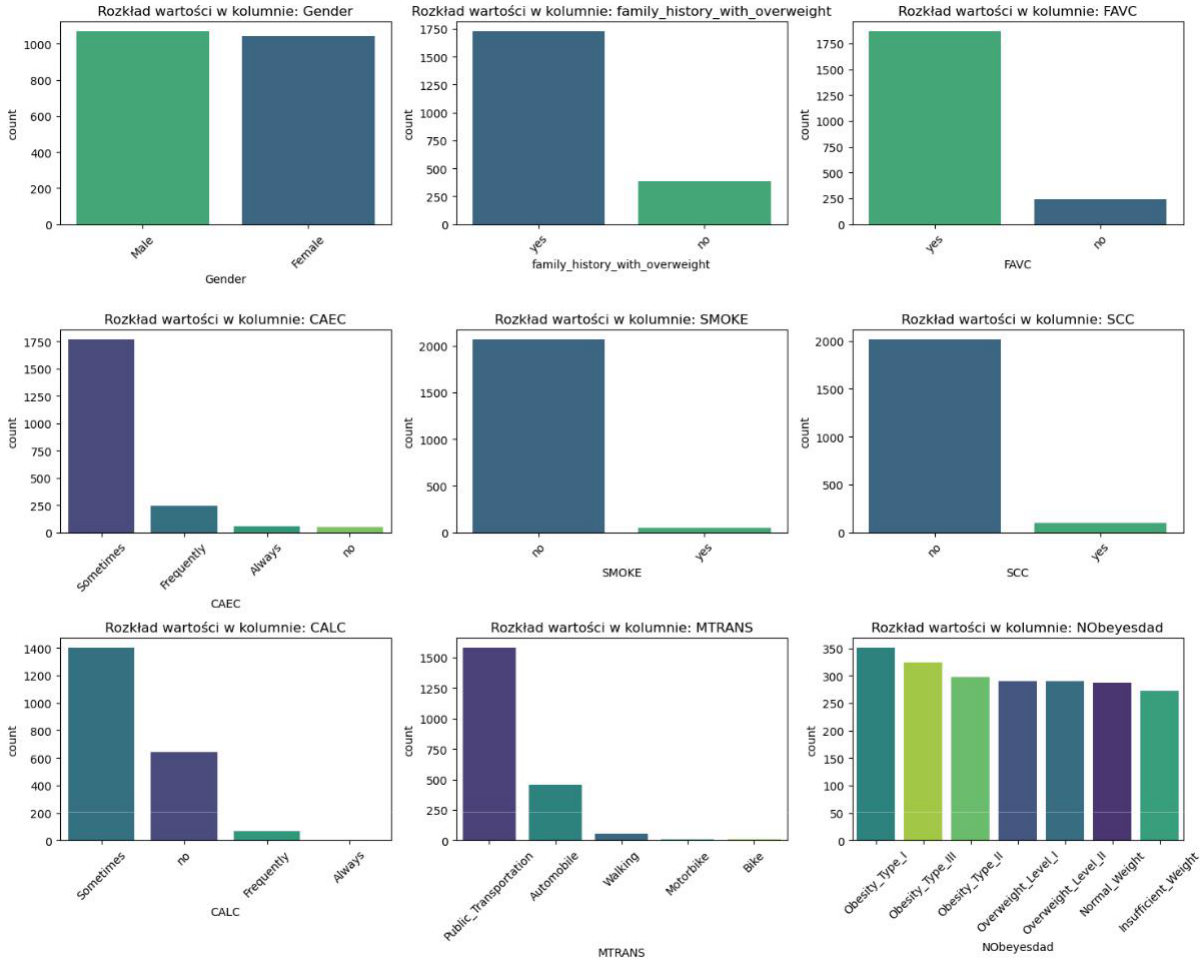
I. Pierwsza część

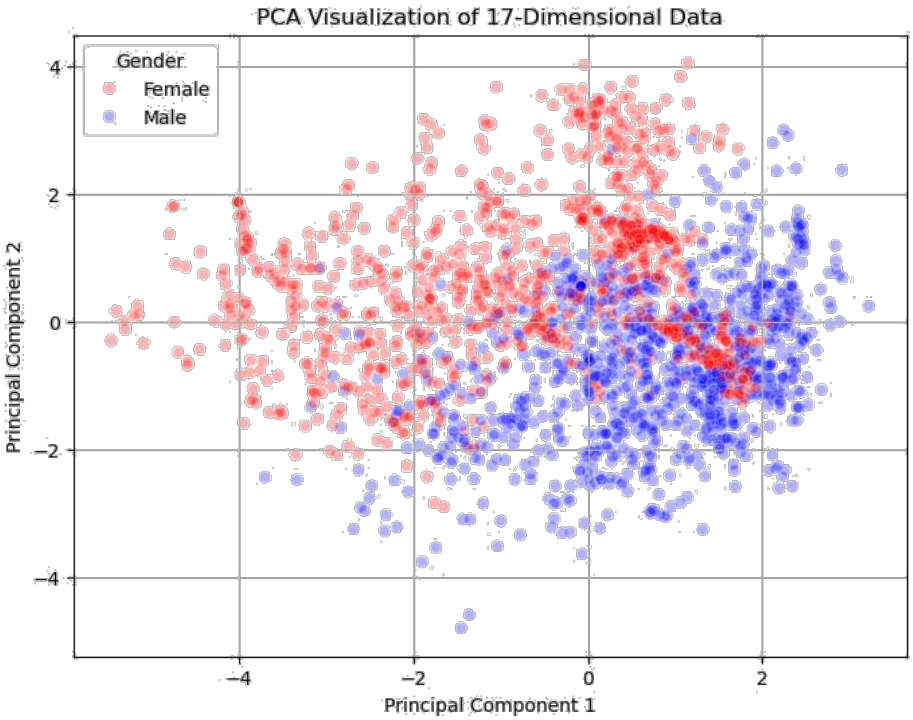
1. Wprowadzenie

Celem analizy jest zrozumienie zależności pomiędzy cechami zawartymi w zbiorze danych oraz identyfikacja zmiennych o najwyższej wartości informacyjnej. Analiza uwzględnia wizualizacje i statystyki opisowe.

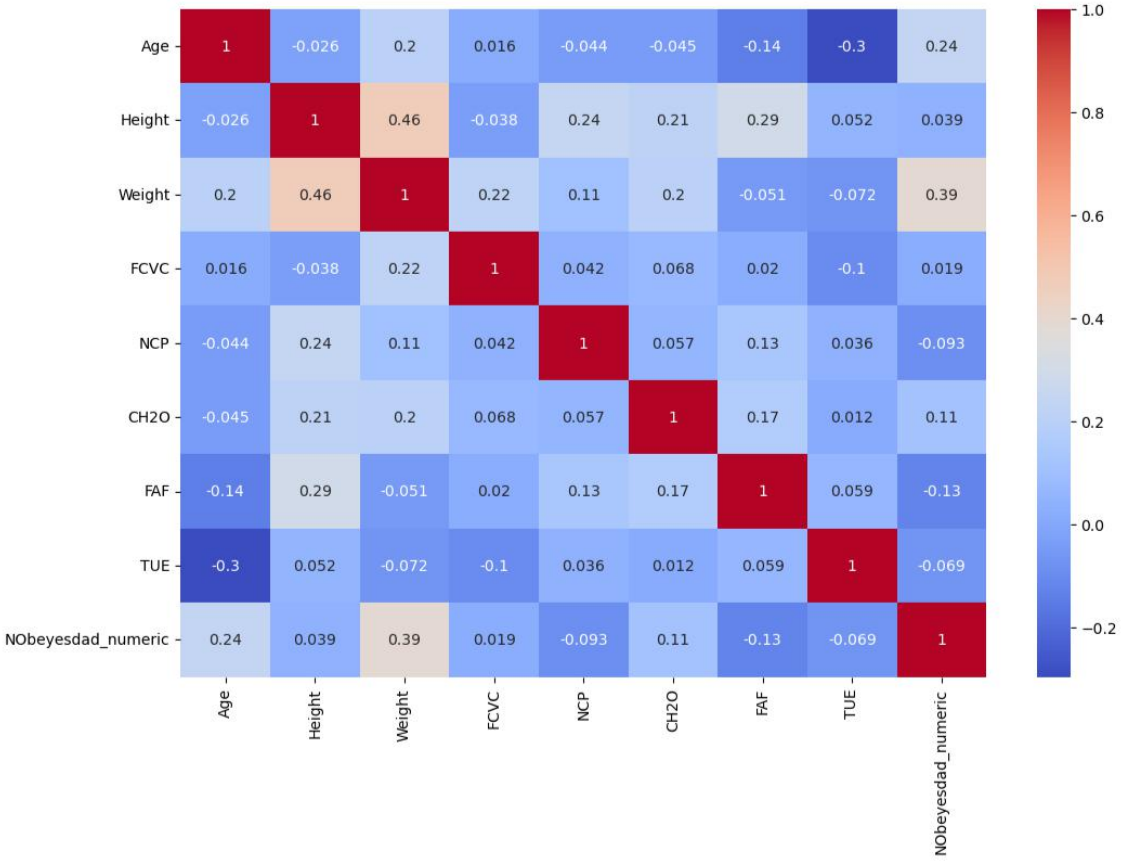
2. Obserwacje

- Na cechy numeryczne skrajne wartości bardzo nie wpływają, ponieważ średnia i mediana wszędzie mają podobne wartości
- Cechy płeć i stopień otyłości są zbalansowane w przeciwieństwie do reszty cech.
- W cechach numerycznych które się odnoszą do częstości wykonywania czynności jako wartości przeważają liczby całkowite

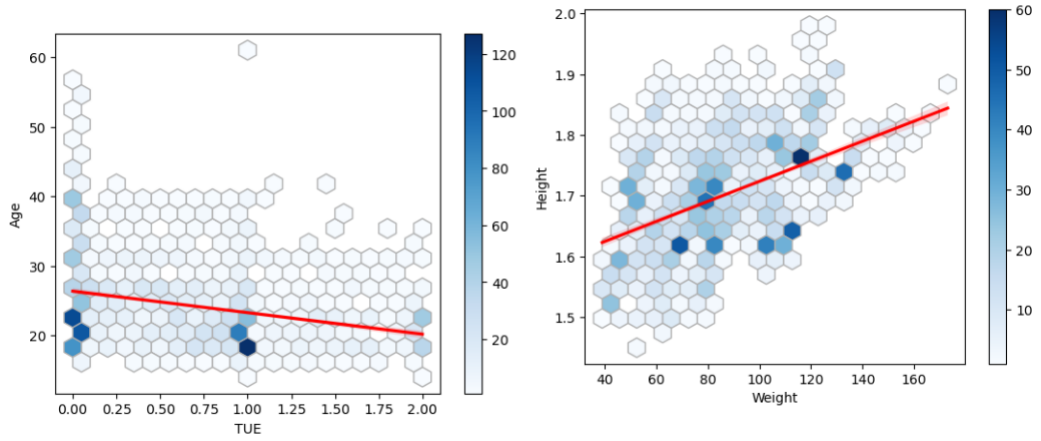




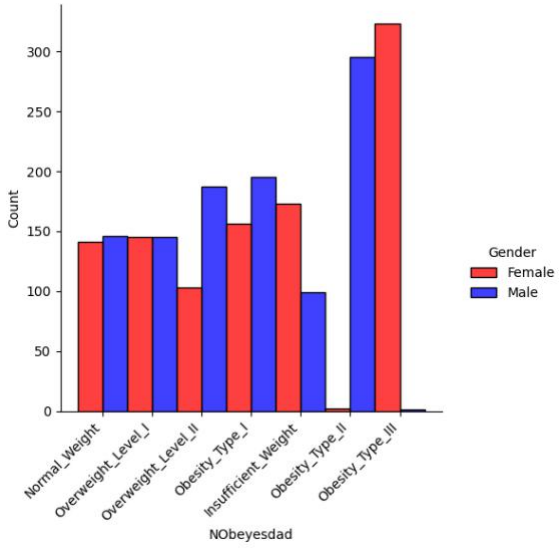
Dwie płcie wykazują pewien stopień separacji w dwuwymiarowej przestrzeni PCA. W środkowej części występuje znaczące nakładanie się danych. Punkty danych są szeroko rozproszone wzdłuż obu głównych składowych, co wskazuje, że te dwie składowe przechwytują znaczną wariancję oryginalnych 17-wymiarowych danych.



W zbiorze danych można zauważyć silnie skorelowane dane takie jak wzrost i waga. Ponadto poziom otyłości jest skorelowany z wagą i ilością spożywanej wody i niegatywanie skorelowany z częstotliwością uprawianego sportu.



A jeśli chodzi odwrotnie skorelowane dane to najlepiej wypada częstotliwość używania urządzeń elektronicznych z wiekiem.



Wiele kobiet ma otyłość typu 3, a prawie żadnych mężczyzn. Dokładnie odwrotna sytuacja ma miejsce w przypadku typu 2.

3. Wnioski

Na podstawie wykresu o typie otyłości rozdzielającego płcie można wywnioskować, że dane mogą nie być prawidłowe co sugeruje ilość kobiet w obesity type II i mężczyzn obesity type III.

Ciekawą kwestią jest to, że osoby otyłe piją dużo wody co się kłóci z powszechnymi przekonaniami. W zbiorze danych występują korelacje takie jak wzrost z wagą, poziom otyłości z wagą itd.

II. Druga część

1. Wstęp

Cel projektu:
Klasyfikacja stanów otyłości (7 klas) na podstawie nawyków żywieniowych i parametrów fizycznych.
Zbiór danych:
obesity_data.csv (2111 próbek, 17 cech)

2. Szczegółowa analiza preprocessingu

2.1 Struktura transformacji

Pipeline przetwarzania wykorzystuje `ColumnTransformer` do równoległego przetwarzania różnych typów cech:

2.2 Rodzaje transformacji

1. **Cechy binarne** (np. płeć):
    - Kodowanie 0/1 przez `OrdinalEncoder`
    - Przykład: “Male” → 0, “Female” → 1
2. **Cechy kategoryczne** (sposób przemieszczania się):

- Pełne one-hot encoding przez `OneHotEncoder`

2.3 Skalowanie

- `RobustScaler` stosowany globalnie po transformacji kategorycznej
- Odporny na outliery poprzez użycie mediany

2.4 Wyniki modeli scikit-learn

Model	Dokładność
DecisionTree	60.9%
LogisticRegression	53.9%
SVC	62.4%

3. Implementacje w numpy

3.1 Regresja zamkniętą formułą

Ograniczenia:
- Brak stabilności numerycznej dla `X.T @ X` przy dużych wymiarach - Podatność na outliery - Nieefektywne n^3 - Nie radzi sobie dobrze z danymi, które mają nieliniową strukturę

3.2 Regresja logistyczna z GD

Ograniczenia:
- Kluczowe hiperparametry, jak learning rate czy liczba epok, trzeba dobrać ręcznie lub metodami optymalizacji. - Gradient descent minimalizuje błąd ogólny, więc przy dużej dysproporcji klas może ignorować mniejszościową klasę.

3.3 Porównanie wydajności

Metoda	Dokładność
Własna regresja	49.3%
Scikit-learn	53.9%

Analiza:
Różnica wynika z braku optymalizacji w implementacji własnej (brak regularyzacji, prosty GD)

4. Implementacja w PyTorch

4.1 Wyniki GPU vs CPU

Metryka	CPU	GPU
Czas	9.73s	13.1s

Analiza: Powody dlaczego czas trenowania na cpu jest mniejszy niż na gpu - Mały rozmiar danych – GPU zyskuje przewagę przy dużych macierzach, dla małych zbiorów dane mogą się szybciej przetwarzać na CPU ze względu na mniejsze koszty narzutu. - Overhead związany z inicjalizacją GPU – uruchomienie kerneli GPU i zarządzanie nimi może zająć więcej czasu niż bezpośrednie wykonanie kodu na CPU. - Niewielka liczba operacji matematycznych – regresja logistyczna to stosunkowo prosty model, który nie wykorzystuje w pełni równoległych możliwości GPU. - Koszt przenoszenia danych do/z GPU – kopiowanie danych między pamięcią RAM (CPU) a pamięcią GPU może zająć więcej czasu niż samo obliczenie na CPU.

III. Trzecia część

1. Porównanie wyników dla różnych modeli

2. Wyniki cząstkowe

	Precision	Recall	F1-score
MyLogisticRegression	0.5078	0.4993	0.4652
+ PolynomialFeatures	0.7699	0.7755	0.7662
+ SMOTE	0.7853	0.7878	0.7799
+ undersample	0.7838	0.7854	0.7770
DecisionTreeClassifier	0.5954	0.5827	0.5666
+ PolynomialFeatures	0.9253	0.9247	0.9245
+ Smote	0.9243	0.9237	0.9237
+ undersample	0.9165	0.9152	0.9149
+ hiperparameter optimalization	0.9408	0.9403	0.9402
RandomForestClassifier	0.6130	0.5950	0.5798
+ PolynomialFeatures	0.9516	0.9507	0.9508
+ Smote	0.9550	0.9545	0.9545
+ undersample	0.9477	0.9469	0.9470
+ hiperparameter optimalization	0.9555	0.9550	0.9550

a. Walidacja krzyżowa

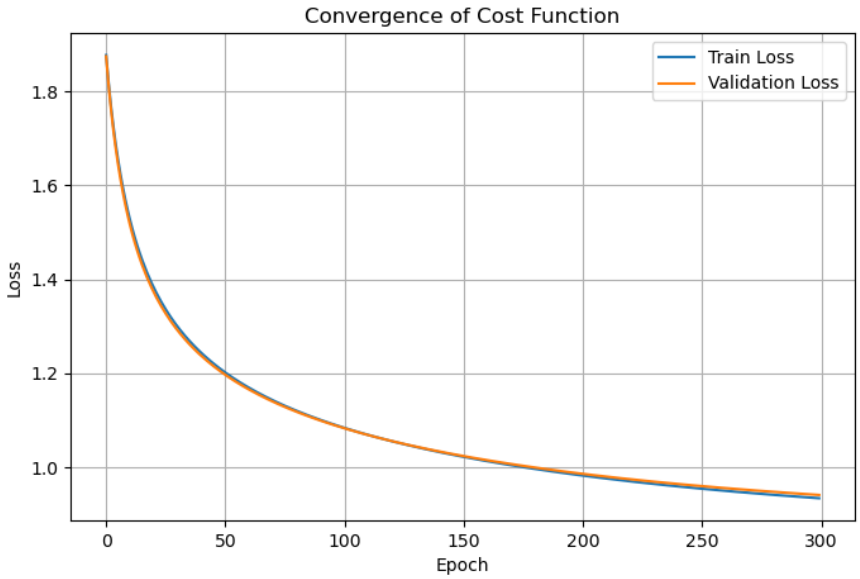
Podzbiór	Wynik
I	0.9347
II	0.9403
III	0.9459

Średni wynik 0.9403 ± 0.0046

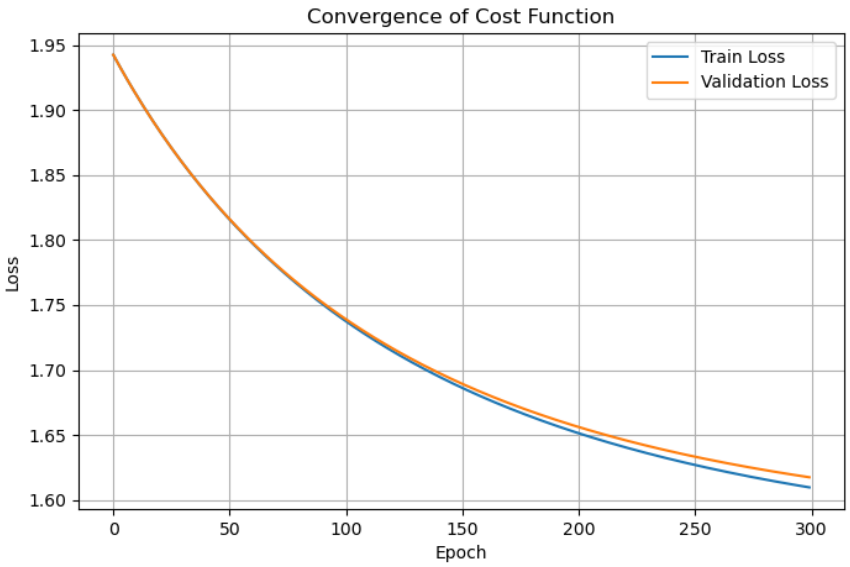
Wniosek: różnica pomiędzy podzbiorami jest niewielka co oznacza, że model dobrze się generalizuje (uogólnia na nowe dane)

b. Wykresy zbieżności i analiza błędów

Wykres z PolynomialFeatures



Wykres bez PolynomialFeatures



Na wykresach można zauważyć, że nie ma zjawiska underfitting i overfittingu. Dodanie dodatkowych kolumn poprzez PolynomialFeatures zmiejsza ilość błędów na koniec uczenia. Też krzywa jest bardziej stroma co oznacza, że ilość błędów szybciej się zmniejsza.

c. Usprawnienie danych – balansowanie zbiorów

Zarówno RandomForestClassifier, jak i DecisionTreeClassifier uzyskały nieco lepsze wyniki po balansowaniu zbioru danych, zwłaszcza w metrykach takich jak Recall i F1-score. Poprawa skuteczności wyniosła około 1–2%, co jest zgodne z oczekiwaniami, biorąc pod uwagę, że dane były już wstępnie dosyć zbalansowane. Największe korzyści wynikające z zastosowania technik balansowania zaobserwowano w poprawie

równowagi między precyzją a czułością, co może mieć kluczowe znaczenie w zastosowaniach, w których istotne jest wykrywanie mniejszościowych klas, takich jak np. detekcja oszustw. Mimo że wzrost metryk był stosunkowo niewielki, balansowanie danych warto traktować jako istotny element procesu przygotowania danych, szczególnie w przypadkach, gdy rozkład klas jest bardziej nierównomierny.

d. Optymalizacja hiperparametrów

Model	Wynik
RandomForestClassifier – z hiperparametrami	0.9550
RandomForestClassifier – bez hiperparametrów	0.9545
DecisionTreeClassifier – z hiperparametrami	0.9384
DecisionTreeClassifier – bez hiperparametrów	0.9271

Decision Tree

- criterion: 'entropy'
- max_depth: 10
- min_samples_split: 2

Random Forest

- class_weight: 'balanced'
- max_depth: 20
- min_samples_split: 2
- n_estimators: 100

Strojenie hiperparametrów to istotny etap w budowie modeli uczenia maszynowego, jednak wiąże się z kilkoma trudnościami. Po pierwsze, GridSearchCV cechuje się wysoką złożonością obliczeniową, ponieważ testuje każdą możliwą kombinację parametrów. Przy większej liczbie hiperparametrów i wartości, które mogą one przyjmować, liczba kombinacji szybko rośnie, co znacząco wydłuża czas obliczeń. Dodatkowo, parametry są silnie zależne od danych – ustawienia, które sprawdzają się dobrze na jednym zbiorze, mogą nie przynieść dobrych rezultatów na innym. Co więcej, GridSearchCV nie gwarantuje znalezienia globalnego optimum – jeżeli najlepsze ustawienia nie zostały uwzględnione w przeszukiwanej siatce, mogą zostać pominięte. Istnieje również ryzyko przeuczenia (overfittingu), gdy model jest zbyt dobrze dopasowany do danych treningowych podczas strojenia.

W przeprowadzonym eksperymencie zastosowano strojenie hiperparametrów dla dwóch modeli: RandomForestClassifier oraz DecisionTreeClassifier. RandomForestClassifier wykazał wyższą stabilność i lepszą dokładność zarówno przed, jak i po strojenie hiperparametrów, choć różnica w wynikach była minimalna. Dla DecisionTreeClassifier

tuning parametrów przyniósł wyraźny wzrost dokładności o 1,1 punktu procentowego, co pokazuje, jak duży wpływ mogą mieć odpowiednio dobrane parametry. W przypadku RandomForestClassifier istotne okazało się ustawienie (class_weight='balanced'), co może być szczególnie ważne w przypadku nierównomiernego rozkładu klas w danych. Podsumowując, strojenie hiperparametrów jest niezbędne, aby wydobyć pełny potencjał modeli, szczególnie tych prostszych, jak drzewa decyzyjne.