Stroke Prediction Dataset

źródło: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

Baza zawiera 5110 rekordów z 12 atrybutami, na podstawie których prognozujemy szanse, na wystąpienie zawału.

Informacje o kolumnach:

- id: unikalny identyfikator
- gender: płeć – „Male”, „Female” lub „Other”
- age: wiek pacjenta
- hypertenstion: 0 jeśli pacjent nie ma nadciśnienia, 1 jeśli pacjent ma nadciśnienie
- heart_disease: 0 jeśli pacjet nie ma problemów z sercem, 1 jeśli ma
- ever_married: zaręczony – „No” lub „Yes”
- work_type: praca – "children", "Govt_jov", "Never_worked", "Private" ,"Self-employed"
- Residence_type: miejsce zamieszkania – „Rural” lub „Urban”
- avg_glucose_level: średni poziom glukozy we krwi
- bmi: wskaźnik masy ciała
- smoking_status: „formerly_smoked”, „never smoked”, „smokes” lub „Unknown”.
- „stroke”: 1 jeśli pacjent miał zawał, 0 jeśli nie miał

Załadowanie danych

```
train = pd.read_csv("healthcare-dataset-stroke-data.csv")

print(train.head())
```

```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2        Private          Rural             105.92  32.5     never smoked
3        Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```

Zduplikowane wiersze:

```
print("Liczba zduplikowanych wierszy:", train.duplicated().sum())
```

```
Liczba zduplikowanych wierszy: 0
```

Dane kategoryczne:

```
categorical = train.select_dtypes(include=['object']).columns.tolist()
for i in categorical:
    print(train[i].value_counts().to_frame(), '\n')
```

	gender
Female	2994
Male	2115
Other	1

	ever_married
Yes	3353
No	1757

	work_type
Private	2925
Self-employed	819
children	687
Govt_job	657
Never_worked	22

	Residence_type
Urban	2596
Rural	2514

	smoking_status
never smoked	1892
Unknown	1544
formerly smoked	885
smokes	789

Z powyższej analizy widzimy, że tylko jedna osoba została zidentyfikowana jako „Other” w kolumnie płeć. Możemy usunąć ten wiersz ze zbioru danych, gdyż nie jest on zbyt znaczący dla predykcji. W predykcji nie wykorzystamy także kolumny smoking_status, która zawiera aż 1544 rekordów ‘Unknown”.

Modyfikacja danych

Usunięcie kolumny z „id”:

```
dataset = train.drop('id', axis=1)
```

Usunięcie wiersza z płcią „Other”, gdyż występuje tylko raz w naszych danych.

```
dataset = dataset[dataset['gender'] != 'Other']
```

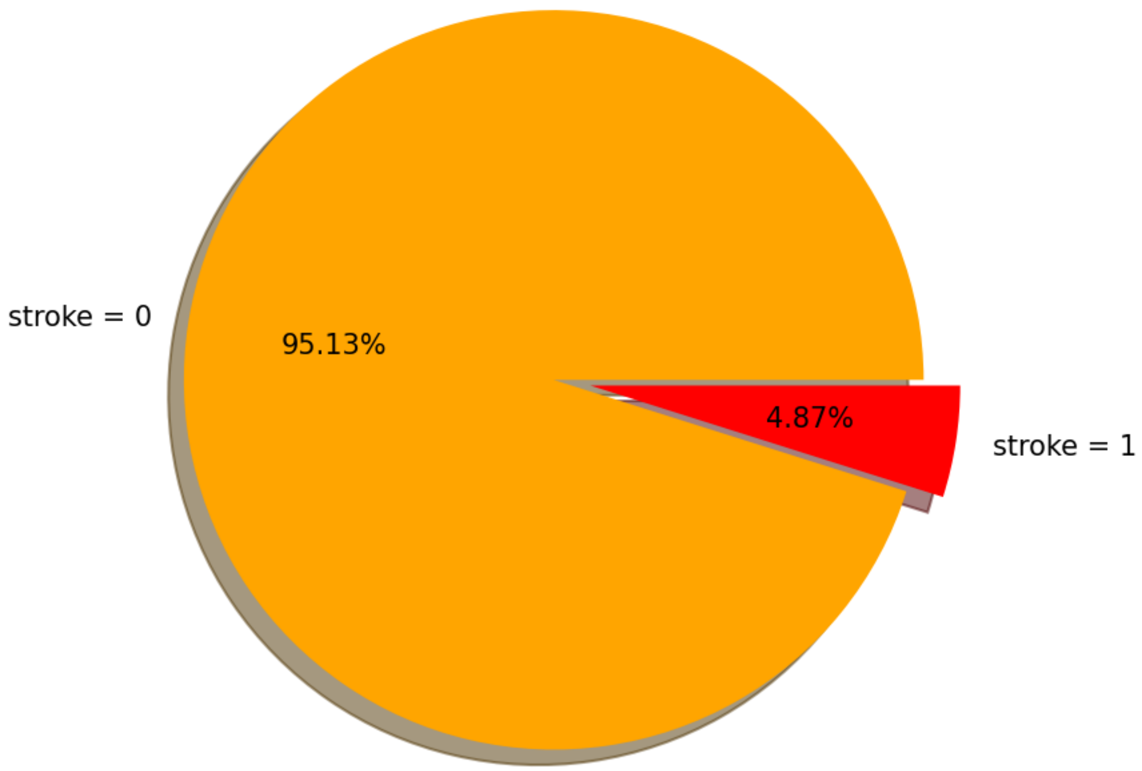
Sprawdzenie brakujących danych

```
print(dataset.isna().sum())
>>
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

Uzupełnimy brakujące dane dla kolumny **bmi** średnią

```
dataset['bmi'].fillna(dataset['bmi'].mean(), inplace = True)
print(dataset.isna().sum())
>>
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

```
data_balance_check_labels = ['stroke = 0', 'stroke = 1']
total_instances_per_value = df['stroke'].value_counts()
pie_chart_colors = ['orange', 'red']
plt.figure(figsize=(6,6))
plt.pie(total_instances_per_value, labels = data_balance_check_labels,
shadow = 1, explode = (0.1, 0), autopct='%1.2f%%', colors =
pie_chart_colors)
plt.show()
```



Wykres pokazujący rozkład wierszy w których stroke = 1 lub stoke = 0

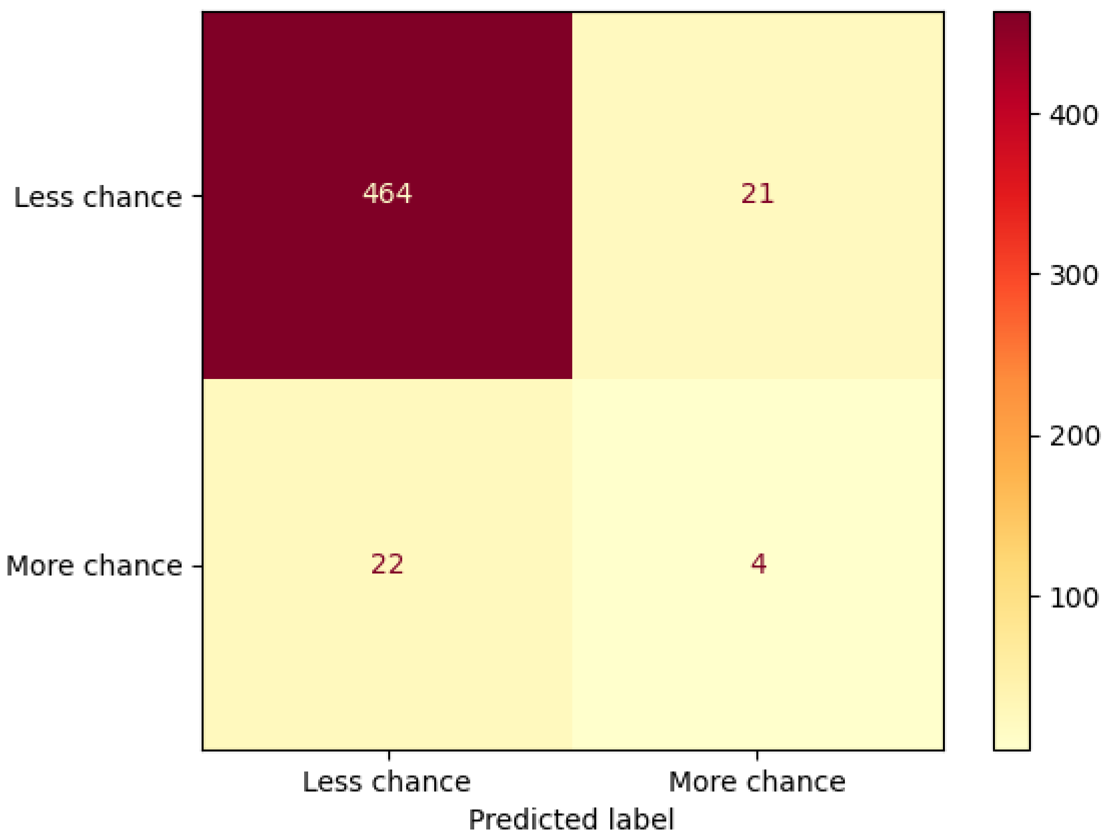
kNN

```
import sklearn
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
df = dataset[["age", "hypertension", "heart_disease","avg_glucose_level",
"bmi", "stroke"]]
print(df)
predict = "stroke"
x = np.array(df.drop([predict], 1))
y = np.array(df[predict])
x_train, x_test, y_train, y_test =
sklearn.model_selection.train_test_split(x, y, test_size=0.1)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train, y_train)
from sklearn import metrics
y_pred1 = knn.predict(x_test)
acc1 = metrics.accuracy_score(y_test, y_pred1)
accuracy_1_rounded = round(acc1*100, 2)
print("Accuracy k=1:", accuracy_1_rounded, "% \n")
```

k = 1

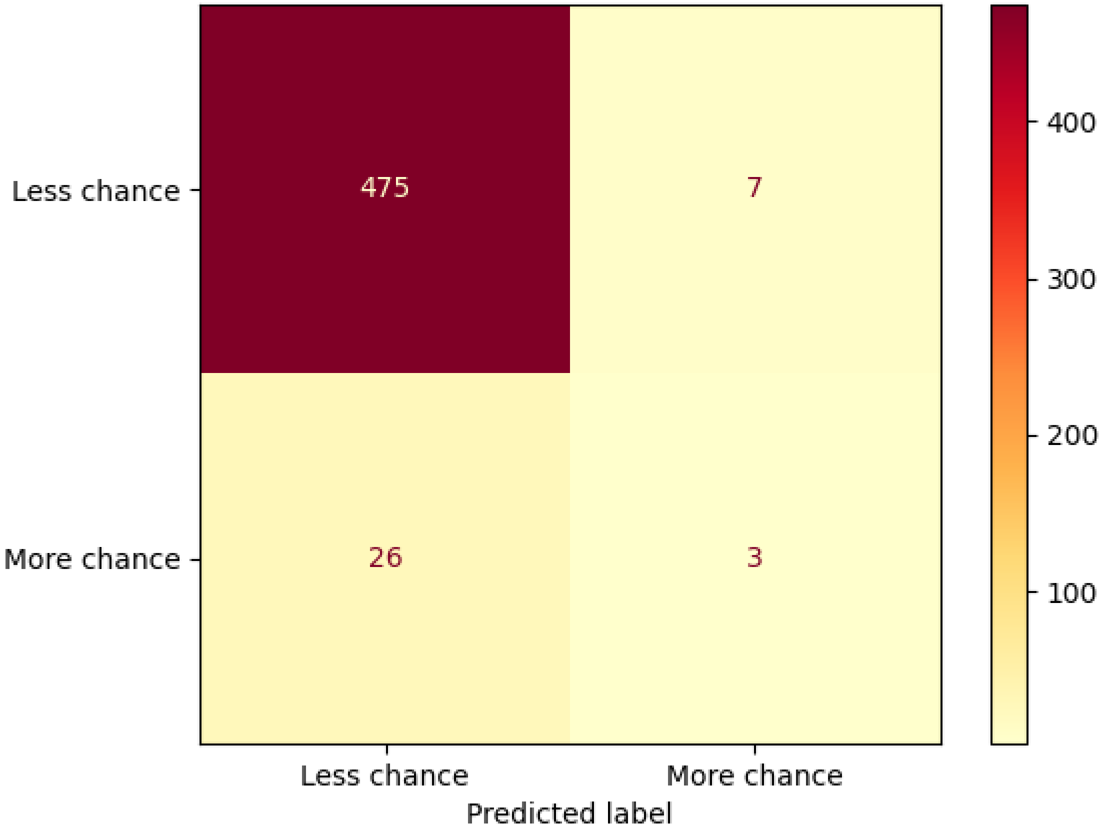
```
Accuracy k=1: 91.0 %
```

```
plot_confusion_matrix(knn, x_test, y_test, display_labels=["Less chance",
"More chance"], cmap=plt.cm.YlOrRd)
plt.show()
```



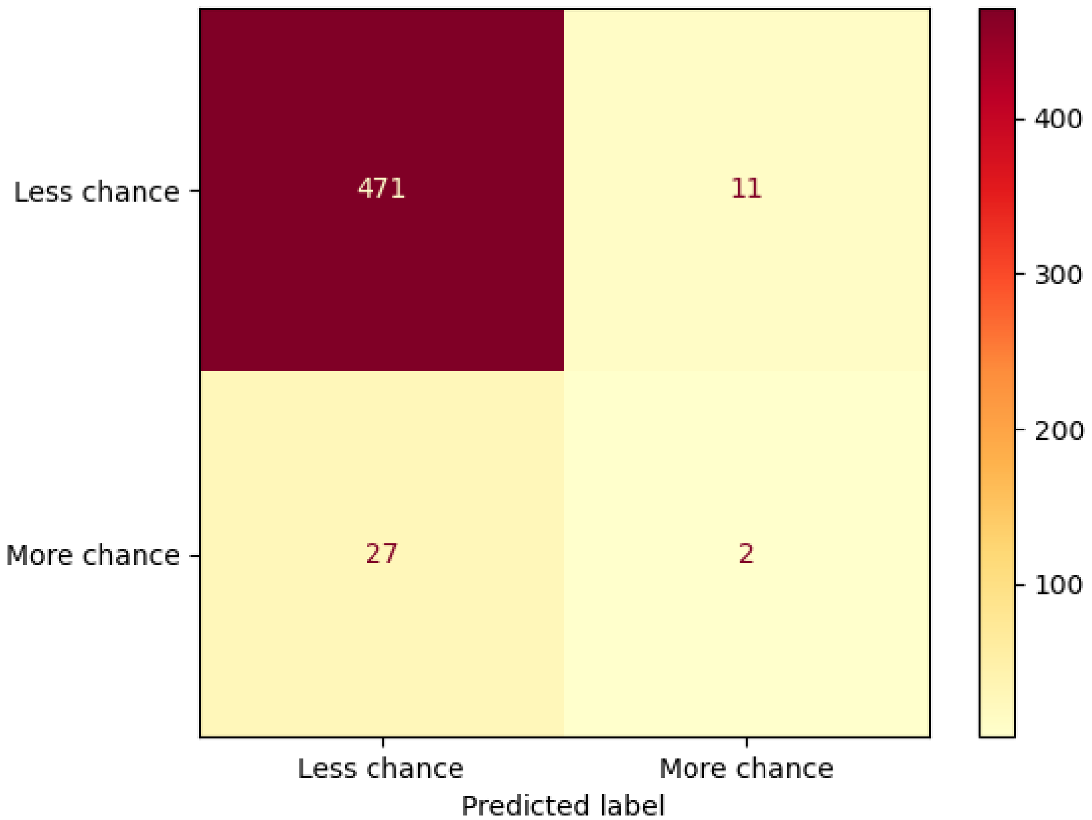
k = 3

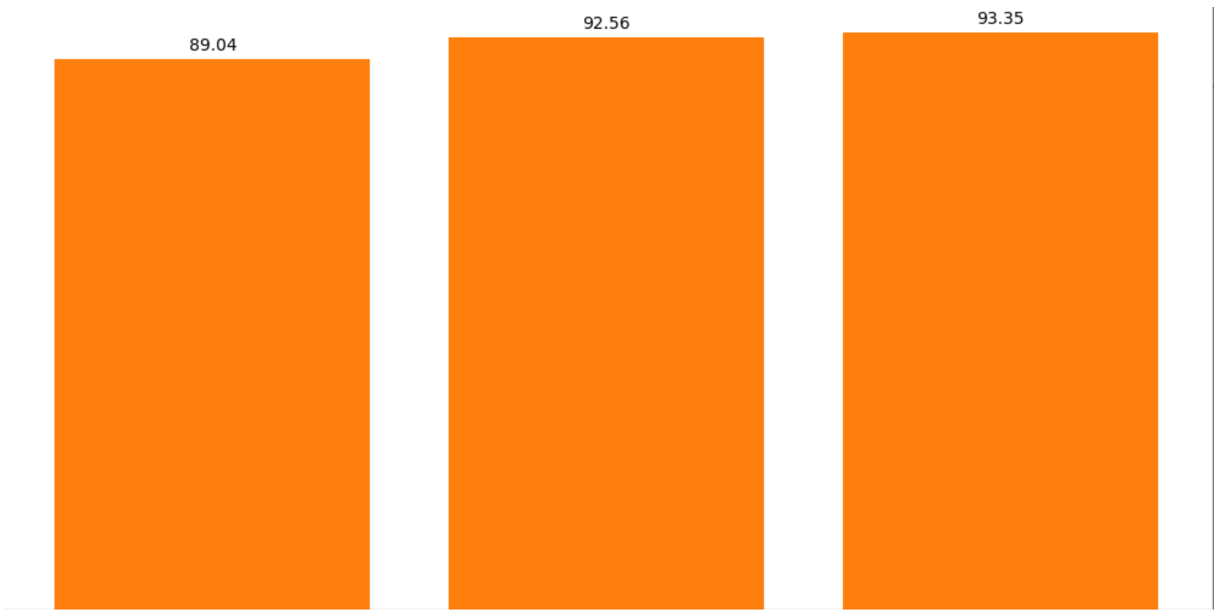
```
Accuracy k=3: 93.54 %
```



k = 5

```
Accuracy k=5: 93.35 %
```





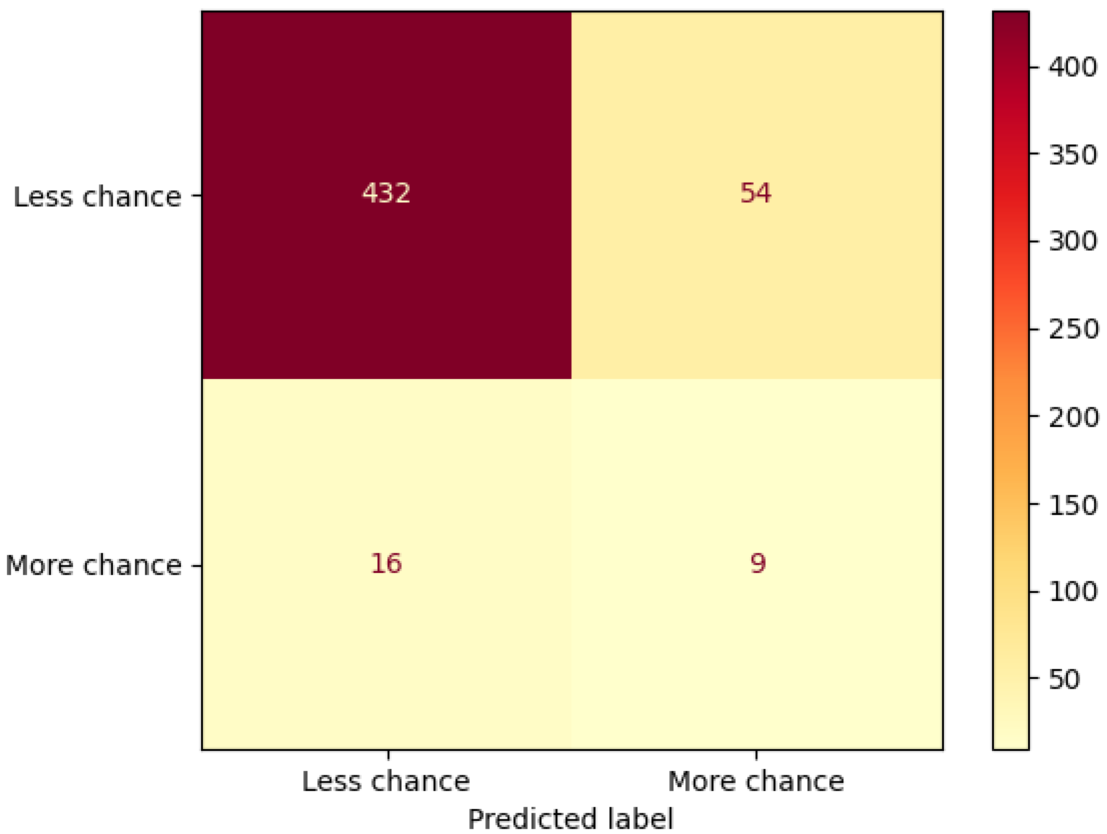
k = 1, k = 3, k = 5

Wniosek: Dokładność KNN maleje wraz ze spadkiem liczby sąsiadów.

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)
accuracy_bayes = metrics.accuracy_score(y_test, y_pred)
accuracy_bayes_round = round(accuracy_bayes*100, 2)
print("Accuracy Naive Bayes:", accuracy_bayes_round, "% \n")
plot_confusion_matrix(gnb, x_test, y_test, display_labels=["Less chance",
"More chance"], cmap=plt.cm.YlOrRd)
plt.show()
```

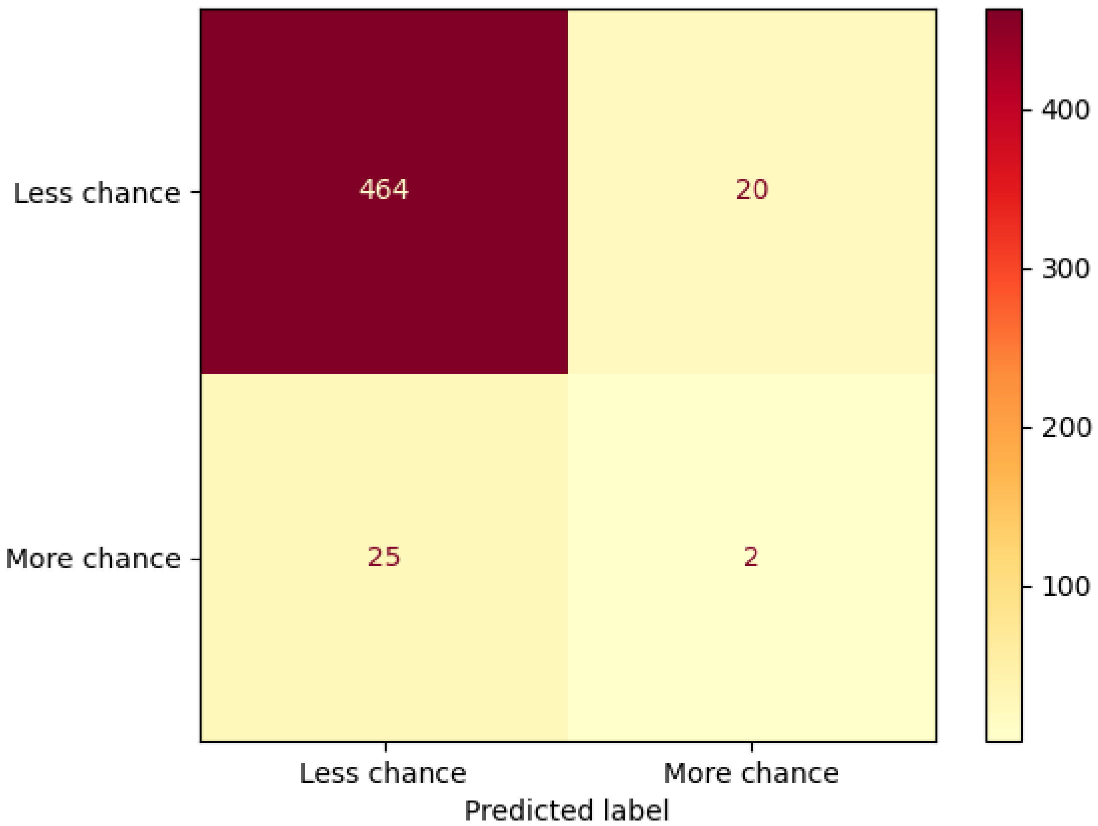
```
Accuracy Naive Bayes: 86.3 %
```



Drzewa decyzyjne

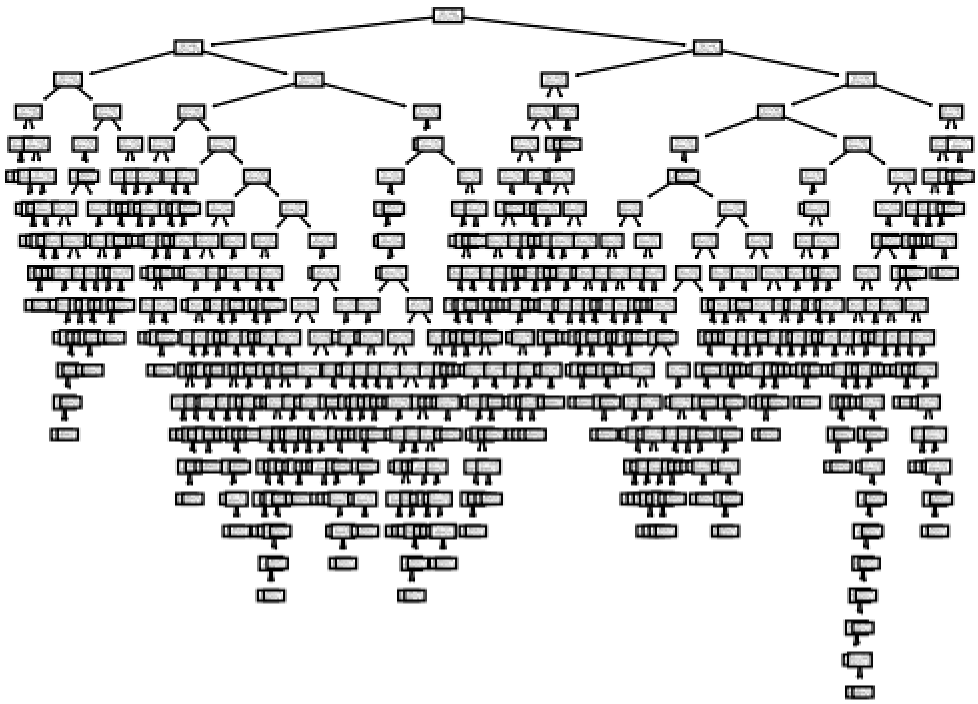
```
from sklearn import metrics, tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy_tree = accuracy_score(y_test, y_pred)
accuracy_tree_round = round(accuracy_tree*100,2)
print("Accuracy decision tree: ", accuracy_tree_round, "% \n")
print("Confusion matrix:")
plot_confusion_matrix(clf, x_test, y_test, display_labels=["Less chance",
"More chance"], cmap=plt.cm.YlOrRd)
plt.show()
```

```
Accuracy decision tree:  91.19 %
```



Drzewo decyzyjne w postaci grafu

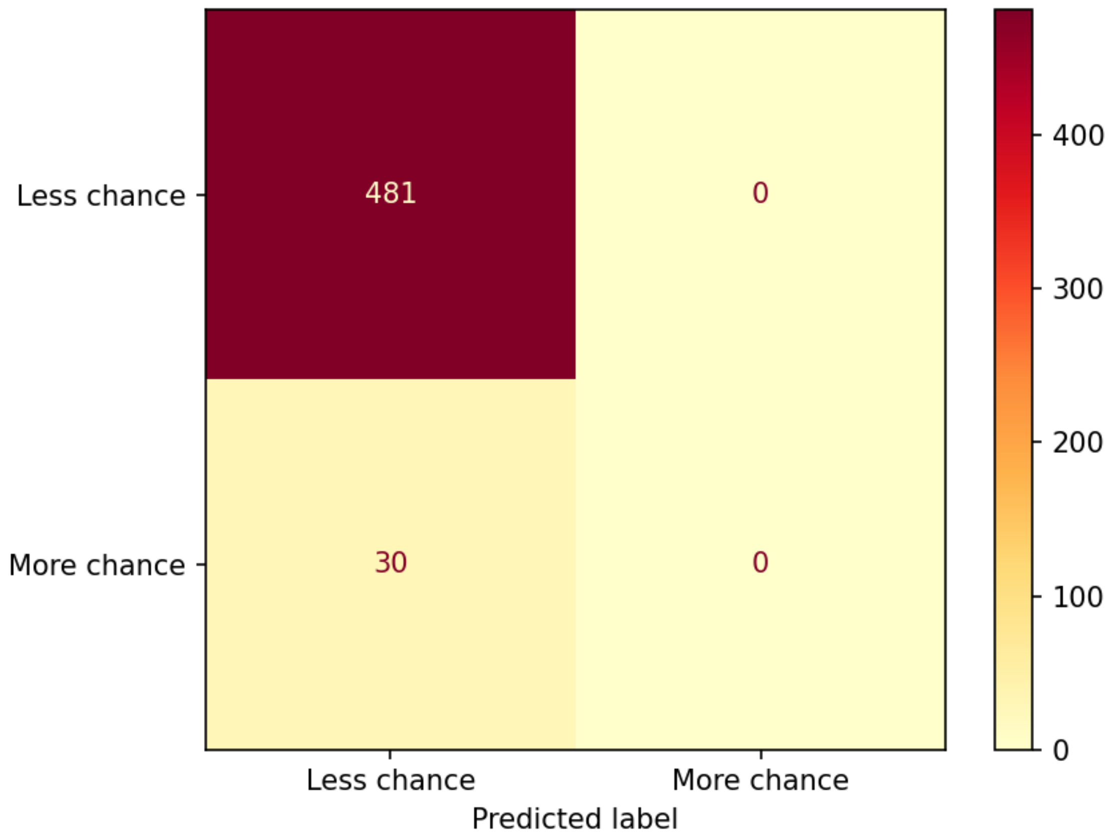
```
tree.plot_tree(clf)
plt.savefig('tree.pdf
```



Regresja logistyczna

```
from sklearn.linear_model import LogisticRegression
LogisticRegressionclf = LogisticRegression(random_state=0, max_iter = 400)
LogisticRegressionclf.fit(x_train,y_train)
y_predict_test = LogisticRegressionclf.predict(x_test)

cm = confusion_matrix(y_test, y_predict_test)
print(classification_report(y_test, y_predict_test))
print('Accuracy Logistic Regression: ' ,
accuracy_score(y_test,y_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       481
           1       0.00      0.00      0.00        30

    accuracy                           0.94       511
   macro avg       0.47      0.50      0.48       511
weighted avg       0.89      0.94      0.91       511

Accuracy Logistic Regression:  0.9412915851272016
```



Sieci neuronowe

Siec wykorzystuje funkcję aktywacji „relu”

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(2, input_shape=(x_train.shape[1],), activation =
'relu'),
    tf.keras.layers.Dense(2),
    tf.keras.layers.Softmax()])
model.summary()
model.compile(optimizer = 'Adam', loss =
tf.keras.losses.BinaryCrossentropy(), metrics = ['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 38

 dense_1 (Dense)             (None, 2)                 6

 softmax (Softmax)           (None, 2)                 0

=================================================================
Total params: 44
Trainable params: 44
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x_train, y_train, epochs = 200, validation_split=0.2, verbose=1)
model.evaluate( x_test, y_test)
```

```
16/16 [==============================] - 0s 750us/step - loss: 0.6931 - accuracy: 0.9667
```

Accuracy: 0.9667

Skuteczność klasyfikatorów