pokebook

Alicja Hołowiecka

December 11, 2019

1 Analiza danych - pokemony

1.1 Wczytanie potrzebnych bibliotek

[201]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.stats as st
```

Biblioteka `pandas` - struktury danych i narzędzia do ich analizy

Biblioteka `numpy` - działania na wektorach i macierzach, algebra liniowa.

Biblioteka `matplotlib.pyplot` - rysowanie wykresów.

Biblioteka `os` - funkcje związane z systemem operacyjnym.

Biblioteka `scipy.stats` - rozkłady prawdopodobieństwa, funkcje statystyczne.

1.2 Wczytanie danych

Na początek wczytamy dane z pliku `.csv`. Zapiszemy dane w zmiennej o nazwie `df`, od `Data Frame`. Korzystamy z funkcji `read_csv` biblioteki `pandas`. Jeżeli potrzebujemy podać separator, możemy podać argument `sep` lub `delimiter`, ale na ogół Python powinien sam wykryć separator (o ile separator ma długość 1).

[24]:
```
df = pd.read_csv("https://raw.githubusercontent.com/AlicjaHol/pokemony/master/
↪pokemon_data.csv")
```

Możemy także wczytać te same dane z plików tekstowych lub Excela. Dla pliku `.txt` możemy także użyć funkcji `read_csv`, jednak musimy ustawić `delimiter` jako (w tym przypadku) tabulator, czyli `\t`, bo domyślnym separatorem dla `.csv` jest przecinek (jak sama nazwa wskazuje, CSV = Comma Separated Values).

[32]:
```
df_txt = pd.read_csv("https://raw.githubusercontent.com/AlicjaHol/pokemony/
↪master/pokemon_data.txt", delimiter = "\t")
```

Dla plików `.xls` i `.xslx` istieje funkcja `read_excel`.

[33]:
```
df_excel = pd.read_excel("pokemon_data.xlsx")
```

Nie musieliśmy podawać pełnej ścieżki dostępu do pliku, ponieważ:

[34]:
```
os.getcwd()
```

[34]:
```
'C:\\Users\\alaho\\Desktop\\Ala\\analiza_danych\\pokemony'
```

A zatem znajdujemy się w odpowiednim folderze. Gdybyśmy mieli inny folder roboczy (`getcwd()` = get current working directory), to trzeba by było podać pełną ścieżkę.

1.3 Podstawowe działania na danych

Za pomocą funkcji `head()` możemy wyświetlić daną liczbę wierszy (od góry). Domyślnie jest to 5 wierszy. Analogicznie działa funkcja `tail()`, tyle że wyświetla końcowe wiersze.

[25]:
```
df.head()
```

[25]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	False

[26]:
```
df.tail(3)
```

[26]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
797	720	HoopaHoopa Confined	Psychic	Ghost	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	80	160	60	170	130	80	6	True
799	721	Volcanion	Fire	Water	80	110	120	130	90	70	6	True

[36]:
```
df_txt.head()
```

[36]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False

```
1  2               Ivysaur  Grass  Poison  60      62       63       80
2  3              Venusaur  Grass  Poison  80      82       83      100
3  3  VenusaurMega Venusaur  Grass  Poison  80     100      123      122
4  4             Charmander   Fire     NaN  39      52       43       60

   Sp. Def  Speed  Generation  Legendary
0       65     45           1      False
1       80     60           1      False
2      100     80           1      False
3      120     80           1      False
4       50     65           1      False
```

[37]:
```
df_excel.head()
```

[37]:
```
   #                  Name Type 1  Type 2  HP  Attack  Defense  Sp. Atk  \
0  1             Bulbasaur  Grass  Poison  45      49       49       65
1  2               Ivysaur  Grass  Poison  60      62       63       80
2  3              Venusaur  Grass  Poison  80      82       83      100
3  3  VenusaurMega Venusaur  Grass  Poison  80     100      123      122
4  4             Charmander   Fire     NaN  39      52       43       60

   Sp. Def  Speed  Generation  Legendary
0       65     45           1      False
1       80     60           1      False
2      100     80           1      False
3      120     80           1      False
4       50     65           1      False
```

Sprawdzimy, jakiego typu są poszczególne kolumny w ramce danych. `dtypes` zwraca nazwę kolumny oraz typ danych.

1.4 Oczyszczenie danych

[27]:
```
df.dtypes
```

[27]:
```
#              int64
Name          object
Type 1        object
Type 2        object
HP             int64
Attack         int64
Defense        int64
Sp. Atk        int64
Sp. Def        int64
Speed          int64
Generation     int64
Legendary       bool
```

```
dtype: object
```

Nie ma potrzeby zmieniania typów danych.

Teraz wyświetlimy nazwy kolumn:

[38]:
```
df.columns
```

[38]:
```
Index(['#', 'Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary'],
      dtype='object')
```

Aby uniknąć błędów związanych ze spacjami i kropkami w nazwach, zmienimy nazwy kolumn. Nie jest to konieczne, jeżeli będziemy pisać nazwy w cudzysłowie, to wszystko powinno działać. Ale jeśli "oczyścimy" nazwy kolumn, to znacząco ułatwimy sobie pracę.

[43]:
```
df.columns = \
['Number', 'Name', 'Type1', 'Type2', 'HP', 'Attack', 'Defense', 'Sp_Atk',␣
 ↪'Sp_Def', 'Speed', 'Generation', 'Legendary']
```

[44]:
```
df.head()
```

[44]:

	Number	Name	Type1	Type2	HP	Attack	Defense	Sp_Atk	\
0	1	Bulbasaur	Grass	Poison	45	49	49	65	
1	2	Ivysaur	Grass	Poison	60	62	63	80	
2	3	Venusaur	Grass	Poison	80	82	83	100	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	
4	4	Charmander	Fire	NaN	39	52	43	60	

	Sp_Def	Speed	Generation	Legendary
0	65	45	1	False
1	80	60	1	False
2	100	80	1	False
3	120	80	1	False
4	50	65	1	False

Możemy też zobaczyć, w jaki sposób są nazwane wiersze, czyli wyświetlimy zakres indeksów.

[45]:
```
df.index
```

[45]:
```
RangeIndex(start=0, stop=800, step=1)
```

Sprawdzimy, czy są jakieś braki danych

[76]:
```
pd.isna(df).sum()
```

[76]:
```
Number          0
Name            0
Type1           0
```

```
Type2         386
HP              0
Attack          0
Defense         0
Sp_Atk          0
Sp_Def          0
Speed           0
Generation      0
Legendary       0
dtype: int64
```

Jak widać, braki występują tylko w kolumnie `Type2`. Nie wszystkie pokemony muszą mieć dwa typy, także wszystko ok.

Zatem mamy dane dla 800 pokemonów.

1.5 Podstawowe statystyki

Wyświetlimy podstawowe statystyki opisowe dla danych.

[46]:
```
df.describe()
```

[46]:
```
            Number          HP      Attack     Defense      Sp_Atk      Sp_Def  \
count   800.000000  800.000000  800.000000  800.000000  800.000000  800.000000
mean    362.813750   69.258750   79.001250   73.842500   72.820000   71.902500
std     208.343798   25.534669   32.457366   31.183501   32.722294   27.828916
min       1.000000    1.000000    5.000000    5.000000   10.000000   20.000000
25%     184.750000   50.000000   55.000000   50.000000   49.750000   50.000000
50%     364.500000   65.000000   75.000000   70.000000   65.000000   70.000000
75%     539.250000   80.000000  100.000000   90.000000   95.000000   90.000000
max     721.000000  255.000000  190.000000  230.000000  194.000000  230.000000

            Speed  Generation
count  800.000000   800.00000
mean    68.277500     3.32375
std     29.060474     1.66129
min      5.000000     1.00000
25%     45.000000     2.00000
50%     65.000000     3.00000
75%     90.000000     5.00000
max    180.000000     6.00000
```

Możemy wybrać, które kwantyle mają być wyświetlone, za pomocą parametru `percentiles`.

[47]:
```
df.describe(percentiles = [.1, .2, .5, .9])
```

[47]:
```
            Number          HP      Attack     Defense      Sp_Atk      Sp_Def  \
count   800.000000  800.000000  800.000000  800.000000  800.000000  800.000000
```

```
mean   362.813750   69.258750   79.001250   73.842500   72.820000   71.902500
std    208.343798   25.534669   32.457366   31.183501   32.722294   27.828916
min      1.000000    1.000000    5.000000    5.000000   10.000000   20.000000
10%     73.900000   40.000000   40.000000   40.000000   35.000000   40.000000
20%    147.800000   50.000000   50.000000   48.000000   45.000000   49.800000
50%    364.500000   65.000000   75.000000   70.000000   65.000000   70.000000
90%    651.100000  100.000000  125.000000  115.000000  120.000000  107.000000
max    721.000000  255.000000  190.000000  230.000000  194.000000  230.000000

              Speed  Generation
count  800.000000   800.00000
mean    68.277500     3.32375
std     29.060474     1.66129
min      5.000000     1.00000
10%     30.000000     1.00000
20%     41.000000     1.00000
50%     65.000000     3.00000
90%    106.000000     6.00000
max    180.000000     6.00000
```

Możemy także policzyć m. in. średnią dla poszczególnych kolumn (domyślnie funkcja `mean()` wybiera tylko kolumny numeryczne, tj. liczbowe albo logiczne).

[54]:
```
df.mean()
```

[54]:
```
Number        362.81375
HP             69.25875
Attack         79.00125
Defense        73.84250
Sp_Atk         72.82000
Sp_Def         71.90250
Speed          68.27750
Generation      3.32375
Legendary       0.08125
dtype: float64
```

[178]:
```
df.median()
```

[178]:
```
Number        364.5
HP             65.0
Attack         75.0
Defense        70.0
Sp_Atk         65.0
Sp_Def         70.0
Speed          65.0
Generation      3.0
Legendary       0.0
```

```
dtype: float64
```

1.6 Odwoływanie się do poszczególnych kolumn, wierszy, komórek

Do danej kolumny możemy się odwołać na kilka sposobów.

Najłatwiej jest odwołać się do kolumny poprzez jej nazwę, to jest:

[62]:
```
df.HP
```

[62]:
```
0      45
1      60
2      80
3      80
4      39
      ..
795    50
796    50
797    80
798    80
799    80
Name: HP, Length: 800, dtype: int64
```

UWAGA! Powyższe polecenie zadziała tylko dla odpowiednio "oczyszczonych" nazw kolumn, tj. bez spacji i kropek.

Z kolei dla dowolnych nazw kolumn zadziała poniższe polecenie:

[63]:
```
df['HP']
```

[63]:
```
0      45
1      60
2      80
3      80
4      39
      ..
795    50
796    50
797    80
798    80
799    80
Name: HP, Length: 800, dtype: int64
```

Możemy się odwołać do kolumny poprzez jej numer. Przykładowo, patrząc na ramkę danych, widzimy, że kolumna `HP` ma indeks 4 (numeracja w Pythonie jest od zera). Należy użyć funkcji `iloc()` (integer location). Jako pierwszy argument podajemy numer wiersza (w naszym przypadku `:`, bo wybieramy wszystkie wiersze), jako drugi numer kolumny.

[61]:
```
df.iloc[:,4]
```

[61]:
```
0      45
1      60
2      80
3      80
4      39
      ..
795    50
796    50
797    80
798    80
799    80
Name: HP, Length: 800, dtype: int64
```

Możemy wyświetlić dane wiersze tabeli bez używania funkcji `iloc`.

[67]:
```
df[7:28]
```

[67]:

	Number	Name	Type1	Type2	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Generation	Legendary
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	False
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	False
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	False
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	1	False
11	9	Blastoise	Water	NaN	79	83	100					
12	9	BlastoiseMega Blastoise	Water	NaN	79	103	120					
13	10	Caterpie	Bug	NaN	45	30	35					
14	11	Metapod	Bug	NaN	50	20	55					
15	12	Butterfree	Bug	Flying	60	45	50					
16	13	Weedle	Bug	Poison	40	35	30					
17	14	Kakuna	Bug	Poison	45	25	50					
18	15	Beedrill	Bug	Poison	65	90	40					
19	15	BeedrillMega Beedrill	Bug	Poison	65	150	40					
20	16	Pidgey	Normal	Flying	40	45	40					
21	17	Pidgeotto	Normal	Flying	63	60	55					
22	18	Pidgeot	Normal	Flying	83	80	75					
23	18	PidgeotMega Pidgeot	Normal	Flying	83	80	80					
24	19	Rattata	Normal	NaN	30	56	35					
25	20	Raticate	Normal	NaN	55	81	60					
26	21	Spearow	Normal	Flying	40	60	30					
27	22	Fearow	Normal	Flying	65	90	65					

```
11      85     105     78           1      False
12     135     115     78           1      False
13      20      20     45           1      False
14      25      25     30           1      False
15      90      80     70           1      False
16      20      20     50           1      False
17      25      25     35           1      False
18      45      80     75           1      False
19      15      80    145           1      False
20      35      35     56           1      False
21      50      50     71           1      False
22      70      70    101           1      False
23     135      80    121           1      False
24      25      35     72           1      False
25      50      70     97           1      False
26      31      31     70           1      False
27      61      61    100           1      False
```

Używając funkcji `loc`, możemy się odnosić do kolumny nie tylko przez indeksy, ale także przez nazwy kolumn. Wyświetlimy w ten sposób całą kolumnę `Type1`.

[68]:
```
df.loc[:, 'Type1']
```

[68]:
```
0        Grass
1        Grass
2        Grass
3        Grass
4         Fire
        ...
795       Rock
796       Rock
797    Psychic
798    Psychic
799       Fire
Name: Type1, Length: 800, dtype: object
```

Można wyświetlić kilka kolumn jednocześnie, wtedy należy je przekazać do funkcji `loc` jako listę.

Wyświetlimy kolumny `Type1` i `Type2` dla kilku wierszy.

[69]:
```
df.loc[2:8, ['Type1', 'Type2']]
```

[69]:
```
   Type1   Type2
2  Grass  Poison
3  Grass  Poison
4   Fire     NaN
5   Fire     NaN
6   Fire  Flying
```

```
7   Fire  Dragon
8   Fire  Flying
```

A teraz wyświetlmy tylko pokemony, dla których `Type1` to `Grass`.

[71]:
```
df[df['Type1']== 'Grass']
```

[71]:
```
     Number                  Name  Type1     Type2   HP  Attack  Defense  \
0         1             Bulbasaur  Grass    Poison   45      49       49
1         2               Ivysaur  Grass    Poison   60      62       63
2         3              Venusaur  Grass    Poison   80      82       83
3         3  VenusaurMega Venusaur  Grass    Poison   80     100      123
48       43                Oddish  Grass    Poison   45      50       55
..      ...                   ...    ...       ...  ...     ...      ...
718     650               Chespin  Grass       NaN   56      61       65
719     651             Quilladin  Grass       NaN   61      78       95
720     652            Chesnaught  Grass  Fighting   88     107      122
740     672                Skiddo  Grass       NaN   66      65       48
741     673                Gogoat  Grass       NaN  123     100       62

     Sp_Atk  Sp_Def  Speed  Generation  Legendary
0        65      65     45           1      False
1        80      80     60           1      False
2       100     100     80           1      False
3       122     120     80           1      False
48       75      65     30           1      False
..      ...     ...    ...         ...        ...
718      48      45     38           6      False
719      56      58     57           6      False
720      74      75     64           6      False
740      62      57     52           6      False
741      97      81     68           6      False

[70 rows x 12 columns]
```

A teraz chcemy pokemony, dla których `Type1` to `Grass` lub `Fire`

[72]:
```
df[df['Type1'].isin(['Grass', 'Fire'])]
```

[72]:
```
     Number                  Name  Type1   Type2   HP  Attack  Defense  \
0         1             Bulbasaur  Grass  Poison   45      49       49
1         2               Ivysaur  Grass  Poison   60      62       63
2         3              Venusaur  Grass  Poison   80      82       83
3         3  VenusaurMega Venusaur  Grass  Poison   80     100      123
4         4            Charmander   Fire     NaN   39      52       43
..      ...                   ...    ...     ...  ...     ...      ...
735     667               Litleo   Fire  Normal   62      50       58
736     668               Pyroar   Fire  Normal   86      68       72
```

```
740     672                Skiddo  Grass     NaN   66      65       48
741     673                Gogoat  Grass     NaN  123     100       62
799     721             Volcanion   Fire   Water   80     110      120

     Sp_Atk  Sp_Def  Speed  Generation  Legendary
0        65      65     45           1      False
1        80      80     60           1      False
2       100     100     80           1      False
3       122     120     80           1      False
4        60      50     65           1      False
..      ...     ...    ...         ...        ...
735      73      54     72           6      False
736     109      66    106           6      False
740      62      57     52           6      False
741      97      81     68           6      False
799     130      90     70           6       True

[122 rows x 12 columns]
```

[73]:
```
df[df.Type1.isin(['Grass', 'Fire'])]
```

[73]:
```
     Number                  Name  Type1   Type2   HP  Attack  Defense  \
0         1             Bulbasaur  Grass  Poison   45      49       49
1         2               Ivysaur  Grass  Poison   60      62       63
2         3              Venusaur  Grass  Poison   80      82       83
3         3  VenusaurMega Venusaur  Grass  Poison   80     100      123
4         4            Charmander   Fire     NaN   39      52       43
..      ...                   ...    ...     ...  ...     ...      ...
735     667                Litleo   Fire  Normal   62      50       58
736     668                Pyroar   Fire  Normal   86      68       72
740     672                Skiddo  Grass     NaN   66      65       48
741     673                Gogoat  Grass     NaN  123     100       62
799     721             Volcanion   Fire   Water   80     110      120

     Sp_Atk  Sp_Def  Speed  Generation  Legendary
0        65      65     45           1      False
1        80      80     60           1      False
2       100     100     80           1      False
3       122     120     80           1      False
4        60      50     65           1      False
..      ...     ...    ...         ...        ...
735      73      54     72           6      False
736     109      66    106           6      False
740      62      57     52           6      False
741      97      81     68           6      False
799     130      90     70           6       True
```

```
[122 rows x 12 columns]
```

Możemy policzyć średnią lub inne statystyki dla konkretnych kolumn.

[77]:
```
df.HP.mean()
```

[77]:
```
69.25875
```

[78]:
```
df.Sp_Atk.max()
```

[78]:
```
194
```

1.7 Funkcje agregujące

Grupujemy pokemony według `Type1` i sumujemy pozostałe cechy.

[79]:
```
df.groupby('Type1').sum()
```

[79]:

Type1	Number	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Generation	Legendary
Bug	23080	3925	4897	4880	3717	4471	4256	222	0.0
Dark	14302	2071	2740	2177	2314	2155	2361	125	2.0
Dragon	15180	2666	3588	2764	3099	2843	2657	124	12.0
Electric	15994	2631	3040	2917	3961	3243	3718	144	4.0
Fairy	7642	1260	1046	1117	1335	1440	826	70	1.0
Fighting	9824	1886	2613	1780	1434	1747	1784	91	0.0
Fire	17025	3635	4408	3524	4627	3755	3871	167	
Flying	2711	283	315	265	377	290	410	22	
Ghost	15568	2062	2361	2598	2539	2447	2059	134	
Grass	24141	4709	5125	4956	5425	4930	4335	235	
Ground	11401	2361	3064	2715	1807	2008	2045	101	
Ice	10165	1728	1746	1714	1861	1831	1523	85	
Normal	31279	7573	7200	5865	5470	6245	7012	299	
Poison	7050	1883	2091	1927	1692	1803	1780	71	
Psychic	21706	4026	4073	3858	5609	4918	4645	193	
Rock	17280	2876	4086	4435	2787	3321	2460	152	
Steel	11957	1761	2503	3412	1823	2177	1492	104	
Water	33946	8071	8305	8170	8379	7898	7388	320	

```
Fire            5.0
Flying          2.0
Ghost           2.0
Grass           3.0
Ground          4.0
Ice             2.0
Normal          2.0
Poison          0.0
Psychic        14.0
Rock            4.0
Steel           4.0
Water           4.0
```

Widzimy, że sumowanie dla kolumn `Number` i `Generation` jest bez sensu. Sumowanie kolumn od `HP` do `Speed` nie daje miarodajnych wyników, ponieważ w każdej podgrupie może znajdować się inna liczba pokemonów. Należałoby te wyniki uśrednić, dzieląc przez liczebności. Sumowanie w kolumnie `Legendary` dało nam informację, ile jest pokemonów legendarnych danego typu, ale także nie możemy porównać tej wartości między podgrupami, ze względu na różną liczebność.

Policzymy ile jest pokemonów danego typu.

[83]:
```
df.groupby('Type1').count().Number
```

[83]:
```
Type1
Bug          69
Dark         31
Dragon       32
Electric     44
Fairy        17
Fighting     27
Fire         52
Flying        4
Ghost        32
Grass        70
Ground       32
Ice          24
Normal       98
Poison       28
Psychic      57
Rock         44
Steel        27
Water       112
Name: Number, dtype: int64
```

Zapiszemy liczebności w wektorze `n`.

[173]:
```
n = df.groupby('Type1').size()
```

[174]:
```
n = list(n)
```

[175]:
```
n
```

[175]:
```
[69, 31, 32, 44, 17, 27, 52, 4, 32, 70, 32, 24, 98, 28, 57, 44, 27, 112]
```

Ustalmy, dla których kolumn sumowanie w ogóle ma sens.

[96]:
```
df.groupby('Type1').sum()[['HP', 'Attack', 'Defense', 'Sp_Atk', 'Sp_Def',␣
↪'Speed', 'Legendary']]
```

[96]:

Type1	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Legendary
Bug	3925	4897	4880	3717	4471	4256	0.0
Dark	2071	2740	2177	2314	2155	2361	2.0
Dragon	2666	3588	2764	3099	2843	2657	12.0
Electric	2631	3040	2917	3961	3243	3718	4.0
Fairy	1260	1046	1117	1335	1440	826	1.0
Fighting	1886	2613	1780	1434	1747	1784	0.0
Fire	3635	4408	3524	4627	3755	3871	5.0
Flying	283	315	265	377	290	410	2.0
Ghost	2062	2361	2598	2539	2447	2059	2.0
Grass	4709	5125	4956	5425	4930	4335	3.0
Ground	2361	3064	2715	1807	2008	2045	4.0
Ice	1728	1746	1714	1861	1831	1523	2.0
Normal	7573	7200	5865	5470	6245	7012	2.0
Poison	1883	2091	1927	1692	1803	1780	0.0
Psychic	4026	4073	3858	5609	4918	4645	14.0
Rock	2876	4086	4435	2787	3321	2460	4.0
Steel	1761	2503	3412	1823	2177	1492	4.0
Water	8071	8305	8170	8379	7898	7388	4.0

[148]:
```
sumy = df.groupby('Type1').sum()[['HP', 'Attack', 'Defense', 'Sp_Atk', 'Sp_Def',␣
↪'Speed', 'Legendary']]
```

[104]:
```
type(sumy)
```

[104]:
```
pandas.core.frame.DataFrame
```

Widzimy, że otrzymaliśmy obiekt typu `Data Frame`.

[140]:
```
nazwy_wierszy = sumy.index
```

[163]:
```
srednie = []
```

[164]:
```
for i in range(0, len(n)):
    srednie.append(sumy.loc[nazwy_wierszy[i]]/n[i])
```

Bardziej elegancki zapis pętli `for`.

[169]:
```
srednie1 = []
```

[170]:
```
for (i, typ) in enumerate(nazwy_wierszy):
    srednie1.append(sumy.loc[typ]/n[i])
```

Policzyliśmy w ten sposób średnie dla wszystkich typów pokemonów. Sprawdzimy, czy zgadzają się z tymi wyliczonymi przez funkcję `mean`.

[165]:
```
srednie[0]
```

[165]:
```
HP           56.884058
Attack       70.971014
Defense      70.724638
Sp_Atk       53.869565
Sp_Def       64.797101
Speed        61.681159
Legendary     0.000000
Name: Bug, dtype: float64
```

[171]:
```
srednie1[0]
```

[171]:
```
HP           56.884058
Attack       70.971014
Defense      70.724638
Sp_Atk       53.869565
Sp_Def       64.797101
Speed        61.681159
Legendary     0.000000
Name: Bug, dtype: float64
```

[168]:
```
df[df.Type1=='Bug'].mean().loc[['HP', 'Attack', 'Defense', 'Sp_Atk', 'Sp_Def',␣
↪'Speed', 'Legendary']]
```

[168]:
```
HP           56.884058
Attack       70.971014
Defense      70.724638
Sp_Atk       53.869565
Sp_Def       64.797101
Speed        61.681159
Legendary     0.000000
dtype: float64
```

Możemy grupować dane według kilku kolumn jednocześnie. Na przykład, pogrupujmy według `Type1` i `Type2`, i policzmy, ile wśród pokemonów danego typu 1 jest pokemonów o określonym typie 2.

[177]:
```
df.groupby(['Type1', 'Type2']).size()
```

[177]:
```
Type1  Type2
Bug    Electric     2
       Fighting     2
       Fire         2
       Flying      14
       Ghost        1
                   ..
Water  Ice          3
       Poison       3
       Psychic      5
       Rock         4
       Steel        1
Length: 136, dtype: int64
```

1.8 Dodawanie kolumny

Utworzymy nową kolumnę, dodamy ją do ramki danych i zapiszemy do pliku.

W nowej kolumnie zsumujemy wszystkie liczbowe parametry pokemona.

[179]:
```
df['Total'] = df.HP + df.Attack + df.Defense + df.Sp_Atk + df.Sp_Def +df.Speed
```

[180]:
```
df.head()
```

[180]:

	Number	Name	Type1	Type2	HP	Attack	Defense	Sp_Atk	\
0	1	Bulbasaur	Grass	Poison	45	49	49	65	
1	2	Ivysaur	Grass	Poison	60	62	63	80	
2	3	Venusaur	Grass	Poison	80	82	83	100	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	
4	4	Charmander	Fire	NaN	39	52	43	60	

	Sp_Def	Speed	Generation	Legendary	Total
0	65	45	1	False	318
1	80	60	1	False	405
2	100	80	1	False	525
3	120	80	1	False	625
4	50	65	1	False	309

Dobrze jest sprawdzić, czy suma się zgadza.

[181]:
```
45+49+49+65+65+45
```

[181]:
```
318
```

Zapisywanie do pliku wykonujemy za pomocą funkcji `to_csv` lub `to_excel`. Ustawiamy parametr `index = False`, aby pozbyć się kolumny z indeksami.

[184]:
```
df.to_csv("poke_modified.csv", index = False)
```

[187]:
```
df.to_csv("poke_modified.txt", index = False, sep = "\t")
```

[188]:
```
df.to_excel("poke_modified.xlsx", index = False)
```

Pliki powinny się pojawić w folderze, w którym pracujemy. Folder ten można zmienić przy użyciu np. polecenia `chdir` (change directory) z biblioteki `os`.

Teraz wybierzemy pokemony typu `Fire` i zapiszemy do pliku ramkę danych z tymi i tylko tymi pokemonami.

[189]:
```
new_df = df[df.Type1 == 'Fire']
```

[190]:
```
new_df.head()
```

[190]:

	Number	Name	Type1	Type2	HP	Attack	Defense	\
4	4	Charmander	Fire	NaN	39	52	43	
5	5	Charmeleon	Fire	NaN	58	64	58	
6	6	Charizard	Fire	Flying	78	84	78	
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	

	Sp_Atk	Sp_Def	Speed	Generation	Legendary	Total
4	60	50	65	1	False	309
5	80	65	80	1	False	405
6	109	85	100	1	False	534
7	130	85	100	1	False	634
8	159	115	100	1	False	634

Widzimy, że indeksy nie uległy zmianie, zatem zresetujemy je (inaczej praca na danych przy użyciu indeksów, np. odwoływanie się do odpowiednich wierszy, będzie problematyczne). `drop = True` usuwa stare indeksy, `inplace = True` oznacza "nie twórz nowego obiektu", zatem zmieniamy indeksy w istniejącej ramce `new_df`.

[193]:
```
new_df.reset_index(drop = True, inplace  = True)
```

Zapisujemy do pliku `.csv`

[195]:
```
new_df.to_csv("poke_only_fire.csv", index = False)
```

1.9 Rysowanie

Dla zapoznania się z biblioteką `matplotlib` (a dokładniej `matplotlib.pyplot`), narysujemy wykres zależności szybkości pokemona od jego HP. Najpierw posortujemy dane względem `HP`.

[196]:
```
df_sorted = df.sort_values('HP')
```

[197]:
```
df_sorted.head()
```

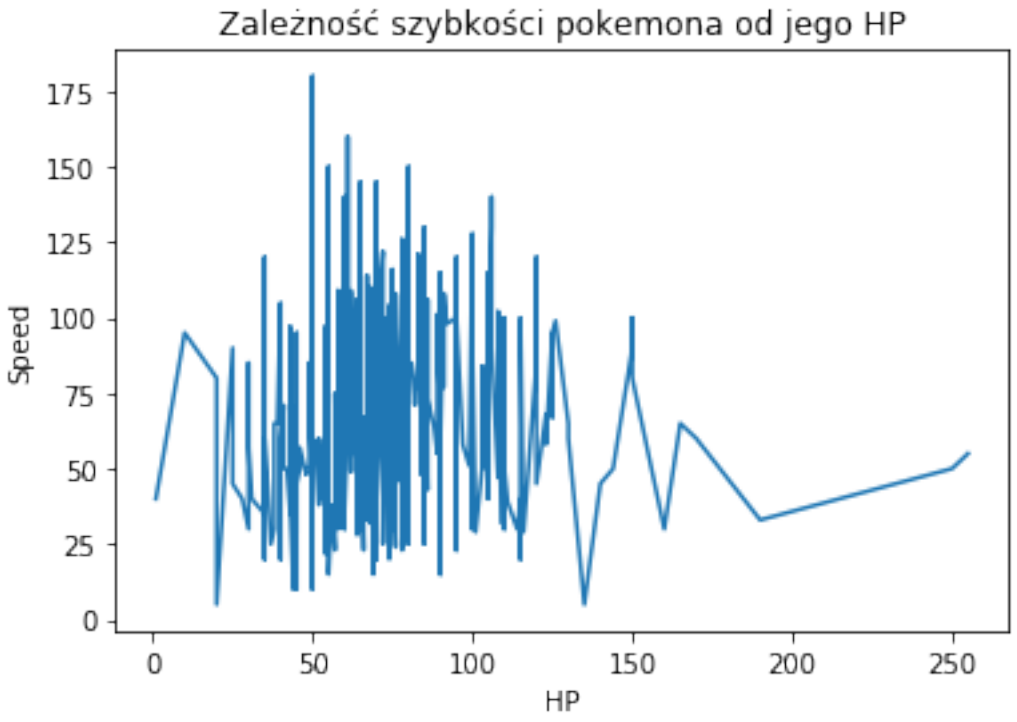
[197]:

	Number	Name	Type1	Type2	HP	Attack	Defense	Sp_Atk	Sp_Def	\
316	292	Shedinja	Bug	Ghost	1	90	45	30	30	
55	50	Diglett	Ground	NaN	10	55	25	35	45	
139	129	Magikarp	Water	NaN	20	10	55	15	20	
388	355	Duskull	Ghost	NaN	20	40	90	30	90	
487	439	Mime Jr.	Psychic	Fairy	20	25	45	70	90	

	Speed	Generation	Legendary	Total
316	40	3	False	236
55	95	1	False	265
139	80	1	False	200
388	25	3	False	295
487	60	4	False	310

Domyślne sortowanie jest rosnące, aby to zmienić, musielibyśmy ustawić `ascending = False`.

[200]:
```
plt.plot(df_sorted['HP'], df_sorted['Speed'])
plt.xlabel('HP')
plt.ylabel('Speed')
plt.title('Zależność szybkości pokemona od jego HP')
plt.show()
```



Co prawda domyślamy się, że wykres nie jest do końca poprawny, ponieważ dla takiego samego `HP` możemy mieć po kilka różnych wartości `Speed`... ale zoabczyliśmy podstawowe komendy służące do rysowania.

1.10 Testy statystyczne

Testy statystyczne można znaleźć np. w bibliotece `scipy.stats`.

Sprawdzimy, czy cecha `HP` ma rozkład normalny w populacji pokemonów.

[203]:
```
W, p = st.shapiro(df.HP)
```

Funkcja `shapiro`, służąca do wykonania testu Shapiro-Wilka, zwraca wartość statystyki `W` oraz `p-value`. Nas będzie interesować przede wszystkim `p-value`.

[204]:
```
p
```

[204]:
```
1.1518300198312678e-20
```

P-value jest znacznie mniejsze od poziomu istotności (standardowo 0.05), zatem musimy odrzucić hipotezę o normalności rozkładu. Jest to ciekawe spostrzeżenie, bo ze względu na dużą liczność próbki można było się spodziewać, że rozkład dąży do normalnego.

[207]:
```
st.normaltest(df.HP).pvalue
```

[207]:
```
7.421639067361935e-67
```

Analogiczny wynik otrzymujemy przy pomocy testu `normaltest`.