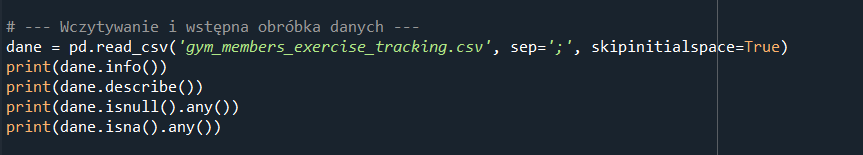
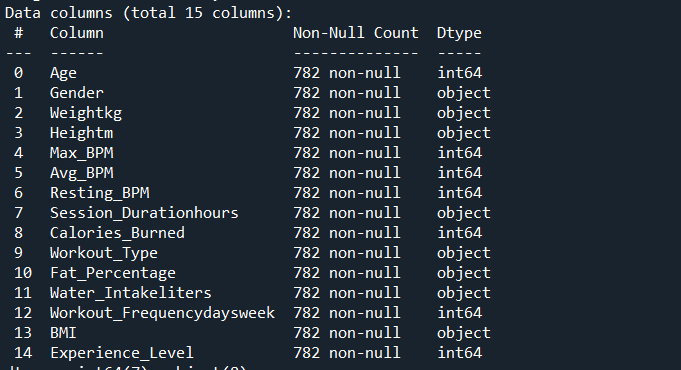
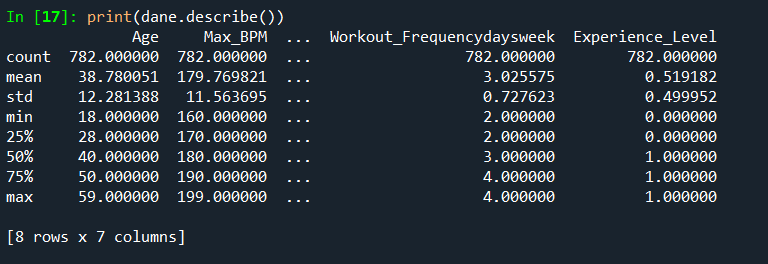
Raport zespołowy 1

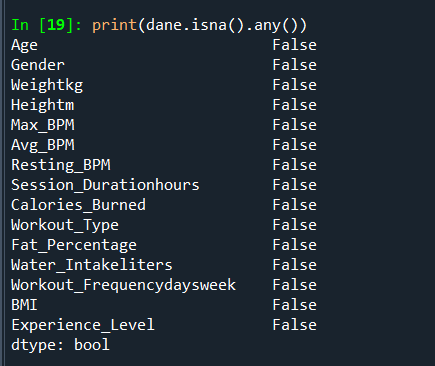
- **gimnastyka**

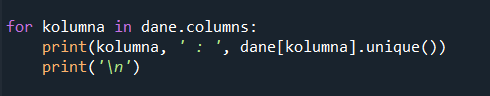
**1. Wczytanie danych, wstępna obróbka oraz sprawdzenie jakości danych**

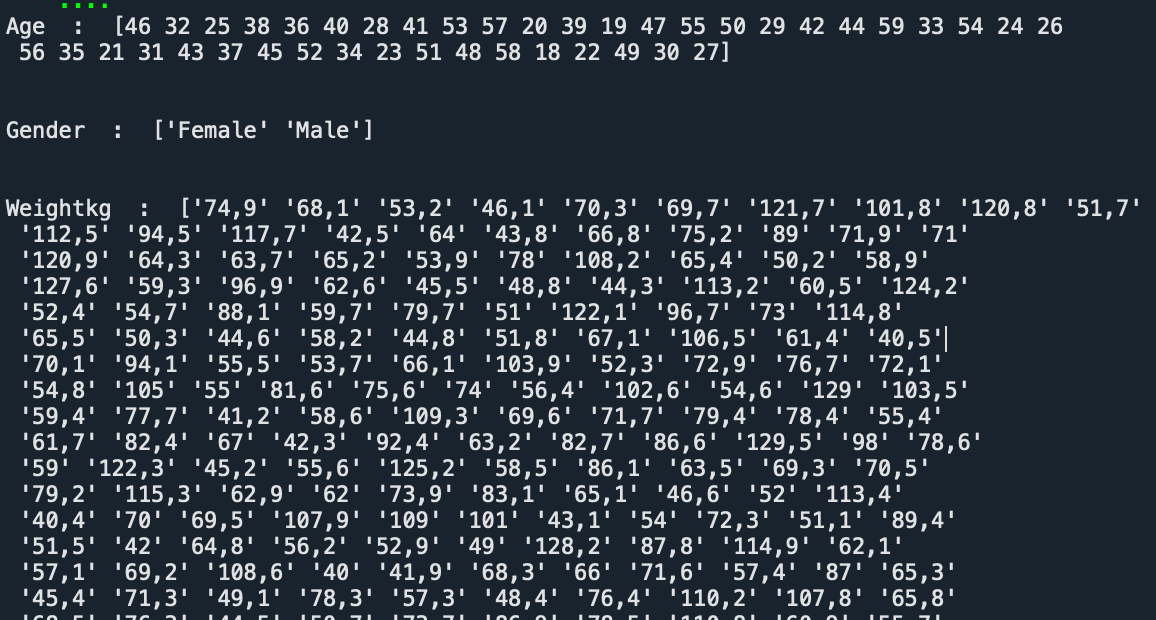
Następnie zweryfikowaliśmy dane znajdujące się w tabeli za pomocą następujących komend:  
Pierwszą rzeczą na którą było trzeba zwrócić uwagę to separator jakim nasze dane są oddzielne i ustawić poprawną jego wartość aby zostały one poprawnie wczytane i zapisane do zmiennej.  
  
Kolejna linijka kodu odpowiada za wypisanie odpowiednich informacji odnośnie wczytanych rekordów:  


Po sprawdzeniu tego wyniku może zauważyć że niektóre z kolumn są wczytane jako typ object co może wzbudzić nasze zaniepokojenie np. Kolumna *Weightkg* przyjmuję typ object natomiast jest ona odpowiedzialna za wagę danej osoby więc powinna być wczytana jako float ponieważ może ona przyjmować wartości zmiennoprzecinkowe np 71,6.

Describe() pokaże nam podstawowe statystyki takie jak: średnia, odchylenie standardowe, kwartyle oraz wartości min i max:  


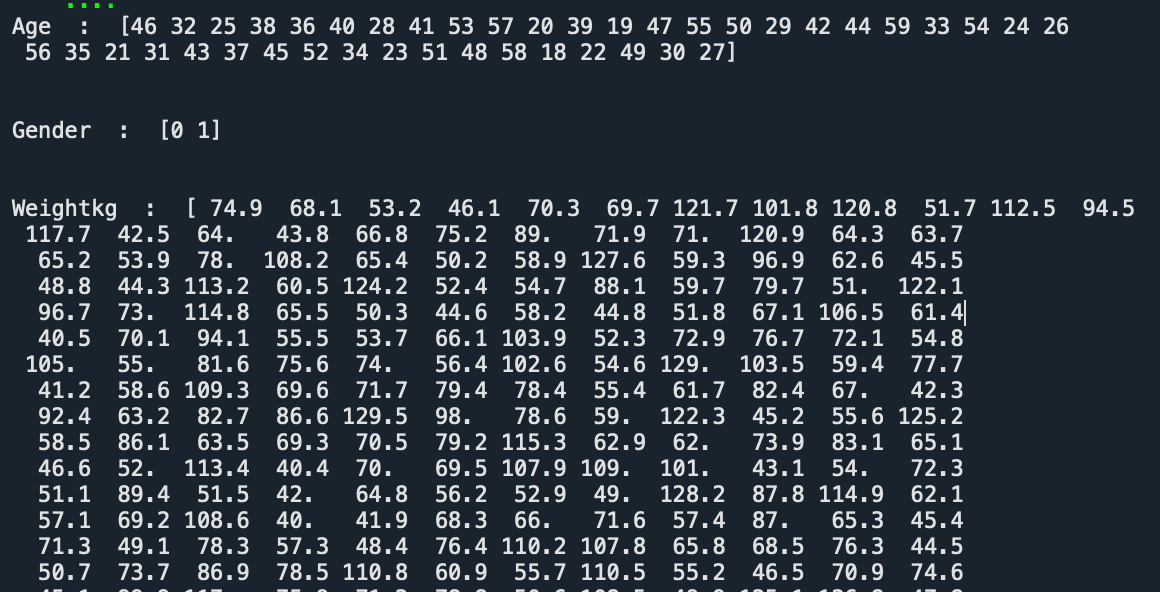
Następne fragmenty kodu odpowiadają za sprawdzenie czy w naszych danych nie występują jakieś braki musimy to zweryfikować ponieważ w przypadku kiedy takie braki by występowały musielibyśmy je uzupełnić jakąś metodą lub usunąć rekordy w których te braki występują:  
 

Jak możemy zauważyć wyniki jakie otrzymaliśmy to False co oznacza że w naszych danych nie występują żadne braki (NaN). Gdybyśmy np otrzymali w którejś z linii wartość *True* to by oznaczało że akurat w tym miejscu występują jakieś braki i należało by to zweryfikować/poprawić.  
W następnym kroku powrócimy do tego co wcześniej napisaliśmy i przyjrzymy się jeszcze raz naszym danym:  


Ten fragment kodu jest odpowiedzialny za wylistowanie z każdej kolumny unikatowych danych przez co możemy zweryfikować czy są one wczytane poprawnie i czy nie ma jakiś dziwnych wyników które nie pasowałyby do odpowiednich kategorii przyjrzyjmy się im: 

Jak możemy zauważyć po części wypisanych danych zmienna *Age* jest wczytana poprawnie ponieważ wartość jest liczbowa, *Gender* również ponieważ jest ona stringiem natomiast *Weightkg* również jest wczytana jako string co jak wcześniej przypuszczaliśmy nie jest poprawne ponieważ chcielibyśmy aby przyjmowała ona wartość liczbową a dokładniej zmiennoprzecinkową (float).  
Wrzucanie tutaj wszystkich wartości unikalnych mija się z celem bo jest ich po prostu za dużo, a podobny problem wystąpił w kolumnach: *Heightm*, *Session\_Durationhours*, *Fat\_Percentage*, *Water\_Intakeliters* i *BMI.*

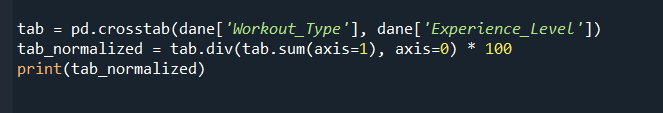
Aby rozwiązać ten problem należy wykonać odpowiednie fragmenty kodu które zamiennią nam wartość tekstową na liczby typu float, albo płeć na wartość binarną ponieważ do dalszej analizy i wykonywania modeli wartości zmiennej string będzie nam przeszkadzać.



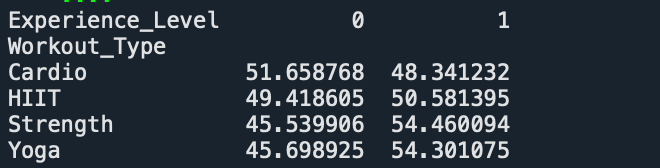
Jak widzimy dane zostały poprawnie zmienione i możemy przystąpić do dalszej analizy naszych zmiennych.  
  
Na koniec jeszcze ostatnia weryfikacja czy w zbiorze danych nie ma żadnych duplikatów. 

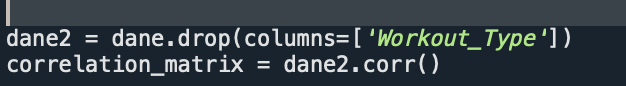
**2. Analiza Predyktorów**

Przejdźmy teraz do analizy predyktorów ponieważ są one niezbędne do zbudowania modeli które znajdą się w dalszej części naszego raportu. Na początku zanim przystąpiliśmy do budowania modeli przyjrzeliśmy się znowu naszym danym ale w celu sprawdzenia co może mieć wpływ na zmienną celu i nasze podejrzenia wzbudziła zmienna *Workout\_Type* ponieważ wraz z kolegą uznaliśmy że naszym zdaniem nie ma ona wpływu na zmienną celu ale możemy to zweryfikować wykonując macierz krzyżową.

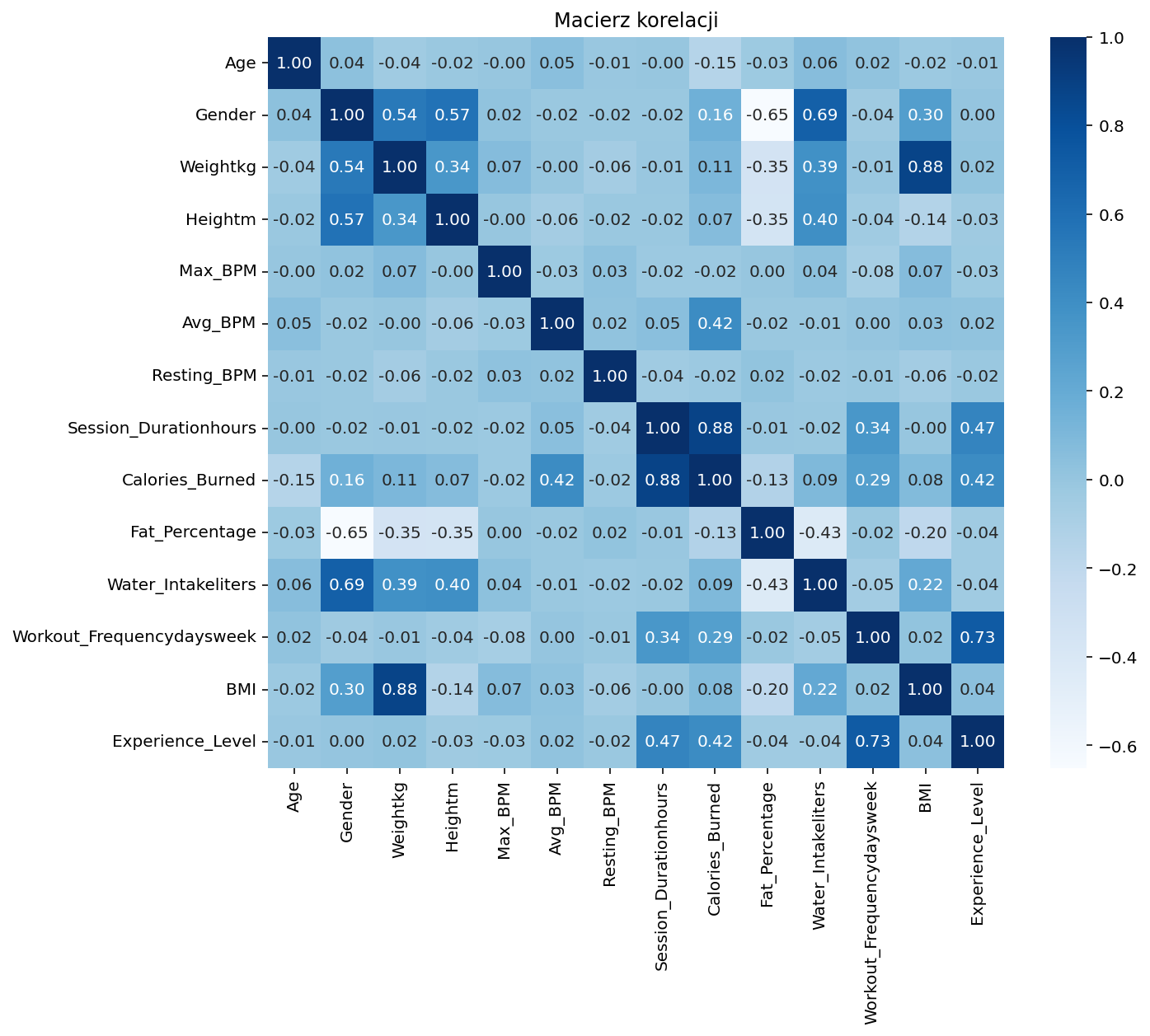


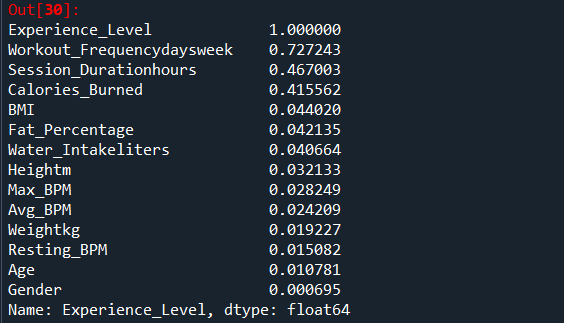
Na poniższym screenie widzimy, że typ wykonywanych ćwiczeń (*Workout\_Type*) nieznacznie wpływa na zmienną celu (*Experience\_Level*) ponieważ wyniki są bardzo zbliżone zarówno dla osób niedoświadczonych jak i doświadczonych. Nasze doświadczenia jako osoby trenujące okazały się trafne przez co możemy potwierdzić ten schemat, osoby bardziej doświadczone częściej wybierają treningi siłowe choć to nie znaczy o kompletnym odpuszczeniu treningów aerobowych (*Cardio, Yoga, HIIT*).



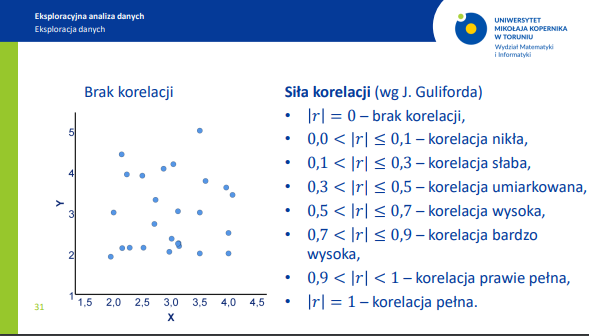
Skoro zmienna Workout\_Type nie jest istotna możemy ją usunąć ponieważ nie będzie ona miała dużego wpływu na naszą zmienną celu a pozbędziemy się jej wykonując odpowiedni fragment kodu:  


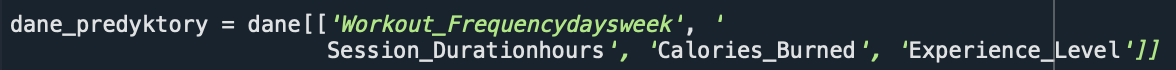
Pozbyliśmy się kolumny która była bezużyteczna to następnym krokiem który wykonamy będzie znalezienie predyktorów które posłużą nam w dalszej pracy i zbudowaniu modeli. W celu sprawdzenia tego możemy wykonać macierz korelacji która wyświetli nam zależności wszystkich zmiennych między sobą pokazując wpływ na daną zmienną ale my zwrócimy uwagę na naszą zmienną celu:



Możemy odczytać nasze wartości z tabeli które mają większy a który mniejszy wpływ na zmienną celu ale wypiszmy to jeszcze w konsoli aby to było bardziej czytelne wykonując odpowiednie polecenie:  
Wyniki:  


Jak możemy zauważyć największy wpływ na zmienną celu mają zmienne *Workout\_Frequencydaysweek* (0.73), *Session\_Durationhours* (0.47), *Calories\_Burned* (0.42) natomiast pozostałe nie są tak istotne i są one poniżej 0.05. Naszymi predyktorami zostaną zatem zmienne *Workout\_Frequencydaysweek*, *Session\_Durationhours*, *Calories\_Burned*.



Wartości na powyższym zdjęciu ilustrują nam siłę korelacji jak istotna może być nasza zmienna zatem wybór tamtych trzech zmiennych będzie dobrą opcją, więc zapiszmy sobie je do zmiennej aby posłużyć się nimi do budowania modeli:  


**3. Budowa modelu k-NN**

W tym rozdziale zaprezentujemy proces tworzenia modelu k-NN (k-Nearest Neighbors) w oparciu o wybrane predyktory. Wcześniej zidentyfikowaliśmy zmienne o największym wpływie na zmienną celu, a teraz wykorzystamy je do zbudowania i dostrojenia klasyfikatora k-NN.

**Kroki, jakie podejmiemy:**

**3.1. Ustalanie ziarna losowego i podział danych**

Ustaliliśmy ziarno (seed) generatora liczb losowych, by każdy, kto uruchomi ten kod, uzyskał te same wyniki. Ziarno generowania:



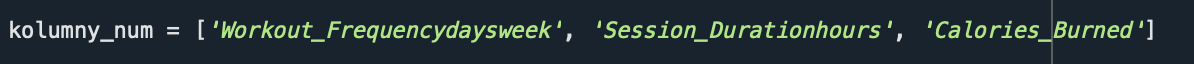
Wyodrębniamy z naszych danych tylko kolumny z cechami (bez kolumny z wartością docelową).

Pobieramy informację o tym, czy osoba jest doświadczona (1) czy niedoświadczona (0).



Dzielimy dane na część uczącą (70%) i testową (30%) w sposób losowy, ale kontrolowany, dzięki ustawionemu ziarnu. Parametr stratify=y powoduje, że proporcje klas (0 i 1) są podobne w obu zbiorach.

**3.2. Normalizacja danych**

Wymieniamy, które kolumny są numeryczne i wymagają normalizacji.

Tworzymy obiekt, który przekształci wszystkie nasze dane numeryczne w wartości mieszczące się w przedziale od -1 do 1.



Dopasowujemy skalowanie do danych uczących. Oznacza to, że na podstawie zbioru uczącego wyliczana jest skala i przesunięcie, za pomocą których prze transformujemy dane.



Transformujemy dane uczące - teraz każda cecha ma porównywalny zakres.



Również dane testowe normalizujemy przy użyciu tych samych parametrów. Ważne jest, aby nie dopasowywać skalera na zbiorze testowym, tylko zastosować ten już wyliczony na zbiorze uczącym. To odzwierciedla realną sytuację, w której nowe dane (testowe) są skalowane tak samo, jak dane użyte do trenowania.



**4. Model k-NN – Ocena, strojenie i porównanie z innymi wariantami**

W poprzednim rozdziale (rozdział 3) zbudowaliśmy model k-NN. Teraz skupimy się na tym, jak dobrać jego parametry (tzw. hiperparametry), jak ocenić jego jakość na danych uczących i testowych oraz co zrobić, gdy zauważymy zjawisko przeuczenia. Zbadamy także alternatywną konfigurację modelu k-NN i porównamy uzyskane rezultaty.

**Podpunkty omawiane w tym rozdziale:**

**4.1. Strojenie modelu k-NN za pomocą GridSearchCV**

Definiujemy zestaw wartości parametrów, które chcemy przetestować. Tym razem próbujemy różną liczbę sąsiadów (3, 5, 7, 9, 11) oraz dwa sposoby obliczania wagi sąsiadów (uniform – jednakowa waga, distance – bliższe punkty mają większe znaczenie).



Automatyczne testowanie każdej kombinacji podanych parametrów, w celu wybrania tej, która daje najlepsze wyniki. Korzysta z walidacji krzyżowej, aby uniknąć przypadkowych wyników.



Przeprowadzamy proces uczenia i oceny dla wszystkich kombinacji parametrów.



Sprawdzamy, który zestaw parametrów okazał się optymalny.   
W tym przypadku np. n=3 i weights='uniform'.

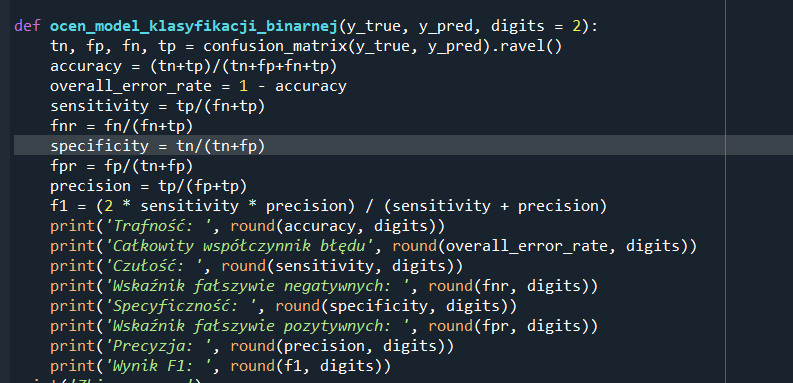


**4.2. Ocena optymalnego modelu k-NN na zbiorze uczącym i testowym**

Korzystając z wybranego najlepszego modelu, przewidujemy klasy dla zbioru uczącego i testowego.



Funkcja ta oblicza najważniejsze miary oceny modelu, takie jak trafność (accuracy), czułość (jak dobrze model rozpoznaje pozytywnych), swoistość (jak dobrze radzi sobie z negatywnymi), precyzja oraz wartość F1 (podsumowuje zarówno czułość, jak i precyzję).



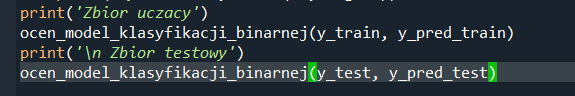
Dzięki temu dowiadujemy się, jak dobrze model radzi sobie z danymi, na których się uczył, oraz czy potrafi z podobną skutecznością klasyfikować nowe, niewidziane wcześniej dane (zbiór testowy).

### 

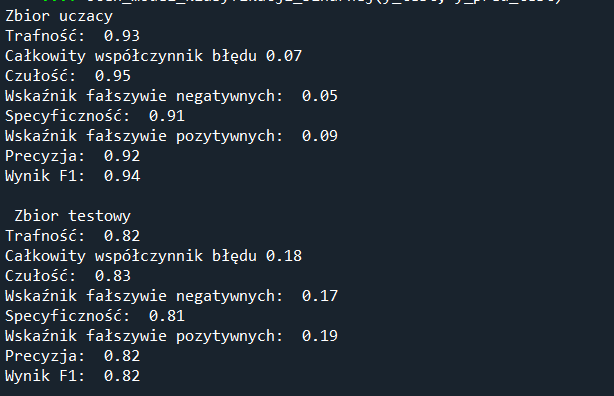
### 

### **4.3. Sprawdzenie problemu przeuczenia (overfitting)**

Fragment kodu:



Wynik:i



Możemy zauważyć zbyt duże różnice na poziomie 10% między zbiorem testowy a uczącym co sugeruje nam że model ten dla parametrów n=3, weights='uniform' może być przeuczony więc powinniśmy poszukać bardziej zbilansowanego modelu.

**4.4. Analiza alternatywnej konfiguracji k-NN**

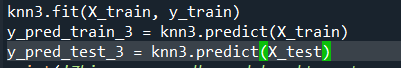
Wyświetlamy wyniki wszystkich testowanych konfiguracji k-NN. Pokazują one, które parametry były najlepsze, a które drugie czy trzecie w kolejności.

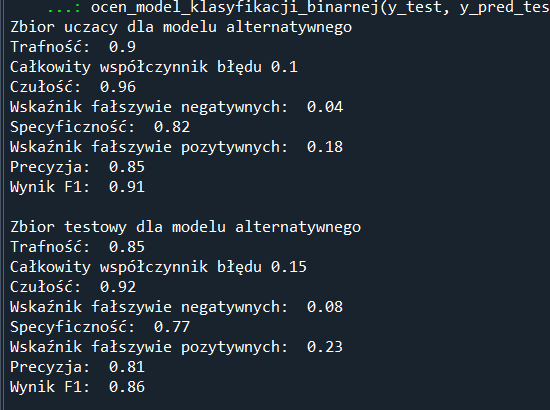


Tym razem wybieramy inną konfigurację modelu k-NN, np. 9 sąsiadów i wagi jednakowe dla wszystkich.



Uczymy model i sprawdzamy jego jakość na zbiorze uczącym i testowym, jak poprzednio.



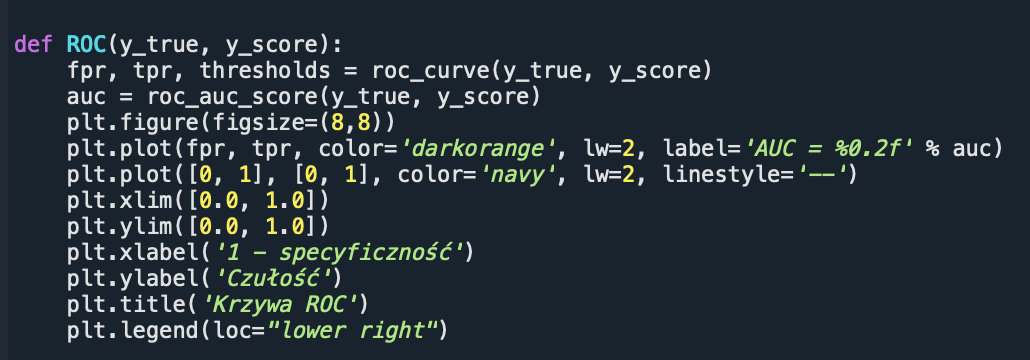


Chociaż GridSearchCV wskazał, że najlepsze parametry to n\_neighbors=3 i weights='uniform', to analiza wyników końcowych ujawnia, że ten model jest bardziej podatny na przeuczenie. Jego trafność i F1 spadają znacznie, gdy przechodzimy z danych uczących na testowe.

Z kolei model alternatywny (n=9, uniform) charakteryzuje się mniejszą różnicą między wynikami na zbiorach uczącym i testowym. Choć na danych treningowych nie jest tak "idealnie dopasowany", to na danych testowych osiąga wyższą trafność (0.85 vs 0.82) oraz wyższy wynik F1 (0.86 vs 0.82), co czyni go bardziej godnym zaufania przy przewidywaniu rzeczywistych, nowych obserwacji.

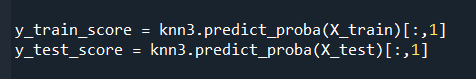
Ostatecznie, dane te pokazują, że nawet po wybraniu najlepszych parametrów za pomocą GridSearchCV warto sprawdzić inne konfiguracje. Czasem nieco mniej "idealnie" dopasowany model na zbiorze uczącym może lepiej radzić sobie w praktyce, lepiej generalizując na dane, których nigdy wcześniej nie widział.

**4.5. Wykorzystanie krzywej ROC i AUC do oceny zdolności rozróżniania klas**

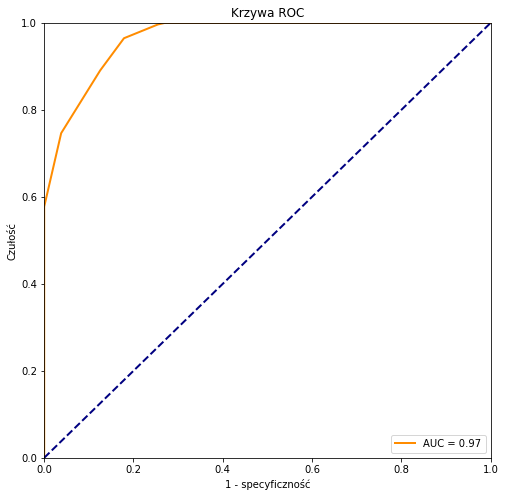
Przedstawia zależność między czułością (odsetek poprawnie wykrytych pozytywnych przypadków) a 1 - swoistość (odsetek fałszywych alarmów) dla różnych progów decyzyjnych.

**AUC (Area Under Curve)**: Pole pod krzywą ROC. Im wyższa wartość AUC, tym lepiej model potrafi odróżnić dwie klasy.

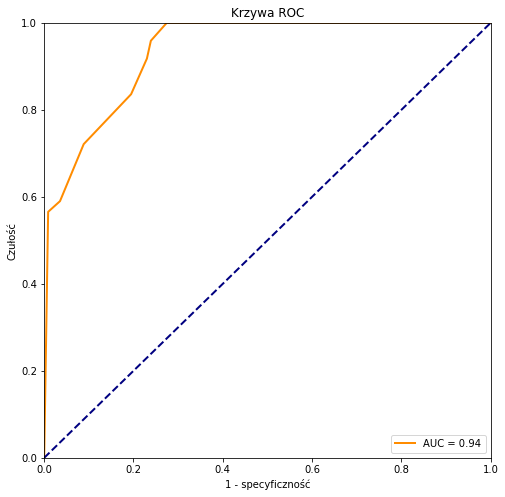
Uzyskujemy nie tylko klasy (0 lub 1), ale też prawdopodobieństwa. Używając prawdopodobieństw możemy sprawdzić różne progi decyzyjne i narysować krzywą ROC.

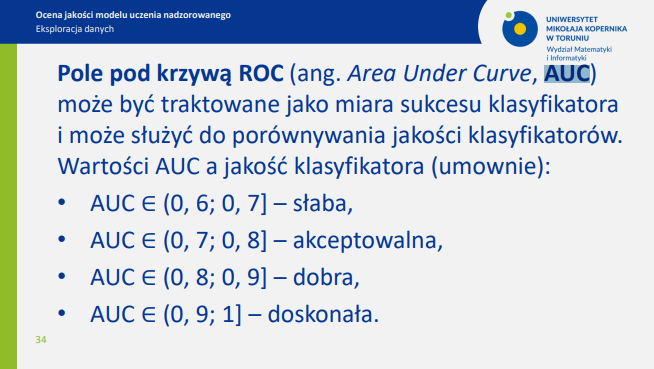


Wykresy ROC uczący:



Wykresy ROC testowy:



Analiza ROC:  


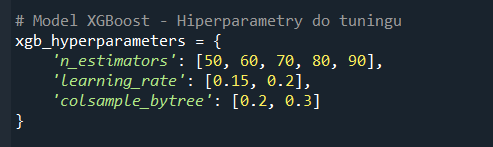
Wyniki:  
AUC na zbiorze uczącym ~97%, a na zbiorze testowym ~94%. Oznacza to, że model bardzo dobrze radzi sobie z odróżnieniem osób bardziej doświadczonych od mniej doświadczonych, co potwierdza wysoką jakość klasyfikatora.

**5. Model XGBoost**

W poprzednich rozdziałach zapoznaliśmy się z modelami k-NN i ich różnymi konfiguracjami. Teraz przejdziemy do bardziej zaawansowanej metody, jaką jest XGBoost (eXtreme Gradient Boosting). Jest to algorytm wykorzystujący technikę boosting, oparty o sumowanie wielu prostych modeli (drzew decyzyjnych), które są kolejno uczone na podstawie błędów popełnionych przez poprzednie modele. Dzięki temu XGBoost często osiąga bardzo dobre wyniki w zadaniach klasyfikacji binarnej i regresji.

**5.1. Strojenie hiperparametrów XGBoost za pomocą GridSearchCV**

Definiujemy xgb\_hyperparameters, czyli zestaw wartości różnych parametrów, które chcemy przetestować.

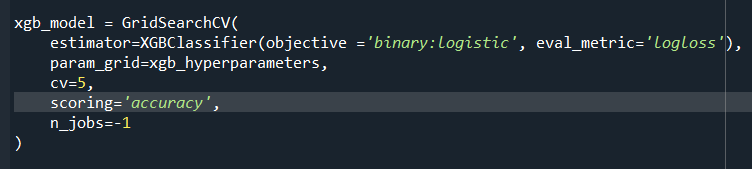


n\_estimators: liczba drzew decyzyjnych, które składają się na ostateczny model.

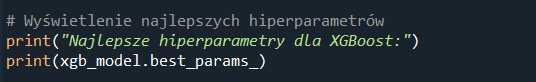
learning\_rate: tempo uczenia, które kontroluje, jak bardzo każdy kolejny model koryguje błędy poprzedników.

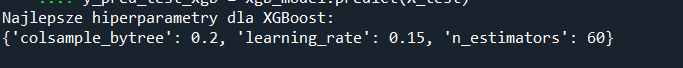
colsample\_bytree: proporcja cech losowo wybieranych do budowy każdego drzewa, co może pomóc w redukcji przeuczenia.

GridSearchCV sprawdzi wszystkie kombinacje parametrów, używając pięciokrotnej walidacji krzyżowej.



Na koniec wyświetlimy najlepsze znalezione ustawienia.





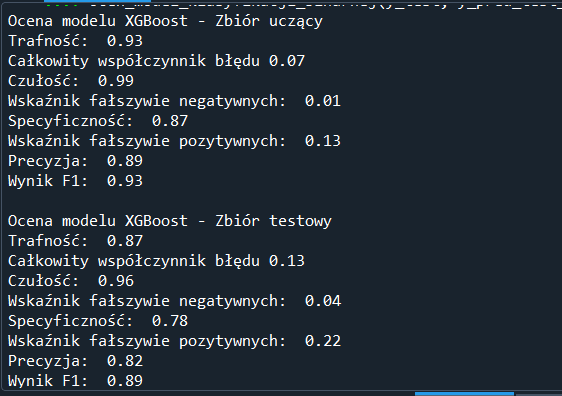
W ten sposób XGBoost zostanie dopasowany do danych w sposób optymalny dla zadanej siatki parametrów.

**5.2. Ocena modelu XGBoost na zbiorze uczącym i testowym**

Przewidujemy wartości zmiennej celu dla zbiorów uczącego i testowego, korzystając z najlepszego modelu XGBoost wybranego przez GridSearchCV.

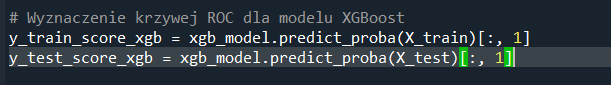


Podobnie jak w poprzednich rozdziałach, oceniamy model według trafności, czułości, swoistości, precyzji i F1. Pozwala to porównać model XGBoost z poprzednio ocenianymi modelami k-NN na dokładnie tych samych podstawowych miarach jakości.

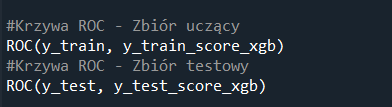


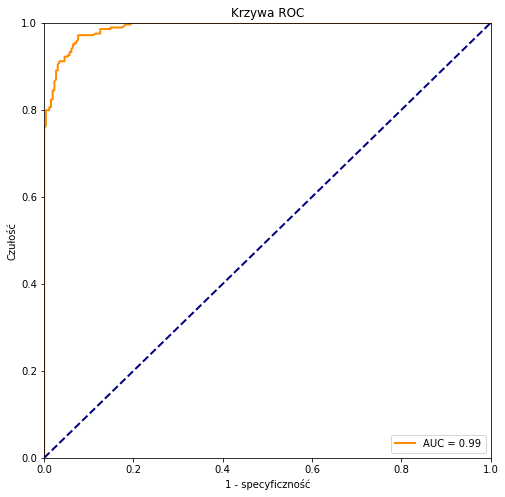
**5.3. Analiza krzywej ROC i obliczenie AUC dla XGBoost**

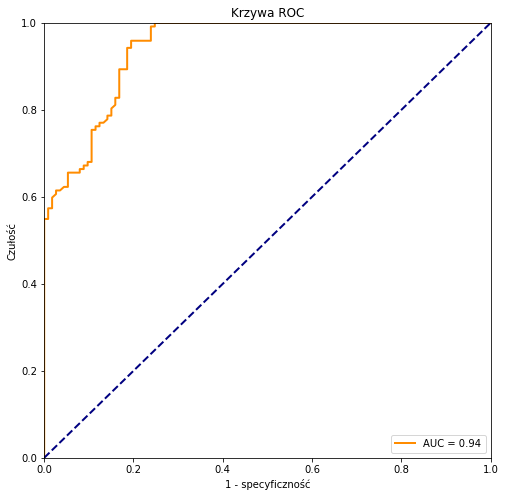
**predict\_proba**: Zamiast twardych decyzji 0 lub 1, uzyskujemy prawdopodobieństwo przynależności do klasy pozytywnej.



Wywołanie funkcji ROC narysuje krzywą ROC i wyświetli wartość AUC, zarówno dla zbioru uczącego, jak i testowego.



Zbiór uczący:  


Zbiór testowy:  


AUC umożliwia ocenę jakości klasyfikatora niezależnie od wybranego progu decyzyjnego. Im większe AUC (bliskie 1), tym lepiej model rozróżnia klasy.

Ocena krzywej ROC i AUC będzie dopełnieniem standardowych miar i pozwoli na jeszcze bardziej szczegółową analizę jakości modelu XGBoost.

**6. Wnioski i podsumowanie**

W niniejszej analizie porównaliśmy dwa podejścia do klasyfikacji poziomu doświadczenia w treningach: k-NN oraz XGBoost. Pracowaliśmy na wyselekcjonowanych predyktorach (Workout\_Frequencydaysweek, Session\_Durationhours, Calories\_Burned), które okazały się mieć największy wpływ na zmienną celu. Dane zostały podzielone na zbiór uczący (70%) i testowy (30%), a następnie znormalizowane w celu zapewnienia lepszej pracy algorytmów opartych na odległościach (jak k-NN).

**6.1. Wnioski dotyczące modeli k-NN**Choć za pomocą GridSearchCV początkowo ustalono, że najlepszym zestawem hiperparametrów jest n\_neighbors=3 i weights='uniform', to analiza wyników wykazała tendencję do przeuczenia. Model ten bardzo dobrze dopasowywał się do zbioru uczącego, ale jego skuteczność spadała wyraźnie na zbiorze testowym.

Poszukując alternatywy, wypróbowaliśmy inną konfigurację: n\_neighbors=9 i weights='uniform'. Ten model k-NN cechował się nieco niższą trafnością na zbiorze uczącym (90%), ale jednocześnie zachował wyższą trafność (85%) i lepszy wynik F1 (0.86) na zbiorze testowym. Różnica między zbiorami uczącym i testowym stała się mniejsza, co oznaczało lepszą zdolność do generalizacji i mniejsze ryzyko przeuczenia. Krzywa ROC oraz wartość AUC (91% na zbiorze testowym) potwierdziły bardzo dobre rozróżnianie klas przez ten model.

**6.2. Wnioski dotyczące modelu XGBoost**XGBoost, po dostrojeniu hiperparametrów za pomocą GridSearchCV, osiągnął jeszcze wyższe wartości trafności i F1 na zbiorze testowym niż najlepszy znaleziony model k-NN. Na zbiorze testowym uzyskaliśmy trafność 87% i F1 = 0.89, co przewyższyło wyniki k-NN (trafność 85%, F1=0.86). Różnica w wynikach pomiędzy zbiorem uczącym a testowym również była względnie niewielka (93% vs 87% trafności), sugerując brak znacznego przeuczenia.

Model XGBoost charakteryzował się bardzo wysoką czułością (0.96 na zbiorze testowym), co oznacza, że niemal zawsze rozpoznawał "doświadczonych" poprawnie. Specyficzność (0.78) i precyzja (0.82) były nieco niższe, ale wciąż na dobrym poziomie. Wysoki wynik F1 (0.89) potwierdził zbalansowane podejście do obu klas, a analiza ROC i AUC (wartości AUC zwykle powyżej 0.9) udowodniła, że XGBoost bardzo dobrze rozróżnia obie grupy.

**6.3. Porównanie modeli i rekomendacje**

* **k-NN (n=9, uniform)**:
  + Zalety: Stabilniejsze wyniki między zbiorem uczącym a testowym, dobra trafność (85%) i wysoki F1 (0.86) na danych testowych.
  + Wady: Nieco niższa trafność niż XGBoost i odrobinę mniejsza zdolność do maksymalizowania F1.
* **XGBoost**:
  + Zalety: Wyższa trafność na zbiorze testowym (87%), wyższy F1 (0.89), bardzo dobra czułość i generalnie lepsza zdolność do rozróżniania klas. Mniej podatny na przeuczenie niż pierwotne modele k-NN.
  + Wady: Nieznacznie niższa specyficzność i wyższy wskaźnik fałszywie pozytywnych niż w najlepszym k-NN, choć wciąż na akceptowalnym poziomie.

Biorąc pod uwagę wszystkie powyższe aspekty, XGBoost okazuje się bardziej efektywnym modelem dla naszych danych, potrafiącym lepiej wykorzystać dostępne predyktory i osiągającym wyższe miary jakości. Choć k-NN (n=9, uniform) również prezentuje dobre wyniki oraz mniejszą podatność na przeuczenie niż model k-NN wskazany przez GridSearchCV, to ostatecznie XGBoost uzyskał lepsze rezultaty pod względem kluczowych miar.

**6.4. Podsumowanie**

* Początkowo wybrany model k-NN (n=3, uniform) okazał się przeuczony, jednak modyfikując liczbę sąsiadów znaleziono stabilniejszą konfigurację (n=9, uniform).
* Mimo to, ostatecznie XGBoost okazał się modelem o najlepszych ogólnych parametrach jakości, oferując wyższą trafność i F1 na zbiorze testowym.
* Ostateczna rekomendacja to model XGBoost, który gwarantuje lepszą jakość predykcji i lepszą zdolność do generalizacji na nowe dane.