Analiza danych - projekt zaliczeniowy

January 9, 2020

PROJEKT ZALICZENIOWY - ANALIZA DANYCH

AUTORZY: M. J, K. S, T. P

OPIS:

Analizowane dane dotyczą własności fizycznych sześciu typów gwiazd: brązowych karłów, czerwonych karłów, białych karłów, gwiazd ciągu głównego, supergigantów oraz hipergigantów. W analizowanych danych typy te zostały odpowiednio oznaczone przez liczby od 0 do 5. Plik CSV z danymi pobrany został ze strony www.kaggle.com i zawiera pomiary dla 240 gwiazd. Dane zawierają parametry takie jak: temperaturę powierzchni, jasność (względem Słońca), promień (względem Słońca), absolutną wielkość gwiazdową, typ gwiady, kolor oraz klasę spektralną.

[7]:
```
# Wczytywanie potrzebnych modułów

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
from sklearn import preprocessing
from matplotlib.colors import ListedColormap
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from statsmodels.distributions.empirical_distribution import ECDF
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import learning_curve
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

[8]:
```
# Wczytywanie i wyświetlanie danych

dane = pd.read_csv('1.csv')
```

```
dane.head()
```

[8]:
```
   Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute magnitude(Mv)  \
0             3068          0.002400        0.1700                   16.12
1             3042          0.000500        0.1542                   16.60
2             2600          0.000300        0.1020                   18.70
3             2800          0.000200        0.1600                   16.65
4             1939          0.000138        0.1030                   20.06

   Star type Star color Spectral Class
0          0        Red              M
1          0        Red              M
2          0        Red              M
3          0        Red              M
4          0        Red              M
```

[9]:
```
# Modyfikacja nazw kolumn

dane.columns = ['Temperatura(K)','Jasność(L/Lo)','Promień(R/
 ↪Ro)','Wielkość_gwiazdowa','Typ','Kolor','Klasa_spektralna']
dane.head()
```

[9]:
```
   Temperatura(K)  Jasność(L/Lo)  Promień(R/Ro)  Wielkość_gwiazdowa  Typ  \
0            3068       0.002400         0.1700               16.12    0
1            3042       0.000500         0.1542               16.60    0
2            2600       0.000300         0.1020               18.70    0
3            2800       0.000200         0.1600               16.65    0
4            1939       0.000138         0.1030               20.06    0

  Kolor Klasa_spektralna
0   Red                M
1   Red                M
2   Red                M
3   Red                M
4   Red                M
```

[10]:
```
# Sprawdzanie ilości braków

print('Suma braków w danej kolumnie:\n')
dane.isnull().sum()
```

```
Suma braków w danej kolumnie:
```

[10]:
```
Temperatura(K)        0
Jasność(L/Lo)         0
Promień(R/Ro)         0
```

```
Wielkość_gwiazdowa    0
Typ                   0
Kolor                 0
Klasa_spektralna      0
dtype: int64
```

W analizowanych danych nie występują braki - dane są kompletne.

[11]:
```
# Podstawowe parametry statystyczne

round(dane.drop('Typ',axis = 1).describe(),2)
```

[11]:

	Temperatura(K)	Jasność(L/Lo)	Promień(R/Ro)	Wielkość_gwiazdowa
count	240.00	240.00	240.00	240.00
mean	10497.46	107188.36	237.16	4.38
std	9552.43	179432.24	517.16	10.53
min	1939.00	0.00	0.01	-11.92
25%	3344.25	0.00	0.10	-6.23
50%	5776.00	0.07	0.76	8.31
75%	15055.50	198050.00	42.75	13.70
max	40000.00	849420.00	1948.50	20.06

[12]:
```
# Modyfikacja elementów kolumn: Kolor, Klasa_spektralna (potrzebne w dalszej␣
 ↪analizie)

print(dane.Kolor.unique())
print(dane.Klasa_spektralna.unique())
```

```
['Red' 'Blue White' 'White' 'Yellowish White' 'Blue white'
 'Pale yellow orange' 'Blue' 'Blue-white' 'Whitish' 'yellow-white'
 'Orange' 'White-Yellow' 'white' 'Blue ' 'yellowish' 'Yellowish'
 'Orange-Red' 'Blue white ' 'Blue-White']
['M' 'B' 'A' 'F' 'O' 'K' 'G']
```

[13]:
```
LE = preprocessing.LabelEncoder()
dane['Kolor'] = LE.fit_transform(dane['Kolor'])
```

[14]:
```
dane.head()
```

[14]:

	Temperatura(K)	Jasność(L/Lo)	Promień(R/Ro)	Wielkość_gwiazdowa	Typ	\
0	3068	0.002400	0.1700	16.12	0	
1	3042	0.000500	0.1542	16.60	0	
2	2600	0.000300	0.1020	18.70	0	
3	2800	0.000200	0.1600	16.65	0	
4	1939	0.000138	0.1030	20.06	0	

	Kolor	Klasa_spektralna

```
0     10                M
1     10                M
2     10                M
3     10                M
4     10                M
```

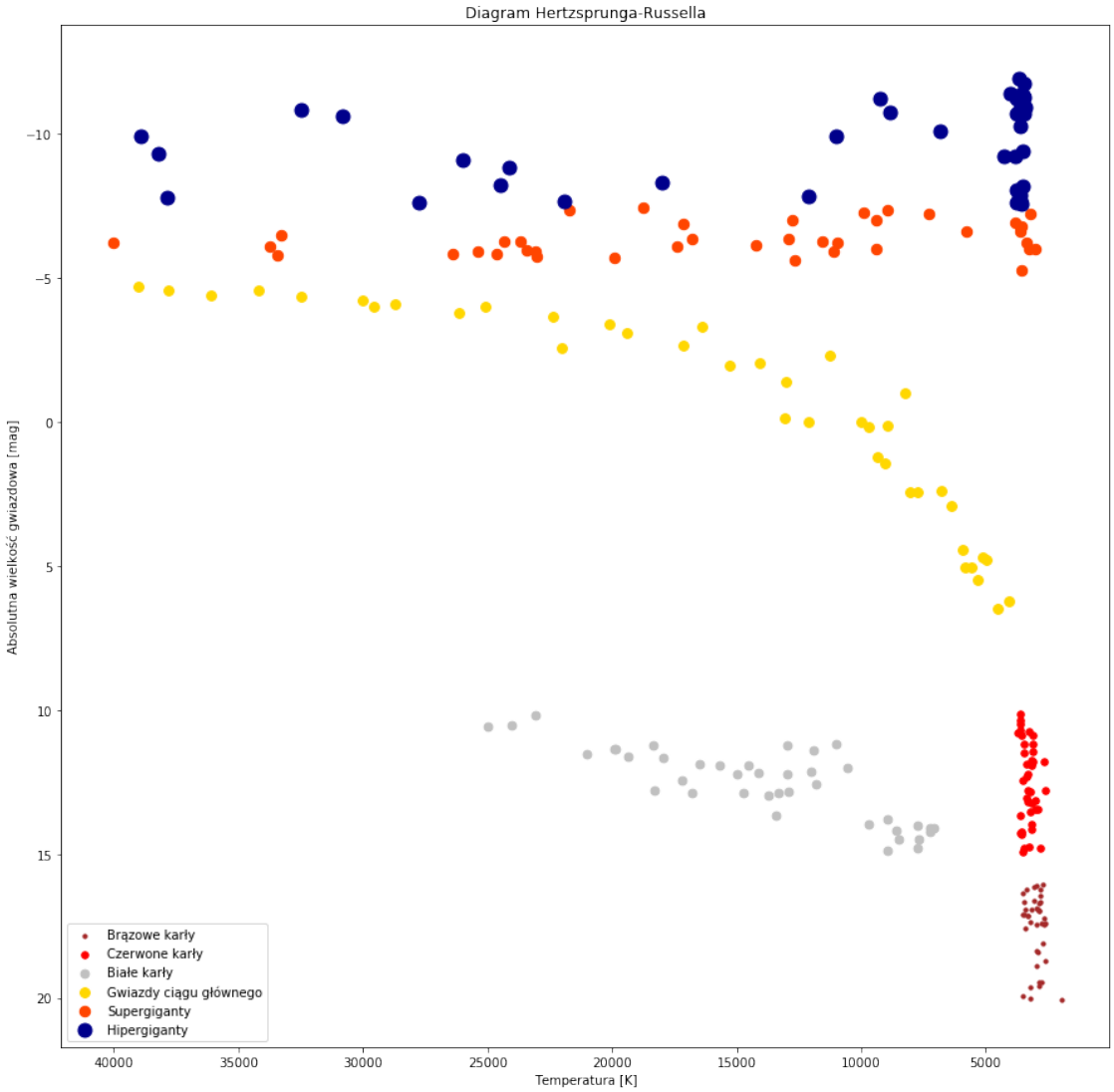
[15]:
```
dane['Klasa_spektralna'] = LE.fit_transform(dane['Klasa_spektralna'])
```

[16]:
```
dane.head()
```

[16]:
```
   Temperatura(K)  Jasność(L/Lo)  Promień(R/Ro)  Wielkość_gwiazdowa  Typ  \
0            3068       0.002400         0.1700               16.12    0
1            3042       0.000500         0.1542               16.60    0
2            2600       0.000300         0.1020               18.70    0
3            2800       0.000200         0.1600               16.65    0
4            1939       0.000138         0.1030               20.06    0

   Kolor  Klasa_spektralna
0     10                 5
1     10                 5
2     10                 5
3     10                 5
4     10                 5
```

[17]:
```
plt.figure(figsize = (15,15))
for i in range(0,len(dane['Typ'])):
    if dane['Typ'][i] == 0:
        b_d = plt.
 ↪scatter(dane['Temperatura(K)'][i],dane['Wielkość_gwiazdowa'][i], s = 10 , c =␣
 ↪'brown')
    elif dane['Typ'][i] == 1:
        r_d = plt.
 ↪scatter(dane['Temperatura(K)'][i],dane['Wielkość_gwiazdowa'][i],s = 30 , c =␣
 ↪'red')
    elif dane['Typ'][i] == 2:
        w_d = plt.
 ↪scatter(dane['Temperatura(K)'][i],dane['Wielkość_gwiazdowa'][i],s = 45 , c =␣
 ↪'silver')
    elif dane['Typ'][i] == 3:
        mss = plt.
 ↪scatter(dane['Temperatura(K)'][i],dane['Wielkość_gwiazdowa'][i],s = 60 , c =␣
 ↪'gold')
    elif dane['Typ'][i] == 4:
        sg = plt.
 ↪scatter(dane['Temperatura(K)'][i],dane['Wielkość_gwiazdowa'][i],s = 70 , c =␣
 ↪'orangered')
    elif dane['Typ'][i] == 5:
```

```
        hg = plt.
 ↪scatter(dane['Temperatura(K)'][i],dane['Wielkość_gwiazdowa'][i],s = 120 , c =␣
 ↪'darkblue')
plt.xlabel('Temperatura [K]')
plt.ylabel('Absolutna wielkość gwiazdowa [mag]')
plt.title('Diagram Hertzsprunga-Russella')
plt.legend((b_d,r_d,w_d,mss,sg,hg),('Brązowe karły','Czerwone karły','Białe␣
 ↪karły','Gwiazdy ciągu głównego','Supergiganty','Hipergiganty'))
plt.gca().invert_xaxis()
plt.gca().invert_yaxis()
plt.show()
print('W zbiorze danych znajduje się %s brązowych karłów.\n' %␣
 ↪len(dane['Typ'][dane['Typ'] == 0]))
print('W zbiorze danych znajduje się %s czerwonych karłów.\n' %␣
 ↪len(dane['Typ'][dane['Typ'] == 1]))
print('W zbiorze danych znajduje się %s białych karłów.\n' %␣
 ↪len(dane['Typ'][dane['Typ'] == 2]))
print('W zbiorze danych znajduje się %s gwiazd ciągu głównego.\n' %␣
 ↪len(dane['Typ'][dane['Typ'] == 3]))
print('W zbiorze danych znajduje się %s supergigantów.\n' %␣
 ↪len(dane['Typ'][dane['Typ'] == 4]))
print('W zbiorze danych znajduje się %s hipergigantów.\n' %␣
 ↪len(dane['Typ'][dane['Typ'] == 5]))
```



W zbiorze danych znajduje się 40 brązowych karłów.

W zbiorze danych znajduje się 40 czerwonych karłów.

W zbiorze danych znajduje się 40 białych karłów.

W zbiorze danych znajduje się 40 gwiazd ciągu głównego.

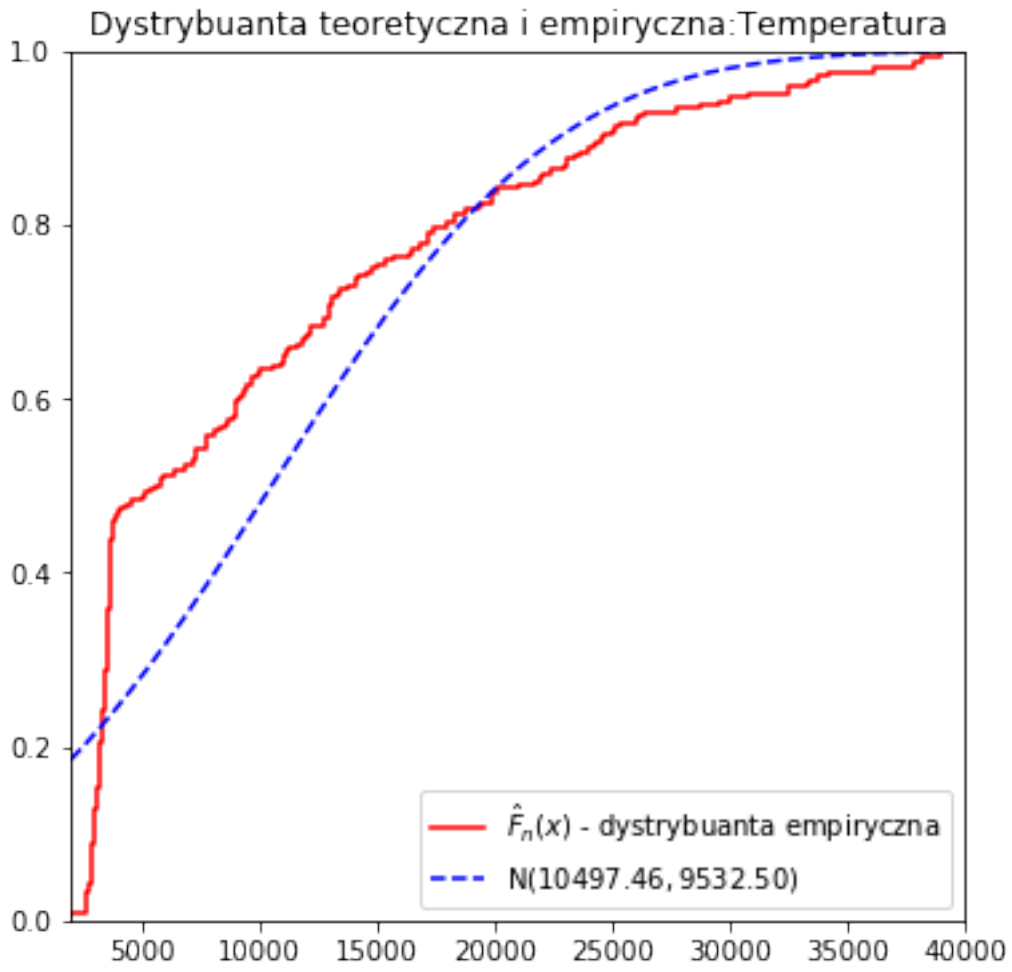
W zbiorze danych znajduje się 40 supergigantów.

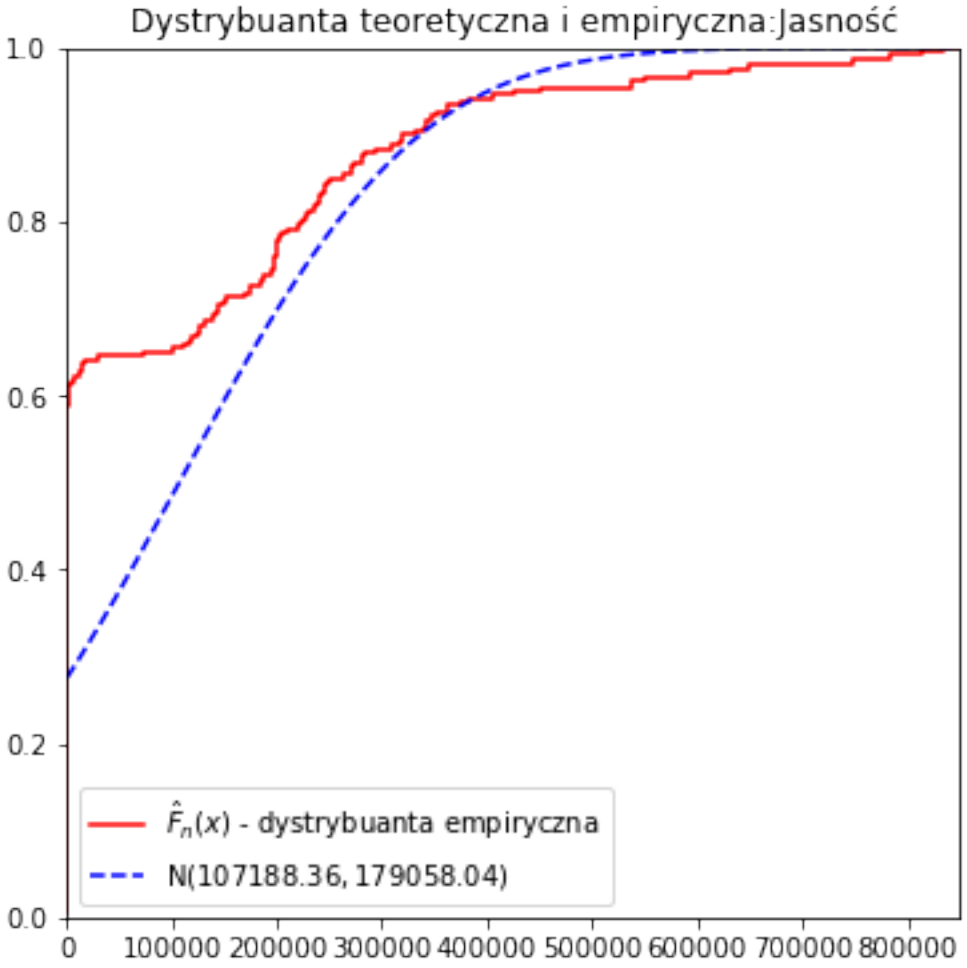
W zbiorze danych znajduje się 40 hipergigantów.

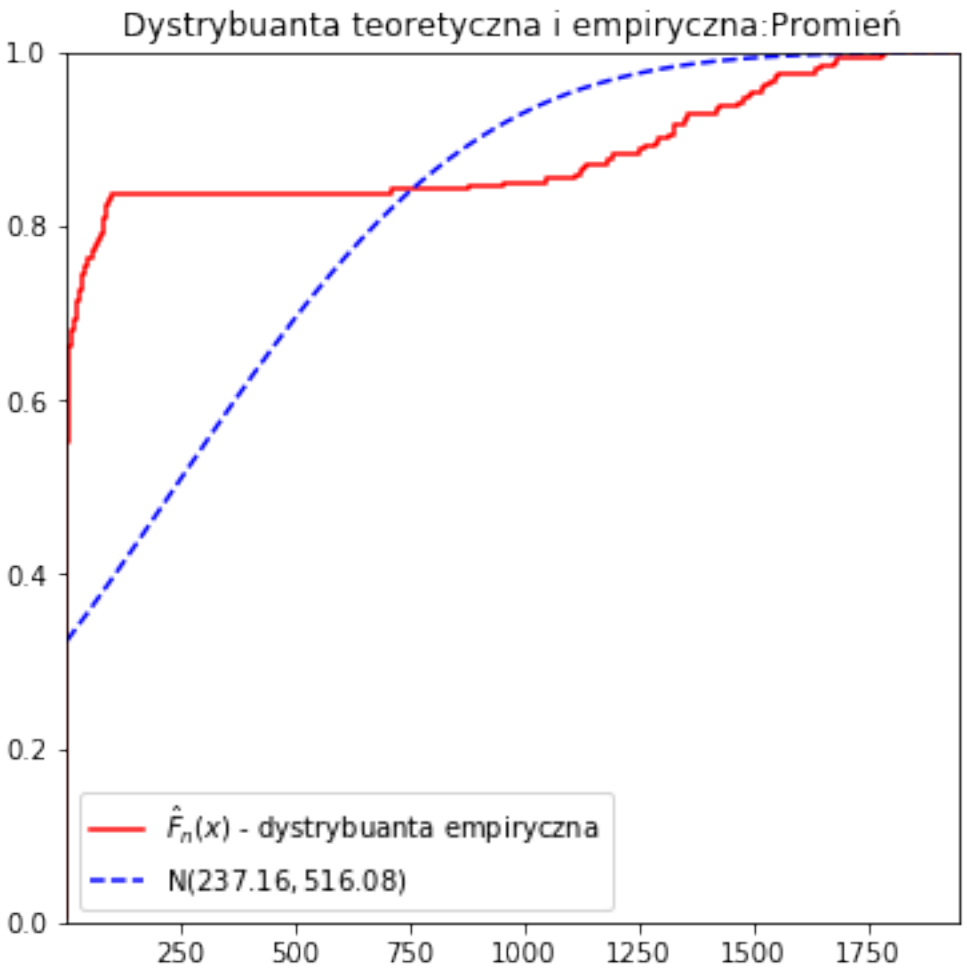
[18]:
```
class AnalizaStat:

        def __init__(self, k):
            self.k = k

            if self.k == 'Temperatura':
                    self.k1 = dane['Temperatura(K)'].values
            elif self.k == 'Jasność':
                    self.k1 = dane['Jasność(L/Lo)'].values
            elif self.k == 'Promień':
                    self.k1 = dane['Promień(R/Ro)'].values
            elif self.k == 'Wielkość gwiazdowa':
                    self.k1 = dane['Wielkość_gwiazdowa'].values
            else:
                print('Nie ma takiej kolumny!')

        def rozkład(self):
            alfa = 0.05
            m, s = sp.stats.norm.fit(self.k1)
            plt.figure(figsize=(6,6))
            plt.title('Dystrybuanta teoretyczna i empiryczna:'+ self.k)
            plt.axis([min(self.k1), max(self.k1), 0, 1])
            ecdf = ECDF(self.k1)
            plt.step(ecdf.x, ecdf.y, 'r-', label = r'$\hat{F}_n(x)$ -␣
 ↪dystrybuanta empiryczna')
            t1 = np.linspace(min(self.k1), max(self.k1), 1000)
            t2 = sp.stats.norm.cdf(t1, loc = m, scale = s)
            plt.plot(t1, t2, 'b--', label = r'$\mathrm{N}(%.2f, %.2f)$' % (m, s))
            plt.legend(loc = 'best')
            test = sp.stats.shapiro(self.k1)
            print('Dla zmiennej '+self.k+' wartosc statystyki testowej: %s a␣
 ↪p-value: %s' %(round(test[0],4),round(test[1],4)))
            if round(test[1],4) < alfa:
                print('Na podstanie przeprowadzonego testu odrzucamy hipotezę o␣
 ↪normalności rozkładu zmiennej: %s\n' %self.k)
            else:
                print('Nie ma podstaw do odrzucenia hipotezy o normalności␣
 ↪rozkładu zmiennej: %s\n' %self.k)


A1 = AnalizaStat('Temperatura')
A1.rozkład()
A2 = AnalizaStat('Jasność')
A2.rozkład()
A3 = AnalizaStat('Promień')
A3.rozkład()
A4 = AnalizaStat('Wielkość gwiazdowa')
```

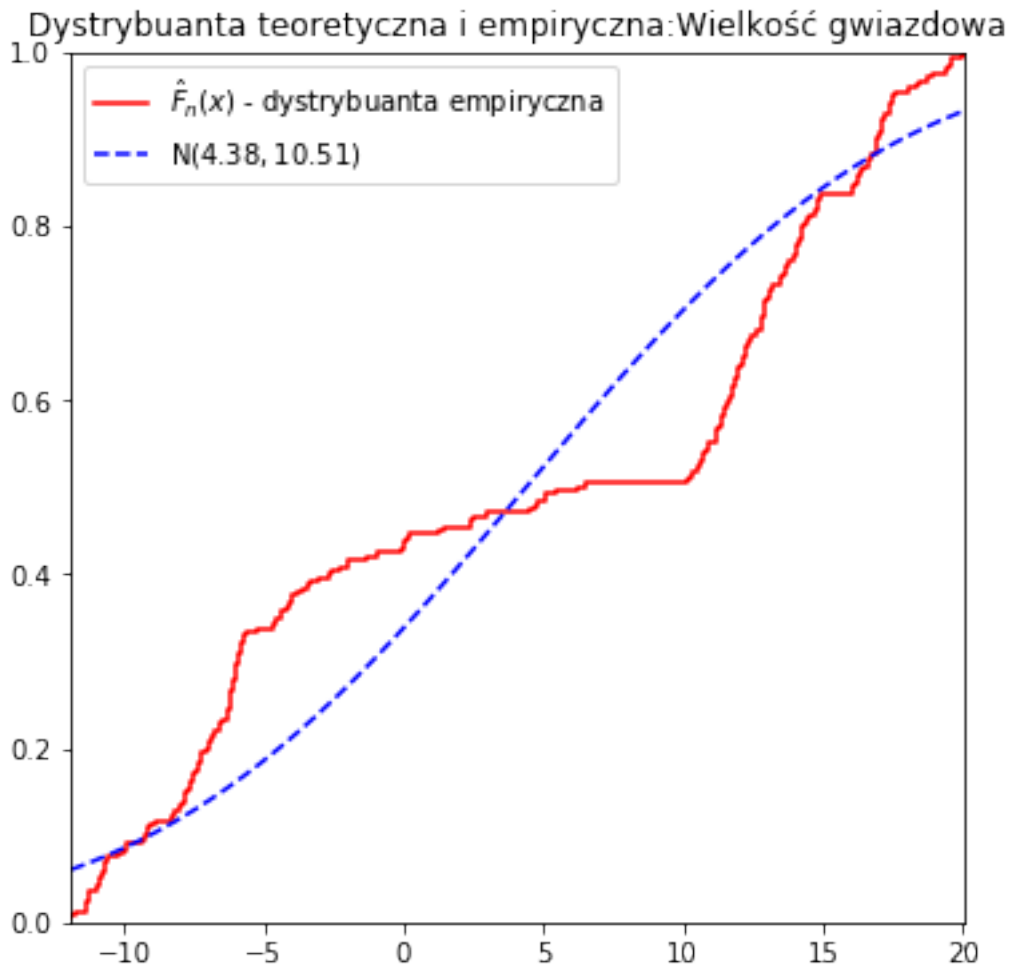
```
A4.rozkład()
```

```
Dla zmiennej Temperatura wartosc statystyki testowej: 0.7932 a p-value: 0.0
Na podstanie przeprowadzonego testu odrzucamy hipotezę o normalności rozkładu
zmiennej: Temperatura

Dla zmiennej Jasność wartosc statystyki testowej: 0.6597 a p-value: 0.0
Na podstanie przeprowadzonego testu odrzucamy hipotezę o normalności rozkładu
zmiennej: Jasność

Dla zmiennej Promień wartosc statystyki testowej: 0.5022 a p-value: 0.0
Na podstanie przeprowadzonego testu odrzucamy hipotezę o normalności rozkładu
zmiennej: Promień

Dla zmiennej Wielkość gwiazdowa wartosc statystyki testowej: 0.8691 a p-value:
0.0
Na podstanie przeprowadzonego testu odrzucamy hipotezę o normalności rozkładu
zmiennej: Wielkość gwiazdowa
```



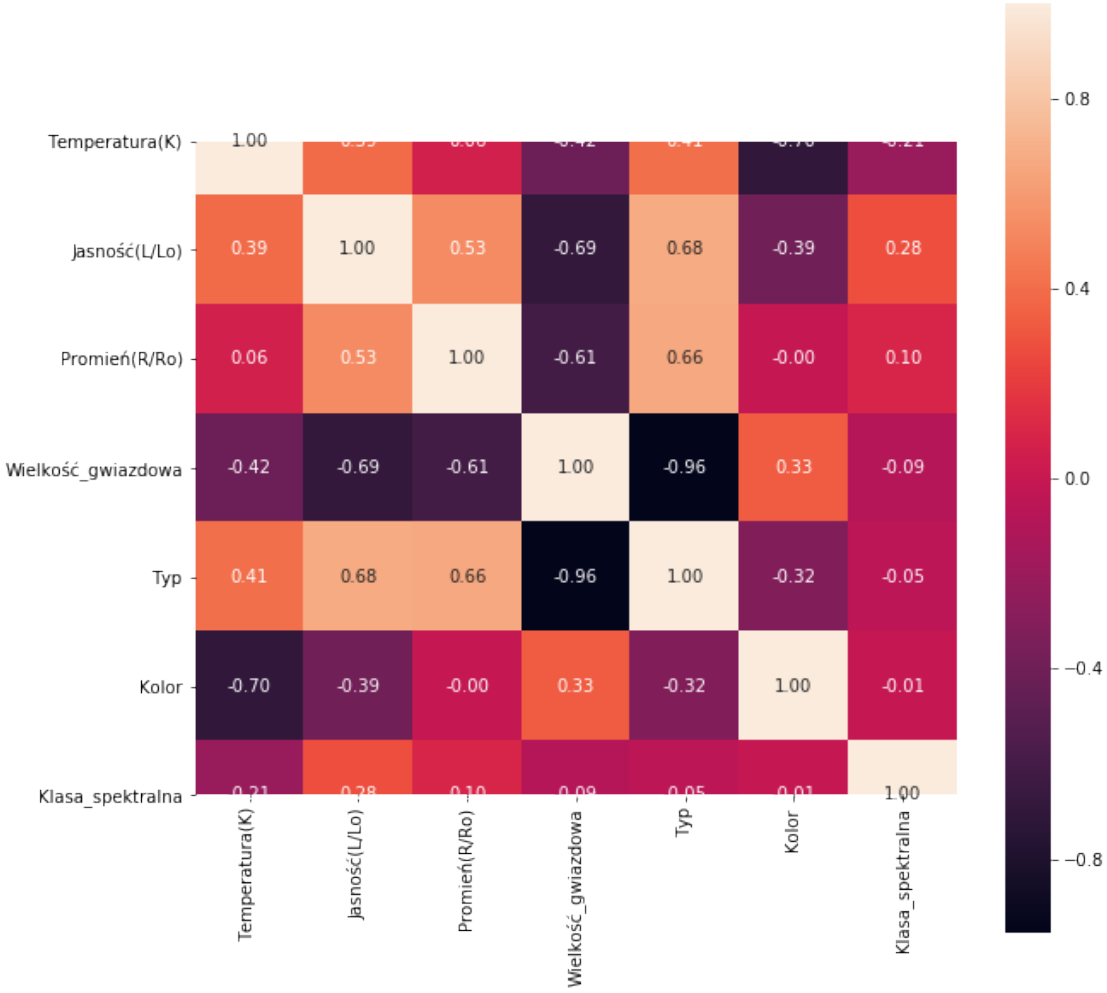

Dystrybuanta teoretyczna i empiryczna:Jasność
1.0
0.8
0.6
0.4
0.2
0.0
0
100000
200000
300000
400000
500000
600000
700000
800000
$\hat{F}_n(x)$ - dystrybuanta empiryczna
N(107188.36, 179058.04)


Dystrybuanta teoretyczna i empiryczna:Promień
1.0
0.8
0.6
0.4
0.2
0.0
250
500
750
1000
1250
1500
1750
$\hat{F}_n(x)$ - dystrybuanta empiryczna
N(237.16, 516.08)



[19]:
```
def cov_matrix(m):
    cov_data = np.corrcoef(m.T);
    plt.figure(figsize=(10,10))
    sns.heatmap(cov_data, cbar=True, annot=True, square=True, annot_kws={'size':
 ↪10}, fmt='.2f', xticklabels= m.columns, yticklabels=m.columns)
    return(cov_data)
```

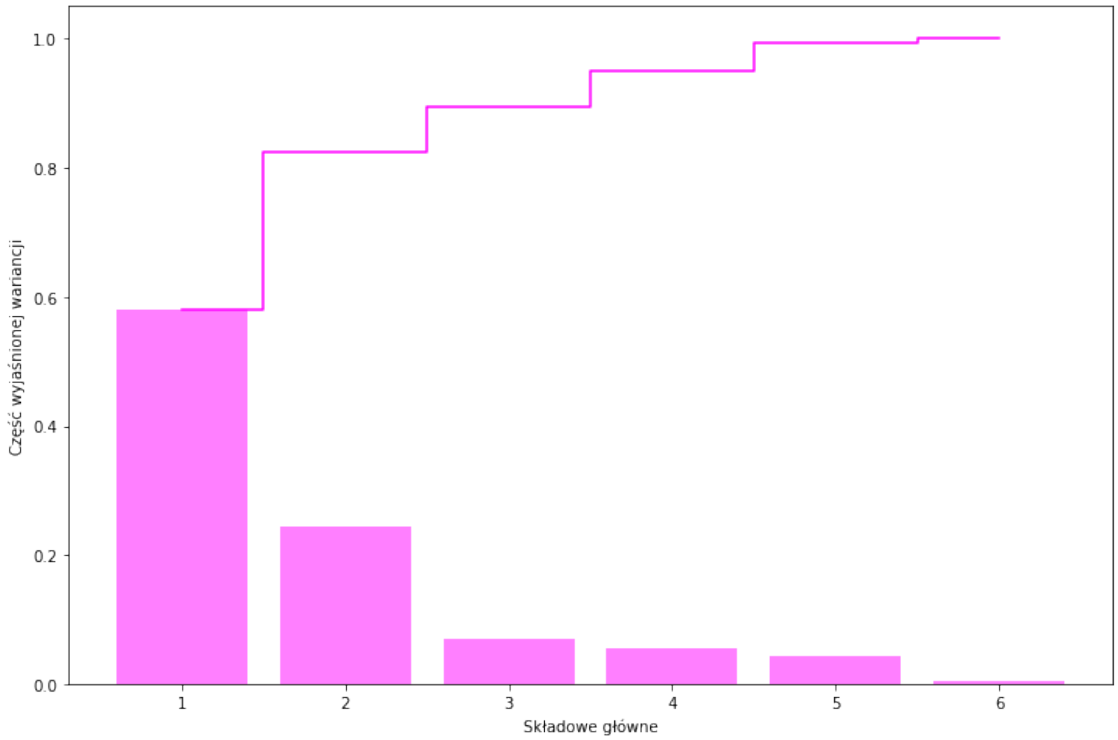
[20]:
```
# Mapa korelacji między analizowanymi zmiennymi

korelacja = cov_matrix(dane)
```



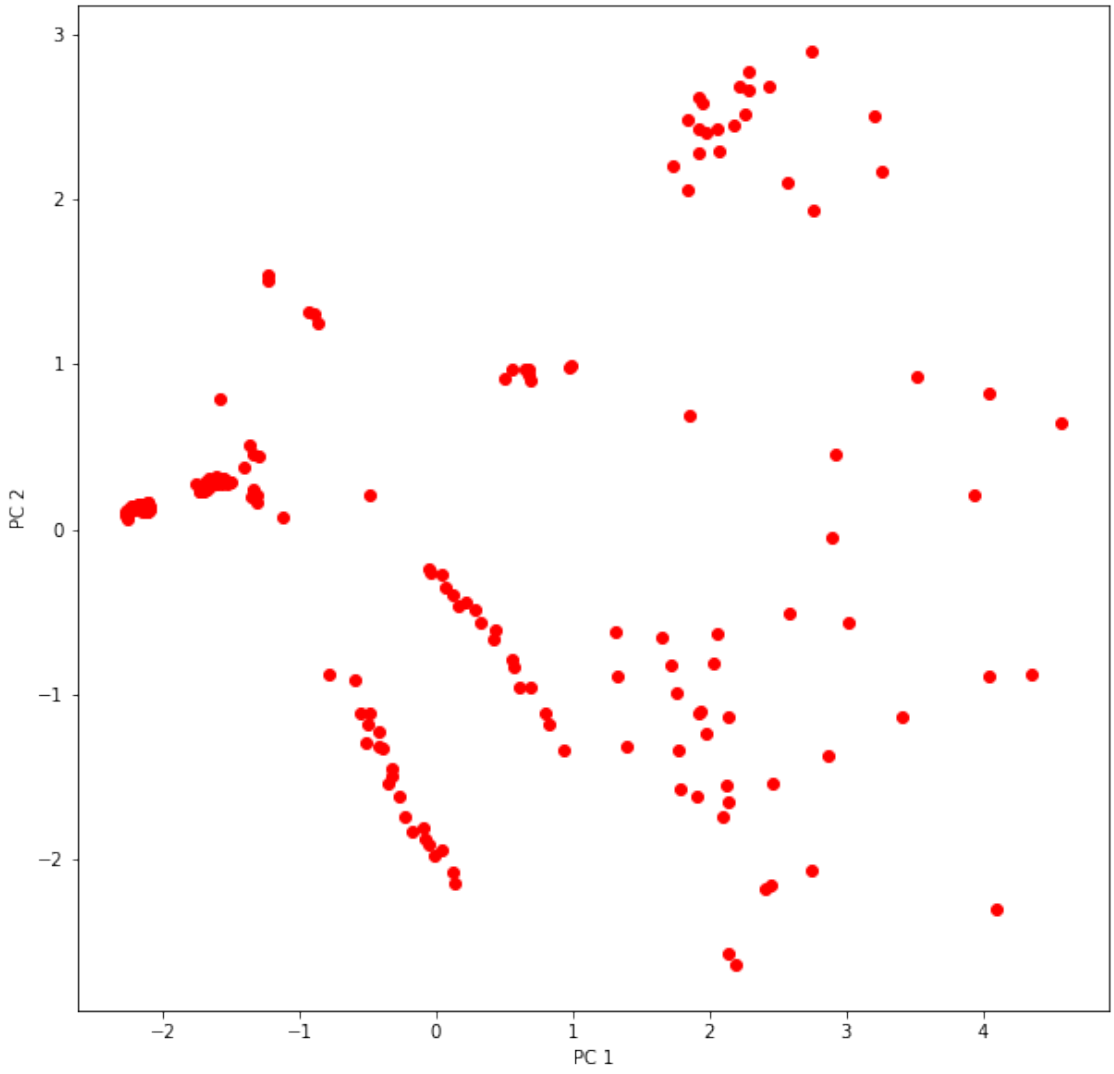
Z otrzymanej powyżej mapy widać znaczącą dotatnią korelację między Jasnością a Promieniem, Jasnością a Typem, Promieniem a Typem, oraz Typem a Temperaturą. Znacząca korelacja ujemna występuje między Jasnością a Wielkością gwiazdową, Promieniem a Wielkością gwiazdową, Wielkością gwiazdową a Typem, oraz Kolorem a Temperaturą.

[21]:
```
# Podział danych na treningowe i testowe

x = dane.loc[:, 'Temperatura(K)':'Kolor'].values
y = dane.loc[:, 'Klasa_spektralna'].values
x_train, x_test, y_train, y_test =  train_test_split(x, y, test_size=0.20,␣
 ↪random_state=1)

sc = StandardScaler()
X_train_std = sc.fit_transform(x_train)
X_test_std = sc.transform(x_test)
```

[22]:
```
# Analiza składowych głównych

pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)
plt.figure(figsize = (12,8))
plt.bar(range(1, 7), pca.explained_variance_ratio_, alpha=0.5, align='center',␣
 ↪color = 'magenta')
plt.step(range(1, 7), np.cumsum(pca.explained_variance_ratio_), where='mid',␣
 ↪color = 'magenta')
plt.ylabel('Część wyjaśnionej wariancji')
plt.xlabel('Składowe główne')
plt.show()
```



[23]:
```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
plt.figure(figsize = (10,10))
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], color = 'red')
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```



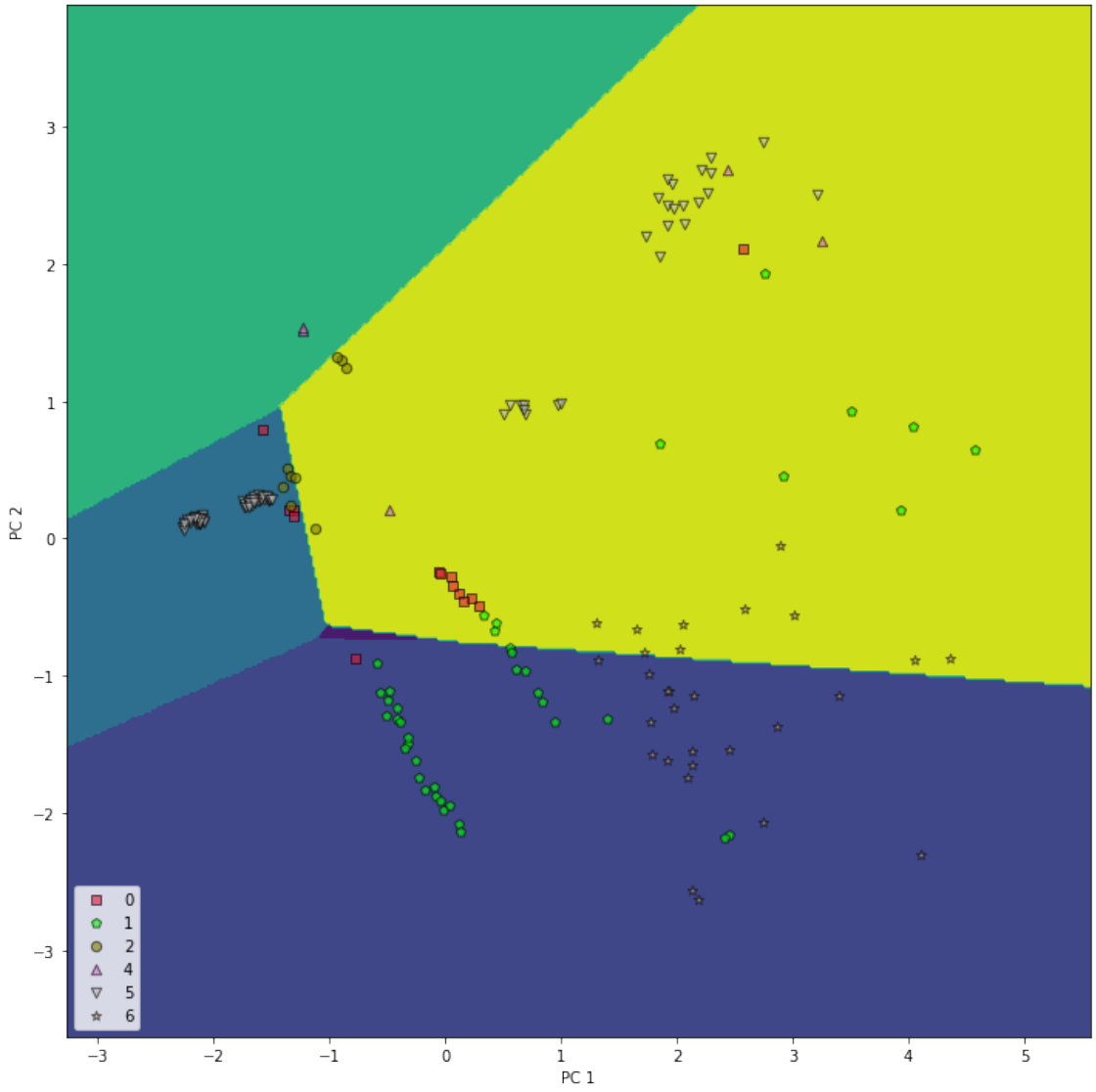
[24]:
```
def plot_decision_regions(X, y, classifier, resolution=0.02):

    # markery i kolory
    markers = ('s', 'p', 'o', '^', 'v','*','D')
    colors = ('crimson', 'lime', 'olive', 'orchid', 'silver','tan','white')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # obszary decyzyjne
    plt.figure(figsize = (10,10))
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution), np.
↪arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
```

```
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())


    # wykres
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.6, color =␣
↪cmap(idx), edgecolor='black', marker=markers[idx], s = 40, label=cl)
```

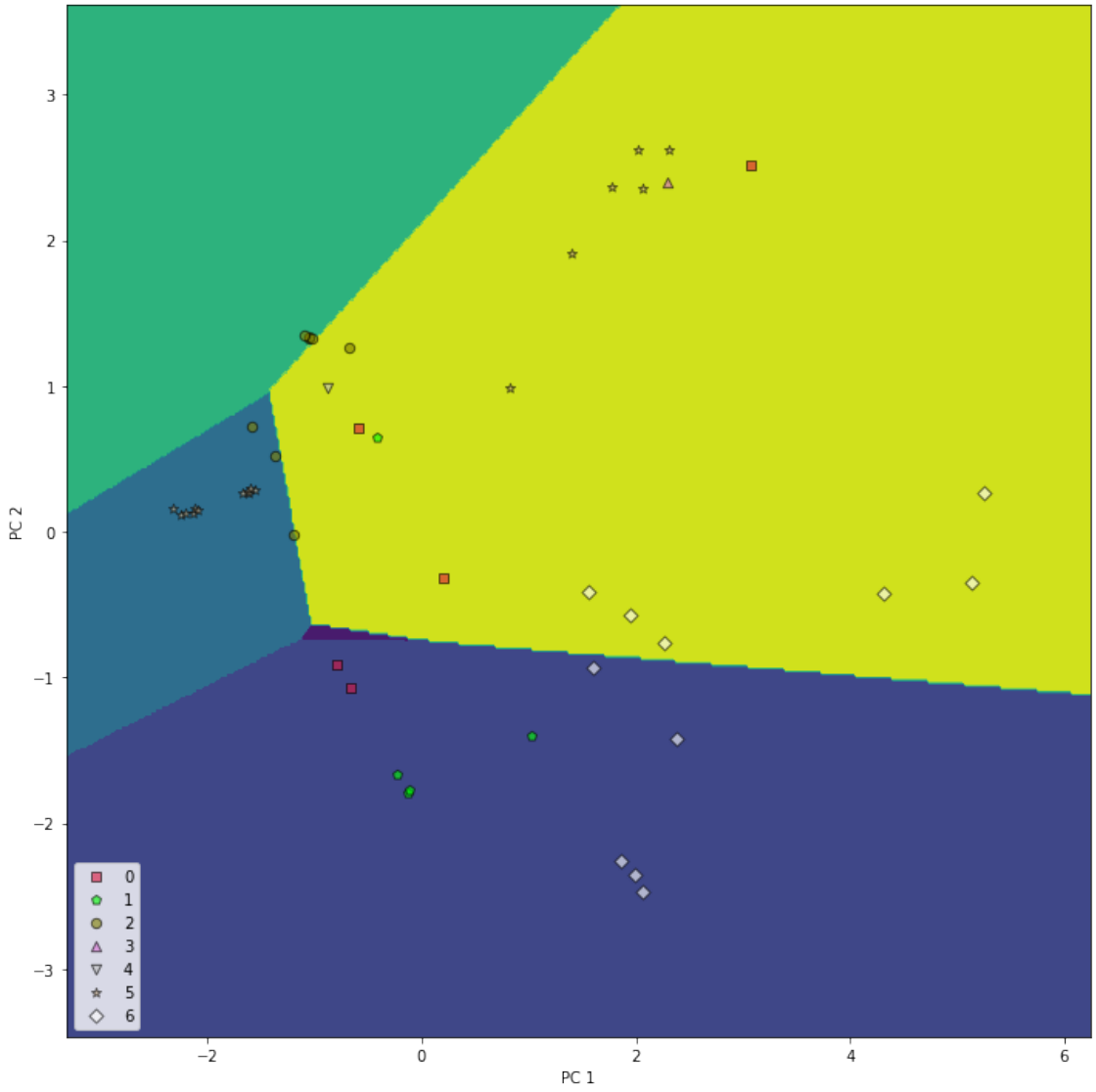
[25]:
```
klasyfikator = SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001,␣
↪l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True,␣
↪verbose=0, epsilon=0.1, n_jobs=None, random_state=None,␣
↪learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False,␣
↪validation_fraction=0.1, n_iter_no_change=5, class_weight=None,␣
↪warm_start=False, average=False)

klasyfikator = klasyfikator.fit(X_train_pca, y_train)

print('Dokładność dla danych treningowych:', round(klasyfikator.
↪score(X_train_pca, y_train),3))
print('Dokładność dla danych testowych:', round(klasyfikator.score(X_test_pca,␣
↪y_test),3))
```

```
Dokładność dla danych treningowych: 0.328
Dokładność dla danych testowych: 0.271
```

[26]:
```
plot_decision_regions(X_train_pca, y_train, classifier = klasyfikator)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```



[27]:
```
plot_decision_regions(X_test_pca, y_test, classifier = klasyfikator)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```



[28]:
```
# Strumień danych

pipe_1 = make_pipeline(StandardScaler(), SVC(random_state=1))

parametr_1 = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
parametr_2 = np.arange(0,0.5,0.1)

zbiór_parametrów = {'svc__C': parametr_1, 'svc__kernel':␣
↪['linear','rbf','poly','sigmoid'],'svc__gamma':['scale', 'auto'],'svc__coef0':
↪parametr_2}

wyszukiwanie = GridSearchCV(estimator = pipe_1, param_grid = zbiór_parametrów,␣
↪scoring = 'accuracy', cv = 5, n_jobs=-1)
```

```
wyszukiwanie = wyszukiwanie.fit(x_train, y_train)

lista_dokładności = cross_val_score(wyszukiwanie, x_train, y_train, scoring =␣
 ↪'accuracy', cv = 5)

print('Dokładność dla k-tego podziału: %s' % np.round(lista_dokładności,3))
print('Średnia dokładność z odchyleniem: %s +/- %s' % (round(np.
 ↪mean(lista_dokładności),3), round(np.std(lista_dokładności),3)))
print('Najlepszy wybór parametrów: %s' % wyszukiwanie.best_params_)
```

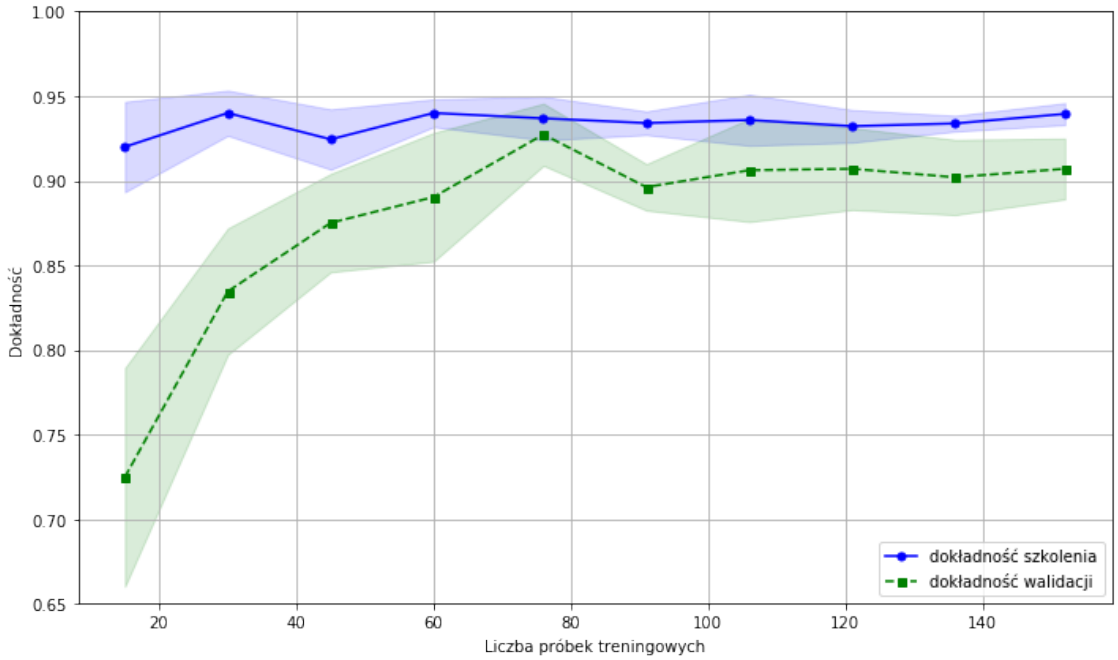
```
Dokładność dla k-tego podziału: [0.9   0.897 0.897 0.897 0.914]
Średnia dokładność z odchyleniem: 0.901 +/- 0.007
Najlepszy wybór parametrów: {'svc__C': 1.0, 'svc__coef0': 0.0, 'svc__gamma':
'scale', 'svc__kernel': 'linear'}
```

[29]:
```
# Aplikacja najlepszego wyboru parametrów do strumienia danych

pipe_opt = wyszukiwanie.best_estimator_
pipe_opt.fit(x_train, y_train)
y_pred = pipe_opt.predict(x_test)
print('Dokładność przewidywania dla danych testowych: %s' % round(pipe_opt.
 ↪score(x_test, y_test),3))
```

```
Dokładność przewidywania dla danych testowych: 0.854
```

[30]:
```
train_sizes, train_scores, test_scores = learning_curve(estimator=pipe_opt,␣
 ↪X=x_train, y=y_train, train_sizes=np.linspace(0.1, 1.0, 10), cv=5, n_jobs=-1)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.figure(figsize = (10,6))
plt.plot(train_sizes, train_mean,
         color='blue', marker='o',
         markersize=5, label='dokładność szkolenia')

plt.fill_between(train_sizes,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='dokładność walidacji')
```

```
plt.fill_between(train_sizes,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xlabel('Liczba próbek treningowych')
plt.ylabel('Dokładność')
plt.legend(loc='lower right')
plt.ylim([0.65, 1.0])
plt.tight_layout()
plt.show()
```



[31]:
```
# Powrót do nazw typów spektralnych

oks = dict(zip(LE.classes_, LE.transform(LE.classes_)))
print(oks)
```

```
{'A': 0, 'B': 1, 'F': 2, 'G': 3, 'K': 4, 'M': 5, 'O': 6}
```

[32]:
```
# Tablica pomyłek

confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
fig, ax = plt.subplots(figsize=(10, 10))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
    for j in range(confmat.shape[1]):
        ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')

ax.set_xticklabels(['']+list(oks.keys()))
ax.set_yticklabels(['']+list(oks.keys()))
plt.xlabel('Wartości przewidywane')
plt.ylabel('Wartości prawdziwe')

plt.tight_layout()
plt.show()
```