PRiAD 2

Wizualizacja danych

1. Pakiety służące do wizualizacji danych

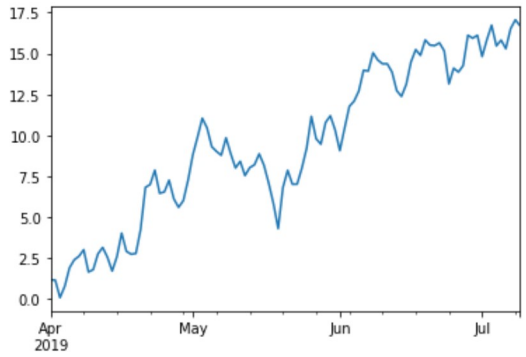
In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import cv2
```

2. Podstawowe wykresy grafiki prezentacyjnej

Wykresy liniowe

In [2]:
```
ts = pd.Series(np.random.randn(100),index=pd.date_range('2019-04-01', periods=100))
ts = ts.cumsum()
ts.plot()
```

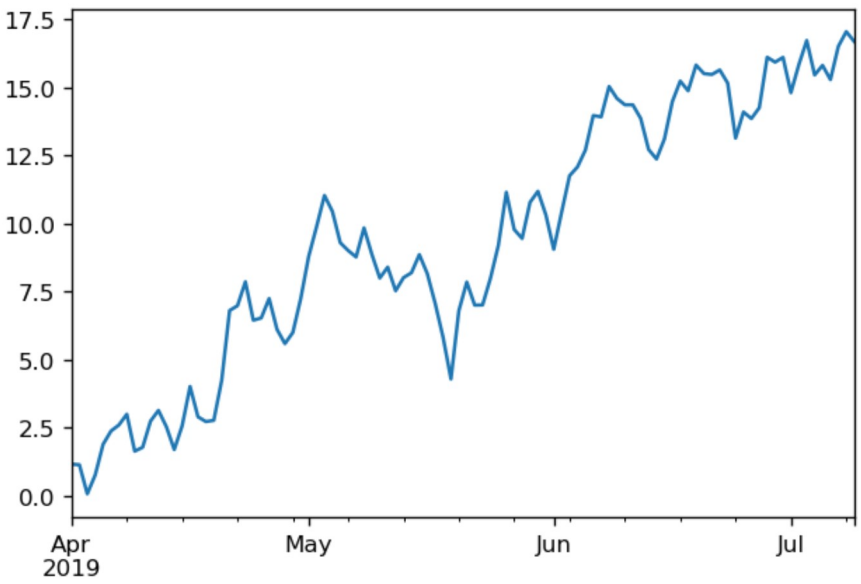
Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x176c0f73ef0>



Rysunek pokazany powyżej nie wypełnia całego dostępnego miejsca w oknie widoku. W celu uzyskania większego wykresu należy określić jego rozdzielczość.

In [3]:
```
plt.figure(dpi = 120)
ts.plot()
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x176c1296080>

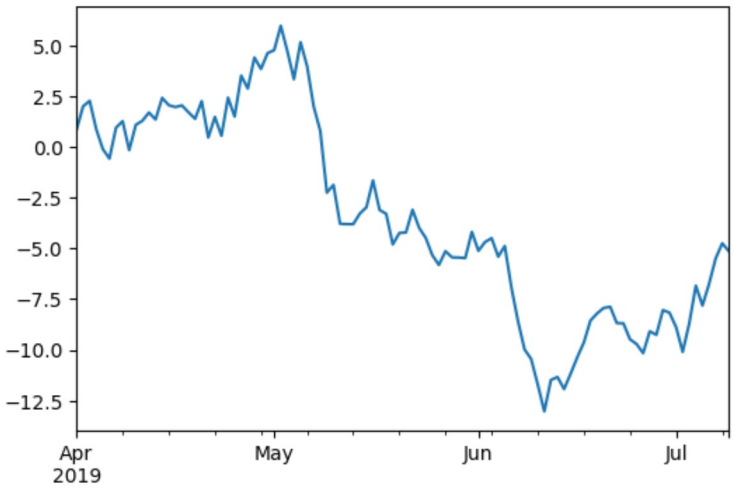


Parametr ten może być także zdefiniowany globalnie. W tym przypadku jego ustawienie pozostaje w mocy także dla kolejnych wykresów.

In [4]:
```
#plt.rcParams['figure.figsize'] = [5, 5]
plt.rcParams['figure.dpi'] = 100
ts = pd.Series(np.random.randn(100),index=pd.date_range('2019-04-01', periods=100))
print(ts.head())
ts = ts.cumsum()
print(ts.head())
ts.plot()
```

```
2019-04-01    0.846709
2019-04-02    1.166780
2019-04-03    0.272121
2019-04-04   -1.404098
2019-04-05   -0.978000
Freq: D, dtype: float64
2019-04-01    0.846709
2019-04-02    2.013489
2019-04-03    2.285610
2019-04-04    0.881512
2019-04-05   -0.096489
Freq: D, dtype: float64
```

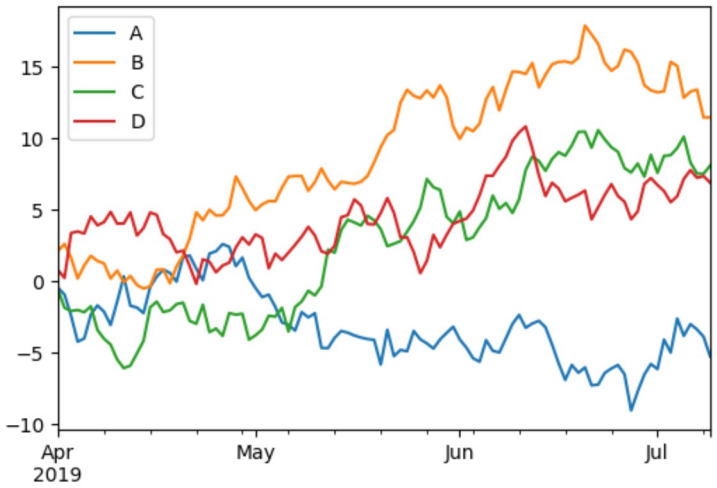
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x176c137a2b0>



Podobnie do wizualizacji szeregu przebiega wizualizacja danych znajdujących się w ramce.

In [5]:
```
df = pd.DataFrame(np.random.randn(100, 4), index=ts.index, columns=list('ABCD'))
print(df.head())
df = df.cumsum()
plt.figure();
df.plot();
```

```
                   A         B         C         D
2019-04-01 -0.463532  2.098750 -0.575122  0.771952
2019-04-02 -0.489541  0.501479 -1.304622 -0.558951
2019-04-03 -1.479883 -0.972029 -0.223766  3.145385
2019-04-04 -1.819973 -1.451251  0.070357  0.111330
2019-04-05  0.224256  0.913758 -0.153366 -0.145105

<Figure size 600x400 with 0 Axes>
```

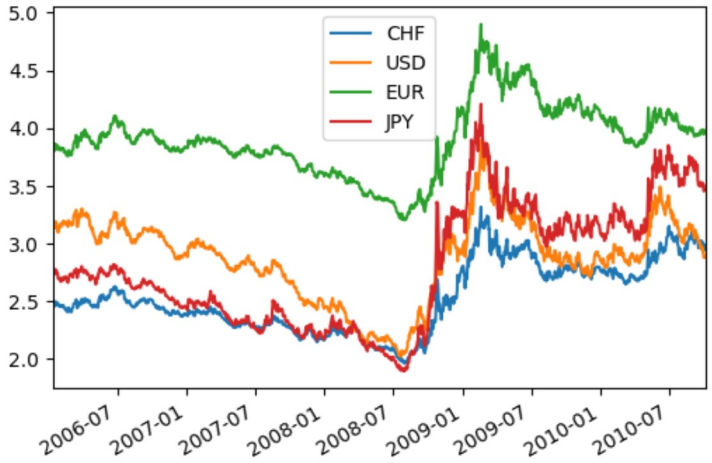


Wykres dla rzeczywistych danych - kursów walut (zwróć uwagę na konwersję danych):

In [6]:
```
w = pd.read_excel('waluty1.xls')
print(w.head())
w.index = pd.to_datetime(w['rok'].map(str) + "-" + w['mies'].map(str) + "-" + w['dzien'].map(str))
w = w.drop(columns=['rok','mies','dzien'])
print("\n",w.head())
w.plot()
```

```
    rok  mies  dzien     CHF     USD     EUR     JPY
0  2010    10      5  2.9809  2.8838  3.9742  3.4617
1  2010    10      4  2.9614  2.8922  3.9577  3.4764
2  2010    10      1  2.9376  2.8772  3.9465  3.4535
3  2010     9     30  2.9955  2.9250  3.9870  3.5129
4  2010     9     29  2.9925  2.9227  3.9710  3.4933

                CHF     USD     EUR     JPY
2010-10-05  2.9809  2.8838  3.9742  3.4617
2010-10-04  2.9614  2.8922  3.9577  3.4764
2010-10-01  2.9376  2.8772  3.9465  3.4535
2010-09-30  2.9955  2.9250  3.9870  3.5129
2010-09-29  2.9925  2.9227  3.9710  3.4933
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176c17527f0>
```



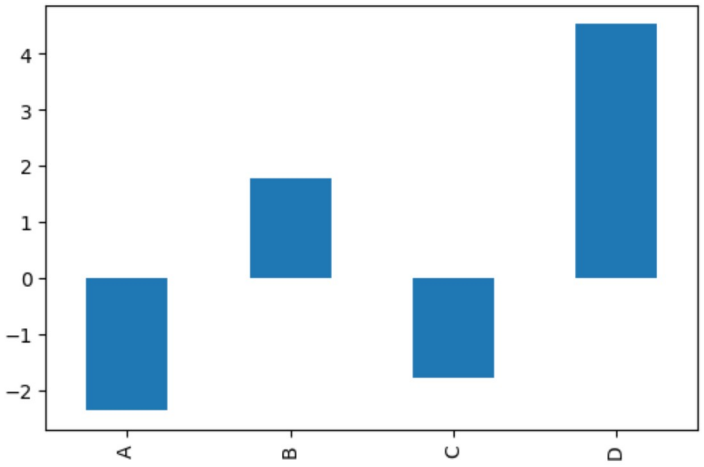
Wyświetl identyczny wykres, ale średnich kursów miesięcznych (wskazówka - użyj metody `groupby`)

In [7]:
```
# rozwiązanie zadania
```

Wykres kolumnowy

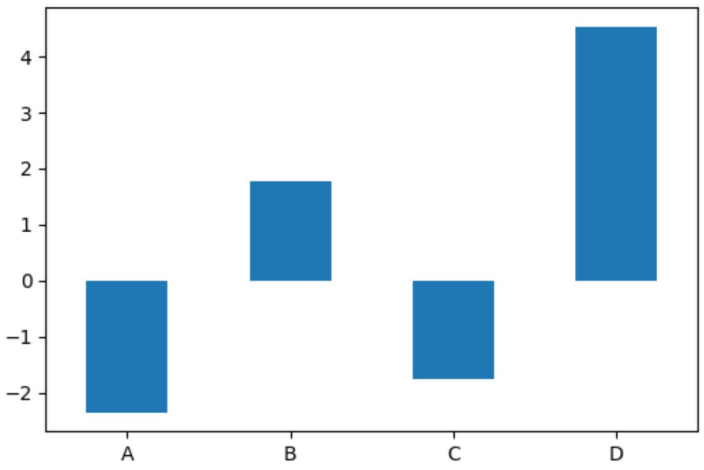
In [8]:
```
print(df.head())
plt.figure();
df.iloc[5].plot(kind='bar');
```

```
                   A         B         C         D
2019-04-01 -0.463532  2.098750 -0.575122  0.771952
2019-04-02 -0.953074  2.600230 -1.879744  0.213000
2019-04-03 -2.432957  1.628201 -2.103510  3.358385
2019-04-04 -4.252930  0.176950 -2.033154  3.469715
2019-04-05 -4.028675  1.090708 -2.186520  3.324610
```



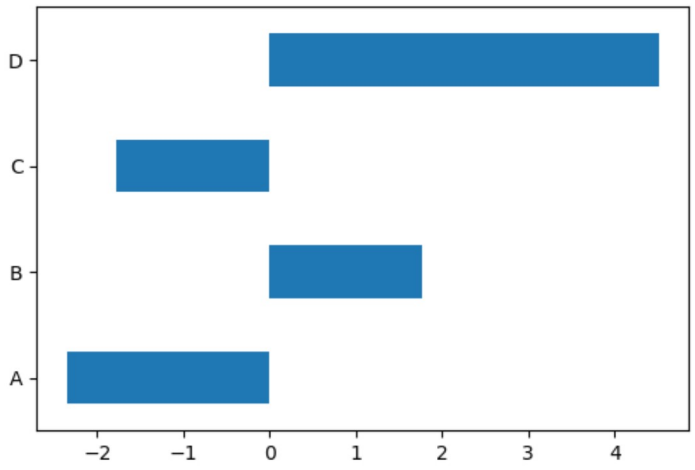
Standardowo etykiety osi x sa odwrócone o 90 stopni. W celu uzyskania ich właściwej orientacji należy użyć argumentu `rot`

In [9]:
```
plt.figure();
df.iloc[5].plot(kind='bar',rot=0);
```



Zmiana typu wykresu na `barh` pozwala na uzyskanie wykresu słupkowego.

In [10]:
```
plt.figure();
df.iloc[5].plot(kind='barh',rot=0);
```



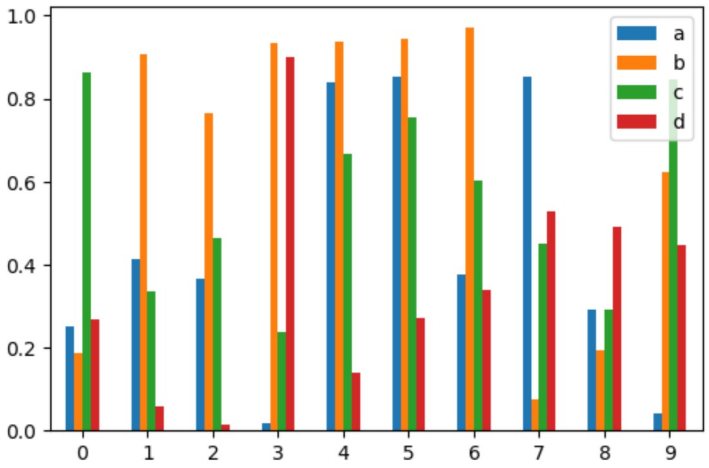
Wyświetl wykres słupkowy pokazujący liczbę ludności wszystkich dużych państw świata tj. liczących więcej niż 35 milionów mieszkańców od najmniejszego do największego (wskazówka - wykonaj kolejno: utworzenie nowej ramki danych zawierającej państwa duże, sortowanie tej ramki względem liczby ludności, wyświetlenie wykresu)

In [11]:
```
# wczytanie i transformacja danych
d = pd.read_html('http://www.worldometers.info/geography/alphabetical-list-of-countries/')
panstwa = d[0]
print(panstwa.head())
panstwa.index = panstwa['Country'].rename('Kraj')
panstwa.columns = ['a','b','Ludnosc','Powierzchnia','e']
panstwa = panstwa.drop(columns=['a','b','e'])
print("\n", panstwa.head())
# rozwiązanie zadania
```

```
   #      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

               Ludnosc  Powierzchnia
Kraj
Afghanistan  37209007        652860
Albania       2938428         27400
Algeria      42679018       2381740
Andorra         77072           470
Angola       31787566       1246700
```

Dla ramek danych o większej liczbie kolumn, bez ograniczania zakresu do jedynie wybranych, wykres kolumnowy zawiera kolumny przyporządkowane poszczególnym atrybutom (kolumnomy ramki danych).

In [12]:
```
df2 = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
df2.plot.bar(rot=0);
```



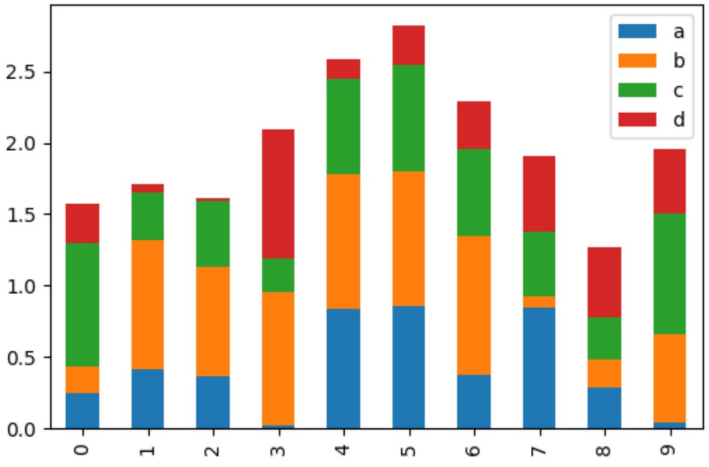
Wyswietl wykres kolumnowy przedstawiający średnie kursy czterech walut w każdym roku na podstawie kursów dziennych zawartych w pliku `waluty1.xls`

In [13]:
```
# rozwiązanie zadania
```

Słupki odpowiadające poszczególnym kategoriom mogą być także umieszczone jeden nad drugim.

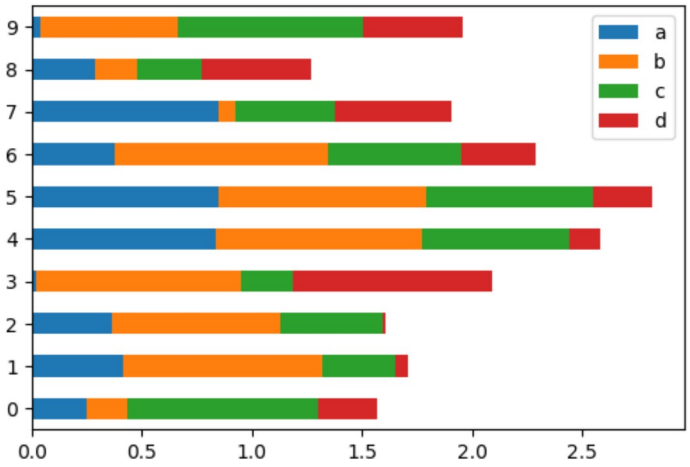
In [14]:
```
print(df2.head())
df2.plot.bar(stacked=True);
```

```
          a         b         c         d
0  0.251959  0.186850  0.863629  0.267148
1  0.413947  0.905282  0.335128  0.058089
2  0.366276  0.765091  0.464431  0.015257
3  0.018801  0.933600  0.236610  0.900492
4  0.839746  0.935728  0.666064  0.140932
```



Wykres może być także odwrócony o 90 stopni - otrzymujemy wówczas wykres słupkowy.

In [15]:
```
df2.plot.barh(stacked=True);
```



Wyświetl analogiczny wykres ale wartości posortowanych - najkrótszy słupek powinien znajdować na górze, a najdłuższy - na dole (wskazówka - stwórz nową kolumnę zawierającą sumę wartości, a następnie posortuj rosnąco względem tej kolumny).

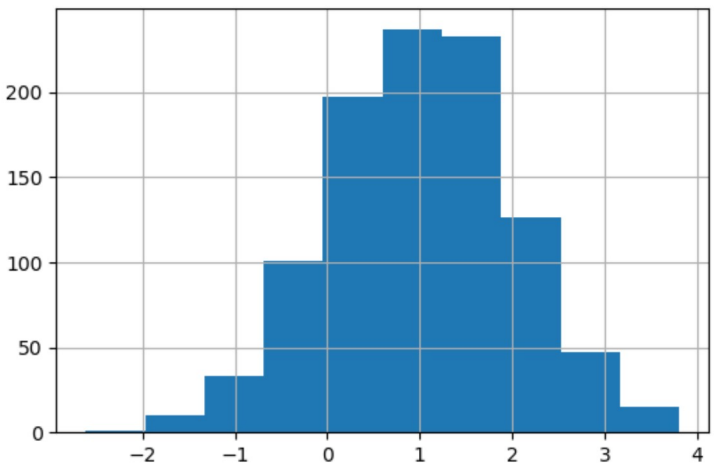
In [16]:
```
# rozwiązanie
```

2.3 Statystyczne wykresy jednowymiarowe - histogramy i wykresy pudełkowe

Histogram jest wykresem częstości występowania poszczególnych wartości w zbiorze. W przypadku atrybutu ilościowego, liczby wystąpień liczy się w ustalonej liczbie zakresów. Dla więszkej liczby atrybutów na pojedynczym wykresie, poszczególne histogramy mogą zostać na siebie nałożone.

In [17]:
```
df4 = pd.DataFrame({'a': np.random.randn(1000) + 1, 'b': np.random.randn(1000),
                    'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
plt.figure();
#df4['a'].plot.hist()
df4['a'].hist()
```

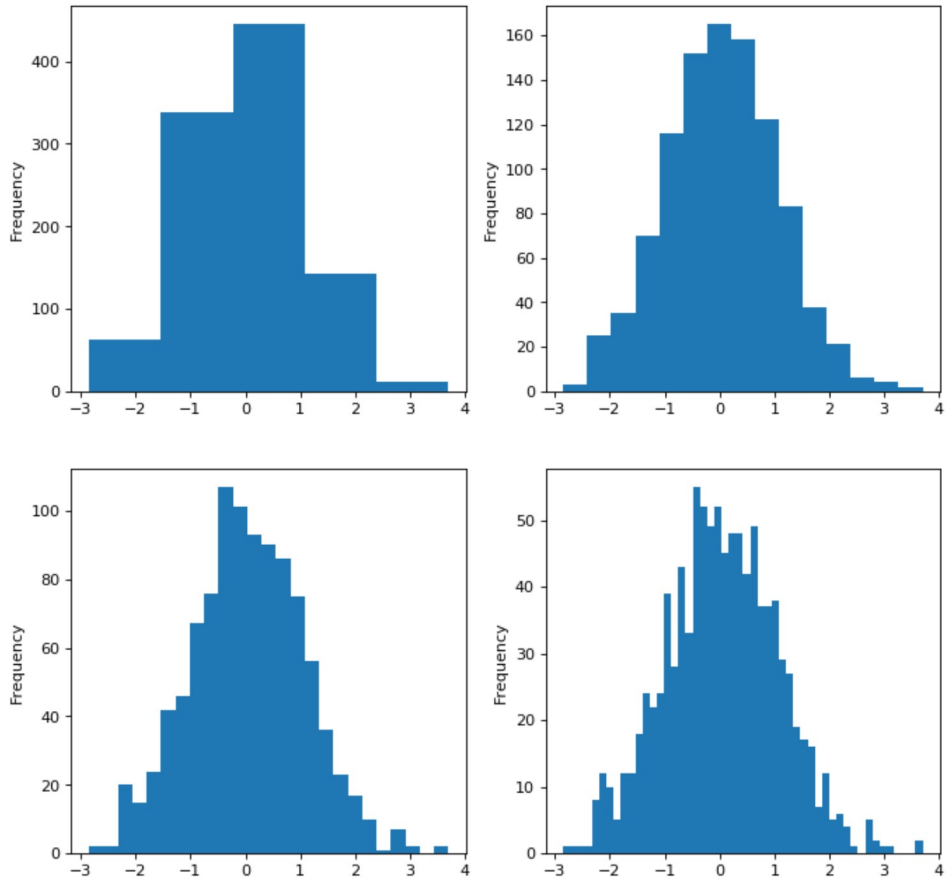
Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176c1cb90f0>
```



Liczbę zakresów ustala się przy pomocy argumentu `bins` .

In [18]:
```
plt.figure(figsize=(10,10), dpi= 80)
plt.subplot(2,2,1)
df4['b'].plot.hist(bins=5)
plt.subplot(2,2,2)
df4['b'].plot.hist(bins=15)
plt.subplot(2,2,3)
df4['b'].plot.hist(bins=25)
plt.subplot(2,2,4)
df4['b'].plot.hist(bins=50)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x176c1dc0128>

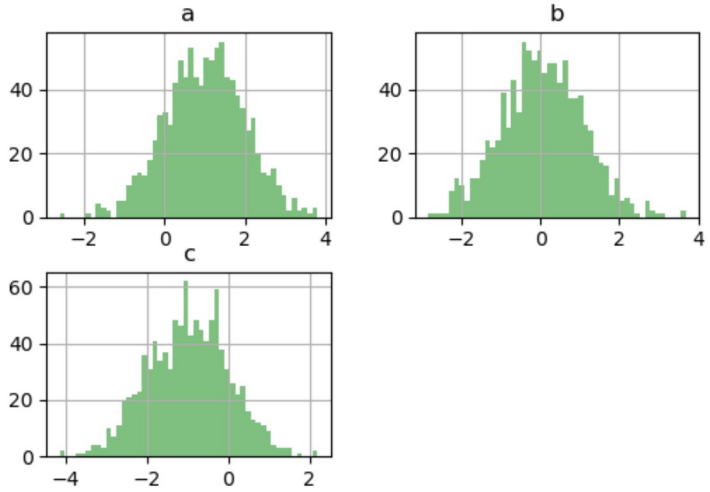


Histogramy większej liczby atrybutów jako odrębne wykresy.

In [19]:
```
print(df4.head())
df4.hist(color='g', alpha=0.5, bins=50)
```

```
          a         b         c
0  1.227359  0.577227 -0.497073
1  0.960023 -0.904056 -0.614453
2  0.597107  1.484918 -1.720856
3  1.617604  0.455074 -0.290833
4  1.881173 -1.181889 -1.203898
```

Out[19]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000176C1DCC4A8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000176C223F630>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000176C1F94BA8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000176C1FC4160>]],
      dtype=object)
```

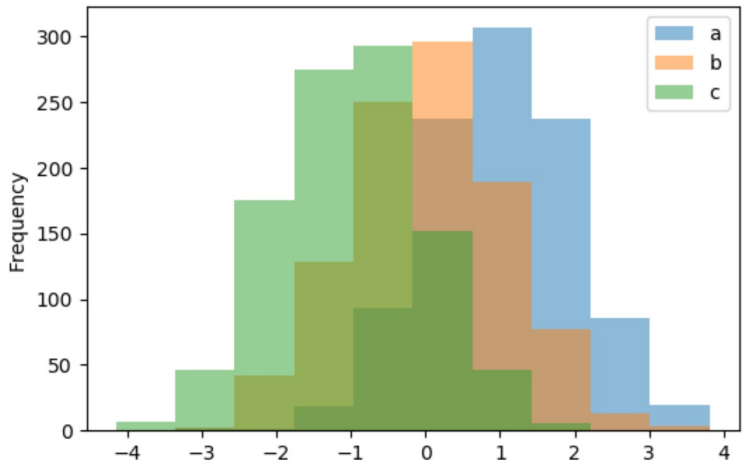


Histogramy większej liczby atrybutów na jednym wykresie

In [20]:
```
print(df4.head())
plt.figure();
df4.plot.hist(alpha=0.5)
```

```
          a         b         c
0  1.227359  0.577227 -0.497073
1  0.960023 -0.904056 -0.614453
2  0.597107  1.484918 -1.720856
3  1.617604  0.455074 -0.290833
4  1.881173 -1.181889 -1.203898
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176c1ceef28>

<Figure size 600x400 with 0 Axes>
```

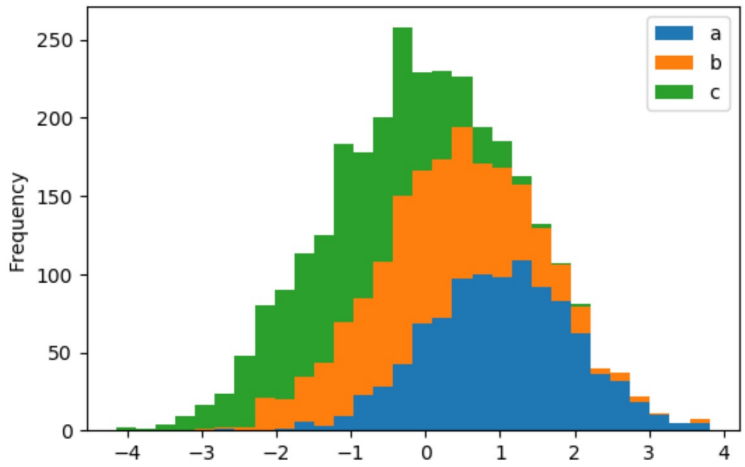


Atrybut `stacked` umożliwia uzyskanie sumarycznego histogramu wszystkich atrybutów.

In [21]:
```
print(df4.head())
plt.figure();
df4.plot.hist(stacked=True, bins=30)
```

```
          a         b         c
0  1.227359  0.577227 -0.497073
1  0.960023 -0.904056 -0.614453
2  0.597107  1.484918 -1.720856
3  1.617604  0.455074 -0.290833
4  1.881173 -1.181889 -1.203898
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176c2186da0>

<Figure size 600x400 with 0 Axes>
```

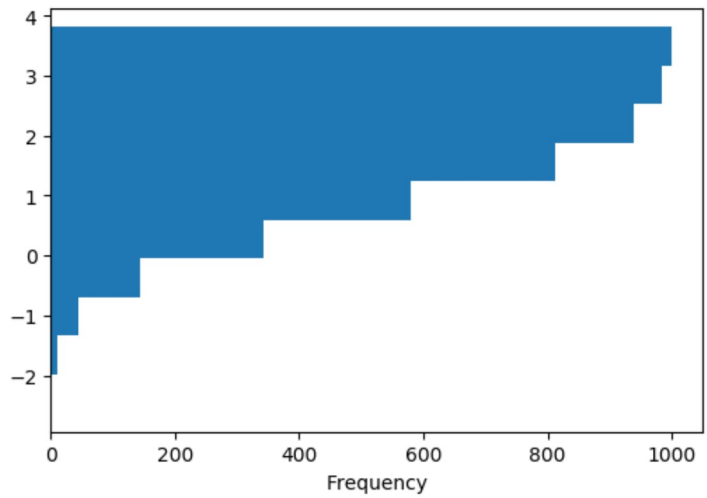


Histogram może zostać wyświetlony w formie obróconej o 90 stopni.

In [22]:
```
print(df4.head())
plt.figure();
df4['a'].plot.hist(orientation='horizontal', cumulative=True)
```

```
          a         b         c
0  1.227359  0.577227 -0.497073
1  0.960023 -0.904056 -0.614453
2  0.597107  1.484918 -1.720856
3  1.617604  0.455074 -0.290833
4  1.881173 -1.181889 -1.203898
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176c20b8eb8>
```

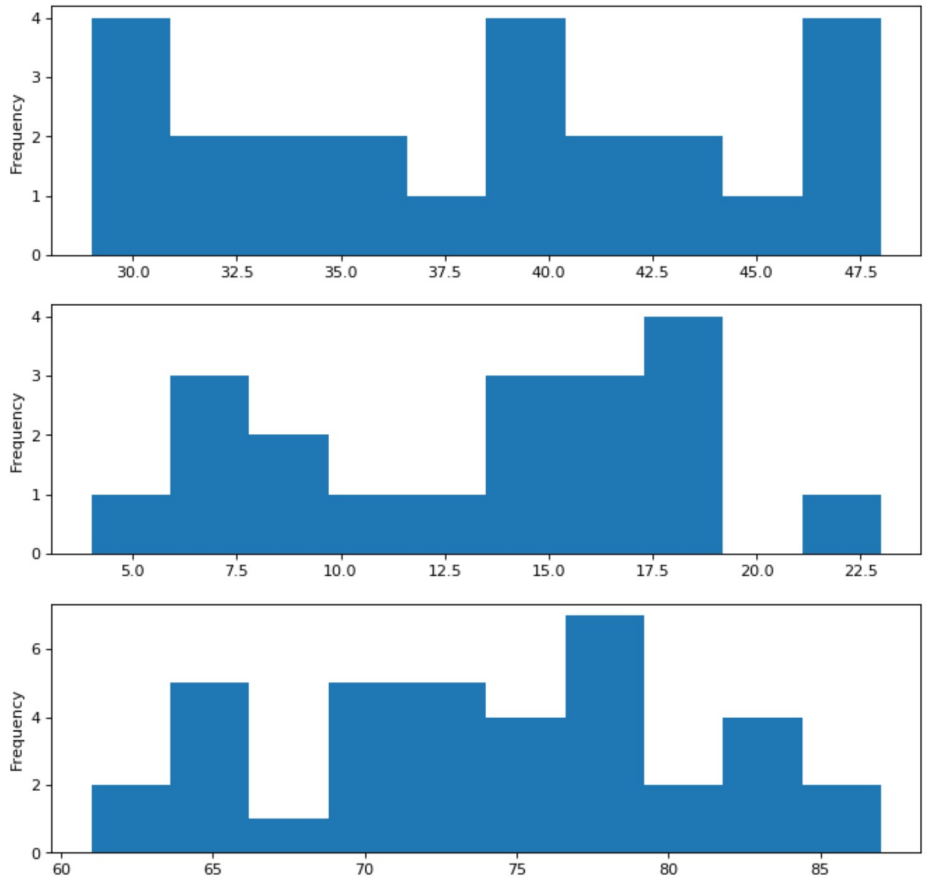


Histogramy uwidaczniają zakresy zmienności atrybutów, co ma duże znaczenie w kontekście określania ich zmienności dla poszczególnych klas.

In [23]:
```
# wczytanie danych
d = pd.read_csv('dane1.csv')
print(d.head())
print(d.tail())

atr = 'atrybut1'
plt.figure(figsize=(10,10), dpi= 80)
plt.subplot(3,1,1)
d.loc[d.klasa == 'klasa 1',atr].plot.hist()
plt.subplot(3,1,2)
d.loc[d.klasa == 'klasa 2',atr].plot.hist()
plt.subplot(3,1,3)
d.loc[d.klasa == 'klasa 3',atr].plot.hist()
```

```
   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
    atrybut1  atrybut2    klasa
75        14        22  klasa 2
76         7        28  klasa 2
77         8        22  klasa 2
78         6        16  klasa 2
79         4        22  klasa 2
```

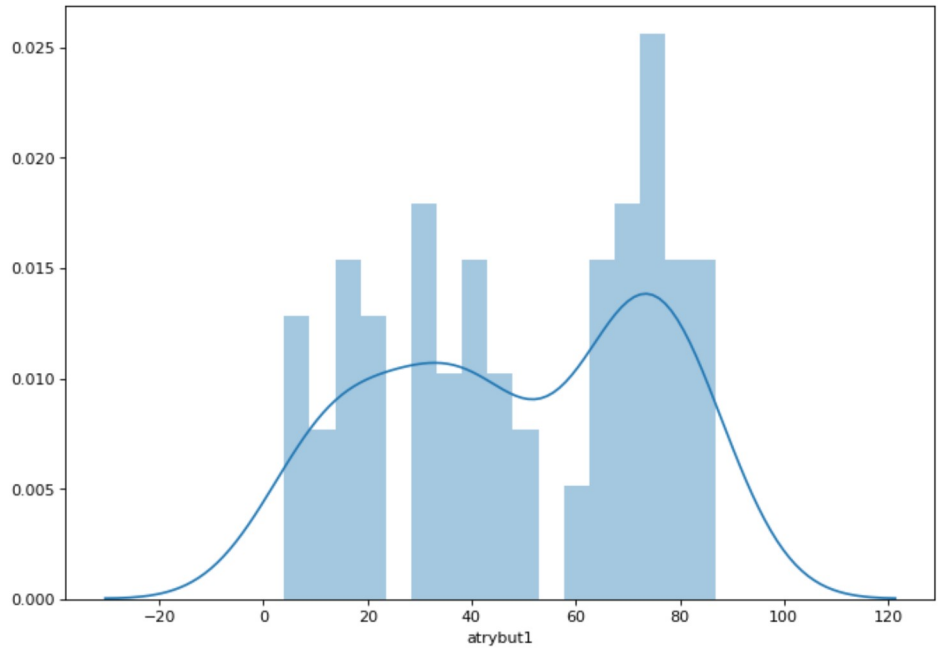
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x176c341a9e8>



Taka prezentacja histogramów klas nie daje czytelnego obrazu charakterystyk poszczególnych klas. Już znacznie bardziej czytelny wykres uzyskujemy wyświetlając histogram bez podziału na klasy.

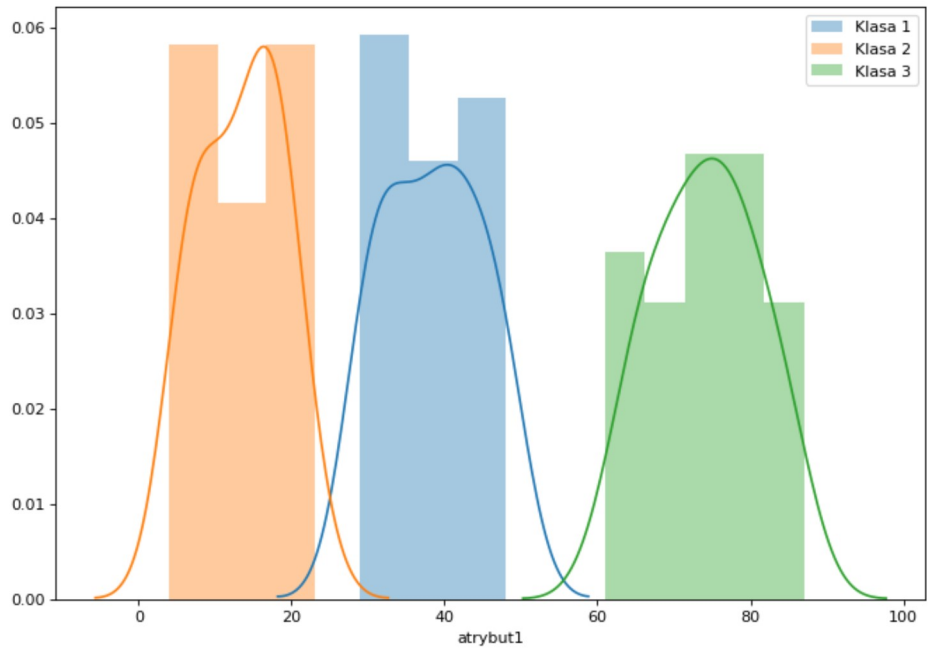
In [24]:
```
df = pd.read_csv('dane1.csv')
atr = 'atrybut1'
plt.figure(figsize=(10,7), dpi= 80)
sns.distplot(df.loc[:,atr], bins = 17)
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176c348b358>
```



Wyraźnie większą czytelność uzyskujemy wyznaczając histogramy dla każdej klasy z osobna.

In [25]:
```
df = pd.read_csv('dane1.csv')
atr = 'atrybut1'
plt.figure(figsize=(10,7), dpi= 80)
sns.distplot(df.loc[df.klasa == 'klasa 1',atr], label="Klasa 1")
sns.distplot(df.loc[df.klasa == 'klasa 2',atr], label="Klasa 2")
sns.distplot(df.loc[df.klasa == 'klasa 3',atr], label="Klasa 3")
plt.legend();
```



Dwa histogramy jeden pod drugim.

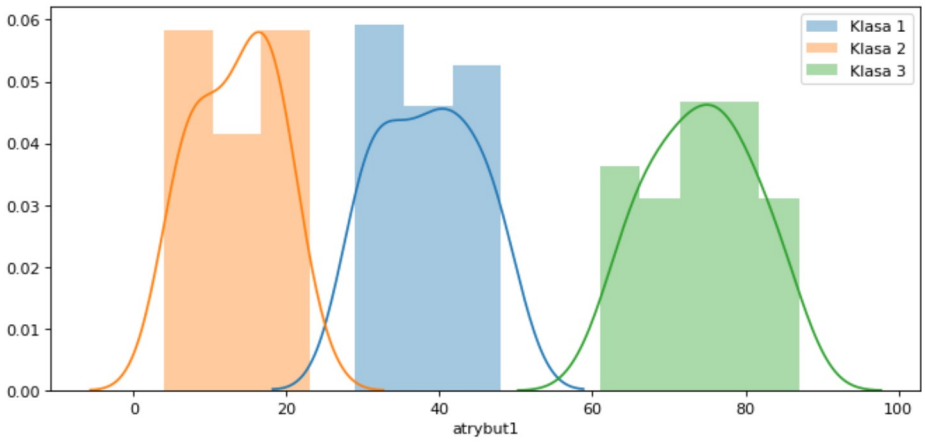
In [26]:
```
daneX = 'dane1.csv'

def pokaz_jeden_atrybut(atr):

    sns.distplot(df.loc[df.klasa == 'klasa 1',atr], label="Klasa 1")
    sns.distplot(df.loc[df.klasa == 'klasa 2',atr], label="Klasa 2")
    sns.distplot(df.loc[df.klasa == 'klasa 3',atr], label="Klasa 3")
    plt.legend();

df = pd.read_csv(daneX)
print(df.info())
for kl in ['klasa 1','klasa 2','klasa 3']:
    print("\n",kl)
    print(df[df.klasa == str(kl)].describe())
plt.figure(figsize=(10,10), dpi= 80)
plt.subplot(2,1,1)
atr = 'atrybut1'
pokaz_jeden_atrybut(atr)
plt.subplot(2,1,2)
atr = 'atrybut2'
pokaz_jeden_atrybut(atr)
```

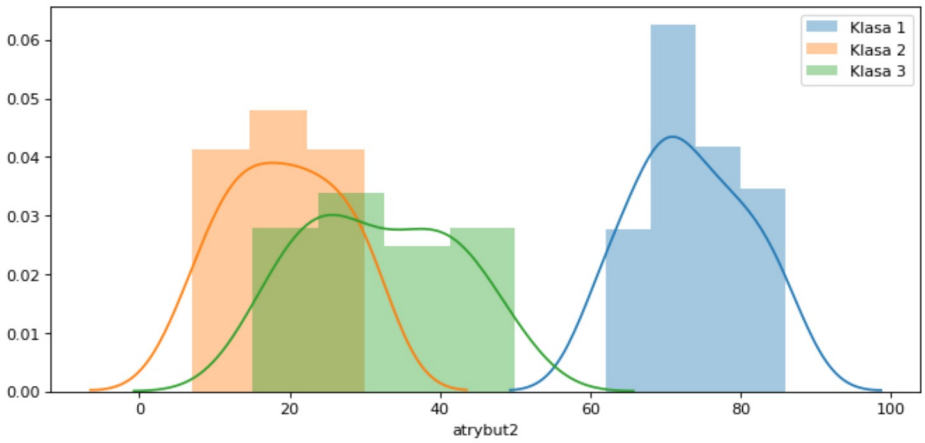
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 3 columns):
atrybut1    80 non-null int64
atrybut2    80 non-null int64
klasa       80 non-null object
dtypes: int64(2), object(1)
memory usage: 2.0+ KB
None
```

```
 klasa 1
         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
```

```
 klasa 2
         atrybut1   atrybut2
count  19.000000  19.000000
mean   13.368421  19.210526
std     5.469212   7.663615
min     4.000000   7.000000
25%     8.500000  13.500000
50%    14.000000  20.000000
75%    18.000000  25.000000
max    23.000000  30.000000
```

```
 klasa 3
         atrybut1   atrybut2
count  37.000000  37.000000
mean   74.162162  31.864865
std     6.958263  10.203927
min    61.000000  15.000000
25%    69.000000  24.000000
50%    74.000000  31.000000
75%    79.000000  41.000000
max    87.000000  50.000000
```





Oceń na podstawie histogramów przydatność atrybutów do rozróżnienia klasy - który z dwóch atrybutów w większym stopniu determinuje klasę obiektu ? Dokonaj podobnej analizy dla pozostałych zbiorów 'daneX.csv' (X = {2,3,4,5,6,7}) - o czym świadczą różnice w histogramach tych zbiorów ?

Narysuj histogramy poszczególnych atrybutów zbioru danych `iris` z podziałem na klasy (setosa, virginica, versicolor) i wykonaj taką samą (jak w zadaniu powyżej) analizę.

In [27]:
```
# wczytanie danych
iris = pd.read_csv('iris.csv', usecols = range(1,6))
iris.head()
# rozwiązanie
```

Out[27]:

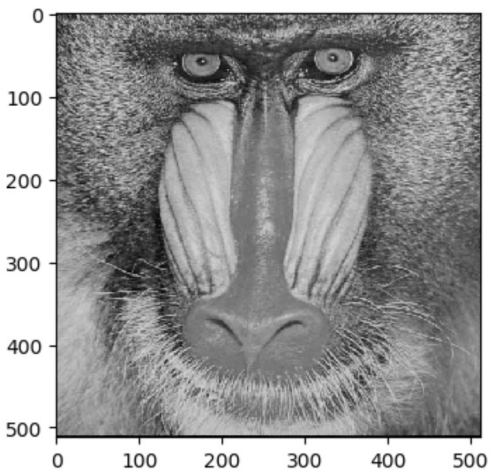
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Napisz funkcję wyświetlającą statystyki opisowe klas oraz histogramy atrybutów z podziałem na klasy dla DOWOLNEJ ramki danych o atrybutach ilościowych (dowolna liczba/nazwy klas, dowolna liczba/nazwy atrybutów).

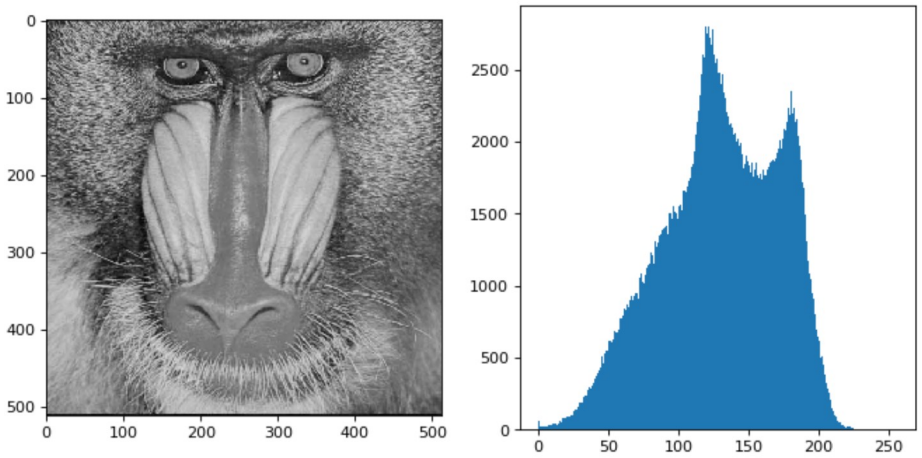
Histogram jest wykorzystywane także do określania własności obrazu cyfrowego.

In [28]:
```
obraz = cv2.imread('baboon.jpg',0)
plt.imshow(obraz,cmap='gray')
```

Out[28]: `<matplotlib.image.AxesImage at 0x176c3667a20>`

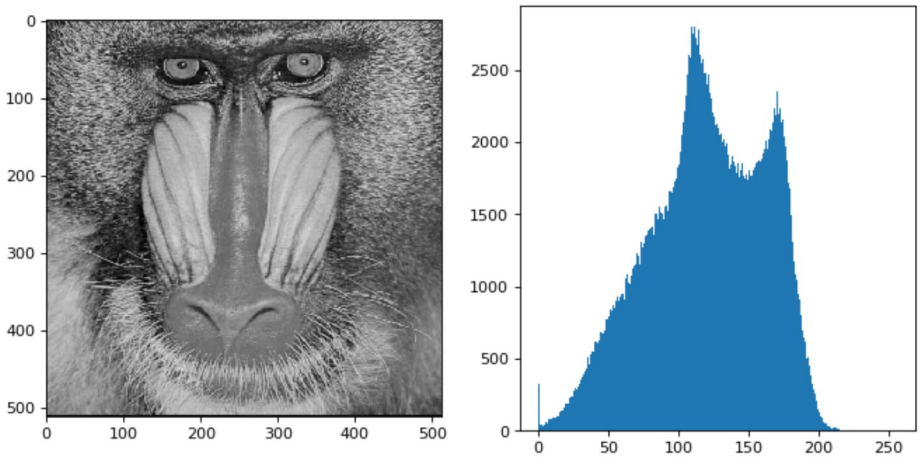


In [29]:
```
plt.figure(figsize=(10,5), dpi= 80)
plt.subplot(1,2,1)
plt.imshow(obraz,cmap='gray')
plt.subplot(1,2,2)
plt.hist(obraz.ravel(),256,[0,256]); plt.show()
```



Zmiany jasności obrazu wpływają na histogram.

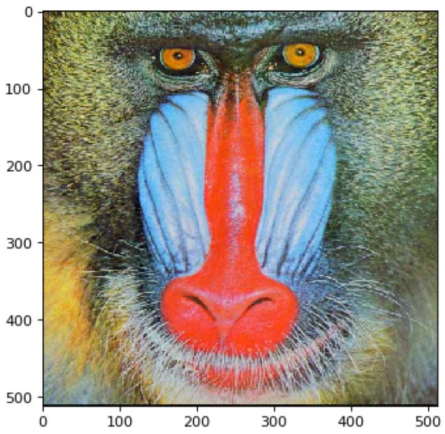
In [30]:
```
zmiana_jasnosci = -10
obraz2 = cv2.add(obraz,zmiana_jasnosci)
plt.figure(figsize=(10,5), dpi= 80)
plt.subplot(1,2,1)
plt.imshow(obraz2,cmap='gray')
plt.subplot(1,2,2)
plt.hist(obraz2.ravel(),256,[0,256]); plt.show()
```

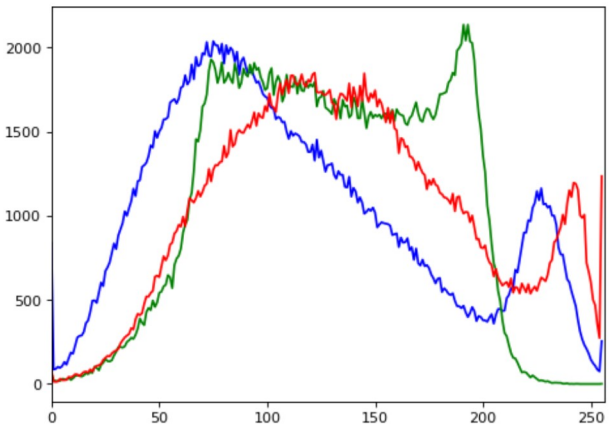


Poeksperymentuj ze zmianami wartości zmiennej `zmiana_jasnosci` (dodatnimi i ujemnymi). Zaobserwuj jak zmienia się histogram obrazu.

W przypadku obrazu kolorowego najczęściej wyznaczane są histogramy poszczególnych składowych.

In [31]:
```
obraz_rgb = cv2.imread('baboon.jpg')
color = ('b','g','r')
plt.figure(figsize=(15,5), dpi= 80)
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(obraz_rgb,cv2.COLOR_BGR2RGB))
plt.subplot(1,2,2)
for i,col in enumerate(color):
    histr = cv2.calcHist([obraz_rgb],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```



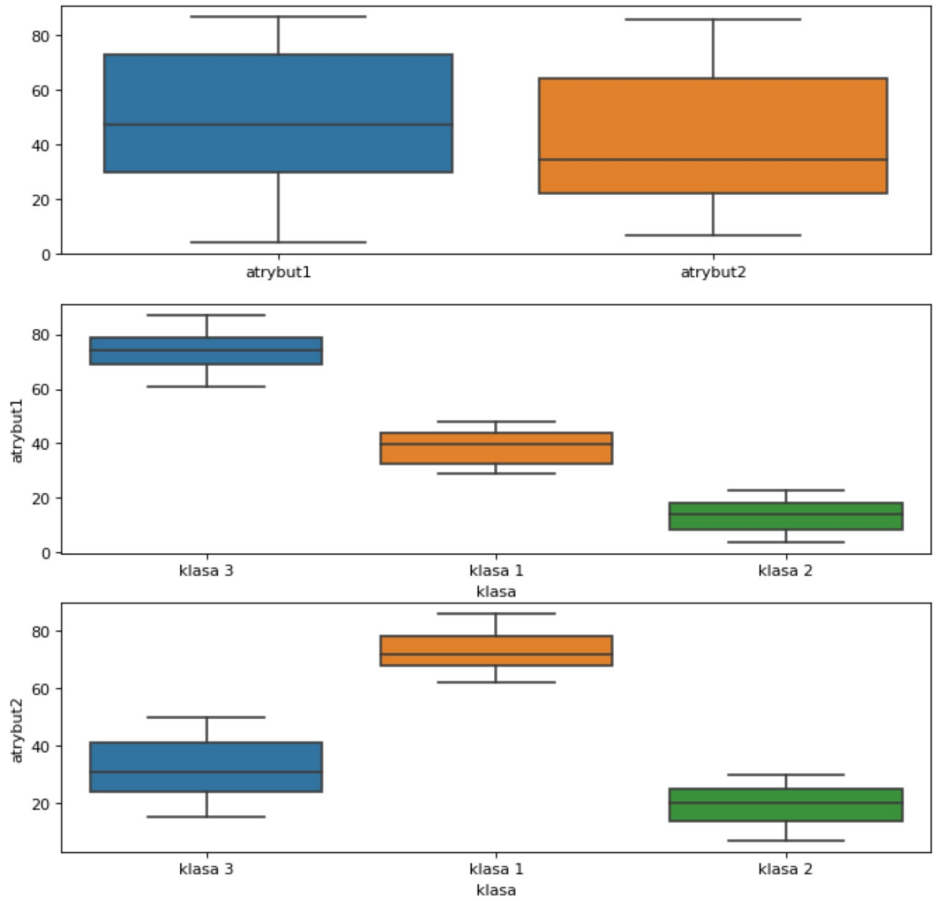


Obejrzyj histogram obrazu `kostka.bmp` . Skąd mogą wynikać różnice w wyglądzie histogramu obu obrazów ?

Wykresy pudełkowe umożliwiają wyświetlanie zakresów zmienności atrybutów, pokazując podstawoe miary pozycyjne (kwartyle).

In [32]:
```
df = pd.read_csv('dane1.csv')
plt.figure(figsize=(10,10), dpi= 80)
plt.subplot(3,1,1)
sns.boxplot(data=df)
plt.subplot(3,1,2)
sns.boxplot(x="klasa", y="atrybut1", data=df)
plt.subplot(3,1,3)
sns.boxplot(x="klasa", y="atrybut2", data=df)
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x176c3fa3400>

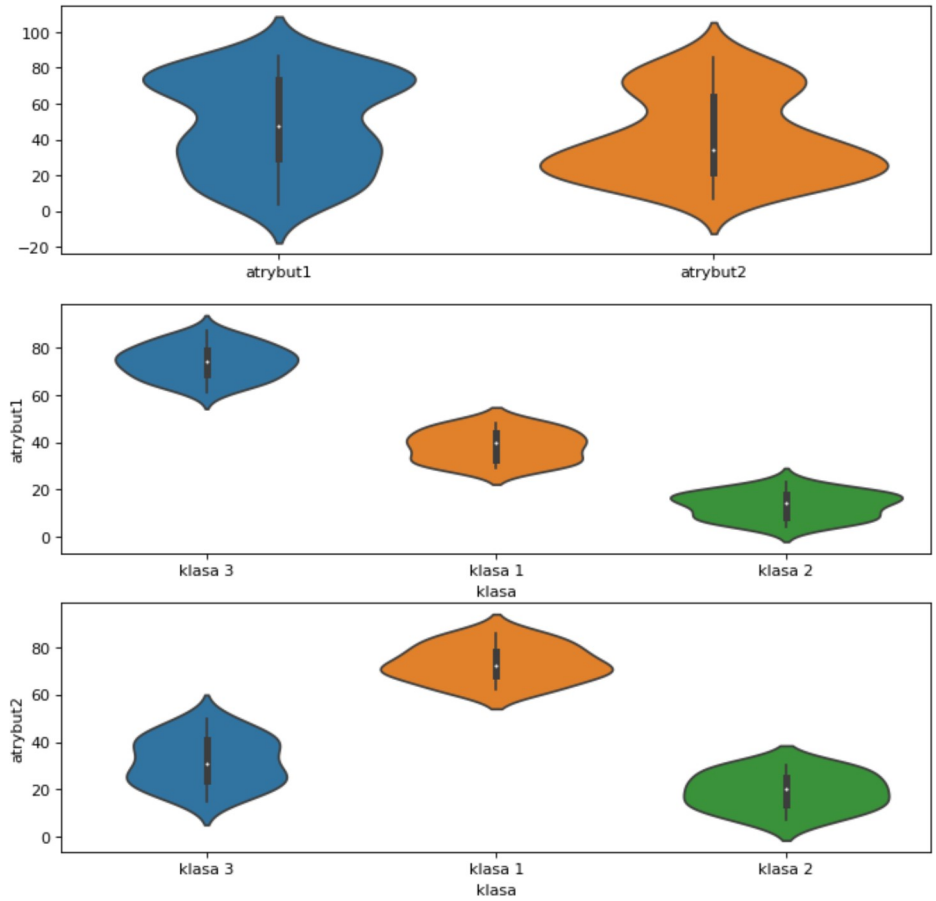


Wyświetl wykresy pudekowe dla dla drugiego atrybutu i dla kolejnych zbiorów `daneX` X = {2,3,4,,5,6,7}. Jakie wnioski mógłbys wyciągnąć z analizy tych wykresów ?

Ciekawą kombinacją histogramu i wykresu pudełkowego jest tzw. "wykres skrzypcowy", pokazujący jednocześnie dystrybucję (histogram) oraz miary kwartylowe.

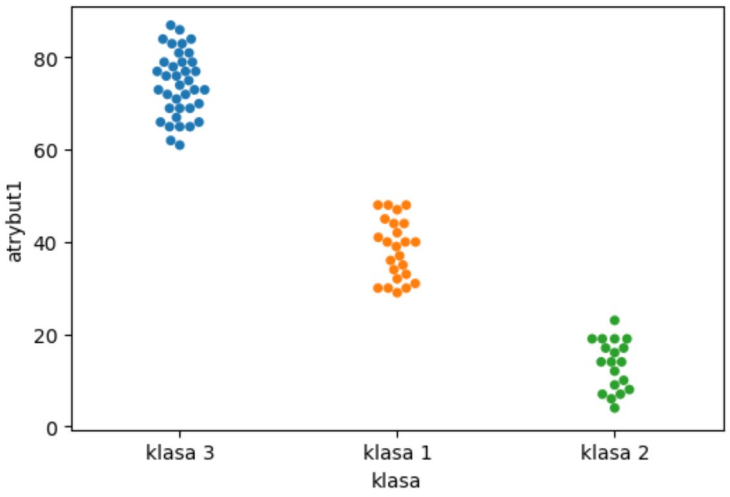
In [33]:
```
df = pd.read_csv('dane1.csv')
plt.figure(figsize=(10,10), dpi= 80)
plt.subplot(3,1,1)
sns.violinplot(data=df)
plt.subplot(3,1,2)
sns.violinplot(x="klasa", y="atrybut1", data=df)
plt.subplot(3,1,3)
sns.violinplot(x="klasa", y="atrybut2", data=df)
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x176c41822e8>



Inny wariant wykresu.

In [34]:
```
sns.swarmplot(x="klasa", y="atrybut1", data=df)
plt.show()
```



Obejrzyj zbiór `iris` korzystając z wykresów pudełkowego i skrzypcowego. Wyciągnij wnioski.

In [35]:
```
# rozwiązanie zadania
```

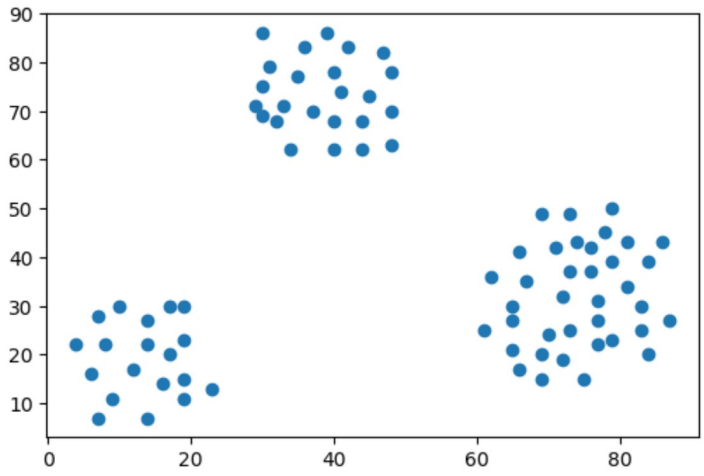
2.4 Wykresy statystyczne wielowymiarowe - wykresy punktowe

Najpopularniejszym wykresem pozwalającym na obserwację większej liczby atrybutów, w tym nie tylko ich zmienności, ale także i wzajemnych **zależności** jest wykres punktowy zwany także wykresem rozrzutu.

Najprostszy wykres punktowy prezentuje zależnosć dwóch atrybutów.

In [36]:
```
# wczytanie danych
d = pd.read_csv('dane1.csv')
print(d.head())
print(d.tail())
plt.plot( 'atrybut1', 'atrybut2', data=df, linestyle='none', marker='o')
plt.show()
```

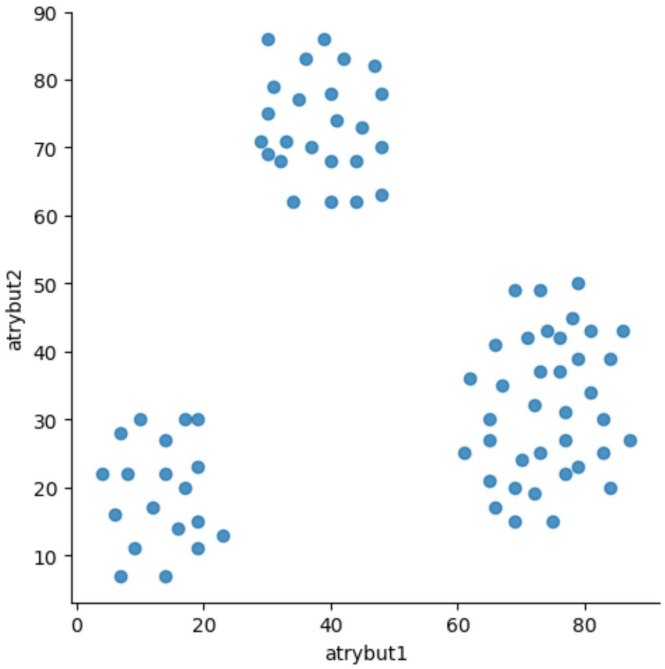
```
   atrybut1  atrybut2    klasa
0        86        43  klasa 3
1        79        50  klasa 3
2        73        49  klasa 3
3        69        49  klasa 3
4        74        43  klasa 3
    atrybut1  atrybut2    klasa
75        14        22  klasa 2
76         7        28  klasa 2
77         8        22  klasa 2
78         6        16  klasa 2
79         4        22  klasa 2
```



Inna wersja (seaborn)

In [37]:
```
sns.lmplot( x="atrybut1", y="atrybut2", data=d, fit_reg=False, legend=False)
```

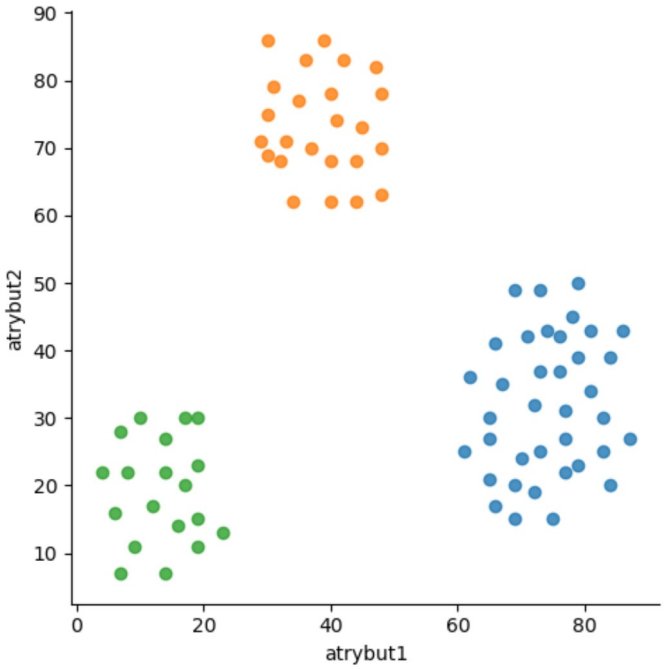
Out[37]:
```
<seaborn.axisgrid.FacetGrid at 0x176c3f23390>
```



Jeśli obiekty są przyporządkowane do zadanych klas (a tak jest w przypadku zbioru danych `dane1.csv`), to przynależność tam może zostać oznaczona odpowiednim kolorem.

In [38]:
```
sns.lmplot( x="atrybut1", y="atrybut2", data=d, fit_reg=False, hue='klasa', legend=False)
```

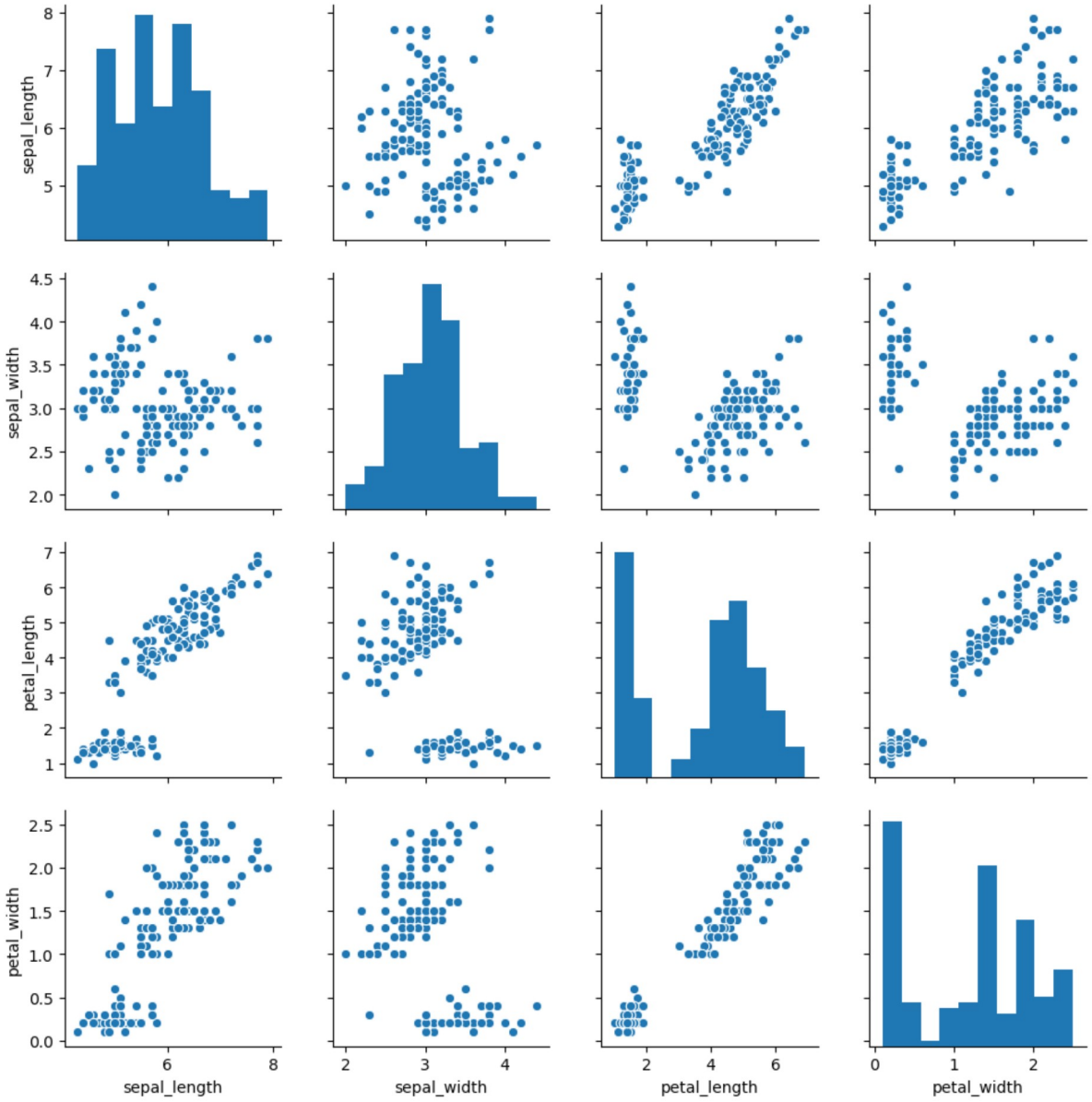
Out[38]:
```
<seaborn.axisgrid.FacetGrid at 0x176c3f23278>
```



Obejrzyj macierze wykresów punktowych zbiorów `dane1.csv` ... `dane11.csv` . Czym różnią się te zbiory ?

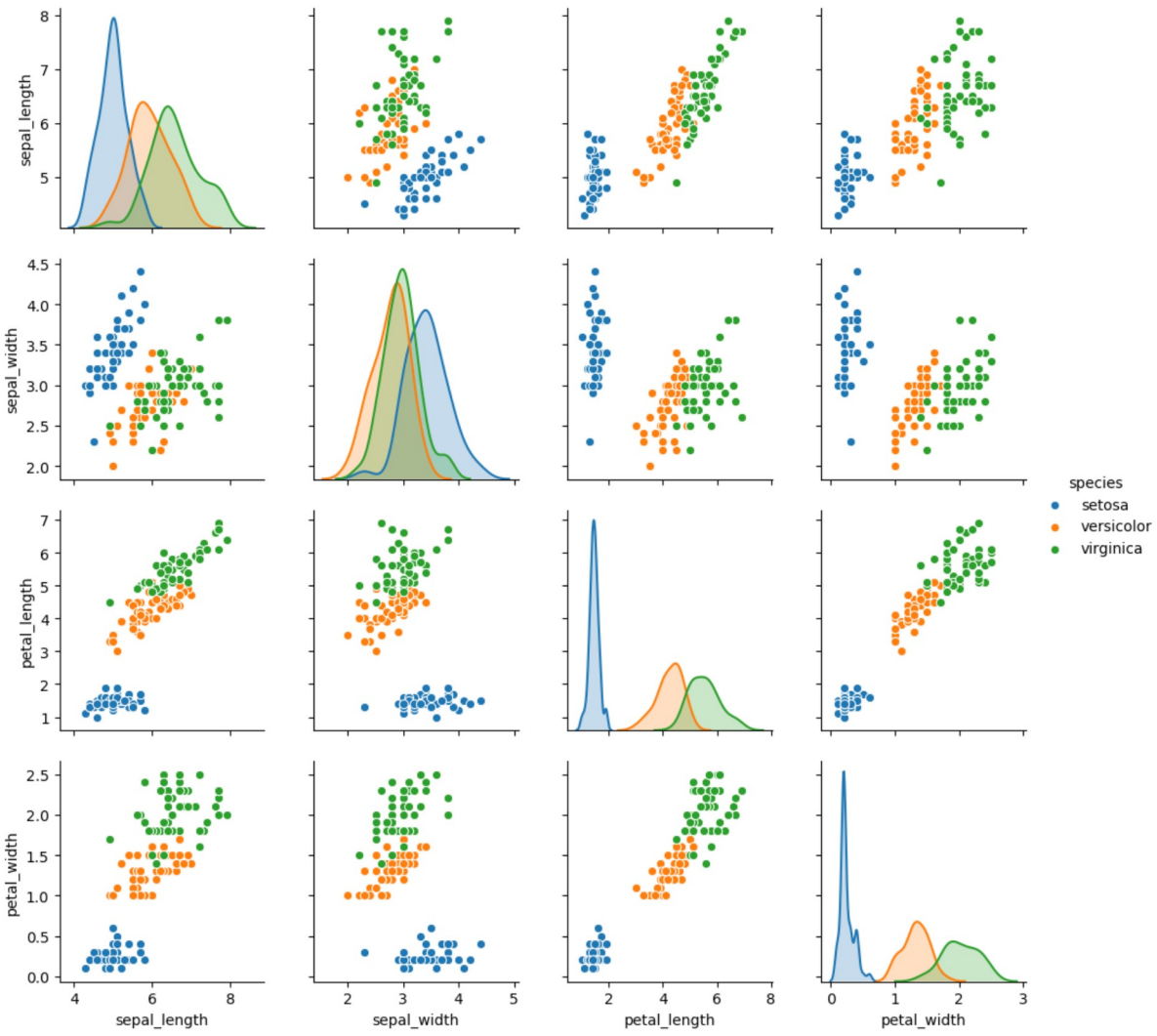
W przypadku większej liczby atrybutów, wyznacza się macierz wykresów punktowych, w której znajdują się wykresy punktowe par atrybutów.

In [39]:
```
i = pd.read_csv('iris.csv', usecols = range(1,6))
sns.pairplot(i, kind="scatter")
plt.show()
```



Wersja z oznaczonymi klasami

In [40]:
```
i = pd.read_csv('iris.csv', usecols = range(1,6))
sns.pairplot(i, kind="scatter", hue = "species")
plt.show()
```



Jak sądzisz, które cechy botanicy biorą pod uwagę rozróżniając poszczególne odmiany kwiatów irysa ?

Dla dociekliwych

- Wykresy w Pythonie (https://python-graph-gallery.com/)
- Histogramy (https://towardsdatascience.com/histograms-and-density-plots-in-python-f6bda88f5ac0)
- Histogramy obrazów (https://docs.opencv.org/3.1.0/d1/db7/tutorial_py_histogram_begins.html)
- Wykresy pudełkowe (https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51)
- Wizualizacja zbioru iris (http://www.learn4master.com/machine-learning/visualize-iris-dataset-using-python)