Diabetes_prediction

December 30, 2024

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

[16]:
```
import os
import kaggle
import zipfile

dataset_path = 'Datasets/prosperchuks/health-dataset'
file_name = 'diabetes_data.csv'
zip_path = os.path.join(dataset_path, file_name + '.zip')

if not os.path.exists(dataset_path):
    kaggle.api.dataset_download_file('prosperchuks/health-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: diabetes_data.csv:

[17]:
```
import pandas as pd
import warnings
import os
""""""""""""""""""""
warnings.filterwarnings("ignore", category=FutureWarning)
df = pd.read_csv(os.getcwd() + "/Datasets/prosperchuks/health-dataset/
 ↪diabetes_data.csv")
```

0.0.1 Cechy badane przez ankieterów:

- Age: 13-poziomowa kategoria wiekowa: 1 = 18-24 // 9 = 60-64 // 13 = 80 lub starszy
- Sex: płeć pacjenta: 1 - mężczyzna // 0 - kobieta
- HighChol: 0 - brak wysokiego cholesterolu // 1 - wysoki cholesterol
- CholCheck: 0 = brak kontroli cholesterolu w ciągu 5 lat // 1 = tak, kontrola cholesterolu w ciągu 5 lat
- BMI: Indeks Masy Ciała

- Smoker: Czy wypaliłeś przynajmniej 100 papierosów w swoim życiu: 0 = nie // 1 = tak
- HeartDiseaseorAttack: choroba wieńcowa (CHD) / zawał mięśnia sercowego (MI): 0 = nie // 1 = tak
- 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: (dorośli mężczyźni >=14 drinków na tydzień i dorosłe kobiety>=7 drinków na tydzień): 0 = nie // 1 = tak
- GenHlth: Czy powiedziałbyś, że ogólnie twoje zdrowie jest: skala 1-5: 1 = doskonałe // 2 = bardzo dobre // 3 = dobre // 4 = dość dobre // 5 = słabe
- MentHlth: dni złego zdrowia psychicznego skala: 1-30 dni
- PhysHlth: dni choroby fizycznej lub urazu w ciągu ostatnich 30 dni skala: 1-30
- DiffWalk: Czy masz poważne trudności z chodzeniem lub wchodzeniem po schodach: 0 = nie // 1 = tak
- Stroke: czy kiedykolwiek miałeś udar: 0 = nie // 1 = tak
- HighBP: 0 = brak wysokiego ciśnienia // 1 = wysokie ciśnienie krwi
- Heartdisease: 0 = brak cukrzycy // 1 = cukrzyca

Sprawdzenie oraz usunięcie ewentualnych duplikatów w badanym zbiorze:

[18]:
```python
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: 64020
```

Sprawdzenie i usunięcie ewentualnych pustych rekordów:

[19]:
```python
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: 64020 entries, 0 to 70691
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Age                   64020 non-null  float64
```

```
 1   Sex                   64020 non-null  float64
 2   HighChol              64020 non-null  float64
 3   CholCheck             64020 non-null  float64
 4   BMI                   64020 non-null  float64
 5   Smoker                64020 non-null  float64
 6   HeartDiseaseorAttack  64020 non-null  float64
 7   PhysActivity          64020 non-null  float64
 8   Fruits                64020 non-null  float64
 9   Veggies               64020 non-null  float64
 10  HvyAlcoholConsump     64020 non-null  float64
 11  GenHlth               64020 non-null  float64
 12  MentHlth              64020 non-null  float64
 13  PhysHlth              64020 non-null  float64
 14  DiffWalk              64020 non-null  float64
 15  Stroke                64020 non-null  float64
 16  HighBP                64020 non-null  float64
 17  Diabetes              64020 non-null  float64
dtypes: float64(18)
memory usage: 9.3 MB
```

Liczba unikalnych wartości dla każdej z kolumn:

[20]:
```python
df_unique_values = pd.DataFrame(df.nunique(), columns=['unique'])
print(df_unique_values)
```

```
                      unique
Age                       13
Sex                        2
HighChol                   2
CholCheck                  2
BMI                       80
Smoker                     2
HeartDiseaseorAttack       2
PhysActivity               2
Fruits                     2
Veggies                    2
HvyAlcoholConsump          2
GenHlth                    5
MentHlth                  31
PhysHlth                  31
DiffWalk                   2
Stroke                     2
HighBP                     2
Diabetes                   2
```

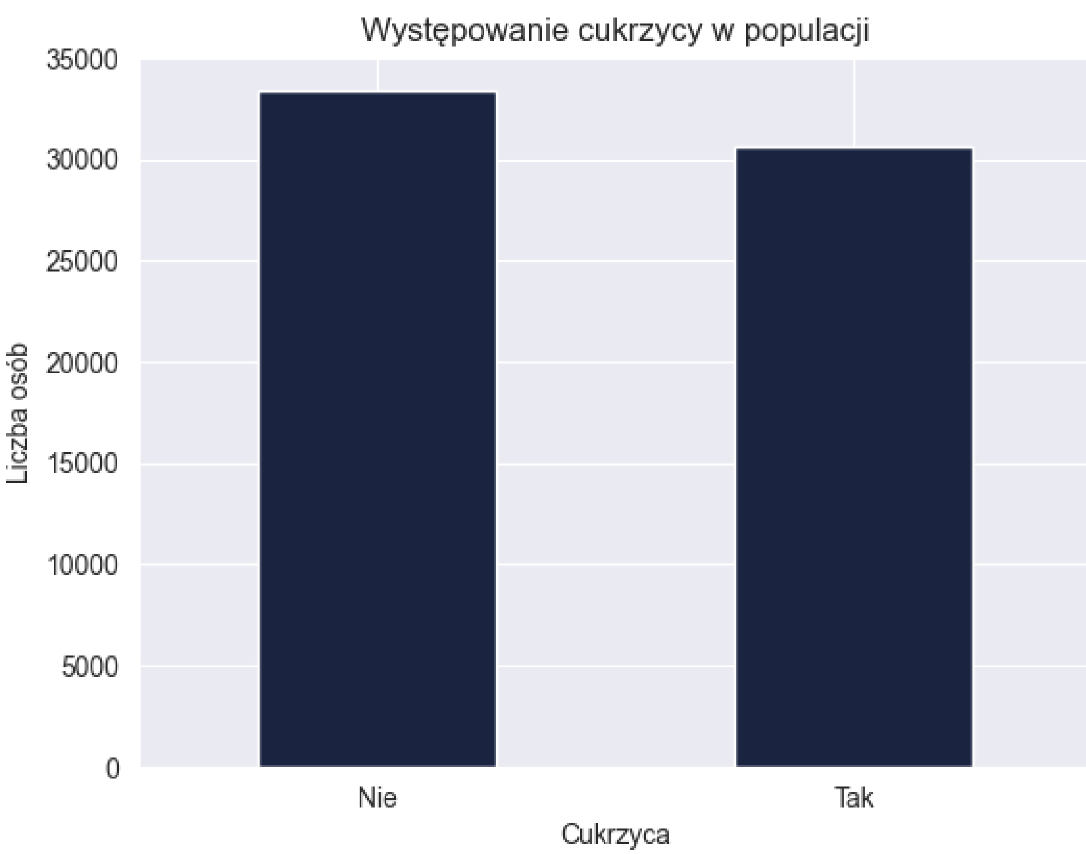
Preferowana paleta kolorów stosowana do tworzenia wykresów:

[21]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.color_palette("cubehelix")
sns.set_palette('cubehelix')
```

Sprawdzenie reprezentatywności danych:

[22]:
```
df['Diabetes'].value_counts().plot(kind='bar')
plt.title('Występowanie cukrzycy w populacji')
plt.xlabel('Cukrzyca')
plt.ylabel('Liczba osób')
plt.xticks([0, 1], ['Nie', 'Tak'], rotation=0)
plt.show()

counts = df['Diabetes'].value_counts()
majority = counts.max()
minority = counts.min()

percentage_difference = ((majority - minority) / minority) * 100
print(f"Klasa większościowa jest większa o {percentage_difference:.2f}%.\n")
```

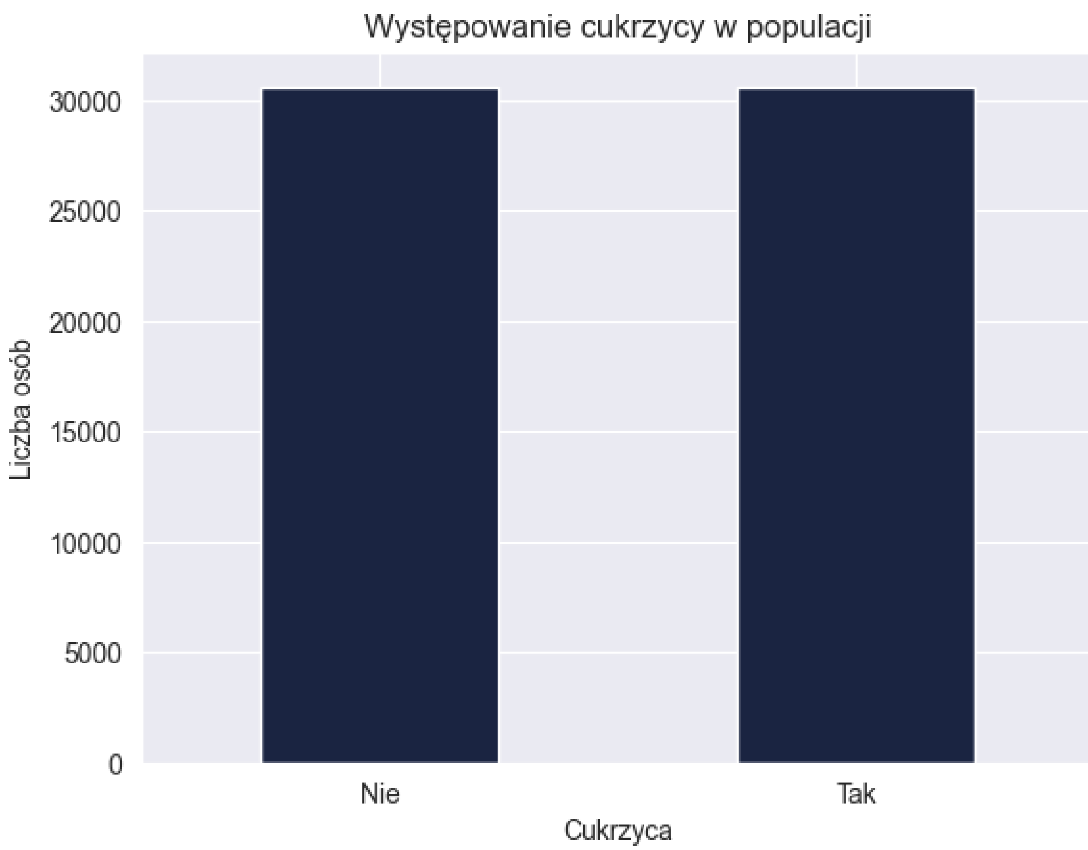



```
Klasa większościowa jest większa o 9.07%.
```

Liczba osób zdrowych w badanym zbiorze jest większa od ilości cukrzyków. W celu jej wyrównania (by zapobiec nadmiernemu dopasowaniu modelu do którejś z klas) możemy zastosować jedną z poniższych technik: - 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 liczniejsza tylko o 9%, zastosuję pierwsze podejście, które polega na losowym duplikowaniu wartości z klasy mniejszościowej.

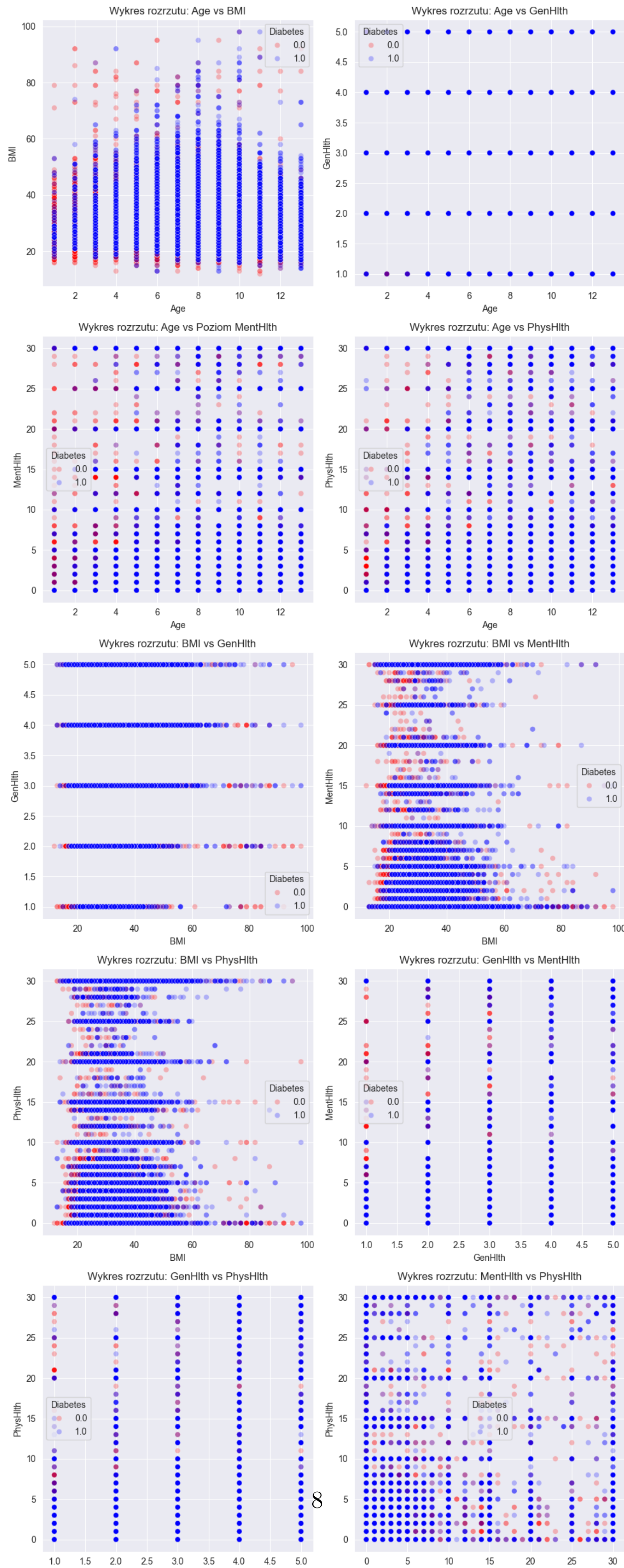
[23]:
```
from sklearn.utils import resample

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

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

X_minority_upsampled, y_minority_upsampled = resample(X_minority, y_minority,
                                                      replace=True,
                                                      n_samples=len(y_majority),
                                                      random_state=123)

X_upsampled = pd.concat([X_majority, X_minority_upsampled])
y_upsampled = pd.concat([y_majority, y_minority_upsampled])

df_upsampled = pd.concat([X_upsampled, y_upsampled], axis=1)
df = df_upsampled

df_upsampled['Diabetes'].value_counts().plot(kind='bar')
plt.title('Występowanie cukrzycy w populacji')
plt.xlabel('Cukrzyca')
plt.ylabel('Liczba osób')
plt.xticks([0, 1], ['Nie', 'Tak'], rotation=0)
plt.show()
```



Wykresy rozrzutu:

[24]:
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axs = plt.subplots(5, 2, figsize=(10, 25))

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

sns.scatterplot(x='Age', y='BMI', hue='Diabetes', data=df, ax=axs[0, 0],alpha=0.
↪25,palette=color_dict)
axs[0, 0].set_title('Wykres rozrzutu: Age vs BMI')

sns.scatterplot(x='Age', y='GenHlth', hue='Diabetes', data=df, ax=axs[0,␣
↪1],alpha=0.25,palette=color_dict)
axs[0, 1].set_title('Wykres rozrzutu: Age vs GenHlth')

sns.scatterplot(x='Age', y='MentHlth', hue='Diabetes', data=df, ax=axs[1,␣
↪0],alpha=0.25,palette=color_dict)
axs[1, 0].set_title('Wykres rozrzutu: Age vs Poziom MentHlth')
```

```
sns.scatterplot(x='Age', y='PhysHlth', hue='Diabetes', data=df, ax=axs[1, 1],alpha=0.25,palette=color_dict)
axs[1, 1].set_title('Wykres rozrzutu: Age vs PhysHlth')

#---

sns.scatterplot(x='BMI', y='GenHlth', hue='Diabetes', data=df, ax=axs[2, 0],alpha=0.25,palette=color_dict)
axs[2, 0].set_title('Wykres rozrzutu: BMI vs GenHlth')

sns.scatterplot(x='BMI', y='MentHlth', hue='Diabetes', data=df, ax=axs[2, 1],alpha=0.25,palette=color_dict)
axs[2, 1].set_title('Wykres rozrzutu: BMI vs MentHlth')

sns.scatterplot(x='BMI', y='PhysHlth', hue='Diabetes', data=df, ax=axs[3, 0],alpha=0.25,palette=color_dict)
axs[3, 0].set_title('Wykres rozrzutu: BMI vs PhysHlth')

#---

sns.scatterplot(x='GenHlth', y='MentHlth', hue='Diabetes', data=df, ax=axs[3, 1],alpha=0.25,palette=color_dict)
axs[3, 1].set_title('Wykres rozrzutu: GenHlth vs MentHlth')

sns.scatterplot(x='GenHlth', y='PhysHlth', hue='Diabetes', data=df, ax=axs[4, 0],alpha=0.25,palette=color_dict)
axs[4, 0].set_title('Wykres rozrzutu: GenHlth vs PhysHlth')

#---

sns.scatterplot(x='MentHlth', y='PhysHlth', hue='Diabetes', data=df, ax=axs[4, 1],alpha=0.25,palette=color_dict)
axs[4, 1].set_title('Wykres rozrzutu: MentHlth vs PhysHlth')

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


Wykres rozrzutu: Age vs BMI
BMI
Age
Diabetes
0.0
1.0
Wykres rozrzutu: Age vs GenHlth
GenHlth
Age
Diabetes
0.0
1.0
Wykres rozrzutu: Age vs Poziom MentHlth
MentHlth
Age
Diabetes
0.0
1.0
Wykres rozrzutu: Age vs PhysHlth
PhysHlth
Age
Diabetes
0.0
1.0
Wykres rozrzutu: BMI vs GenHlth
GenHlth
BMI
Diabetes
0.0
1.0
Wykres rozrzutu: BMI vs MentHlth
MentHlth
BMI
Diabetes
0.0
1.0
Wykres rozrzutu: BMI vs PhysHlth
PhysHlth
BMI
Diabetes
0.0
1.0
Wykres rozrzutu: GenHlth vs MentHlth
MentHlth
GenHlth
Diabetes
0.0
1.0
Wykres rozrzutu: GenHlth vs PhysHlth
PhysHlth
GenHlth
Diabetes
0.0
1.0
Wykres rozrzutu: MentHlth vs PhysHlth
PhysHlth
MentHlth
Diabetes
0.0
1.0

Rozdzielenie zmiennych na kategoryczne i ilościowe:

[25]:
```
zmienne_ilosciowe = ['Age', 'BMI', 'GenHlth', 'MentHlth', 'PhysHlth']
zmienne_jakosciowe = ['Sex', 'HighChol', 'CholCheck', 'Smoker',␣
↪'HeartDiseaseorAttack', 'PhysActivity', 'Fruits',
                      'Veggies', 'HvyAlcoholConsump', 'DiffWalk', 'Stroke',␣
↪'HighBP']

df_ilosciowe = df[zmienne_ilosciowe]
df_jakosciowe = df[zmienne_jakosciowe]
```

[26]:
```
print(f"Zmienne ilościowe:\n {df_ilosciowe}")
print(f"Zmienne jakościowe:\n {df_jakosciowe}")
```

```
Zmienne ilościowe:
         Age   BMI  GenHlth  MentHlth  PhysHlth
0        4.0  26.0      3.0       5.0      30.0
1       12.0  26.0      3.0       0.0       0.0
2       13.0  26.0      1.0       0.0      10.0
3       11.0  28.0      3.0       0.0       3.0
4        8.0  29.0      2.0       0.0       0.0
…        …     …        …         …         …
69699   13.0  28.0      3.0       0.0       5.0
59781    7.0  33.0      2.0       0.0       0.0
40026    8.0  23.0      3.0       0.0       0.0
65936   12.0  28.0      4.0      10.0      15.0
54915   13.0  25.0      2.0       0.0      10.0

[61244 rows x 5 columns]
Zmienne jakościowe:
       Sex  HighChol  CholCheck  Smoker  HeartDiseaseorAttack  PhysActivity  \
0      1.0       0.0        1.0     0.0                   0.0           1.0
1      1.0       1.0        1.0     1.0                   0.0           0.0
2      1.0       0.0        1.0     0.0                   0.0           1.0
3      1.0       1.0        1.0     1.0                   0.0           1.0
4      0.0       0.0        1.0     1.0                   0.0           1.0
…      …         …          …       …                     …             …
69699  1.0       1.0        1.0     0.0                   0.0           1.0
59781  0.0       0.0        1.0     1.0                   0.0           1.0
40026  1.0       1.0        1.0     1.0                   0.0           1.0
65936  0.0       1.0        1.0     0.0                   0.0           0.0
54915  0.0       0.0        1.0     0.0                   0.0           1.0

       Fruits  Veggies  HvyAlcoholConsump  DiffWalk  Stroke  HighBP
0         0.0      1.0                0.0       0.0     0.0     1.0
```

```
1          1.0      0.0                  0.0       0.0      1.0      1.0
2          1.0      1.0                  0.0       0.0      0.0      0.0
3          1.0      1.0                  0.0       0.0      0.0      1.0
4          1.0      1.0                  0.0       0.0      0.0      0.0
…       …        …                …        …        …        …
69699      0.0      1.0                  0.0       0.0      0.0      1.0
59781      1.0      1.0                  0.0       0.0      0.0      0.0
40026      1.0      1.0                  0.0       0.0      0.0      0.0
65936      1.0      1.0                  0.0       1.0      0.0      1.0
54915      1.0      1.0                  0.0       0.0      0.0      1.0

[61244 rows x 12 columns]
```

[27]:
```
print(f"Próbka:\n {df.sample()}")
```

```
Próbka:
        Age  Sex  HighChol  CholCheck   BMI  Smoker  HeartDiseaseorAttack  \
61305  8.0  1.0       1.0        1.0  49.0     1.0                   0.0

       PhysActivity  Fruits  Veggies  HvyAlcoholConsump  GenHlth  MentHlth  \
61305           0.0     0.0      1.0                0.0      4.0       0.0

       PhysHlth  DiffWalk  Stroke  HighBP  Diabetes
61305      10.0       1.0     0.0     1.0       1.0
```

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

[28]:
```
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 badanej populacji:

[14]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from pandas.plotting import table

df_ilosciowe = df[zmienne_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()
print(df_ilosciowe.describe(include='all'))
```

	Age	BMI	GenHlth	MentHlth	PhysHlth
count	61244.0	61244.0	61244.0	61244.0	61244.0
mean	8.57	30.09	2.9	4.1	6.29
std	2.88	7.29	1.11	8.43	10.31
min	1.0	12.0	1.0	0.0	0.0
25%	7.0	25.0	2.0	0.0	0.0
50%	9.0	29.0	3.0	0.0	0.0
75%	11.0	33.0	4.0	3.0	7.0
max	13.0	98.0	5.0	30.0	30.0

```
                Age           BMI       GenHlth      MentHlth      PhysHlth
count  61244.000000  61244.000000  61244.000000  61244.000000  61244.000000
mean       8.569982     30.086735      2.900905      4.104124      6.292420
std        2.878228      7.285623      1.108131      8.426973     10.313694
min        1.000000     12.000000      1.000000      0.000000      0.000000
25%        7.000000     25.000000      2.000000      0.000000      0.000000
50%        9.000000     29.000000      3.000000      0.000000      0.000000
75%       11.000000     33.000000      4.000000      3.000000      7.000000
max       13.000000     98.000000      5.000000     30.000000     30.000000
```

Age: Średnia wartość kategorii wiekowej wynosi ok. 8.6, 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. 30.1 z odchyleniem standardowym bliskim 7.3. Zakres tego parametru waha się od 12.0 do 98.0.

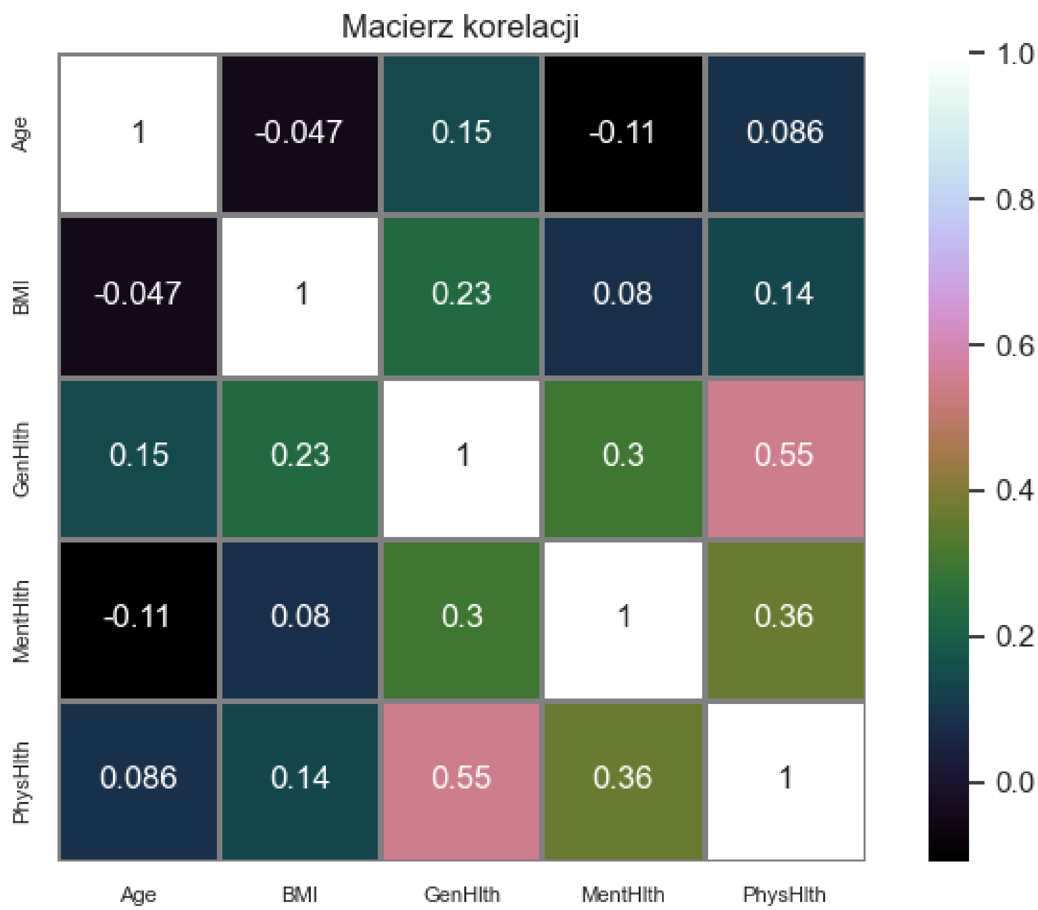
GenHlth: Średnia ocena ogólnego stanu zdrowia wynosi ok. 2.9 na skali 1-5, gdzie 1 oznacza doskonałe zdrowie, a 5 oznacza słabe zdrowie. Odchylenie standardowe tej oceny wynosi blisko 1.1. Zakres ocen waha się od 1 do 5.

MentHlth: Średnia liczba dni złego zdrowia psychicznego w ciągu ostatnich 30 dni wynosi ok. 4.1, z odchyleniem standardowym bliskim 8.4. 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. 6.3, z odchyleniem standardowym bliskim 10.3. Zakres tego parametru waha się od 0 do 30 dni.

Macierz Korelacji dla zmiennych ilościowych:

[15]:
```python
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()
```

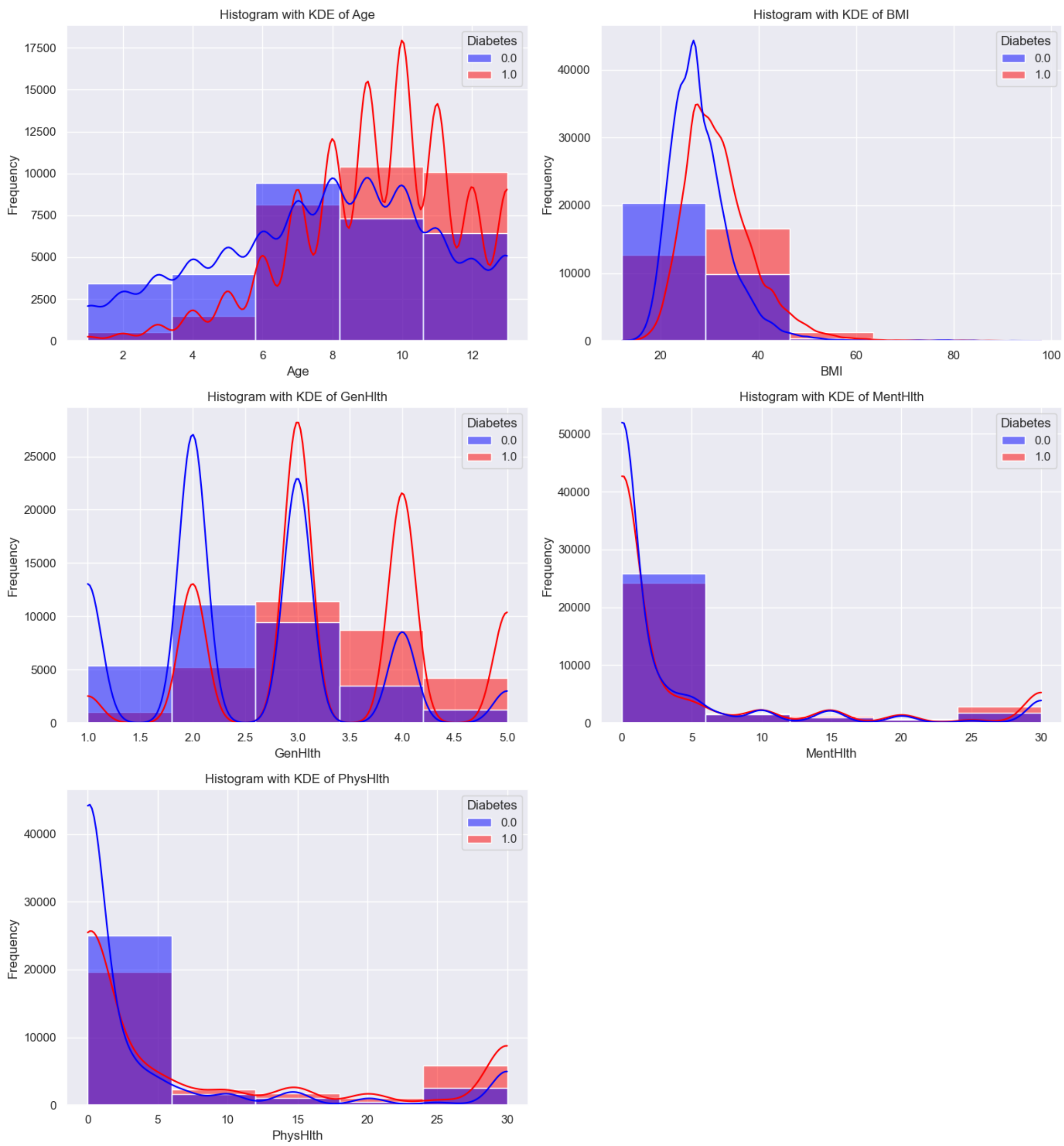


- GenHlth: Ogólny stan zdrowia ma silną, dodatnią korelację z PhysHlth i umiarkowaną z MentHlth, co wskazuje, że lepsze postrzeganie stanu zdrowia jest związane z mniejszą liczbą dni złego zdrowia fizycznego oraz lepszym zdrowiem psychicznym.
- MentHlth: Zdrowie psychiczne ma umiarkowaną korelację z PhysHlth, sugerując, że gorsze zdrowie psychiczne może być powiązane z większą liczbą dni złego zdrowia fizycznego.

Histogramy skategoryzowane przy pomocy zmiennej kategorycznej: “Diabetes”:

[16]:
```
num_cols = ['Age', 'BMI', 'GenHlth', 'MentHlth', 'PhysHlth']
fig, axs = plt.subplots(3, 2, figsize=(14, 15))

axs = axs.ravel()

for i, col in enumerate(num_cols):
    sns.histplot(df, x=col, hue='Diabetes', 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 - U osób z cukrzycą obserwuje się statystycznie wyższy wskaźnik masy ciała (BMI) w porównaniu do osób zdrowych. - Osoby chore częściej deklarują gorszy ogólny stan zdrowia w porównaniu do osób zdrowych.

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 kombinac- jach kategorii dwóch zmiennych wejściowych".

https://www.ibm.com/docs/pl/cognos- analytics/11.2.0?topic=tests-analysis-variance-anova

---

1.0.1 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

zmienne_ilościowe = ['Age', 'BMI', 'GenHlth', 'MentHlth', 'PhysHlth']

#------------------------------------------------------------------------------------

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

    result = stats.anderson(df[zmienna])
    if result.statistic < result.critical_values[2]:
        print(f"Dane mają rozkład normalny (statystyka={result.statistic})")
    else:
        print(f"Dane nie mają rozkładu normalnego (statystyka={result.
↪statistic})")

#------------------------------------------------------------------------------------

# 2)
zmienne_ilosciowe = ['Age', 'BMI', 'GenHlth', 'MentHlth', 'PhysHlth']
zmienne_jakościowe = ['Sex', 'HighChol', 'CholCheck', 'Smoker',␣
↪'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',␣
↪'HvyAlcoholConsump', 'DiffWalk', 'Stroke', 'HighBP']

for zmienna_ilosciowa in zmienne_ilościowe:
    print(f"\n2) Zmienna ilościowa: {zmienna_ilosciowa}")
    for zmienna_jakosciowa in zmienne_jakościowe:
        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) Provenance -> The underlying uncleaned data comes from the CDC's␣
↪BRFSS 2015")
```

```
1) Zmienna: Age
Dane nie mają rozkładu normalnego (statystyka=717.7442893748157)

1) Zmienna: BMI
Dane nie mają rozkładu normalnego (statystyka=1012.8380145915507)

1) Zmienna: GenHlth
Dane nie mają rozkładu normalnego (statystyka=2081.1946976690015)

1) Zmienna: MentHlth
Dane nie mają rozkładu normalnego (statystyka=12050.615713489213)

1) Zmienna: PhysHlth
Dane nie mają rozkładu normalnego (statystyka=9730.149137795)

2) Zmienna ilościowa: Age
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=4.696504613021341,
p=0.030227866860082934)
Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=2729.0470117343593, p=0.0)
Grupy zdefiniowane przez CholCheck nie mają równej wariancji
(W=7.6731078623343345, p=0.005606671390102972)
Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=526.2385175945345,
p=5.745135281716739e-116)
Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=1162.184521548586, p=2.3579519472147905e-252)
Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=236.58996445201763, p=2.7421018374346894e-53)
Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=29.86186954556128,
p=4.657512609492298e-08)
Grupy zdefiniowane przez Veggies nie mają równej wariancji
(W=25.087503560468686, p=5.493873689747063e-07)
Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
```

(W=6.553154435869483, p=0.010472215727913506)
Grupy zdefiniowane przez DiffWalk nie mają równej wariancji
(W=1080.014716075212, p=8.134487460953005e-235)
Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=368.90899287771305, p=5.6409798023294e-82)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=2400.4834549591437, p=0.0)

2) Zmienna ilościowa: BMI
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=656.8392602940776, p=4.202307565166787e-144)
Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=4.570771035073808, p=0.0325258712765848)
Grupy zdefiniowane przez CholCheck nie mają równej wariancji
(W=25.91103298584196, p=3.585755255819403e-07)
Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=27.331323104040678, p=1.719732498250378e-07)
Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=4.291791386296216, p=0.03830097101836268)
Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=422.3048680032292, p=1.5921607371889658e-93)
Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=34.05786969626146, p=5.3765689669649e-09)
Grupy zdefiniowane przez Veggies nie mają równej wariancji
(W=3.9357990865895935, p=0.04727313569846953)
Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
(W=98.98376208949267, p=2.651708892133844e-23)
Grupy zdefiniowane przez DiffWalk nie mają równej wariancji
(W=1193.0467869313309, p=6.190689817258235e-259)
Grupy zdefiniowane przez Stroke mają równą wariancję (W=0.004967379003531898, p=0.9438121389655117)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=313.1320843212042, p=6.78078027147451e-70)

2) Zmienna ilościowa: GenHlth
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=10.560594006720322, p=0.0011558637061015718)
Grupy zdefiniowane przez HighChol nie mają równej wariancji (W=96.5945404630519, p=8.845156115652941e-23)
Grupy zdefiniowane przez CholCheck mają równą wariancję (W=0.05769542517405458, p=0.8101770097548162)
Grupy zdefiniowane przez Smoker mają równą wariancję (W=2.7328108273218383, p=0.09831125591394989)
Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=18.844688352229124, p=1.420325574022495e-05)
Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=41.80716302370786, p=1.0148895174099109e-10)
Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=10.865340273328142,

p=0.0009803644333375525)
Grupy zdefiniowane przez Veggies mają równą wariancję (W=2.815310274087732,
p=0.09337394399226098)
Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
(W=5.058853652363837, p=0.02450397660486704)
Grupy zdefiniowane przez DiffWalk nie mają równej wariancji
(W=31.804915664780847, p=1.7120976286433406e-08)
Grupy zdefiniowane przez Stroke mają równą wariancję (W=2.337011404908597,
p=0.12633688170925791)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=104.96541380570359,
p=1.3010815019798198e-24)

2) Zmienna ilościowa: MentHlth
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=567.9227585712999,
p=5.908513598548576e-125)
Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=310.5628296159958, p=2.444153259681877e-69)
Grupy zdefiniowane przez CholCheck mają równą wariancję (W=1.8375641816615162,
p=0.17524170746169734)
Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=435.29031349075325,
p=2.485069860160568e-96)
Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji
(W=246.78236730916285, p=1.676806193452093e-55)
Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=718.7313694679885, p=2.059518036594069e-157)
Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=102.36993890832706,
p=4.811492606617359e-24)
Grupy zdefiniowane przez Veggies nie mają równej wariancji (W=84.09166080631971,
p=4.8651211620720206e-20)
Grupy zdefiniowane przez HvyAlcoholConsump mają równą wariancję
(W=2.3651287240515932, p=0.12407884466363063)
Grupy zdefiniowane przez DiffWalk nie mają równej wariancji
(W=3428.498277656508, p=0.0)
Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=341.82212526139347,
p=4.1269759958832946e-76)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=138.45179775533,
p=6.283536889569203e-32)

2) Zmienna ilościowa: PhysHlth
Grupy zdefiniowane przez Sex nie mają równej wariancji (W=135.2234605665597,
p=3.182038210341204e-31)
Grupy zdefiniowane przez HighChol nie mają równej wariancji
(W=1082.4856052218347, p=2.4132666567869314e-235)
Grupy zdefiniowane przez CholCheck nie mają równej wariancji
(W=106.32029484234776, p=6.574650502444785e-25)
Grupy zdefiniowane przez Smoker nie mają równej wariancji (W=756.5495674465726,
p=1.5431575364981397e-165)
Grupy zdefiniowane przez HeartDiseaseorAttack nie mają równej wariancji

```
(W=1961.7790831494028, p=0.0)
Grupy zdefiniowane przez PhysActivity nie mają równej wariancji
(W=2825.7022770671188, p=0.0)
Grupy zdefiniowane przez Fruits nie mają równej wariancji (W=44.98930043174396,
p=1.9982951886573433e-11)
Grupy zdefiniowane przez Veggies nie mają równej wariancji
(W=124.89675510275757, p=5.719089597469593e-29)
Grupy zdefiniowane przez HvyAlcoholConsump nie mają równej wariancji
(W=131.97902865211768, p=1.6250890989196228e-30)
Grupy zdefiniowane przez DiffWalk nie mają równej wariancji
(W=15496.293323686332, p=0.0)
Grupy zdefiniowane przez Stroke nie mają równej wariancji (W=1116.301536572964,
p=1.4538310489171994e-242)
Grupy zdefiniowane przez HighBP nie mają równej wariancji (W=1681.0490270114392,
p=0.0)
```

```
3) Provenance -> The underlying uncleaned data comes from the CDC's BRFSS 2015
```

2 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 no 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 = '{} ~ '.format(zmienna) + ' + '.join(['C({})'.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} && {var}\n', tukey,␣
↪'\n')
#
#     print('\033[92m - - - - - - - - - - - - - - - - - - - - - - - - - - - -␣
↪- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -\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, ze dla większość badanych grup nie jest z sobą porównywalna. Konieczne staje się więc wzięce pod uwagę tego faktu poprzez korektę uwzględniająca heteroskedastyczność rozkładu - 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 prawdopodobienstwem 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']
zmienne_jakościowe = ['Sex', 'HighChol', 'CholCheck', 'Smoker',␣
↪'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',␣
↪'HvyAlcoholConsump', 'DiffWalk', 'Stroke', 'HighBP']

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-Wallisa dla Sex: H=2.2639757019029543,

p=0.13241373550499863
Test Kruskala-Wallisa dla HighChol: H=2459.916283727517, 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-Wallisa dla CholCheck: H=658.6115961927059,

p=2.994177057619259e-145

```
Test post-hoc Conovera dla CholCheck:
                0.0           1.0
0.0   1.000000e+00  5.059127e-146
1.0  5.059127e-146   1.000000e+00
```

Test Kruskala-Wallisa dla Smoker: H=414.3436907386676,

p=4.155550329339285e-92

```
Test post-hoc Conovera dla Smoker:
               0.0           1.0
0.0  1.000000e+00  2.062388e-92
1.0  2.062388e-92  1.000000e+00
```

Test Kruskala-Wallisa dla HeartDiseaseorAttack: H=3308.6404045885065,

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-Wallisa dla PhysActivity: H=492.7707915695214,

p=3.5556816247330817e-109

```
Test post-hoc Conovera dla PhysActivity:
                0.0           1.0
0.0   1.000000e+00  1.317886e-109
1.0  1.317886e-109   1.000000e+00
```

Test Kruskala-Wallisa dla Fruits: H=376.6144364376028,

p=6.792977983518974e-84

```
Test post-hoc Conovera dla Fruits:
               0.0           1.0
0.0  1.000000e+00  3.809929e-84
1.0  3.809929e-84  1.000000e+00
```

Test Kruskala-Wallisa dla Veggies: H=11.060472270214634,

p=0.000881876778303605

```
Test post-hoc Conovera dla Veggies:
          0.0       1.0
0.0  1.000000  0.000882
```

```
1.0  0.000882  1.000000
```

Test Kruskala-Wallisa dla HvyAlcoholConsump: H=213.0138890032476,

p=3.022333624668867e-48

```
Test post-hoc Conovera dla HvyAlcoholConsump:
              0.0           1.0
0.0  1.000000e+00  2.514656e-48
1.0  2.514656e-48  1.000000e+00
```

Test Kruskala-Wallisa dla DiffWalk: H=2252.9969418995624, p=0.0

```
Test post-hoc Conovera dla DiffWalk:
     0.0  1.0
0.0  1.0  0.0
1.0  0.0  1.0
```

Test Kruskala-Wallisa dla Stroke: H=1034.7487051151434,

p=5.026636966905747e-227

```
Test post-hoc Conovera dla Stroke:
               0.0            1.0
0.0   1.000000e+00  6.097474e-229
1.0  6.097474e-229   1.000000e+00
```

Test Kruskala-Wallisa dla HighBP: H=5682.515221444696, 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
```

- -

- -

Zmienna ilościowa: BMI

Test Kruskala-Wallisa dla Sex: H=18.27454895616037,

p=1.9124529520337573e-05

```
Test post-hoc Conovera dla Sex:
          0.0       1.0
0.0  1.000000  0.000019
1.0  0.000019  1.000000
```

Test Kruskala-Wallisa dla HighChol: H=1159.0269561249731,

p=4.8979424816534696e-254

```
Test post-hoc Conovera dla HighChol:
               0.0            1.0
0.0   1.000000e+00  1.914995e-256
1.0  1.914995e-256   1.000000e+00
```

```
Test Kruskala-Wallisa dla CholCheck: H=246.7745796272375,
p=1.31107744788015e-55
Test post-hoc Conovera dla CholCheck:
              0.0           1.0
0.0  1.000000e+00  1.023909e-55
1.0  1.023909e-55  1.000000e+00

Test Kruskala-Wallisa dla Smoker: H=0.4575977784450538,
p=0.4987487996913431
Test Kruskala-Wallisa dla HeartDiseaseorAttack: H=237.2680191790706,
p=1.5502910114073992e-53
Test post-hoc Conovera dla HeartDiseaseorAttack:
              0.0           1.0
0.0  1.000000e+00  1.233681e-53
1.0  1.233681e-53  1.000000e+00

Test Kruskala-Wallisa dla PhysActivity: H=1305.8704240291127,
p=5.990988193188192e-286
Test post-hoc Conovera dla PhysActivity:
               0.0            1.0
0.0   1.000000e+00  5.191644e-289
1.0  5.191644e-289   1.000000e+00

Test Kruskala-Wallisa dla Fruits: H=297.9432740301681,
p=9.244166186676275e-67
Test post-hoc Conovera dla Fruits:
              0.0           1.0
0.0  1.000000e+00  6.442412e-67
1.0  6.442412e-67  1.000000e+00

Test Kruskala-Wallisa dla Veggies: H=113.06928730561066,
p=2.0836484006321465e-26
Test post-hoc Conovera dla Veggies:
              0.0           1.0
0.0  1.000000e+00  1.979435e-26
1.0  1.979435e-26  1.000000e+00

Test Kruskala-Wallisa dla HvyAlcoholConsump: H=350.760306024839,
p=2.8944943360271977e-78
Test post-hoc Conovera dla HvyAlcoholConsump:
              0.0           1.0
0.0  1.000000e+00  1.753379e-78
1.0  1.753379e-78  1.000000e+00
```

```
Test Kruskala-Wallisa dla DiffWalk: H=2887.927176896437, p=0.0
Test post-hoc Conovera dla DiffWalk:
      0.0  1.0
0.0  1.0  0.0
1.0  0.0  1.0

Test Kruskala-Wallisa dla Stroke: H=23.614997754008797,
p=1.1766683525618813e-06
Test post-hoc Conovera dla Stroke:
           0.0       1.0
0.0  1.000000  0.000001
1.0  0.000001  1.000000

Test Kruskala-Wallisa dla HighBP: H=3858.9900835960843, 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

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - -

Zmienna ilościowa: GenHlth
Test Kruskala-Wallisa dla Sex: H=33.6904877089779,
p=6.461588476238948e-09
Test post-hoc Conovera dla Sex:
               0.0           1.0
0.0  1.000000e+00  6.433598e-09
1.0  6.433598e-09  1.000000e+00

Test Kruskala-Wallisa dla HighChol: H=2671.7025238527476, 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-Wallisa dla CholCheck: H=300.3411139150829,
p=2.7762225407001237e-67
Test post-hoc Conovera dla CholCheck:
               0.0           1.0
0.0  1.000000e+00  1.923480e-67
1.0  1.923480e-67  1.000000e+00
```

```
Test Kruskala-Wallisa dla Smoker: H=970.6342238843598,
p=4.3393260603320065e-213
Test post-hoc Conovera dla Smoker:
                0.0           1.0
0.0   1.000000e+00  8.970144e-215
1.0  8.970144e-215   1.000000e+00

Test Kruskala-Wallisa dla HeartDiseaseorAttack: H=4160.236084429845,
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-Wallisa dla PhysActivity: H=3633.314880115065, 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-Wallisa dla Fruits: H=248.73477489140353,
p=4.900866899130628e-56
Test post-hoc Conovera dla Fruits:
               0.0          1.0
0.0  1.000000e+00  3.812274e-56
1.0  3.812274e-56  1.000000e+00

Test Kruskala-Wallisa dla Veggies: H=488.9504191890401,
p=2.4109568601103072e-108
Test post-hoc Conovera dla Veggies:
                0.0           1.0
0.0   1.000000e+00  9.074730e-109
1.0  9.074730e-109   1.000000e+00

Test Kruskala-Wallisa dla HvyAlcoholConsump: H=352.13560263857596,
p=1.452398725330619e-78
Test post-hoc Conovera dla HvyAlcoholConsump:
               0.0          1.0
0.0  1.000000e+00  8.763349e-79
1.0  8.763349e-79  1.000000e+00

Test Kruskala-Wallisa dla DiffWalk: H=12654.784248041677, p=0.0
Test post-hoc Conovera dla DiffWalk:
      0.0  1.0
0.0  1.0  0.0
```

```
1.0  0.0  1.0
```

Test Kruskala-Wallisa dla Stroke: H=1857.4472502191625, 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-Wallisa dla HighBP: H=5220.314029351669, 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
```

- -

- -

Zmienna ilościowa: MentHlth

Test Kruskala-Wallisa dla Sex: H=1151.7283175231942,

p=1.889131968456205e-252

```
Test post-hoc Conovera dla Sex:
                0.0            1.0
0.0   1.000000e+00  7.922535e-255
1.0  7.922535e-255   1.000000e+00
```

Test Kruskala-Wallisa dla HighChol: H=183.87927232335454,

p=6.893923440003644e-42

```
Test post-hoc Conovera dla HighChol:
               0.0           1.0
0.0  1.000000e+00  6.012645e-42
1.0  6.012645e-42  1.000000e+00
```

Test Kruskala-Wallisa dla CholCheck: H=4.405164437599216,

p=0.035830270255246155

```
Test post-hoc Conovera dla CholCheck:
           0.0       1.0
0.0  1.000000  0.035829
1.0  0.035829  1.000000
```

Test Kruskala-Wallisa dla Smoker: H=142.73119441376483,

p=6.729833854003185e-33

```
Test post-hoc Conovera dla Smoker:
               0.0           1.0
0.0  1.000000e+00  6.199362e-33
1.0  6.199362e-33  1.000000e+00
```

```
Test Kruskala-Wallisa dla HeartDiseaseorAttack: H=65.91135476317423,
p=4.716665099730207e-16
Test post-hoc Conovera dla HeartDiseaseorAttack:
              0.0           1.0
0.0  1.000000e+00  4.636284e-16
1.0  4.636284e-16  1.000000e+00

Test Kruskala-Wallisa dla PhysActivity: H=265.91693205252744,
p=8.806854656805893e-60
Test post-hoc Conovera dla PhysActivity:
              0.0           1.0
0.0  1.000000e+00  6.607698e-60
1.0  6.607698e-60  1.000000e+00

Test Kruskala-Wallisa dla Fruits: H=57.54164773809624,
p=3.3089805290526874e-14
Test post-hoc Conovera dla Fruits:
              0.0           1.0
0.0  1.000000e+00  3.266128e-14
1.0  3.266128e-14  1.000000e+00

Test Kruskala-Wallisa dla Veggies: H=13.297162212760497,
p=0.00026580812744934646
Test post-hoc Conovera dla Veggies:
          0.0       1.0
0.0  1.000000  0.000266
1.0  0.000266  1.000000

Test Kruskala-Wallisa dla HvyAlcoholConsump: H=5.536373405108957,
p=0.01862514149712672
Test post-hoc Conovera dla HvyAlcoholConsump:
          0.0       1.0
0.0  1.000000  0.018624
1.0  0.018624  1.000000

Test Kruskala-Wallisa dla DiffWalk: H=2338.0376670947285, p=0.0
Test post-hoc Conovera dla DiffWalk:
     0.0  1.0
0.0  1.0  0.0
1.0  0.0  1.0

Test Kruskala-Wallisa dla Stroke: H=162.52140893286304,
p=3.182562253003657e-37
```

```
Test post-hoc Conovera dla Stroke:
               0.0           1.0
0.0  1.000000e+00  2.860575e-37
1.0  2.860575e-37  1.000000e+00
```

Test Kruskala-Wallisa dla HighBP: H=13.602159483702712,

p=0.00022592548812835054

```
Test post-hoc Conovera dla HighBP:
          0.0       1.0
0.0  1.000000  0.000226
1.0  0.000226  1.000000
```

- -

- -

Zmienna ilościowa: PhysHlth

Test Kruskala-Wallisa dla Sex: H=320.4948823152471,

p=1.1300701785251155e-71

```
Test post-hoc Conovera dla Sex:
               0.0           1.0
0.0  1.000000e+00  7.439019e-72
1.0  7.439019e-72  1.000000e+00
```

Test Kruskala-Wallisa dla HighChol: H=1100.1508772434124,

p=3.0626152936573335e-241

```
Test post-hoc Conovera dla HighChol:
                0.0            1.0
0.0   1.000000e+00  2.081004e-243
1.0  2.081004e-243   1.000000e+00
```

Test Kruskala-Wallisa dla CholCheck: H=121.14030830841521,

p=3.5603942549714536e-28

```
Test post-hoc Conovera dla CholCheck:
               0.0           1.0
0.0  1.000000e+00  3.356495e-28
1.0  3.356495e-28  1.000000e+00
```

Test Kruskala-Wallisa dla Smoker: H=425.68502013408033,

p=1.4127129932097826e-94

```
Test post-hoc Conovera dla Smoker:
               0.0           1.0
0.0  1.000000e+00  6.742638e-95
1.0  6.742638e-95  1.000000e+00
```

```
Test Kruskala-Wallisa dla HeartDiseaseorAttack: H=1729.0487711692442,
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-Wallisa dla PhysActivity: H=1930.9663608666763, 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-Wallisa dla Fruits: H=22.46719597353973,
p=2.1376317722169706e-06
Test post-hoc Conovera dla Fruits:
           0.0       1.0
0.0  1.000000  0.000002
1.0  0.000002  1.000000

Test Kruskala-Wallisa dla Veggies: H=53.98733886800328,
p=2.017856993028833e-13
Test post-hoc Conovera dla Veggies:
               0.0           1.0
0.0  1.000000e+00  1.994895e-13
1.0  1.994895e-13  1.000000e+00

Test Kruskala-Wallisa dla HvyAlcoholConsump: H=153.68609456980911,
p=2.712252191968111e-35
Test post-hoc Conovera dla HvyAlcoholConsump:
               0.0           1.0
0.0  1.000000e+00  2.465704e-35
1.0  2.465704e-35  1.000000e+00

Test Kruskala-Wallisa dla DiffWalk: H=11644.350887758299, p=0.0
Test post-hoc Conovera dla DiffWalk:
      0.0  1.0
0.0  1.0  0.0
1.0  0.0  1.0

Test Kruskala-Wallisa dla Stroke: H=1089.4945831518323,
p=6.341427784210238e-239
Test post-hoc Conovera dla Stroke:
                0.0            1.0
0.0   1.000000e+00  4.747320e-241
```

```
1.0  4.747320e-241   1.000000e+00

Test Kruskala-Wallisa dla HighBP: H=1580.238439312764, 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

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

Pomiędzy wszystkimi badanymi grupami (dla wszystkich zmiennych) istnieją istotne statystyczne różnice przy poziomie istotności 0.05. Oznacza to, że każda z badanych grup jest unikalna pod względem badanych zmiennych - różnice te nie są przypadkowe.

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

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

```
[[ 4.   1.   0. …  0.   0.   1.]
 [12.   1.   1. …  0.   1.   1.]
 [13.   1.   0. …  0.   0.   0.]
 …
 [ 8.   1.   1. …  0.   0.   0.]
 [12.   0.   1. …  1.   0.   1.]
 [13.   0.   0. …  0.   0.   1.]]
```

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

[21]:
```
Y = df_copy['Diabetes'].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 wejścowych modelu do tablic numpy array:

[23]:
```
import torch
import numpy as np

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

```
X_val = np.array(X_val, dtype=np.float32)
X_test = np.array(X_test, dtype=np.float32)
y_train = np.array(y_train, dtype=np.float32).ravel()
y_val = np.array(y_val, dtype=np.float32).ravel()
y_test = np.array(y_test, dtype=np.float32).ravel()
```

Zdefiniowanie struktury sieci neuronowe: > Model zbudowany jest na bazie klasy TabNetClassifier, co umożliwia mu efektywne przetwarzanie danych tabelarycznych.

Składa się on z następujących elementów:

- Liczba jednostek w warstwie decyzyjnej i warstwie uwagi - ustalona na 8 neuronów
- Liczba kroków decyzyjnych, określająca ilość przetworzeń danych przed podjęciem finalnej decyzji
- gamma - współczynnik wzmacniający decyzjność modelu
- lambda_sparse - współczynnik karzący model za brak zdecydowania przy wyborze kluczowych cech
- optimizer_fn - wybór optymalizatora
- optimizer_params - dodatkowe parametry przekazywane do optymalizaotra takie jak krok uczenia
- mask_type - typ maskowania

[24]:
```
%%writefile ../backend/Models/net/predict_2.py
from pytorch_tabnet.tab_model import TabNetClassifier
import torch
""""""""""""""""""""""""""
class Model_2:
    def __init__(self, input_dim=17, output_dim=1):
        self.model = TabNetClassifier(
            input_dim=input_dim,
            output_dim=output_dim,
            n_d=8, n_a=8,
            n_steps=3,
            gamma=1.3,
            lambda_sparse=1e-3,
            optimizer_fn=torch.optim.AdamW,
            optimizer_params=dict(lr=0.0001),
            mask_type="sparsemax"
        )
```

```
Overwriting ../backend/Models/net/predict_2.py
```

[25]:
```
from pytorch_tabnet.tab_model import TabNetClassifier
warnings.filterwarnings("ignore")
""""""""""""""""""""""""""
class Model_2:
    def __init__(self, input_dim=17, output_dim=1):
        self.model = TabNetClassifier(
            input_dim=input_dim,
```

```
            output_dim=output_dim,
            n_d=8, n_a=8,
            n_steps=3,
            gamma=1.3,
            lambda_sparse=1e-3,
            optimizer_fn=torch.optim.AdamW,
            optimizer_params=dict(lr=0.0001),
            mask_type="sparsemax"
        )

model = Model_2()
print(model)
```

```
<__main__.Model_2 object at 0x13b81cb90>
```

Inicjalizacja niezbędnych zmiennych w procesie szkolenia i postprocessingu:

[26]:

```
patience = 25  # "Cierpliwość modelu" - maksymalna możliwa ilość epok bez poprawy straty na zbiorze testowym
rg = 500       # Maksymalna liczba epok szkolenia

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

Trening modelu: - Część treningowa, testowa jak i mechanizm early stopping został dostarczony przez bibliotekę pytorch_tabnet - Dane testowe ładowane będą do modelu w paczkach po 1024 próbki, lecz rzeczywiste wielkości partii będą mniejsze (128 próbek) w celu zapewnienia większej stabilności uczenia kosztem szybkości obliczeń

[43]:

```
import os
import warnings

warnings.filterwarnings("ignore", message="Best weights from best epoch are automatically used!")

model.model.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_val, y_val)],
    eval_name=["train", "valid"],
    eval_metric=["logloss", "accuracy"],
    max_epochs=rg,
    patience=patience,
    batch_size=1024,
    virtual_batch_size=128
)

model_save_path = os.path.join(os.getcwd(), '../backend/Models/predict_2.pth')
```

```python
if os.path.exists(model_save_path):
    os.remove(model_save_path)

torch.save(model.model, model_save_path)

print(f'Model saved to {model_save_path}')
```

```
epoch 0  | loss: 0.90938 | train_logloss: 0.7942  | train_accuracy: 0.47247 |
valid_logloss: 0.78945 | valid_accuracy: 0.47535 |  0:00:07s
epoch 1  | loss: 0.88217 | train_logloss: 0.7498  | train_accuracy: 0.48092 |
valid_logloss: 0.75241 | valid_accuracy: 0.47371 |  0:00:14s
epoch 2  | loss: 0.85296 | train_logloss: 0.74223 | train_accuracy: 0.47733 |
valid_logloss: 0.74345 | valid_accuracy: 0.4834  |  0:00:21s
epoch 3  | loss: 0.83228 | train_logloss: 0.74602 | train_accuracy: 0.47831 |
valid_logloss: 0.74962 | valid_accuracy: 0.47491 |  0:00:28s
epoch 4  | loss: 0.80886 | train_logloss: 0.75251 | train_accuracy: 0.47429 |
valid_logloss: 0.75657 | valid_accuracy: 0.47643 |  0:00:36s
epoch 5  | loss: 0.79098 | train_logloss: 0.74739 | train_accuracy: 0.47926 |
valid_logloss: 0.74851 | valid_accuracy: 0.47948 |  0:00:43s
epoch 6  | loss: 0.77287 | train_logloss: 0.7411  | train_accuracy: 0.48787 |
valid_logloss: 0.74207 | valid_accuracy: 0.48667 |  0:00:51s
epoch 7  | loss: 0.76029 | train_logloss: 0.73186 | train_accuracy: 0.49207 |
valid_logloss: 0.73174 | valid_accuracy: 0.49352 |  0:00:58s
epoch 8  | loss: 0.75198 | train_logloss: 0.72511 | train_accuracy: 0.50341 |
valid_logloss: 0.72469 | valid_accuracy: 0.50441 |  0:01:05s
epoch 9  | loss: 0.73692 | train_logloss: 0.71837 | train_accuracy: 0.51257 |
valid_logloss: 0.71836 | valid_accuracy: 0.51791 |  0:01:12s
epoch 10 | loss: 0.73224 | train_logloss: 0.71028 | train_accuracy: 0.52666 |
valid_logloss: 0.70979 | valid_accuracy: 0.53282 |  0:01:18s
epoch 11 | loss: 0.71949 | train_logloss: 0.70256 | train_accuracy: 0.53326 |
valid_logloss: 0.70405 | valid_accuracy: 0.53619 |  0:01:25s
epoch 12 | loss: 0.71536 | train_logloss: 0.69902 | train_accuracy: 0.54458 |
valid_logloss: 0.70211 | valid_accuracy: 0.54588 |  0:01:32s
epoch 13 | loss: 0.71182 | train_logloss: 0.69357 | train_accuracy: 0.55227 |
valid_logloss: 0.69626 | valid_accuracy: 0.55274 |  0:01:38s
epoch 14 | loss: 0.7039  | train_logloss: 0.68785 | train_accuracy: 0.56485 |
valid_logloss: 0.69172 | valid_accuracy: 0.56123 |  0:01:45s
epoch 15 | loss: 0.69832 | train_logloss: 0.68389 | train_accuracy: 0.56956 |
valid_logloss: 0.68706 | valid_accuracy: 0.56711 |  0:01:52s
epoch 16 | loss: 0.69213 | train_logloss: 0.67864 | train_accuracy: 0.5805  |
valid_logloss: 0.68238 | valid_accuracy: 0.57679 |  0:01:59s
epoch 17 | loss: 0.68806 | train_logloss: 0.67404 | train_accuracy: 0.58967 |
valid_logloss: 0.67745 | valid_accuracy: 0.58441 |  0:02:05s
epoch 18 | loss: 0.68331 | train_logloss: 0.67043 | train_accuracy: 0.5979  |
valid_logloss: 0.6742  | valid_accuracy: 0.5892  |  0:02:12s
epoch 19 | loss: 0.67731 | train_logloss: 0.66668 | train_accuracy: 0.60446 |
valid_logloss: 0.66908 | valid_accuracy: 0.599   |  0:02:19s
```

```
epoch 20 | loss: 0.67649 | train_logloss: 0.6639  | train_accuracy: 0.60886 |
valid_logloss: 0.66815 | valid_accuracy: 0.6027  |  0:02:25s
epoch 21 | loss: 0.67264 | train_logloss: 0.66031 | train_accuracy: 0.61369 |
valid_logloss: 0.66563 | valid_accuracy: 0.59987 |  0:02:32s
epoch 22 | loss: 0.66908 | train_logloss: 0.65657 | train_accuracy: 0.61903 |
valid_logloss: 0.66156 | valid_accuracy: 0.60869 |  0:02:39s
epoch 23 | loss: 0.66566 | train_logloss: 0.65352 | train_accuracy: 0.62349 |
valid_logloss: 0.65736 | valid_accuracy: 0.61489 |  0:02:45s
epoch 24 | loss: 0.66443 | train_logloss: 0.65131 | train_accuracy: 0.62554 |
valid_logloss: 0.65588 | valid_accuracy: 0.6175  |  0:02:52s
epoch 25 | loss: 0.65984 | train_logloss: 0.64843 | train_accuracy: 0.6313  |
valid_logloss: 0.65173 | valid_accuracy: 0.6248  |  0:02:58s
epoch 26 | loss: 0.65689 | train_logloss: 0.64559 | train_accuracy: 0.63716 |
valid_logloss: 0.64953 | valid_accuracy: 0.62752 |  0:03:05s
epoch 27 | loss: 0.65465 | train_logloss: 0.64317 | train_accuracy: 0.6383  |
valid_logloss: 0.647   | valid_accuracy: 0.62741 |  0:03:11s
epoch 28 | loss: 0.65288 | train_logloss: 0.64057 | train_accuracy: 0.64166 |
valid_logloss: 0.64449 | valid_accuracy: 0.63383 |  0:03:18s
epoch 29 | loss: 0.65168 | train_logloss: 0.63791 | train_accuracy: 0.64388 |
valid_logloss: 0.64191 | valid_accuracy: 0.63427 |  0:03:24s
epoch 30 | loss: 0.6484  | train_logloss: 0.63596 | train_accuracy: 0.64761 |
valid_logloss: 0.63889 | valid_accuracy: 0.64003 |  0:03:31s
epoch 31 | loss: 0.64541 | train_logloss: 0.63313 | train_accuracy: 0.64978 |
valid_logloss: 0.63633 | valid_accuracy: 0.64504 |  0:03:37s
epoch 32 | loss: 0.64234 | train_logloss: 0.63072 | train_accuracy: 0.65491 |
valid_logloss: 0.63415 | valid_accuracy: 0.64776 |  0:03:44s
epoch 33 | loss: 0.64041 | train_logloss: 0.62934 | train_accuracy: 0.65491 |
valid_logloss: 0.63137 | valid_accuracy: 0.64874 |  0:03:50s
epoch 34 | loss: 0.63898 | train_logloss: 0.62725 | train_accuracy: 0.65958 |
valid_logloss: 0.62959 | valid_accuracy: 0.65462 |  0:03:57s
epoch 35 | loss: 0.63621 | train_logloss: 0.62528 | train_accuracy: 0.66093 |
valid_logloss: 0.62701 | valid_accuracy: 0.65832 |  0:04:03s
epoch 36 | loss: 0.63489 | train_logloss: 0.62201 | train_accuracy: 0.66289 |
valid_logloss: 0.62422 | valid_accuracy: 0.65941 |  0:04:10s
epoch 37 | loss: 0.63248 | train_logloss: 0.62064 | train_accuracy: 0.66475 |
valid_logloss: 0.62285 | valid_accuracy: 0.66355 |  0:04:16s
epoch 38 | loss: 0.63007 | train_logloss: 0.61815 | train_accuracy: 0.66648 |
valid_logloss: 0.61998 | valid_accuracy: 0.66561 |  0:04:23s
epoch 39 | loss: 0.62834 | train_logloss: 0.61767 | train_accuracy: 0.66755 |
valid_logloss: 0.61933 | valid_accuracy: 0.66736 |  0:04:30s
epoch 40 | loss: 0.62623 | train_logloss: 0.6153  | train_accuracy: 0.66972 |
valid_logloss: 0.61711 | valid_accuracy: 0.67019 |  0:04:37s
epoch 41 | loss: 0.62553 | train_logloss: 0.61355 | train_accuracy: 0.67157 |
valid_logloss: 0.61656 | valid_accuracy: 0.66757 |  0:04:44s
epoch 42 | loss: 0.62292 | train_logloss: 0.61206 | train_accuracy: 0.67261 |
valid_logloss: 0.61429 | valid_accuracy: 0.67313 |  0:04:50s
epoch 43 | loss: 0.62085 | train_logloss: 0.61042 | train_accuracy: 0.67404 |
valid_logloss: 0.61265 | valid_accuracy: 0.67389 |  0:04:57s
```

```
epoch 44 | loss: 0.62055 | train_logloss: 0.6094  | train_accuracy: 0.67411 |
valid_logloss: 0.61172 | valid_accuracy: 0.67378 |  0:05:03s
epoch 45 | loss: 0.61911 | train_logloss: 0.60828 | train_accuracy: 0.67483 |
valid_logloss: 0.61002 | valid_accuracy: 0.67672 |  0:05:10s
epoch 46 | loss: 0.61695 | train_logloss: 0.60774 | train_accuracy: 0.67569 |
valid_logloss: 0.60848 | valid_accuracy: 0.67944 |  0:05:17s
epoch 47 | loss: 0.61615 | train_logloss: 0.60629 | train_accuracy: 0.67639 |
valid_logloss: 0.60715 | valid_accuracy: 0.67987 |  0:05:23s
epoch 48 | loss: 0.6174  | train_logloss: 0.60449 | train_accuracy: 0.67996 |
valid_logloss: 0.60592 | valid_accuracy: 0.67911 |  0:05:30s
epoch 49 | loss: 0.61432 | train_logloss: 0.60413 | train_accuracy: 0.67777 |
valid_logloss: 0.60539 | valid_accuracy: 0.68031 |  0:05:37s
epoch 50 | loss: 0.61382 | train_logloss: 0.60301 | train_accuracy: 0.68113 |
valid_logloss: 0.6039  | valid_accuracy: 0.68379 |  0:05:43s
epoch 51 | loss: 0.61272 | train_logloss: 0.60121 | train_accuracy: 0.68248 |
valid_logloss: 0.60252 | valid_accuracy: 0.68564 |  0:05:50s
epoch 52 | loss: 0.60974 | train_logloss: 0.6006  | train_accuracy: 0.6815  |
valid_logloss: 0.60155 | valid_accuracy: 0.68455 |  0:05:56s
epoch 53 | loss: 0.61079 | train_logloss: 0.59947 | train_accuracy: 0.68234 |
valid_logloss: 0.60054 | valid_accuracy: 0.68466 |  0:06:03s
epoch 54 | loss: 0.60809 | train_logloss: 0.59872 | train_accuracy: 0.68295 |
valid_logloss: 0.59997 | valid_accuracy: 0.6876  |  0:06:09s
epoch 55 | loss: 0.60731 | train_logloss: 0.59796 | train_accuracy: 0.68472 |
valid_logloss: 0.59865 | valid_accuracy: 0.68826 |  0:06:16s
epoch 56 | loss: 0.60746 | train_logloss: 0.59713 | train_accuracy: 0.68404 |
valid_logloss: 0.59779 | valid_accuracy: 0.68858 |  0:06:22s
epoch 57 | loss: 0.60669 | train_logloss: 0.59608 | train_accuracy: 0.68446 |
valid_logloss: 0.59759 | valid_accuracy: 0.68826 |  0:06:29s
epoch 58 | loss: 0.60568 | train_logloss: 0.5955  | train_accuracy: 0.68563 |
valid_logloss: 0.59707 | valid_accuracy: 0.69    |  0:06:35s
epoch 59 | loss: 0.60459 | train_logloss: 0.59482 | train_accuracy: 0.68654 |
valid_logloss: 0.59652 | valid_accuracy: 0.68934 |  0:06:42s
epoch 60 | loss: 0.60443 | train_logloss: 0.59386 | train_accuracy: 0.68684 |
valid_logloss: 0.59519 | valid_accuracy: 0.68923 |  0:06:48s
epoch 61 | loss: 0.6028  | train_logloss: 0.59282 | train_accuracy: 0.68845 |
valid_logloss: 0.59448 | valid_accuracy: 0.6888  |  0:06:55s
epoch 62 | loss: 0.60116 | train_logloss: 0.59239 | train_accuracy: 0.68859 |
valid_logloss: 0.59406 | valid_accuracy: 0.68923 |  0:07:01s
epoch 63 | loss: 0.60295 | train_logloss: 0.59127 | train_accuracy: 0.68904 |
valid_logloss: 0.59293 | valid_accuracy: 0.68793 |  0:07:08s
epoch 64 | loss: 0.60184 | train_logloss: 0.58995 | train_accuracy: 0.68995 |
valid_logloss: 0.59263 | valid_accuracy: 0.68902 |  0:07:14s
epoch 65 | loss: 0.60024 | train_logloss: 0.58944 | train_accuracy: 0.69025 |
valid_logloss: 0.591   | valid_accuracy: 0.68804 |  0:07:21s
epoch 66 | loss: 0.59938 | train_logloss: 0.58897 | train_accuracy: 0.69111 |
valid_logloss: 0.59077 | valid_accuracy: 0.68945 |  0:07:27s
epoch 67 | loss: 0.59692 | train_logloss: 0.58841 | train_accuracy: 0.69069 |
valid_logloss: 0.591   | valid_accuracy: 0.69065 |  0:07:34s
```

```
epoch 68 | loss: 0.59669 | train_logloss: 0.58713 | train_accuracy: 0.6924  |
valid_logloss: 0.59031 | valid_accuracy: 0.68934 |  0:07:40s
epoch 69 | loss: 0.5988  | train_logloss: 0.58651 | train_accuracy: 0.69286 |
valid_logloss: 0.58934 | valid_accuracy: 0.69065 |  0:07:47s
epoch 70 | loss: 0.59762 | train_logloss: 0.58667 | train_accuracy: 0.69128 |
valid_logloss: 0.58986 | valid_accuracy: 0.68967 |  0:07:53s
epoch 71 | loss: 0.59728 | train_logloss: 0.58588 | train_accuracy: 0.6931  |
valid_logloss: 0.58893 | valid_accuracy: 0.69076 |  0:08:00s
epoch 72 | loss: 0.59534 | train_logloss: 0.58478 | train_accuracy: 0.6924  |
valid_logloss: 0.58817 | valid_accuracy: 0.69    |  0:08:06s
epoch 73 | loss: 0.59387 | train_logloss: 0.5845  | train_accuracy: 0.69436 |
valid_logloss: 0.58811 | valid_accuracy: 0.69239 |  0:08:13s
epoch 74 | loss: 0.59513 | train_logloss: 0.58407 | train_accuracy: 0.69389 |
valid_logloss: 0.5874  | valid_accuracy: 0.69011 |  0:08:19s
epoch 75 | loss: 0.59451 | train_logloss: 0.58373 | train_accuracy: 0.6934  |
valid_logloss: 0.58676 | valid_accuracy: 0.69326 |  0:08:26s
epoch 76 | loss: 0.59351 | train_logloss: 0.58308 | train_accuracy: 0.69422 |
valid_logloss: 0.58577 | valid_accuracy: 0.69283 |  0:08:32s
epoch 77 | loss: 0.59255 | train_logloss: 0.58248 | train_accuracy: 0.69403 |
valid_logloss: 0.58541 | valid_accuracy: 0.69272 |  0:08:39s
epoch 78 | loss: 0.5934  | train_logloss: 0.58147 | train_accuracy: 0.69517 |
valid_logloss: 0.58523 | valid_accuracy: 0.6925  |  0:08:45s
epoch 79 | loss: 0.59118 | train_logloss: 0.58145 | train_accuracy: 0.69613 |
valid_logloss: 0.58498 | valid_accuracy: 0.69163 |  0:08:52s
epoch 80 | loss: 0.5909  | train_logloss: 0.58128 | train_accuracy: 0.69596 |
valid_logloss: 0.58385 | valid_accuracy: 0.69392 |  0:08:58s
epoch 81 | loss: 0.59027 | train_logloss: 0.58098 | train_accuracy: 0.69659 |
valid_logloss: 0.58291 | valid_accuracy: 0.69642 |  0:09:05s
epoch 82 | loss: 0.58858 | train_logloss: 0.5801  | train_accuracy: 0.69741 |
valid_logloss: 0.58297 | valid_accuracy: 0.69413 |  0:09:11s
epoch 83 | loss: 0.58923 | train_logloss: 0.57983 | train_accuracy: 0.69655 |
valid_logloss: 0.58278 | valid_accuracy: 0.69522 |  0:09:18s
epoch 84 | loss: 0.5903  | train_logloss: 0.57972 | train_accuracy: 0.69727 |
valid_logloss: 0.58293 | valid_accuracy: 0.69762 |  0:09:24s
epoch 85 | loss: 0.58984 | train_logloss: 0.57898 | train_accuracy: 0.69804 |
valid_logloss: 0.58264 | valid_accuracy: 0.69435 |  0:09:31s
epoch 86 | loss: 0.58825 | train_logloss: 0.579   | train_accuracy: 0.69676 |
valid_logloss: 0.58218 | valid_accuracy: 0.69642 |  0:09:37s
epoch 87 | loss: 0.58873 | train_logloss: 0.57826 | train_accuracy: 0.69888 |
valid_logloss: 0.58129 | valid_accuracy: 0.69642 |  0:09:44s
epoch 88 | loss: 0.58716 | train_logloss: 0.57807 | train_accuracy: 0.69874 |
valid_logloss: 0.58115 | valid_accuracy: 0.69598 |  0:09:50s
epoch 89 | loss: 0.58681 | train_logloss: 0.57799 | train_accuracy: 0.69881 |
valid_logloss: 0.58091 | valid_accuracy: 0.69457 |  0:09:57s
epoch 90 | loss: 0.58451 | train_logloss: 0.57674 | train_accuracy: 0.69916 |
valid_logloss: 0.58026 | valid_accuracy: 0.69729 |  0:10:03s
epoch 91 | loss: 0.58523 | train_logloss: 0.57648 | train_accuracy: 0.69914 |
valid_logloss: 0.57953 | valid_accuracy: 0.69587 |  0:10:09s
```

```
epoch 92 | loss: 0.58566 | train_logloss: 0.57599 | train_accuracy: 0.69993 |
valid_logloss: 0.57933 | valid_accuracy: 0.6987  |  0:10:16s
epoch 93 | loss: 0.58488 | train_logloss: 0.5756  | train_accuracy: 0.70063 |
valid_logloss: 0.57891 | valid_accuracy: 0.69881 |  0:10:23s
epoch 94 | loss: 0.58469 | train_logloss: 0.57539 | train_accuracy: 0.70033 |
valid_logloss: 0.57818 | valid_accuracy: 0.6999  |  0:10:29s
epoch 95 | loss: 0.58276 | train_logloss: 0.57501 | train_accuracy: 0.70047 |
valid_logloss: 0.57823 | valid_accuracy: 0.70077 |  0:10:35s
epoch 96 | loss: 0.58127 | train_logloss: 0.57442 | train_accuracy: 0.7007  |
valid_logloss: 0.57768 | valid_accuracy: 0.70088 |  0:10:42s
epoch 97 | loss: 0.58412 | train_logloss: 0.57439 | train_accuracy: 0.70177 |
valid_logloss: 0.57779 | valid_accuracy: 0.70056 |  0:10:48s
epoch 98 | loss: 0.58289 | train_logloss: 0.57385 | train_accuracy: 0.70124 |
valid_logloss: 0.57707 | valid_accuracy: 0.7023  |  0:10:55s
epoch 99 | loss: 0.58211 | train_logloss: 0.57361 | train_accuracy: 0.70194 |
valid_logloss: 0.57705 | valid_accuracy: 0.70251 |  0:11:01s
epoch 100| loss: 0.583   | train_logloss: 0.5733  | train_accuracy: 0.70152 |
valid_logloss: 0.57633 | valid_accuracy: 0.70153 |  0:11:08s
epoch 101| loss: 0.58235 | train_logloss: 0.57288 | train_accuracy: 0.70261 |
valid_logloss: 0.57661 | valid_accuracy: 0.70175 |  0:11:14s
epoch 102| loss: 0.58083 | train_logloss: 0.57293 | train_accuracy: 0.70233 |
valid_logloss: 0.57625 | valid_accuracy: 0.70306 |  0:11:21s
epoch 103| loss: 0.58063 | train_logloss: 0.57183 | train_accuracy: 0.70329 |
valid_logloss: 0.57703 | valid_accuracy: 0.70241 |  0:11:27s
epoch 104| loss: 0.58016 | train_logloss: 0.57144 | train_accuracy: 0.70327 |
valid_logloss: 0.57564 | valid_accuracy: 0.7023  |  0:11:34s
epoch 105| loss: 0.58046 | train_logloss: 0.57133 | train_accuracy: 0.7039  |
valid_logloss: 0.57549 | valid_accuracy: 0.70317 |  0:11:40s
epoch 106| loss: 0.58085 | train_logloss: 0.57059 | train_accuracy: 0.70481 |
valid_logloss: 0.57493 | valid_accuracy: 0.7036  |  0:11:47s
epoch 107| loss: 0.58028 | train_logloss: 0.57052 | train_accuracy: 0.70469 |
valid_logloss: 0.57504 | valid_accuracy: 0.70545 |  0:11:53s
epoch 108| loss: 0.57992 | train_logloss: 0.57029 | train_accuracy: 0.7049  |
valid_logloss: 0.57497 | valid_accuracy: 0.70458 |  0:12:00s
epoch 109| loss: 0.58003 | train_logloss: 0.57001 | train_accuracy: 0.70611 |
valid_logloss: 0.57407 | valid_accuracy: 0.70502 |  0:12:06s
epoch 110| loss: 0.57866 | train_logloss: 0.5696  | train_accuracy: 0.7063  |
valid_logloss: 0.57347 | valid_accuracy: 0.706   |  0:12:13s
epoch 111| loss: 0.57576 | train_logloss: 0.56915 | train_accuracy: 0.70513 |
valid_logloss: 0.57329 | valid_accuracy: 0.70643 |  0:12:19s
epoch 112| loss: 0.57795 | train_logloss: 0.56946 | train_accuracy: 0.70655 |
valid_logloss: 0.57298 | valid_accuracy: 0.70567 |  0:12:26s
epoch 113| loss: 0.57823 | train_logloss: 0.56875 | train_accuracy: 0.70653 |
valid_logloss: 0.57307 | valid_accuracy: 0.70665 |  0:12:32s
epoch 114| loss: 0.57778 | train_logloss: 0.56772 | train_accuracy: 0.70732 |
valid_logloss: 0.5723  | valid_accuracy: 0.70589 |  0:12:40s
epoch 115| loss: 0.57635 | train_logloss: 0.56782 | train_accuracy: 0.70721 |
valid_logloss: 0.57221 | valid_accuracy: 0.70807 |  0:12:46s
```

```
epoch 116| loss: 0.57809 | train_logloss: 0.56733 | train_accuracy: 0.70753 |
valid_logloss: 0.57224 | valid_accuracy: 0.70676 |  0:12:52s
epoch 117| loss: 0.57708 | train_logloss: 0.5674  | train_accuracy: 0.70742 |
valid_logloss: 0.57193 | valid_accuracy: 0.706   |  0:12:59s
epoch 118| loss: 0.57593 | train_logloss: 0.56691 | train_accuracy: 0.70835 |
valid_logloss: 0.57183 | valid_accuracy: 0.70872 |  0:13:05s
epoch 119| loss: 0.57632 | train_logloss: 0.56675 | train_accuracy: 0.70854 |
valid_logloss: 0.57119 | valid_accuracy: 0.70785 |  0:13:12s
epoch 120| loss: 0.5749  | train_logloss: 0.56626 | train_accuracy: 0.70858 |
valid_logloss: 0.57109 | valid_accuracy: 0.70665 |  0:13:18s
epoch 121| loss: 0.57562 | train_logloss: 0.56679 | train_accuracy: 0.70933 |
valid_logloss: 0.57219 | valid_accuracy: 0.70654 |  0:13:25s
epoch 122| loss: 0.57438 | train_logloss: 0.56577 | train_accuracy: 0.70954 |
valid_logloss: 0.57133 | valid_accuracy: 0.70665 |  0:13:31s
epoch 123| loss: 0.5737  | train_logloss: 0.56573 | train_accuracy: 0.70919 |
valid_logloss: 0.57163 | valid_accuracy: 0.706   |  0:13:38s
epoch 124| loss: 0.57468 | train_logloss: 0.56544 | train_accuracy: 0.71012 |
valid_logloss: 0.57102 | valid_accuracy: 0.70676 |  0:13:44s
epoch 125| loss: 0.57253 | train_logloss: 0.56521 | train_accuracy: 0.71022 |
valid_logloss: 0.57145 | valid_accuracy: 0.70741 |  0:13:51s
epoch 126| loss: 0.57375 | train_logloss: 0.56505 | train_accuracy: 0.71026 |
valid_logloss: 0.5717  | valid_accuracy: 0.70654 |  0:13:57s
epoch 127| loss: 0.57327 | train_logloss: 0.56469 | train_accuracy: 0.71059 |
valid_logloss: 0.57142 | valid_accuracy: 0.7073  |  0:14:04s
epoch 128| loss: 0.57264 | train_logloss: 0.56458 | train_accuracy: 0.71038 |
valid_logloss: 0.57053 | valid_accuracy: 0.70763 |  0:14:10s
epoch 129| loss: 0.5713  | train_logloss: 0.56452 | train_accuracy: 0.71122 |
valid_logloss: 0.57047 | valid_accuracy: 0.70937 |  0:14:17s
epoch 130| loss: 0.5722  | train_logloss: 0.56395 | train_accuracy: 0.71115 |
valid_logloss: 0.57044 | valid_accuracy: 0.7085  |  0:14:23s
epoch 131| loss: 0.57097 | train_logloss: 0.56358 | train_accuracy: 0.71164 |
valid_logloss: 0.57005 | valid_accuracy: 0.70796 |  0:14:30s
epoch 132| loss: 0.57341 | train_logloss: 0.56311 | train_accuracy: 0.71166 |
valid_logloss: 0.57008 | valid_accuracy: 0.70763 |  0:14:36s
epoch 133| loss: 0.57356 | train_logloss: 0.56289 | train_accuracy: 0.71225 |
valid_logloss: 0.57005 | valid_accuracy: 0.70709 |  0:14:43s
epoch 134| loss: 0.57204 | train_logloss: 0.56292 | train_accuracy: 0.71227 |
valid_logloss: 0.57041 | valid_accuracy: 0.70828 |  0:14:49s
epoch 135| loss: 0.56951 | train_logloss: 0.56275 | train_accuracy: 0.7119  |
valid_logloss: 0.56991 | valid_accuracy: 0.7085  |  0:14:56s
epoch 136| loss: 0.57313 | train_logloss: 0.56237 | train_accuracy: 0.71204 |
valid_logloss: 0.56952 | valid_accuracy: 0.70817 |  0:15:02s
epoch 137| loss: 0.5699  | train_logloss: 0.56202 | train_accuracy: 0.71276 |
valid_logloss: 0.56948 | valid_accuracy: 0.70807 |  0:15:09s
epoch 138| loss: 0.57152 | train_logloss: 0.56179 | train_accuracy: 0.71311 |
valid_logloss: 0.56908 | valid_accuracy: 0.70872 |  0:15:15s
epoch 139| loss: 0.56884 | train_logloss: 0.56142 | train_accuracy: 0.71358 |
valid_logloss: 0.5695  | valid_accuracy: 0.71024 |  0:15:22s
```

```
epoch 140| loss: 0.5691  | train_logloss: 0.56114 | train_accuracy: 0.71292 |
valid_logloss: 0.56914 | valid_accuracy: 0.71035 |  0:15:28s
epoch 141| loss: 0.56833 | train_logloss: 0.56079 | train_accuracy: 0.71379 |
valid_logloss: 0.56888 | valid_accuracy: 0.71024 |  0:15:35s
epoch 142| loss: 0.56861 | train_logloss: 0.56072 | train_accuracy: 0.71409 |
valid_logloss: 0.56893 | valid_accuracy: 0.70959 |  0:15:41s
epoch 143| loss: 0.56831 | train_logloss: 0.56046 | train_accuracy: 0.7146  |
valid_logloss: 0.56837 | valid_accuracy: 0.71079 |  0:15:48s
epoch 144| loss: 0.56798 | train_logloss: 0.55973 | train_accuracy: 0.71479 |
valid_logloss: 0.5679  | valid_accuracy: 0.7109  |  0:15:54s
epoch 145| loss: 0.56784 | train_logloss: 0.55974 | train_accuracy: 0.71393 |
valid_logloss: 0.56723 | valid_accuracy: 0.70992 |  0:16:01s
epoch 146| loss: 0.5682  | train_logloss: 0.5592  | train_accuracy: 0.71484 |
valid_logloss: 0.56804 | valid_accuracy: 0.71079 |  0:16:07s
epoch 147| loss: 0.56879 | train_logloss: 0.55945 | train_accuracy: 0.7139  |
valid_logloss: 0.568   | valid_accuracy: 0.71253 |  0:16:14s
epoch 148| loss: 0.5667  | train_logloss: 0.55867 | train_accuracy: 0.7153  |
valid_logloss: 0.56786 | valid_accuracy: 0.71394 |  0:16:20s
epoch 149| loss: 0.56508 | train_logloss: 0.55825 | train_accuracy: 0.71661 |
valid_logloss: 0.56699 | valid_accuracy: 0.71569 |  0:16:27s
epoch 150| loss: 0.56429 | train_logloss: 0.558   | train_accuracy: 0.71631 |
valid_logloss: 0.56705 | valid_accuracy: 0.71471 |  0:16:33s
epoch 151| loss: 0.56412 | train_logloss: 0.5577  | train_accuracy: 0.71523 |
valid_logloss: 0.5672  | valid_accuracy: 0.71383 |  0:16:40s
epoch 152| loss: 0.56632 | train_logloss: 0.55717 | train_accuracy: 0.71686 |
valid_logloss: 0.56677 | valid_accuracy: 0.71449 |  0:16:46s
epoch 153| loss: 0.56557 | train_logloss: 0.55724 | train_accuracy: 0.71612 |
valid_logloss: 0.56681 | valid_accuracy: 0.71362 |  0:16:53s
epoch 154| loss: 0.56397 | train_logloss: 0.55701 | train_accuracy: 0.71589 |
valid_logloss: 0.56656 | valid_accuracy: 0.71416 |  0:17:00s
epoch 155| loss: 0.56666 | train_logloss: 0.55699 | train_accuracy: 0.71693 |
valid_logloss: 0.56607 | valid_accuracy: 0.71427 |  0:17:06s
epoch 156| loss: 0.5655  | train_logloss: 0.55632 | train_accuracy: 0.71649 |
valid_logloss: 0.56555 | valid_accuracy: 0.71307 |  0:17:14s
epoch 157| loss: 0.56729 | train_logloss: 0.55637 | train_accuracy: 0.71747 |
valid_logloss: 0.56578 | valid_accuracy: 0.71503 |  0:17:22s
epoch 158| loss: 0.56503 | train_logloss: 0.55641 | train_accuracy: 0.71626 |
valid_logloss: 0.56543 | valid_accuracy: 0.71416 |  0:17:30s
epoch 159| loss: 0.56378 | train_logloss: 0.55611 | train_accuracy: 0.71638 |
valid_logloss: 0.56607 | valid_accuracy: 0.71253 |  0:17:37s
epoch 160| loss: 0.56427 | train_logloss: 0.55616 | train_accuracy: 0.7164  |
valid_logloss: 0.56532 | valid_accuracy: 0.71438 |  0:17:44s
epoch 161| loss: 0.56547 | train_logloss: 0.55593 | train_accuracy: 0.71703 |
valid_logloss: 0.56565 | valid_accuracy: 0.71394 |  0:17:51s
epoch 162| loss: 0.56345 | train_logloss: 0.55572 | train_accuracy: 0.71645 |
valid_logloss: 0.56602 | valid_accuracy: 0.71307 |  0:17:58s
epoch 163| loss: 0.56509 | train_logloss: 0.55568 | train_accuracy: 0.71642 |
valid_logloss: 0.56545 | valid_accuracy: 0.71383 |  0:18:05s
```

```
epoch 164| loss: 0.56335 | train_logloss: 0.55524 | train_accuracy: 0.71773 |
valid_logloss: 0.56549 | valid_accuracy: 0.71362 |  0:18:12s
epoch 165| loss: 0.56351 | train_logloss: 0.55531 | train_accuracy: 0.71689 |
valid_logloss: 0.56565 | valid_accuracy: 0.7122  |  0:18:19s
epoch 166| loss: 0.56378 | train_logloss: 0.55461 | train_accuracy: 0.7185  |
valid_logloss: 0.56461 | valid_accuracy: 0.71438 |  0:18:26s
epoch 167| loss: 0.56294 | train_logloss: 0.55479 | train_accuracy: 0.71775 |
valid_logloss: 0.56463 | valid_accuracy: 0.71623 |  0:18:32s
epoch 168| loss: 0.56228 | train_logloss: 0.55431 | train_accuracy: 0.71885 |
valid_logloss: 0.56467 | valid_accuracy: 0.71558 |  0:18:39s
epoch 169| loss: 0.56317 | train_logloss: 0.55411 | train_accuracy: 0.71887 |
valid_logloss: 0.56458 | valid_accuracy: 0.7159  |  0:18:46s
epoch 170| loss: 0.56188 | train_logloss: 0.55408 | train_accuracy: 0.7195  |
valid_logloss: 0.56476 | valid_accuracy: 0.71601 |  0:18:53s
epoch 171| loss: 0.56299 | train_logloss: 0.5538  | train_accuracy: 0.71854 |
valid_logloss: 0.56456 | valid_accuracy: 0.71536 |  0:19:00s
epoch 172| loss: 0.56201 | train_logloss: 0.5536  | train_accuracy: 0.71892 |
valid_logloss: 0.56441 | valid_accuracy: 0.71525 |  0:19:07s
epoch 173| loss: 0.56268 | train_logloss: 0.55387 | train_accuracy: 0.71959 |
valid_logloss: 0.56453 | valid_accuracy: 0.71688 |  0:19:14s
epoch 174| loss: 0.55995 | train_logloss: 0.55352 | train_accuracy: 0.71913 |
valid_logloss: 0.56442 | valid_accuracy: 0.71623 |  0:19:21s
epoch 175| loss: 0.56176 | train_logloss: 0.55342 | train_accuracy: 0.71908 |
valid_logloss: 0.56423 | valid_accuracy: 0.71677 |  0:19:28s
epoch 176| loss: 0.56213 | train_logloss: 0.5531  | train_accuracy: 0.71971 |
valid_logloss: 0.56417 | valid_accuracy: 0.71634 |  0:19:35s
epoch 177| loss: 0.56114 | train_logloss: 0.55299 | train_accuracy: 0.71959 |
valid_logloss: 0.56398 | valid_accuracy: 0.71569 |  0:19:42s
epoch 178| loss: 0.56012 | train_logloss: 0.55274 | train_accuracy: 0.71945 |
valid_logloss: 0.56336 | valid_accuracy: 0.71612 |  0:19:49s
epoch 179| loss: 0.56173 | train_logloss: 0.55262 | train_accuracy: 0.71878 |
valid_logloss: 0.56333 | valid_accuracy: 0.7171  |  0:19:56s
epoch 180| loss: 0.55937 | train_logloss: 0.55261 | train_accuracy: 0.71957 |
valid_logloss: 0.56395 | valid_accuracy: 0.71873 |  0:20:03s
epoch 181| loss: 0.56204 | train_logloss: 0.55238 | train_accuracy: 0.72008 |
valid_logloss: 0.56348 | valid_accuracy: 0.71699 |  0:20:09s
epoch 182| loss: 0.56038 | train_logloss: 0.55232 | train_accuracy: 0.72043 |
valid_logloss: 0.56334 | valid_accuracy: 0.71699 |  0:20:16s
epoch 183| loss: 0.56079 | train_logloss: 0.55201 | train_accuracy: 0.71978 |
valid_logloss: 0.56288 | valid_accuracy: 0.71677 |  0:20:22s
epoch 184| loss: 0.55899 | train_logloss: 0.55195 | train_accuracy: 0.72095 |
valid_logloss: 0.56283 | valid_accuracy: 0.71775 |  0:20:29s
epoch 185| loss: 0.56039 | train_logloss: 0.5516  | train_accuracy: 0.72064 |
valid_logloss: 0.56321 | valid_accuracy: 0.71819 |  0:20:36s
epoch 186| loss: 0.55954 | train_logloss: 0.55146 | train_accuracy: 0.71955 |
valid_logloss: 0.56285 | valid_accuracy: 0.71732 |  0:20:43s
epoch 187| loss: 0.55958 | train_logloss: 0.55121 | train_accuracy: 0.71987 |
valid_logloss: 0.56284 | valid_accuracy: 0.71819 |  0:20:49s
```

```
epoch 188| loss: 0.55907 | train_logloss: 0.5514  | train_accuracy: 0.7213  |
valid_logloss: 0.56208 | valid_accuracy: 0.7196  |  0:20:56s
epoch 189| loss: 0.56035 | train_logloss: 0.55108 | train_accuracy: 0.72034 |
valid_logloss: 0.56197 | valid_accuracy: 0.71873 |  0:21:03s
epoch 190| loss: 0.55964 | train_logloss: 0.55078 | train_accuracy: 0.72071 |
valid_logloss: 0.56248 | valid_accuracy: 0.71786 |  0:21:10s
epoch 191| loss: 0.55915 | train_logloss: 0.55088 | train_accuracy: 0.7202  |
valid_logloss: 0.56131 | valid_accuracy: 0.71786 |  0:21:16s
epoch 192| loss: 0.55935 | train_logloss: 0.55059 | train_accuracy: 0.72027 |
valid_logloss: 0.56148 | valid_accuracy: 0.71873 |  0:21:23s
epoch 193| loss: 0.55963 | train_logloss: 0.5504  | train_accuracy: 0.72118 |
valid_logloss: 0.56053 | valid_accuracy: 0.71852 |  0:21:30s
epoch 194| loss: 0.55674 | train_logloss: 0.55062 | train_accuracy: 0.7212  |
valid_logloss: 0.55995 | valid_accuracy: 0.71949 |  0:21:37s
epoch 195| loss: 0.55874 | train_logloss: 0.5503  | train_accuracy: 0.72139 |
valid_logloss: 0.56057 | valid_accuracy: 0.7183  |  0:21:43s
epoch 196| loss: 0.55847 | train_logloss: 0.55047 | train_accuracy: 0.72116 |
valid_logloss: 0.55976 | valid_accuracy: 0.71884 |  0:21:50s
epoch 197| loss: 0.55935 | train_logloss: 0.55026 | train_accuracy: 0.72188 |
valid_logloss: 0.55997 | valid_accuracy: 0.72091 |  0:21:57s
epoch 198| loss: 0.55959 | train_logloss: 0.54982 | train_accuracy: 0.72251 |
valid_logloss: 0.55987 | valid_accuracy: 0.72026 |  0:22:04s
epoch 199| loss: 0.55731 | train_logloss: 0.54983 | train_accuracy: 0.72223 |
valid_logloss: 0.55991 | valid_accuracy: 0.72156 |  0:22:11s
epoch 200| loss: 0.55841 | train_logloss: 0.54965 | train_accuracy: 0.72244 |
valid_logloss: 0.55951 | valid_accuracy: 0.7208  |  0:22:17s
epoch 201| loss: 0.55794 | train_logloss: 0.54942 | train_accuracy: 0.72314 |
valid_logloss: 0.55941 | valid_accuracy: 0.72211 |  0:22:24s
epoch 202| loss: 0.55726 | train_logloss: 0.5493  | train_accuracy: 0.72214 |
valid_logloss: 0.55908 | valid_accuracy: 0.72167 |  0:22:30s
epoch 203| loss: 0.55825 | train_logloss: 0.54928 | train_accuracy: 0.72307 |
valid_logloss: 0.55894 | valid_accuracy: 0.72189 |  0:22:38s
epoch 204| loss: 0.55867 | train_logloss: 0.54887 | train_accuracy: 0.72281 |
valid_logloss: 0.55855 | valid_accuracy: 0.72189 |  0:22:44s
epoch 205| loss: 0.55783 | train_logloss: 0.5486  | train_accuracy: 0.72235 |
valid_logloss: 0.55881 | valid_accuracy: 0.72222 |  0:22:51s
epoch 206| loss: 0.55767 | train_logloss: 0.54866 | train_accuracy: 0.72286 |
valid_logloss: 0.55822 | valid_accuracy: 0.71928 |  0:22:58s
epoch 207| loss: 0.55768 | train_logloss: 0.54862 | train_accuracy: 0.72263 |
valid_logloss: 0.55874 | valid_accuracy: 0.72069 |  0:23:05s
epoch 208| loss: 0.55796 | train_logloss: 0.54827 | train_accuracy: 0.72286 |
valid_logloss: 0.55901 | valid_accuracy: 0.72058 |  0:23:12s
epoch 209| loss: 0.55573 | train_logloss: 0.54799 | train_accuracy: 0.72342 |
valid_logloss: 0.55949 | valid_accuracy: 0.72135 |  0:23:19s
epoch 210| loss: 0.55558 | train_logloss: 0.54803 | train_accuracy: 0.72321 |
valid_logloss: 0.55912 | valid_accuracy: 0.72047 |  0:23:26s
epoch 211| loss: 0.5563  | train_logloss: 0.54832 | train_accuracy: 0.72305 |
valid_logloss: 0.55919 | valid_accuracy: 0.7196  |  0:23:32s
```

```
epoch 212| loss: 0.55592 | train_logloss: 0.5477  | train_accuracy: 0.7234  |
valid_logloss: 0.55827 | valid_accuracy: 0.72167 |  0:23:40s
epoch 213| loss: 0.55743 | train_logloss: 0.54821 | train_accuracy: 0.72204 |
valid_logloss: 0.55861 | valid_accuracy: 0.71884 |  0:23:47s
epoch 214| loss: 0.55527 | train_logloss: 0.54774 | train_accuracy: 0.72295 |
valid_logloss: 0.55824 | valid_accuracy: 0.722   |  0:23:55s
epoch 215| loss: 0.55586 | train_logloss: 0.54785 | train_accuracy: 0.72323 |
valid_logloss: 0.55837 | valid_accuracy: 0.72015 |  0:24:02s
epoch 216| loss: 0.55466 | train_logloss: 0.54782 | train_accuracy: 0.72349 |
valid_logloss: 0.55787 | valid_accuracy: 0.722   |  0:24:10s
epoch 217| loss: 0.55483 | train_logloss: 0.54765 | train_accuracy: 0.72279 |
valid_logloss: 0.5578  | valid_accuracy: 0.72156 |  0:24:17s
epoch 218| loss: 0.55586 | train_logloss: 0.54745 | train_accuracy: 0.72356 |
valid_logloss: 0.55806 | valid_accuracy: 0.72069 |  0:24:24s
epoch 219| loss: 0.55562 | train_logloss: 0.54731 | train_accuracy: 0.72361 |
valid_logloss: 0.55805 | valid_accuracy: 0.72091 |  0:24:31s
epoch 220| loss: 0.55551 | train_logloss: 0.54712 | train_accuracy: 0.72396 |
valid_logloss: 0.55769 | valid_accuracy: 0.72124 |  0:24:38s
epoch 221| loss: 0.55397 | train_logloss: 0.54701 | train_accuracy: 0.7237  |
valid_logloss: 0.55773 | valid_accuracy: 0.72222 |  0:24:45s
epoch 222| loss: 0.5555  | train_logloss: 0.54691 | train_accuracy: 0.72496 |
valid_logloss: 0.55832 | valid_accuracy: 0.72254 |  0:24:54s
epoch 223| loss: 0.55439 | train_logloss: 0.5469  | train_accuracy: 0.72372 |
valid_logloss: 0.55722 | valid_accuracy: 0.72298 |  0:25:01s
epoch 224| loss: 0.55556 | train_logloss: 0.5465  | train_accuracy: 0.72508 |
valid_logloss: 0.55806 | valid_accuracy: 0.72135 |  0:25:08s
epoch 225| loss: 0.55416 | train_logloss: 0.54653 | train_accuracy: 0.72459 |
valid_logloss: 0.55763 | valid_accuracy: 0.72341 |  0:25:15s
epoch 226| loss: 0.55478 | train_logloss: 0.5462  | train_accuracy: 0.7251  |
valid_logloss: 0.5581  | valid_accuracy: 0.72211 |  0:25:23s
epoch 227| loss: 0.55421 | train_logloss: 0.54643 | train_accuracy: 0.72547 |
valid_logloss: 0.55818 | valid_accuracy: 0.72233 |  0:25:31s
epoch 228| loss: 0.55434 | train_logloss: 0.54648 | train_accuracy: 0.72417 |
valid_logloss: 0.55761 | valid_accuracy: 0.72211 |  0:25:39s
epoch 229| loss: 0.55506 | train_logloss: 0.54617 | train_accuracy: 0.72589 |
valid_logloss: 0.55731 | valid_accuracy: 0.72222 |  0:25:46s
epoch 230| loss: 0.55373 | train_logloss: 0.54603 | train_accuracy: 0.72484 |
valid_logloss: 0.55759 | valid_accuracy: 0.72167 |  0:25:54s
epoch 231| loss: 0.55283 | train_logloss: 0.54585 | train_accuracy: 0.72515 |
valid_logloss: 0.5575  | valid_accuracy: 0.72233 |  0:26:01s
epoch 232| loss: 0.55462 | train_logloss: 0.54595 | train_accuracy: 0.72517 |
valid_logloss: 0.55748 | valid_accuracy: 0.72298 |  0:26:08s
epoch 233| loss: 0.55562 | train_logloss: 0.54599 | train_accuracy: 0.72554 |
valid_logloss: 0.55787 | valid_accuracy: 0.72069 |  0:26:16s
epoch 234| loss: 0.55259 | train_logloss: 0.54559 | train_accuracy: 0.7254  |
valid_logloss: 0.55798 | valid_accuracy: 0.72178 |  0:26:24s
epoch 235| loss: 0.55232 | train_logloss: 0.54554 | train_accuracy: 0.72601 |
valid_logloss: 0.5569  | valid_accuracy: 0.72276 |  0:26:31s
```

```
epoch 236| loss: 0.55357 | train_logloss: 0.54546 | train_accuracy: 0.72545 |
valid_logloss: 0.55745 | valid_accuracy: 0.72069 |  0:26:38s
epoch 237| loss: 0.55302 | train_logloss: 0.54526 | train_accuracy: 0.72554 |
valid_logloss: 0.55749 | valid_accuracy: 0.72047 |  0:26:46s
epoch 238| loss: 0.55366 | train_logloss: 0.54524 | train_accuracy: 0.72545 |
valid_logloss: 0.5574  | valid_accuracy: 0.72015 |  0:26:53s
epoch 239| loss: 0.55307 | train_logloss: 0.54498 | train_accuracy: 0.72561 |
valid_logloss: 0.55674 | valid_accuracy: 0.72211 |  0:27:01s
epoch 240| loss: 0.5529  | train_logloss: 0.54481 | train_accuracy: 0.72564 |
valid_logloss: 0.5573  | valid_accuracy: 0.7196  |  0:27:08s
epoch 241| loss: 0.55311 | train_logloss: 0.54466 | train_accuracy: 0.72564 |
valid_logloss: 0.55693 | valid_accuracy: 0.7208  |  0:27:16s
epoch 242| loss: 0.55273 | train_logloss: 0.54467 | train_accuracy: 0.72599 |
valid_logloss: 0.55657 | valid_accuracy: 0.72167 |  0:27:24s
epoch 243| loss: 0.55342 | train_logloss: 0.54456 | train_accuracy: 0.72652 |
valid_logloss: 0.55577 | valid_accuracy: 0.72135 |  0:27:31s
epoch 244| loss: 0.55248 | train_logloss: 0.54437 | train_accuracy: 0.7262  |
valid_logloss: 0.55539 | valid_accuracy: 0.72145 |  0:27:38s
epoch 245| loss: 0.55232 | train_logloss: 0.54413 | train_accuracy: 0.72594 |
valid_logloss: 0.55516 | valid_accuracy: 0.72135 |  0:27:44s
epoch 246| loss: 0.55148 | train_logloss: 0.54408 | train_accuracy: 0.72666 |
valid_logloss: 0.55529 | valid_accuracy: 0.72189 |  0:27:51s
epoch 247| loss: 0.55231 | train_logloss: 0.54411 | train_accuracy: 0.72564 |
valid_logloss: 0.55515 | valid_accuracy: 0.72145 |  0:27:57s
epoch 248| loss: 0.55229 | train_logloss: 0.54415 | train_accuracy: 0.72636 |
valid_logloss: 0.5553  | valid_accuracy: 0.72178 |  0:28:04s
epoch 249| loss: 0.55212 | train_logloss: 0.54398 | train_accuracy: 0.72601 |
valid_logloss: 0.55525 | valid_accuracy: 0.72189 |  0:28:10s
epoch 250| loss: 0.55109 | train_logloss: 0.54375 | train_accuracy: 0.72636 |
valid_logloss: 0.55473 | valid_accuracy: 0.72341 |  0:28:17s

Early stopping occurred at epoch 250 with best_epoch = 225 and
best_valid_accuracy = 0.72341
Model saved to /Users/przemek899/Desktop/Medical_prediction/jupyter_notebooks/..
/backend/Models/predict_2.pth
```

Wyświetlenie historii uczenia

[27]:
```python
model = torch.load(os.getcwd() + '/../backend/Models/predict_2.pth')
history = model.history
print(history)
```

```
{'loss': [0.9093760600904139, 0.8821707949405764, 0.8529584233353777,
0.8322830738090887, 0.8088633083715672, 0.7909796833992004, 0.7728699387573614,
0.7602867440479559, 0.7519827659537153, 0.7369150054164049, 0.7322407611986486,
0.7194922842630525, 0.7153618001356359, 0.711815620340952, 0.7039036140209292,
0.6983171061771672, 0.6921253306109733, 0.6880551881906463, 0.6833138102438392,
0.6773073615097418, 0.6764857318343186, 0.672642081249051, 0.6690819641438924,
0.6656644794999098, 0.6644307142350732, 0.6598448549828878, 0.6568897613664952,
```

0.6546546060864519, 0.6528814216939415, 0.6516752897239314, 0.6483978876253453,
0.6454116687542055, 0.64233609670546, 0.6404053205397069, 0.6389800019380525,
0.6362137227523618, 0.634892752984675, 0.6324767339520337, 0.6300696236331289,
0.6283360879595686, 0.6262319291510233, 0.62552816111867, 0.622920860604542,
0.6208486120875288, 0.6205461941114288, 0.6191119217291111, 0.6169537494822248,
0.6161522080258626, 0.6173985556858343, 0.614320084816072, 0.6138187385186915,
0.6127198323970888, 0.6097415045994085, 0.6107881025570195, 0.6080894979034982,
0.6073139964080438, 0.6074551721898522, 0.6066889050530225, 0.6056809846947833,
0.604588004147134, 0.6044270759675563, 0.6027951589444789, 0.6011556575937969,
0.6029476334409016, 0.6018385160260085, 0.6002428647948476, 0.5993826650991673,
0.5969224441342238, 0.596691543009223, 0.598796018740026, 0.5976243847753944,
0.5972810343998235, 0.5953443093997676, 0.5938663002921314, 0.5951280230429116,
0.5945053289576274, 0.5935095955685872, 0.5925453697762838, 0.5934023057542196,
0.5911845811983434, 0.5909009328702599, 0.590273826587491, 0.5885824110449814,
0.5892307453039217, 0.5903023510444457, 0.5898397346822227, 0.5882540301578801,
0.5887264507572824, 0.5871566039759938, 0.5868096206246353, 0.5845058502220526,
0.5852347408852928, 0.5856583729022886, 0.5848789622144003, 0.584688141578581 3,
0.5827585589594957, 0.5812737432921805, 0.5841235838285307, 0.5828887674866653,
0.5821088886842495, 0.5830008591093669, 0.5823467097631315, 0.580826238888066,
0.5806337478684216, 0.5801641955608274, 0.5804598113385643, 0.5808477154592186,
0.5802776057545732, 0.5799205375880729, 0.5800339594119932, 0.5786585822338011,
0.5757643304220061, 0.577947430494355, 0.5782318885733442, 0.5777778829016337,
0.5763460484946644, 0.5780864343410586, 0.5770778059959409, 0.575925877908381,
0.5763159641405431, 0.5749011737544362, 0.5756155208843511, 0.5743776414452532,
0.5737023571642433, 0.5746787466653963, 0.5725295398293473, 0.5737536215200657,
0.5732732691415927, 0.5726392414511703, 0.5712957425815303, 0.5721951606797009,
0.5709735998293248, 0.5734098117525986, 0.5735571849636916, 0.5720427065360837,
0.5695084842239938, 0.5731318142355942, 0.5699005068802254, 0.5715227679508488,
0.5688445175566325, 0.5690977050036918, 0.5683337479102902, 0.5686135917175107,
0.5683052336297384, 0.5679823276473256, 0.5678423846640238, 0.5681956424945738,
0.5687859305521337, 0.566704364811502, 0.5650806892208935, 0.5642942684452708,
0.5641193549807481, 0.5663200340619902, 0.5655697438775038, 0.5639748457001477,
0.5666569122454017, 0.5655018890776286, 0.5672887287488797, 0.5650314674144838,
0.5637816496011686, 0.5642654561414951, 0.5654718003621916, 0.5634538138785012,
0.5650934882280302, 0.5633504434329707, 0.5635127701410432, 0.5637767154995988,
0.5629430660387365, 0.562277729918317, 0.5631729320781987, 0.561878841097762,
0.5629911190126, 0.5620091237672945, 0.5626766245539593, 0.5599532272757554,
0.5617649366215962, 0.5621329691351914, 0.5611391358259248, 0.5601185880056242,
0.5617315216762265, 0.5593737160287251, 0.5620408261694556, 0.5603769029059062,
0.5607858605501129, 0.5589864879119686, 0.5603877087918723, 0.5595397571238077,
0.5595798201677276, 0.5590654582512088, 0.5603465975784674, 0.559638055359445,
0.5591456090531698, 0.5593515532772716, 0.5596310598094288, 0.5567421578779453,
0.5587395894818191, 0.5584699351613113, 0.5593454183601752, 0.5595872678407808,
0.557307689655118, 0.5584133267402648, 0.5579368399410713, 0.5572604362557573,
0.55825224155333, 0.5586660518878845, 0.5578305823046987, 0.5576688705421078,
0.5576789001139197, 0.5579639062648867, 0.555731888224439, 0.5555779599561924,
0.5562998070949461, 0.5559188985243076, 0.557431258806368, 0.5552691919047658,
0.5558572324310862, 0.5546557089177572, 0.5548301469988941, 0.5558585190191503,

0.5556177991192514, 0.5555070086223324, 0.5539665919978444, 0.5555001642645859,
0.5543916210895632, 0.555557726359949, 0.5541646960305003, 0.5547839388614748,
0.5542149311158715, 0.5543424865094627, 0.5550586959210838, 0.5537327324471821,
0.5528327531930877, 0.5546200362647452, 0.5556160720383249, 0.552589130110857,
0.5523249608714406, 0.5535650267833616, 0.5530241422536897, 0.5536554484832578,
0.5530698561086888, 0.5528952261296716, 0.5531119762397394, 0.5527320911244649,
0.5534192992419731, 0.5524797337811168, 0.5523191588680917, 0.5514816569118967,
0.5523088559871768, 0.5522920939980482, 0.5521158488785347, 0.551088216828137],
'lr': [0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001,
0.0001, 0.0001], 'train_logloss': [0.7941985271132274, 0.749802791030264l,
0.7422296862216812, 0.7460223736040557, 0.7525055399509372, 0.7473858169994814,
0.7411008959665475, 0.7318614980333664, 0.7251086238541501, 0.7183678699667997,
0.7102755038643139, 0.7025554739358316, 0.6990181375221717, 0.6935671335523841,
0.687847091262508, 0.6838879704312923, 0.6786409781561861, 0.6740448416084384,
0.6704264331573349, 0.6666758635625261, 0.6638979555774432, 0.6603082560499988,
0.6565732965741098, 0.6535197426348487, 0.6513054145108973, 0.648429836379343,
0.6455870301324181, 0.6431688396729811, 0.6405748169270948, 0.6379120033170164,
0.6359638681620049, 0.6331349155512778, 0.6307227257924851, 0.6293395302948168,
0.627247288424026, 0.6252797796413293, 0.6220104580113094, 0.6206389790299717,
0.6181510487557376, 0.6176703847452821, 0.6152969673376548, 0.6135482056149996,
0.6120577377020122, 0.6104197086544709, 0.6094002111478218, 0.6082828531241186,
0.6077421409235646, 0.6062903385848067, 0.6044931141946095, 0.6041284497620111,
0.6030083115262067, 0.6012143080141517, 0.6006024647397462, 0.599465739514479,
0.5987231355998509, 0.5979581043884585, 0.5971286761115464, 0.5960772010505209,

0.5955017567846175, 0.5948153515962883, 0.5938583728908589, 0.5928248860744662,
0.5923919835976631, 0.5912715573564545, 0.5899521601098685, 0.5894374250377457,
0.588972459366708, 0.5884128666644567, 0.5871347210173011, 0.5865122957426571,
0.586670436098646, 0.5858837983021798, 0.5847813300140062, 0.5845004501635419,
0.5840687262965165, 0.5837304637415452, 0.5830835018271968, 0.5824814184561825,
0.5814701335651129, 0.5814512784496713, 0.5812848982378712, 0.5809815489408301,
0.5800960991432086, 0.579830320795385, 0.5797191636242786, 0.578982769317406,
0.5789993720088564, 0.5782610267805924, 0.578070659568022, 0.5779876138431699,
0.5767410937822802, 0.5764819359239352, 0.5759865551676957, 0.5755969283756401,
0.5753906460319982, 0.5750115851077626, 0.5744212140493423, 0.5743926624271187,
0.5738510531904741, 0.5736086828339017, 0.5733025663420579, 0.5728756775426294,
0.5729299831239003, 0.5718328854859885, 0.5714422842667988, 0.5713322499486585,
0.570591222452727, 0.5705175311451749, 0.5702919280880774, 0.5700125257707112,
0.5695956392022966, 0.5691469363152029, 0.5694569900022096, 0.5687461796730388,
0.5677150992542992, 0.5678227699265196, 0.5673330560865103, 0.5674019384563912,
0.5669099777665203, 0.5667526732467401, 0.5662629769696161, 0.566794700139079,
0.5657698837556497, 0.5657325036842082, 0.5654430268285522, 0.5652054119535082,
0.5650533431189007, 0.564692029678077‌1, 0.5645814803919509, 0.5645231259454399,
0.563948000607393, 0.5635783722496915, 0.5631063805073089, 0.5628917019765279,
0.5629153323723348, 0.5627539585179263, 0.5623707514502184, 0.5620209049753495,
0.561791041977977, 0.5614185021040081, 0.5611407716430078, 0.560787564508073,
0.560722647240032, 0.5604569594061741, 0.5597327676962885, 0.5597354317479964,
0.5592012282007192, 0.5594548509834155, 0.5586658453752599, 0.5582507125626532,
0.5579982569469794, 0.5576953506127676, 0.5571749881653201, 0.557237216693035,
0.55701335616299, 0.5569894180599751, 0.5563244288848276, 0.556367938954212,
0.5564083431015713, 0.5561078721883109, 0.556158759487131, 0.555927309480508,
0.5557171811065725, 0.5556811283902598, 0.5552409738685264, 0.5553089309725718,
0.5546089966647452, 0.5547860582885398, 0.554305625207266, 0.5541063447195302,
0.5540831455062208, 0.5537988657910292, 0.5536009554618404, 0.5538671964491972,
0.5535240832898305, 0.5534219388350876, 0.5531030789176009, 0.5529905945817539,
0.552740715402853, 0.5526153645301096, 0.5526082452441631, 0.5523781468869833,
0.5523181508135969, 0.5520095918624879, 0.5519476934689145, 0.5515995566030699,
0.5514628547607514, 0.5512097478400414, 0.5514043677019695, 0.551078039575927,
0.5507791281927893, 0.5508832614342758, 0.5505885954706692, 0.5503955837138232,
0.5506171355925321, 0.5502953415150885, 0.5504709037544968, 0.5502648195136081,
0.5498155915262939, 0.5498339121860806, 0.5496510834025387, 0.5494206918135224,
0.5493011157379263, 0.5492829957699839, 0.5488731201217901, 0.5486018651330268,
0.5486569753133475, 0.5486190161746384, 0.5482734638990997, 0.5479897106870558,
0.548029486859977, 0.5483218207317554, 0.5476960127378243, 0.5482145772937337,
0.5477432557810216, 0.5478532892260387, 0.5478158000477116, 0.5476519569181785,
0.5474520003568465, 0.5473110690173638, 0.5471240908085233, 0.5470126105256713,
0.5469110742279646, 0.546895500335192, 0.5464962166083462, 0.5465306143731431,
0.5462005386367501, 0.5464327910628765, 0.5464801110312479, 0.5461665168299452,
0.5460300087866495, 0.5458451242623265, 0.5459487126231974, 0.5459888916923874,
0.5455853955730056, 0.5455384318534467, 0.545455339747793, 0.5452626485727019,
0.5452428227467006, 0.5449835175483826, 0.5448142014222166, 0.5446637861748558,
0.5446721395309947, 0.5445575243214472, 0.5443738518358712, 0.5441273263482623,
0.5440781887741704, 0.5441086956510204, 0.5441501011550015, 0.5439834705946475,

0.5437506623470006], 'train_accuracy': [0.47247492418940984,
0.48091905761604853, 0.4773268019594122, 0.47830650804758573,
0.47429437835316074, 0.47926288780032655, 0.48787030557499417,
0.4920690459528808, 0.5034056449731747, 0.5125728947982272, 0.5266620013995801,
0.5332633543270352, 0.5445766270118965, 0.5522743177046886, 0.5648472125029158,
0.569559132260322, 0.5804991835782598, 0.5896664334033124, 0.5979006298110566,
0.6044553300676464, 0.6088640074644274, 0.613692558898997, 0.619034289713086,
0.6234896197807325, 0.6255423372988104, 0.6313039421506881, 0.6371588523442967,
0.6383018427804992, 0.6416608350828085, 0.6438768369489153, 0.6476090506181479,
0.6497783998133894, 0.654910193608584, 0.654910193608584, 0.6595754606951248,
0.6609283881502216, 0.6628878003265687, 0.664753907161185, 0.666480055983205,
0.6675530674131094, 0.6697224166083509, 0.6715651971075344, 0.672614882202006,
0.674037788663401, 0.674107767669699, 0.6748308840681129, 0.675693958479123,
0.676393748542104, 0.6799626778633077, 0.6777700023326335, 0.6811289946349428,
0.6824819220900397, 0.6815022160018661, 0.6823419640774434, 0.6829484487986938,
0.6847212502915792, 0.6840447865640308, 0.6844646606018194, 0.6856309773734546,
0.6865407044553301, 0.6868439468159552, 0.6884534639608118, 0.688593421973408,
0.6890366223466293, 0.6899463494285047, 0.6902495917891299, 0.69111266620014,
0.6906927921623512, 0.6923956146489386, 0.6928621413575927, 0.6912759505481689,
0.6930954047119198, 0.6923956146489386, 0.6943550268252857, 0.6938885001166317,
0.693398647072545, 0.6942150688126896, 0.6940284581292279, 0.6951714485654303,
0.6961278283181712, 0.6959645439701423, 0.6965943550268253, 0.6974107767669699,
0.6965477023559599, 0.6972708187543737, 0.6980405878236529, 0.6967576393748542,
0.6988803358992303, 0.698740377886634, 0.6988103568929321, 0.6991602519244227,
0.69913692558899, 0.6999300209937019, 0.700629811056683, 0.7003265686960578,
0.7004665267086541, 0.7006997900629811, 0.7017728014928855, 0.7012362957779333,
0.7019360858409144, 0.7015162118031257, 0.7026125495684628, 0.7023326335432704,
0.7032890132960112, 0.7032656869605785, 0.7038954980172615, 0.7048052250991369,
0.7046885934219734, 0.7048985304408677, 0.7061114998833683, 0.70629811056683,
0.7051317937951948, 0.7065547002565897, 0.706531373921157, 0.7073244693258689,
0.7072078376487054, 0.7075344063447633, 0.7074177746675997, 0.7083508280849079,
0.7085374387683695, 0.7085840914392348, 0.7093305341730815, 0.7095404711919757,
0.7091905761604852, 0.7101236295777933, 0.7102169349195241, 0.7102635875903895,
0.7105901562864474, 0.710380219267553, 0.7112199673431304, 0.7111499883368323,
0.7116398413809191, 0.7116631677163517, 0.7122463261021693, 0.7122696524376021,
0.7118964310706788, 0.712036389083275, 0.7127595054816889, 0.7131094005131794,
0.7135759272218335, 0.7129227898297178, 0.7137858642407278, 0.7140891066013529,
0.7146022859808724, 0.714788896664334, 0.713925822253324, 0.7148355493351994,
0.7139024959178913, 0.7153020760438535, 0.7166083508280849, 0.7163051084674598,
0.7152320970375554, 0.7168649405178447, 0.7161184977839982, 0.7158852344296711,
0.7169349195241428, 0.7164917191509214, 0.7174714252390949, 0.7162584557965943,
0.7163750874737579, 0.7163984138091906, 0.7170282248658736, 0.716445066480056,
0.7164217401446232, 0.7177280149288546, 0.7168882668532773, 0.7184977839981339,
0.7177513412642874, 0.7188476790296244, 0.7188710053650571, 0.7195008164217401,
0.7185444366689993, 0.7189176580359226, 0.7195941217634709, 0.7191275950548169,
0.7190809423839515, 0.7197107534406345, 0.7195941217634709, 0.7194541637508748,
0.7187777000233263, 0.7195707954280383, 0.7200839748075577, 0.7204338698390483,
0.7197807324469326, 0.7209470492185678, 0.7206438068579426, 0.7195474690926056,

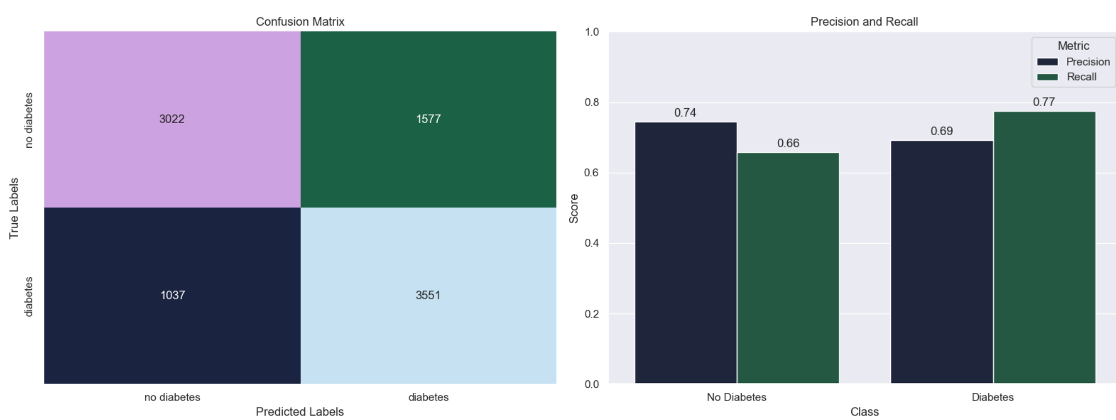
0.7198740377886634, 0.7212969442500583, 0.7203405644973174, 0.7207137858642407, 0.7202006064847213, 0.7202705854910194, 0.7211803125728948, 0.7212036389083275, 0.7213902495917891, 0.721156986237462, 0.7218801026358759, 0.7225099136925589, 0.7222299976673665, 0.7224399346862608, 0.7231397247492419, 0.7221366923256356, 0.7230697457429438, 0.7228131560531841, 0.72234662934453, 0.7228598087240494, 0.7226265453697224, 0.7228598087240494, 0.7234196407744343, 0.7232097037555401, 0.7230464194075111, 0.7233963144390017, 0.7220433869839048, 0.7229531140657802, 0.7232330300909727, 0.7234896197807325, 0.7227898297177513, 0.7235595987870306, 0.7236062514578959, 0.7239561464893866, 0.7236995567996268, 0.7249591789129928, 0.7237228831350595, 0.7250758105901562, 0.7245859575460695, 0.725099136925589, 0.7254723582925122, 0.7241660835082808, 0.7258922323303009, 0.7248425472358293, 0.7251457895964544, 0.7251691159318872, 0.7255423372988103, 0.7254023792862141, 0.7260088640074644, 0.7254490319570795, 0.7255423372988103, 0.7254490319570795, 0.7256123163051085, 0.7256356426405411, 0.7256356426405411, 0.7259855376720318, 0.7265220433869839, 0.726195474690926, 0.7259388850011663, 0.7266620013995801, 0.7256356426405411, 0.726358759038955, 0.7260088640074644, 0.726358759038955],
'valid_logloss': [0.7894460907549933, 0.7524098689188868, 0.743445382184081, 0.7496167837491862, 0.756574826904691, 0.7485084950343038, 0.7420697115172599, 0.7317382550808511, 0.7246865914422216, 0.7183599761342725, 0.7097889828476729, 0.7040510609809172, 0.7021135480364745, 0.6962567135464017, 0.691721168772207, 0.6870597712509822, 0.6823840695325476, 0.6774520311523985, 0.6741980541754328, 0.6690830889529001, 0.6681540683037333, 0.6656296973700985, 0.6615580439350606, 0.6573552264504694, 0.6558802452470596, 0.6517263182563643, 0.6495258108158832, 0.6469995600132051, 0.6444947288439844, 0.6419085907112795, 0.638894428437908, 0.6363268543988211, 0.634151675457459, 0.6313745656319099, 0.6295854146067729, 0.6270088110882039, 0.6242188979908796, 0.6228489841344406, 0.6199794929664826, 0.6193283858543777, 0.6171141623930483, 0.6165617859411684, 0.6142876631348219, 0.6126495914870284, 0.611718038195358, 0.610015034008218, 0.6084820435917477, 0.6071483935997565, 0.6059238975004523, 0.6053920417954575, 0.6039043624429751, 0.602524023103822, 0.6015460420156558, 0.6005383755366748, 0.5999668399459304, 0.5986509826076541, 0.5977880792351312, 0.5975885301050253, 0.5970666914338995, 0.5965183393043516, 0.5951915327488271, 0.5944758906960017, 0.5940632521006561, 0.592929726688655, 0.5926320988169886, 0.5910015938362714, 0.5907669034659193, 0.5910018551794538, 0.5903050727064796, 0.5893423183585044, 0.5898575573462499, 0.5889250601055971, 0.5881653185685107, 0.5881144427786624, 0.5873969725607975, 0.5867583839434622, 0.5857727487251194, 0.5854083604085476, 0.5852314732128292, 0.5849816680332853, 0.583850901702429, 0.5829114313321632, 0.5829734976157845, 0.5827794205434117, 0.5829258755273763, 0.582641435928511, 0.5821786656665389, 0.58129133449627, 0.5811545466949074, 0.5809125306688944, 0.5802630623887023, 0.5795332201124471, 0.5793315817074923, 0.5789054319950111, 0.5781795327546186, 0.5782319101654058, 0.5776807697295641, 0.577793128658095, 0.5770746877712728, 0.5770463234461026, 0.5763269715527827, 0.5766134049000963, 0.5762498872201057, 0.5770258018443714, 0.5756383373152409, 0.5754947079684016, 0.5749349248478144, 0.5750411335073837, 0.574974952059917, 0.5740714048190083, 0.5734677074800643, 0.5732914327582487, 0.5729766826439862, 0.5730701124643979, 0.5722986597574393, 0.5722064696530311, 0.5722391963023866, 0.57192862189805, 0.571833782465742, 0.5711909157379194, 0.571092832067678, 0.5721866262275888, 0.5713308840087421, 0.5716297968332249, 0.5710231599014962, 0.5714543703217663, 0.5716951819765581,

0.5714183116030033, 0.570534227053173, 0.570468062804331, 0.5704405667485396,
0.5700526637867873, 0.5700829870000307, 0.5700549944938584, 0.5704129260637664,
0.569914080547046, 0.5695233190028727, 0.5694787092556837, 0.5690844034938793,
0.569504420927923, 0.5691359762179345, 0.5688838271554787, 0.5689250137187858,
0.5683732796276315, 0.5678996528613944, 0.5672322609476059, 0.5680354445241146,
0.5679951859194512, 0.5678556454345418, 0.5669876557382465, 0.567047847440369,
0.5671989581661424, 0.5667676107877635, 0.56680770869486, 0.5665620538806706,
0.5660673752984827, 0.5655490783761852, 0.5657752460598166, 0.5654315094699144,
0.5660687713792623, 0.5653181760763821, 0.5656538498061431, 0.5660216355516104,
0.565451500037529, 0.565490716962757, 0.565654937331139, 0.5646133273888724,
0.5646280652492702, 0.5646680834998254, 0.5645804104414748, 0.564762657867996,
0.5645624593184351, 0.5644144821550726, 0.564527677207133, 0.5644231221210328,
0.564225049017911, 0.5641700291312581, 0.5639824434886691, 0.5633616306109822,
0.563326545225496, 0.5639455929066469, 0.5634824523898033, 0.5633419087739258,
0.5628841681623826, 0.5628278893075045, 0.563206999694606, 0.5628475150852282,
0.5628406391786249, 0.56208255676279, 0.5619734747082435, 0.5624785491995302,
0.5613126368190912, 0.5614813739637441, 0.5605258681597604, 0.5599538214503569,
0.5605684850188394, 0.559759154025137, 0.5599650110224, 0.5598741429818646,
0.5599135909578287, 0.5595147193626575, 0.5594063579947828, 0.5590768720617181,
0.5589405881139184, 0.5585478653503911, 0.5588106876572095, 0.558218114294314,
0.5587426694098488, 0.5590123236647444, 0.5594864524566827, 0.5591239267966059,
0.5591868896468126, 0.558270342867635, 0.5586089507124952, 0.5582382558542172,
0.5583655469133716, 0.557872668940237, 0.5577955836100855, 0.558056749023813,
0.5580485387092078, 0.5576935557867703, 0.5577285500878639, 0.5583222692352696,
0.557220088081653, 0.5580587292498659, 0.5576348482224759, 0.5580998617725725,
0.5581781711981787, 0.55761403365785, 0.5573076101808291, 0.5575911819700181,
0.5574968066930398, 0.5574816174004411, 0.5578685603073131, 0.557980364339617,
0.5569024924731566, 0.5574509202653368, 0.5574941303224368, 0.5574000970449159,
0.5567419499881655, 0.5572978906912734, 0.5569336919721807, 0.5565672937627096,
0.5557677603809706, 0.555386114889388, 0.5551633822989241, 0.5552908724672965,
0.5551461565867403, 0.5553033136505546, 0.5552470852730571, 0.5547317469831794],
'valid_accuracy': [0.47534559703929463, 0.4737128551213671, 0.483400457167737,
0.474910199194514, 0.47643409165124634, 0.47948187656471103, 0.486665941003592,
0.49352345705888756, 0.5044084031784043, 0.517905736366605, 0.5328181125503428,
0.536192445847393, 0.5458800478937629, 0.5527375639490585, 0.5612278219222815,
0.5671056928268206, 0.5767932948731904, 0.584412757156852, 0.5892021334494394,
0.5989985849570044, 0.6026994666376402, 0.5998693806465658, 0.6086861870033743,
0.6148906062914988, 0.6175029933601829, 0.624795907260259, 0.6275171437901382,
0.6274082943289431, 0.6338304125394579, 0.6342658103842386, 0.6400348318275825,
0.6450419070425601, 0.6477631435724394, 0.6487427887231958, 0.6546206596277349,
0.6583215413083705, 0.6594100359203222, 0.6635463154457385, 0.6656144552084468,
0.6673560465875694, 0.6701861325786438, 0.6675737455099597, 0.6731250680309132,
0.6738870142592794, 0.6737781647980843, 0.6767171002503538, 0.6794383367802329,
0.6798737346250137, 0.6791117883966474, 0.6803091324697943, 0.6837923152280396,
0.6856427560683575, 0.6845542614564057, 0.6846631109176009, 0.6876020463698704,
0.6882551431370415, 0.688581691520627, 0.6882551431370415, 0.6899967345161642,
0.6893436377489931, 0.6892347882877979, 0.6887993904430173, 0.6892347882877979,
0.687928594753456, 0.6890170893654076, 0.6880374442146512, 0.6894524872101883,

0.6906498312833351, 0.6893436377489931, 0.6906498312833351, 0.6896701861325787,
0.6907586807445303, 0.6899967345161642, 0.6923914226624578, 0.6901055839773593,
0.6932622183520192, 0.6928268205072385, 0.6927179710460433, 0.692500272123653,
0.6916294764340917, 0.6939153151191901, 0.696418852726679, 0.6941330140415805,
0.6952215086535322, 0.6976161967998259, 0.6943507129639708, 0.696418852726679,
0.696418852726679, 0.6959834548818984, 0.6945684118863612, 0.6972896484162403,
0.6958746054207031, 0.6987046914117775, 0.6988135408729726, 0.6999020354849244,
0.7007728311744856, 0.7008816806356809, 0.7005551322520953, 0.702296723631218,
0.7025144225536084, 0.7015347774028519, 0.7017524763252422, 0.7030586698595842,
0.7024055730924131, 0.702296723631218, 0.7031675193207794, 0.70360291716556,
0.7054533580058778, 0.7045825623163166, 0.7050179601610972, 0.7059976053118537,
0.7064330031566344, 0.7056710569282683, 0.7066507020790247, 0.7058887558506586,
0.7080657450745619, 0.7067595515402199, 0.7059976053118537, 0.7087188418417328,
0.7078480461521716, 0.7066507020790247, 0.7065418526178295, 0.7066507020790247,
0.7059976053118537, 0.7067595515402199, 0.7074126483073909, 0.7065418526178295,
0.7073037988461958, 0.7076303472297812, 0.7093719386089039, 0.7085011429193425,
0.7079568956133667, 0.7076303472297812, 0.7070860999238053, 0.7082834439969522,
0.7085011429193425, 0.708174594535757, 0.7080657450745619, 0.7087188418417328,
0.7102427342984652, 0.7103515837596603, 0.7102427342984652, 0.7095896375312942,
0.7107869816044411, 0.7108958310656363, 0.7099161859148797, 0.7107869816044411,
0.7125285729835638, 0.7139436159791009, 0.7156852073582236, 0.7147055622074671,
0.7138347665179058, 0.7144878632850767, 0.7136170675955154, 0.7141613149014913,
0.714270164362 6864, 0.7130728202895396, 0.7150321105910525, 0.7141613149014913,
0.7125285729835638, 0.7143790138238816, 0.7139436159791009, 0.7130728202895396,
0.7138347665179058, 0.7136170675955154, 0.7122020245999783, 0.7143790138238816,
0.7162294546641994, 0.7155763578970284, 0.7159029062806139, 0.7160117557418091,
0.7153586589746381, 0.7152498095134429, 0.7168825514313704, 0.7162294546641994,
0.7167737019701752, 0.7163383041253946, 0.7156852073582236, 0.7161206052030042,
0.7171002503537608, 0.7187329922716883, 0.7169914008925656, 0.7169914008925656,
0.7167737019701752, 0.7177533471209318, 0.7181887449657124, 0.7173179492761511,
0.7181887449657124, 0.7196037879612496, 0.7187329922716883, 0.7178621965821269,
0.7178621965821269, 0.7187329922716883, 0.7185152933492979, 0.7194949385000544,
0.7182975944269075, 0.7188418417328835, 0.7209099814955916, 0.7202568847284206,
0.7215630782627626, 0.7208011320343964, 0.7221073255687385, 0.7216719277239577,
0.7218896266463481, 0.7218896266463481, 0.7222161750299336, 0.7192772395776641,
0.7206922825732013, 0.7205834331120061, 0.7213453793403722, 0.720474583650811,
0.7196037879612496, 0.7216719277239577, 0.7188418417328835, 0.7219984761075433,
0.7201480352672254, 0.7219984761075433, 0.7215630782627626, 0.7206922825732013,
0.7209099814955916, 0.7212365298791771, 0.7222161750299336, 0.7225427234135191,
0.7229781212582997, 0.7213453793403722, 0.7234135191030805, 0.7221073255687385,
0.7223250244911288, 0.7221073255687385, 0.7222161750299336, 0.7216719277239577,
0.7223250244911288, 0.7229781212582997, 0.7206922825732013, 0.721780777185153,
0.7227604223359094, 0.7206922825732013, 0.720474583650811, 0.7201480352672254,
0.7221073255687385, 0.7196037879612496, 0.7208011320343964, 0.7216719277239577,
0.7213453793403722, 0.7214542288015674, 0.7213453793403722, 0.7218896266463481,
0.7214542288015674, 0.721780777185153, 0.7218896266463481, 0.7234135191030805]}

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

[34]:
```python
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

y_pred = model.predict(X_test)

classification_report_test = classification_report(y_test, y_pred, output_dict=True)

true_labels = np.array(y_test)
predictions = np.array(y_pred)
cm = confusion_matrix(true_labels, predictions)

plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap=sns.color_palette("cubehelix"),
            xticklabels=['no diabetes', 'diabetes'],
            yticklabels=['no diabetes', 'diabetes'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

print(classification_report_test)
```



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

```
[34]: 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": "Diabetes_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(history["valid_accuracy"][-1], 2),
    "loss": round(history["loss"][-1], 2),
    "date_inserted": datetime.now(),
    "train_samples": len(X_train),
    "val_samples": len(X_val),
    "test_samples": len(X_test),
}

collection.insert_one(document)
```

```
[34]: <pymongo.results.InsertOneResult at 0x140022a40>
```

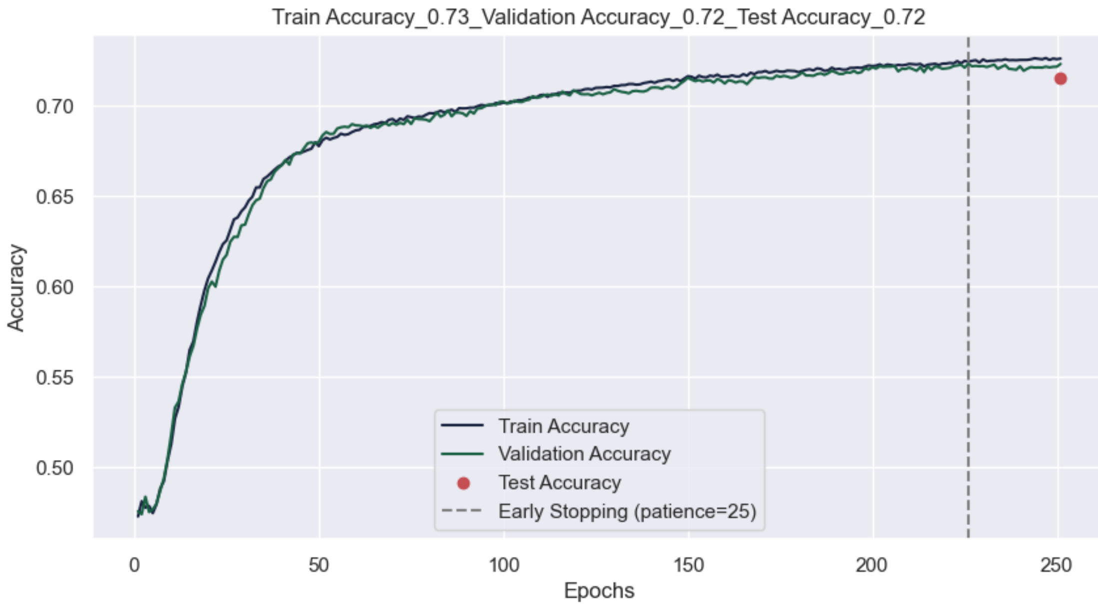
Zdefiniowanie odpowiednich zmiennych na potrzeby obliczeń przeprowadzanych przez backend aplikacji:

```
[35]: import os

train_samples = len(X_train)
val_samples = len(X_val)
test_samples = len(X_test)

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['DIABETES_TRAIN'] = train_samples
env_vars['DIABETES_VAL'] = val_samples
env_vars['DIABETES_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 valid_accuracy po szybkim wzroście ustabilizował się na pułapie ok 72 % dokładności. Wydaje się być on górna granica możliwości modelu z uwagi na niemedyczny i często subiektywny charakter danych wejściowych.

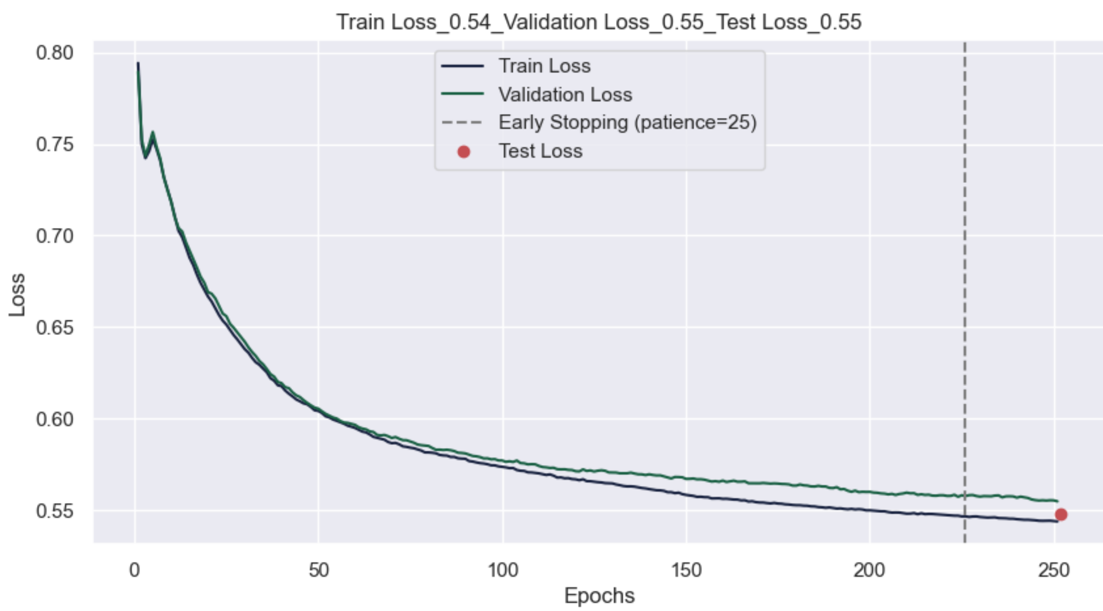
[36]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

val_accuracies = history['valid_accuracy']
train_accuracies = history['train_accuracy']

y_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

stopping_epoch = len(val_accuracies) - patience if len(val_accuracies) >␣
 ↪patience else len(val_accuracies)

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(val_accuracies)], [test_accuracy], 'ro', label='Test Accuracy')

if stopping_epoch > 0:
    plt.axvline(x=stopping_epoch, color='gray', linestyle='--', label=f'Early␣
 ↪Stopping (patience={patience})')

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()
```



Strata na zbiorze walidacyjnym w kolejnych epokach systematycznie maleje w sposób hiperboliczny, zbliżając się asymptotycznie do wartości wynoszącej około 0,55

[37]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import log_loss

y_pred_proba = model.predict_proba(X_test)
test_loss = log_loss(y_test, y_pred_proba)

train_loss = history['train_logloss']
val_loss = history['valid_logloss']

stopping_epoch = len(val_loss) - patience if len(val_loss) > patience else␣
↪len(val_loss)

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

plt.plot(range(1, len(train_loss) + 1), train_loss, label='Train Loss')

plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss')

if stopping_epoch > 0:
    plt.axvline(x=stopping_epoch, color='gray', linestyle='--', label=f'Early␣
↪Stopping (patience={patience})')

plt.plot(len(val_loss) + 1, test_loss, 'ro', label='Test Loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.title(f"Train Loss_{train_loss[-1]:.2f}_Validation Loss_{val_loss[-1]:.
↪2f}_Test Loss_{test_loss:.2f}")

plt.show()
```

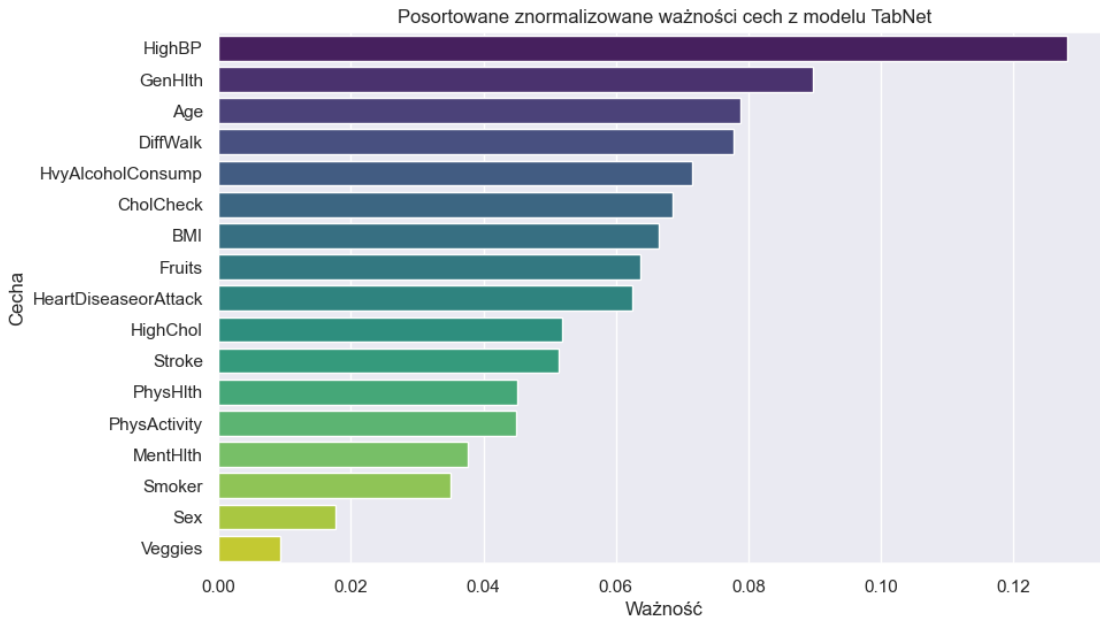


Wykres ważności poszczególnych cech modelu TabNet dla pierwszego, początkowego kroku decyzyjnego

[56]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from pytorch_tabnet.tab_model import TabNetClassifier

model = torch.load(os.getcwd() + '/../backend/Models/predict_2.pth')

X_test_tensor = torch.tensor(X_test, dtype=torch.float32)

masks, aggregated_importance = model.explain(X_test_tensor)

importance_values_all_steps = []
```

```
for step in range(3):
    importance_values = aggregated_importance.get(step)
    if importance_values is not None and isinstance(importance_values, np.
 ↪ndarray):
        if importance_values.ndim == 2:
            importance_values = importance_values.mean(axis=0)
        importance_values_all_steps.append(importance_values)
    else:
        print(f"Error: Unable to extract valid importance values for step␣
 ↪{step}.")

if importance_values_all_steps:
    total_importance = np.mean(importance_values_all_steps, axis=0)

    total_importance_normalized = total_importance / total_importance.sum()

    features = [
        'Age', 'Sex', 'HighChol', 'CholCheck', 'BMI', 'Smoker',␣
 ↪'HeartDiseaseorAttack',
        'HvyAlcoholConsump', 'PhysActivity', 'GenHlth', 'MentHlth', 'Fruits',␣
 ↪'Veggies',
        'PhysHlth', 'DiffWalk', 'Stroke', 'HighBP'
    ]

    if len(features) == len(total_importance_normalized):
        feature_importance_pairs = list(zip(total_importance_normalized,␣
 ↪features))

        feature_importance_pairs.sort(reverse=True, key=lambda x: x[0])

        sorted_importance, sorted_features = zip(*feature_importance_pairs)

        plt.figure(figsize=(10, 6))
        sns.barplot(x=sorted_importance, y=sorted_features, palette="viridis")
        plt.title("Posortowane znormalizowane ważności cech z modelu TabNet")
        plt.xlabel("Ważność")
        plt.ylabel("Cecha")
        plt.show()

    else:
        print("Error: The number of features does not match the length of␣
 ↪importance values.")
else:
    print("Error: No valid importance values found for any step.")
```

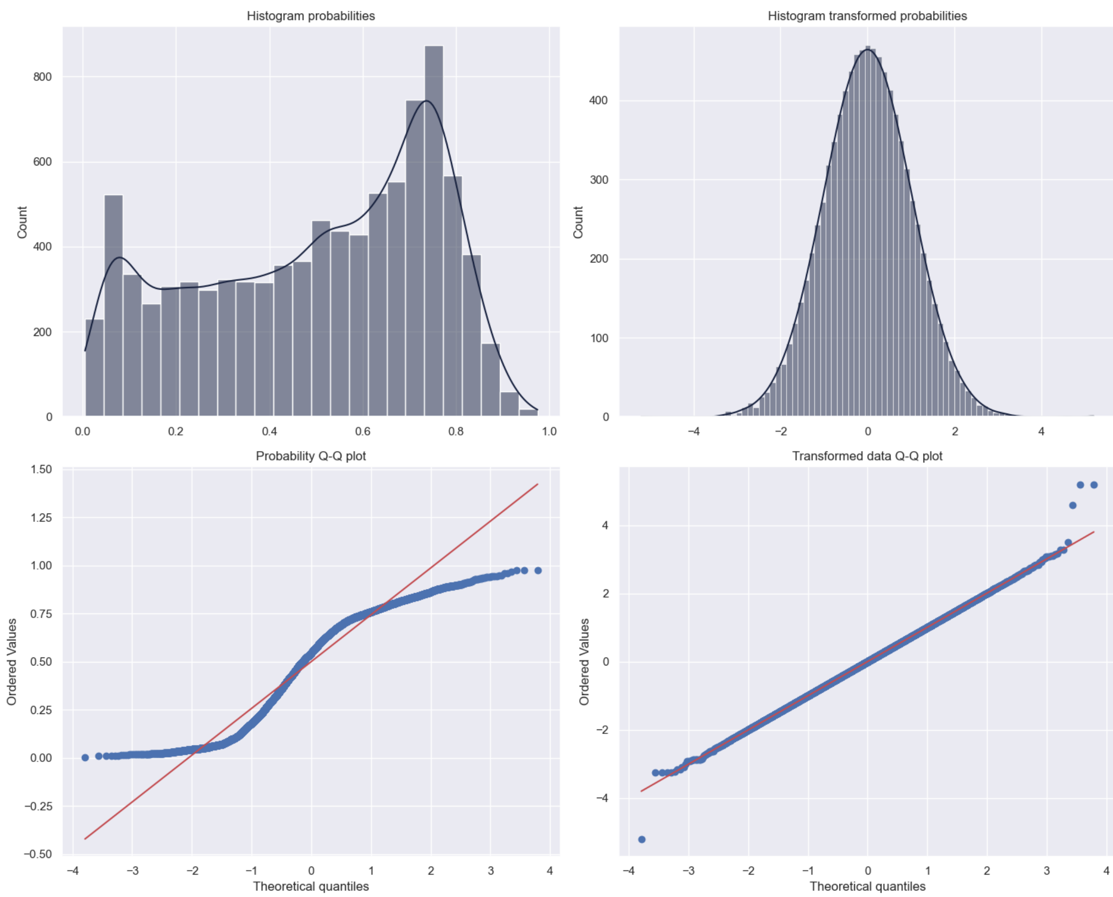
```
Suma znormalizowanych ważności cech: 1.0
```



Po wytrenowaniu modelu, możemy wybrać np. 10 pacjentów z zbioru testowego (połowa z nich będzie chora, a druga połowa zdrowa):

[42]:
```
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 = pd.DataFrame(X_test)

outputs = model.predict_proba(inputs.values)

probabilities = torch.tensor(outputs[:, 1])

probabilities = probabilities.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.title("Histogram probabilities")

plt.subplot(2, 2, 2)
sns.histplot(transformed_data, kde=True)
plt.title("Histogram transformed probabilities")

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

plt.subplot(2, 2, 4)
stats.probplot(transformed_data, dist="norm", plot=plt)
plt.title("Transformed data Q-Q plot")

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



I sprawdzić jakie prawdopodobieństwo występowania choroby wraz z określonymi przedziałami ufności przypisze im wytrenowany model:

[59]:
```python
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_2.pth')

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

outputs = model.predict_proba(df_XY.drop('Diabetes', axis=1).values)

probabilities = outputs[:, 1]

transformer = QuantileTransformer(n_quantiles=10, output_distribution='normal')
transformed_data = transformer.fit_transform(probabilities.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 * 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['Diabetes'] == 1].sample(5, random_state=42)
samples_without_sick = df_XY[df_XY['Diabetes'] == 0].sample(5, random_state=42)
samples = pd.concat([samples_with_sick, samples_without_sick])
```

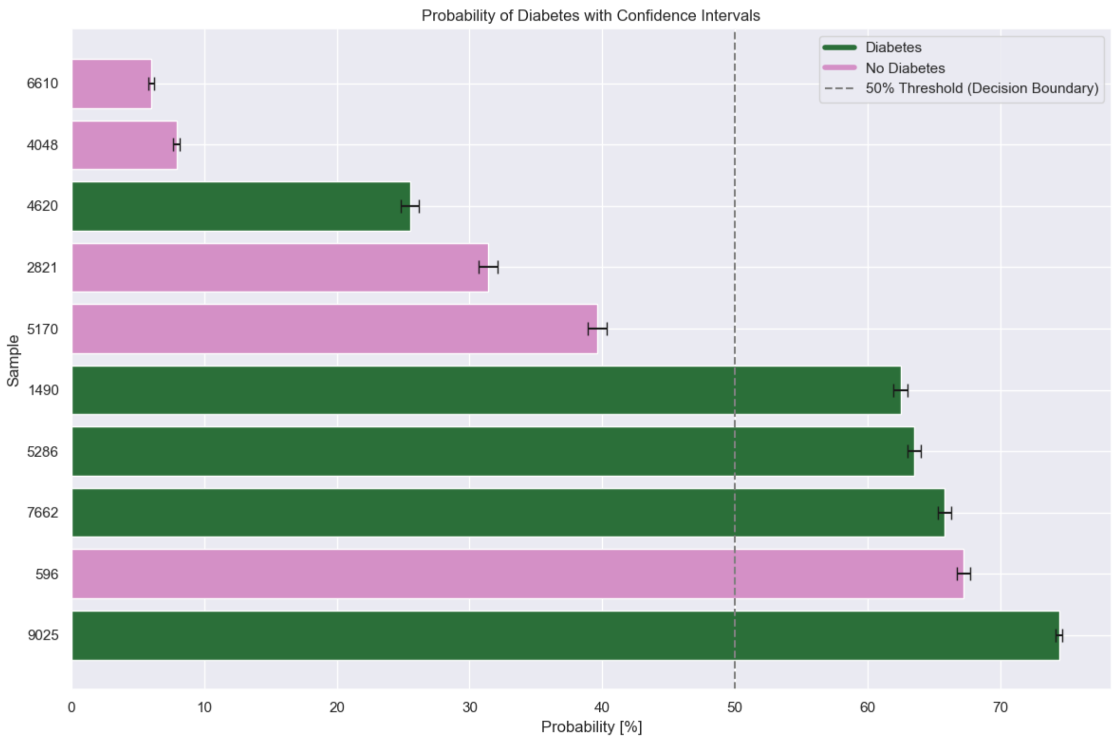
```
print(samples[['P(A)', 'Lower Bound P(A)', 'Upper Bound P(A)', '~P(A)', 'Lower␣
↪Bound ~P(A)', 'Upper Bound ~P(A)']])
```

```
           P(A)  Lower Bound P(A)  Upper Bound P(A)      ~P(A)  \
9025  74.435738         74.200645         74.667168  25.564262
1490  62.499058         61.964535         63.031124  37.500942
5286  63.518768         62.989120         64.045639  36.481232
4620  25.534306         24.835773         26.202810  74.465698
7662  65.825043         65.308952         66.337624  34.174957
2821  31.414356         30.696934         32.139988  68.585648
5170  39.611538         38.906208         40.322254  60.388462
4048   7.923836          7.665071          8.189652  92.076164
596   67.252159         66.746262         67.710129  32.747841
6610   5.980353          5.776088          6.190790  94.019646

      Lower Bound ~P(A)  Upper Bound ~P(A)
9025          25.332832          25.799355
1490          36.968876          38.035465
5286          35.954361          37.010880
4620          73.797188          75.164230
7662          33.662376          34.691048
2821          67.860016          69.303070
5170          59.677746          61.093792
4048          91.810349          92.334930
596           32.289871          33.253738
6610          93.809212          94.223915
```

[60]:

```
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]}
```

```python
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()
```



[45]:
```python
data_labels = {
    0: "Age",
    1: "Sex",
    2: "HighChol",
    3: "CholCheck",
    4: "BMI",
    5: "Smoker",
    6: "HeartDiseaseorAttack",
    7: "PhysActivity",
    8: "Fruits",
    9: "Veggies",
    10: "HvyAlcoholConsump",
    11: "GenHlth",
    12: "MentHlth",
    13: "PhysHlth",
    14: "DiffWalk",
    15: "Stroke",
    16: "HighBP"
}

for index, label in data_labels.items():
    print(f"{index} - {label}:")
```

```
0 - Age:
```

```
1 - Sex:
2 - HighChol:
3 - CholCheck:
4 - BMI:
5 - Smoker:
6 - HeartDiseaseorAttack:
7 - PhysActivity:
8 - Fruits:
9 - Veggies:
10 - HvyAlcoholConsump:
11 - GenHlth:
12 - MentHlth:
13 - PhysHlth:
14 - DiffWalk:
15 - Stroke:
16 - HighBP:
```

Przykładowa przebadana przeze mnie osoba:

[46]:
```python
print(samples.loc[4620])
print(samples.loc[596])
```

```
0                       8.000000
1                       0.000000
2                       0.000000
3                       1.000000
4                      37.000000
5                       1.000000
6                       0.000000
7                       1.000000
8                       1.000000
9                       1.000000
10                      0.000000
11                      2.000000
12                      0.000000
13                      0.000000
14                      0.000000
15                      0.000000
16                      0.000000
Diabetes                1.000000
P(A)                   25.534306
Lower Bound P(A)       24.835773
Upper Bound P(A)       26.202810
~P(A)                  74.465698
Lower Bound ~P(A)      73.797188
Upper Bound ~P(A)      75.164230
Name: 4620, dtype: float32
0                       6.000000
1                       1.000000
```

```
2                       1.000000
3                       1.000000
4                      31.000000
5                       0.000000
6                       0.000000
7                       1.000000
8                       0.000000
9                       1.000000
10                      0.000000
11                      3.000000
12                      3.000000
13                      0.000000
14                      0.000000
15                      0.000000
16                      1.000000
Diabetes                0.000000
P(A)                   67.252159
Lower Bound P(A)       66.746262
Upper Bound P(A)       67.710129
~P(A)                  32.747841
Lower Bound ~P(A)      32.289871
Upper Bound ~P(A)      33.253738
Name: 596, dtype: float32
```

[47]:

```python
print(" P(A) -> Diabetes [%] \n~P(A) -> No diabetes [%]\n")
print(samples)
```

```
 P(A) -> Diabetes [%]
~P(A) -> No diabetes [%]

         0    1    2    3     4    5    6    7    8    9  …   14   15   16  \
9025  11.0  1.0  1.0  1.0  33.0  1.0  0.0  0.0  0.0  1.0  …  1.0  0.0  1.0
1490  12.0  1.0  1.0  1.0  36.0  0.0  1.0  1.0  1.0  0.0  …  1.0  0.0  1.0
5286   4.0  0.0  1.0  1.0  22.0  1.0  0.0  1.0  0.0  1.0  …  0.0  0.0  0.0
4620   8.0  0.0  0.0  1.0  37.0  1.0  0.0  1.0  1.0  1.0  …  0.0  0.0  0.0
7662   6.0  1.0  1.0  1.0  35.0  0.0  1.0  0.0  0.0  1.0  …  1.0  0.0  1.0
2821   8.0  1.0  1.0  1.0  26.0  0.0  0.0  1.0  0.0  0.0  …  0.0  0.0  0.0
5170   7.0  0.0  1.0  1.0  30.0  0.0  0.0  1.0  0.0  1.0  …  0.0  0.0  1.0
4048   2.0  1.0  0.0  1.0  22.0  0.0  0.0  1.0  1.0  1.0  …  0.0  0.0  0.0
596    6.0  1.0  1.0  1.0  31.0  0.0  0.0  1.0  0.0  1.0  …  0.0  0.0  1.0
6610   3.0  1.0  0.0  1.0  33.0  1.0  0.0  1.0  1.0  0.0  …  0.0  0.0  0.0

      Diabetes       P(A)  Lower Bound P(A)  Upper Bound P(A)      ~P(A)  \
9025       1.0  74.435738         74.200645         74.667168  25.564262
1490       1.0  62.499058         61.964535         63.031124  37.500942
5286       1.0  63.518768         62.989120         64.045639  36.481232
4620       1.0  25.534306         24.835773         26.202810  74.465698
7662       1.0  65.825043         65.308952         66.337624  34.174957
```

```
2821        0.0  31.414356          30.696934          32.139988  68.585648
5170        0.0  39.611538          38.906208          40.322254  60.388462
4048        0.0   7.923836           7.665071           8.189652  92.076164
596         0.0  67.252159          66.746262          67.710129  32.747841
6610        0.0   5.980353           5.776088           6.190790  94.019646

      Lower Bound ~P(A)  Upper Bound ~P(A)
9025          25.332832          25.799355
1490          36.968876          38.035465
5286          35.954361          37.010880
4620          73.797188          75.164230
7662          33.662376          34.691048
2821          67.860016          69.303070
5170          59.677746          61.093792
4048          91.810349          92.334930
596           32.289871          33.253738
6610          93.809212          94.223915

[10 rows x 24 columns]
```

[48]:
```python
import os
import pandas as pd
import numpy as np
import torch

person = {
    'Age': 6,
    'Sex': 0,
    'HighChol': 0,
    'CholCheck': 1,
    'BMI': 22.92,
    'Smoker': 0,
    'HeartDiseaseorAttack': 0,
    'PhysActivity': 0,
    'Fruits': 1,
    'Veggies': 0,
    'HvyAlcoholConsump': 0,
    'GenHlth': 3,
    'MentHlth': 15,
    'PhysHlth': 0,
    'DiffWalk': 0,
    'Stroke': 0,
    'HighBP': 0,
}

df_person = pd.DataFrame([person])
```

```
model = torch.load(os.getcwd() + '/../backend/Models/predict_2.pth')

outputs = model.predict_proba(df_person.values)

probabilities = outputs[:, 1]

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

print(df_person)
```

```
   Age  Sex  HighChol  CholCheck    BMI  Smoker  HeartDiseaseorAttack  \
0    6    0         0          1  22.92       0                     0

   PhysActivity  Fruits  Veggies  HvyAlcoholConsump  GenHlth  MentHlth  \
0             0       1        0                  0        3        15

   PhysHlth  DiffWalk  Stroke  HighBP   P(A)      ~P(A)
0         0         0       0       0  28.48  71.519997
```

[6]:
```
!jupyter nbconvert --to pdf --output-dir='../frontend/public/PDF'␣
↪Diabetes_prediction.ipynb
```

```
[NbConvertApp] Converting notebook Diabetes_prediction.ipynb to pdf
[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'][61]:
{'cell_type': 'markdown',
 'execution_count': 31,
 'metadata': {'ExecuteTime': {'end_time': '2024-09-21T21:04:17.140228Z',
                              'start_time': '2024-09-21T21:04:15.569232Z'}},
 'outputs': ['…1 outputs…'],
 'source': 'Nawiązanie połączenia z bazą i zapis parametrów modelu '
           '[Pamiętaj…'}
[NbConvertApp] Support files will be in Diabetes_prediction_files/
[NbConvertApp] Making directory ./Diabetes_prediction_files
[NbConvertApp] Writing 240117 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[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 1484685 bytes to
../frontend/public/PDF/Diabetes_prediction.pdf
```

[ ]:
```
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')
```