

```
1
2
3     Praktyczne_zastosowanie_Pythona = {
4
5         w_naukach : ['biologicznych', 'i medycznych'],
6         dla :      ['początkujących'],
7         poziom :   ['drugi'],
8         część :    ['druga'],
9
10
11     }
12     # Mateusz Dobrychłop, 21 listopada 2023
13
14
```

```
1 harmonogram_listopad = {
```

```
2  
3     M    T    W    T    F    S    S
```

```
4  
5         01  02  03  04  05
```

```
6  
7     06  07  08  09  10  11  12
```

```
8     13  14  15  16  17  18  19     Spotkanie 1 (17:00 – 19:30)
```

```
9  
10    20  21  22  23  24  25  26     Spotkanie 2 (17:00 – 19:30)
```

```
11    27  28  29  30     Spotkanie 3 (17:00 – 19:30)
```

```
12  
13  
14 }
```

```
1 Spotkanie_02 = {
2
3     01   profilowanie danych
4
5
6
7     02   czyszczenie danych
8
9
10
11     03   operacje na danych
12
13 }
14
```

1. Otwieramy linię komend (cmd, PowerShell, terminal itp.)
2. Przechodzimy do folderu z naszymi skryptami.
3. Aktywujemy środowisko wirtualne:



```
.\venv\Scripts\activate.ps1
```



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

```
PS C:\Users\Dobry\Documents\szkolenie2023> .\venv\Scripts\activate.ps1  
(venv) PS C:\Users\Dobry\Documents\szkolenie2023>
```



Aby umożliwić uruchamianie skryptów (i aktywację środowiska), jeśli są problemy

1. Uruchamiamy PowerShell **jako administrator**
2. Wpisujemy i zatwierdzamy następującą komendę:



```
Set-ExecutionPolicy unrestricted
```

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Pierwsze wiersze danych:

	GeneID	Control	TreatmentA	TreatmentB	ExperimentDate
0	Gene_00001	10.740511	11.758023	10.739191	2023-01-25
1	Gene_00002	12.081062	13.697375	9.325335	2023-01-24
2	Gene_00003	NaN	7.894456	NaN	2023-01-16
3	Gene_00004	8.267448	NaN	6.450783	2023-01-06
4	Gene_00005	9.889930	11.973398	5.140973	2023-01-10

Ostatnie wiersze danych:

	GeneID	Control	TreatmentA	TreatmentB	ExperimentDate
100	Gene_00045	NaN	6.053067	2.079006	2023-01-17
101	Gene_00048	NaN	7.96566	7.398985	2023-01-24
102	Gene_00065	8.015485	8.003116	5.826274	2023-01-27
103	Gene_00068	12.022571	16.096984	8.221162	2023-01-04
104	Gene_00068	12.800880	15.381249	9.356385	2023-01-27

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Pierwsze wiersze danych:

	GeneID	Control	TreatmentA	TreatmentB	ExperimentDate
0	Gene_00001	10.740511	11.758023	10.739191	2023-01-25
1	Gene_00002	12.081062	13.697375	9.325335	2023-01-24
2	Gene_00003	NaN	7.894456	NaN	2023-01-16
3	Gene_00004	8.267448	NaN	6.450783	2023-01-06
4	Gene_00005	9.889930	11.973398	5.140973	2023-01-10

Ostatnie wiersze danych:

	GeneID	Control	TreatmentA	TreatmentB	ExperimentDate
100	Gene_00045	NaN	6.053067	2.079006	2023-01-17
101	Gene_00048	NaN	7.96566	7.398985	2023-01-24
102	Gene_00065	8.015485	8.003116	5.826274	2023-01-27
103	Gene_00068	12.022571	16.096984	8.221162	2023-01-04
104	Gene_00068	12.800880	15.381249	9.356385	2023-01-27

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Informacje o danych:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 105 entries, 0 to 104  
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	GeneID	105 non-null	object
1	Control	95 non-null	float64
2	TreatmentA	95 non-null	object
3	TreatmentB	95 non-null	float64
4	ExperimentDate	105 non-null	datetime64[ns]

dtypes: datetime64[ns](1), float64(2), object(2)  
memory usage: 4.2+ KB  
None



```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Informacje o danych:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 105 entries, 0 to 104  
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	GeneID	105 non-null	object
1	Control	95 non-null	float64
2	TreatmentA	95 non-null	object
3	TreatmentB	95 non-null	float64
4	ExperimentDate	105 non-null	datetime64[ns]

dtypes: datetime64[ns](1), float64(2), object(2)  
memory usage: 4.2+ KB  
None

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Informacje o danych:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 105 entries, 0 to 104  
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	GeneID	105 non-null	object
1	Control	95 non-null	float64
2	TreatmentA	95 non-null	object
3	TreatmentB	95 non-null	float64
4	ExperimentDate	105 non-null	datetime64[ns]

dtypes: datetime64[ns](1), float64(2), object(2)  
memory usage: 4.2+ KB  
None

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Statystyki opisowe:

	Control	TreatmentB	ExperimentDate
count	95.000000	95.000000	105
mean	10.286831	8.254858	2023-01-14 09:08:34.285714176
min	4.965126	1.317493	2023-01-01 00:00:00
25%	9.053453	6.480723	2023-01-06 00:00:00
50%	10.237213	8.291575	2023-01-15 00:00:00
75%	12.052610	10.299730	2023-01-22 00:00:00
max	14.764449	14.484112	2023-01-28 00:00:00
std	2.106738	2.609714	NaN

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Podgląd pierwszych i ostatnich wierszy
7 print("Pierwsze wiersze danych:")
8 print(data.head())
9 print("\nOstatnie wiersze danych:")
10 print(data.tail())
11
12 # Informacje o strukturze i typach danych
13 print("\nInformacje o danych:")
14 print(data.info())
15
16 # Podstawowe statystyki opisowe
17 print("\nStatystyki opisowe:")
18 print(data.describe())
19
```

Statystyki opisowe:

	Control	TreatmentB	TreatmentA?	ExperimentDate
count	95.000000	95.000000		105
mean	10.286831	8.254858	2023-01-14 09:08:34.285714	176
min	4.965126	1.317493		2023-01-01 00:00:00
25%	9.053453	6.480723		2023-01-06 00:00:00
50%	10.237213	8.291575		2023-01-15 00:00:00
75%	12.052610	10.299730		2023-01-22 00:00:00
max	14.764449	14.484112		2023-01-28 00:00:00
std	2.106738	2.609714		NaN

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Liczba unikalnych wartości w każdej kolumnie
7 print("Liczba unikalnych wartości w kolumnach:")
8 print(data.nunique())
9
10 # Liczba unikalnych wartości dla konkretnej kolumny:
11 print("\nLiczba unikalnych wartości w kolumnie 'GeneID':")
12 print(data['GeneID'].nunique())
13
14 # Unikalne wartości dla konkretnej kolumny:
15 print("\nUnikalne wartości w kolumnie 'GeneID':")
16 print(data['GeneID'].unique())
17
18 # Liczba brakujących wartości w każdej kolumnie
19 print("\nLiczba brakujących wartości w kolumnach:")
20 print(data.isnull().sum())
21
22 # Liczba brakujących wartości dla konkretnej kolumny:
23 print("\nLiczba brakujących wartości w kolumnie 'Control':")
24 print(data['Control'].isnull().sum())
25
```

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Liczba unikalnych wartości w każdej kolumnie
7 print("Liczba unikalnych wartości w kolumnach:")
8 print(data.nunique())
9
10 # Liczba unikalnych wartości dla konkretnej kolumny:
11 print("\nLiczba unikalnych wartości w kolumnie 'GeneID':")
12 print(data['GeneID'].nunique())
13
14 # Unikalne wartości dla konkretnej kolumny:
15 print("\nUnikalne wartości w kolumnie 'GeneID':")
16 print(data['GeneID'].unique())
17
18 # Liczba brakujących wartości w każdej kolumnie
19 print("\nLiczba brakujących wartości w kolumnach:")
20 print(data.isnull().sum())
21
22 # Liczba brakujących wartości dla konkretnej kolumny:
23 print("\nLiczba brakujących wartości w kolumnie 'Control':")
24 print(data['Control'].isnull().sum())
25
```

Liczba unikalnych wartości w kolumnach:

GeneID	100
Control	95
TreatmentA	95
TreatmentB	95
ExperimentDate	27
dtype:	int64

Liczba unikalnych wartości w kolumnie 'GeneID':  
100

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Liczba unikalnych wartości w każdej kolumnie
7 print("Liczba unikalnych wartości w kolumnach:")
8 print(data.nunique())
9
10 # Liczba unikalnych wartości dla konkretnej kolumny:
11 print("\nLiczba unikalnych wartości w kolumnie 'GeneID':")
12 print(data['GeneID'].nunique())
13
14 # Unikalne wartości dla konkretnej kolumny:
15 print("\nUnikalne wartości w kolumnie 'GeneID':")
16 print(data['GeneID'].unique())
17
18 # Liczba brakujących wartości w każdej kolumnie
19 print("\nLiczba brakujących wartości w kolumnach:")
20 print(data.isnull().sum())
21
22 # Liczba brakujących wartości dla konkretnej kolumny:
23 print("\nLiczba brakujących wartości w kolumnie 'Control':")
24 print(data['Control'].isnull().sum())
25
```

```
Unikalne wartości w kolumnie 'GeneID':
['Gene_00001' 'Gene_00002' 'Gene_00003' 'Gene_00004' 'Gene_00005'
 'Gene_00006' 'Gene_00007' 'Gene_00008' 'Gene_00009' 'Gene_00010'
 'Gene_00011' 'Gene_00012' 'Gene_00013' 'Gene_00014' 'Gene_00015'
 'Gene_00016' 'Gene_00017' 'Gene_00018' 'Gene_00019' 'Gene_00020'
 'Gene_00021' 'Gene_00022' 'Gene_00023' 'Gene_00024' 'Gene_00025'
 'Gene_00026' 'Gene_00027' 'Gene_00028' 'Gene_00029' 'Gene_00030'
 'Gene_00031' 'Gene_00032' 'Gene_00033' 'Gene_00034' 'Gene_00035'
 'Gene_00036' 'Gene_00037' 'Gene_00038' 'Gene_00039' 'Gene_00040'
 'Gene_00041' 'Gene_00042' 'Gene_00043' 'Gene_00044' 'Gene_00045'
 'Gene_00046' 'Gene_00047' 'Gene_00048' 'Gene_00049' 'Gene_00050'
 'Gene_00051' 'Gene_00052' 'Gene_00053' 'Gene_00054' 'Gene_00055'
 'Gene_00056' 'Gene_00057' 'Gene_00058' 'Gene_00059' 'Gene_00060'
 'Gene_00061' 'Gene_00062' 'Gene_00063' 'Gene_00064' 'Gene_00065'
 'Gene_00066' 'Gene_00067' 'Gene_00068' 'Gene_00069' 'Gene_00070'
 'Gene_00071' 'Gene_00072' 'Gene_00073' 'Gene_00074' 'Gene_00075'
 'Gene_00076' 'Gene_00077' 'Gene_00078' 'Gene_00079' 'Gene_00080'
 'Gene_00081' 'Gene_00082' 'Gene_00083' 'Gene_00084' 'Gene_00085'
 'Gene_00086' 'Gene_00087' 'Gene_00088' 'Gene_00089' 'Gene_00090'
 'Gene_00091' 'Gene_00092' 'Gene_00093' 'Gene_00094' 'Gene_00095'
 'Gene_00096' 'Gene_00097' 'Gene_00098' 'Gene_00099' 'Gene_00100']
```

```
1 import pandas as pd
2
3 # Wczytywanie danych
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Liczba unikalnych wartości w każdej kolumnie
7 print("Liczba unikalnych wartości w kolumnach:")
8 print(data.nunique())
9
10 # Liczba unikalnych wartości dla konkretnej kolumny:
11 print("\nLiczba unikalnych wartości w kolumnie 'GeneID':")
12 print(data['GeneID'].nunique())
13
14 # Unikalne wartości dla konkretnej kolumny:
15 print("\nUnikalne wartości w kolumnie 'GeneID':")
16 print(data['GeneID'].unique())
17
18 # Liczba brakujących wartości w każdej kolumnie
19 print("\nLiczba brakujących wartości w kolumnach:")
20 print(data.isnull().sum())
21
22 # Liczba brakujących wartości dla konkretnej kolumny:
23 print("\nLiczba brakujących wartości w kolumnie 'Control':")
24 print(data['Control'].isnull().sum())
25
```

Liczba brakujących wartości w kolumnach:

GeneID	0
Control	10
TreatmentA	10
TreatmentB	10
ExperimentDate	0
dtype:	int64

Liczba brakujących wartości w kolumnie 'Control':  
10



Zdefiniuj funkcję, która jako argumenty będzie przyjmowała ścieżkę do pliku Excela z danymi, oraz ścieżkę do tekstowego pliku wyjściowego.

Funkcja powinna tworzyć (pod podaną ścieżką wyjściową) plik tekstowy, o strukturze kolumnowej (tzn. wartości w wierszach powinny być oddzielone tabulatorem). Zawartością pliku powinien być raport uwzględniający następujące informacje o pliku wejściowym:

- liczba unikalnych wartości we wszystkich kolumnach łącznie
- liczba brakujących wartości we wszystkich kolumnach łącznie
- liczba brakujących wartości w kolumnie o największej liczbie brakujących wartości

```
1  import pandas as pd
2
3  data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5  print("\nStatystyki opisowe:")
6  print(data.describe())
7
8  # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
9
10 # Ustalamy dominujący typ danych w kolumnie:
11 main_data_type = data['TreatmentA'].apply(type).mode().values[0]
12
13 # Wyszukujemy komórki, których typ jest inny niż dominujący:
14 cells_with_different_type = data[data['TreatmentA'].apply(type) != main_data_type]
15
16 print(cells_with_different_type)
17
```

```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 print("\nStatystyki opisowe:")
6 print(data.describe())
7
8 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
9
10 # Ustalamy dominujący typ danych w kolumnie:
11 main_data_type = data['TreatmentA'].apply(type).mode().values[0]
12
13 # Wyszukujemy komórki, których typ jest inny niż dominujący:
14 cells_with_different_type = data[data['TreatmentA'].apply(type) != main_data_type]
15
16 print(cells_with_different_type)
17
```

	GeneID	Control	TreatmentA	TreatmentB	ExperimentDate	
35	Gene_00036	8.086137	10.242918526454'	5.224108	2023-01-07	
38	Gene_00039	NaN	11.6905220055877'	7.700813	2023-01-02	
50	Gene_00051	6.985808		error	4.929958	2023-01-19

```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
6
7 # Ustalamy dominujący typ danych w kolumnie (rozwiniecie krok po kroku):
8 # main_data_type = data['TreatmentA'].apply(type).mode().values[0]
9
10 # Wybieramy kolumnę 'TreatmentA' z DataFrame i zwraca jej typy danych:
11 ta_column = data['TreatmentA']
12 data_types = ta_column.apply(type)
13 print(data_types)
14
15 # Dodatkowo, możemy sprawdzić, jakie typy danych występują w kolumnie 'TreatmentA':
16 print(data_types.unique())
17
18 # Możemy też sprawdzić, ile razy występuje każdy typ danych:
19 print(data_types.value_counts())
20
21 # Znajdujemy najczęściej występujący typ danych w kolumnie 'TreatmentA'.
22 # .mode() zwraca Series, czyli, technicznie, sekwencję wartości.
23 # W praktyce jest to DataFrame z jedną kolumną i jednym wierszem.
24 # Aby uzyskać wartość z tej komórki, możemy użyć .values[0]:
25 main_data_type_series = data_types.mode()
26 main_data_type = main_data_type_series.values[0]
27 print(main_data_type)
28
29
30 # Wyszukujemy komórki, których typ jest inny niż dominujący (rozwiniecie krok po kroku):
31 cells_with_different_type = data[data_types != main_data_type]
32
33 print(cells_with_different_type)
34
```


```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
6
7 # Ustalamy dominujący typ danych w kolumnie (rozwiniecie krok po kroku):
8 # main_data_type = data['TreatmentA'].apply(type).mode().values[0]
9
10 # Wybieramy kolumnę 'TreatmentA' z DataFrame i zwraca jej typy danych:
11 ta_column = data['TreatmentA']
12 data_types = ta_column.apply(type)
13 print(data_types)
14
15 # Dodatkowo, możemy sprawdzić, jakie typy danych występują w kolumnie 'TreatmentA':
16 print(data_types.unique())
17
18 # Możemy też sprawdzić, ile razy występuje każdy typ danych:
19 print(data_types.value_counts())
20
21 # Znajdujemy najczęściej występujący typ danych w kolumnie 'TreatmentA'.
22 # .mode() zwraca Series, czyli, technicznie, sekwencję wartości.
23 # W praktyce jest to DataFrame z jedną kolumną i jednym wierszem.
24 # Aby uzyskać wartość z tej komórki, możemy użyć .values[0]:
25 main_data_type_series = data_types.mode()
26 main_data_type = main_data_type_series.values[0]
27 print(main_data_type)
28
29
30 # Wyszukujemy komórki, których typ jest inny niż dominujący (rozwiniecie krok po kroku):
31 cells_with_different_type = data[data_types != main_data_type]
32
33 print(cells_with_different_type)
34
```

```
0      <class 'float'>
1      <class 'float'>
2      <class 'float'>
3      <class 'float'>
4      <class 'float'>
...
100     <class 'float'>
101     <class 'float'>
102     <class 'float'>
103     <class 'float'>
104 <class 'float'>
Name: TreatmentA, Length: 105, dtype: object
```

```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
6
7 # Ustalamy dominujący typ danych w kolumnie (rozwiniecie krok po kroku):
8 # main_data_type = data['TreatmentA'].apply(type).mode().values[0]
9
10 # Wybieramy kolumnę 'TreatmentA' z DataFrame i zwraca jej typy danych:
11 ta_column = data['TreatmentA']
12 data_types = ta_column.apply(type)
13 print(data_types)
14
15 # Dodatkowo, możemy sprawdzić, jakie typy danych występują w kolumnie 'TreatmentA':
16 print(data_types.unique())
17
18 # Możemy też sprawdzić, ile razy występuje każdy typ danych:
19 print(data_types.value_counts())
20
21 # Znajdujemy najczęściej występujący typ danych w kolumnie 'TreatmentA'.
22 # .mode() zwraca Series, czyli, technicznie, sekwencję wartości.
23 # W praktyce jest to DataFrame z jedną kolumną i jednym wierszem.
24 # Aby uzyskać wartość z tej komórki, możemy użyć .values[0]:
25 main_data_type_series = data_types.mode()
26 main_data_type = main_data_type_series.values[0]
27 print(main_data_type)
28
29
30 # Wyszukujemy komórki, których typ jest inny niż dominujący (rozwiniecie krok po kroku):
31 cells_with_different_type = data[data_types != main_data_type]
32
33 print(cells_with_different_type)
34
```

[<class 'float'> <class 'str'>]

```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
6
7 # Ustalamy dominujący typ danych w kolumnie (rozwiniecie krok po kroku):
8 # main_data_type = data['TreatmentA'].apply(type).mode().values[0]
9
10 # Wybieramy kolumnę 'TreatmentA' z DataFrame i zwraca jej typy danych:
11 ta_column = data['TreatmentA']
12 data_types = ta_column.apply(type)
13 print(data_types)
14
15 # Dodatkowo, możemy sprawdzić, jakie typy danych występują w kolumnie 'TreatmentA':
16 print(data_types.unique())
17
18 # Możemy też sprawdzić, ile razy występuje każdy typ danych:
19 print(data_types.value_counts())
20
21 # Znajdujemy najczęściej występujący typ danych w kolumnie 'TreatmentA'.
22 # .mode() zwraca Series, czyli, technicznie, sekwencję wartości.
23 # W praktyce jest to DataFrame z jedną kolumną i jednym wierszem.
24 # Aby uzyskać wartość z tej komórki, możemy użyć .values[0]:
25 main_data_type_series = data_types.mode()
26 main_data_type = main_data_type_series.values[0]
27 print(main_data_type)
28
29
30 # Wyszukujemy komórki, których typ jest inny niż dominujący (rozwiniecie krok po kroku):
31 cells_with_different_type = data[data_types != main_data_type]
32
33 print(cells_with_different_type)
34
```



```
TreatmentA
<class 'float'>    102
<class 'str'>      3
Name: count, dtype: int64
```

```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
6
7 # Ustalamy dominujący typ danych w kolumnie (rozwiniecie krok po kroku):
8 # main_data_type = data['TreatmentA'].apply(type).mode().values[0]
9
10 # Wybieramy kolumnę 'TreatmentA' z DataFrame i zwraca jej typy danych:
11 ta_column = data['TreatmentA']
12 data_types = ta_column.apply(type)
13 print(data_types)
14
15 # Dodatkowo, możemy sprawdzić, jakie typy danych występują w kolumnie 'TreatmentA':
16 print(data_types.unique())
17
18 # Możemy też sprawdzić, ile razy występuje każdy typ danych:
19 print(data_types.value_counts())
20
21 # Znajdujemy najczęściej występujący typ danych w kolumnie 'TreatmentA'.
22 # .mode() zwraca Series, czyli, technicznie, sekwencję wartości.
23 # W praktyce jest to DataFrame z jedną kolumną i jednym wierszem.
24 # Aby uzyskać wartość z tej komórki, możemy użyć .values[0]:
25 main_data_type_series = data_types.mode()
26 main_data_type = main_data_type_series.values[0]
27 print(main_data_type)
28
29
30 # Wyszukujemy komórki, których typ jest inny niż dominujący (rozwiniecie krok po kroku):
31 cells_with_different_type = data[data_types != main_data_type]
32
33 print(cells_with_different_type)
34
```



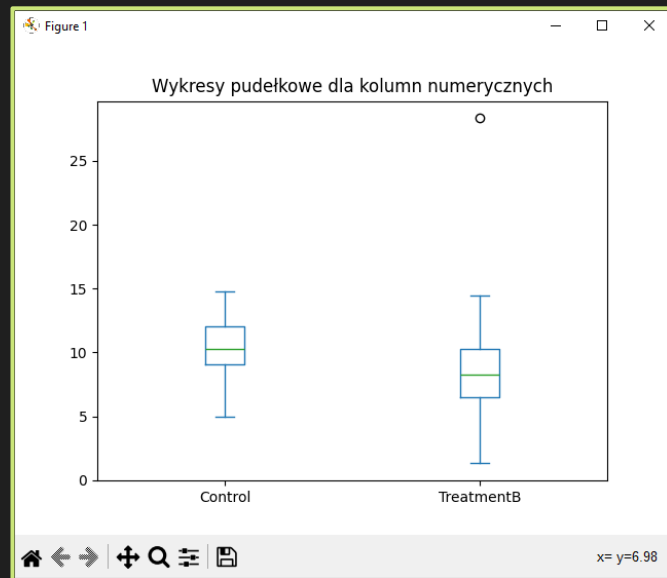
<class 'float'>



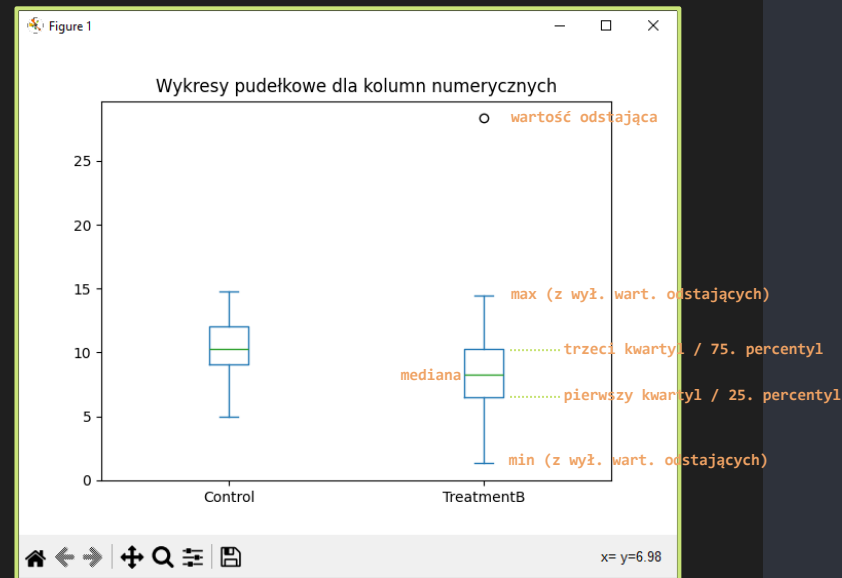
```
1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Szukamy wartości, których typ nie jest zgodny z typem dominującym w kolumnie.
6
7 # Ustalamy dominujący typ danych w kolumnie (rozwiniecie krok po kroku):
8 # main_data_type = data['TreatmentA'].apply(type).mode().values[0]
9
10 # Wybieramy kolumnę 'TreatmentA' z DataFrame i zwraca jej typy danych:
11 tA_column = data['TreatmentA']
12 data_types = tA_column.apply(type)
13 print(data_types)
14
15 # Dodatkowo, możemy sprawdzić, jakie typy danych występują w kolumnie 'TreatmentA':
16 print(data_types.unique())
17
18 # Możemy też sprawdzić, ile razy występuje każdy typ danych:
19 print(data_types.value_counts())
20
21 # Znajdujemy najczęściej występujący typ danych w kolumnie 'TreatmentA'.
22 # .mode() zwraca Series, czyli, technicznie, sekwencję wartości.
23 # W praktyce jest to DataFrame z jedną kolumną i jednym wierszem.
24 # Aby uzyskać wartość z tej komórki, możemy użyć .values[0]:
25 main_data_type_series = data_types.mode()
26 main_data_type = main_data_type_series.values[0]
27 print(main_data_type)
28
29
30 # Wyszukujemy komórki, których typ jest inny niż dominujący (rozwiniecie krok po kroku):
31 cells_with_different_type = data[data_types != main_data_type]
32
33 print(cells_with_different_type)
34
```

	GeneID	Control	TreatmentA	TreatmentB	ExperimentDate	
35	Gene_00036	8.086137	10.242918526454'	5.224108	2023-01-07	
38	Gene_00039	NaN	11.6905220055877'	7.700813	2023-01-02	
50	Gene_00051	6.985808		error	4.929958	2023-01-19

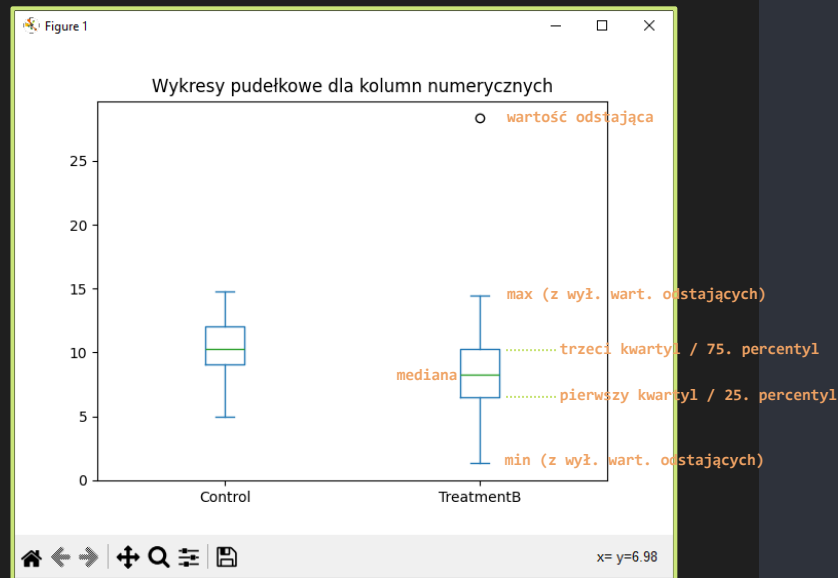
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Wykresy pudełkowe dla każdej kolumny numerycznej
7 data.plot(kind='box')
8 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
9 plt.show()
10
11 # Histogram dla konkretnej kolumny
12 data['TreatmentB'].hist()
13 plt.title('Histogram kolumny')
14 plt.show()
15
16 # Wykres słupkowy dla brakujących danych
17 data.isnull().sum().plot(kind='bar')
18 plt.title('Brakujące dane w każdej kolumnie')
19 plt.show()
20
```



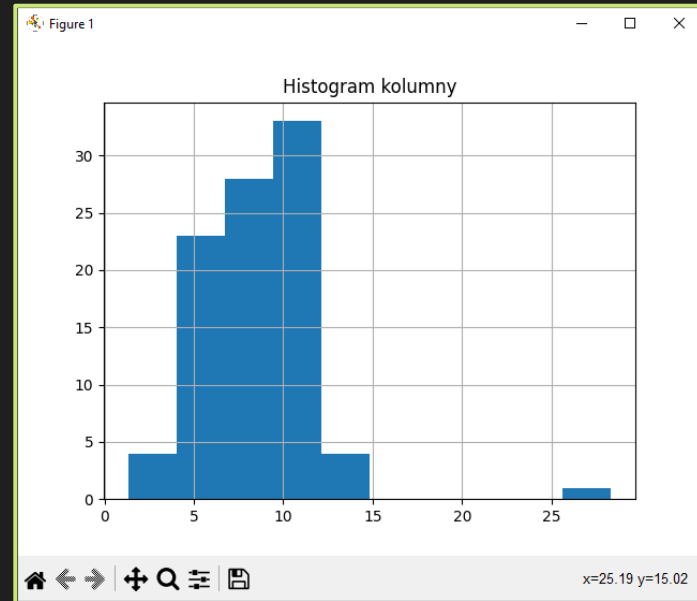
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Wykresy pudełkowe dla każdej kolumny numerycznej
7 data.plot(kind='box')
8 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
9 plt.show()
10
11 # Histogram dla konkretnej kolumny
12 data['TreatmentB'].hist()
13 plt.title('Histogram kolumny')
14 plt.show()
15
16 # Wykres słupkowy dla brakujących danych
17 data.isnull().sum().plot(kind='bar')
18 plt.title('Brakujące dane w każdej kolumnie')
19 plt.show()
20
```



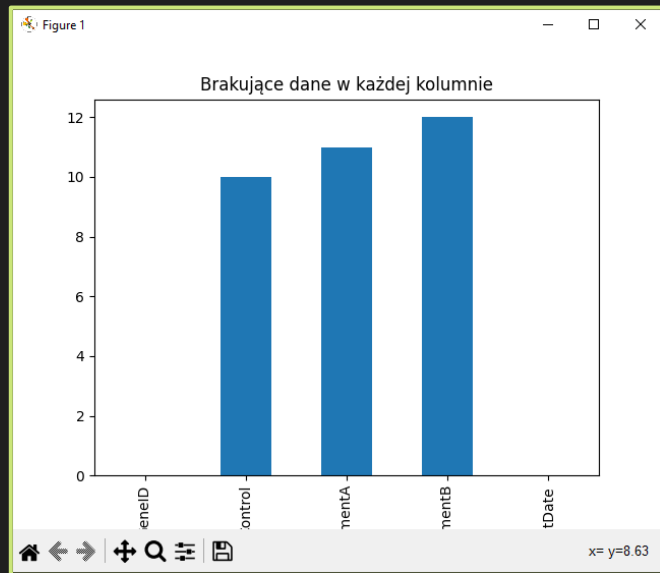
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Wykresy pudełkowe dla każdej kolumny numerycznej
7 data.plot(kind='box')
8 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
9 plt.show()
10
11 # Histogram dla konkretnej kolumny
12 data['TreatmentB'].hist()
13 plt.title('Histogram kolumny')
14 plt.show()
15
16 # Wykres słupkowy dla brakujących danych
17 data.isnull().sum().plot(kind='bar')
18 plt.title('Brakujące dane w każdej kolumnie')
19 plt.show()
20
```



```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Wykresy pudełkowe dla każdej kolumny numerycznej
7 data.plot(kind='box')
8 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
9 plt.show()
10
11 # Histogram dla konkretnej kolumny
12 data['TreatmentB'].hist()
13 plt.title('Histogram kolumny')
14 plt.show()
15
16 # Wykres słupkowy dla brakujących danych
17 data.isnull().sum().plot(kind='bar')
18 plt.title('Brakujące dane w każdej kolumnie')
19 plt.show()
20
```



```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Wykresy pudełkowe dla każdej kolumny numerycznej
7 data.plot(kind='box')
8 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
9 plt.show()
10
11 # Histogram dla konkretnej kolumny
12 data['TreatmentB'].hist()
13 plt.title('Histogram kolumny')
14 plt.show()
15
16 # Wykres słupkowy dla brakujących danych
17 data.isnull().sum().plot(kind='bar')
18 plt.title('Brakujące dane w każdej kolumnie')
19 plt.show()
20
```



```
1  import pandas as pd
2
3  data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5  # Zamiana wartości w kolumnie 'TreatmentA' na typ float "na siłę", tzn.
6  # jeśli wartość nie jest liczbą, to zostanie zamieniona na NaN.
7  data['TreatmentA'] = pd.to_numeric(data['TreatmentA'], errors='coerce')
8
```

```

1 import pandas as pd
2
3 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
4
5 # Ustalamy listę komórek z wartościami, które nie są typu float:
6 main_data_type = data['TreatmentA'].apply(type).mode().values[0]
7 cells_with_different_type = data[data['TreatmentA'].apply(type) != main_data_type]
8 print(cells_with_different_type)
9
10 # Wynik (3 komórki o indeksach 35, 38 i 50):
11 #      GeneID      Control      TreatmentA      TreatmentB      ExperimentDate
12 # 35  Gene_00036  8.086137  10.242918526454'  5.224108      2023-01-07
13 # 38  Gene_00039      NaN  11.6905220055877'  7.700813      2023-01-02
14 # 50  Gene_00051  6.985808          error  4.929958      2023-01-19
15
16 # Zauważmy, że w komórkach 35 i 38 wartości są poprzedzone znakiem '.
17 # Możemy usunąć ten znak i przekształcić wartości na float.
18 data.at[35, 'TreatmentA'] = data.at[35, 'TreatmentA'].replace("'", "")
19 data.at[35, 'TreatmentA'] = float(data.at[35, 'TreatmentA'])
20
21 data.at[38, 'TreatmentA'] = data.at[38, 'TreatmentA'].replace("'", "")
22 data.at[38, 'TreatmentA'] = float(data.at[38, 'TreatmentA'])
23
24 # data.at[35, 'TreatmentA'] = 1.0
25 # data.at[38, 'TreatmentA'] = 2.0
26 # data.at[50, 'TreatmentA'] = 2.0
27 # Wartość 'error' w komórce 50 jest trudna do przekształcenia na float.
28 # Możemy usunąć cały wiersz, jeśli nie jest nam potrzebny.
29 # inplace=True oznacza, że zmiany będą dokonane na obiekcie data. (bez przypisania do nowej zmiennej)
30 data.drop([50], inplace=True)
31
32 # W DataFrame kolumna TreatmentA nadal ma przypisany typ "object".
33 # Zmieniamy typ na float:
34 data['TreatmentA'] = pd.to_numeric(data['TreatmentA'])
35
36 print(data.info())
37

```

```

GeneID      Control      TreatmentA      TreatmentB      ExperimentDate
35  Gene_00036  8.086137  10.242918526454'  5.224108      2023-01-07
38  Gene_00039      NaN  11.6905220055877'  7.700813      2023-01-02
50  Gene_00051  6.985808          error  4.929958      2023-01-19
<class 'pandas.core.frame.DataFrame'>
Index: 104 entries, 0 to 104
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   GeneID                104 non-null   object
1   Control               94 non-null    float64
2   TreatmentA            93 non-null    float64
3   TreatmentB            92 non-null    float64
4   ExperimentDate        104 non-null   datetime64[ns]
dtypes: datetime64[ns](1), float64(3), object(1)
memory usage: 4.9+ KB
None

```



```

1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6  # Ujednolicanie typów danych w kolumnie 'TreatmentA':
7
8  data.at[35, 'TreatmentA'] = data.at[35, 'TreatmentA'].replace("", "")
9  data.at[35, 'TreatmentA'] = float(data.at[35, 'TreatmentA'])
10 data.at[38, 'TreatmentA'] = data.at[38, 'TreatmentA'].replace("", "")
11 data.at[38, 'TreatmentA'] = float(data.at[38, 'TreatmentA'])
12 data.drop([50], inplace=True)
13 data['TreatmentA'] = pd.to_numeric(data['TreatmentA'])
14
15 # Pozbywanie się duplikatów:
16 data.drop_duplicates(inplace=True)
17
18 # Usuwanie brakujących wartości:
19 data.dropna(inplace=True)
20
21 # Analiza wyczyszczonego DataFrame:
22 print(data.info())
23
24 data.plot(kind='box')
25 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
26 plt.show()
27

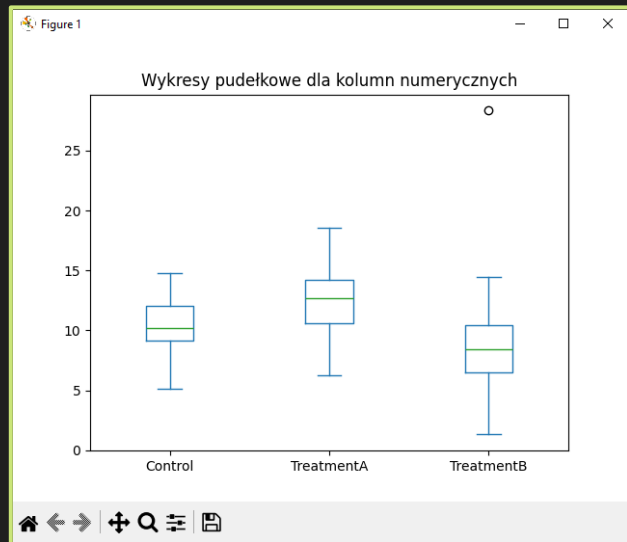
```

```

<class 'pandas.core.frame.DataFrame'>
Index: 74 entries, 0 to 104
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   GeneID          74 non-null    object
1   Control         74 non-null    float64
2   TreatmentA      74 non-null    float64
3   TreatmentB      74 non-null    float64
4   ExperimentDate  74 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(3), object(1)
memory usage: 3.5+ KB
None

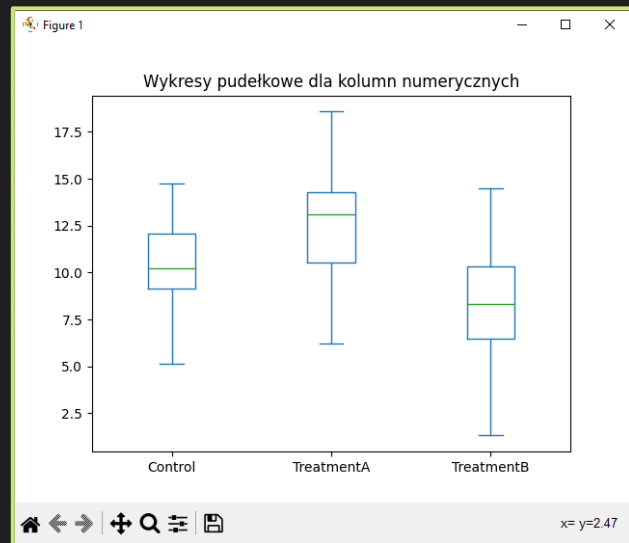
```

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Ujednolicanie typów danych w kolumnie 'TreatmentA':
7
8 data.at[35, 'TreatmentA'] = data.at[35, 'TreatmentA'].replace("", "")
9 data.at[35, 'TreatmentA'] = float(data.at[35, 'TreatmentA'])
10 data.at[38, 'TreatmentA'] = data.at[38, 'TreatmentA'].replace("", "")
11 data.at[38, 'TreatmentA'] = float(data.at[38, 'TreatmentA'])
12 data.drop([50], inplace=True)
13 data['TreatmentA'] = pd.to_numeric(data['TreatmentA'])
14
15 # Pozbywanie się duplikatów:
16 data.drop_duplicates(inplace=True)
17
18 # Usuwanie brakujących wartości:
19 data.dropna(inplace=True)
20
21 # Analiza wyczyszczonego DataFrame:
22 print(data.info())
23
24 data.plot(kind='box')
25 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
26 plt.show()
27
```



```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6  # Ujednolicanie typów danych w kolumnie 'TreatmentA':
7
8  data.at[35, 'TreatmentA'] = data.at[35, 'TreatmentA'].replace("", "")
9  data.at[35, 'TreatmentA'] = float(data.at[35, 'TreatmentA'])
10 data.at[38, 'TreatmentA'] = data.at[38, 'TreatmentA'].replace("", "")
11 data.at[38, 'TreatmentA'] = float(data.at[38, 'TreatmentA'])
12 data.drop([50], inplace=True)
13 data['TreatmentA'] = pd.to_numeric(data['TreatmentA'])
14
15 # Pozbywanie się duplikatów:
16 data.drop_duplicates(inplace=True)
17
18 # Usuwanie brakujących wartości:
19 data.dropna(inplace=True)
20
21 # Pozbywanie się wartości odstających:
22 data.drop(data[data['TreatmentB'] > 25].index, inplace=True)
23
24 # Analiza wyczyszczonego DataFrame:
25 print(data.info())
26
27 data.plot(kind='box')
28 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
29 plt.show()
30
```

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6 # Ujednolicanie typów danych w kolumnie 'TreatmentA':
7
8 data.at[35, 'TreatmentA'] = data.at[35, 'TreatmentA'].replace("", "")
9 data.at[35, 'TreatmentA'] = float(data.at[35, 'TreatmentA'])
10 data.at[38, 'TreatmentA'] = data.at[38, 'TreatmentA'].replace("", "")
11 data.at[38, 'TreatmentA'] = float(data.at[38, 'TreatmentA'])
12 data.drop([50], inplace=True)
13 data['TreatmentA'] = pd.to_numeric(data['TreatmentA'])
14
15 # Pozbywanie się duplikatów:
16 data.drop_duplicates(inplace=True)
17
18 # Usuwanie brakujących wartości:
19 data.dropna(inplace=True)
20
21 # Pozbywanie się wartości odstających:
22 data.drop(data[data['TreatmentB'] > 25].index, inplace=True)
23
24 # Analiza wyczyszczonego DataFrame:
25 print(data.info())
26
27 data.plot(kind='box')
28 plt.title('Wykresy pudełkowe dla kolumn numerycznych')
29 plt.show()
30
```



```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  data = pd.read_excel('synthetic_gene_expression_data_2023.xlsx')
5
6  # Ujednolicanie typów danych w kolumnie 'TreatmentA':
7
8  data.at[35, 'TreatmentA'] = data.at[35, 'TreatmentA'].replace("", "")
9  data.at[35, 'TreatmentA'] = float(data.at[35, 'TreatmentA'])
10 data.at[38, 'TreatmentA'] = data.at[38, 'TreatmentA'].replace("", "")
11 data.at[38, 'TreatmentA'] = float(data.at[38, 'TreatmentA'])
12 data.drop([50], inplace=True)
13 data['TreatmentA'] = pd.to_numeric(data['TreatmentA'])
14
15 # Pozbywanie się duplikatów:
16 data.drop_duplicates(inplace=True)
17
18 # Usuwanie brakujących wartości:
19 data.dropna(inplace=True)
20
21 # Pozbywanie się wartości odstających:
22 data.drop(data[data['TreatmentB'] > 25].index, inplace=True)
23
24 # Zapis do pliku excela:
25 data.to_excel('cleaned_data.xlsx', index=False)
26
```

Napisz funkcję "profiler()", która będzie umożliwiać profilowanie oraz czyszczenie danych z dowolnego arkusza Excela, pod kątem podanych w formie argumentów aspektów (brakujące wartości / wartości odstające / duplikaty / błędny typ danych).

Na przykład, użytkownik powinien być w stanie podać w argumentach funkcji, jaki plik ma zostać poddany analizie i czyszczeniu, oraz również za pomocą argumentów zdefiniować, że funkcja ma znaleźć i usunąć z pliku wiersze z brakującymi wartościami, a potem wypisać krótkie podsumowanie w konsoli, lub wyświetlić wykres wizualizujący skalę problemu.

Jednym z argumentów powinna być ścieżka do pliku wejściowego. Proszę samodzielnie zaproponować, w jaki sposób użytkownik będzie decydował o tym, jaka analiza oraz czyszczenie są wykonywane.

```
1 harmonogram_listopad = {
```

```
2     M    T    W    T    F    S    S
```

```
3         01  02  03  04  05
```

```
4         06  07  08  09  10  11  12
```

```
5         13  14  15  16  17  18  19      Do 25 listopada
```

```
6         20  21  22  23  24  25  26      Zadania domowe
```

```
7         27  28  29  30      mateusz.dobrychlop@gmail.com
```

```
8     }
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**