

Heartdisease_prediction

December 30, 2024

Pobranie zbioru danych z platformy kaggle, przy pomocy wcześniej skonfigurowanego klucza API (katalog ~/.kaggle):

```
[25]: import os
import kaggle
import zipfile

dataset_path = 'Datasets/alexteboul/heart-disease-health-indicators-dataset'
file_name = 'heart_disease_health_indicators_BRFSS2015.csv'
zip_path = os.path.join(dataset_path, file_name + '.zip')

if not os.path.exists(dataset_path):
    kaggle.api.dataset_download_file('alexteboul/
↳heart-disease-health-indicators-dataset', file_name, path=dataset_path)
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(dataset_path)
else:
    print("Zbior danych został już pobrany")
```

Zbior danych został już pobrany

Utworzenie ramki danych na podstawie pliku: heart_disease_health_indicators_BRFSS2015.csv:

```
[26]: import pandas as pd
import warnings
import os

warnings.filterwarnings("ignore", category=FutureWarning)
df = pd.read_csv(os.getcwd() + "/Datasets/alexteboul/
↳heart-disease-health-indicators-dataset/
↳heart_disease_health_indicators_BRFSS2015.csv")
```

0.0.1 Cechy poszczególnych pacjentów w wczytanym zbiorze:

- HeartDiseaseorAttack: choroba wieńcowa (CHD) / zawał mięśnia sercowego (MI): 0 = nie // 1 = tak
- HighBP: wysokie ciśnienie krwi: 0 = brak wysokiego ciśnienia // 1 = wysokie ciśnienie krwi
- HighChol: wysoki cholesterol: 0 = brak wysokiego cholesterolu // 1 = wysoki cholesterol
- CholCheck: kontrola cholesterolu w ciągu ostatnich 5 lat: 0 = brak kontroli // 1 = kontrola

- BMI: Indeks Masy Ciała
- Smoker: Czy wypaliłeś przynajmniej 100 papierosów w swoim życiu: 0 = nie // 1 = tak
- Stroke: czy kiedykolwiek miałeś udar: 0 = nie // 1 = tak
- Diabetes: cukrzyca: 0 = brak cukrzycy // 2 = Stan przedcukrzycowy // 2 = cukrzyca
- PhysActivity: aktywność fizyczna w ciągu ostatnich 30 dni - nie licząc pracy: 0 = nie // 1 = tak
- Fruits: Spożywanie owoców 1 lub więcej razy dziennie: 0 = nie // 1 = tak
- Veggies: Spożywanie warzyw 1 lub więcej razy dziennie: 0 = nie // 1 = tak
- HvyAlcoholConsump: intensywne spożycie alkoholu (mężczyźni ≥ 14 drinków na tydzień, kobiety ≥ 7 drinków na tydzień): 0 = nie // 1 = tak
- AnyHealthcare: Czy miałeś dostęp do jakiegokolwiek formy opieki zdrowotnej: 0 = nie // 1 = tak
- NoDocbcCost: Czy zrezygnowałeś z wizyty u lekarza z powodu kosztów: 0 = nie // 1 = tak
- GenHlth: Jak oceniałbyś swoje ogólne zdrowie: skala 1-5: 1 = doskonale // 2 = bardzo dobre // 3 = dobre // 4 = dość dobre // 5 = słabe
- MentHlth: liczba dni ze złym zdrowiem psychicznym w ciągu ostatnich 30 dni: skala 1-30 dni
- PhysHlth: liczba dni ze złym zdrowiem fizycznym lub urazem w ciągu ostatnich 30 dni: skala 1-30 dni
- DiffWalk: Czy masz poważne trudności z chodzeniem lub wchodzeniem po schodach: 0 = nie // 1 = tak
- Sex: płeć pacjenta: 1 = męczyzna // 0 = kobieta
- Age: 13-poziomowa kategoria wiekowa: 1 = 18-24 // 9 = 60-64 // 13 = 80 lub starszy
- Education: poziom wykształcenia: 1 = bez wykształcenia // 2 = podstawowe // 3 = gimnazjalne // 4 = średnie // 5 = wyższe
- Income: dochód roczny osoby badanej: 1 = $< \$10,000$ // 2 = $\$10,000-\$14,999$ // 3 = $\$15,000-\$19,999$ // 4 = $\$20,000-\$24,999$ // 5 = $\$25,000-\$34,999$ // 6 = $\$35,000-\$49,999$ // 7 = $\$50,000-\$74,999$ // 8 = $\$75,000$

Umieszczenie kolumny HeartDiseaseorAttack na samym końcu ramki danych

```
[27]: df = df.reindex(columns=df.columns[1:].tolist() + [df.columns[0]])
```

Sprawdzenie i usunięcie ewentualnych duplikatów:

```
[28]: row, column = df.shape
df.drop_duplicates(inplace=True)
if df.shape == (row, column):
    print('Zbiór nie zawiera duplikatów')
else:
    print(f'Liczba duplikatów: {df.shape[0]}')
```

Liczba duplikatów: 229781

Sprawdzenie i usunięcie ewentualnych pustych rekordów:

```
[29]: rows_before = df.shape[0]

if df.isnull().values.any():
```

```

df = df.dropna()
rows_after = df.shape[0]
print(f"Usunięto {rows_before - rows_after} pustych rekordów.\n")
else:
    print("DataFrame nie zawiera pustych rekordów.\n")
df.info()

```

DataFrame nie zawiera pustych rekordów.

```

<class 'pandas.core.frame.DataFrame'>
Index: 229781 entries, 0 to 253679
Data columns (total 22 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   HighBP                                229781 non-null float64
1   HighChol                             229781 non-null float64
2   CholCheck                            229781 non-null float64
3   BMI                                   229781 non-null float64
4   Smoker                               229781 non-null float64
5   Stroke                               229781 non-null float64
6   Diabetes                             229781 non-null float64
7   PhysActivity                         229781 non-null float64
8   Fruits                               229781 non-null float64
9   Veggies                              229781 non-null float64
10  HvyAlcoholConsump                    229781 non-null float64
11  AnyHealthcare                        229781 non-null float64
12  NoDocbcCost                          229781 non-null float64
13  GenHlth                              229781 non-null float64
14  MentHlth                             229781 non-null float64
15  PhysHlth                             229781 non-null float64
16  DiffWalk                             229781 non-null float64
17  Sex                                   229781 non-null float64
18  Age                                   229781 non-null float64
19  Education                            229781 non-null float64
20  Income                               229781 non-null float64
21  HeartDiseaseorAttack                 229781 non-null float64
dtypes: float64(22)
memory usage: 40.3 MB

```

Preferowana paleta kolorów stosowana do tworzenia wykresów:

```

[30]: import matplotlib.pyplot as plt
import seaborn as sns
sns.set_palette('cubehelix')
sns.color_palette("cubehelix")

```

```

[30]: [(0.10231025194333628, 0.13952898866828906, 0.2560120319409181),
      (0.10594361078604106, 0.3809739011595331, 0.27015111282899046),

```

```
(0.4106130272672762, 0.48044780541672255, 0.1891154277778484),
(0.7829183382530567, 0.48158303462490826, 0.48672451968362596),
(0.8046168329276406, 0.6365733569301846, 0.8796578402926125),
(0.7775608374378459, 0.8840392521212448, 0.9452007992345052)]
```

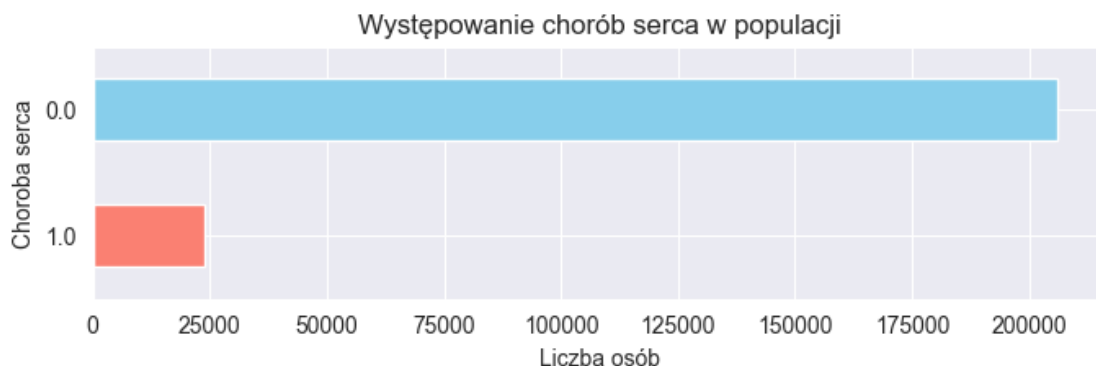
Sprawdzenie reprezentatywności danych:

```
[31]: minority_indices = df[df['HeartDiseaseorAttack'] == 1].index

majority_indices = df[df['HeartDiseaseorAttack'] == 0].index

plt.figure(figsize=(8, 2))
df['HeartDiseaseorAttack'].value_counts().plot(kind='barh', color=['skyblue', 'salmon'])
plt.title('Występowanie chorób serca w populacji')
plt.xlabel('Liczba osób')
plt.ylabel('Choroba serca')
plt.xticks(rotation=0)
plt.gca().invert_yaxis()
plt.savefig(os.getcwd() + '/Files/choroba_serca.png', bbox_inches='tight', dpi=300)
percentage_difference = ((len(majority_indices) - len(minority_indices)) / len(minority_indices)) * 100
print(f"Klasa większościowa jest większa o {percentage_difference:.2f}%.\n")
plt.show()
```

Klasa większościowa jest większa o 768.85%.



Powyższy wykres słupkowy ukazuje znaczną przewagę pacjentów nieposiadających choroby serca. W związku z tym model wyszkolony na tak zbalansowanych danych może być bardziej skłonny do błędnej klasyfikacji pacjentów. By rozwiązać ten problem, można zastosować jedno z 3 podejść: - Oversampling klasy mniejszościowej - Downsampling klasy większościowej - Wyodrębnienie z populacji kohorty pacjentów o określonym profilu ryzyka

Zważywszy, że wytrenowany model będzie pełnił rolę przesiewową (dla całej populacji) i klasa większościowa jest znacznie liczniejsza, zastosuję drugie podejście, które polega na losowym usuwaniu wartości z klasy większościowej.

```
[32]: from sklearn.utils import resample
import pandas as pd
import matplotlib.pyplot as plt

X = df.drop('HeartDiseaseorAttack', axis=1)
y = df['HeartDiseaseorAttack']

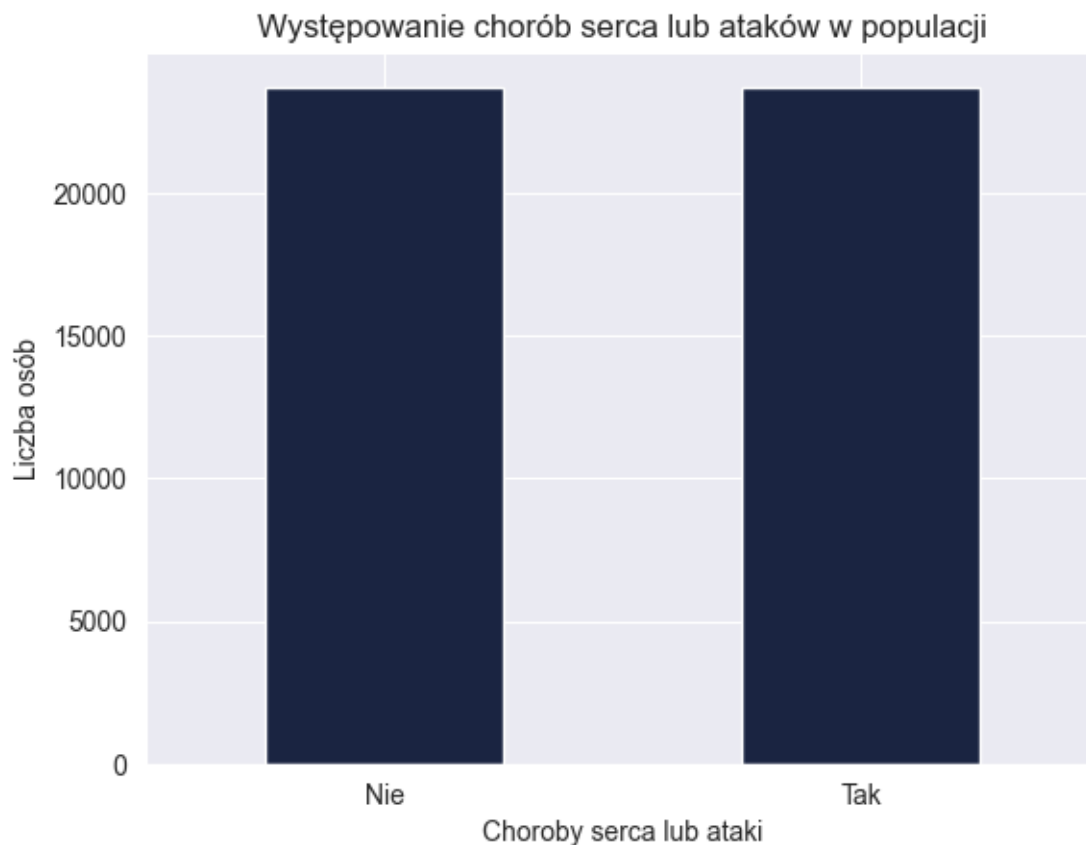
X_majority = X[y == 0]
X_minority = X[y == 1]
y_majority = y[y == 0]
y_minority = y[y == 1]

X_majority_downsampled, y_majority_downsampled = resample(X_majority,
    y_majority,
    replace=False,
    n_samples=len(y_minority),
    random_state=123)

X_downsampled = pd.concat([X_majority_downsampled, X_minority])
y_downsampled = pd.concat([y_majority_downsampled, y_minority])

df_downsampled = pd.concat([X_downsampled, y_downsampled], axis=1)
df = df_downsampled

df_downsampled['HeartDiseaseorAttack'].value_counts().plot(kind='bar')
plt.title('Występowanie chorób serca lub ataków w populacji')
plt.xlabel('Choroby serca lub ataki')
plt.ylabel('Liczba osób')
plt.xticks([0, 1], ['Nie', 'Tak'], rotation=0)
plt.show()
```



Wykresy rozrzutu:

```
[33]: import matplotlib.pyplot as plt
import seaborn as sns

# Przygotowanie figury
fig, axs = plt.subplots(11, 2, figsize=(10, 55))

color_dict = {0: 'red', 1: 'blue'}

# Wykresy dla BMI
sns.scatterplot(x='BMI', y='GenHlth', hue='HeartDiseaseorAttack', data=df,
               ax=axs[0, 0], alpha=0.25, palette=color_dict)
axs[0, 0].set_title('Wykres rozrzutu: BMI vs GenHlth')
axs[0, 0].legend(loc='upper right')

sns.scatterplot(x='BMI', y='MentHlth', hue='HeartDiseaseorAttack', data=df,
               ax=axs[0, 1], alpha=0.25, palette=color_dict)
axs[0, 1].set_title('Wykres rozrzutu: BMI vs MentHlth')
axs[0, 1].legend(loc='upper right')
```

```

# Wykresy dla BMI i innych zmiennych
sns.scatterplot(x='BMI', y='PhysHlth', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[1, 0], alpha=0.25, palette=color_dict)
axes[1, 0].set_title('Wykres rozrzutu: BMI vs PhysHlth')
axes[1, 0].legend(loc='upper right')

sns.scatterplot(x='BMI', y='Age', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[1, 1], alpha=0.25, palette=color_dict)
axes[1, 1].set_title('Wykres rozrzutu: BMI vs Age')
axes[1, 1].legend(loc='upper right')

sns.scatterplot(x='BMI', y='Education', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[2, 0], alpha=0.25, palette=color_dict)
axes[2, 0].set_title('Wykres rozrzutu: BMI vs Education')
axes[2, 0].legend(loc='upper right')

sns.scatterplot(x='BMI', y='Income', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[2, 1], alpha=0.25, palette=color_dict)
axes[2, 1].set_title('Wykres rozrzutu: BMI vs Income')
axes[2, 1].legend(loc='upper right')

# Wykresy dla GenHlth
sns.scatterplot(x='GenHlth', y='MentHlth', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[3, 0], alpha=0.25, palette=color_dict)
axes[3, 0].set_title('Wykres rozrzutu: GenHlth vs MentHlth')
axes[3, 0].legend(loc='upper right')

sns.scatterplot(x='GenHlth', y='PhysHlth', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[3, 1], alpha=0.25, palette=color_dict)
axes[3, 1].set_title('Wykres rozrzutu: GenHlth vs PhysHlth')
axes[3, 1].legend(loc='upper right')

sns.scatterplot(x='GenHlth', y='Age', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[4, 0], alpha=0.25, palette=color_dict)
axes[4, 0].set_title('Wykres rozrzutu: GenHlth vs Age')
axes[4, 0].legend(loc='upper right')

sns.scatterplot(x='GenHlth', y='Education', hue='HeartDiseaseorAttack',
    ↪data=df, ax=axes[4, 1], alpha=0.25, palette=color_dict)
axes[4, 1].set_title('Wykres rozrzutu: GenHlth vs Education')
axes[4, 1].legend(loc='upper right')

sns.scatterplot(x='GenHlth', y='Income', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[5, 0], alpha=0.25, palette=color_dict)
axes[5, 0].set_title('Wykres rozrzutu: GenHlth vs Income')

```

```

axs[5, 0].legend(loc='upper right')

# Wykresy dla MentHlth
sns.scatterplot(x='MentHlth', y='PhysHlth', hue='HeartDiseaseorAttack',
               ↪data=df, ax=axs[5, 1], alpha=0.25, palette=color_dict)
axs[5, 1].set_title('Wykres rozrzutu: MentHlth vs PhysHlth')
axs[5, 1].legend(loc='upper right')

sns.scatterplot(x='MentHlth', y='Age', hue='HeartDiseaseorAttack', data=df,
               ↪ax=axs[6, 0], alpha=0.25, palette=color_dict)
axs[6, 0].set_title('Wykres rozrzutu: MentHlth vs Age')
axs[6, 0].legend(loc='upper right')

sns.scatterplot(x='MentHlth', y='Education', hue='HeartDiseaseorAttack',
               ↪data=df, ax=axs[6, 1], alpha=0.25, palette=color_dict)
axs[6, 1].set_title('Wykres rozrzutu: MentHlth vs Education')
axs[6, 1].legend(loc='upper right')

sns.scatterplot(x='MentHlth', y='Income', hue='HeartDiseaseorAttack', data=df,
               ↪ax=axs[7, 0], alpha=0.25, palette=color_dict)
axs[7, 0].set_title('Wykres rozrzutu: MentHlth vs Income')
axs[7, 0].legend(loc='upper right')

# Wykresy dla PhysHlth
sns.scatterplot(x='PhysHlth', y='Age', hue='HeartDiseaseorAttack', data=df,
               ↪ax=axs[7, 1], alpha=0.25, palette=color_dict)
axs[7, 1].set_title('Wykres rozrzutu: PhysHlth vs Age')
axs[7, 1].legend(loc='upper right')

sns.scatterplot(x='PhysHlth', y='Education', hue='HeartDiseaseorAttack',
               ↪data=df, ax=axs[8, 0], alpha=0.25, palette=color_dict)
axs[8, 0].set_title('Wykres rozrzutu: PhysHlth vs Education')
axs[8, 0].legend(loc='upper right')

sns.scatterplot(x='PhysHlth', y='Income', hue='HeartDiseaseorAttack', data=df,
               ↪ax=axs[8, 1], alpha=0.25, palette=color_dict)
axs[8, 1].set_title('Wykres rozrzutu: PhysHlth vs Income')
axs[8, 1].legend(loc='upper right')

# Wykresy dla Age
sns.scatterplot(x='Age', y='Education', hue='HeartDiseaseorAttack', data=df,
               ↪ax=axs[9, 0], alpha=0.25, palette=color_dict)
axs[9, 0].set_title('Wykres rozrzutu: Age vs Education')
axs[9, 0].legend(loc='upper right')

```



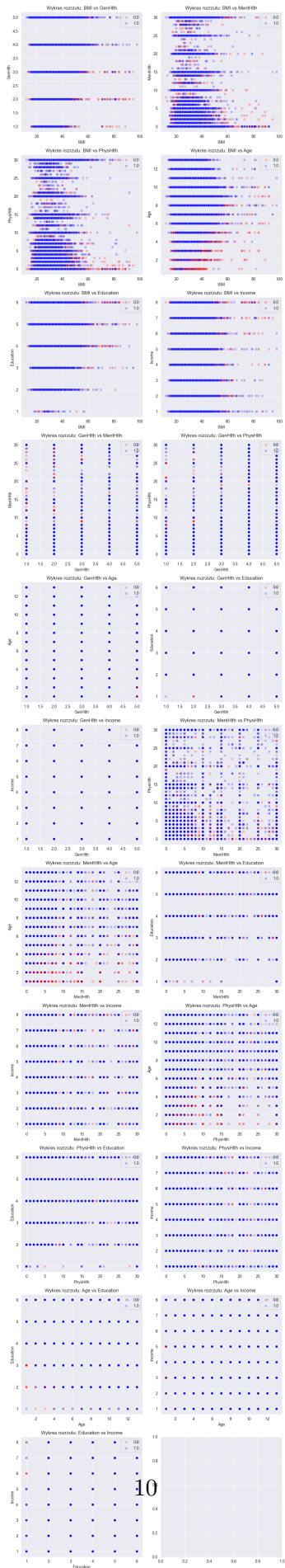
```

sns.scatterplot(x='Age', y='Income', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[9, 1], alpha=0.25, palette=color_dict)
axes[9, 1].set_title('Wykres rozrzutu: Age vs Income')
axes[9, 1].legend(loc='upper right')

# Wykresy dla Education
sns.scatterplot(x='Education', y='Income', hue='HeartDiseaseorAttack', data=df,
    ↪ax=axes[10, 0], alpha=0.25, palette=color_dict)
axes[10, 0].set_title('Wykres rozrzutu: Education vs Income')
axes[10, 0].legend(loc='upper right')

plt.tight_layout()
plt.show()

```



Rozdzielenie zmiennych na katagoryczne i ilościowe:

```
[10]: zmienne_iloosciowe = ['Age', 'BMI', 'GenHlth', 'MentHlth',
    ↪ 'PhysHlth', 'Education', 'Income']
zmienne_jakosciowe = ['HeartDiseaseorAttack', 'HighBP', 'HighChol', 'CholCheck',
    'Smoker', 'Stroke', 'Diabetes', 'PhysActivity',
    'Fruits', 'Veggies', 'HvyAlcoholConsump',
    'AnyHealthcare', 'NoDocbcCost', 'Sex']

df_iloosciowe = df[zmienne_iloosciowe]
df_jakosciowe = df[zmienne_jakosciowe]

[11]: print(f"Zmienne ilościowe:\n {df_iloosciowe}")
print(f"Zmienne jakościowe:\n {df_jakosciowe}")
```

Zmienne ilościowe:

	Age	BMI	GenHlth	MentHlth	PhysHlth	Education	Income
80705	3.0	31.0	1.0	0.0	0.0	6.0	8.0
163785	3.0	25.0	2.0	0.0	0.0	6.0	4.0
190627	10.0	25.0	3.0	30.0	0.0	4.0	2.0
166270	7.0	31.0	3.0	5.0	7.0	6.0	8.0
214464	9.0	33.0	1.0	0.0	0.0	5.0	7.0
...
253668	10.0	29.0	2.0	0.0	0.0	3.0	6.0
253670	13.0	25.0	5.0	15.0	0.0	6.0	4.0
253671	8.0	23.0	4.0	0.0	5.0	3.0	2.0
253672	12.0	30.0	3.0	0.0	0.0	2.0	1.0
253679	9.0	25.0	2.0	0.0	0.0	6.0	2.0

[47434 rows x 7 columns]

Zmienne jakościowe:

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	Smoker	Stroke	\
80705	0.0	0.0	0.0	1.0	0.0	0.0	
163785	0.0	0.0	0.0	1.0	0.0	0.0	
190627	0.0	0.0	1.0	1.0	1.0	0.0	
166270	0.0	1.0	1.0	1.0	0.0	0.0	
214464	0.0	1.0	0.0	1.0	0.0	0.0	
...	
253668	1.0	0.0	1.0	1.0	1.0	0.0	
253670	1.0	1.0	1.0	1.0	0.0	0.0	
253671	1.0	1.0	1.0	1.0	0.0	1.0	
253672	1.0	1.0	0.0	1.0	1.0	0.0	
253679	1.0	1.0	1.0	1.0	0.0	0.0	

	Diabetes	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	\
80705	0.0	1.0	0.0	1.0	0.0	

163785	0.0	1.0	1.0	1.0	0.0
190627	0.0	0.0	0.0	1.0	0.0
166270	0.0	0.0	0.0	1.0	0.0
214464	0.0	1.0	0.0	1.0	0.0
...
253668	2.0	0.0	1.0	1.0	0.0
253670	2.0	0.0	1.0	0.0	0.0
253671	0.0	0.0	0.0	0.0	0.0
253672	0.0	1.0	1.0	1.0	0.0
253679	2.0	1.0	1.0	0.0	0.0

	AnyHealthcare	NoDocbcCost	Sex
80705	1.0	0.0	1.0
163785	1.0	0.0	1.0
190627	1.0	1.0	0.0
166270	1.0	0.0	1.0
214464	1.0	0.0	1.0
...
253668	1.0	0.0	1.0
253670	1.0	0.0	0.0
253671	1.0	1.0	1.0
253672	1.0	0.0	1.0
253679	1.0	0.0	0.0

[47434 rows x 14 columns]

```
[12]: print(f"Próbka:\n {df.sample()}")
```

Próbka:

	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	\
191606	0.0	0.0	1.0	30.0	0.0	0.0	0.0	

	PhysActivity	Fruits	Veggies	...	NoDocbcCost	GenHlth	MentHlth	\
191606	0.0	1.0	0.0	...	0.0	1.0	0.0	

	PhysHlth	DiffWalk	Sex	Age	Education	Income	HeartDiseaseorAttack
191606	0.0	0.0	1.0	13.0	5.0	5.0	1.0

[1 rows x 22 columns]

Zakodowanie zmiennych jakościowych za pomocą Label Encoding na format liczbowy (jeśli to konieczne):

```
[13]: from sklearn.preprocessing import LabelEncoder
import pandas as pd

df_copy = df.copy()
label_encoder = LabelEncoder()
```

```

label_mappings = {}

for column in df_copy.columns:
    if df_copy[column].dtype == 'object':
        df_copy[column] = label_encoder.fit_transform(df_copy[column])
        label_mappings[column] = dict(zip(label_encoder.classes_, label_encoder.
↳ transform(label_encoder.classes_)))
print(label_mappings)

{}

```

Statystyki opisowe dla zmiennych ilościowych w wyodrębnionej kohorcie:

```

[14]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import table

df_ilosciowe = df[zmiennie_ilosciowe]

description = df_ilosciowe.describe(include='all')

description.loc['mean'] = description.loc['mean'].round(2)
description.loc['std'] = description.loc['std'].round(2)

sns.set(style="darkgrid")
sns.set_palette('cubehelix')

fig, ax = plt.subplots(figsize=(14, 4))
ax.axis('off')

tbl = table(ax, description, loc='center', cellLoc='center', colWidths=[0.
↳ 15]*len(description.columns))

tbl.auto_set_font_size(False)
tbl.set_fontsize(10)
tbl.scale(1.2, 2.2)

for key, cell in tbl.get_celld().items():
    cell.set_edgecolor('black')
    if key[0] == 0:
        cell.set_text_props(weight='bold', color='white')
        cell.set_facecolor('#4c72b0')
    else:
        cell.set_facecolor('#f2f2f2')

```

```
plt.savefig(os.getcwd() + "/Files/opis_tabeli.png", bbox_inches='tight',
            pad_inches=0, dpi=300)
plt.show()
plt.close()
```

	Age	BMI	GenHlth	MentHlth	PhysHlth	Education	Income
count	47434.0	47434.0	47434.0	47434.0	47434.0	47434.0	47434.0
mean	8.99	29.03	2.94	4.07	6.69	4.88	5.56
std	2.91	6.78	1.14	8.48	10.66	1.03	2.17
min	1.0	12.0	1.0	0.0	0.0	1.0	1.0
25%	7.0	25.0	2.0	0.0	0.0	4.0	4.0
50%	9.0	28.0	3.0	0.0	0.0	5.0	6.0
75%	11.0	32.0	4.0	3.0	10.0	6.0	8.0
max	13.0	98.0	5.0	30.0	30.0	6.0	8.0

Age: Średnia wartość kategorii wiekowej wynosi ok. 9, co odpowiada przedziałowi wiekowemu 50-59 lat, z odchyleniem standardowym bliskim 2.9. Zakres wieku pacjentów waha się od 18 do 80 lat.

BMI: Średnia wartość wskaźnika masy ciała wynosi ok. 29.0 z odchyleniem standardowym bliskim 6.8. Zakres tego parametru waha się od 12.0 do 98.0.

GenHlth: Średnia ocena ogólnego stanu zdrowia wynosi ok. 2.9 na skali 1-5, gdzie 1 oznacza doskonale zdrowie, a 5 oznacza słabe zdrowie. Odchylenie standardowe tej oceny wynosi blisko 1.1. Zakres ocen waha się od 1 do 5.

MentHlth: Średnia liczba dni złego zdrowia psychicznego w ciągu ostatnich 30 dni wynosi ok. 4.1, z odchyleniem standardowym bliskim 8.5. Zakres tego parametru waha się od 0 do 30 dni.

PhysHlth: Średnia liczba dni złego zdrowia fizycznego lub urazu w ciągu ostatnich 30 dni wynosi ok. 6.7, z odchyleniem standardowym bliskim 10.7. Zakres tego parametru waha się od 0 do 30 dni.

Education: Średni poziom wykształcenia wynosi ok. 4.9, co odpowiada średniemu poziomowi edukacji w kategorii “wykształcenie średnie” (przy założeniu skali od 1 do 6). Odchylenie standardowe tej zmiennej wynosi blisko 1.0. Zakres poziomów wykształcenia waha się od 1 do 6.

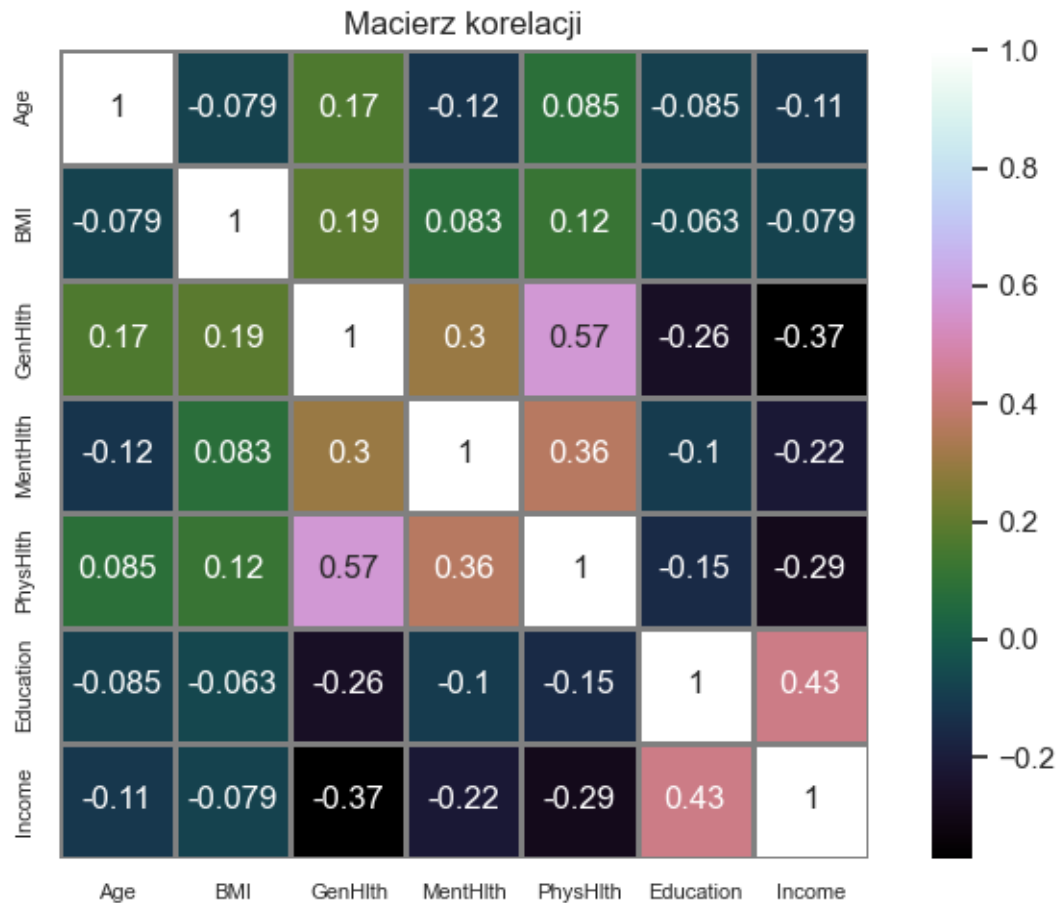
Income: Średni roczny dochód wynosi ok. 5.6 na skali dochodowej, co odpowiada przedziałowi dochodowemu od \$25,000 do \$34,999. Odchylenie standardowe tej zmiennej wynosi blisko 2.2. Zakres dochodów waha się od poniżej \$10,000 do \$75,000 i więcej.

Macierz Korelacji dla zmiennych ilościowych:

```
[15]: plt.figure(figsize=(7, 5))
heatmap = sns.heatmap(df_ilosciowe.corr(), annot=True, cmap="cubehelix",
                        square=True,
                        linewidths=1.0, linecolor='gray',
                        xticklabels=df_ilosciowe.columns,
                        yticklabels=df_ilosciowe.columns)
```

```
plt.xticks(rotation=0, ha='center',fontsize=8)
plt.yticks(rotation=90, va='center',fontsize=8)

plt.title('Macierz korelacji')
plt.tight_layout()
plt.show()
```

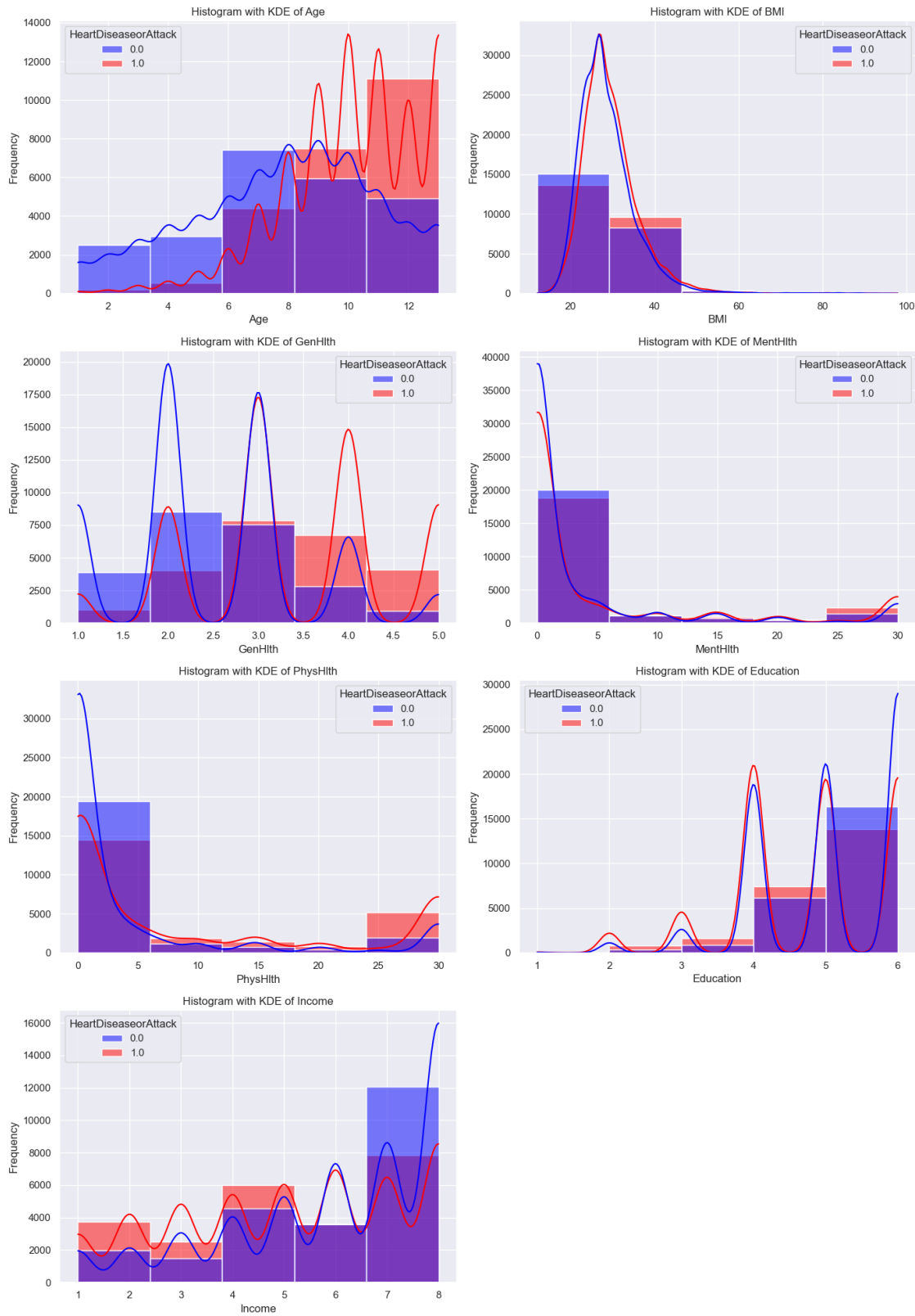


- GenHlth (Ogólny Stan Zdrowia) ma silną, dodatnią korelację z PhysHlth i umiarkowaną, dodatnią z MentHlth (Zdrowie Psychiczne), co sugeruje, że ogólny stan zdrowia pacjentów może wpływać na ich zdrowie fizyczne i psychiczne.
- Education (Wykształcenie) ma umiarkowaną, dodatnią korelację z Income (Dochód), co jest zgodne z oczekiwaniami, że wyższy poziom wykształcenia często prowadzi do wyższych dochodów.
- Income (Dochód) ma umiarkowaną ujemną korelację z GenHlth i umiarkowaną z PhysHlth, co może wskazywać, że osoby o niższych dochodach mogą postrzegać swoje zdrowie jako gorsze i doświadczać większej liczby dni złego zdrowia fizycznego.

Histogramy dla zmiennych ilościowych skategoryzowane za pomocą zmiennej “HeartDiseaseorAt-

tack”:

```
[16]: num_cols = ['Age', 'BMI', 'GenHlth', 'MentHlth',  
    ↪ 'PhysHlth', 'Education', 'Income']  
fig, axs = plt.subplots(4, 2, figsize=(14, 20))  
  
axs = axs.ravel()  
  
for i, col in enumerate(num_cols):  
    sns.histplot(df, x=col, hue='HeartDiseaseorAttack', kde=True, ax=axs[i],  
    ↪ bins=5, palette=['blue', 'red'])  
    axs[i].set_title(f'Histogram with KDE of {col}')  
    axs[i].set_xlabel(col)  
    axs[i].set_ylabel('Frequency')  
  
fig.delaxes(axs[-1])  
  
plt.tight_layout()  
plt.show()
```

Wnioski: - Wraz z wiekiem rośnie odsetek osób chorych w przebadanej grupie - Osoby chore częściej deklarują gorszy ogólny stan zdrowia w porównaniu do osób zdrowych. - Czym wyższy dochód i edukacja tym statystycznie niższe ryzyko wystąpienia choroby serca.

1 ANOVA

“Analiza wariancji lub ANOVA jest metodą modelowania liniowego do oceny relacji między zmiennymi. W przypadku kluczowych czynników oraz spostrzeżeń związanych z wieloma wykresami test ANOVA sprawdza, czy średnia wartość przewidywana różni się w poszczególnych kategoriach jednej zmiennej wejściowej lub kombinacjach kategorii dwóch zmiennych wejściowych”. <https://www.ibm.com/docs/pl/cognos-analytics/11.2.0?topic=tests-analysis-variance-anova>

Założenia ANOVY:

- 1) Normalność rozkładu: Zmienna zależna musi mieć rozkład normalny w każdej kombinacji poziomów czynników.
- 2) Homogeniczność wariancji: Wariancja zmiennej zależnej musi być taka sama we wszystkich grupach.
- 3) Niezależność obserwacji: Obserwacje w każdej grupie muszą być niezależne od siebie.

```
[17]: import scipy.stats as stats

zmiennie_ilościowe = ['Age', 'BMI', 'GenHlth', 'MentHlth', 'PhysHlth', 'Education', 'Income']

#-----

# 1)
for zmienna in zmiennie_ilościowe:
    print(f"\n1) Zmienna: {zmienna}")

    # Test Andersona-Darlinga na normalność
    result = stats.anderson(df[zmienna])
    if result.statistic < result.critical_values[2]: # dla poziomu istotności
    ↪ 0.05
        print(f"Dane mają rozkład normalny (statystyka={result.statistic})")
    else:
        print(f"Dane nie mają rozkładu normalnego (statystyka={result.
    ↪ statistic})")

#-----

# 2)
zmiennie_ilościowe = ['Age', 'BMI', 'GenHlth', 'MentHlth', 'PhysHlth', 'Education', 'Income']
```

```

zmienne_jakościowe = ['HeartDiseaseorAttack', 'HighBP', 'HighChol', 'CholCheck',
                      'Smoker', 'Stroke', 'Diabetes', 'PhysActivity',
                      'Fruits', 'Veggies', 'HvyAlcoholConsump',
                      'AnyHealthcare', 'NoDocbcCost', 'Sex']

for zmienna_ilosciowa in zmienne_ilościowe:
    print(f"\n2) Zmienna ilościowa: {zmienna_ilosciowa}")
    for zmienna_jakosciowa in zmienne_jakościowe:
        # Test Levene'a na homogeniczność wariancji
        W, p = stats.levene(*[group[zmienna_ilosciowa].values for name, group
↪in df.groupby(zmienna_jakosciowa)])
        if p > 0.05:
            print(f"Grupy zdefiniowane przez {zmienna_jakosciowa} mają równą
↪wariancję (W={W}, p={p})")
        else:
            print(f"Grupy zdefiniowane przez {zmienna_jakosciowa} nie mają
↪równej wariancji (W={W}, p={p})")

#-----

# 3)
print("\n3) The underlying uncleaned data comes from the CDC's BRFSS 2015")

```

1) Zmienna: Age

Dane nie mają rozkładu normalnego (statystyka=704.8613411246552)

1) Zmienna: BMI

Dane nie mają rozkładu normalnego (statystyka=813.707267210084)

1) Zmienna: GenHlth

Dane nie mają rozkładu normalnego (statystyka=1560.054445163929)

1) Zmienna: MentHlth

Dane nie mają rozkładu normalnego (statystyka=9608.577690415717)

1) Zmienna: PhysHlth

Dane nie mają rozkładu normalnego (statystyka=7403.10788661526)

1) Zmienna: Education

Dane nie mają rozkładu normalnego (statystyka=2594.38074333001)

1) Zmienna: Income

Dane nie mają rozkładu normalnego (statystyka=1566.459255260932)

2) Zmienna ilościowa: Age

Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji

(W=2386.975180575602, p=0.0)
 Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=2501.7430449096414, p=0.0)
 Grupy zdefiniowane przez HighChol nie mają równej wariancji
 (W=2765.472494821281, p=0.0)
 Grupy zdefiniowane przez CholCheck nie mają równej wariancji
 (W=6.566462690337576, p=0.010394887658192927)
 Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=561.3478102052215, p=2.236860505395304e-123)
 Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=354.9761686040973, p=6.794427382714653e-79)
 Grupy zdefiniowane przez Diabetes nie mają równej wariancji
 (W=812.4374072907232, p=0.0)
 Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
 (W=119.58261354062853, p=8.428215050325313e-28)
 Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=46.49925527672851, p=9.274807604284883e-12)
 Grupy zdefiniowane przez Veggies nie mają równej wariancji
 (W=10.699358353871526, p=0.0010724915280727737)
 Grupy zdefiniowane przez HvyAlcoholConsump mają równą wariancję
 (W=3.7159267967817753, p=0.053901659827446435)
 Grupy zdefiniowane przez AnyHealthcare nie mają równej wariancji
 (W=14.137809732322662, p=0.00017009823800331759)
 Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji
 (W=19.211666579656747, p=1.1724724841263816e-05)
 Grupy zdefiniowane przez Sex nie mają równej wariancji (W=48.964311513637305, p=2.6410986687193765e-12)

2) Zmienna ilościowa: BMI

Grupy zdefiniowane przez HeartDiseaseorAttack mają równą wariancję
 (W=3.628633945180924, p=0.056799456834397036)
 Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=216.29749164419437, p=7.443524770292038e-49)
 Grupy zdefiniowane przez HighChol mają równą wariancję (W=2.5447571607902395, p=0.1106675863922704)
 Grupy zdefiniowane przez CholCheck mają równą wariancję (W=2.0342170967519593, p=0.15379993732568314)
 Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=13.292709519308543, p=0.00026672486143834647)
 Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=6.070284182662928, p=0.01375100551342831)
 Grupy zdefiniowane przez Diabetes nie mają równej wariancji
 (W=153.8945300479007, p=2.4009089251578498e-67)
 Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
 (W=404.2787265149974, p=1.5253639960771842e-89)
 Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=26.635964048483043, p=2.4661681342772273e-07)
 Grupy zdefiniowane przez Veggies nie mają równej wariancji (W=19.40800343739751,

p=1.0579471583862943e-05)

Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
(W=48.55315994745958, p=3.2563732605662675e-12)

Grupy zdefiniowane przez AnyHealthcare nie mają równej wariancji
(W=18.720444092020355, p=1.5166017738914765e-05)

Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji
(W=116.02143564327115, p=5.0531406087855744e-27)

Grupy zdefiniowane przez Sex nie mają równej wariancji (W=483.1228434478056,
p=1.5244079977753383e-106)

2) Zmienna ilościowa: GenHlth

Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=44.0832000506761, p=3.1809477980021385e-11)

Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=30.24572862468118,
p=3.8258780988114114e-08)

Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=81.36148986148318, p=1.9481845043420372e-19)

Grupy zdefiniowane przez CholCheck nie mają równej wariancji
(W=4.755496592041138, p=0.02920976081167241)

Grupy zdefiniowane przez Smoker mają równą wariancję (W=0.07165327268246169,
p=0.7889458172235391)

Grupy zdefiniowane przez Stroke mają równą wariancję (W=1.5460396218516572,
p=0.21372719915874927)

Grupy zdefiniowane przez Diabetes nie mają równej wariancji
(W=4.693415954672478, p=0.0091596107963183)

Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=97.30175286032065, p=6.26330392182605e-23)

Grupy zdefiniowane przez Fruits mają równą wariancję (W=0.07750421117573743,
p=0.780709770968177)

Grupy zdefiniowane przez Veggies nie mają równej wariancji (W=7.063482005929451,
p=0.007869730987125595)

Grupy zdefiniowane przez HvyAlcoholConsump mają równą wariancję
(W=0.3278850840363168, p=0.5669098030933502)

Grupy zdefiniowane przez AnyHealthcare mają równą wariancję
(W=0.7342938180478841, p=0.39149925497034954)

Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji
(W=52.616582002363415, p=4.1163297330161436e-13)

Grupy zdefiniowane przez Sex nie mają równej wariancji (W=56.12120000625693,
p=6.931874730533405e-14)

2) Zmienna ilościowa: MentHlth

Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=271.0349196459649, p=9.956446000655866e-61)

Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=143.1395386918025,
p=6.111945468972383e-33)

Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=156.2743814552585, p=8.398489968456203e-36)

Grupy zdefiniowane przez CholCheck nie mają równej wariancji

(W=8.472071277374873, p=0.003608082791491972)
 Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=276.028968520936, p=8.240714037513764e-62)
 Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=413.6706582514044, p=1.4336910620394786e-91)
 Grupy zdefiniowane przez Diabetes nie mają równej wariancji (W=151.94805859466632, p=1.6607395303303234e-66)
 Grupy zdefiniowane przez PhysActivity nie mają równej wariancji (W=811.3838062362921, p=5.632799885566873e-177)
 Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=187.47030033086048, p=1.366533700002506e-42)
 Grupy zdefiniowane przez Veggies nie mają równej wariancji (W=149.75921832745505, p=2.2054733980949442e-34)
 Grupy zdefiniowane przez HvyAlcoholConsump mają równą wariancję (W=2.445708428365205, p=0.11785334161549085)
 Grupy zdefiniowane przez AnyHealthcare nie mają równej wariancji (W=137.823607912729, p=8.814366045225945e-32)
 Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji (W=1940.9910661913775, p=0.0)
 Grupy zdefiniowane przez Sex nie mają równej wariancji (W=459.315667919633, p=2.054610373922047e-101)

2) Zmienna ilościowa: PhysHlth

Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji (W=3037.231134218436, p=0.0)
 Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=1442.3603274823365, p=6.1911184870793e-311)
 Grupy zdefiniowane przez HighChol nie mają równej wariancji (W=762.9855160093418, p=1.2641942507705063e-166)
 Grupy zdefiniowane przez CholCheck nie mają równej wariancji (W=76.9221195881756, p=1.8360928703088992e-18)
 Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=693.6956753969564, p=8.723144001461508e-152)
 Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=1138.5789057151046, p=1.1482349335762604e-246)
 Grupy zdefiniowane przez Diabetes nie mają równej wariancji (W=783.1181759084593, p=0.0)
 Grupy zdefiniowane przez PhysActivity nie mają równej wariancji (W=2796.813926894697, p=0.0)
 Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=69.85019834659309, p=6.569676461870514e-17)
 Grupy zdefiniowane przez Veggies nie mają równej wariancji (W=161.53498007160064, p=6.006386892524221e-37)
 Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji (W=109.68797883227829, p=1.2233286488026282e-25)
 Grupy zdefiniowane przez AnyHealthcare mają równą wariancję (W=0.059693314995258845, p=0.806982383577273)
 Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji

(W=747.0333423463989, p=3.281058237926139e-163)
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=145.24562691253556, p=2.1238053878871626e-33)

2) Zmienna ilościowa: Education

Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=136.7369444135937, p=1.5211282236463174e-31)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=67.1464625438738, p=2.583046335671811e-16)
Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=34.105980524202444, p=5.253007520876747e-09)
Grupy zdefiniowane przez CholCheck mają równą wariancję (W=2.8614629534329743, p=0.09073152027532701)
Grupy zdefiniowane przez Smoker mają równą wariancję (W=0.711274399954293, p=0.3990253193644573)
Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=88.1790321709107, p=6.2346029276039046e-21)
Grupy zdefiniowane przez Diabetes nie mają równej wariancji
(W=40.64870102866763, p=2.299351468306857e-18)
Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=159.44678446173072, p=1.7112435886175488e-36)
Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=31.655544376197945, p=1.8512367466521668e-08)
Grupy zdefiniowane przez Veggies nie mają równej wariancji
(W=130.53266538361896, p=3.431881291102095e-30)
Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
(W=6.106390736391594, p=0.013472877932755527)
Grupy zdefiniowane przez AnyHealthcare nie mają równej wariancji
(W=40.975534964933594, p=1.555826432563899e-10)
Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji
(W=27.44884887917655, p=1.6199264112746663e-07)
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=29.23977932431823, p=6.426092637632629e-08)

2) Zmienna ilościowa: Income

Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=272.8998105076566, p=3.926335440705325e-61)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=225.07710294515337, p=9.238811697318645e-51)
Grupy zdefiniowane przez HighChol nie mają równej wariancji (W=68.0250296798295, p=1.6553292231426222e-16)
Grupy zdefiniowane przez CholCheck mają równą wariancję (W=1.6252736739168334, p=0.2023642589685058)
Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=31.566280156375402, p=1.9382577962393272e-08)
Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=62.739427955564224, p=2.4104463856370724e-15)
Grupy zdefiniowane przez Diabetes nie mają równej wariancji

(W=90.36296394227242, p=6.76767990699267e-40)
 Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
 (W=166.4550870815889, p=5.099072479382306e-38)
 Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=146.4598892522658,
 p=1.1547049100169292e-33)
 Grupy zdefiniowane przez Veggies nie mają równej wariancji
 (W=122.40134033160435, p=2.0429060165994533e-28)
 Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
 (W=89.38143231611996, p=3.3988032492564486e-21)
 Grupy zdefiniowane przez AnyHealthcare mają równą wariancję
 (W=3.456371235710565, p=0.06301463985519683)
 Grupy zdefiniowane przez NoDocbcCost nie mają równej wariancji
 (W=22.812999666591317, p=1.7908651866767126e-06)
 Grupy zdefiniowane przez Sex nie mają równej wariancji (W=624.3992005291503,
 p=6.374872826281646e-137)

3) The underlying uncleaned data comes from the CDC's BRFSS 2015

1.0.1 Wnioski:

- 1) Przeprowadzono test Andersona-Darlinga. Jego wybór uwarunkowany był wcześniejszym ostrzeżeniem użytkownika wskazującym na niedopasowanie testu Shapiro-Wilka do liczebności badanej kohorty. Wykazał on, że wszystkie badane zmienne nie mają rozkładu normalnego. W związku z tym analiza wieloczynnikowa ANOVA nie jest możliwa. W zamian przeprowadzony zostanie nieparametryczny test Kruskala-Wallisa, który będzie odpowiedni do badanego rozkładu zmiennych.

```
[18]: ## ANOVA wieloczynnikowa -> nie można przeprowadzić bo naruszono 1) & 2) :(
# import statsmodels.api as sm
# from statsmodels.formula.api import ols
# from statsmodels.stats.multicomp import pairwise_tukeyhsd
# for zmienna in zmienne_ilościowe:
#     formula = '{f} ~ '.format(zmienna) + ' + '.join(['C({f})'.format(var) for
# ↪var in df_jakosciowe.columns])
#
#     model = ols(formula, data=df).fit()
#
#     anova_table = sm.stats.anova_lm(model, typ=2)
#     print(f'\033[94mANOVA: {zmienna}\n', anova_table, '\n\033[0m')
#
#     for var in df_jakosciowe.columns:
#         if anova_table.loc[f'C({var})', 'PR(>F)'] < 0.05:
#             tukey = pairwise_tukeyhsd(endog=df[zmienna], groups=df[var],
# ↪alpha=0.05)
#             if var == 'diabetes':
#                 print(f'\033[91mTest post-hoc Tukeya: {zmienna} & {var}\n',
# ↪tukey, '\033[0m\n')
#             else:
```



```
# print(f'Test post-hoc Tukeya: {zmienna} vs {var}\n', tukey,
# ↪ '\n')
#
# print('\033[92m - - - - - \n')
# ↪ - - - - - \033[0m')
```

- 2) Do zbadania homogeniczności wariancji poszczególnych grup użyty został test Levene'a. Nie wymaga on rozkładu normalnego, więc był to odpowiedni wybór dla analizowanych danych. Jego wyniki wskazują, że poszczególne grupy w większości nie są z sobą porównywalne. Konieczne stało się więc wzięcie pod uwagę tego czynnika poprzez korektę uwzględniającą heteroskedastyczność - test post-hoc Conovera.
- 3) Dane pochodzą z 2015 roku i obejmują informacje zebrane zarówno przez telefony stacjonarne, jak i komórkowe z 50 stanów, Dystryktu Kolumbii, Guamu i Portoryko. Z racji ich liczebności i zastosowanej metodyki z dużym prawdopodobieństwem obserwacje te są od siebie niezależne.

```
[19]: from scipy.stats import kruskal
from scikit_posthocs import posthoc_conover

zmienne_ilościowe = ['Age', 'BMI', 'GenHlth', 'MentHlth',
    ↪ 'PhysHlth', 'Education', 'Income']
zmienne_jakościowe = ['HeartDiseaseorAttack', 'HighBP', 'HighChol',
    ↪ 'CholCheck', 'Smoker', 'Stroke', 'Diabetes', 'PhysActivity',
    'Fruits', 'Veggies', 'HvyAlcoholConsump',
    'AnyHealthcare', 'NoDocbcCost', 'Sex']

for zmienna_ilosciowa in zmienne_ilościowe:
    print(f"\n\033[94mZmienna ilościowa: {zmienna_ilosciowa}\033[0m")
    for zmienna_jakosciowa in zmienne_jakościowe:
        H, p = kruskal(*[group[zmienna_ilosciowa].values for name, group in df.
    ↪ groupby(zmienna_jakosciowa)])
        print(f"\033[91mTest Kruskala-Wallisa dla {zmienna_jakosciowa}: H={H},
    ↪ p={p}\033[0m")

        if p < 0.05:
            posthoc = posthoc_conover(df, val_col=zmienna_ilosciowa,
    ↪ group_col=zmienna_jakosciowa)
            print(f"Test post-hoc Conovera dla {zmienna_jakosciowa}:\n",
    ↪ posthoc, '\n')

    print('\033[92m -----
    ↪ ----- \033[0m')
```

Zmienna ilościowa: Age

Test Kruskala-Wallis dla HeartDiseaseorAttack: $H=6839.845434096799$,

$p=0.0$

Test post-hoc Conovera dla HeartDiseaseorAttack:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla HighBP: H=4247.9805974524315, p=0.0

Test post-hoc Conovera dla HighBP:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla HighChol: H=2222.6508120287594, p=0.0

Test post-hoc Conovera dla HighChol:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla CholCheck: H=594.3363221141012,

p=2.85564164991837e-131

Test post-hoc Conovera dla CholCheck:

	0.0	1.0
0.0	1.000000e+00	4.396249e-132
1.0	4.396249e-132	1.000000e+00

Test Kruskala-Wallis dla Smoker: H=317.6125025012863,

p=4.796847194000664e-71

Test post-hoc Conovera dla Smoker:

	0.0	1.0
0.0	1.000000e+00	2.821514e-71
1.0	2.821514e-71	1.000000e+00

Test Kruskala-Wallis dla Stroke: H=735.866849271173,

p=4.747841652967501e-162

Test post-hoc Conovera dla Stroke:

	0.0	1.0
0.0	1.000000e+00	2.675745e-163
1.0	2.675745e-163	1.000000e+00

Test Kruskala-Wallis dla Diabetes: H=1015.255339032773,

p=3.4682010679119144e-221

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
0.0	1.000000e+00	2.686755e-22	1.438917e-214
1.0	2.686755e-22	1.000000e+00	1.001626e-01
2.0	1.438917e-214	1.001626e-01	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: H=258.13483704405263,

p=4.37605379869035e-58

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.000000e+00	3.084606e-58
1.0	3.084606e-58	1.000000e+00

Test Kruskala-Wallis dla Fruits: H=451.0964069975477,

p=4.1639712267427293e-100

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	1.421798e-100
1.0	1.421798e-100	1.000000e+00

Test Kruskala-Wallis dla Veggies: H=0.3312403844167476,

p=0.5649296160216588

Test Kruskala-Wallis dla HvyAlcoholConsump: H=185.42355735062335,

p=3.1719936927690536e-42

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000e+00	2.650261e-42
1.0	2.650261e-42	1.000000e+00

Test Kruskala-Wallis dla AnyHealthcare: H=1361.7897312884493,

p=4.223609881734174e-298

Test post-hoc Conovera dla AnyHealthcare:

	0.0	1.0
0.0	1.000000e+00	2.014279e-302
1.0	2.014279e-302	1.000000e+00

Test Kruskala-Wallis dla NoDocbcCost: H=1230.049724217095,

p=1.7977219639538318e-269

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.000000e+00	5.446084e-273
1.0	5.446084e-273	1.000000e+00

Test Kruskala-Wallis dla Sex: H=16.259291005733896,

p=5.5238054950642386e-05

Test post-hoc Conovera dla Sex:

	0.0	1.0
0.0	1.000000	0.000055
1.0	0.000055	1.000000

Zmienna ilościowa: BMI

Test Kruskala-Wallis dla HeartDiseaseorAttack: H=322.51038765451233,

p=4.1123590557108434e-72

Test post-hoc Conovera dla HeartDiseaseorAttack:

	0.0	1.0
0.0	1.000000e+00	2.379110e-72
1.0	2.379110e-72	1.000000e+00

Test Kruskala-Wallis dla HighBP: H=2144.225379970724, p=0.0

Test post-hoc Conovera dla HighBP:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla HighChol: H=650.3621898285237,

p=1.8635543057038996e-143

Test post-hoc Conovera dla HighChol:

	0.0	1.0
0.0	1.000000e+00	1.978033e-144
1.0	1.978033e-144	1.000000e+00

Test Kruskala-Wallis dla CholCheck: H=68.11867415152581,

p=1.5394581078947327e-16

Test post-hoc Conovera dla CholCheck:

	0.0	1.0
0.0	1.000000e+00	1.503348e-16
1.0	1.503348e-16	1.000000e+00

Test Kruskala-Wallis dla Smoker: H=0.07508181271407045,

p=0.784076470357582

Test Kruskala-Wallis dla Stroke: H=7.03541823026797,

p=0.007991320937464302

Test post-hoc Conovera dla Stroke:

	0.0	1.0
0.0	1.00000	0.00799
1.0	0.00799	1.00000

Test Kruskala-Wallis dla Diabetes: H=2872.4857454050075, p=0.0

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
0.0	1.000000e+00	9.745883e-46	0.000000e+00

1.0	9.745883e-46	1.000000e+00	7.153533e-08
2.0	0.000000e+00	7.153533e-08	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: H=714.8009197554815,
p=1.8079836219084147e-157

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.000000e+00	1.199726e-158
1.0	1.199726e-158	1.000000e+00

Test Kruskala-Wallis dla Fruits: H=254.92208278458494,
p=2.1949374817968147e-57

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	1.560695e-57
1.0	1.560695e-57	1.000000e+00

Test Kruskala-Wallis dla Veggies: H=61.40585946906697,
p=4.644263231106856e-15

Test post-hoc Conovera dla Veggies:

	0.0	1.0
0.0	1.000000e+00	4.555862e-15
1.0	4.555862e-15	1.000000e+00

Test Kruskala-Wallis dla HvyAlcoholConsump: H=187.4482794779205,
p=1.1463991692890623e-42

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000e+00	9.540399e-43
1.0	9.540399e-43	1.000000e+00

Test Kruskala-Wallis dla AnyHealthcare: H=0.03776682399851932,
p=0.8459121078415075

Test Kruskala-Wallis dla NoDocbcCost: H=112.2216957713607,
p=3.195087186146079e-26

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.000000e+00	2.993200e-26
1.0	2.993200e-26	1.000000e+00

Test Kruskala-Wallis dla Sex: H=150.54098615959708,
p=1.3204270286202207e-34

Test post-hoc Conovera dla Sex:

	0.0	1.0
--	-----	-----

```
0.0 1.000000e+00 1.173360e-34
1.0 1.173360e-34 1.000000e+00
```

```
- - - - -
- - - - -
```

Zmienna ilościowa: GenHlth

Test Kruskala-Wallis dla HeartDiseaseorAttack: H=6854.543328526066,

p=0.0

Test post-hoc Conovera dla HeartDiseaseorAttack:

```
0.0 1.0
0.0 1.0 0.0
1.0 0.0 1.0
```

Test Kruskala-Wallis dla HighBP: H=4233.501953685382, p=0.0

Test post-hoc Conovera dla HighBP:

```
0.0 1.0
0.0 1.0 0.0
1.0 0.0 1.0
```

Test Kruskala-Wallis dla HighChol: H=2211.194659890236, p=0.0

Test post-hoc Conovera dla HighChol:

```
0.0 1.0
0.0 1.0 0.0
1.0 0.0 1.0
```

Test Kruskala-Wallis dla CholCheck: H=191.08227528688005,

p=1.8454282643457846e-43

Test post-hoc Conovera dla CholCheck:

```
0.0 1.0
0.0 1.000000e+00 1.524696e-43
1.0 1.524696e-43 1.000000e+00
```

Test Kruskala-Wallis dla Smoker: H=1216.8995461367194,

p=1.2959181792322035e-266

Test post-hoc Conovera dla Smoker:

```
0.0 1.0
0.0 1.000000e+00 4.671780e-270
1.0 4.671780e-270 1.000000e+00
```

Test Kruskala-Wallis dla Stroke: H=2138.4369313805373, p=0.0

Test post-hoc Conovera dla Stroke:

```
0.0 1.0
0.0 1.0 0.0
1.0 0.0 1.0
```

Test Kruskala-Wallis dla Diabetes: $H=4284.104863619207$, $p=0.0$

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
0.0	1.000000e+00	1.237770e-50	0.000000e+00
1.0	1.237770e-50	1.000000e+00	5.700700e-21
2.0	0.000000e+00	5.700700e-21	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: $H=3296.043580765923$, $p=0.0$

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla Fruits: $H=246.67923788914467$,

$p=1.3753549251816627e-55$

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	9.995098e-56
1.0	9.995098e-56	1.000000e+00

Test Kruskala-Wallis dla Veggies: $H=417.45150896736607$,

$p=8.753050847134502e-93$

Test post-hoc Conovera dla Veggies:

	0.0	1.0
0.0	1.000000e+00	3.490133e-93
1.0	3.490133e-93	1.000000e+00

Test Kruskala-Wallis dla HvyAlcoholConsump: $H=219.0803892124425$,

$p=1.4354101775159208e-49$

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000e+00	1.116324e-49
1.0	1.116324e-49	1.000000e+00

Test Kruskala-Wallis dla AnyHealthcare: $H=13.964522404797087$,

$p=0.00018629298514255682$

Test post-hoc Conovera dla AnyHealthcare:

	0.0	1.0
0.0	1.000000	0.000186
1.0	0.000186	1.000000

Test Kruskala-Wallis dla NoDocbcCost: $H=1209.5125533501578$,

$p=5.223513103473983e-265$

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.000000e+00	2.074614e-268
1.0	2.074614e-268	1.000000e+00

Test Kruskala-Wallis dla Sex: $H=9.0649488237901$,

$p=0.0026055582718585084$

Test post-hoc Conovera dla Sex:

	0.0	1.0
0.0	1.000000	0.002605
1.0	0.002605	1.000000

Zmienna ilościowa: MentHlth

Test Kruskala-Wallis dla HeartDiseaseorAttack: $H=46.610228734880295$,

$p=8.660699592377352e-12$

Test post-hoc Conovera dla HeartDiseaseorAttack:

	0.0	1.0
0.0	1.000000e+00	8.566454e-12
1.0	8.566454e-12	1.000000e+00

Test Kruskala-Wallis dla HighBP: $H=24.972757598220102$,

$p=5.814611535907313e-07$

Test post-hoc Conovera dla HighBP:

	0.0	1.0
0.0	1.000000e+00	5.797187e-07
1.0	5.797187e-07	1.000000e+00

Test Kruskala-Wallis dla HighChol: $H=62.780577805854804$,

$p=2.3106622840163e-15$

Test post-hoc Conovera dla HighChol:

	0.0	1.0
0.0	1.000000e+00	2.264671e-15
1.0	2.264671e-15	1.000000e+00

Test Kruskala-Wallis dla CholCheck: $H=5.866756505808795$,

$p=0.015429454746059385$

Test post-hoc Conovera dla CholCheck:

	0.0	1.0
0.0	1.000000	0.015428
1.0	0.015428	1.000000

Test Kruskala-Wallis dla Smoker: H=104.59081448747891,

p=1.5015128017913248e-24

Test post-hoc Conovera dla Smoker:

	0.0	1.0
0.0	1.000000e+00	1.418871e-24
1.0	1.418871e-24	1.000000e+00

Test Kruskala-Wallis dla Stroke: H=252.09000607051217,

p=9.09508875866175e-57

Test post-hoc Conovera dla Stroke:

	0.0	1.0
0.0	1.000000e+00	6.516196e-57
1.0	6.516196e-57	1.000000e+00

Test Kruskala-Wallis dla Diabetes: H=141.2712748592482,

p=2.105389431668546e-31

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
0.0	1.000000e+00	0.000002	2.221026e-29
1.0	1.560728e-06	1.000000	4.908689e-01
2.0	2.221026e-29	0.490869	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: H=391.1115222913457,

p=4.7413103667666255e-87

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.000000e+00	2.116549e-87
1.0	2.116549e-87	1.000000e+00

Test Kruskala-Wallis dla Fruits: H=126.71353136040926,

p=2.1462118680734946e-29

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	1.974451e-29
1.0	1.974451e-29	1.000000e+00

Test Kruskala-Wallis dla Veggies: H=70.97918786559347,

p=3.61013123650135e-17

Test post-hoc Conovera dla Veggies:

	0.0	1.0
0.0	1.000000e+00	3.518160e-17
1.0	3.518160e-17	1.000000e+00

Test Kruskala-Wallis dla HvyAlcoholConsump: $H=14.804087340572979$,

$p=0.0001192767000712262$

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000	0.000119
1.0	0.000119	1.000000

Test Kruskala-Wallis dla AnyHealthcare: $H=115.66674329629627$,

$p=5.622698763836056e-27$

Test post-hoc Conovera dla AnyHealthcare:

	0.0	1.0
0.0	1.000000e+00	5.245807e-27
1.0	5.245807e-27	1.000000e+00

Test Kruskala-Wallis dla NoDocbcCost: $H=1741.9073021735383$, $p=0.0$

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla Sex: $H=931.0876824977184$,

$p=1.7133979399976286e-204$

Test post-hoc Conovera dla Sex:

	0.0	1.0
0.0	1.000000e+00	1.688253e-206
1.0	1.688253e-206	1.000000e+00

Zmienna ilościowa: PhysHlth

Test Kruskala-Wallis dla HeartDiseaseorAttack: $H=2418.5493735891177$,

$p=0.0$

Test post-hoc Conovera dla HeartDiseaseorAttack:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla HighBP: $H=1263.7163476498595$,

$p=8.674691418337892e-277$

Test post-hoc Conovera dla HighBP:

	0.0	1.0
0.0	1.000000e+00	1.668821e-280
1.0	1.668821e-280	1.000000e+00

Test Kruskala-Wallis dla HighChol: H=742.6411859013946,

p=1.5976556108986427e-163

Test post-hoc Conovera dla HighChol:

	0.0	1.0
0.0	1.000000e+00	8.534417e-165
1.0	8.534417e-165	1.000000e+00

Test Kruskala-Wallis dla CholCheck: H=95.19558833426883,

p=1.7247422060154753e-22

Test post-hoc Conovera dla CholCheck:

	0.0	1.0
0.0	1.000000e+00	1.645889e-22
1.0	1.645889e-22	1.000000e+00

Test Kruskala-Wallis dla Smoker: H=467.74837396131556,

p=9.902423249820515e-104

Test post-hoc Conovera dla Smoker:

	0.0	1.0
0.0	1.000000e+00	3.117334e-104
1.0	3.117334e-104	1.000000e+00

Test Kruskala-Wallis dla Stroke: H=1353.7390017419877,

p=2.372264845901247e-296

Test post-hoc Conovera dla Stroke:

	0.0	1.0
0.0	1.000000e+00	1.273730e-300
1.0	1.273730e-300	1.000000e+00

Test Kruskala-Wallis dla Diabetes: H=1580.0637926321167, p=0.0

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
0.0	1.000000e+00	1.298936e-18	0.000000e+00
1.0	1.298936e-18	1.000000e+00	2.784888e-08
2.0	0.000000e+00	2.784888e-08	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: H=2030.4992064526487, p=0.0

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla Fruits: H=62.69074164574764,

p=2.418498910792187e-15

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	2.370500e-15
1.0	2.370500e-15	1.000000e+00

Test Kruskala-Wallis dla Veggies: $H=114.1590976393289$,

$p=1.2026170158258176e-26$

Test post-hoc Conovera dla Veggies:

	0.0	1.0
0.0	1.000000e+00	1.124043e-26
1.0	1.124043e-26	1.000000e+00

Test Kruskala-Wallis dla HvyAlcoholConsump: $H=130.35719384281956$,

$p=3.422926748380284e-30$

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000e+00	3.133561e-30
1.0	3.133561e-30	1.000000e+00

Test Kruskala-Wallis dla AnyHealthcare: $H=0.1349853025373879$,

$p=0.7133180909675929$

Test Kruskala-Wallis dla NoDocbcCost: $H=1169.499347255941$,

$p=2.5942567654347234e-256$

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.000000e+00	1.723052e-259
1.0	1.723052e-259	1.000000e+00

Test Kruskala-Wallis dla Sex: $H=300.62020700997294$,

$p=2.4135128196493395e-67$

Test post-hoc Conovera dla Sex:

	0.0	1.0
0.0	1.000000e+00	1.500727e-67
1.0	1.500727e-67	1.000000e+00

Zmienna ilościowa: Education

Test Kruskala-Wallis dla HeartDiseaseorAttack: $H=837.8998911691073$,

$p=3.105829836848314e-184$

Test post-hoc Conovera dla HeartDiseaseorAttack:

	0.0	1.0
0.0	1.000000e+00	7.410278e-186
1.0	7.410278e-186	1.000000e+00

Test Kruskala-Wallis dla HighBP: H=753.6317289573462,

p=6.51231603075726e-166

Test post-hoc Conovera dla HighBP:

	0.0	1.0
0.0	1.000000e+00	3.185898e-167
1.0	3.185898e-167	1.000000e+00

Test Kruskala-Wallis dla HighChol: H=177.2184976114488,

p=1.9622790332820965e-40

Test post-hoc Conovera dla HighChol:

	0.0	1.0
0.0	1.000000e+00	1.665392e-40
1.0	1.665392e-40	1.000000e+00

Test Kruskala-Wallis dla CholCheck: H=1.5412338010708346,

p=0.21443424974779304

Test Kruskala-Wallis dla Smoker: H=981.9428728733069,

p=1.5110198631438243e-215

Test post-hoc Conovera dla Smoker:

	0.0	1.0
0.0	1.000000e+00	8.826994e-218
1.0	8.826994e-218	1.000000e+00

Test Kruskala-Wallis dla Stroke: H=327.210665962163,

p=3.8932809115807754e-73

Test post-hoc Conovera dla Stroke:

	0.0	1.0
0.0	1.000000e+00	2.216269e-73
1.0	2.216269e-73	1.000000e+00

Test Kruskala-Wallis dla Diabetes: H=554.5974311546005,

p=3.721328301994776e-121

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
0.0	1.000000e+00	2.412873e-14	2.230921e-116
1.0	2.412873e-14	1.000000e+00	4.829584e-01
2.0	2.230921e-116	4.829584e-01	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: H=1546.7379620194881, p=0.0

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla Fruits: H=485.20239019177205,

p=1.576622605121674e-107

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	4.543511e-108
1.0	4.543511e-108	1.000000e+00

Test Kruskala-Wallis dla Veggies: H=867.1927258086091,

p=1.3300718323568254e-190

Test post-hoc Conovera dla Veggies:

	0.0	1.0
0.0	1.000000e+00	2.427994e-192
1.0	2.427994e-192	1.000000e+00

Test Kruskala-Wallis dla HvyAlcoholConsump: H=68.50623568887329,

p=1.2647703561574025e-16

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000e+00	1.234763e-16
1.0	1.234763e-16	1.000000e+00

Test Kruskala-Wallis dla AnyHealthcare: H=411.64903607990504,

p=1.6038930486773176e-91

Test post-hoc Conovera dla AnyHealthcare:

	0.0	1.0
0.0	1.000000e+00	6.560549e-92
1.0	6.560549e-92	1.000000e+00

Test Kruskala-Wallis dla NoDocbcCost: H=339.1322010830018,

p=9.859671934394474e-76

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.000000e+00	5.381610e-76
1.0	5.381610e-76	1.000000e+00

Test Kruskala-Wallis dla Sex: H=113.70362070698928,

p=1.5131654251193105e-26

Test post-hoc Conovera dla Sex:

	0.0	1.0
0.0	1.000000e+00	1.415071e-26
1.0	1.415071e-26	1.000000e+00

Zmienna ilościowa: Income

Test Kruskala-Wallis dla HeartDiseaseorAttack: H=1987.086443644765,

p=0.0

Test post-hoc Conovera dla HeartDiseaseorAttack:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla HighBP: H=1455.9322946381167, p=0.0

Test post-hoc Conovera dla HighBP:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla HighChol: H=352.2694262289578,

p=1.3581395299710683e-78

Test post-hoc Conovera dla HighChol:

	0.0	1.0
0.0	1.000000e+00	7.064980e-79
1.0	7.064980e-79	1.000000e+00

Test Kruskala-Wallis dla CholCheck: H=0.31403767990842035,

p=0.5752130081991844

Test Kruskala-Wallis dla Smoker: H=585.7664751206878,

p=2.0881688673637763e-129

Test post-hoc Conovera dla Smoker:

	0.0	1.0
0.0	1.000000e+00	3.392699e-130
1.0	3.392699e-130	1.000000e+00

Test Kruskala-Wallis dla Stroke: H=1184.676623759166,

p=1.3047102069437284e-259

Test post-hoc Conovera dla Stroke:

	0.0	1.0
0.0	1.000000e+00	7.145026e-263
1.0	7.145026e-263	1.000000e+00

Test Kruskala-Wallis dla Diabetes: H=1312.5664238595616,

p=9.546023367993069e-286

Test post-hoc Conovera dla Diabetes:

	0.0	1.0	2.0
--	-----	-----	-----

0.0	1.000000e+00	3.806257e-29	3.030854e-277
1.0	3.806257e-29	1.000000e+00	8.199572e-02
2.0	3.030854e-277	8.199572e-02	1.000000e+00

Test Kruskala-Wallis dla PhysActivity: H=1815.7507559877617, p=0.0

Test post-hoc Conovera dla PhysActivity:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallis dla Fruits: H=185.84971582425874,

p=2.5603533962293097e-42

Test post-hoc Conovera dla Fruits:

	0.0	1.0
0.0	1.000000e+00	2.137443e-42
1.0	2.137443e-42	1.000000e+00

Test Kruskala-Wallis dla Veggies: H=817.9338791809075,

p=6.807167948602811e-180

Test post-hoc Conovera dla Veggies:

	0.0	1.0
0.0	1.000000e+00	1.938874e-181
1.0	1.938874e-181	1.000000e+00

Test Kruskala-Wallis dla HvyAlcoholConsump: H=274.6154571322274,

p=1.1194930820613237e-61

Test post-hoc Conovera dla HvyAlcoholConsump:

	0.0	1.0
0.0	1.000000e+00	7.533614e-62
1.0	7.533614e-62	1.000000e+00

Test Kruskala-Wallis dla AnyHealthcare: H=688.92253751017,

p=7.666523834662174e-152

Test post-hoc Conovera dla AnyHealthcare:

	0.0	1.0
0.0	1.000000e+00	6.176819e-153
1.0	6.176819e-153	1.000000e+00

Test Kruskala-Wallis dla NoDocbcCost: H=1790.2941743984657, p=0.0

Test post-hoc Conovera dla NoDocbcCost:

	0.0	1.0
0.0	1.0	0.0
1.0	0.0	1.0

Test Kruskala-Wallisa dla Sex: $H=1339.0411674811078$,

$p=3.70786854416268e-293$

Test post-hoc Conovera dla Sex:

	0.0	1.0
0.0	1.000000e+00	2.467231e-297
1.0	2.467231e-297	1.000000e+00

Pomiędzy wszystkimi badanymi grupami istnieją statystycznie istotne różnice, z wyjątkiem osób z cukrzycą i stanem przedcukrzycowym, gdzie nie ma wystarczających dowodów na istotne różnice w samoocenie zdrowia psychicznego, poziomie wykształcenia oraz dochodach tych dwóch grup

Utworzenie ramki danych wyłącznie z zmiennymi zależnymi (bez kolumny docelowej):

```
[20]: df_copy = df
      X = df_copy.drop('HeartDiseaseorAttack', axis=1).values
      print(X)
```

```
[[ 0.  0.  1. ...  3.  6.  8.]
 [ 0.  0.  1. ...  3.  6.  4.]
 [ 0.  1.  1. ... 10.  4.  2.]
 ...
 [ 1.  1.  1. ...  8.  3.  2.]
 [ 1.  0.  1. ... 12.  2.  1.]
 [ 1.  1.  1. ...  9.  6.  2.]]
```

Utworzenie ramki danych wyłącznie z zmienną niezależną (kolumna docelowa):

```
[21]: Y = df_copy['HeartDiseaseorAttack'].values
      print(Y)
```

```
[0. 0. 0. ... 1. 1. 1.]
```

Podział danych na zbiór treningowy, walidacyjny i testowy w proporcjach: 70% - 15% - 15%

```
[22]: from sklearn.model_selection import train_test_split
      X_train, X_temp, y_train, y_temp = train_test_split(X, Y, test_size=0.3,
      ↪random_state=42)
      X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
      ↪random_state=42)
```

Konwersja danych wejściowych modelu do tensorów Pytorch'a:

```
[23]: import torch
      from torch.utils.data import TensorDataset
      import numpy as np

      X_train = torch.tensor(np.array(X_train), dtype=torch.float32)
```

```
X_val = torch.tensor(np.array(X_val), dtype=torch.float32)
X_test = torch.tensor(np.array(X_test), dtype=torch.float32)
y_train = torch.tensor(np.array(y_train), dtype=torch.float32).reshape(-1, 1)
y_val = torch.tensor(np.array(y_val), dtype=torch.float32).reshape(-1, 1)
y_test = torch.tensor(np.array(y_test), dtype=torch.float32).reshape(-1, 1)
```

Utworzenie z powstałych tensorów zestawów danych oraz loaderów ułatwiających ładowanie danych do sieci - paczkami (batch_size=16), w sposób losowy (shuffle=True)

```
[24]: train_dataset = TensorDataset(X_train, y_train)
      val_dataset = TensorDataset(X_val, y_val)
      test_dataset = TensorDataset(X_test, y_test)
      batch_size = 64
      trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
      ↪shuffle=True)
      valloader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size,
      ↪shuffle=False)
      testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
      ↪shuffle=False)
```

Sprawdzenie dostępności procesora graficznego GPU (w razie wykrycia, będzie można przenieść na niego dane w celu przyspieszenia obliczeń):

```
[25]: device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
      print(device)
```

cpu

Zdefiniowanie struktury głębokiej sieci neuronowej: > Model dziedziczy po klasie nn.Module, która zapewnia mu niezbędne funkcjonalności w procesie trenu, jak i inicjalizuje wagi

Następnie zdefiniowane są 3 warstwy w pełni połączone:

- Pierwsza przyjmuje 21 cech wejściowych - czyli tyle ile mamy zmiennych zależnych. Generuje ona 15 cech wyjściowych, wprowadza nieliniowość przy użyciu LeakyReLU. Stosuje także mechanizm normalizacji oraz dropout w celu zapobiegania przeuczeniu.
- Druga stosując podobne mechanizmy co pierwsza redukuje złożoność modelu do 10 cech wyjściowych.
- Trzecia warstwa redukuje złożoność do 5 cech wyjściowych, które wyjściowa warstwa przekształca w finalną predykcję.

```
[26]: %%writefile ../backend/Models/net/predict_1.py
import torch
import torch.nn as nn
#####
class Model_1(nn.Module):
    def __init__(self):
        super(Model_1, self).__init__()
        self.hidden1 = nn.Linear(21, 15)
```

```

self.bn1 = nn.BatchNorm1d(15)
self.dropout1 = nn.Dropout(p=0.3)
self.act1 = nn.LeakyReLU(negative_slope=0.01)

self.hidden2 = nn.Linear(15, 10)
self.bn2 = nn.BatchNorm1d(10)
self.dropout2 = nn.Dropout(p=0.3)
self.act2 = nn.LeakyReLU(negative_slope=0.01)

self.hidden3 = nn.Linear(10, 5)
self.bn3 = nn.BatchNorm1d(5)
self.dropout3 = nn.Dropout(p=0.3)
self.act3 = nn.LeakyReLU(negative_slope=0.01)

self.output = nn.Linear(5, 1)

def forward(self, x):
    x = self.act1(self.dropout1(self.bn1(self.hidden1(x))))
    x = self.act2(self.dropout2(self.bn2(self.hidden2(x))))
    x = self.act3(self.dropout3(self.bn3(self.hidden3(x))))
    x = self.output(x)
    return x

```

Overwriting ../backend/Models/net/predict_1.py

```

[27]: import torch
import torch.nn as nn
#####
class Model_1(nn.Module):
    def __init__(self):
        super(Model_1, self).__init__()
        self.hidden1 = nn.Linear(21, 15)
        self.bn1 = nn.BatchNorm1d(15)
        self.dropout1 = nn.Dropout(p=0.3)
        self.act1 = nn.LeakyReLU(negative_slope=0.01)

        self.hidden2 = nn.Linear(15, 10)
        self.bn2 = nn.BatchNorm1d(10)
        self.dropout2 = nn.Dropout(p=0.3)
        self.act2 = nn.LeakyReLU(negative_slope=0.01)

        self.hidden3 = nn.Linear(10, 5)
        self.bn3 = nn.BatchNorm1d(5)
        self.dropout3 = nn.Dropout(p=0.3)
        self.act3 = nn.LeakyReLU(negative_slope=0.01)

        self.output = nn.Linear(5, 1)

```

```
def forward(self, x):
    x = self.act1(self.dropout1(self.bn1(self.hidden1(x))))
    x = self.act2(self.dropout2(self.bn2(self.hidden2(x))))
    x = self.act3(self.dropout3(self.bn3(self.hidden3(x))))
    x = self.output(x)
    return x
```

- Na koniec model przenoszony jest na dostępny procesor, graficzny lub centralny

```
[28]: model = Model_1()
model = model.to(device)
print(model)
```

```
Model_1(
  (hidden1): Linear(in_features=21, out_features=15, bias=True)
  (bn1): BatchNorm1d(15, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (dropout1): Dropout(p=0.3, inplace=False)
  (act1): LeakyReLU(negative_slope=0.01)
  (hidden2): Linear(in_features=15, out_features=10, bias=True)
  (bn2): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (dropout2): Dropout(p=0.3, inplace=False)
  (act2): LeakyReLU(negative_slope=0.01)
  (hidden3): Linear(in_features=10, out_features=5, bias=True)
  (bn3): BatchNorm1d(5, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (dropout3): Dropout(p=0.3, inplace=False)
  (act3): LeakyReLU(negative_slope=0.01)
  (output): Linear(in_features=5, out_features=1, bias=True)
)
```

Odpowiednią funkcją straty (mierzącą jak dobrze model radzi sobie z przewidywaniem wyników) wydaje się być BCEWithLogitsLoss. Łączy ona w sobie warstwę sigmoidalną oraz stratę BCE w jednej klasie. Jest więc odpowiednia do określania prawdopodobieństwa przynależności określonego pacjenta do grupy chorych lub zdrowych.

<https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html>

```
[29]: criterion = nn.BCEWithLogitsLoss()
print(criterion)
```

```
BCEWithLogitsLoss()
```

Optymizatorem odpowiednim do dostosowywania parametrów modelu w celu minimalizacji funkcji straty jest AdamW. Jest to modyfikacja popularnego algorytmu Adam, która często zapewnia dokładniejsze wyniki. Wartość współczynnika uczenia (learning rate) została doświadczalnie ustawiona na 0.01, aby model nie przeskakiwał przez szukane minima oraz nie uczył się zbyt wolno. Dodatkowo zastosowano scheduler ReduceLROnPlateau, który zmniejsza współczynnik uczenia,

gdy model przestaje poprawiać wyniki, co pozwala na dalsze fine-tuning modeli.

<https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html>

```
[30]: import torch.optim as optim
optimizer = optim.AdamW(model.parameters(), lr=0.0001, weight_decay=1e-4)
print(optimizer)
```

```
AdamW (
Parameter Group 0
  amsgrad: False
  betas: (0.9, 0.999)
  capturable: False
  differentiable: False
  eps: 1e-08
  foreach: None
  fused: None
  lr: 0.0001
  maximize: False
  weight_decay: 0.0001
)
```

Inicjalizacja niezbędnych zmiennych w procesie szkolenia i postprocessingu:

```
[31]: patience = 10          # "Cierpliwość modelu" - maksymalna możliwa ilość epok
      ↪ bez poprawy straty na zbiorze testowym
best_loss = None            # Najmniejsza wartość straty na zbiorze testowym
best_acc = None             # Największa dokładność na zbiorze testowym
early_stop_counter = 0      # Licznik dla mechanizmu "early stopping"
best_model = None           # Najlepszy model - taki o najwyższej dokładności na
      ↪ zbiorze testowym
epochs = 75                 # Maksymalna liczba epok szkolenia

train_losses, val_losses, test_losses = [], [], [] # Listy start
train_accuracies, val_accuracies, test_accuracies = [], [], [] # Listy
      ↪ dokładności

classification_report_test = None # Swoistość (Precision) i Czułość (Recall)
      ↪ modelu na zbiorze testowym
```

Trening modelu:

- Część treningowa: > Przeniesienie danych na odpowiednie urządzenie (CPU/GPU) > Wyzerowanie gradientów (w celu zapobiegnięcia ich akumulacji, a w konsekwencji uzyskania niepoprawnych wyników predykcji) > Przepuszczenie danych przez model > Obliczenie gradientów funkcji straty oraz aktualizacja na ich podstawie wag parametrów modelu > Dokonanie predykcji oraz wyliczenie straty/dokładności na zbiorze treningowym
- Część testowa: > Ustawienie modelu w tryb ewaluacji - w celu uzyskania niezakłóconych wyników > Wyłączenie obliczania gradientów - podczas ewaluacji są one zbędne (nie aktu-

alizujemy wag modelu) > Przepuszczenie danych przez model > Dokonanie predykcji oraz wyliczenie straty/dokładności na zbiorze testowym

- Early stopping (monitorowanie procesu uczenia w celu zapobiegnięcia przeuczenia modelu w każdej epoce): > Sprawdzenie czy strata na zbiorze testowym jest najmniejsza > Sprawdzenie czy dokładność na zbiorze testowym jest najmniejsza > Obliczanie Swoistości i Czułości modelu na zbiorze testowym > Modyfikacja licznika oraz przerwanie procesu uczenia, gdy model osiągnie szczyt swojej cierpliwości > Dostosowanie współczynnika uczenia w przypadku braku poprawy wyników
- Zapisanie najlepszego uzyskanego modelu pod określoną ścieżką

```
[32]: from sklearn.metrics import classification_report
import numpy as np
import torch
from tqdm import tqdm

for epoch in range(epochs):
    model.train()
    train_true_labels, train_predictions = [], []
    train_loss = 0

    for inputs, labels in tqdm(trainloader, desc=f"Epoch {epoch+1}/{epochs} - Training"):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels.float())
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        predicted = (outputs > 0.5).int()
        train_predictions.extend(predicted.cpu().numpy())
        train_true_labels.extend(labels.cpu().numpy())

    train_loss /= len(trainloader)
    train_accuracy = np.mean(np.array(train_predictions) == np.
    array(train_true_labels))
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    # Validation phase
    model.eval()
    val_true_labels, val_predictions = [], []
    val_loss = 0
```

```

    with torch.no_grad():
        for inputs, labels in tqdm(valloader, desc=f"Epoch {epoch+1}/{epochs} - Validation"):
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels.float())
            val_loss += loss.item()

            predicted = (outputs > 0.5).int()
            val_predictions.extend(predicted.cpu().numpy())
            val_true_labels.extend(labels.cpu().numpy())

        val_loss /= len(valloader)
        val_accuracy = np.mean(np.array(val_predictions) == np.
    array(val_true_labels))
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)

        print(f'Epoch {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}, Train
    Accuracy: {train_accuracy:.4f}, Val Loss: {val_loss:.4f}, Val Accuracy:
    {val_accuracy:.4f}')

    # Early Stopping
    if best_loss is None or val_loss < best_loss:
        best_loss = val_loss
        best_acc = val_accuracy
        best_model = model.state_dict()
        early_stop_counter = 0
    else:
        early_stop_counter += 1
        if early_stop_counter >= patience:
            print("Early stopping")
            break

model.load_state_dict(best_model)

model.eval()
test_true_labels, test_predictions = [], []
test_loss = 0

with torch.no_grad():
    for inputs, labels in tqdm(testloader, desc="Testing"):
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, labels.float())

```

```

        test_loss += loss.item()

        predicted = (outputs > 0.5).int()
        test_predictions.extend(predicted.cpu().numpy())
        test_true_labels.extend(labels.cpu().numpy())

test_loss /= len(testloader)
test_accuracy = np.mean(np.array(test_predictions) == np.
    ↪array(test_true_labels))
test_losses.append(test_loss)
test accuracies.append(test_accuracy)

print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')

torch.save(model, os.path.join(os.getcwd(), '../backend/Models/predict_1.pth'))

```

```

Epoch 1/150 - Training: 100%|      | 519/519 [00:02<00:00, 225.55it/s]
Epoch 1/150 - Validation: 100%|    | 112/112 [00:00<00:00, 769.51it/s]

Epoch 1/150, Train Loss: 0.7179, Train Accuracy: 0.4997, Val Loss: 0.6764, Val
Accuracy: 0.5047

```

```

Epoch 2/150 - Training: 100%|      | 519/519 [00:02<00:00, 229.62it/s]
Epoch 2/150 - Validation: 100%|    | 112/112 [00:00<00:00, 795.97it/s]

Epoch 2/150, Train Loss: 0.6774, Train Accuracy: 0.4997, Val Loss: 0.6610, Val
Accuracy: 0.5047

```

```

Epoch 3/150 - Training: 100%|      | 519/519 [00:02<00:00, 231.07it/s]
Epoch 3/150 - Validation: 100%|    | 112/112 [00:00<00:00, 748.12it/s]

Epoch 3/150, Train Loss: 0.6589, Train Accuracy: 0.5007, Val Loss: 0.6399, Val
Accuracy: 0.5047

```

```

Epoch 4/150 - Training: 100%|      | 519/519 [00:02<00:00, 227.80it/s]
Epoch 4/150 - Validation: 100%|    | 112/112 [00:00<00:00, 752.18it/s]

Epoch 4/150, Train Loss: 0.6410, Train Accuracy: 0.5091, Val Loss: 0.6187, Val
Accuracy: 0.5051

```

```

Epoch 5/150 - Training: 100%|      | 519/519 [00:02<00:00, 231.55it/s]
Epoch 5/150 - Validation: 100%|    | 112/112 [00:00<00:00, 783.28it/s]

Epoch 5/150, Train Loss: 0.6228, Train Accuracy: 0.5219, Val Loss: 0.5984, Val
Accuracy: 0.5081

```

```

Epoch 6/150 - Training: 100%|      | 519/519 [00:02<00:00, 229.13it/s]
Epoch 6/150 - Validation: 100%|    | 112/112 [00:00<00:00, 801.25it/s]

Epoch 6/150, Train Loss: 0.6063, Train Accuracy: 0.5500, Val Loss: 0.5809, Val
Accuracy: 0.5459

```

```

Epoch 7/150 - Training: 100%|      | 519/519 [00:02<00:00, 230.07it/s]
Epoch 7/150 - Validation: 100%|    | 112/112 [00:00<00:00, 775.93it/s]

```


Epoch 7/150, Train Loss: 0.5933, Train Accuracy: 0.5821, Val Loss: 0.5653, Val Accuracy: 0.5930

Epoch 8/150 - Training: 100%| | 519/519 [00:02<00:00, 229.31it/s]

Epoch 8/150 - Validation: 100%| | 112/112 [00:00<00:00, 824.26it/s]

Epoch 8/150, Train Loss: 0.5825, Train Accuracy: 0.6084, Val Loss: 0.5499, Val Accuracy: 0.6720

Epoch 9/150 - Training: 100%| | 519/519 [00:02<00:00, 229.42it/s]

Epoch 9/150 - Validation: 100%| | 112/112 [00:00<00:00, 813.72it/s]

Epoch 9/150, Train Loss: 0.5739, Train Accuracy: 0.6349, Val Loss: 0.5376, Val Accuracy: 0.7020

Epoch 10/150 - Training: 100%| | 519/519 [00:02<00:00, 228.32it/s]

Epoch 10/150 - Validation: 100%| | 112/112 [00:00<00:00, 779.45it/s]

Epoch 10/150, Train Loss: 0.5648, Train Accuracy: 0.6504, Val Loss: 0.5303, Val Accuracy: 0.7278

Epoch 11/150 - Training: 100%| | 519/519 [00:02<00:00, 218.86it/s]

Epoch 11/150 - Validation: 100%| | 112/112 [00:00<00:00, 815.65it/s]

Epoch 11/150, Train Loss: 0.5619, Train Accuracy: 0.6599, Val Loss: 0.5279, Val Accuracy: 0.7307

Epoch 12/150 - Training: 100%| | 519/519 [00:02<00:00, 226.17it/s]

Epoch 12/150 - Validation: 100%| | 112/112 [00:00<00:00, 770.36it/s]

Epoch 12/150, Train Loss: 0.5589, Train Accuracy: 0.6637, Val Loss: 0.5220, Val Accuracy: 0.7376

Epoch 13/150 - Training: 100%| | 519/519 [00:02<00:00, 228.67it/s]

Epoch 13/150 - Validation: 100%| | 112/112 [00:00<00:00, 820.87it/s]

Epoch 13/150, Train Loss: 0.5551, Train Accuracy: 0.6751, Val Loss: 0.5177, Val Accuracy: 0.7404

Epoch 14/150 - Training: 100%| | 519/519 [00:02<00:00, 229.02it/s]

Epoch 14/150 - Validation: 100%| | 112/112 [00:00<00:00, 775.16it/s]

Epoch 14/150, Train Loss: 0.5527, Train Accuracy: 0.6812, Val Loss: 0.5158, Val Accuracy: 0.7414

Epoch 15/150 - Training: 100%| | 519/519 [00:02<00:00, 175.69it/s]

Epoch 15/150 - Validation: 100%| | 112/112 [00:00<00:00, 717.01it/s]

Epoch 15/150, Train Loss: 0.5542, Train Accuracy: 0.6836, Val Loss: 0.5144, Val Accuracy: 0.7442

Epoch 16/150 - Training: 100%| | 519/519 [00:02<00:00, 214.99it/s]

Epoch 16/150 - Validation: 100%| | 112/112 [00:00<00:00, 819.65it/s]

Epoch 16/150, Train Loss: 0.5506, Train Accuracy: 0.6898, Val Loss: 0.5130, Val Accuracy: 0.7498

Epoch 17/150 - Training: 100%| | 519/519 [00:02<00:00, 232.14it/s]
Epoch 17/150 - Validation: 100%| | 112/112 [00:00<00:00, 825.41it/s]
Epoch 17/150, Train Loss: 0.5476, Train Accuracy: 0.6933, Val Loss: 0.5139, Val Accuracy: 0.7495

Epoch 18/150 - Training: 100%| | 519/519 [00:02<00:00, 230.85it/s]
Epoch 18/150 - Validation: 100%| | 112/112 [00:00<00:00, 804.16it/s]
Epoch 18/150, Train Loss: 0.5483, Train Accuracy: 0.6934, Val Loss: 0.5120, Val Accuracy: 0.7507

Epoch 19/150 - Training: 100%| | 519/519 [00:02<00:00, 214.16it/s]
Epoch 19/150 - Validation: 100%| | 112/112 [00:00<00:00, 785.33it/s]
Epoch 19/150, Train Loss: 0.5461, Train Accuracy: 0.6992, Val Loss: 0.5121, Val Accuracy: 0.7514

Epoch 20/150 - Training: 100%| | 519/519 [00:02<00:00, 213.29it/s]
Epoch 20/150 - Validation: 100%| | 112/112 [00:00<00:00, 735.64it/s]
Epoch 20/150, Train Loss: 0.5449, Train Accuracy: 0.7036, Val Loss: 0.5090, Val Accuracy: 0.7521

Epoch 21/150 - Training: 100%| | 519/519 [00:02<00:00, 231.80it/s]
Epoch 21/150 - Validation: 100%| | 112/112 [00:00<00:00, 751.98it/s]
Epoch 21/150, Train Loss: 0.5472, Train Accuracy: 0.7002, Val Loss: 0.5086, Val Accuracy: 0.7529

Epoch 22/150 - Training: 100%| | 519/519 [00:02<00:00, 214.45it/s]
Epoch 22/150 - Validation: 100%| | 112/112 [00:00<00:00, 694.03it/s]
Epoch 22/150, Train Loss: 0.5462, Train Accuracy: 0.6996, Val Loss: 0.5075, Val Accuracy: 0.7531

Epoch 23/150 - Training: 100%| | 519/519 [00:02<00:00, 228.37it/s]
Epoch 23/150 - Validation: 100%| | 112/112 [00:00<00:00, 814.61it/s]
Epoch 23/150, Train Loss: 0.5418, Train Accuracy: 0.7041, Val Loss: 0.5087, Val Accuracy: 0.7512

Epoch 24/150 - Training: 100%| | 519/519 [00:02<00:00, 231.99it/s]
Epoch 24/150 - Validation: 100%| | 112/112 [00:00<00:00, 721.38it/s]
Epoch 24/150, Train Loss: 0.5411, Train Accuracy: 0.7048, Val Loss: 0.5080, Val Accuracy: 0.7538

Epoch 25/150 - Training: 100%| | 519/519 [00:02<00:00, 236.55it/s]
Epoch 25/150 - Validation: 100%| | 112/112 [00:00<00:00, 827.41it/s]
Epoch 25/150, Train Loss: 0.5431, Train Accuracy: 0.7044, Val Loss: 0.5087, Val Accuracy: 0.7507

Epoch 26/150 - Training: 100%| | 519/519 [00:02<00:00, 230.23it/s]
Epoch 26/150 - Validation: 100%| | 112/112 [00:00<00:00, 783.99it/s]

Epoch 26/150, Train Loss: 0.5394, Train Accuracy: 0.7081, Val Loss: 0.5070, Val Accuracy: 0.7519

Epoch 27/150 - Training: 100%| | 519/519 [00:02<00:00, 207.93it/s]

Epoch 27/150 - Validation: 100%| | 112/112 [00:00<00:00, 825.72it/s]

Epoch 27/150, Train Loss: 0.5406, Train Accuracy: 0.7057, Val Loss: 0.5073, Val Accuracy: 0.7514

Epoch 28/150 - Training: 100%| | 519/519 [00:02<00:00, 232.51it/s]

Epoch 28/150 - Validation: 100%| | 112/112 [00:00<00:00, 730.36it/s]

Epoch 28/150, Train Loss: 0.5371, Train Accuracy: 0.7101, Val Loss: 0.5061, Val Accuracy: 0.7533

Epoch 29/150 - Training: 100%| | 519/519 [00:02<00:00, 228.62it/s]

Epoch 29/150 - Validation: 100%| | 112/112 [00:00<00:00, 783.51it/s]

Epoch 29/150, Train Loss: 0.5411, Train Accuracy: 0.7079, Val Loss: 0.5075, Val Accuracy: 0.7533

Epoch 30/150 - Training: 100%| | 519/519 [00:02<00:00, 233.27it/s]

Epoch 30/150 - Validation: 100%| | 112/112 [00:00<00:00, 756.75it/s]

Epoch 30/150, Train Loss: 0.5390, Train Accuracy: 0.7130, Val Loss: 0.5081, Val Accuracy: 0.7553

Epoch 31/150 - Training: 100%| | 519/519 [00:02<00:00, 231.48it/s]

Epoch 31/150 - Validation: 100%| | 112/112 [00:00<00:00, 826.40it/s]

Epoch 31/150, Train Loss: 0.5386, Train Accuracy: 0.7123, Val Loss: 0.5074, Val Accuracy: 0.7545

Epoch 32/150 - Training: 100%| | 519/519 [00:02<00:00, 227.88it/s]

Epoch 32/150 - Validation: 100%| | 112/112 [00:00<00:00, 807.80it/s]

Epoch 32/150, Train Loss: 0.5383, Train Accuracy: 0.7119, Val Loss: 0.5049, Val Accuracy: 0.7529

Epoch 33/150 - Training: 100%| | 519/519 [00:02<00:00, 226.58it/s]

Epoch 33/150 - Validation: 100%| | 112/112 [00:00<00:00, 830.02it/s]

Epoch 33/150, Train Loss: 0.5376, Train Accuracy: 0.7112, Val Loss: 0.5049, Val Accuracy: 0.7553

Epoch 34/150 - Training: 100%| | 519/519 [00:02<00:00, 228.18it/s]

Epoch 34/150 - Validation: 100%| | 112/112 [00:00<00:00, 792.47it/s]

Epoch 34/150, Train Loss: 0.5414, Train Accuracy: 0.7088, Val Loss: 0.5064, Val Accuracy: 0.7535

Epoch 35/150 - Training: 100%| | 519/519 [00:02<00:00, 224.88it/s]

Epoch 35/150 - Validation: 100%| | 112/112 [00:00<00:00, 786.44it/s]

Epoch 35/150, Train Loss: 0.5389, Train Accuracy: 0.7121, Val Loss: 0.5047, Val Accuracy: 0.7443

Epoch 36/150 - Training: 100%| | 519/519 [00:02<00:00, 234.38it/s]
Epoch 36/150 - Validation: 100%| | 112/112 [00:00<00:00, 784.32it/s]
Epoch 36/150, Train Loss: 0.5342, Train Accuracy: 0.7123, Val Loss: 0.5063, Val Accuracy: 0.7535

Epoch 37/150 - Training: 100%| | 519/519 [00:02<00:00, 228.31it/s]
Epoch 37/150 - Validation: 100%| | 112/112 [00:00<00:00, 776.29it/s]
Epoch 37/150, Train Loss: 0.5375, Train Accuracy: 0.7146, Val Loss: 0.5034, Val Accuracy: 0.7504

Epoch 38/150 - Training: 100%| | 519/519 [00:02<00:00, 234.82it/s]
Epoch 38/150 - Validation: 100%| | 112/112 [00:00<00:00, 810.55it/s]
Epoch 38/150, Train Loss: 0.5395, Train Accuracy: 0.7144, Val Loss: 0.5046, Val Accuracy: 0.7524

Epoch 39/150 - Training: 100%| | 519/519 [00:02<00:00, 234.55it/s]
Epoch 39/150 - Validation: 100%| | 112/112 [00:00<00:00, 798.12it/s]
Epoch 39/150, Train Loss: 0.5345, Train Accuracy: 0.7147, Val Loss: 0.5079, Val Accuracy: 0.7554

Epoch 40/150 - Training: 100%| | 519/519 [00:02<00:00, 236.81it/s]
Epoch 40/150 - Validation: 100%| | 112/112 [00:00<00:00, 831.16it/s]
Epoch 40/150, Train Loss: 0.5369, Train Accuracy: 0.7130, Val Loss: 0.5054, Val Accuracy: 0.7519

Epoch 41/150 - Training: 100%| | 519/519 [00:02<00:00, 233.12it/s]
Epoch 41/150 - Validation: 100%| | 112/112 [00:00<00:00, 829.48it/s]
Epoch 41/150, Train Loss: 0.5373, Train Accuracy: 0.7133, Val Loss: 0.5041, Val Accuracy: 0.7514

Epoch 42/150 - Training: 100%| | 519/519 [00:02<00:00, 235.13it/s]
Epoch 42/150 - Validation: 100%| | 112/112 [00:00<00:00, 813.29it/s]
Epoch 42/150, Train Loss: 0.5341, Train Accuracy: 0.7157, Val Loss: 0.5043, Val Accuracy: 0.7546

Epoch 43/150 - Training: 100%| | 519/519 [00:02<00:00, 221.11it/s]
Epoch 43/150 - Validation: 100%| | 112/112 [00:00<00:00, 816.91it/s]
Epoch 43/150, Train Loss: 0.5352, Train Accuracy: 0.7167, Val Loss: 0.5058, Val Accuracy: 0.7528

Epoch 44/150 - Training: 100%| | 519/519 [00:02<00:00, 234.18it/s]
Epoch 44/150 - Validation: 100%| | 112/112 [00:00<00:00, 839.76it/s]
Epoch 44/150, Train Loss: 0.5367, Train Accuracy: 0.7135, Val Loss: 0.5055, Val Accuracy: 0.7536

Epoch 45/150 - Training: 100%| | 519/519 [00:02<00:00, 235.51it/s]
Epoch 45/150 - Validation: 100%| | 112/112 [00:00<00:00, 798.36it/s]

Epoch 45/150, Train Loss: 0.5362, Train Accuracy: 0.7145, Val Loss: 0.5033, Val Accuracy: 0.7474

Epoch 46/150 - Training: 100%| | 519/519 [00:02<00:00, 222.54it/s]

Epoch 46/150 - Validation: 100%| | 112/112 [00:00<00:00, 690.31it/s]

Epoch 46/150, Train Loss: 0.5356, Train Accuracy: 0.7142, Val Loss: 0.5046, Val Accuracy: 0.7536

Epoch 47/150 - Training: 100%| | 519/519 [00:02<00:00, 235.10it/s]

Epoch 47/150 - Validation: 100%| | 112/112 [00:00<00:00, 821.05it/s]

Epoch 47/150, Train Loss: 0.5334, Train Accuracy: 0.7133, Val Loss: 0.5045, Val Accuracy: 0.7556

Epoch 48/150 - Training: 100%| | 519/519 [00:02<00:00, 236.38it/s]

Epoch 48/150 - Validation: 100%| | 112/112 [00:00<00:00, 825.68it/s]

Epoch 48/150, Train Loss: 0.5375, Train Accuracy: 0.7147, Val Loss: 0.5041, Val Accuracy: 0.7550

Epoch 49/150 - Training: 100%| | 519/519 [00:02<00:00, 232.04it/s]

Epoch 49/150 - Validation: 100%| | 112/112 [00:00<00:00, 780.97it/s]

Epoch 49/150, Train Loss: 0.5358, Train Accuracy: 0.7156, Val Loss: 0.5047, Val Accuracy: 0.7491

Epoch 50/150 - Training: 100%| | 519/519 [00:02<00:00, 224.70it/s]

Epoch 50/150 - Validation: 100%| | 112/112 [00:00<00:00, 809.64it/s]

Epoch 50/150, Train Loss: 0.5371, Train Accuracy: 0.7162, Val Loss: 0.5047, Val Accuracy: 0.7543

Epoch 51/150 - Training: 100%| | 519/519 [00:02<00:00, 229.25it/s]

Epoch 51/150 - Validation: 100%| | 112/112 [00:00<00:00, 836.48it/s]

Epoch 51/150, Train Loss: 0.5315, Train Accuracy: 0.7159, Val Loss: 0.5032, Val Accuracy: 0.7554

Epoch 52/150 - Training: 100%| | 519/519 [00:02<00:00, 226.99it/s]

Epoch 52/150 - Validation: 100%| | 112/112 [00:00<00:00, 812.39it/s]

Epoch 52/150, Train Loss: 0.5325, Train Accuracy: 0.7181, Val Loss: 0.5030, Val Accuracy: 0.7542

Epoch 53/150 - Training: 100%| | 519/519 [00:02<00:00, 215.61it/s]

Epoch 53/150 - Validation: 100%| | 112/112 [00:00<00:00, 813.11it/s]

Epoch 53/150, Train Loss: 0.5340, Train Accuracy: 0.7176, Val Loss: 0.5047, Val Accuracy: 0.7554

Epoch 54/150 - Training: 100%| | 519/519 [00:02<00:00, 230.22it/s]

Epoch 54/150 - Validation: 100%| | 112/112 [00:00<00:00, 728.93it/s]

Epoch 54/150, Train Loss: 0.5336, Train Accuracy: 0.7185, Val Loss: 0.5033, Val Accuracy: 0.7553

Epoch 55/150 - Training: 100%| | 519/519 [00:02<00:00, 233.73it/s]
Epoch 55/150 - Validation: 100%| | 112/112 [00:00<00:00, 836.17it/s]
Epoch 55/150, Train Loss: 0.5363, Train Accuracy: 0.7148, Val Loss: 0.5038, Val Accuracy: 0.7545

Epoch 56/150 - Training: 100%| | 519/519 [00:02<00:00, 229.13it/s]
Epoch 56/150 - Validation: 100%| | 112/112 [00:00<00:00, 731.25it/s]
Epoch 56/150, Train Loss: 0.5363, Train Accuracy: 0.7160, Val Loss: 0.5020, Val Accuracy: 0.7514

Epoch 57/150 - Training: 100%| | 519/519 [00:02<00:00, 221.91it/s]
Epoch 57/150 - Validation: 100%| | 112/112 [00:00<00:00, 810.67it/s]
Epoch 57/150, Train Loss: 0.5337, Train Accuracy: 0.7173, Val Loss: 0.5027, Val Accuracy: 0.7542

Epoch 58/150 - Training: 100%| | 519/519 [00:02<00:00, 218.35it/s]
Epoch 58/150 - Validation: 100%| | 112/112 [00:00<00:00, 821.03it/s]
Epoch 58/150, Train Loss: 0.5366, Train Accuracy: 0.7152, Val Loss: 0.5047, Val Accuracy: 0.7550

Epoch 59/150 - Training: 100%| | 519/519 [00:02<00:00, 231.87it/s]
Epoch 59/150 - Validation: 100%| | 112/112 [00:00<00:00, 839.85it/s]
Epoch 59/150, Train Loss: 0.5322, Train Accuracy: 0.7160, Val Loss: 0.5036, Val Accuracy: 0.7545

Epoch 60/150 - Training: 100%| | 519/519 [00:02<00:00, 231.39it/s]
Epoch 60/150 - Validation: 100%| | 112/112 [00:00<00:00, 817.33it/s]
Epoch 60/150, Train Loss: 0.5339, Train Accuracy: 0.7176, Val Loss: 0.5040, Val Accuracy: 0.7507

Epoch 61/150 - Training: 100%| | 519/519 [00:02<00:00, 232.00it/s]
Epoch 61/150 - Validation: 100%| | 112/112 [00:00<00:00, 726.88it/s]
Epoch 61/150, Train Loss: 0.5363, Train Accuracy: 0.7171, Val Loss: 0.5028, Val Accuracy: 0.7522

Epoch 62/150 - Training: 100%| | 519/519 [00:02<00:00, 234.90it/s]
Epoch 62/150 - Validation: 100%| | 112/112 [00:00<00:00, 795.15it/s]
Epoch 62/150, Train Loss: 0.5330, Train Accuracy: 0.7161, Val Loss: 0.5025, Val Accuracy: 0.7501

Epoch 63/150 - Training: 100%| | 519/519 [00:02<00:00, 224.35it/s]
Epoch 63/150 - Validation: 100%| | 112/112 [00:00<00:00, 689.14it/s]
Epoch 63/150, Train Loss: 0.5348, Train Accuracy: 0.7160, Val Loss: 0.5027, Val Accuracy: 0.7546

Epoch 64/150 - Training: 100%| | 519/519 [00:02<00:00, 210.45it/s]
Epoch 64/150 - Validation: 100%| | 112/112 [00:00<00:00, 780.98it/s]

Epoch 64/150, Train Loss: 0.5341, Train Accuracy: 0.7177, Val Loss: 0.5019, Val Accuracy: 0.7494

Epoch 65/150 - Training: 100%| | 519/519 [00:02<00:00, 226.85it/s]

Epoch 65/150 - Validation: 100%| | 112/112 [00:00<00:00, 709.29it/s]

Epoch 65/150, Train Loss: 0.5325, Train Accuracy: 0.7176, Val Loss: 0.5038, Val Accuracy: 0.7518

Epoch 66/150 - Training: 100%| | 519/519 [00:02<00:00, 223.21it/s]

Epoch 66/150 - Validation: 100%| | 112/112 [00:00<00:00, 788.31it/s]

Epoch 66/150, Train Loss: 0.5350, Train Accuracy: 0.7188, Val Loss: 0.5024, Val Accuracy: 0.7493

Epoch 67/150 - Training: 100%| | 519/519 [00:02<00:00, 229.14it/s]

Epoch 67/150 - Validation: 100%| | 112/112 [00:00<00:00, 763.16it/s]

Epoch 67/150, Train Loss: 0.5321, Train Accuracy: 0.7217, Val Loss: 0.5022, Val Accuracy: 0.7495

Epoch 68/150 - Training: 100%| | 519/519 [00:02<00:00, 232.34it/s]

Epoch 68/150 - Validation: 100%| | 112/112 [00:00<00:00, 814.31it/s]

Epoch 68/150, Train Loss: 0.5320, Train Accuracy: 0.7197, Val Loss: 0.5038, Val Accuracy: 0.7514

Epoch 69/150 - Training: 100%| | 519/519 [00:02<00:00, 235.84it/s]

Epoch 69/150 - Validation: 100%| | 112/112 [00:00<00:00, 786.85it/s]

Epoch 69/150, Train Loss: 0.5337, Train Accuracy: 0.7192, Val Loss: 0.5028, Val Accuracy: 0.7545

Epoch 70/150 - Training: 100%| | 519/519 [00:02<00:00, 231.92it/s]

Epoch 70/150 - Validation: 100%| | 112/112 [00:00<00:00, 729.15it/s]

Epoch 70/150, Train Loss: 0.5353, Train Accuracy: 0.7196, Val Loss: 0.5044, Val Accuracy: 0.7552

Epoch 71/150 - Training: 100%| | 519/519 [00:02<00:00, 220.06it/s]

Epoch 71/150 - Validation: 100%| | 112/112 [00:00<00:00, 747.06it/s]

Epoch 71/150, Train Loss: 0.5345, Train Accuracy: 0.7191, Val Loss: 0.5018, Val Accuracy: 0.7531

Epoch 72/150 - Training: 100%| | 519/519 [00:02<00:00, 235.67it/s]

Epoch 72/150 - Validation: 100%| | 112/112 [00:00<00:00, 819.68it/s]

Epoch 72/150, Train Loss: 0.5337, Train Accuracy: 0.7201, Val Loss: 0.5021, Val Accuracy: 0.7526

Epoch 73/150 - Training: 100%| | 519/519 [00:02<00:00, 230.10it/s]

Epoch 73/150 - Validation: 100%| | 112/112 [00:00<00:00, 770.46it/s]

Epoch 73/150, Train Loss: 0.5338, Train Accuracy: 0.7183, Val Loss: 0.5024, Val Accuracy: 0.7522

Epoch 74/150 - Training: 100%| | 519/519 [00:02<00:00, 228.89it/s]
Epoch 74/150 - Validation: 100%| | 112/112 [00:00<00:00, 831.08it/s]
Epoch 74/150, Train Loss: 0.5351, Train Accuracy: 0.7207, Val Loss: 0.5017, Val Accuracy: 0.7531

Epoch 75/150 - Training: 100%| | 519/519 [00:02<00:00, 234.06it/s]
Epoch 75/150 - Validation: 100%| | 112/112 [00:00<00:00, 784.63it/s]
Epoch 75/150, Train Loss: 0.5340, Train Accuracy: 0.7173, Val Loss: 0.5030, Val Accuracy: 0.7543

Epoch 76/150 - Training: 100%| | 519/519 [00:02<00:00, 227.95it/s]
Epoch 76/150 - Validation: 100%| | 112/112 [00:00<00:00, 795.16it/s]
Epoch 76/150, Train Loss: 0.5326, Train Accuracy: 0.7208, Val Loss: 0.5037, Val Accuracy: 0.7531

Epoch 77/150 - Training: 100%| | 519/519 [00:02<00:00, 229.99it/s]
Epoch 77/150 - Validation: 100%| | 112/112 [00:00<00:00, 823.53it/s]
Epoch 77/150, Train Loss: 0.5307, Train Accuracy: 0.7224, Val Loss: 0.5027, Val Accuracy: 0.7529

Epoch 78/150 - Training: 100%| | 519/519 [00:02<00:00, 229.65it/s]
Epoch 78/150 - Validation: 100%| | 112/112 [00:00<00:00, 791.74it/s]
Epoch 78/150, Train Loss: 0.5339, Train Accuracy: 0.7179, Val Loss: 0.5021, Val Accuracy: 0.7543

Epoch 79/150 - Training: 100%| | 519/519 [00:02<00:00, 233.40it/s]
Epoch 79/150 - Validation: 100%| | 112/112 [00:00<00:00, 786.53it/s]
Epoch 79/150, Train Loss: 0.5328, Train Accuracy: 0.7176, Val Loss: 0.5025, Val Accuracy: 0.7524

Epoch 80/150 - Training: 100%| | 519/519 [00:02<00:00, 234.53it/s]
Epoch 80/150 - Validation: 100%| | 112/112 [00:00<00:00, 808.83it/s]
Epoch 80/150, Train Loss: 0.5331, Train Accuracy: 0.7196, Val Loss: 0.5026, Val Accuracy: 0.7543

Epoch 81/150 - Training: 100%| | 519/519 [00:02<00:00, 217.79it/s]
Epoch 81/150 - Validation: 100%| | 112/112 [00:00<00:00, 801.56it/s]
Epoch 81/150, Train Loss: 0.5328, Train Accuracy: 0.7198, Val Loss: 0.5013, Val Accuracy: 0.7481

Epoch 82/150 - Training: 100%| | 519/519 [00:02<00:00, 228.35it/s]
Epoch 82/150 - Validation: 100%| | 112/112 [00:00<00:00, 809.03it/s]
Epoch 82/150, Train Loss: 0.5326, Train Accuracy: 0.7196, Val Loss: 0.5019, Val Accuracy: 0.7532

Epoch 83/150 - Training: 100%| | 519/519 [00:02<00:00, 224.20it/s]
Epoch 83/150 - Validation: 100%| | 112/112 [00:00<00:00, 819.41it/s]

Epoch 83/150, Train Loss: 0.5301, Train Accuracy: 0.7217, Val Loss: 0.5032, Val Accuracy: 0.7495

Epoch 84/150 - Training: 100%| | 519/519 [00:02<00:00, 235.86it/s]

Epoch 84/150 - Validation: 100%| | 112/112 [00:00<00:00, 821.76it/s]

Epoch 84/150, Train Loss: 0.5325, Train Accuracy: 0.7189, Val Loss: 0.5031, Val Accuracy: 0.7526

Epoch 85/150 - Training: 100%| | 519/519 [00:02<00:00, 233.72it/s]

Epoch 85/150 - Validation: 100%| | 112/112 [00:00<00:00, 818.00it/s]

Epoch 85/150, Train Loss: 0.5331, Train Accuracy: 0.7173, Val Loss: 0.5024, Val Accuracy: 0.7533

Epoch 86/150 - Training: 100%| | 519/519 [00:02<00:00, 228.31it/s]

Epoch 86/150 - Validation: 100%| | 112/112 [00:00<00:00, 821.90it/s]

Epoch 86/150, Train Loss: 0.5332, Train Accuracy: 0.7166, Val Loss: 0.5053, Val Accuracy: 0.7540

Epoch 87/150 - Training: 100%| | 519/519 [00:02<00:00, 236.08it/s]

Epoch 87/150 - Validation: 100%| | 112/112 [00:00<00:00, 826.54it/s]

Epoch 87/150, Train Loss: 0.5310, Train Accuracy: 0.7193, Val Loss: 0.5017, Val Accuracy: 0.7518

Epoch 88/150 - Training: 100%| | 519/519 [00:02<00:00, 222.33it/s]

Epoch 88/150 - Validation: 100%| | 112/112 [00:00<00:00, 809.67it/s]

Epoch 88/150, Train Loss: 0.5324, Train Accuracy: 0.7190, Val Loss: 0.5016, Val Accuracy: 0.7498

Epoch 89/150 - Training: 100%| | 519/519 [00:02<00:00, 229.80it/s]

Epoch 89/150 - Validation: 100%| | 112/112 [00:00<00:00, 781.11it/s]

Epoch 89/150, Train Loss: 0.5329, Train Accuracy: 0.7182, Val Loss: 0.5018, Val Accuracy: 0.7493

Epoch 90/150 - Training: 100%| | 519/519 [00:02<00:00, 219.74it/s]

Epoch 90/150 - Validation: 100%| | 112/112 [00:00<00:00, 763.41it/s]

Epoch 90/150, Train Loss: 0.5332, Train Accuracy: 0.7167, Val Loss: 0.5018, Val Accuracy: 0.7532

Epoch 91/150 - Training: 100%| | 519/519 [00:02<00:00, 230.75it/s]

Epoch 91/150 - Validation: 100%| | 112/112 [00:00<00:00, 805.92it/s]

Epoch 91/150, Train Loss: 0.5309, Train Accuracy: 0.7201, Val Loss: 0.5032, Val Accuracy: 0.7538

Early stopping

Testing: 100%| | 112/112 [00:00<00:00, 771.19it/s]

Test Loss: 0.4986, Test Accuracy: 0.7551

Raport klasyfikacji zawierający metryki różne metryki oceny modelu tj: precyzja: Dokładność w przewidywaniu pozytywnych klas czułość: Zdolność wykrywania pozytywnych klas f1-score: Harmonijna średnia precyzji i czułości support: Liczba próbek w zbiorze testowym dokładność: Procentowy udział poprawnie sklasyfikowanych próbek

```
[64]: import os
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

model = torch.load(os.getcwd() + '/../backend/Models/predict_1.pth')
model.eval()

true_labels = []
predictions = []

with torch.no_grad():
    for inputs, labels in testloader:
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)

        predicted = (outputs > 0.5).float()

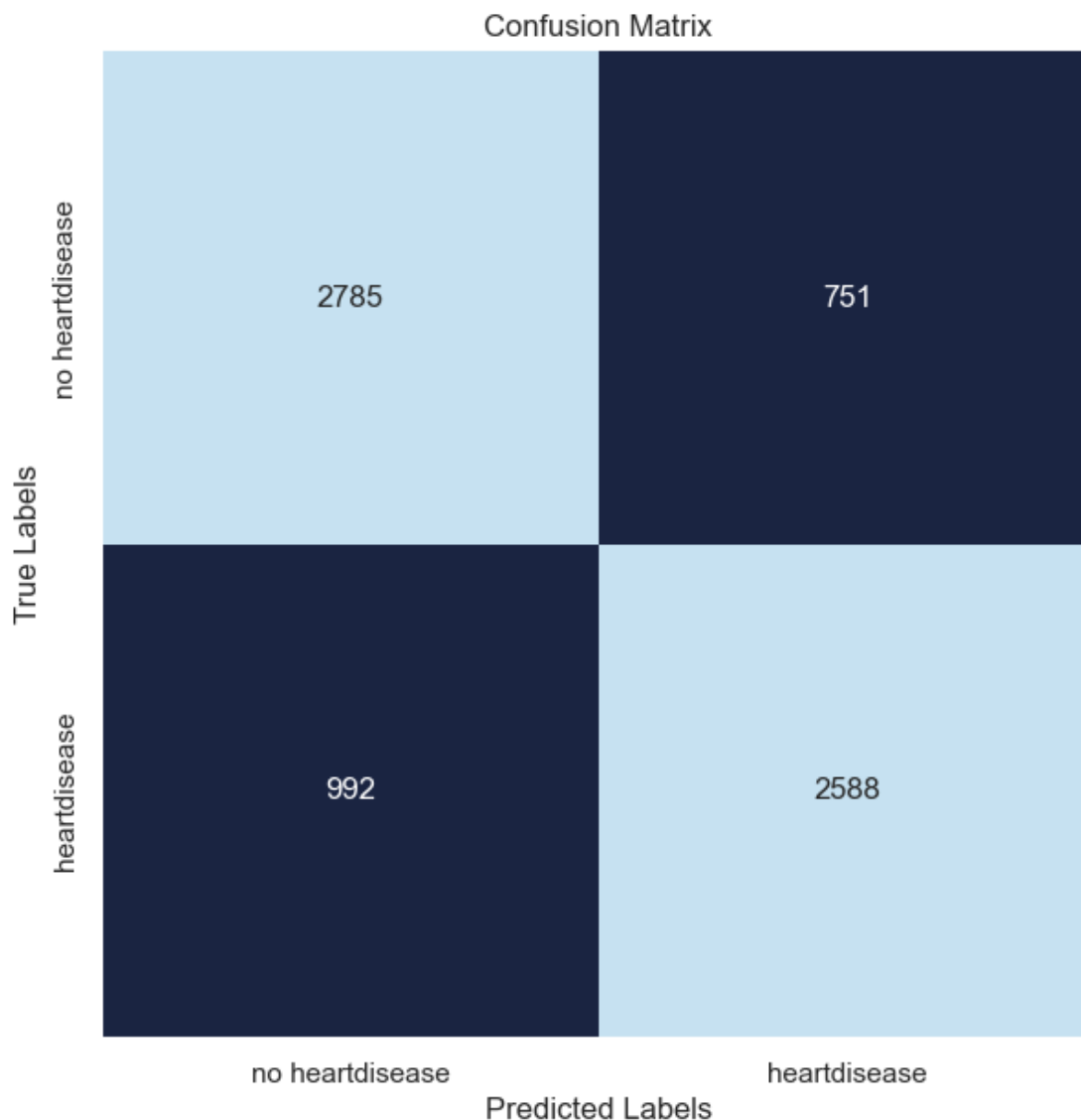
        true_labels.extend(labels.cpu().numpy())
        predictions.extend(predicted.cpu().numpy())

true_labels = np.array(true_labels).reshape(-1)
predictions = np.array(predictions).reshape(-1)

cm = confusion_matrix(true_labels, predictions)

plt.figure(figsize=(7, 7))
sns.heatmap(cm, annot=True, fmt='d', cbar=False, cmap=sns.
    color_palette("cubehelix"),
            xticklabels=['no heartdisease', 'heartdisease'],
            yticklabels=['no heartdisease', 'heartdisease'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

classification_report_test = classification_report(np.array(test_true_labels),
    np.array(test_predictions))
print(classification_report_test)
```



```
{'0.0': {'precision': 0.7373576912893831, 'recall': 0.7876131221719457,
'f1-score': 0.7616573225762342, 'support': 3536}, '1.0': {'precision':
0.7750823599880203, 'recall': 0.7229050279329609, 'f1-score': 0.748084983379101,
'support': 3580}, 'accuracy': 0.7550590219224284, 'macro avg': {'precision':
0.7562200256387017, 'recall': 0.7552590750524533, 'f1-score':
0.7548711529776676, 'support': 7116}, 'weighted avg': {'precision':
0.7563366561490124, 'recall': 0.7550590219224284, 'f1-score':
0.7548291924011727, 'support': 7116}}
```

Nawiązanie połączenia z bazą i zapis parametrów modelu [Pamiętaj by skonfigurować połączenie ustawiając username,password i dodając adres IP swojego hosta]:

```
[65]: from pymongo import MongoClient
from datetime import datetime

username = os.getenv('MONGO_USERNAME')
password = os.getenv('MONGO_PASSWORD')

connection = f"mongodb+srv://{username}:{password}@medicalprediction.ow3dfwa.
↳mongodb.net/"
client = MongoClient(connection)

db = client["Medical_prediction"]
collection = db['Data']

document = {
    "name": "Heartdisease_prediction",
    "precision": [round(classification_report_test["0.0"]["precision"], 2),
↳round(classification_report_test["1.0"]["precision"], 2)],
    "recall": [round(classification_report_test["0.0"]["recall"], 2),
↳round(classification_report_test["1.0"]["recall"], 2)],
    "accuracy": round(test_accuracies[-1], 2),
    "loss": round(test_losses[-1], 2),
    "date_inserted": datetime.now(),
    "train_samples": len(trainloader.dataset),
    "val_samples": len(valloader.dataset),
    "test_samples": len(testloader.dataset)
}
collection.insert_one(document)
```

[65]: <pymongo.results.InsertOneResult at 0x138c7cd00>

Zdefiniowanie odpowiednich zmiennych na potrzeby obliczeń przeprowadzanych przez backend aplikacji:

```
[ ]: import os

train_samples = len(trainloader.dataset)
val_samples = len(valloader.dataset)
test_samples = len(testloader.dataset)

env_path = '../backend/.env'

if os.path.exists(env_path):
    with open(env_path, 'r') as f:
        env_vars = dict(line.strip().split('=') for line in f if line.strip())
else:
    env_vars = {}
```

```

env_vars['HEARTDISEASE_TRAIN'] = train_samples
env_vars['HEARTDISEASE_VAL'] = val_samples
env_vars['HEARTDISEASE_TEST'] = test_samples

with open(env_path, 'w') as f:
    f.writelines(f'{k}={v}\n' for k, v in env_vars.items())

```

Wykres dla `train_accuracy` i `validation_accuracy` pokazuje szybki wzrost dokładności modelu zarówno na zbiorze treningowym, jak i testowym w kolejnych epokach - zakończony stabilizacją na poziomie 75 %

```

[75]: import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))

plt.plot(range(1, len(train_accuracies) + 1), train_accuracies, label='Train_
↳Accuracy')

plt.plot(range(1, len(val_accuracies) + 1), val_accuracies, label='Validation_
↳Accuracy')

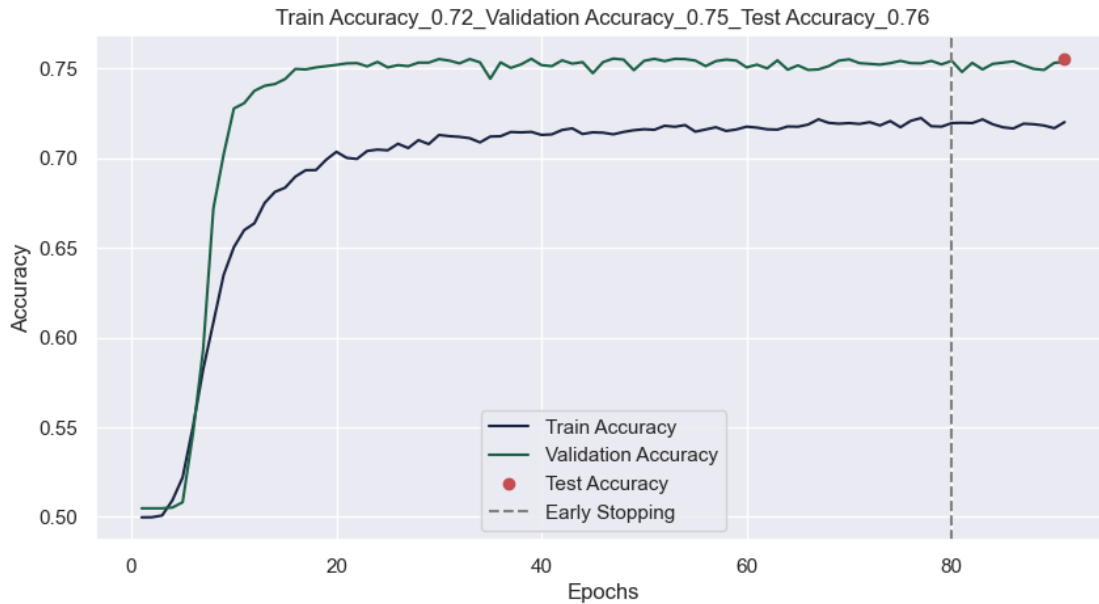
plt.plot([len(train_accuracies)], [test_accuracy], 'ro', label='Test Accuracy')

stopping_epoch = len(train_accuracies) - patience
if stopping_epoch > 0:
    plt.axvline(x=stopping_epoch, color='gray', linestyle='--', label=f'Early_
↳Stopping')

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title(f"Train Accuracy_{train_accuracies[-1]:.2f}_Validation_
↳Accuracy_{val_accuracies[-1]:.2f}_Test Accuracy_{test_accuracy:.2f}")

plt.show()

```



Wykres dla `loss_train` i `loss_validation` ukazuje ich wyraźny trend spadkowy. Niewielkie oscylacje tej metryki spowodowane są m.in. zastosowaniem techniki dropout.

```
[76]: import matplotlib.pyplot as plt

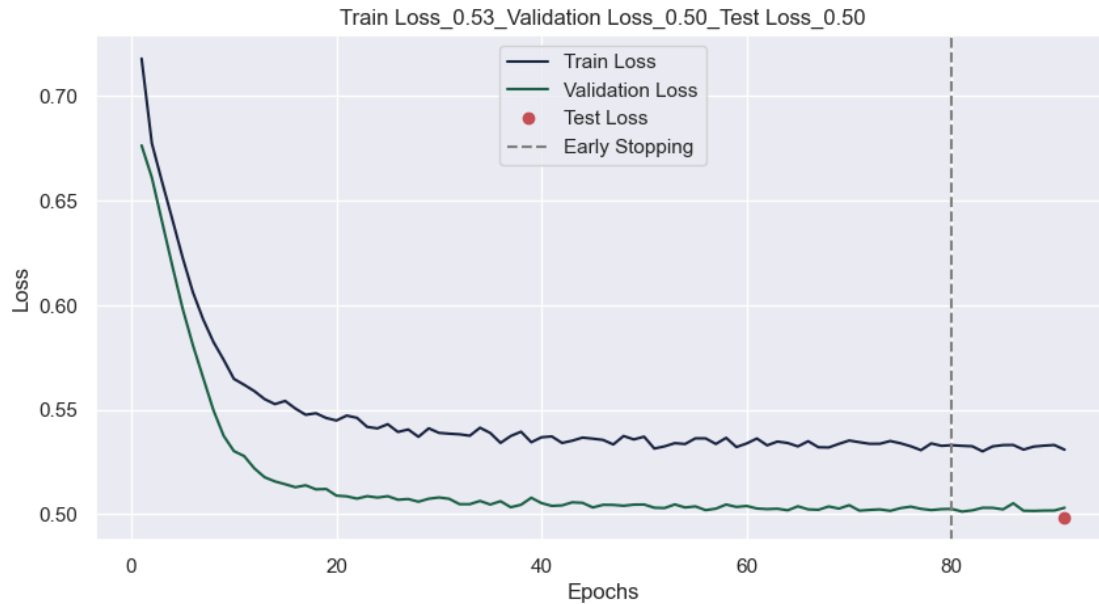
plt.figure(figsize=(10, 5))

plt.plot(range(1, len(train_losses) + 1), train_losses, label='Train Loss')
plt.plot(range(1, len(val_losses) + 1), val_losses, label='Validation Loss')
plt.plot([len(train_losses)], [test_losses[-1]], 'ro', label='Test Loss')

stopping_epoch = len(train_losses) - patience
if stopping_epoch > 0:
    plt.axvline(x=stopping_epoch, color='gray', linestyle='--', label=f'Early Stopping')

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title(f"Train Loss_{train_losses[-1]:.2f}_Validation Loss_{val_losses[-1]:.2f}_Test Loss_{test_losses[-1]:.2f}")

plt.show()
```



Normalizacja rozkładu prawdopodobieństwa modelu, przy wykorzystaniu transformacji kwantylowej:

```
[63]: import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import QuantileTransformer
import torch
import pandas as pd
import scipy.stats as stats

inputs_copy = torch.tensor(pd.DataFrame(X_test).values).float().to(device)

with torch.no_grad():
    outputs = model(inputs_copy)
    probabilities = torch.sigmoid(outputs)

probabilities = probabilities.cpu().numpy().flatten()

transformer = QuantileTransformer(output_distribution='normal')
transformed_data = transformer.fit_transform(probabilities.reshape(-1, 1)).
    ↪flatten()

plt.figure(figsize=(15, 12))

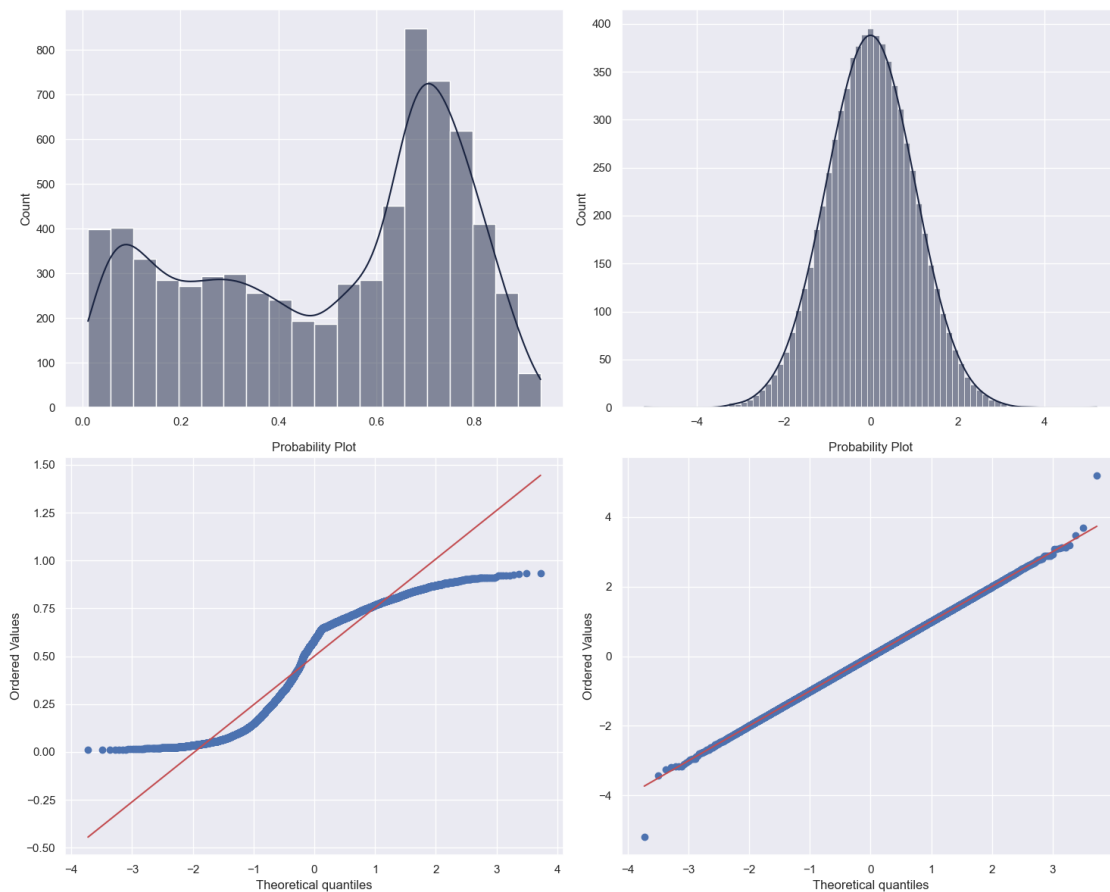
plt.subplot(2, 2, 1)
sns.histplot(probabilities, kde=True)
```

```
plt.subplot(2, 2, 2)
sns.histplot(transformed_data, kde=True)

plt.subplot(2, 2, 3)
stats.probplot(probabilities, dist="norm", plot=plt)

plt.subplot(2, 2, 4)
stats.probplot(transformed_data, dist="norm", plot=plt)

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



Wybranie 10 pacjentów z zbioru testowego (połowa z nich będzie chora, a druga połowa zdrowa) oraz sprawdzenie jakie prawdopodobieństwo występowania choroby wraz z określonymi przedziałami ufności przypisze im wytrenowany model:

```
[81]: import numpy as np
import pandas as pd
import torch
```



```

import os
from sklearn.preprocessing import QuantileTransformer

model = torch.load(os.getcwd() + '/../backend/Models/predict_1.pth')
model.eval()

df_XY = pd.DataFrame(X_test)
df_XY['HeartDiseaseorAttack'] = y_test

inputs = torch.tensor(df_XY.drop('HeartDiseaseorAttack', axis=1).values).
    ↪float().to(device)

with torch.no_grad():
    outputs = model(inputs)
    probabilities = torch.sigmoid(outputs)

transformer = QuantileTransformer(n_quantiles=10, output_distribution='normal')
transformed_data = transformer.fit_transform(probabilities.cpu().numpy().
    ↪reshape(-1, 1)).flatten()

z = 1.96
std_errors_transformed = z * np.std(transformed_data) / np.sqrt(len(df_XY))

lower_bounds_transformed = transformed_data - std_errors_transformed
upper_bounds_transformed = transformed_data + std_errors_transformed

lower_bounds = transformer.inverse_transform(lower_bounds_transformed.
    ↪reshape(-1, 1)).flatten()
upper_bounds = transformer.inverse_transform(upper_bounds_transformed.
    ↪reshape(-1, 1)).flatten()

df_XY['P(A)'] = probabilities.cpu().numpy().flatten() * 100
df_XY['Lower Bound P(A)'] = lower_bounds * 100
df_XY['Upper Bound P(A)'] = upper_bounds * 100

df_XY['Lower Bound P(A)'] = df_XY['Lower Bound P(A)'].clip(lower=0, upper=100)
df_XY['Upper Bound P(A)'] = df_XY['Upper Bound P(A)'].clip(lower=0, upper=100)

df_XY['~P(A)'] = 100 - df_XY['P(A)']
df_XY['Lower Bound ~P(A)'] = 100 - df_XY['Upper Bound P(A)']
df_XY['Upper Bound ~P(A)'] = 100 - df_XY['Lower Bound P(A)']

df_XY['Lower Bound ~P(A)'] = df_XY['Lower Bound ~P(A)'].clip(lower=0, upper=100)
df_XY['Upper Bound ~P(A)'] = df_XY['Upper Bound ~P(A)'].clip(lower=0, upper=100)

samples_with_sick = df_XY[df_XY['HeartDiseaseorAttack'] == 1].sample(5,
    ↪random_state=40)

```

```

samples_without_sick = df_XY[df_XY['HeartDiseaseorAttack'] == 0].sample(5,
↳random_state=40)
samples = pd.concat([samples_with_sick, samples_without_sick])

print(samples[['P(A)', 'Lower Bound P(A)', 'Upper Bound P(A)', '~P(A)', 'Lower_
↳Bound ~P(A)', 'Upper Bound ~P(A)']])

```

	P(A)	Lower Bound P(A)	Upper Bound P(A)	~P(A)	\
3698	80.448540	79.944099	80.939163	19.551460	
2617	81.369118	80.890808	81.833908	18.630882	
2435	72.556839	72.262802	72.846527	27.443161	
6204	85.732742	85.391327	86.062798	14.267258	
848	14.922960	14.357180	15.501513	85.077042	
1616	59.592457	58.591885	60.592426	40.407543	
5207	10.864134	10.396025	11.344588	89.135864	
3389	71.744781	71.439270	72.046158	28.255219	
2367	40.962997	39.708118	42.226822	59.037003	
3341	27.856789	27.054478	28.669933	72.143211	

	Lower Bound ~P(A)	Upper Bound ~P(A)
3698	19.060837	20.055901
2617	18.166092	19.109192
2435	27.153473	27.737198
6204	13.937202	14.608673
848	84.498489	85.642822
1616	39.407574	41.408115
5207	88.655411	89.603973
3389	27.953842	28.560730
2367	57.773178	60.291882
3341	71.330063	72.945526

Następnie tak przygotowany zbiór danych możemy zwizualizować w postaci wykresu słupkowego z przedziałami ufności dla każdego pacjenta:

```

[96]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

samples_copy = samples.copy()
samples_copy['Condition'] = ["HeartDiseaseorAttack"] * 5 + ["No_
↳HeartDiseaseorAttack"] * 5
samples_copy = samples_copy.sort_values(by='P(A)', ascending=False)

P_A = samples_copy['P(A)']
lower_bound_A = samples_copy['Lower Bound P(A)']
upper_bound_A = samples_copy['Upper Bound P(A)']
conditions = samples_copy['Condition']

```

```

labels = samples_copy.index

cubehelix_colors = sns.color_palette("cubehelix", 2)
color_map = {'HeartDiseaseorAttack': cubehelix_colors[0], 'No_HeartDiseaseorAttack': cubehelix_colors[1]}
colors = [color_map[condition] for condition in conditions]

y_pos = np.arange(len(labels))

errors_A = [np.abs(np.array(P_A) - np.array(lower_bound_A)),
            np.abs(np.array(upper_bound_A) - np.array(P_A))]

plt.figure(figsize=(12, 8))
bars = plt.barh(y_pos, P_A, xerr=errors_A, align='center', color=colors,
               capsize=5)

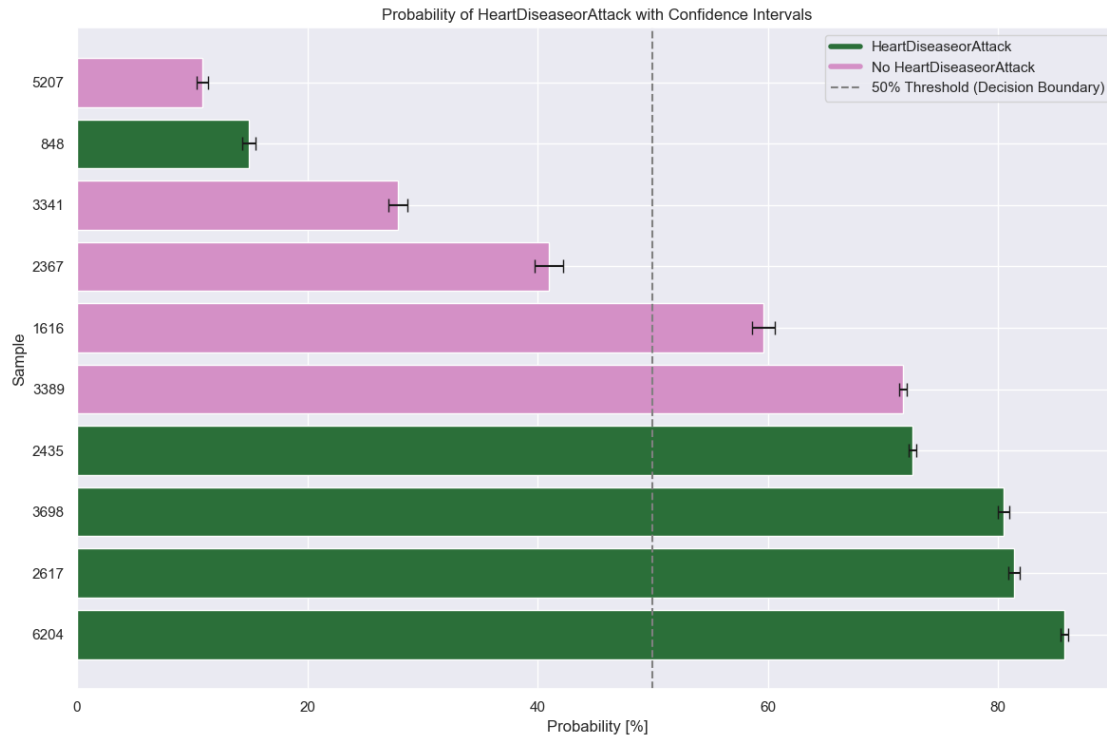
plt.yticks(y_pos, labels)
plt.xlabel('Probability [%]')
plt.ylabel('Sample')
plt.title('Probability of HeartDiseaseorAttack with Confidence Intervals')

plt.legend(handles=[
    plt.Line2D([0], [0], color=cubehelix_colors[0], lw=4,
               label='HeartDiseaseorAttack'),
    plt.Line2D([0], [0], color=cubehelix_colors[1], lw=4, label='No_HeartDiseaseorAttack'),
    plt.Line2D([0], [0], color='gray', linestyle='--', linewidth=1.5,
               label='50% Threshold (Decision Boundary)')
], loc='upper right')

plt.axvline(x=50, color='gray', linestyle='--', linewidth=1.5)

plt.tight_layout()
plt.show()

```



```
[73]: features = [
    "HighBP",
    "HighChol",
    "CholCheck",
    "BMI",
    "Smoker",
    "Stroke",
    "Diabetes",
    "PhysActivity",
    "Fruits",
    "Veggies",
    "HvyAlcoholConsump",
    "AnyHealthcare",
    "NoDocbcCost",
    "GenHlth",
    "MentHlth",
    "PhysHlth",
    "DiffWalk",
    "Sex",
    "Age",
    "Education",
    "Income",
]
```

```
for i, feature in enumerate(features):  
    print(f"{i} - {feature}")
```

```
0 - HighBP  
1 - HighChol  
2 - CholCheck  
3 - BMI  
4 - Smoker  
5 - Stroke  
6 - Diabetes  
7 - PhysActivity  
8 - Fruits  
9 - Veggies  
10 - HvyAlcoholConsump  
11 - AnyHealthcare  
12 - NoDocbcCost  
13 - GenHlth  
14 - MentHlth  
15 - PhysHlth  
16 - DiffWalk  
17 - Sex  
18 - Age  
19 - Education  
20 - Income
```

```
[93]: print(samples.loc[848])  
      print(samples.loc[1616])  
      print(samples.loc[3389])
```

```
0          0.000000  
1          0.000000  
2          1.000000  
3         27.000000  
4          1.000000  
5          0.000000  
6          2.000000  
7          1.000000  
8          0.000000  
9          1.000000  
10         0.000000  
11         0.000000  
12         0.000000  
13         2.000000  
14         2.000000  
15         2.000000  
16         0.000000  
17         0.000000
```

18	6.000000
19	4.000000
20	3.000000
HeartDiseaseorAttack	1.000000
P(A)	14.922960
Lower Bound P(A)	14.357180
Upper Bound P(A)	15.501513
~P(A)	85.077042
Lower Bound ~P(A)	84.498489
Upper Bound ~P(A)	85.642822
Name: 848, dtype: float32	
0	1.000000
1	0.000000
2	1.000000
3	28.000000
4	1.000000
5	0.000000
6	0.000000
7	1.000000
8	1.000000
9	1.000000
10	0.000000
11	1.000000
12	0.000000
13	3.000000
14	0.000000
15	7.000000
16	1.000000
17	0.000000
18	11.000000
19	4.000000
20	7.000000
HeartDiseaseorAttack	0.000000
P(A)	59.592457
Lower Bound P(A)	58.591885
Upper Bound P(A)	60.592426
~P(A)	40.407543
Lower Bound ~P(A)	39.407574
Upper Bound ~P(A)	41.408115
Name: 1616, dtype: float32	
0	1.000000
1	1.000000
2	1.000000
3	31.000000
4	1.000000
5	0.000000
6	2.000000
7	0.000000

```

8          1.000000
9          1.000000
10         0.000000
11         1.000000
12         0.000000
13         3.000000
14         0.000000
15         0.000000
16         0.000000
17         0.000000
18        12.000000
19         5.000000
20         2.000000
HeartDiseaseorAttack 0.000000
P(A)                71.744781
Lower Bound P(A)    71.439270
Upper Bound P(A)    72.046158
~P(A)               28.255219
Lower Bound ~P(A)   27.953842
Upper Bound ~P(A)   28.560730
Name: 3389, dtype: float32

```

```

[66]: print(" P(A) -> HeartDiseaseorAttack [%] \n~P(A) -> No HeartDiseaseorAttack_
      ↪ [%]\n")
print(samples)

```

```

P(A) -> HeartDiseaseorAttack [%]
~P(A) -> No HeartDiseaseorAttack [%]

      0    1    2    3    4    5    6    7    8    9  ...   18   19   20  \
2795  1.0  1.0  1.0  41.0  0.0  0.0  2.0  1.0  1.0  1.0  ...   9.0  6.0  6.0
852   1.0  1.0  1.0  57.0  0.0  0.0  2.0  0.0  0.0  0.0  ...   9.0  4.0  4.0
1622  1.0  0.0  1.0  22.0  0.0  0.0  0.0  1.0  1.0  1.0  ...   9.0  4.0  6.0
6016  1.0  1.0  1.0  22.0  0.0  0.0  0.0  1.0  1.0  1.0  ...  11.0  4.0  3.0
3068  1.0  1.0  1.0  28.0  0.0  0.0  2.0  1.0  0.0  1.0  ...  13.0  6.0  8.0
5827  0.0  0.0  1.0  22.0  1.0  0.0  2.0  1.0  1.0  1.0  ...   7.0  6.0  7.0
525   1.0  1.0  1.0  30.0  0.0  0.0  0.0  1.0  0.0  0.0  ...   5.0  6.0  7.0
1573  0.0  1.0  1.0  27.0  0.0  0.0  0.0  1.0  0.0  1.0  ...  11.0  6.0  6.0
2508  0.0  0.0  1.0  19.0  0.0  0.0  0.0  1.0  1.0  1.0  ...   8.0  5.0  7.0
5555  0.0  1.0  1.0  24.0  0.0  0.0  1.0  1.0  1.0  1.0  ...  11.0  5.0  6.0

      HeartDiseaseorAttack      P(A)  Lower Bound P(A)  Upper Bound P(A)  \
2795                    1.0  68.227791          67.918556          68.534462
852                     1.0  69.462212          69.129982          69.791069
1622                    1.0  66.334496          66.012627          66.654602
6016                    1.0  74.254539          73.933304          74.570152
3068                    1.0  74.485054          74.167908          74.796547
5827                    0.0  11.489424          11.005291          11.986012

```

525	0.0	22.089008	21.382669	22.810202
1573	0.0	30.798288	29.958586	31.647890
2508	0.0	15.789472	15.204658	16.387062
5555	0.0	36.514099	35.295513	37.743465

	$\sim P(A)$	Lower Bound	$\sim P(A)$	Upper Bound	$\sim P(A)$
2795	31.772209		31.465538		32.081444
852	30.537788		30.208931		30.870018
1622	33.665504		33.345398		33.987373
6016	25.745461		25.429848		26.066696
3068	25.514946		25.203453		25.832092
5827	88.510574		88.013985		88.994705
525	77.910995		77.189796		78.617332
1573	69.201714		68.352112		70.041412
2508	84.210526		83.612938		84.795341
5555	63.485901		62.256535		64.704483

[10 rows x 28 columns]

Przykładowa przebadana przeze mnie osoba:

```
[67]: import os
import pandas as pd
import numpy as np
import torch

person = {
    "HighBP": 0,
    "HighChol": 0,
    "CholCheck": 1,
    "BMI": 22.92,
    "Smoker": 0,
    "Stroke": 0,
    "Diabetes": 0,
    "PhysActivity": 0,
    "Fruits": 1,
    "Veggies": 0,
    "HvyAlcoholConsump": 0,
    "AnyHealthcare": 1,
    "NoDocbcCost": 0,
    "GenHlth": 3,
    "MentHlth": 15,
    "PhysHlth": 0,
    "DiffWalk": 0,
    "Sex": 0,
    "Age": 6,
    "Education": 4,
    "Income": 1,
```



```

}

df_person = pd.DataFrame([person])

model = torch.load(os.path.join(os.getcwd(), '../backend/Models/predict_1.pth'))

input_tensor = torch.tensor(df_person.values, dtype=torch.float32)

with torch.no_grad():
    outputs = model(input_tensor)
    probabilities = torch.sigmoid(outputs).numpy()

df_person['P(A)'] = np.round(probabilities * 100, 2)
df_person['~P(A)'] = np.round((1 - probabilities) * 100, 2)

print(df_person)

```

```

      HighBP  HighChol  CholCheck    BMI  Smoker  Stroke  Diabetes  PhysActivity  \
0          0         0          1  22.92        0         0          0          0

      Fruits  Veggies  ...  GenHlth  MentHlth  PhysHlth  DiffWalk  Sex  Age  \
0          1         0  ...        3        15          0          0   0    6

      Education  Income  P(A)    ~P(A)
0             4         1  14.94  85.059998

[1 rows x 23 columns]

```

```

[4]: !jupyter nbconvert --to pdf --output-dir='../frontend/public/PDF' \
      ↪Heartdisease_prediction.ipynb

```

```

[NbConvertApp] Converting notebook Heartdisease_prediction.ipynb to notebook
[NbConvertApp] ERROR | Notebook JSON is invalid: Additional properties are not
allowed ('execution_count', 'outputs' were unexpected)

```

Failed validating 'additionalProperties' in markdown_cell:

```

On instance['cells'][70]:
{'cell_type': 'markdown',
 'execution_count': 63,
 'metadata': {'ExecuteTime': {'end_time': '2024-09-18T15:19:36.869459Z',
                              'start_time': '2024-09-18T15:19:36.831676Z'},
              'collapsed': False},
 'outputs': ['...1 outputs...'],
 'source': 'Nawiązanie połączenia z bazą i zapis parametrów modelu '
           '[Pamiętaj...]'
[NbConvertApp] Writing 268418 bytes to Heartdisease_prediction.ipynb

```

```
[2]: import os
import platform

if platform.system() == 'Darwin': # macOS
    os.system('say "Model training completed"')
elif platform.system() == 'Windows': # Windows
    os.system(
        'PowerShell -Command "Add-Type -TypeDefinition \'public class Speech {
        ↪public static void Speak(string text) { new System.Speech.Synthesis.
        ↪SpeechSynthesizer().Speak(text); } }\' ; [Speech]::Speak(\'Model training
        ↪completed\')"'')
else: # Linux/UNIX
    os.system('echo "Model training completed" | espeak')
```

```
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1917342 bytes to
../frontend/public/PDF/Heartdisease_prediction.pdf
```