Sprawozdanie

Zgłębianie danych

Matsvei Pilkou

1. Wstęp:

Dla analizy danych była wybrana baza **Drug consumption**.

Baza zawiera w sobie:

1. Zestandaryzowane wartości:
   Age (wiek), **Gender** (płeć), **Education** (wykształcenie),
   Country (kraj), **Ethnicity** (pochodzenie etniczne),
   [N, E,O,A, C, I]score - to wartości NEO-FFI-R (neurotyzm,extrawersja, otwartość na doświadczenie,ugodowność, sumienność),
   Impulsive – wartość BIS-11 (impulsywność),
   SS – wartości ImpSS (poszukiwanie sensacji)
2. Wartości w postaci CL[0..6], które odpowiadają za częstość przyjmowania tego lub innego środka, te środki to:
   Alcohol, **Amphet**, **Amyl**, **Benzos**, **Caff**, **Cannabis**, **Choc**, **Coke**, **Crack**, **Ecstasy**, **Heroin**, **Ketamine**, **Legalh**, **LSD**, **Meth**, **Mushrooms**, **Nicotine** , **Semer**, **VSA**

Celem projektu będzie zależenie klasyfikatora którzy bym najliepiej oszacował kto z podanych osób jest osobą mającą skończone wykształcenie wyższe.

2. Przetwarzanie / obróbka / łączenie/ dzielenie baz danych

```
def changeCLNtoN():
    try:
        for i in range(len(df.Age)):
            for j in range(start,len(column_names)):
                df.iloc[i,j] = np.float64(df.iloc[i,j][-1])
    except IndexError:
           print('Data already changed')
    finally:
        if(type(df.iloc[1,16]) != np.float64):
            for i in range(len(df.Age)):
                for j in range(start,len(column_names)):
                    df.iloc[i,j] = np.float64(df.iloc[i,j])

    print('Compleate')

    return None
changeCLNtoN()
```

Rys. 1. Datamin_Data_1

Na rysunku 1 jest przedstawiona funkcja zmieniająca wartości w postaci CL[0..6] na liczby typu float64.

```
df_to_scale = pd.concat([df.iloc[:,start:],df.iloc[:,5:start-2]],axis=1)
```

Rys. 2. Datamin_Data_2

Na rysunku 2 jest przedstawiony zbiór danych do obróbki do którego doszli wszystjie kolumny oprócz kolumn: Age, Gender, Education, Country, Ethnicity, Impulsive, SS

```
scaler = StandardScaler()
scaler.fit(df_to_scale)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
scaled_data = scaler.transform(df_to_scale)
```

```
pca = PCA(n_components=3)
```

```
pca.fit(scaled_data)
```

```
PCA(copy=True, iterated_power='auto', n_components=3, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
x_pca = pca.transform(scaled_data)
```

Rys. 3. Datamin_Data_3

Na rysunku 3 jest przedstawione skalowanie i wykorzystanie algorytmu PCA, dla standaryzacji i zmniejszenia danych.

```
scaled_df = pd.DataFrame(x_pca,columns=['Type1','Type2','Type3'])
scaled_df.index +=1
scaled_df.head()
```

	Type1	Type2	Type3
1	-1.879984	1.340416	-0.295996
2	1.272495	-2.390065	-0.168470
3	-1.961490	0.344110	-0.577363
4	-1.236894	0.416153	0.638793
5	-1.282846	0.661236	-0.883872

```
new_df = df.drop(df.columns[start:].tolist(),axis=1)
new_df = new_df.drop(new_df.columns[5:start-2].tolist(),axis=1)
new_df.head()
```

	Age	Gender	Education	Country	Ethnicity	Impulsive	SS
1	0.49788	0.48246	-0.05921	0.96082	0.12600	-0.21712	-1.18084
2	-0.07854	-0.48246	1.98437	0.96082	-0.31685	-0.71126	-0.21575
3	0.49788	-0.48246	-0.05921	0.96082	-0.31685	-1.37983	0.40148
4	-0.95197	0.48246	1.16365	0.96082	-0.31685	-1.37983	-1.18084
5	0.49788	0.48246	1.98437	0.96082	-0.31685	-0.21712	-0.21575

```
# NEEDDED!
new_df['Id'] = range(1, len(new_df) + 1)
scaled_df['Id'] = range(1, len(scaled_df) + 1)
```

Rys. 4. Datamin_Data_4

Na rysunku 4 są przedstawione zeskanowane dane z zredukowaną wielowymiarowością danych (scaled_df) i dane z które były przeskalowywane na początku (new_df), także dodajemy kolumnę dla identyfikacji rekordów.

```
df = pd.merge(new_df, scaled_df, how='outer', on=['Id'])
print('Is it Working? {}'.format(df.isnull().values.any() == False))
df = df.drop('Id',axis=1)
```

```
Is it Working? True
```

Rys. 5. Datamin_Data_5

Na rysunku 5 jest przedstawione łączenie danych zależnie od identyfikatora, a potem usunięcie go. Także sprawdzenie czy są wartości typu brak danych.

```
def addClassByEducation():
    try:
        classOfEdu = []
        for i in range(len(df.Age)):
            #Education
            if analis_by == 'Education':
                if round(df.iloc[i]['Education'],5) > -0.05921:
                    #df.iloc[i]['Class'] = 1
                    #print(1)
                    classOfEdu.append(1)
                elif round(df.iloc[i]['Education'],5) <= -0.05921:
                    classOfEdu.append(0)
                    #print(0)
                    #df.iloc[i]['Class'] = 0
                else:
                    classOfEdu.append(None)
            elif analis_by == 'Age':
                if round(df.iloc[i]['Age'],5) > 0:
                    #df.iloc[i]['Class'] = 1
                    #print(1)
                    classOfEdu.append(1)
                elif round(df.iloc[i]['Age'],5) <= 0:
                    classOfEdu.append(0)
                    #print(0)
                    #df.iloc[i]['Class'] = 0
                else:
                    classOfEdu.append(None)
        df['Class'] = classOfEdu
    except IndexError:
                print('Data already changed')

    return None
addClassByEducation()
```

Rys. 6. Datamin_Data_6

Na rysunku 6 jest przedstawione dodanie klasyfikacji do zbioru danych w zależności od parametru analizy.

3. Klasyfikatory i ich ewaluacja

1. Podział danych na zbiory

```
X = df.drop(['Class',analis_by],axis=1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Rys. 7. Datamin_Train_Test_Split

2. Gaussian Naive Bayes

Gaussian Naive Bayes ¶

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb = GaussianNB()
```

```
gnb.fit(X_train,y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
gnb_predictions = gnb.predict(X_test)
```

```
print(classification_report(y_test,gnb_predictions))
```

```
              precision    recall  f1-score   support

           0       0.72      0.68      0.70       344
           1       0.63      0.67      0.65       279

   micro avg       0.68      0.68      0.68       623
   macro avg       0.68      0.68      0.68       623
weighted avg       0.68      0.68      0.68       623
```

```
print(confusion_matrix(y_test,gnb_predictions))
```

```
[[235 109]
 [ 91 188]]
```

Rys. 7. Datamin_3_NB

3. Decision tree

Decision tree

```
dtree = DecisionTreeClassifier()
```

```
dtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
dtree_predictions = dtree.predict(X_test)
```

```
print(classification_report(y_test,dtree_predictions))
```

```
              precision    recall  f1-score   support

           0       0.64      0.63      0.63       344
           1       0.55      0.56      0.56       279

   micro avg       0.60      0.60      0.60       623
   macro avg       0.59      0.60      0.60       623
weighted avg       0.60      0.60      0.60       623
```

```
print(confusion_matrix(y_test,dtree_predictions))
```

```
[[216 128]
 [122 157]]
```

4. Random forest

Random forest

```
rfc = RandomForestClassifier(n_estimators=400)
```

```
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=400, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
rfc_predictions = rfc.predict(X_test)
```

```
print(classification_report(y_test,rfc_predictions))
```

```
              precision    recall  f1-score   support

           0       0.71      0.74      0.72       344
           1       0.66      0.63      0.65       279

   micro avg       0.69      0.69      0.69       623
   macro avg       0.69      0.68      0.69       623
weighted avg       0.69      0.69      0.69       623
```

```
print(confusion_matrix(y_test,rfc_predictions))
```

```
[[254  90]
 [103 176]]
```

5. KNN

KNN

```
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=1, p=2,
           weights='uniform')
```

```
knn_predictions = knn.predict(X_test)
```

```
print(confusion_matrix(y_test,knn_predictions))
```

```
[[230 114]
 [117 162]]
```

```
print(classification_report(y_test,knn_predictions))
```

```
              precision    recall  f1-score   support

           0       0.66      0.67      0.67       344
           1       0.59      0.58      0.58       279

   micro avg       0.63      0.63      0.63       623
   macro avg       0.62      0.62      0.62       623
weighted avg       0.63      0.63      0.63       623
```

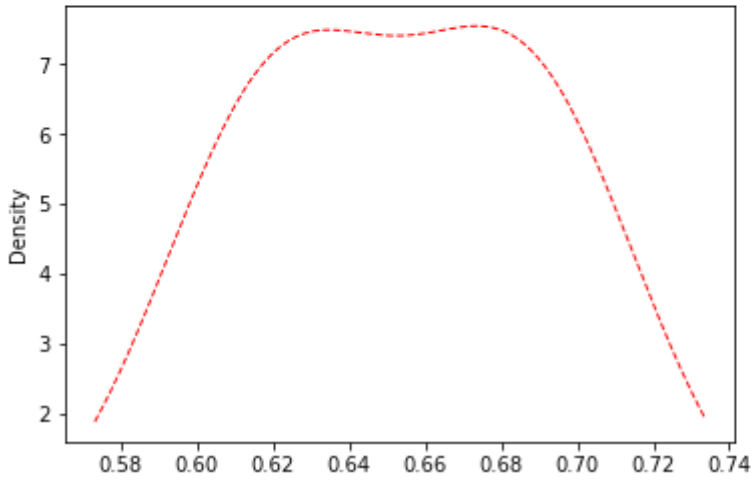
6. Acurasy compare

Acurasy compare

```
acur_comp = pd.DataFrame()
alg = [gnb, dtree, rfc, knn]
algName = ['gnb', 'dtree', 'rfc', 'knn']
for i in range(4):
    acur_comp.loc[i,'name'] = algName[i]
    acur_comp.loc[i,'test'] = round(alg[i].score(X_test,y_test),4)
```

```
acur_comp['test'].plot.density(lw=1,ls='--',c='red')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f61e320>
```



3.1 Odpowiedzi

1. Rozumienie macierzy błędu

		Klasa testowa	
		pozytywna	**negatywna**
Klasa prawidłowa	**pozytywna**	Prawdziwie pozytywna (TP)	Fałszywie pozytywna (FP)
	negatywna	Fałszywie negatywna (FN)	Prawdziwie negatywna (TN)

2. Obliczenie TPR(recall, sensitivity)i FPR(fall-out, false alarm)

```
acur_comp.drop(['test'],axis=1)
```

	name	TPR	FPR
0	gnb	0.680233	0.322581
1	dtree	0.654070	0.437276
2	rfc	0.750000	0.376344
3	knn	0.668605	0.419355

3. Podać wzory dla TPR i FPR są miary FNR i TNR.

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

miss rate or false negative rate (FNR)

$$\mathrm{FNR} = \frac{\mathrm{FN}}{P} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{N} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

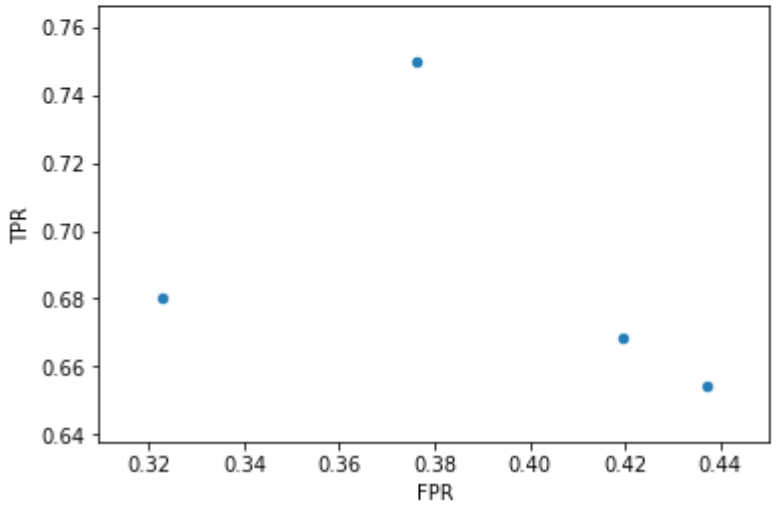
4. Błąd pierwszego i drugiego rodzaju
   Błąd pierwszego rodzaju to wartość **FP** w macierzy błędów
   Błąd drugiego rodzaju to wartość **FN** w macierzy błędów
   Im więcej błędów pierwszego rodzaju tym więcej jest wartość TPR
   Im więcej błędów drugiego rodzaju tym więcej jest wartość FNR
5. W zależności od systemu w którym działamy
   Jeżeli to system dla produkcji opakowania to lepiej jest popełnić błąd pierwszego rodzaju (gzie pozytywna to nie dobra jakość w produkcji), jeżeli to system medyczny to błąd drugiego rodzaju (gzie pozytywna to pacjent jest chory)

6. Dla czterech klasyfikatorów obliczyłem parę (FPR,TPR) i zaznaczyęm jako punkt na wykresie

0	gnb	0.680233	0.322581
1	dtree	0.654070	0.437276
2	rfc	0.750000	0.376344
3	knn	0.668605	0.419355

```
acur_comp.drop(['test'],axis=1).plot.scatter('FPR','TPR')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21f6f358>
```



Najbliżej idealnego jest Random Forest być może GaussianNB

4.Grupowanie metodą k-średnich

K-means

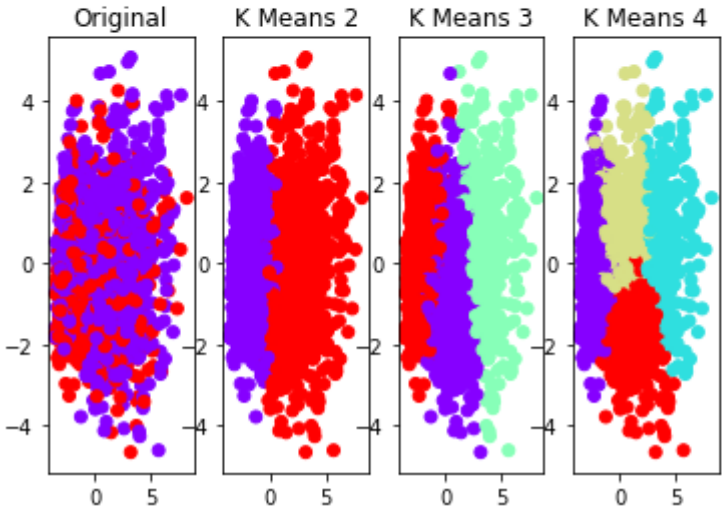
```
kmeans = []
for i in range(3):
    kmeans.append(KMeans(n_clusters=(i+2)))
    kmeans[i].fit(X)
print(confusion_matrix(y,kmeans[0].labels_))
print(classification_report(y,kmeans[0].labels_))
```

```
[[603 249]
 [454 579]]
              precision    recall  f1-score   support

           0       0.57      0.71      0.63       852
           1       0.70      0.56      0.62      1033

   micro avg       0.63      0.63      0.63      1885
   macro avg       0.63      0.63      0.63      1885
weighted avg       0.64      0.63      0.63      1885
```

Na rysunku jest próba wykorzystania algorytmy k-means, jako klasyfikatora dla zadania. Obliczone wartości macierzy błędu i dokładność.

```
f, (ax, ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=4)
ax.set_title("Original")
ax.scatter(df["Type1"],df["Type2"],c=df["Class"],cmap='rainbow')
ax1.set_title('K Means 2')
ax1.scatter(df["Type1"],df["Type2"],c=kmeans[0].labels_,cmap='rainbow')
ax2.set_title('K Means 3')
ax2.scatter(df["Type1"],df["Type2"],c=kmeans[1].labels_,cmap='rainbow')
ax3.set_title('K Means 4')
ax3.scatter(df["Type1"],df["Type2"],c=kmeans[2].labels_,cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x18c41d30>
```



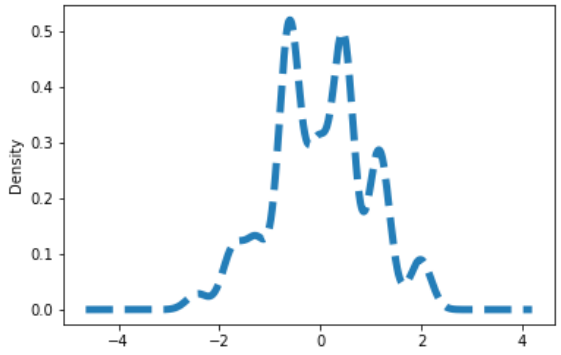
Graficzna reprezentacja działania algorytmu K-means

6.Badania dodatkowe

1. Wykresy

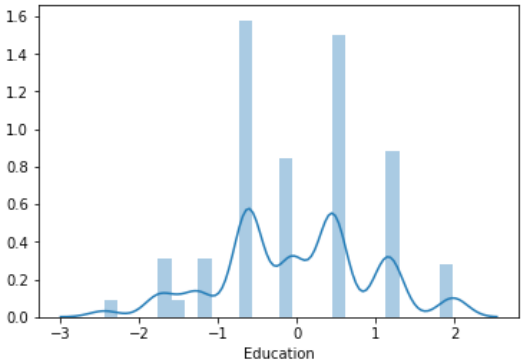
```
df['Education'].plot.density(lw=5,ls='--')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xba27d30>
```



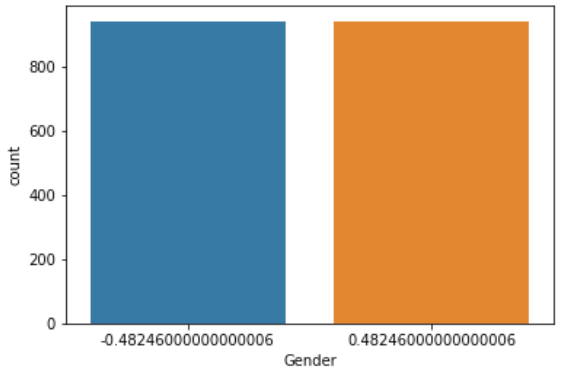

```
sns.distplot(df.Education)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd2f5630>
```



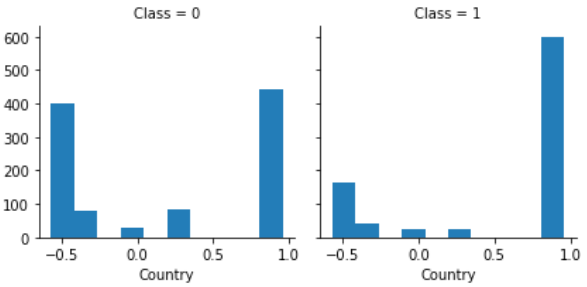

```
sns.countplot(x='Gender',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd3a3780>
```



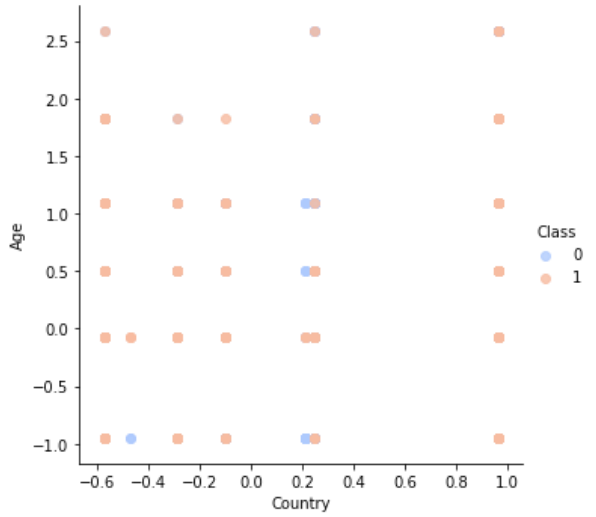

```
g = sns.FacetGrid(data=df,col='Class')
g.map(plt.hist,'Country')
```

```
<seaborn.axisgrid.FacetGrid at 0x18d5e898>
```



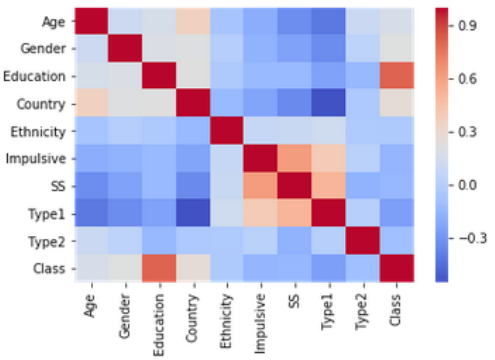
```
sns.lmplot('Country','Age',data=df, hue='Class',
           palette='coolwarm',fit_reg=False)
```

```
<seaborn.axisgrid.FacetGrid at 0xd4da128>
```



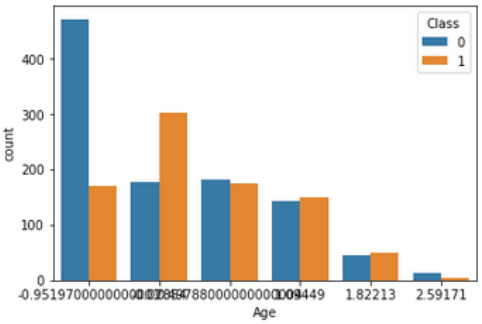

```
sns.heatmap(df.corr(),cmap='coolwarm')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11f1ea58>
```




```
sns.countplot(x='Age',hue='Class',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x14bb2b38>
```



2.Badania o wiek

Klasifikator	Macierz błędu	Dokładność
Gaussian Naive Bayes	[[264 128] [66 165]]	precision recall f1-score support 0 0.80 0.67 0.73 392 1 0.56 0.71 0.63 231 micro avg 0.69 0.69 0.69 623 macro avg 0.68 0.69 0.68 623 weighted avg 0.71 0.69 0.69 623
Decision tree	[[264 128] [98 133]]	precision recall f1-score support 0 0.73 0.67 0.70 392 1 0.51 0.58 0.54 231 micro avg 0.64 0.64 0.64 623 macro avg 0.62 0.62 0.62 623 weighted avg 0.65 0.64 0.64 623

Random forest

```
[[277 115]
 [ 78 153]]
```

	precision	recall	f1-score	support
0	0.78	0.71	0.74	392
1	0.57	0.66	0.61	231
micro avg	0.69	0.69	0.69	623
macro avg	0.68	0.68	0.68	623
weighted avg	0.70	0.69	0.69	623

KNN

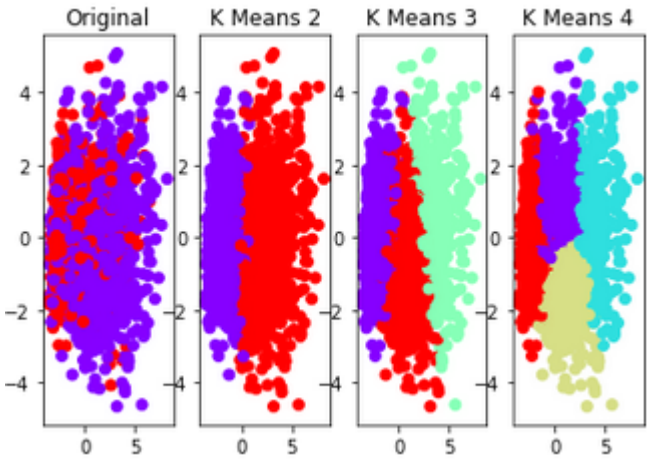
```
[[267 125]
 [ 93 138]]
```

	precision	recall	f1-score	support
0	0.74	0.68	0.71	392
1	0.52	0.60	0.56	231
micro avg	0.65	0.65	0.65	623
macro avg	0.63	0.64	0.63	623
weighted avg	0.66	0.65	0.65	623

K-means

```
[[589 172]
 [468 656]]
```

	precision	recall	f1-score	support
0	0.56	0.77	0.65	761
1	0.79	0.58	0.67	1124
micro avg	0.66	0.66	0.66	1885
macro avg	0.67	0.68	0.66	1885
weighted avg	0.70	0.66	0.66	1885



Acurasy compare

	name	TPR	FPR
0	gnb	0.673469	0.285714
1	dtree	0.673469	0.424242
2	rfc	0.706633	0.337662
3	knn	0.681122	0.402597

7.Wyniki

Dało się znaleźć odpowiedzi na interesujące pytania. Najlepsze wyniku pokazały algorytmy Random Forest i Gaussian Naive Bias. To było zaskakująco, ponieważ przy takich danych które mieliśmy trudno było-bym dla człowieka oszacować wartość wyniku patrosząc na wykresy danych.