- sep=' ' #spacja
- sep='/t'

Np.

```
pd.read_csv(plik, sep=' ')
```

Dalsze opcje funkcji `pd.read_csv`

2. Nagłówek

Pierwsza linia zostanie automatycznie wczytana jako nagłówek. Jeśli plik nie ma nagłówka, dodajemy argument „`header=None`”:

➤ `pd.read_csv(plik, header=None)`

Zostaną wtedy nadane nagłówki „numeryczne”. Jeśli chcemy od razu nadać własne nagłówki, możemy to zrobić za pomocą argumentu „`names`”:

➤ `pd.read_csv(plik, header=None, names=['nazwa1', 'nazwa2'])`

Możemy też zacząć wczytywać od np. drugiej linii i potraktować ją jako nagłówek, wpisując:

➤ `header=1` #Uwaga – w Pythonie liczymy od 0!

Dalsze opcje funkcji `pd.read_csv`

3. Wczytujemy tylko określoną ilość rzędów – argument „`nrows`”

➤ `pd.read_csv(plik, nrows=3)`

Uwaga – jest jeszcze wiele argumentów, które można dodać w funkcji `pd.read_csv`. Można o nich poczytać chociażby za pomocą komendy:

`help(pd.read_csv)`

O niektórych z nich wspomnimy w dalszej części wykładu.

Wczytywanie plików „.xlsx”

- Aby wczytać dane z pliku Excelowego, najpierw musimy przypisać do zmiennej odpowiedni plik:

```
temp = pd.ExcelFile(plik)
```

- Możemy teraz zobaczyć nazwy arkuszy występujących w pliku:

```
print(temp.sheet_names)
```

- Aby załadować konkretny arkusz jak DataFrame korzystamy z funkcji „parse”

```
data1 = temp.parse('sheetname1')    #odwołujemy się do nazwy arkusza
```

```
data2 = temp.parse(0)                #odwołujemy się do numeru arkusza
```

```
data3 = temp.parse(1, usecols=[0], skiprows=[0], names=['Country'])
```

#dodatkowe opcje: wybieramy kolumny („usecols”), opuszczamy odpowiednie rzędy („skiprows”) i nadajemy nazwę wczytamy kolumnom („names”). Uwaga – wszystkie wartości musimy tutaj wpisywać w listach.

Zapis plików

- `data.to_csv('file.csv')`
- `data.to_excel('file.xlsx')`

6. Zapoznanie się ze strukturą zbioru

- Do ogólnego zapoznania się ze strukturą zbioru służą następujące atrybuty i metody:

➤ <code>data.info()</code>	#przegląd struktury zbioru
➤ <code>data.shape</code>	#wymiary zbioru
➤ <code>data.columns</code>	#nazwy kolumn
➤ <code>data.index</code>	#etykiety wierszy, czyli tzw. indeksy
➤ <code>data.head(<i>n</i>)</code>	#wyświetla pierwszych <i>n</i> rzędów zbioru (domyślnie 5)
➤ <code>data.tail(<i>n</i>)</code>	#wyświetla ostatnich <i>n</i> rzędów zbioru (domyślnie 5)

Typy zmiennych – wstępne uwagi

- Każda zmienna w DataFrame to obiekt typu „Series” (który – przypomnijmy - można rozumieć jako „ndarray” wyposażony w etykiety).
- Typologia obiektów „Series” (czyli rodzaje zmiennych) różni się nieco od typów „ndarrays”:
 - Zmienne tekstowe (w podstawowej wersji Pythona typ „str”) przechowywane są zazwyczaj jako typ „object”.
 - Typy liczbowe są analogiczne jak w NumPy – int32, int64, float32, itd...
- Ważniejsze nowe typy zmiennych:
 - Typ „datetime64” do przechowywania dat
 - Typ „Categorical” do przechowywania (i sprawnego operowania) zmiennych zawierających kategorie.
- Braki danych oznaczone są jako „NaN”.

Wczytywanie zmiennych jako daty

- Bardzo często zdarza się, że chcemy daną zmienną wczytać jako datę, ale Python automatycznie narzuca inny typ danych (najczęściej „object”).
- Jeśli chcemy korzystać ze specjalnych funkcji dedykowanych datom czy np. tworzyć wykresy dla szeregów czasowych, musimy zadbać o to, aby odpowiednie zmienne były wczytane jako obiekt „datetime64”.
- Funkcja „pd.read_csv” może wczytać zmienne tekstowe jako daty. Domyślnie wczytywane są przy tym tylko daty o formacie: „yyyy-mm-dd hh:mm:ss”. Można to zrobić:
 - dodając argument „parse_dates=True”. Wtedy każda zmienna, którą można zmienić na datę, zostanie zmieniona.
 - dodając argument „parse_dates=['Date]”. Wtedy zmienimy tylko zmienną o nazwie „Date”.

Przykład: `data = pd.read_csv(file, parse_dates=True)`

Bardziej szczegółowa informacja o zmiennych numerycznych

- Jeśli chcemy przyjrzeć się nieco bliżej strukturze zmiennych numerycznych występujących w zbiorze, możemy to zrobić za pomocą funkcji „describe”. Wyświetla ona podstawowe statystyki, jak średnia, maksimum, itd.
- Po wpisaniu poniższej komendy statystyki pojawią się dla wszystkich zmiennych numerycznych w zbiorze):

```
data.describe()
```

- Aby wyświetlić je tylko dla jednej zmiennej, np. o etykiecie „zmienna”, można zastosować komendę:

```
data.zmienna.describe()
```

Bardziej szczegółowa informacja o zmiennych typu „object”

- Aby zapoznać się z częstościami poszczególnych kategorii w zmiennych typu „object” (o etykiecie „zmienne”), możemy zastosować następującą funkcję:

```
data.zmienna.value_counts(dropna=False)
```

Argument `dropna` jest opcjonalny. Jeśli oznaczymy go jako `False`, braki danych pokażą się nam jako osobna kategoria. Często jest to wartościowa informacja.

7. Podzbiory obiektu DataFrame (subsetting)

Zapoznając się ze strukturą zbioru danych, lub prowadząc już dalsze analizy, bardzo często chcemy rozważać podzbiory naszego DataFrame'u.

Poznamy trzy sposoby generowania podzbiorów:

- 1) Generowanie podzbiorów kolumn przez odwołanie do ich etykiet
- 2) Generowanie podzbiorów z użyciem funkcji „loc” i „iloc”
- 3) Generowanie podzbiorów rzędów przez określenie warunków, które muszą spełniać

7.1. Generowanie podzbiorów kolumn przez odwołanie do ich etykiet.

1. Wyświetlamy konkretną zmienną (korzystając z jej etykiety):

```
data['Zmienna']
```

```
data.Zmienna
```

Możemy też przypisać zmienną do nowego obiektu np. za pomocą komendy:

```
var = data.Zmienna
```

W tym przypadku będzie ona typu „Series”.

7.1. Generowanie podzbiorów kolumn przez odwołanie do ich etykiet.

2. Wyświetlamy jedną lub więcej zmiennych jako nowy DataFrame

```
data[['Zmienna']]
```

```
data[['Zmienna1', 'Zmienna2']]
```

W tym przypadku obiekt stworzony za pomocą poniższej komendy będzie typu „DataFrame”:

```
data_subset = data[['Zmienna1', 'Zmienna2']]
```

7.2. Generowanie podzbiorów z użyciem funkcji „loc” i „iloc”

1. Aby „wybrać” odpowiednie rzędy i kolumny korzystając z ich numerycznych indeksów, korzystamy z metody „.iloc”

`data.iloc[0]` #tu wyświetlamy tylko pierwszy wiersz

`data.iloc[1,2]` #przecinek rozdziela odwołanie do wierszy (po lewej) i
kolumn (po prawej)

`data.iloc[2:4,1]`

`data.iloc[:5,4]`

`data.iloc[-5:,3]`

`data.iloc[2:,[1,3,5]]`

`data.iloc[:,1]`

7.2. Generowanie podzbiorów z użyciem funkcji „loc” i „iloc”

2. Aby „wybrać” odpowiednie rzędy i kolumny korzystając z etykiet kolumn i wierszy, korzystamy z metody „.loc”

```
data.loc['index1', 'id']
```

```
data.loc[:, 'id':'Data urodzin']
```

```
data.loc[1:3, 'wiek']
```

```
data.loc[5, :'wiek']
```

Uwaga – w przypadku funkcji „loc” i „iloc” podzbiór będzie obiektem typu „DataFrame”, „Series” albo np. „int32” w zależności od tego, czy podzbiór będzie, 2-wymiarowy, 1-wymiarowy bądź też będzie pojedynczym rekordem.

7.3. Generowanie podzbiorów rzędów przez określenie warunków, które muszą spełniać (filtrowanie)

Możemy również filtrować dane za pomocą dowolnie skonstruowanych warunków logicznych:

- Wpisując następującą komendę:

```
data.zmienna >3
```

tworzymy obiekt typu „Boolean series”, a więc wektor wartości True i False, które nadane są w zależności od tego czy odpowiedni element zmiennej o nazwie „zmienna” jest faktycznie większy niż 3.

- Możemy przypisać powyższy wektor do nowej zmiennej i użyć jako filtra.

```
filtr = data.zmienna>3
```

```
data[filtr] #wyświetlamy tylko te wiersze zbioru „filtr”, które spełniają  
            warunek określony przez zmienną „filtr”
```

7.3. Filtrowanie danych (cd)

- Możemy łączyć filtry za pomocą standardowych operatorów logicznych – oraz („&”), lub („|”), nie („~”). Na przykład:

```
data[(data.zmienna >=30) & (data.zmienna <50)]
```

```
data[(data.wiek>=18) | (data.liczba_dzieci>0)]
```

```
data[~(data.zmienna==3)]
```

- Można również wybrać podzbiór danej kolumny, korzystając z warunku zadanego na innej kolumnie:

```
data.zmienna1[data.zmienna2 >=30]
```

7.3. Filtrowanie danych – zmienne typu „object”

- Można też korzystać z warunków odwołujących się do zmiennych typu „object” (tekstowych). Podamy tu dwa przykłady funkcji:

`data.zmienna.isin(['tekst1', 'tekst2'])` #zwraca wektor logiczny wskazujący, które elementy kolumny „zmienna” są równe jednemu ze wskazanych tekstów (inaczej mówiąc, wektor logiczny mówi nam, dla których obserwacji ów warunek jest spełniony)

`data.zmienna.str.contains('tekst')` #zwraca wektor logiczny wskazujący, które elementy kolumny „zmienna” zawierają wskazany tekst

Sortowanie danych

- `dane = dane.sort_values(by = "wiek", ascending = False)`

Polecane kursy na platformie data.camp

- Introduction to *Python*
- Importing Data in *Python* (Part 1)
- pandas Foundations


Ćwiczenia

Ćwiczenie 1. (powtórka)

- Obiekt DataFrame możemy stworzyć za pomocą tzw. słowników w następujący sposób:

```
df = pd.DataFrame({  
    'Brand': ['VW', 'Audi', 'Dacia'],  
    'Price': [100, 140, 60]})
```

```
print(df)
```



	Brand	Price
0	VW	100
1	Audi	140
2	Dacia	60


- Można też najpierw stworzyć odpowiednie listy (w tym przypadku mogą to być albo „zwykłe” listy albo obiekty typu ndarray):

```
list1 = [2,4,5]
```

```
list2 = np.linspace(0,1,3) #to jest obiekt typu ndarray
```

```
df = pd.DataFrame({  
    'zmienna1': list1,  
    'zmienna2': list2})
```

```
print(df)
```



	zmienna1	zmienna2
0	2	0.0
1	4	0.5
2	5	1.0

Ćwiczenie 1.

1. Korzystając z powtórki z poprzedniego slajdu, stwórz DataFrame o 10 wierszach i następujących kolumnach:

- Liczby naturalne od 1 do 10 (nazwa: Kolumna1)
- Kolumna wypełniona 10 zerami (nazwa: Kolumna2)
- Kolumna wypełniona 10 liczbami rzeczywistymi od 0 do 7 (włącznie), takimi, że różnica pomiędzy liczbami w kolejnych wierszach jest stała (nazwa: Kolumna3)
- Kolumna wypełniona 10 losowymi liczbami z przedziału (0,5) (nazwa: Kolumna4)

2. Wyświetl:

- Wartość w trzecim rzędzie i drugiej kolumnie
- Całą drugą kolumnę
- Dwa pierwsze wiersze z trzeciej kolumny (funkcja „iloc”)

Ćwiczenie 1.

3. Przypisz do nowej zmiennej następujące dataframe'y:

- DF składający się z pierwszej i trzeciej kolumny wyjściowego zbioru
- DF składający się takich wierszy, dla których Kolumna1 jest większa od 4
- DF składający się takich wierszy, które spełniają warunek: Kolumna1 jest mniejsza od 5 lub Kolumna4 jest większa od 2
- DF składający się z kolumn nr. 1,2 oraz 4 oraz takich, że wszystkie wartości są równe co najwyżej 5

Ćwiczenie 2.

W folderze znajduje się plik o nazwie „Dane_z_Wyk2.txt” z danymi omawianymi na wykładzie.

1. Ustal folder roboczy za pomocą omawianych na wykładzie funkcji z pakietu „os” i umieść w nim pobrany plik.
2. Wczytaj plik za pomocą funkcji `pd.read_csv()`. Zwróć uwagę, że separatorem jest tutaj średnik. Przypisz dane do zmiennej o wybranej przez Ciebie nazwie.
3. Zapoznaj się ze strukturą zbioru za pomocą funkcji opisanych slajdzie 25.

Ćwiczenie 2. (cd)

4. Wyświetl następujące podzbiory badanego zbioru:

- jedynie kolumnę z nazwiskiem
- trzy kolumny – z imieniem, nazwiskiem i zarobkami
- pierwszy wiersz i drugą kolumnę
- trzy pierwsze wiersze i czwartą kolumnę (funkcja `.iloc`)
- trzy ostatnie wiersze i kolumny o („Pythonowskich”) indeksach 3 i 5