

# Ryzyko Zawału Serca

Łukasz Nowosielski

## 1 Baza danych

Do mojego projektu starłem się znaleźć bazę danych, która zawiera przynajmniej 20 kolumn i więcej niż 5000 rekordów. Zawały serca, czyli zawały mięśnia sercowego, w dalszym ciągu stanowią istotny problem zdrowotny na świecie. Wybrana baza danych obejmuje różnorodne czynniki, w tym wiek, poziom cholesterolu, ciśnienie krwi, nawyki palenia, aktywność fizyczna, preferencje żywieniowe i wiele innych, a celem analizy tych danych jest zrozumienie wzajemnego oddziaływania tych czynników w określaniu prawdopodobieństwa zawału serca.

### Znaczenie poszczególnych kolumn:

- Patient ID - Id pacjenta
- Age - Wiek
- Sex - Płeć (Mężczyzna/Kobieta)
- Cholesterol - Poziom cholesterolu
- Blood Pressure - Ciśnienie krwi (Skurczowe/Rozkurczowe)
- Heart Rate - Tętno
- Diabetes - Cukrzyca (1:Tak, 0:Nie)
- Family History - Czy w rodzinie miał ktoś problemy z sercem (1:Tak, 0:Nie)
- Smoking - Czy pacjent pali papierosy (1:Tak, 0:Nie)
- Obesity - Czy pacjent jest otyły (1:Tak, 0:Nie)
- Alcohol Consumption - Poziom spożycia alkoholu (Brak/Niskie/Umiarkowane/Wysokie)
- Exercise Hours Per Week - Liczba godzin spędzonych na ćwiczeniach w tygodniu
- Diet - Nawyki żywieniowe (Zdrowe/Średnie/Niezdrowe)
- Previous Heart Problems - Czy pacjent miał już problemy z sercem (1:Tak, 0:Nie)
- Medication Use - Zażywanie leków (1:Tak, 0:Nie)
- Stress Level - Poziom stresu (1-10)
- Sedentary Hours Per Day - Liczba godzin dziennie siedząc
- Income - Dochód
- BMI - Wskaźnik masy ciała
- Triglycerides - Poziom trójglicerydów
- Physical Activity Days Per Week - Liczba godzin spędzonych aktywnie w tygodniu
- Sleep Hours Per Day - Liczba godzin snu
- Country - Kraj pacjenta
- Continent - Kontynent, na którym przebywa pacjent
- Hemisphere - Półkula, na której przebywa pacjent
- Heart Attack Risk - Ryzyko zawału Serca (1:Tak, 0:Nie)

```
df = pd.read_csv("dataset.csv")
print(df.shape)
(8763, 26)
print(f"Brakujące wartości: {df.isnull().sum().sum()}")
Brakujące wartości: 0

print(df.head())
```

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	...	Sleep Hours	Per Day	Country	Continent	Hemisphere	Heart Attack Risk
0	BMW7812	67	Male	208	158/88	...	6		Argentina	South America	Southern Hemisphere	
1	CZE1114	21	Male	389	165/93	...	7		Canada	North America	Northern Hemisphere	
2	BNI9906	21	Female	324	174/99	...	4		France	Europe	Northern Hemisphere	
3	JLN3497	84	Male	383	163/100	...	4		Canada	North America	Northern Hemisphere	0
4	GF08847	66	Male	318	91/88	...	5		Thailand	Asia	Northern Hemisphere	0

## 2 Preprocessing

Na początku dokonałem wstępnego preprocessingu danych. W tym celu usunąłem wiersze z brakującymi elementami, pozbyłem się kolumn: Patient ID, Blood Pressure oraz Country. Zmapowałem także wartości kategoryjne na liczby i rozdzieliłem kolumnę Blood Pressure (ciśnienie krwi) na Systolic (skurczowe) oraz Diastolic (rozkurczowe).

```
mappings = {
    "Sex": {"Male": 0, "Female": 1},
    "Diet": {"Healthy": 0, "Average": 1, "Unhealthy": 2},
    "Hemisphere": {"Northern Hemisphere": 0, "Southern Hemisphere": 1},
    "Continent": {
        "Asia": 0,
        "Europe": 1,
        "South America": 2,
        "Australia": 3,
        "Africa": 4,
        "North America": 5,
    },
}

for column, mapping in mappings.items():
    df[column] = df[column].map(mapping)

df[["Systolic", "Diastolic"]] = (
    df["Blood Pressure"].str.strip().str.split("/", expand=True)
)
df["Systolic"] = df["Systolic"].astype(int)
df["Diastolic"] = df["Diastolic"].astype(int)

df = df.dropna()
df = df.drop(columns=["Patient ID", "Blood Pressure", "Country"])
```

Następnie dokonałem upsamplingu na danych, aby zrównoważyć klasy. Jest to ważne, ponieważ nierównomierny rozkład klas może prowadzić do błędów w modelu, gdzie model może być stronniczy wobec klasy większościowej. Przez zrównoważenie klas, model ma równe szanse na naukę z obu klas, co zwykle prowadzi do lepszej ogólnej wydajności. Skorzystałem przy tym z techniki SMOTE, która generuje syntetyczne próbki klasy mniejszościowej poprzez kopiowanie istniejących danych i wprowadzając do nich niewielkie zmiany. Jest to lepsze rozwiązanie niż powielanie poprzednich próbek, które mogłoby doprowadzić do overfittingu.

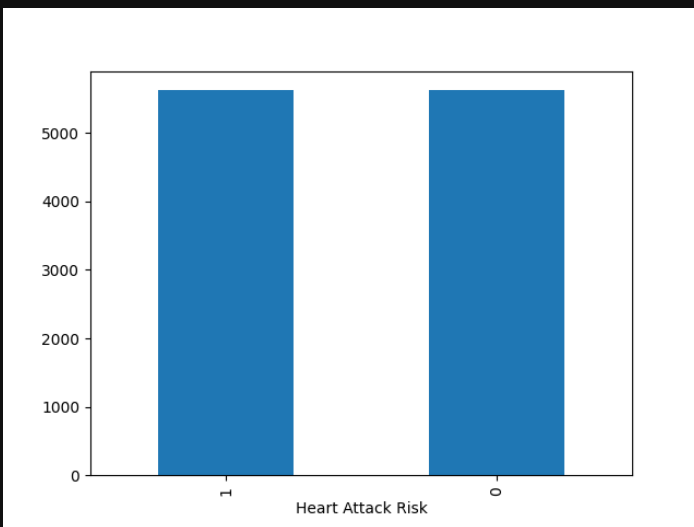
```
def balance_data(df):
    sm = SMOTE(random_state=285806)

    X = df.drop(["Heart Attack Risk"], axis=1)
    y = df["Heart Attack Risk"]

    X_res, y_res = sm.fit_resample(X, y)

    df_res = pd.concat([X_res, y_res], axis=1)

    return df_res
```



```
df.shape przed upsampling:
(8763, 25)
df.shape po upsampling:
(11248, 25)
```

### 3 Klasyfikacja

Wybraną bazę danych przebadalem pod kątem klasyfikacji, aby wybrać taki klasyfikator, który ma najwyższą dokładność oraz najwyższą wartość AUC-ROC. AUC-ROC, czyli Area Under the Receiver Operating Characteristic Curve, to metryka oceny modeli klasyfikacyjnych, która mierzy zdolność modelu do odróżniania między prawdziwie pozytywnymi, a fałszywie pozytywnymi wynikami. Wysoka wartość AUC-ROC wskazuje na to, że model jest skuteczny w rozróżnianiu między klasami. Na koniec sprawdziłem macierz błędów, która pozwoliła mi zobaczyć jak dobrze dany model radzi sobie z prawdziwie pozytywnymi i prawdziwie negatywnymi wynikami, a także jak często występują fałszywe pozytywne i fałszywe negatywne wyniki.

Porównałem 5 różnych klasyfikatorów:

- Klasyfikator Lasów Losowych
- Drzewo Decyzyjne
- Naiwny Bayes
- K-najbliższych sąsiadów  $k=3$ ,  $k=5$ ,  $k=11$
- Sieci Neuronowe

Na początku podzieliłem dane na zestaw treningowy i zestaw testowy:

```
def split_data(df):  
    X = df.drop(["Heart Attack Risk"], axis=1)  
    y = df["Heart Attack Risk"]  
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)  
  
    return X_train, X_test, y_train, y_test
```

Następnie przeskalowałem dane za pomocą *StandardScaler* z biblioteki **sklearn**, która przeskalowuje każdą cechę tak, aby miała średnią równą zero i jednostkową wariancję.

```
def scale_data(X_train, X_test):  
    scaler = StandardScaler()  
    X_train = scaler.fit_transform(X_train)  
    X_test = scaler.transform(X_test)  
  
    return X_train, X_test
```

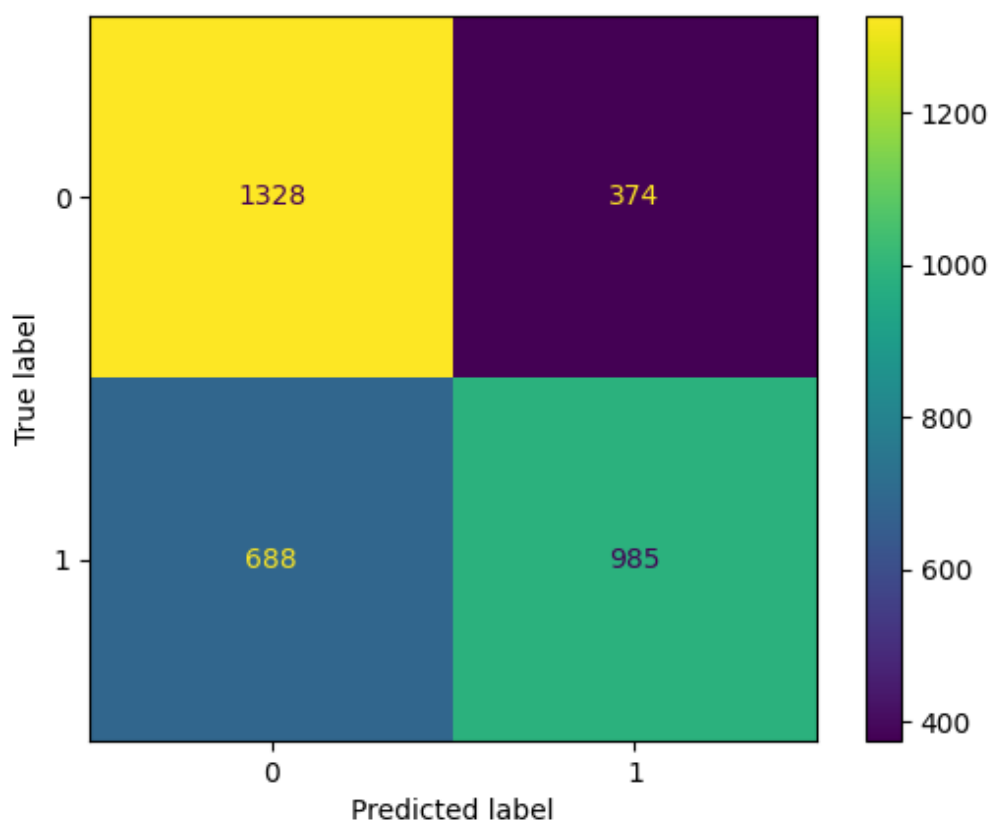
## 1. Klasyfikator Lasów Losowych

```
def rfc(X_train, y_train, X_test, y_test):  
    clf = RandomForestClassifier(random_state=285806)  
    clf = clf.fit(X_train, y_train)  
    predict = clf.predict(X_test)  
    acc = accuracy_score(y_test, predict) * 100  
    roc = roc_auc_score(y_test, predict) * 100  
    print("Random Forest")  
    print(f"Accuracy: {acc}%")  
    print(f"AUC ROC: {roc}%")  
  
    cm = confusion_matrix(y_test, predict)  
    cfd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)  
    cfd.plot()  
    plt.show()
```

Random Forest

Accuracy: 68.53333333333333%

AUC ROC: 68.45106105611836%



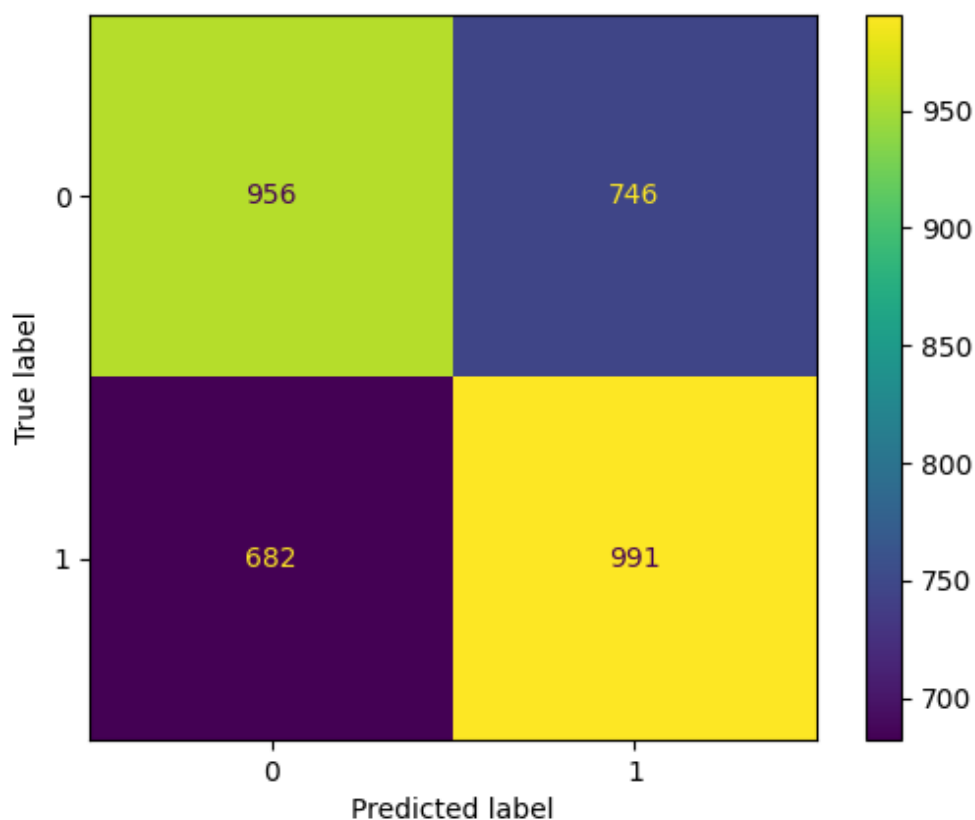
## 2. Drzewo decyzyjne

```
def dtc(X_train, y_train, X_test, y_test):  
    clf = DecisionTreeClassifier(random_state=285806)  
    clf = clf.fit(X_train, y_train)  
    predict = clf.predict(X_test)  
    acc = accuracy_score(y_test, predict) * 100  
    roc = roc_auc_score(y_test, predict) * 100  
    print("Decision Tree")  
    print(f"Accuracy: {acc}%")  
    print(f"AUC ROC: {roc}%")  
  
    cm = confusion_matrix(y_test, predict)  
    cfd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)  
    cfd.plot()  
    plt.show()
```

Decision Tree

Accuracy: 57.68888888888889%

AUC ROC: 57.70206002150699%



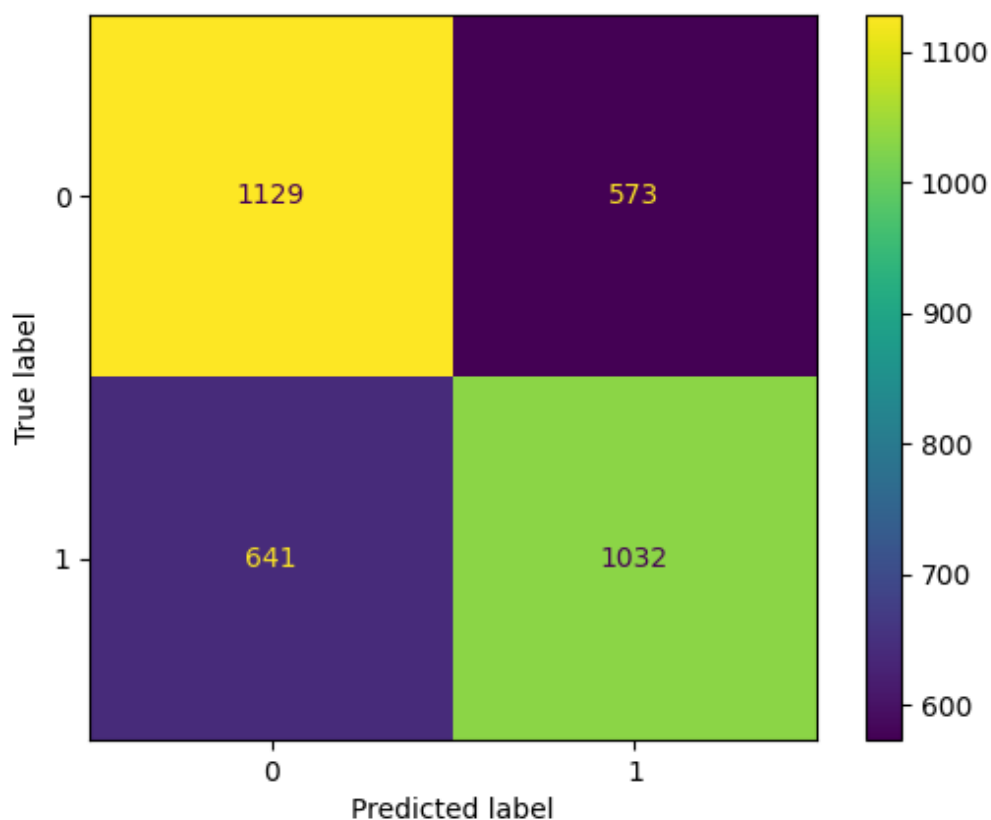
### 3. Naiwny Bayes

```
def gaussian(X_train, y_train, X_test, y_test):  
    gnb = GaussianNB()  
    gnb.fit(X_train, y_train)  
    y_pred = gnb.predict(X_test)  
    acc = accuracy_score(y_test, y_pred) * 100  
    roc = roc_auc_score(y_test, y_pred) * 100  
    print("Naive Bayes")  
    print(f"Accuracy: {acc}%")  
    print(f"AUC ROC: {roc}%")  
  
    cm = confusion_matrix(y_test, y_pred)  
    cfd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=gnb.classes_)  
    cfd.plot()  
    plt.show()
```

Naive Bayes

Accuracy: 64.02962962962962%

AUC ROC: 64.00965988468262%



#### 4. K-najbliższych sąsiadów

```
def knn(X_train, y_train, X_test, y_test):  
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))  
  
    for i, ax in zip([3, 5, 11], axes):  
        knc = KNeighborsClassifier(n_neighbors=i)  
        knc.fit(X_train, y_train)  
        y_pred = knc.predict(X_test)  
  
        acc = accuracy_score(y_test, y_pred) * 100  
        roc = roc_auc_score(y_test, y_pred) * 100  
        print(f"{i}NN")  
        print(f"Accuracy: {acc}%")  
        print(f"AUC ROC: {roc}%")  
  
        cm = confusion_matrix(y_test, y_pred)  
        cfd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knc.classes_)  
        cfd.plot(ax=ax)  
        ax.title.set_text(f"{i}-NN")  
  
    plt.tight_layout()  
    plt.show()
```

3NN

Accuracy: 63.25925925925926%

AUC ROC: 63.365012014275244%

5NN

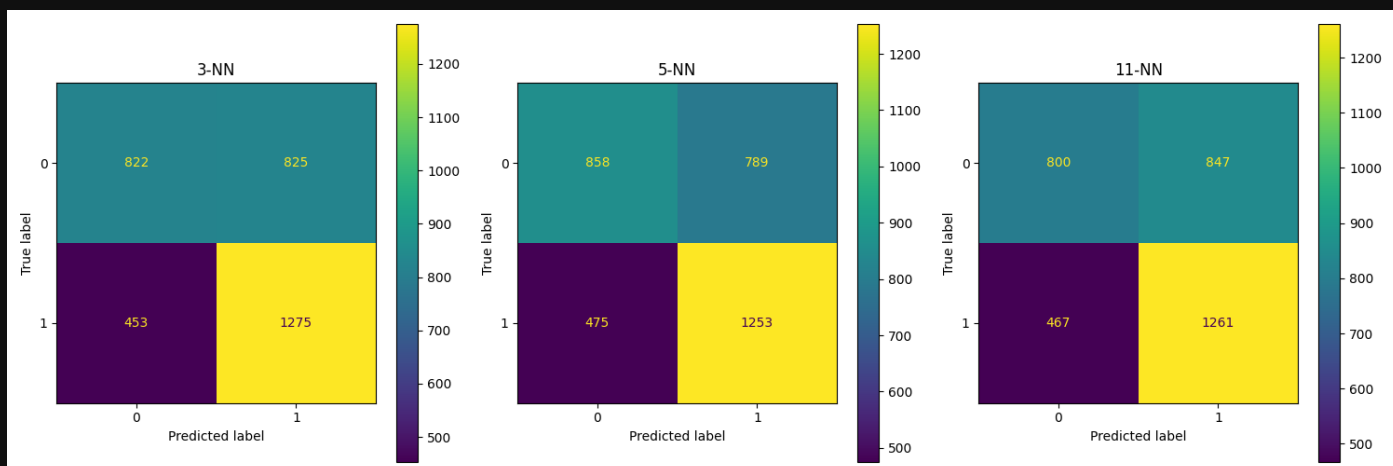
Accuracy: 63.05185185185185%

AUC ROC: 63.15835313470387%

11NN

Accuracy: 60.53333333333333%

AUC ROC: 60.64346786558902%





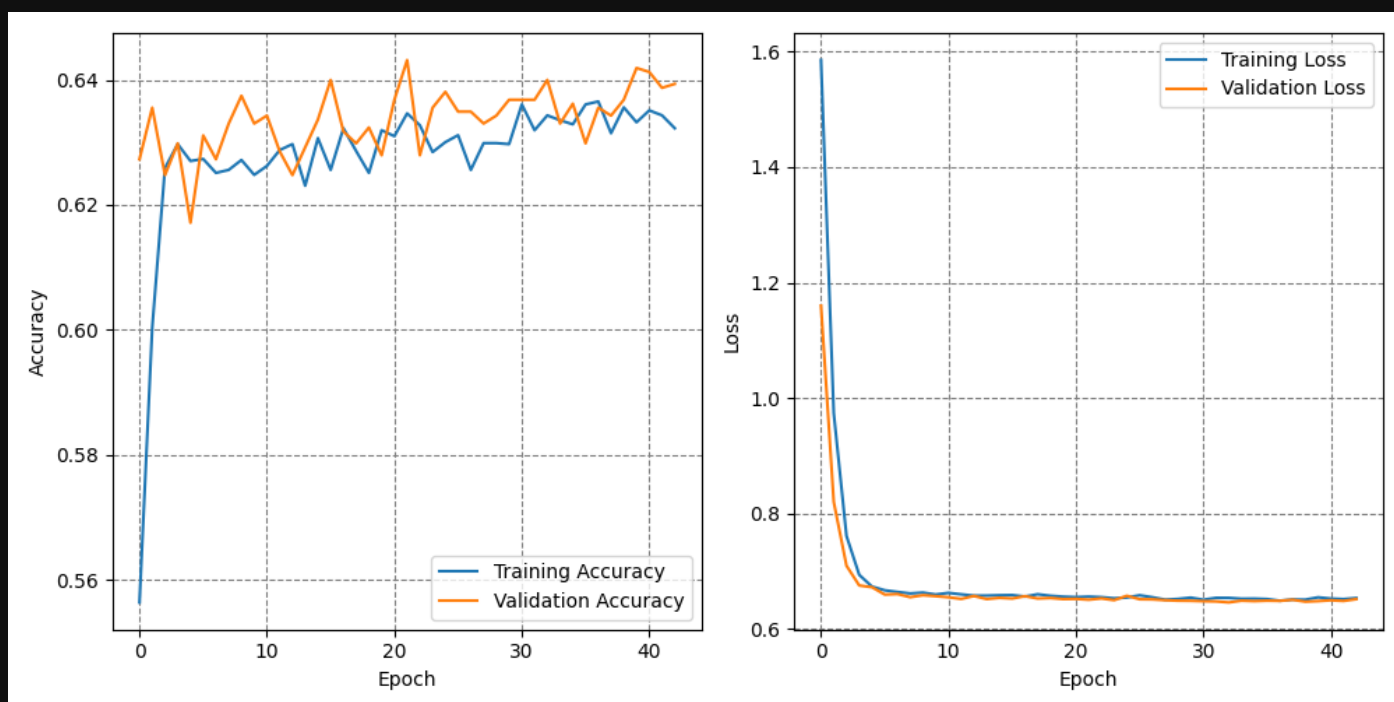
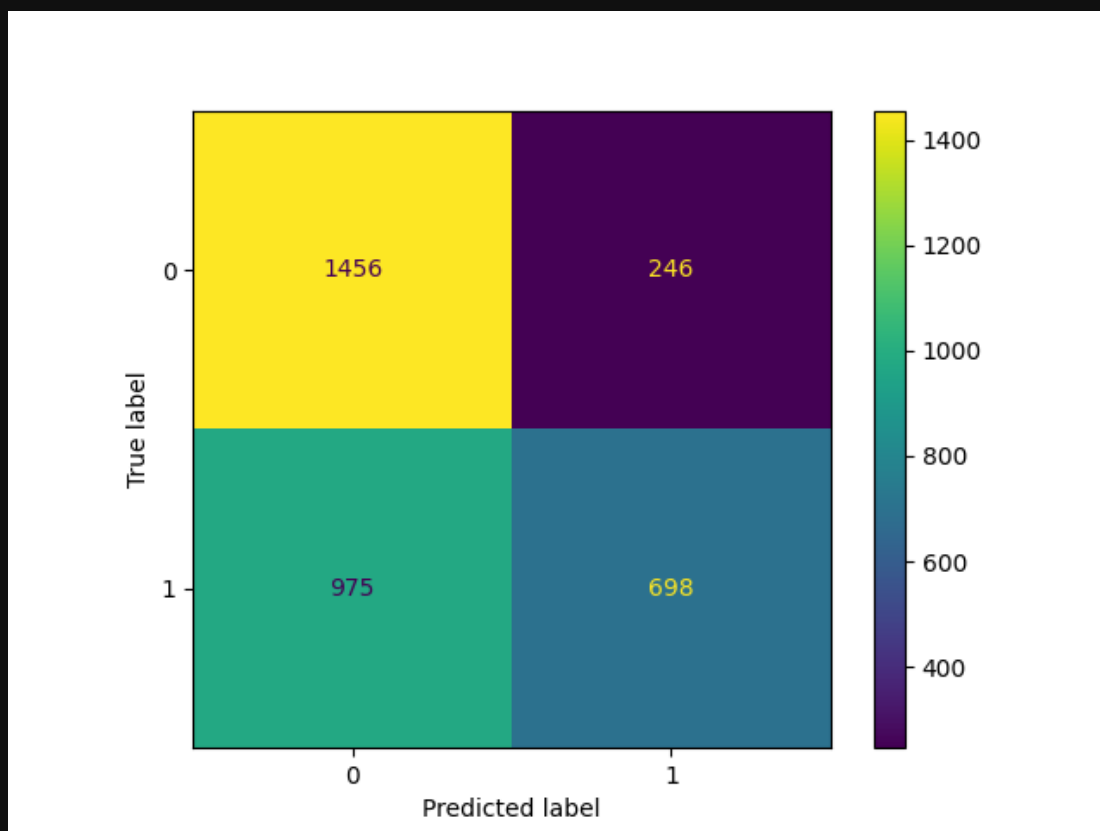
## 5. Sieci neuronowe

```
def neural_network(X_train, y_train, X_test, y_test):  
    model = Sequential(  
        [  
            Dense(  
                64,  
                activation="relu",  
                input_shape=(X_train.shape[1],),  
                kernel_regularizer=l2(0.01),  
            ),  
            Dropout(0.5),  
            Dense(64, activation="relu", kernel_regularizer=l2(0.01)),  
            Dense(32, activation="selu", kernel_regularizer=l2(0.01)),  
            Dropout(0.5),  
            Dense(1, activation="sigmoid"),  
        ]  
    )  
    model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])  
    early_stopping = EarlyStopping(monitor="val_loss", patience=10)  
    history = model.fit(  
        X_train,  
        y_train,  
        epochs=50,  
        batch_size=16,  
        validation_split=0.2,  
        callbacks=[early_stopping],  
        verbose=0,  
    )  
  
    _, test_acc = model.evaluate(X_test, y_test)  
    roc = roc_auc_score(y_test, model.predict(X_test)) * 100  
    print("Neural Network")  
    print(f"Accuracy: {test_acc * 100}%")  
    print(f"AUC ROC: {roc}%")  
  
    cm = confusion_matrix(  
        y_test,  
        np.round(model.predict(X_test)),  
    )  
    cfd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])  
    cfd.plot()  
    plt.show()  
  
    return history
```

Neural Network

Accuracy: 63.822221755981445%

AUC ROC: 67.494589888623%



Jak widzimy model ten można nazwać dość dobrze dopasowanym, ponieważ osiąga podobne dokładność i stratę na zestawie treningowym i walidacyjnym, a do tego są one stabilne w czasie.

## 4 Reguły asocjacyjne

Na początku przekształciłem dane, dzieląc wartości numeryczne wybranych kolumn na przedziały "Low", "Medium" i "High", podczas gdy pozostałe kolumny pozostały numeryczne, żeby móc później łatwiej wyszukać ciekawe zależności. Następnie użyłem algorytmu *fpgrowth* do znalezienia często występujących zestawów elementów i wygenerowania z nich reguł asocjacyjnych. Wybrałem *fpgrowth* zamiast *apriori*, ponieważ jest bardziej wydajny oraz nie generuje i nie przechowuje dużych zestawów, co jest korzystne, gdy mamy do czynienia z dużymi zestawami danych. Następnie zbadałem kilka ciekawych zależności między różnymi czynnikami np. palenie papierosów, występowanie cukrzycy, płeć pacjenta, otyłość, a ryzykiem zawału serca.

```
def create_association_rules(df):
    selected_columns = [
        "Age",
        "Cholesterol",
        "Heart Rate",
        "Exercise Hours Per Week",
        "Sedentary Hours Per Day",
        "Income",
        "BMI",
        "Triglycerides",
        "Physical Activity Days Per Week",
        "Sleep Hours Per Day",
        "Systolic",
        "Diastolic",
    ]
    for col in selected_columns:
        df[col] = pd.cut(df[col], bins=3, labels=["Low", "Medium", "High"])

    for col in df.columns:
        if col in selected_columns:
            continue
        unique_values = df[col].nunique()
        labels = [i for i in range(unique_values)]
        df[col], _ = pd.cut(df[col], bins=unique_values, labels=labels, retbins=True)

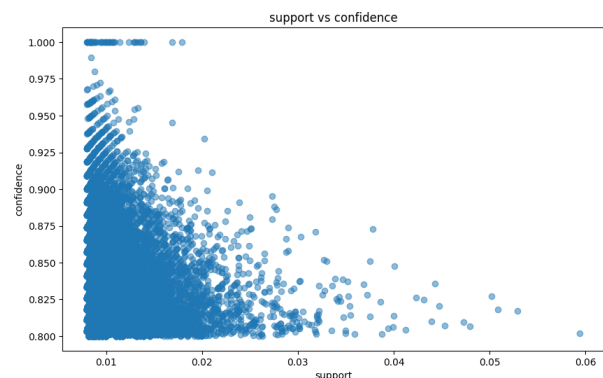
    encoded_df = pd.get_dummies(df)

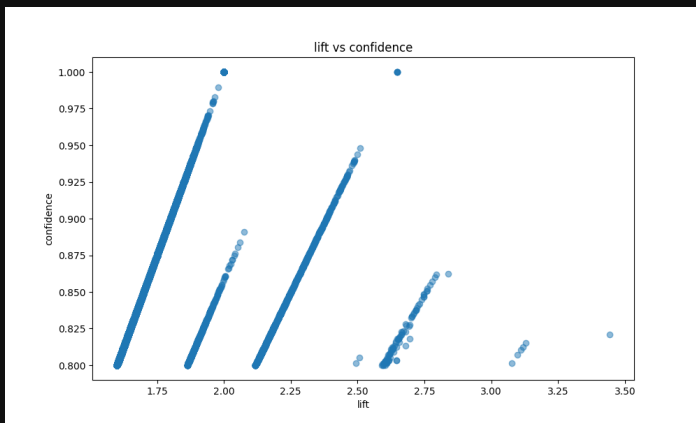
    frequent_itemsets = fpgrowth(encoded_df, min_support=0.008, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)

    rules = rules[rules["consequents"].apply(lambda x: "Heart Attack Risk_1" in str(x))]

    return rules
```

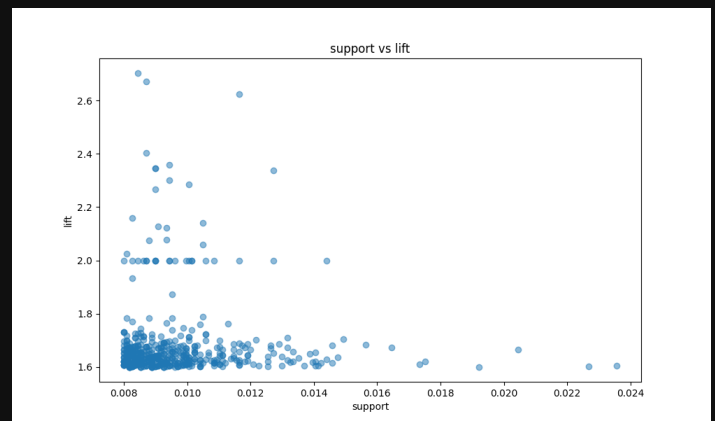
Ten wykres przedstawia zależności między osobami, które palą papierosy, a ryzykiem zawału serca. Wykorzystałem wskaźniki wsparcia i pewności, aby zrozumieć, jak często te dwie zmienne występują razem i jak silna jest ta zależność. Większość reguł ma niższe wsparcie i pewność, co widać po skupisku punktów w lewym dolnym rogu wykresu. Jednakże, istnieją również reguły z wyższym wsparciem i pewnością. To sugeruje, że choć większość reguł dotyczących palenia papierosów i ryzyka ataku serca, które spełniły minimalne wymagania występują rzadziej w danych, istnieją również reguły, które są częściej obserwowane i mają silniejszą zależność.





Ten wykres przedstawia zależność między pacjentami płci żeńskiej, a ryzykiem zawału serca dla wskaźników pewności (confidence) oraz lift. Wartości lift powyżej 1 wskazują na to, że między tymi regułami istnieje pewien związek. Widoczne są też reguły które osiągają wartość lift powyżej 3, co z kolei jest uważane za bardzo wysoką wartość.

Ten wykres przedstawia zależność między wsparciem (support), a wartością lift dla reguł asocjacyjnych, które łączą występowanie cukrzycy z ryzykiem zawału serca. Większość punktów na wykresie skupia się w lewym dolnym rogu, co może sugerować, że choć występowanie cukrzycy i ryzyko zawału serca mogą występować razem, nie są często spotykane zestawy. Jednakże, istnieją również punkty, które mają niskie wsparcie, ale osiągają wartość lift na poziomie 2.0 lub wyższą. Sugeruje to, że choć te reguły nie są często obserwowane, to kiedy cukrzyca występuje, ryzyko zawału serca jest wyższe.



## 5 Podsumowanie

Osiągnięte wyniki dokładności wahały się w granicach do 70%. Dzięki zastosowaniu StandardScaler udało mi się zwiększyć dokładność dla moich modeli w znacznej większości przypadków i niekiedy są to różnice kilku, bądź nawet kilkunastu procent jak w przypadku sieci neuronowej (ok. 14.25%), czy K-najbliższych sąsiadów (do 4.8%). Co więcej, wartości AUC-ROC, które są miarą jakości klasyfikacji, również poprawiły się po zastosowaniu standaryzacji. Najbardziej widoczny wzrost wartości AUC-ROC zaobserwowano dla sieci neuronowej, gdzie wartość wzrosła o 17.49%. To pokazuje, że standaryzacja danych może znacznie poprawić nie tylko dokładność, ale również ogólną jakość modeli klasyfikacyjnych.

Klasyfikator	Dokładność / AUC-ROC bez StandardScaler	Dokładność / AUC-ROC z StandardScaler
Lasy Losowe	68.59% / 68.51%	68.53% / 68.45%
Drzewo decyzyjne	57.57% / 57.58%	57.69% / 57.70%
Naiwny Bayes	57.16% / 57.32%	64.03% / 64.01%
KNN, n=3	59.47% / 59.54%	63.26% / 63.37%
KNN, n=5	58.25% / 58.33%	63.05% / 63.16%
KNN, n=11	56.68% / 56.74%	60.53% / 60.64%
Sieć neuronowa	49.57% / 50.00%	63.82% / 67.49%

## 6 Źródła

Baza danych: <https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset>  
Wykłady