PRiAD 1

Przetwarzanie danych w języku Python

1. Dane tabelaryczne - pakiet `pandas`

1.1 Wprowadzenie

Pakiet `pandas` jest jednym z kluczowych pakietów wykorzystywanych w analizie danych. Jego podstawową funkcją jest dostarczenie użytkownikowi efktywnych struktur danych do przechowywania danych wykorzystywanych w eksploracji. W szczególności pakiet oferuje struktury danych **data frame** i **series** , służącą do efektywnego przechowywania i dostępu do odpowiednio macierzy danych i szeregów czasowych. Rozpoczęcia sesji pracy z Pythonem wymaga następującego importu pakietów:

In [8]:
```
import numpy as np
import pandas as pd
```

Podstawową strukturą danych pakietu `pandas` jest macierz danych nazywana tutaj **ramką danych**. Funkcją umożliwiającą tworzenie ramki danych jest funkcja `DataFrame` , której argumenty decydują o formie tworzonej ramki.

In [3]:
```
df0 = pd.DataFrame(np.random.randn(6, 4))
df0
```

Out[3]:

	0	1	2	3
0	0.537252	-0.509326	-0.928483	-2.137860
1	-1.947428	-1.232423	1.110273	0.680328
2	0.023972	1.210799	0.318175	-1.177809
3	-0.006860	0.037508	-0.492197	-0.180420
4	0.406428	1.034886	0.243662	0.519745
5	-1.042372	1.707543	0.128881	0.637226

W tak utworzonej ramce, wiersze (obiekty) oraz kolumny (atrybuty) nie są opisane żadnymi etykietami. Odwołania do poszczególnych można wówczas realizować jedynie poprzez indeksy. Nadanie etykiet atrybutom umożliwia argument `columns` .

In [2]:
```
df = pd.DataFrame(np.random.randn(6, 4), columns=list('ABCD'))
df
```

Out[2]:

	A	B	C	D
0	0.485617	-1.449301	-0.264673	0.573508
1	-0.050150	0.312659	-0.194941	0.342005
2	-0.305805	-1.575256	-0.257962	-0.384226
3	0.546733	0.549862	-0.483756	-1.775870
4	0.147058	-1.267525	0.334363	0.353482
5	0.962670	1.431087	-1.263594	0.520562

Ramka `df` zawiera 5 obiektów (wiersze) opisanych 4 atrybutami (kolumny), etykietami obiektów są kolejne liczby całkowite, zaś etykietami atrybutów są kolejne wielkie litery.

Wiersze (obiekty) także mogą zostać nazwane poprzez nadanie im etykiet.

In [4]:
```
df1 = pd.DataFrame(np.random.randn(6, 4), columns=list('ABCD'), index = ['a','b','c','d','e','f'])
df1
```

Out[4]:

	A	B	C	D
a	-1.842209	1.305783	1.138653	-0.913396
b	1.966700	-0.297468	0.488677	-0.125075
c	-0.659986	-0.780810	-0.601586	-0.650957
d	0.805016	-0.061671	0.576690	0.671351
e	0.750889	-0.091602	1.166743	-2.148529
f	1.687015	1.071921	0.634361	-1.000060

Macierz danych składa się z obiektów opisanych atrybutami. Kolejne atrybuty stanową argumenty funkcji `DataFrame` .

In [5]:
```
df2 = pd.DataFrame({'A': 1.,
   ...:                     'B': pd.Timestamp('20190322'),
   ...:                     'C': pd.Series(1, index=list(range(4)), dtype='float32'),
   ...:                     'D': np.array([3] * 4, dtype='int32'),
   ...:                     'E': pd.Categorical(["test", "train", "test", "train"]),
   ...:                     'F': 'foo'})
df2
```

Out[5]:

	A	B	C	D	E	F
0	1.0	2019-03-22	1.0	3	test	foo
1	1.0	2019-03-22	1.0	3	train	foo
2	1.0	2019-03-22	1.0	3	test	foo
3	1.0	2019-03-22	1.0	3	train	foo

Konkretna ramka danych jest obiektem klasy `DataFrame` . Szereg metod tej klasy udostępnia zarówno dane jak i własności ramki. Metoda `info` zwraca podstawowe informacje o ramce danych.

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
A    6 non-null float64
B    6 non-null float64
C    6 non-null float64
D    6 non-null float64
dtypes: float64(4)
memory usage: 272.0 bytes
```

In [7]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6 entries, a to f
Data columns (total 4 columns):
A    6 non-null float64
B    6 non-null float64
C    6 non-null float64
D    6 non-null float64
dtypes: float64(4)
memory usage: 240.0+ bytes
```

In [8]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 0 to 3
Data columns (total 6 columns):
A    4 non-null float64
B    4 non-null datetime64[ns]
C    4 non-null float32
D    4 non-null int32
E    4 non-null category
F    4 non-null object
dtypes: category(1), datetime64[ns](1), float32(1), float64(1), int32(1), object(1)
memory usage: 260.0+ bytes
```

1.2 Prezentacja danych

Poza trywialnymi przypadkami, ramka danych zwykle składa się z dużej liczby obiektów. Metoda `head` wyświetla początkowe obiekty ramki. Stadardowo jest to 5 pierwszych obiektów.

In [9]:
```
df.head()
```

Out[9]:

	A	B	C	D
0	0.485617	-1.449301	-0.264673	0.573508
1	-0.050150	0.312659	-0.194941	0.342005
2	-0.305805	-1.575256	-0.257962	-0.384226
3	0.546733	0.549862	-0.483756	-1.775870
4	0.147058	-1.267525	0.334363	0.353482

Analogicznie metoda `tail` wyświetla ostatnie obiekty macierzy. Argumentem zarówno tej jak i wcześniejszej funkcji jest liczba wyświetlanych obiektów.

In [10]:
```
df1.tail(3)
```

Out[10]:

	A	B	C	D
d	0.805016	-0.061671	0.576690	0.671351
e	0.750889	-0.091602	1.166743	-2.148529
f	1.687015	1.071921	0.634361	-1.000060

Ramka danych jest opisana nazwami atrybutów i indeksem obiektów. Metoda `index` zwraca indeksy obiektów, zaś metoda `count` liczbę obiektów dla kazdego atrybutu lub atrybutów dla każdego obiektu

In [11]:
```
print("indeks" ,df.index)
ile_obiektow = df.count(0)
ile_atrybutow = df.count(1)
print("obiektów:\n", ile_obiektow, ", atrybutów:\n", ile_atrybutow)
```

```
indeks RangeIndex(start=0, stop=6, step=1)
obiektów:
 A    6
B    6
C    6
D    6
dtype: int64 , atrybutów:
 0    4
1    4
2    4
3    4
4    4
5    4
dtype: int64
```

Metoda `describe` wraca podstawoe statystyki opisowe (miary) atrybutów ramki danych: liczbę obiektów, wartość średnią, odchylenie standardowe, minimum, pierwszy kwartyl, medianę, trzeci kwartyl i maksimum.

In [12]:
```
df.describe()
```

Out[12]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.297687	-0.333079	-0.355094	-0.061756
std	0.457695	1.262615	0.521639	0.907775
min	-0.305805	-1.575256	-1.263594	-1.775870
25%	-0.000848	-1.403857	-0.428986	-0.202668
50%	0.316337	-0.477433	-0.261318	0.347743
75%	0.531454	0.490561	-0.210696	0.478792
max	0.962670	1.431087	0.334363	0.573508

Metoda `T` pozwala na wykoananie transpozycji ramki, w efekcie czego atrybuty stają się obiektami, zaś obiekty - artybutami (stosowac uważnie !).

In [13]:
```
df1.T
```

Out[13]:

	a	b	c	d	e	f
A	-1.842209	1.966700	-0.659986	0.805016	0.750889	1.687015
B	1.305783	-0.297468	-0.780810	-0.061671	-0.091602	1.071921
C	1.138653	0.488677	-0.601586	0.576690	1.166743	0.634361
D	-0.913396	-0.125075	-0.650957	0.671351	-2.148529	-1.000060

Metoda `sort_values` umożliwia sortowanie obiektów.

In [14]:
```
df.sort_values(by='B')
```

Out[14]:

	A	B	C	D
2	-0.305805	-1.575256	-0.257962	-0.384226
0	0.485617	-1.449301	-0.264673	0.573508
4	0.147058	-1.267525	0.334363	0.353482
1	-0.050150	0.312659	-0.194941	0.342005
3	0.546733	0.549862	-0.483756	-1.775870
5	0.962670	1.431087	-1.263594	0.520562

1.3 Dostęp do danych

Istnieje kilka sposobów uzyskiwania dostępu do poszczególnych danych lub fragmentów ramki. Dostęp do poszczególnych atrybutów można uzyskać poprzez podanie ich nazwy, zaś do poszczególnych obiektów - przez podanie zakresu.

In [15]:
```
df['A']
```

Out[15]:
```
0    0.485617
1   -0.050150
2   -0.305805
3    0.546733
4    0.147058
5    0.962670
Name: A, dtype: float64
```

In [16]:
```
df[0:3]
```

Out[16]:

	A	B	C	D
0	0.485617	-1.449301	-0.264673	0.573508
1	-0.050150	0.312659	-0.194941	0.342005
2	-0.305805	-1.575256	-0.257962	-0.384226

Bardziej precyzyjna selekcja fragmentu ramki danych, zawężająca zakres zarówno wierszy jak i kolumn wymaga użycia metod `loc` oraz `iloc`. Metoda `loc` umożliwia odwołanie się poprzez etykiety, zaś `iloc` poprzez pozycję (numer wiersza/kolumny).

In [17]:
```
df.loc[:, ['A', 'B']]
```

Out[17]:

	A	B
0	0.485617	-1.449301
1	-0.050150	0.312659
2	-0.305805	-1.575256
3	0.546733	0.549862
4	0.147058	-1.267525
5	0.962670	1.431087

In [18]:
```
df.loc[1:3, ['A', 'B']]
```

Out[18]:

	A	B
1	-0.050150	0.312659
2	-0.305805	-1.575256
3	0.546733	0.549862

In [19]:
```
df.loc[2, ['A', 'B']]
```

Out[19]:
```
A   -0.305805
B   -1.575256
Name: 2, dtype: float64
```

Jeżeli ramka danych ma przypisane etykiety zarówno do wierszy jak i do kolumn (tak jak w przypadku `df1` powyższy sposób indeksowania wygeneruje błąd).

In [20]:
```
df1.loc[2, ['A', 'B']]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-8580f1b03e9a> in <module>
----> 1 df1.loc[2, ['A', 'B']]

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key
)
   1492             except (KeyError, IndexError, AttributeError):
   1493                 pass
-> 1494             return self._getitem_tuple(key)
   1495         else:
   1496             # we by definition only have the 0th axis

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexing.py in _getitem_tuple(self,
tup)
    866     def _getitem_tuple(self, tup):
    867         try:
--> 868             return self._getitem_lowerdim(tup)
    869         except IndexingError:
    870             pass

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexing.py in _getitem_lowerdim(sel
f, tup)
    986         for i, key in enumerate(tup):
    987             if is_label_like(key) or isinstance(key, tuple):
--> 988                 section = self._getitem_axis(key, axis=i)
    989
    990                 # we have yielded a scalar ?

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, k
ey, axis)
   1910
   1911         # fall thru to straight lookup
-> 1912         self._validate_key(key, axis)
   1913         return self._get_label(key, axis=axis)
   1914

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexing.py in _validate_key(self, k
ey, axis)
   1797
   1798         if not is_list_like_indexer(key):
-> 1799             self._convert_scalar_indexer(key, axis)
   1800
   1801     def _is_scalar_access(self, key):

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexing.py in _convert_scalar_index
er(self, key, axis)
    260         ax = self.obj._get_axis(min(axis, self.ndim - 1))
    261         # a scalar
--> 262         return ax._convert_scalar_indexer(key, kind=self.name)
    263
    264     def _convert_slice_indexer(self, key, axis):

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexes\base.py in _convert_scalar_i
ndexer(self, key, kind)
   2879             elif kind in ['loc'] and is_integer(key):
   2880                 if not self.holds_integer():
-> 2881                     return self._invalid_indexer('label', key)
   2882
   2883         return key

C:\ProgramData\Anaconda3\envs\vis3\lib\site-packages\pandas\core\indexes\base.py in _invalid_indexer(
self, form, key)
   3065                         "indexers [{key}] of {kind}".format(
   3066                             form=form, klass=type(self), key=key,
-> 3067                             kind=type(key)))
   3068
   3069     # --------------------------------------------------------------------

TypeError: cannot do label indexing on <class 'pandas.core.indexes.base.Index'> with these indexers [
2] of <class 'int'>
```

W takim przypadku zarówno wiersze jak i kolumny powinny być indeksowane przez etykiety.

In [21]:
```
print(df1)
df1.loc[['a','c','e'], ['A', 'B']]
```

```
          A         B         C         D
a -1.842209  1.305783  1.138653 -0.913396
b  1.966700 -0.297468  0.488677 -0.125075
c -0.659986 -0.780810 -0.601586 -0.650957
d  0.805016 -0.061671  0.576690  0.671351
e  0.750889 -0.091602  1.166743 -2.148529
f  1.687015  1.071921  0.634361 -1.000060
```

Out[21]:

	A	B
a	-1.842209	1.305783
c	-0.659986	-0.780810
e	0.750889	-0.091602

Indeksowanie poprzez numer wiersz/kolumny jest możliwy przy wykorzystaniu metody `iloc` .

In [22]:
```
df.iloc[3:5, 0:2]
```

Out[22]:

	A	B
3	0.546733	0.549862
4	0.147058	-1.267525

In [23]:
```
df1.iloc[[1, 2, 4], [0, 2]]
```

Out[23]:

	A	C
b	1.966700	0.488677
c	-0.659986	-0.601586
e	0.750889	1.166743

In [24]:
```
df.iloc[1:3, :]
```

Out[24]:

	A	B	C	D
1	-0.050150	0.312659	-0.194941	0.342005
2	-0.305805	-1.575256	-0.257962	-0.384226

In [25]:
```
df.iloc[:, 1:3]
```

Out[25]:

	B	C
0	-1.449301	-0.264673
1	0.312659	-0.194941
2	-1.575256	-0.257962
3	0.549862	-0.483756
4	-1.267525	0.334363
5	1.431087	-1.263594

Metody `loc` praz `iloc` tworzą nową ramkę danych.

In [26]:
```
dff = df.iloc[0:3,1:4]
print(dff,"\n")
print(df.info(),"\n")
print(dff.info())
```

```
          B         C         D
0 -1.449301 -0.264673  0.573508
1  0.312659 -0.194941  0.342005
2 -1.575256 -0.257962 -0.384226

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
A    6 non-null float64
B    6 non-null float64
C    6 non-null float64
D    6 non-null float64
dtypes: float64(4)
memory usage: 272.0 bytes
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
B    3 non-null float64
C    3 non-null float64
D    3 non-null float64
dtypes: float64(3)
memory usage: 152.0 bytes
None
```

Odwołania do elementu poprzez `loc` oraz `iloc` umożliwia także zmianę wartości elementów ramki.

In [27]:
```
print(df,"\n")
df.iloc[1, 1] = 99
df.loc[5,'C'] = -99
df
```

```
          A         B         C         D
0  0.485617 -1.449301 -0.264673  0.573508
1 -0.050150  0.312659 -0.194941  0.342005
2 -0.305805 -1.575256 -0.257962 -0.384226
3  0.546733  0.549862 -0.483756 -1.775870
4  0.147058 -1.267525  0.334363  0.353482
5  0.962670  1.431087 -1.263594  0.520562
```

Out[27]:

	A	B	C	D
0	0.485617	-1.449301	-0.264673	0.573508
1	-0.050150	99.000000	-0.194941	0.342005
2	-0.305805	-1.575256	-0.257962	-0.384226
3	0.546733	0.549862	-0.483756	-1.775870
4	0.147058	-1.267525	0.334363	0.353482
5	0.962670	1.431087	-99.000000	0.520562

Zmienna reprezentująca ramkę danych jest w istocie wskaźnikiem do konkretnego obszaru pamięci. Dlatego przypisanie wartości zmiennej reprezentujej ramkę innej zmiennej jest w istocie przypisaniem wskaźnika. Obie zmienne wskazują na ten sam obszar pamieci. W poniższym przykładzie obie zmienne `df` i `dff` odnoszą się do **tej samej** ramki danych.

In [28]:
```
dff = df
dff.iloc[1,:] = [123, 234, 345, 456]
print(df)
print(dff)
# zawartość jest identyczna
```

```
             A           B           C           D
0     0.485617   -1.449301   -0.264673    0.573508
1   123.000000  234.000000  345.000000  456.000000
2    -0.305805   -1.575256   -0.257962   -0.384226
3     0.546733    0.549862   -0.483756   -1.775870
4     0.147058   -1.267525    0.334363    0.353482
5     0.962670    1.431087  -99.000000    0.520562
             A           B           C           D
0     0.485617   -1.449301   -0.264673    0.573508
1   123.000000  234.000000  345.000000  456.000000
2    -0.305805   -1.575256   -0.257962   -0.384226
3     0.546733    0.549862   -0.483756   -1.775870
4     0.147058   -1.267525    0.334363    0.353482
5     0.962670    1.431087  -99.000000    0.520562
```

Stworzenie faktycznej kopii danych ramki i zapamiętanie ich w innej ramce wymaga użycia metody `copy`. W kolejnym przykładzie `df` oraz `dff` to **różne** ramki danych.

In [29]:
```
dff = df.copy()
dff.iloc[1,:] = [991, 992, 993, 994]
print(df)
print(dff)
# każda ramka ma inną zawartość
```

```
             A           B           C           D
0     0.485617   -1.449301   -0.264673    0.573508
1   123.000000  234.000000  345.000000  456.000000
2    -0.305805   -1.575256   -0.257962   -0.384226
3     0.546733    0.549862   -0.483756   -1.775870
4     0.147058   -1.267525    0.334363    0.353482
5     0.962670    1.431087  -99.000000    0.520562
             A           B           C           D
0     0.485617   -1.449301   -0.264673    0.573508
1   991.000000  992.000000  993.000000  994.000000
2    -0.305805   -1.575256   -0.257962   -0.384226
3     0.546733    0.549862   -0.483756   -1.775870
4     0.147058   -1.267525    0.334363    0.353482
5     0.962670    1.431087  -99.000000    0.520562
```

Selekcja elementów ramki danych może także następować poprzez indeksowanie logiczne. W poniższym przykładzie zwracane są jedynie te obiekty ramki, dla których wartość atrybutu `A` jest większa od 0.

In [30]:
```
df[df.A > 0]
```

Out[30]:

	A	B	C	D
0	0.485617	-1.449301	-0.264673	0.573508
1	123.000000	234.000000	345.000000	456.000000
3	0.546733	0.549862	-0.483756	-1.775870
4	0.147058	-1.267525	0.334363	0.353482
5	0.962670	1.431087	-99.000000	0.520562

Warunki mogą być złożone (atrybuty `A` i `C` większe od 0):

In [31]:
```
df[(df.A > 0) & (df.D > 0)]
```

Out[31]:

	A	B	C	D
0	0.485617	-1.449301	-0.264673	0.573508
1	123.000000	234.000000	345.000000	456.000000
4	0.147058	-1.267525	0.334363	0.353482
5	0.962670	1.431087	-99.000000	0.520562

Warunki filtrowania mozna także nakładać na atrybuty kategoryczne. Utwórzymy nową ramkę danych kopiując poprzednią i dodając piąty atrybut. W przeciwieństwie do czterech wcześnejszych, nowy atrybut jest kategoryczny.

In [32]:
```
df2 = df.copy()
df2['E'] = ['jeden', 'jeden', 'dwa', 'jeden', 'trzy', 'dwa']
df2
```

Out[32]:

	A	B	C	D	E
0	0.485617	-1.449301	-0.264673	0.573508	jeden
1	123.000000	234.000000	345.000000	456.000000	jeden
2	-0.305805	-1.575256	-0.257962	-0.384226	dwa
3	0.546733	0.549862	-0.483756	-1.775870	jeden
4	0.147058	-1.267525	0.334363	0.353482	trzy
5	0.962670	1.431087	-99.000000	0.520562	dwa

In [33]:
```
df2[df2.E == 'jeden']
```

Out[33]:

	A	B	C	D	E
0	0.485617	-1.449301	-0.264673	0.573508	jeden
1	123.000000	234.000000	345.000000	456.000000	jeden
3	0.546733	0.549862	-0.483756	-1.775870	jeden

Selekcja może następować także przez określenie zbioru jego dopuszczalnych wartości.

In [34]:
```
df2[df2['E'].isin(['jeden', 'dwa'])]
```

Out[34]:

	A	B	C	D	E
0	0.485617	-1.449301	-0.264673	0.573508	jeden
1	123.000000	234.000000	345.000000	456.000000	jeden
2	-0.305805	-1.575256	-0.257962	-0.384226	dwa
3	0.546733	0.549862	-0.483756	-1.775870	jeden
5	0.962670	1.431087	-99.000000	0.520562	dwa

1.4 Zewnętrzne źródła danych

Pakiet `pandas` oferuje szereg funkcji umozliwiających wczytywanie ramek danych z różnego rodzaju źródeł. Mogą być nimi np. pliki `csv` .

In [35]:
```
d = pd.read_csv('dane1.csv')
d.head()
```

Out[35]:

	atrybut1	atrybut2	klasa
0	86	43	klasa 3
1	79	50	klasa 3
2	73	49	klasa 3
3	69	49	klasa 3
4	74	43	klasa 3

Informacja o zbiorze

In [36]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 3 columns):
atrybut1    80 non-null int64
atrybut2    80 non-null int64
klasa       80 non-null object
dtypes: int64(2), object(1)
memory usage: 2.0+ KB
```

Podstawowe miary dla całego zbioru.

In [37]:
```
d.describe()
```

Out[37]:

	atrybut1	atrybut2
count	80.000000	80.000000
mean	49.012500	41.275000
std	26.009976	23.377461
min	4.000000	7.000000
25%	29.750000	22.000000
50%	47.500000	34.500000
75%	73.000000	64.250000
max	87.000000	86.000000

I dla pojedynczej klasy.

In [38]:
```
d[d.klasa == 'klasa 1'].describe()
```

Out[38]:

	atrybut1	atrybut2
count	24.000000	24.000000
mean	38.458333	73.250000
std	6.419936	7.531095
min	29.000000	62.000000
25%	32.750000	68.000000
50%	39.500000	72.000000
75%	44.000000	78.250000
max	48.000000	86.000000

Średnia wartość atrybutu 1 w klasie 2.

In [39]:
```
dd = d[d.klasa == 'klasa 2']
np.mean(dd.iloc[:,0:1])
```

Out[39]:
```
atrybut1    13.368421
dtype: float64
```

Ramka danych może zostac zapisana na dysku w formacie zarówno `.csv` . W tym celu należy użyć metody `to_csv` .

In [40]:
```
df.to_csv('przyklad.csv')
print(df.head())
nowa = pd.read_csv('przyklad.csv')
nowa.head()
```

```
             A           B           C           D
0     0.485617   -1.449301   -0.264673    0.573508
1   123.000000  234.000000  345.000000  456.000000
2    -0.305805   -1.575256   -0.257962   -0.384226
3     0.546733    0.549862   -0.483756   -1.775870
4     0.147058   -1.267525    0.334363    0.353482
```

Out[40]:

	Unnamed: 0	A	B	C	D
0	0	0.485617	-1.449301	-0.264673	0.573508
1	1	123.000000	234.000000	345.000000	456.000000
2	2	-0.305805	-1.575256	-0.257962	-0.384226
3	3	0.546733	0.549862	-0.483756	-1.775870
4	4	0.147058	-1.267525	0.334363	0.353482

Zadanie Jednym z najchętniej wykorzystywanych w dydaktyce analizy danych zbiorów danych jest zbiór Fisher's iris (https://en.wikipedia.org/wiki/Iris_flower_data_set). Zbiór tej jest znajduje się w pliku `iris.csv` .

- wczytaj go z dysku do ramki danych
- określ jej parametry: liczbę obiektów, atrybutów, kategorii
- wyznacz średnie wartości atrybutów w kategoriach

In [41]:
```
# miejsce na kod zadania
```

Standardowe zbiory danych można znaleźć w niektórych pakietach języka Python np. w pakiecie `seaborn` (który zostanie wykorzystany w ćwiczeniu poświęconym wizualizacji) oraz `scikit-learn` (ćwiczenia poświęcone uczniu). Pakiety zostaną omówione przy kolejnych okazjach. Poniższe przykłady pokazują jedynie sposób importu zbioru danych `iris` w obu tych pakietach.

In [42]:
```
from seaborn import load_dataset
iris = load_dataset("iris")
iris.head()
```

Out[42]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [43]:
```
from sklearn import datasets
iris = datasets.load_iris()
iris.target.shape
print(iris.data[0:5,:])
print(iris.target[0:5])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0]
```

Format danych zbioru `iris` z pakietu `scikit-learn` nie jest zgodny z formatem pakietu `pandas` . Stosując poznane już narzędzia można jednak łatwo utworzyc stosowną strukturę danych.

In [44]:
```
iris2 = pd.DataFrame(iris.data, columns = ["sepal_length","sepal_width","petal_length","petal_width"])
iris2['species'] = iris.target
iris2.head()
```

Out[44]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Innym popularnym źródłem danych są arkusze kalkulacyjne. Funkcja `read_excel` umożliwia wczytanie arkusza MS Excel.

In [45]:
```
d = pd.read_excel('anscombe.xlsx')
print(d.info())
d.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13 entries, 0 to 12
Data columns (total 10 columns):
Unnamed: 0    0 non-null float64
Unnamed: 1    12 non-null object
Unnamed: 2    12 non-null object
Unnamed: 3    12 non-null object
Unnamed: 4    12 non-null object
Unnamed: 5    12 non-null object
Unnamed: 6    12 non-null object
Unnamed: 7    12 non-null object
Unnamed: 8    12 non-null object
Unnamed: 9    12 non-null object
dtypes: float64(1), object(9)
memory usage: 1.1+ KB
None
```

Out[45]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	Obs.	x1	y1	x2	y2	x3	y3	x4	y4
2	NaN	1	10	8.04	10	9.14	10	7.46	8	6.58
3	NaN	2	8	6.95	8	8.14	8	6.77	8	5.76
4	NaN	3	13	7.58	13	8.74	13	12.74	8	7.71

Przy pomocy dodatkowych argumentów możliwe jest precyzyjne określenie zakresu danych wczytywanych do ramki.

In [46]:
```
d = pd.read_excel('anscombe.xlsx',header = 2,usecols = range(1,10), index_col = 0)
print(d.info())
d
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11 entries, 1 to 11
Data columns (total 8 columns):
x1    11 non-null int64
y1    11 non-null float64
x2    11 non-null int64
y2    11 non-null float64
x3    11 non-null int64
y3    11 non-null float64
x4    11 non-null int64
y4    11 non-null float64
dtypes: float64(4), int64(4)
memory usage: 792.0 bytes
None
```

Out[46]:

Obs.	x1	y1	x2	y2	x3	y3	x4	y4
1	10	8.04	10	9.14	10	7.46	8	6.58
2	8	6.95	8	8.14	8	6.77	8	5.76
3	13	7.58	13	8.74	13	12.74	8	7.71
4	9	8.81	9	8.77	9	7.11	8	8.84
5	11	8.33	11	9.26	11	7.81	8	8.47
6	14	9.96	14	8.10	14	8.84	8	7.04
7	6	7.24	6	6.13	6	6.08	8	5.25
8	4	4.26	4	3.10	4	5.39	19	12.50
9	12	10.84	12	9.13	12	8.15	8	5.56
10	7	4.82	7	7.26	7	6.42	8	7.91
11	5	5.68	5	4.74	5	5.73	8	6.89

Zadanie Wczytaj plik `waluty1.xls` , zawierający kursy trzech walut w pewnym okresie czasu do ramki danych, Zapisz dane w nowej ramce danych, a następnie:

- określ jego paramtery: liczbę obiektów, atrybutów
- określ dla każdej waluty zmiennośc kursu w całym okresie okresie, tj. znajdzie kurs najniższy i najwyższy wskaże daty wystąpienia tych kursów (mogą przydać się funkcje `np.argmin` / `np.argmax` lub `np.idxmin` / `np.idxmax`) oraz policzy różnicę kursową.

In [9]:
```
# miejsce na rozwiązanie zadania
```

Odczyt danych możliwy jest także z pliku `html` znajdującego się pod wskazanym adresem. Poniższy przykład pokazuje pobieranie danych o państwach świata (http://www.worldometers.info/geography/alphabetical-list-of-countries/).

In [10]:
```
import pandas as pd
d = pd.read_html('http://www.worldometers.info/geography/alphabetical-list-of-countries/')
d[0]
```

Out[10]:

	#	Country	Population(2019)	Land Area (Km²)	Density(P/Km²)
0	1	Afghanistan	37209007	652860	57
1	2	Albania	2938428	27400	107
2	3	Algeria	42679018	2381740	18
3	4	Andorra	77072	470	164
4	5	Angola	31787566	1246700	25
5	6	Antigua and Barbuda	104084	440	237
6	7	Argentina	45101781	2736690	16
7	8	Armenia	2936706	28470	103
8	9	Australia	25088636	7682300	3
9	10	Austria	8766201	82409	106
10	11	Azerbaijan	10014575	82658	121
11	12	Bahamas	403095	10010	40
12	13	Bahrain	1637896	760	2155
13	14	Bangladesh	168065920	130170	1291
14	15	Barbados	287010	430	667
15	16	Belarus	9433874	202910	46
16	17	Belgium	11562784	30280	382
17	18	Belize	390231	22810	17
18	19	Benin	11801595	112760	105
19	20	Bhutan	826229	38117	22
20	21	Bolivia	11379861	1083300	11
21	22	Bosnia and Herzegovina	3501774	51000	69
22	23	Botswana	2374636	566730	4
23	24	Brazil	212392717	8358140	25
24	25	Brunei	439336	5270	83
25	26	Bulgaria	6988739	108560	64
26	27	Burkina Faso	20321560	273600	74
27	28	Burundi	11575964	25680	451
28	29	Côte d'Ivoire	25531083	318000	80
29	30	Cabo Verde	560349	4030	139
...	...	...	...	...	...
165	166	Sudan	42514094	1765048	24
166	167	Suriname	573085	156000	4
167	168	Swaziland	1415414	17200	82
168	169	Sweden	10053135	410340	24
169	170	Switzerland	8608259	39516	218
170	171	Syria	18499181	183630	101
171	172	Tajikistan	9292000	139960	66
172	173	Tanzania	60913557	885800	69
173	174	Thailand	69306160	510890	136
174	175	Timor-Leste	1352360	14870	91
175	176	Togo	8186384	54390	151
176	177	Tonga	110041	720	153
177	178	Trinidad and Tobago	1375443	5130	268
178	179	Tunisia	11783168	155360	76
179	180	Turkey	82961805	769630	108
180	181	Turkmenistan	5942561	469930	13
181	182	Tuvalu	11393	30	380
182	183	Uganda	45711874	199810	229
183	184	Ukraine	43795220	579320	76
184	185	United Arab Emirates	9682088	83600	116
185	186	United Kingdom	66959016	241930	277
186	187	United States of America	329093110	9147420	36
187	188	Uruguay	3482156	175020	20
188	189	Uzbekistan	32807368	425400	77

Powyższy przykład pokazuje import danych z prostej tablicy `html` . W przypadku bardziej złożonych danych prawidłowe sformatowanie danych wymaga analizy kodu html oraz odpowiedniego ustawienia argumentów wywołania funkcji `read_html` .

1.6 Łączenie

Ramki danych mogą być na różne sposoby ze sobą łączone.

In [49]:
```
df = pd.DataFrame(np.random.randn(10, 4))
df
```

Out[49]:

	0	1	2	3
0	-0.318846	-0.418990	-0.001081	-1.174568
1	0.796800	-0.082772	0.602547	1.047136
2	0.853615	0.142775	2.228549	-0.738560
3	0.142325	-0.156590	0.195516	-0.343572
4	-0.186083	1.978809	-0.595095	-0.239808
5	-0.821866	-0.951566	0.648175	-1.209138
6	-0.584970	-1.012609	-0.269796	0.750077
7	-0.060005	-0.557839	-0.957297	0.319101
8	-0.308631	0.250326	0.460592	-0.288454
9	-0.338926	-2.289362	-0.866171	-1.540696

Dzielimy na trzy części

In [50]:
```
pieces = [df[7:], df[3:7], df[:3]]
pieces
```

Out[50]:
```
[          0         1         2         3
 7 -0.060005 -0.557839 -0.957297  0.319101
 8 -0.308631  0.250326  0.460592 -0.288454
 9 -0.338926 -2.289362 -0.866171 -1.540696,
           0         1         2         3
 3  0.142325 -0.156590  0.195516 -0.343572
 4 -0.186083  1.978809 -0.595095 -0.239808
 5 -0.821866 -0.951566  0.648175 -1.209138
 6 -0.584970 -1.012609 -0.269796  0.750077,
           0         1         2         3
 0 -0.318846 -0.418990 -0.001081 -1.174568
 1  0.796800 -0.082772  0.602547  1.047136
 2  0.853615  0.142775  2.228549 -0.738560]
```

Funkcja `concat` pozwala na sklejenie kilku ramek danych o takich samych artybutach w jedną.

In [51]:
```
pd.concat(pieces)
```

Out[51]:

	0	1	2	3
7	-0.060005	-0.557839	-0.957297	0.319101
8	-0.308631	0.250326	0.460592	-0.288454
9	-0.338926	-2.289362	-0.866171	-1.540696
3	0.142325	-0.156590	0.195516	-0.343572
4	-0.186083	1.978809	-0.595095	-0.239808
5	-0.821866	-0.951566	0.648175	-1.209138
6	-0.584970	-1.012609	-0.269796	0.750077
0	-0.318846	-0.418990	-0.001081	-1.174568
1	0.796800	-0.082772	0.602547	1.047136
2	0.853615	0.142775	2.228549	-0.738560

Funkcja `merge` pozwala na łączenie ramek w sposób zbliżonych do metod znanych z baz danych.

In [52]:
```
left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})
print(left)
print("\n")
print(right)
pd.merge(left, right, on='key')
```

```
   key  lval
0  foo     1
1  foo     2


   key  rval
0  foo     4
1  foo     5
```

Out[52]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

In [53]:
```
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['foo', 'bar'], 'rval': [4, 5]})
print(left)
print("\n")
print(right)
pd.merge(left, right, on='key')
```

```
   key  lval
0  foo     1
1  bar     2


   key  rval
0  foo     4
1  bar     5
```

Out[53]:

	key	lval	rval
0	foo	1	4
1	bar	2	5

Funkcja `append` pozwala na dodanie nowych obiektów do ramki danych.

In [54]:
```
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
print(df)
print("\n")
s = df.iloc[3]
print(s)
df = df.append(s, ignore_index=True)
print(df)
print("\n")
df = df.append(s)
print(df)
```

```
          A         B         C         D
0 -0.819391  1.178273  1.796685 -0.038548
1 -1.385977 -0.058730  1.552915 -0.823505
2 -0.578786 -1.780740 -0.130782 -0.723072
3  0.250904  0.677396 -2.035268 -0.879723
4  1.308475 -0.676058 -0.775719 -1.541107
5 -0.172335  0.494350  0.925051 -0.513019
6  0.270505 -0.010141  1.839513  0.436675
7 -0.520826 -0.260060 -0.043431  0.393729


A    0.250904
B    0.677396
C   -2.035268
D   -0.879723
Name: 3, dtype: float64
          A         B         C         D
0 -0.819391  1.178273  1.796685 -0.038548
1 -1.385977 -0.058730  1.552915 -0.823505
2 -0.578786 -1.780740 -0.130782 -0.723072
3  0.250904  0.677396 -2.035268 -0.879723
4  1.308475 -0.676058 -0.775719 -1.541107
5 -0.172335  0.494350  0.925051 -0.513019
6  0.270505 -0.010141  1.839513  0.436675
7 -0.520826 -0.260060 -0.043431  0.393729
8  0.250904  0.677396 -2.035268 -0.879723


          A         B         C         D
0 -0.819391  1.178273  1.796685 -0.038548
1 -1.385977 -0.058730  1.552915 -0.823505
2 -0.578786 -1.780740 -0.130782 -0.723072
3  0.250904  0.677396 -2.035268 -0.879723
4  1.308475 -0.676058 -0.775719 -1.541107
5 -0.172335  0.494350  0.925051 -0.513019
6  0.270505 -0.010141  1.839513  0.436675
7 -0.520826 -0.260060 -0.043431  0.393729
8  0.250904  0.677396 -2.035268 -0.879723
3  0.250904  0.677396 -2.035268 -0.879723
```

In [55]:
```
df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
   ....:                'B': ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
   ....:                'C': np.random.randn(8),
   ....:                'D': np.random.randn(8)})
df
```

Out[55]:

	A	B	C	D
0	foo	one	-0.023082	-0.630202
1	bar	one	0.558923	1.309025
2	foo	two	0.300593	-0.284493
3	bar	three	0.842057	0.262256
4	foo	two	-1.142240	-1.176861
5	bar	two	-0.035947	-0.273491
6	foo	one	0.423327	-1.321925
7	foo	three	0.179401	0.399435

Metoda `groupby` pozwala na grupowanie obiektów o takich samych wartościach wybranych atrybutów jednocześnie ustalając wartości pozostałych atrybutów zgodnie z ustalona regułą (np. jako sumę).

In [56]:
```
df.groupby('A').sum()
```

Out[56]:

A	C	D
bar	1.365034	1.297791
foo	-0.262001	-3.014046

In [57]:
```
df.groupby(['A', 'B']).sum()
```

Out[57]:

A	B	C	D
bar	one	0.558923	1.309025
	three	0.842057	0.262256
	two	-0.035947	-0.273491
foo	one	0.400245	-1.952127
	three	0.179401	0.399435
	two	-0.841647	-1.461354

1.7 Szeregi czasowe

Typ danych **series** służy do przechowywania serii danych. Są nimi najczęście szeregi czasowe. Zmienną tego typu tworzymy z wykorzystaniem funkcji `series` , której argument zwiera informację o danych tworzących dany szereg. Informacja ta może być umieszczona w liście zawierającej kolejne wartości.

In [58]:
```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

Out[58]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Specyficzny rodzajem szerego czasowego jest szereg zawierający kolejne daty, który uzyskujemy stosując funkcję `date_range` , której argumentami są dane o szeregu czasowym, np. data początkowa i liczba interwałów czasowych.

In [59]:
```
dates = pd.date_range('20130101', periods=6)

dates
```

Out[59]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

Przy pomocy argumentu `freq` można określić skok czasowy między kolejnymi próbkami.

In [30]:
```
rng = pd.date_range('23/3/2019', periods=100, freq='min')
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
print(ts.head())
print(ts.tail())
```

```
2019-03-23 00:00:00    338
2019-03-23 00:01:00    273
2019-03-23 00:02:00    401
2019-03-23 00:03:00    390
2019-03-23 00:04:00    389
Freq: T, dtype: int32
2019-03-23 01:35:00    193
2019-03-23 01:36:00    176
2019-03-23 01:37:00    450
2019-03-23 01:38:00     87
2019-03-23 01:39:00    303
Freq: T, dtype: int32
```

In [31]:
```
rng = pd.date_range('3/6/2012 00:00', periods=5, freq='D')
ts = pd.Series(np.random.randn(len(rng)), rng)
ts
```

Out[31]:
```
2012-03-06   -0.906725
2012-03-07   -2.463706
2012-03-08    1.207275
2012-03-09    0.589418
2012-03-10    0.600047
Freq: D, dtype: float64
```

2. Dane rastrowe - pakiet `cv2`

Istnieje kilka pakietów umożliwiających wykonywanie operacji na obrazach cyfrowych. Jednym z najważniejszych jest pakiet `cv2`, który umożliwia dostęp z poziomu Pythona do jednej z największych bibliotek przetwarzania obrazów, biblioteki openCV (https://opencv.org/). Innym znanym pakietem wykorystywanym do tego celu jest `scikit-image` (`scimage`).

Jak każdy pakiet, także i `opencv` wymaga wczytania na początku sesji. Dodatkowo jest wczytywana część pakietu `matplotlib` jako niezbędna do wyświetlania obrazów. Pakiet ten służy do wizualizacji danych i zostanie omówiony bardziej wyczerpująco w ćwiczeniu poświęconym wizualizacji.

In [32]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
print(cv2.__version__)
```

```
3.4.2
```

2.1 Wczytywanie i wyświetlanie obrazów

Wczytywanie obrazu do macierzy pakietu `numpy` jest wykonywane z użyciem komendy `imread`. W przypadku gdy otwrcie pliku i ściągnięcie z niego danych się powiedzie, funkcja zwraca macierz - strukturę danych pakietu `numpy`. W zależności od typu obrazu oraz opcji wczytywania obrazu, macierz ta jest jednowymiarowa dla obrazów w skali szarości lub trójwymiarowa - dla obrazów kolorowych. W tym drugim przypadku trzeci wymiar to trzy warstwy odpowiadające poszczególnym składowym koloru.

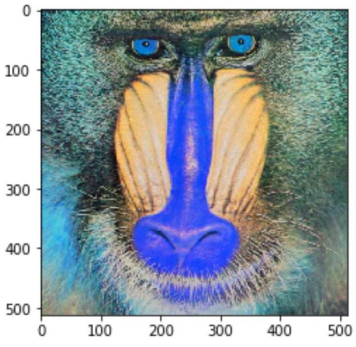
In [33]:
```
obraz = cv2.imread('baboon.jpg')
# parametry macierzy obrazu - obraz kolorowy
print(obraz.shape, type(obraz))
obraz2 = cv2.imread('baboon.jpg',0)
# parametry macierzy obrazu - obraz w skali szarości
print(obraz2.shape, type(obraz2))
```

```
(512, 512, 3) <class 'numpy.ndarray'>
(512, 512) <class 'numpy.ndarray'>
```

Jeden ze sposobów wyświetlenia obrazu (najwygodniejszym w przypadku korzystania z Jupyter Notebook) polega na wykorzystaniu komendy `imshow` z pakietu `matplotlib`.

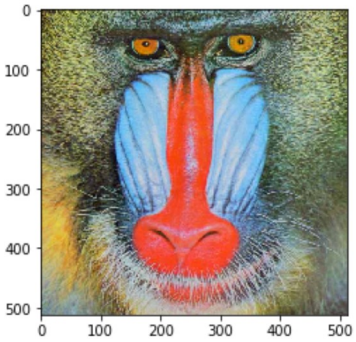
In [34]:
```
# wyświetlanie obrazu kolorowego - błędna kolejnosć składowych
plt.imshow(obraz)
```

Out[34]:
```
<matplotlib.image.AxesImage at 0x20b7fae2f60>
```



In [15]:
```
obraz1 = cv2.cvtColor(obraz,cv2.COLOR_BGR2RGB)
plt.imshow(obraz1)
```

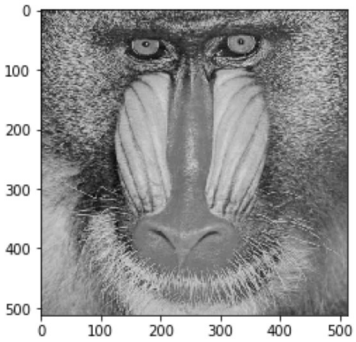
Out[15]:
```
<matplotlib.image.AxesImage at 0x20b7d429f28>
```



Wyświetlenie obrazu w skali szarości wymaga podania palety kolorów w której obraz jest wyświetlany.

In [16]:
```
plt.imshow(obraz2,cmap='gray')
```

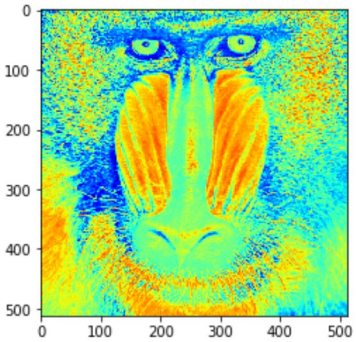
Out[16]:
```
<matplotlib.image.AxesImage at 0x20b7d496048>
```



Przykładowe inne palety kolorów to `Reds` , `Greens` , `Blues` , `plasma` i wiele innych (https://matplotlib.org/tutorials/colors/colormaps.html).

In [17]:
```
plt.imshow(obraz2,cmap='jet')
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x20b7d4eccf8>
```



Wykorzystując funkcję z innego pakietu zawierającego szerego funkcji i procedur obróbki obrazów - `skimage` możliwy jest także import obrazu z sieci, przez podanie jego adresu url.

In [18]:
```
from skimage import io
url = "http://www.ee.pw.edu.pl/wp-content/uploads/2016/11/WE-znak.png"
obraz = io.imread(url)
print(obraz.shape, type(obraz))
plt.imshow(obraz)
```

```
(124, 365, 4) <class 'numpy.ndarray'>
```

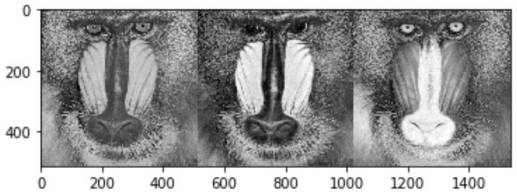
Out[18]:
```
<matplotlib.image.AxesImage at 0x20b7e949e48>
```

2.2 Przestrzenie kolorów

Obraz kolorowy może być reprentowany a także przetwarzany w różnych przestrzeinach kolorów. Najczęściej spotykaną, w której obrazy są wyświetlane jest przestrzeń RGB. Poniższe przykłady przedstawiają separację barwną na trzy składowe przestrzeni RGB, YUV oraz HSV.

In [19]:
```
obraz = cv2.imread('baboon.jpg')
#plt.imshow(cv2.cvtColor(obraz,cv2.COLOR_BGR2RGB))
g = obraz[:,:,0]
r = obraz[:,:,1]
b = obraz[:,:,2]
plt.imshow(np.hstack((r,g,b)),cmap='gray')
```

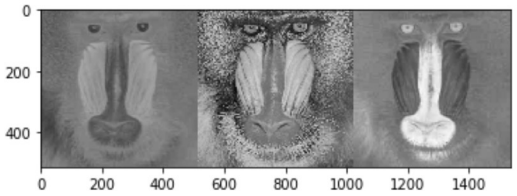
Out[19]:
```
<matplotlib.image.AxesImage at 0x20b7e9b15f8>
```



Przestrzeń YUV

In [21]:
```
obraz1 = cv2.cvtColor(obraz,cv2.COLOR_BGR2YUV)
g = obraz1[:,:,0]
r = obraz1[:,:,1]
b = obraz1[:,:,2]
plt.imshow(np.hstack((r,g,b)),cmap='gray')
```

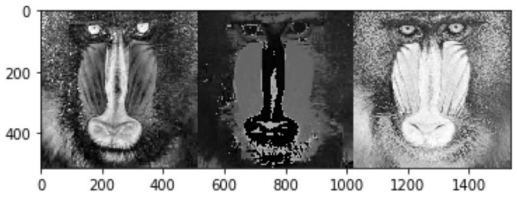
Out[21]:
```
<matplotlib.image.AxesImage at 0x20b7fa24ef0>
```



Przestrzeń HSV

In [23]:
```
obraz1 = cv2.cvtColor(obraz,cv2.COLOR_BGR2HSV)
g = obraz1[:,:,0]
r = obraz1[:,:,1]
b = obraz1[:,:,2]
plt.imshow(np.hstack((r,g,b)),cmap='gray')
```

Out[23]:
```
<matplotlib.figure.AxesImage at 0x20b7fa827f0>
```



Dla dociekliwych

- Pakiet Pandas w 10 minut (https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html)
- Inny tutorial Pandasa (https://data36.com/pandas-tutorial-1-basics-reading-data-files-dataframes-data-selection/)
- I jeszcze jeden (https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python)
- Python+OpenCV (https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html)