

**WYDZIAŁ GEOLOGII, GEOFIZYKI I OCHRONY ŚRODOWISKA**

KATEDRA GEOINFORMATYKI I INFORMATYKI STOSOWANEJ

Projekt inżynierski

*Pakiet w języku Python do optymalizacji procesów analizy danych tabelarycznych oraz modelowania regresyjnego  
i klasyfikacyjnego z wykorzystaniem AI*

*Python package for optimizing processes of tabular data analysis, regression and classification modeling using AI*

Autor: *Filip Andrzej Hałys*

Kierunek studiów: Geoinformatyka

Opiekun pracy: *Dr inż. Monika Chuchro*

Kraków, 2024

**Spis treści**

[1. WSTĘP 3](#_Toc182858978)

[1.1. Cel 3](#_Toc182858979)

[2. TECHNOLOGIA I METODY 4](#_Toc182858980)

[2.1. Środowisko programistyczne 4](#_Toc182858981)

[2.2. Środowisko wirtualne 4](#_Toc182858982)

[2.3. Tworzenie pustego pakietu 5](#_Toc182858983)

[2.4. System kontroli wersji 5](#_Toc182858984)

[3. OGÓLNY OPIS PAKIETU 6](#_Toc182858985)

[3.1. STANDARDY TWORZENIA POSZCZEGÓLNYCH PODMODUŁÓW 6](#_Toc182858986)

[3.2. WYKORZYSTANE PAKIETY 6](#_Toc182858987)

[4. SZCZEGÓŁOWY OPIS POSZCZEGÓLNYCH PODMODUŁÓW 7](#_Toc182858988)

[4.1. PODMODUŁ ‘\_errors’ 7](#_Toc182858989)

[4.2. PODMODUŁ ‘check’ 9](#_Toc182858991)

[4.3. PODMODUŁ ‘transformations’ 15](#_Toc182858992)

[5. PRZYKŁADY UŻYCIA 16](#_Toc182858993)

[6. PODSUMOWANIE, DYSKUSJA, WNIOSKI 17](#_Toc182858994)

[7. LITERATURA 18](#_Toc182858995)

# WSTĘP

Język Python jest jednym z najczęściej aktualnie wykorzystywanych języków programowania. W wielu rankingach znajduje się nawet na 1 miejscu; chociażby według TIOBE INDEX nieprzerwanie od ponad roku Python dzierży miano najbardziej popularnego języka programowania. Swoją pozycję zawdzięcza niewątpliwie dzięki prostej składni, szerokiemu spektrum zastosowań, czy rozwojowi sztucznej inteligencji. Relację pomiędzy rozwojem sztucznej inteligencji   
i językiem Python można określić jako mutualizm. Zdecydowanie tak szerokie wykorzystywanie modeli AI w dzisiejszym świecie jest spowodowane łatwością ich tworzenia, czy implementacji. W tworzeniu tego typu rozwiązań znacznie pomaga język Python i dedykowane dla niego pakiety (między innymi PyTorch, scikit-learn, czy Keras). Z drugiej strony Python również korzysta z przyspieszającego tempa rozwoju AI. Dzięki stworzonym pakietom do tworzenia sieci neuronowych, dużych modeli językowych, czy rozwiązywania problemów uczenia maszynowego Python znacznie zyskał na popularności. Wymienione wcześniej pakiety, to tylko przykłady gotowych rozwiązań, z których użytkownik może korzystać w celu budowy modelów AI. Takich pakietów istnieje bardzo dużo, co więcej atutem Pythona jest możliwość tworzenia własnych, personalizowanych pakietów dedykowanych dla określonych problemów.

# Cel

Podstawowym celem projektu inżynierskiego było utworzenie własnego, personalizowanego pakietu. Jako temat przewodni pakietu wybrano szeroko pojęty proces analizy danych oraz powiązany z nim proces modelowania regresyjnego   
i klasyfikacyjnego, za pomocą wybranych algorytmów uczenia maszynowego. Skupiono się głównie na optymalizacji tychże procesów. Poprzez optymalizację rozumie się ułatwienie, przyspieszenie i poukładanie składowych wymienionych powyżej procesów.

Głównym czynnikiem wpływającym na wybór tego tematu była ciekawość, w jaki sposób przebiega proces tworzenia pakietu od samego początku do finalnego etapu, w którym użytkownik jest w stanie ów pakiet wykorzystać. Dodatkowo, celem było także wykonanie funkcjonalnego pakietu z którego mogą korzystać osoby uczące się analizy danych, modelowania regresyjnego lub klasyfikacyjnego oraz eksploracji danych.

# TECHNOLOGIA I METODY

W tej sekcji skupiono się na wykorzystanych technologiach w projekcie, a także opisie użytych algorytmów uczenia maszynowego.

## Środowisko programistyczne

Do implementacji kodu posłużono się środowiskiem PyCharm od firmy JetBrains [cyt dokumentacji lub strony]. Jest to jedno z dwóch głównych narzędzi wykorzystywanych do tworzenia oprogramowania w języku Python obok programu Microsoft Studio Code. Wybrano akurat to środowisko, gdyż w przeciwieństwie do innych zapewnia ono pomoc przy pisaniu dobrze wyglądającego i czytelnego kodu.

## Środowisko wirtualne

W projekcie wykorzystano również możliwość utworzenia własnego środowiska wirtualnego. Zainicjowano je za pomocą narzędzia „anaconda3”. Jest to popularne narzędzie wspomagające zachowanie ładu i porządku w projektach. Za jego pomocą tworzenie własnych środowisk dedykowanych pod konkretny problem czy projekt jest proste i bardzo szybkie [cyt].

Na początku w aplikacji Anaconda Navigator utworzono środowisko o nazwie ENV-FOR-ET i zapisano je w lokalizacji ‘~\anaconda3\envs\ENV-FOR-ET’. Następnie dostosowano interpreter Pythona w Pycharmie, w taki sposób aby kompilowany kod wywoływał się przy pomocy utworzonego wcześniej środowiska. Po konfiguracji interpretera utworzono plik typu yaml; environment.yml. Celem jego stworzenia było ułatwienie zarządzania pakietami wewnątrz środowiska wirtualnego. Poniżej na Rys. 1 zaprezentowano jego zawartość:



Rys. 1 – zawartość pliku environemnt.yml

W kolejnym etapie zainstalowano narzędzie conda-lock. Jego zdaniem jest blokowanie wersji pakietów, co może być pomocne przy próbie odtworzenia jeden do jeden środowiska, np. na innym urządzeniu. Po instalacji zablokowano wersje pakietów. W celu instalacji i stworzenia blokady wykorzystano polecenia:

*conda install -c conda-forge conda-lock* – instalacja narzędzia cond-lock  
*conda-lock lock --file environment.yml* – utworzenie blokady

Początkowo nie zadeklarowano wszystkich niezbędnych pakietów w sekcji ‘dependencies’. Kolejne pakiety dołączano do środowiska wraz z rozwojem projektu przy pomocy poleceń:

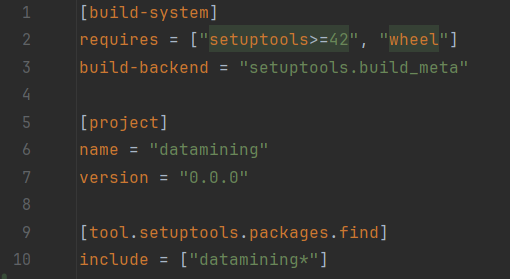
*conda activate base* – aktywowanie domyślnego środowiska Condy

*conda-lock -f environment.yml –* stworzenie na nowo pliku conda-lock.yml

*conda-lock install --name ENV-FOR-ET conda-lock.yml* – instalacja pakietów za pomocą pliku conda-lock.yml

## Tworzenie pustego pakietu

Po przygotowaniu środowiska do pracy utworzono pusty pakiet. W tym celu wykonano trzy kroki. Po pierwsze utworzono pusty folder, będący głównym folderem pakietu. Nadano mu nazwę „datamining”. Następnie utworzono plik pyproject.toml, którego zawartość zaprezentowano na Rys. 2:

  
*Rys. 2 – zawartość pliku pyproject.toml*

W pliku tym zadeklarowano nazwę pakietu (zgodną z nazwą folderu) oraz jego wersję (0.0.0). Wskazano również, iż zawartość pakietu jest kompatybilna   
z zawartością utworzonego w kroku pierwszym folderu. Finalnie wywołano polecenie: ‘pip install -e .’ w celu instalacji pakietu.

Od tego momentu każda napisana funkcja lub klasa obiektów wewnątrz folderu datamining stała się składową tegoż pakietu.

## System kontroli wersji

W trakcie tworzenia pakietu wykorzystywano system kontroli wersji GIT. Na początku utworzono repozytorium wykorzystując portal Github. Następnie sukcesywnie wraz z rozwojem pakietu dodawano do niego wprowadzane zmiany. Pozwoliło to na skuteczne i uporządkowane zarządzanie rozwojem projektu. Utworzenie repozytorium miało również dodatkowy cel; umożliwić udostępnienie funkcji wchodzących w skład pakietu innym użytkownikom.

# OGÓLNY OPIS PAKIETU

Pakiet nazwano „datamining”. W jego skład wchodzi dokładnie 6 podmodułów:

* **\_errors** – podmoduł pomocniczy, niewidzialny z perspektywy użytkownika, głównie przeznaczony do wyświetlania informacji o błędach popełnionych przez użytkownika w momencie tworzenia obiektu klasy lub wywoływania funkcji,
* **check** – podmoduł służący do zaznajamiania się z ramką danych przez użytkownika, przechowujący funkcje wyświetlające podstawowe statystyki badanych danych,
* **preprocessing** – niewielki podmoduł odpowiedzialny za przeprowadzanie preprocessingu, czyli wstępnego ‘czyszczenia’ danych,
* **transformations** – rozbudowany podmoduł przechowujący funkcje transformujące zestaw danych,
* **regression** – podmoduł zbudowany z dwóch dużych klas, dzięki którym możliwe jest optymalizowanie tworzenia wydajnych modeli regresji liniowej,
* **classification** – podmoduł zbudowany z jednej klasy, dzięki której możliwe jest tworzenie obiektów w postaci dobrej jakości klasyfikatorów.

## STANDARDY TWORZENIA POSZCZEGÓLNYCH PODMODUŁÓW

Informacje o opisach funkcji, komentarzach it

## WYKORZYSTANE PAKIETY

Pakiety które wykorzystałem + ich wersje + linki do oficjalnych dokumentacji + wykorzystana wersja Pythona

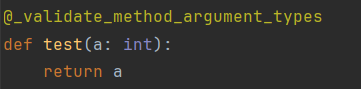
# SZCZEGÓŁOWY OPIS POSZCZEGÓLNYCH PODMODUŁÓW

## PODMODUŁ „\_errors”

Podmoduł ten utworzono w celu przechowywania metod obsługujących błędy w przekazywaniu parametrów przez użytkownika podczas tworzenia instancji klas lub wywoływania innych metod znajdujących się w obrębie całego pakietu. Innymi słowy podmoduł ten jest podmodułem pomocniczym pozwalającym na zatrzymywanie działania metody bądź tworzenia instancji klasy. Zatrzymanie to odbywa się w momencie gdy użytkownik jako argument do wywoływanej metody lub tworzonej instancji klasy poda parametr o innym typie niż oczekiwany. Wewnątrz tego podmodułu utworzono dwie funkcje; jedną w postaci dekoratora, drugą w postaci zwykłej metody.

Pierwsza z nich (dekorator) *\_validate\_method\_argument\_types* pozwala na wyświetlenie błędu o treści: „Argument <nazwa argumentu> must be of type <oczekiwany typ>, got <aktualny typ> instead.”. Dzięki podniesieniu błędu o takiej treści użytkownik popełniający błąd przy wywoływaniu metody wchodzącej w skład pakietu jest w stanie zweryfikować, który podany przez niego argument ma niepoprawny typ. Podniesienie błędu powoduje bezwarunkowe zatrzymanie tej metody i niewykonywanie jej dalszego działania. Poniżej zaprezentowano prosty przykład zastosowania dekoratora *\_validate\_method\_argument\_types*:

* Utworzono funkcję *test* przyjmującą jeden argument *a* typu integer   
  i zwracającą jego wartość, nałożono na nią dekorator:

*   
Rys. 3 – Definicja funkcji test z nałożonym dekoratorem*

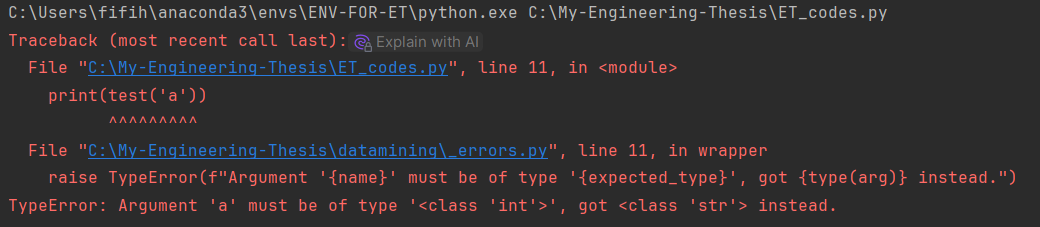
* Wywołanie funkcji z argumentem *a* o typie integer (zgodnym z definicją funkcji) powoduje wykonanie się funkcji (dekorator nie podnosi błędu):

*  
Rys. 4 – Wywołanie funkcji test z poprawnym typem argumentu a*

*Rys. 5 – Wynik działania funkcji test z podanym poprawnym typem   
argumentu a*

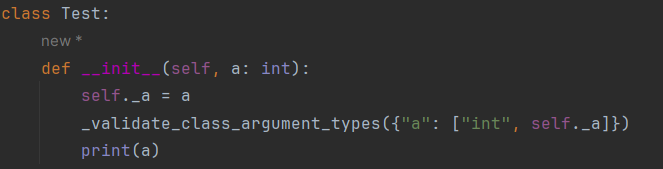
* Wywołanie funkcji z argumentem *a*, o innym typie niż integer (niezgodnym   
  z definicją funkcji) powoduje podniesienie błędu przez dekorator:

 *Rys. 6 – Wywołanie funkcji test z niepoprawnym typem argumentu a*

*Rys. 7 – Wynik działania funkcji test z niepoprawnym podanym typem argumentu a*

Drugą metodę *\_validate\_class\_argument\_types* zaimplementowano nieco inaczej, nie jako dekorator, a jako zwykłą funkcję. Przyjmuje ona jeden argument w postaci słownika, w którego skład wchodzą klucze i wartości. Klucze określają nazwę argumentu, wartości natomiast są dwuelementową listą, w skład której wchodzi oczekiwany typ (element listy o indeksie 0) i przekazany typ przez użytkownika (element listy o indeksie 1). Zarówno klucz, jak i typy argumentów przekazane   
w liście są przekazywane jako tekst (string). Funkcja ta wywoływana jest   
w konstruktorze klasy. Użytkownik podając nieodpowiedni typ jednego   
z parametrów podczas tworzenia instancji klasy (w której zaimplementowano wywołanie tej metody pomocniczej) jako wynik otrzyma błąd o analogicznej treści jak w przypadku powyższego dekoratora. Stworzenie instancji się nie powiedzie. Poniżej zaprezentowano prosty przykład działania metody *\_validate\_class\_argument\_types*:

* Utworzono klasę *Test*, której konstruktor przyjmuje 1 argument *a* typu integer i wypisuje w terminalu jego wartość.

  
*Rys. 8 - Definicja klasy Test z użytą funkcją sprawdzającą typy argumentów konstruktora*

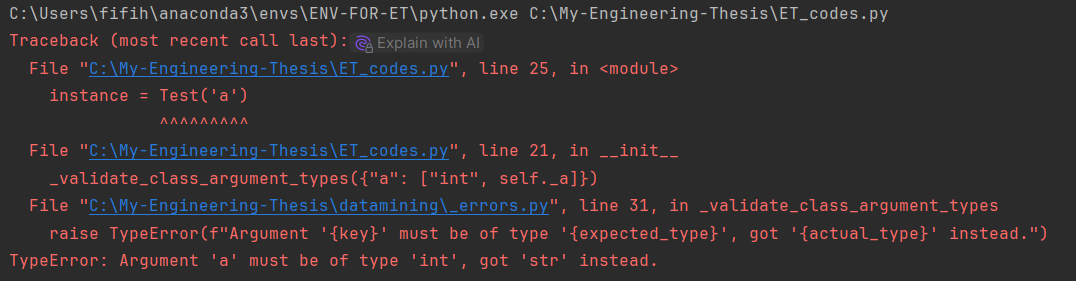
* Utworzenie instancji klasy *Test* z argumentem *a* podanym do konstruktora klasy o typie integer (zgodnym z definicją konstruktora) powoduje stworzenie się instancji (metoda nie podnosi błędu):

*  
Rys. 9 – Utworzenie instacji klasy Test z poprawnym typem argumentu a konstruktora*

  
*Rys. 10 – Wynik utworzenia się instancji klasy Test z poprawnym typem argumentu a konstruktora*

* Utworzenie instancji klasy *Test* z argumentem *a* podanym do konstruktora klasy o typie innym niż integer (niezgodnym z definicją konstruktora) powoduje podniesie błędu przez metodę *\_validate\_class\_argument\_types*

  
*Rys. 11 – Utworzenie instancji klasy Test z niepoprawnym typem argumentu a konstruktora*

 *Rys. 12 – Wynik utworzenia się instancji klasy Test z niepoprawnym typem argumentu a konstruktora*

Każda opisana klasa bądź metoda w kolejnych podmodułach ma zaimplementowany system podnoszenia błędów opisany powyżej ze względu na podane typy argumentów. Dzięki temu rozwiązaniu użytkownik wykorzystujący metody i klasy   
z pakietu jest w stanie wykrywać błędy jakie popełnił przy ich wywoływaniu lub tworzeniu.

## PODMODUŁ „check”

Podmoduł „check” został utworzony w celu ułatwienia użytkownikowi zapoznawania się z badaną ramką danych. Wewnątrz tego pakietu znajdują się cztery metody: *check\_numeric\_data*, *check\_category\_data*, *check\_time\_series\_data*, *check\_time\_interval\_data* służące sprawdzaniu podstawowych statystyk zmiennych o czterech różnych rodzajach zmiennych (każda metoda dla danego rodzaju zmiennej). Wyróżnione rodzaje zmiennych to:

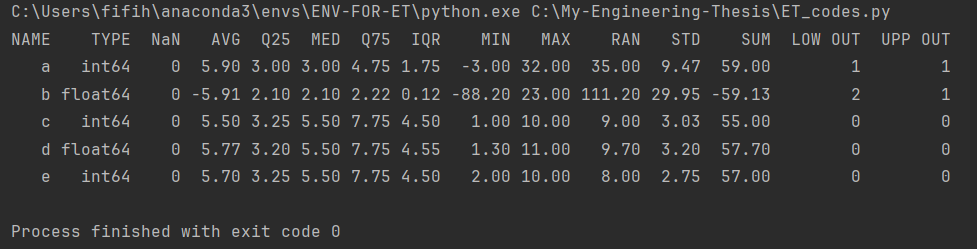
* zmienne numeryczne, np. 1; 1.12; 1+2i; -4
* zmienne kategoryczne, np. ‘a’; ‘man’; True
* zmienne w postaci szeregów czasowych, np. 7 maj 2024 01:02:03.45
* zmienne w postaci interwałów czasowych, np. 2 lata 3 dni 4 godziny; 3 minuty 3 sekundy

Każda z metod przyjmuje nieco inne argumenty, jak również zwraca nieco inne parametry statystyczne. Cechą wspólną dla tych metod są dwa argumenty; argument *df* (czyli badana ramka danych), a także argument *use* (przyjmujący wartości *True* lub *False*, określający czy użytkownik chce na wyjściu otrzymać wynik odpowiednio w postaci ramki danych lub tekstu, domyślnie argument ten przyjmuje wartość *False*). Niezależnie od wartości przekazanej jako argument *use* po wywołaniu metody użytkownik otrzyma tabelkę z każdym polem (kolumną) ramki danych danego typu oraz opisującymi je statystykami. W przypadku gdy w ramce danych nie istnieje żadna kolumna o typie, który użytkownik chce zbadać, w terminalu wyświetla się odpowiednia informacja.

* + 1. Metoda *check\_numeric\_data*

Metoda *check\_numeric\_data* przyjmuje dodatkowo argument *round* określający ilość cyfr do których użytkownik chce zaokrąglić wynikowe statystyki. Domyślnie przyjmuje on wartość 1. Przykładowe wywołanie tej metody dla ramki danych   
o nazwie *ramka\_testowa*, w której skład wchodzi pięć kolumn numerycznych (*a*, *b*, *c*, *d*, *e*) przedstawiono na Rys. 13. Wynik działania tej metody zaprezentowano na Rys. 14

  
*Rys. 13 – Przykładowe wywołanie metody check\_numeric\_data*

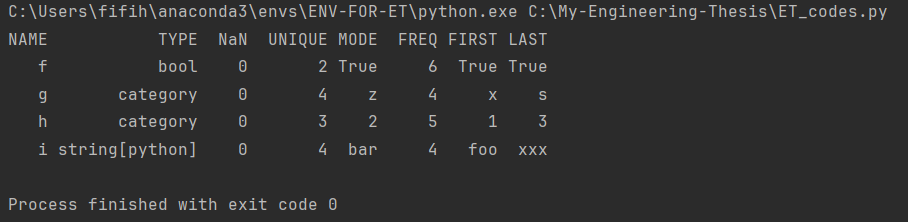
*Rys. 14 – Przykładowy wynik działania metody check\_numeric\_data*

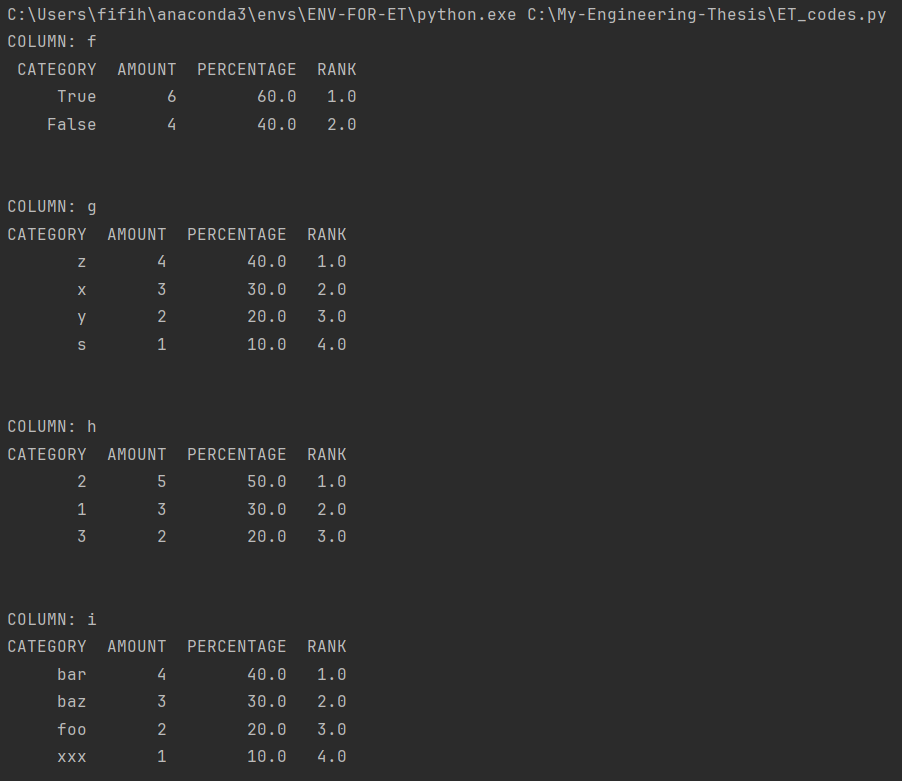
Jak przedstawiono na Rys. 14, użytkownik dzięki uruchomieniu tej metody jest   
w stanie dowiedzieć się o:

* NAME – nazwie kolumny numerycznej,
* TYPE – typie zmiennej,
* NaN – ilości brakujących obserwacji,
* AVG – wartości średniej,
* Q25 – dolnym kwartylu (25% obserwacji jest mniejszych od tej wartości),
* Q50 – medianie,
* Q75 – górnym kwartylu (25% obserwacji jest większych od tej wartości),
* IQR – rozstępie międzykwartylowym (różnicy między Q75 i Q25),
* MIN – wartości minimalnej,
* MAX – wartości maksymalnej,
* RAN – zakresie (różnicy między MAX i MIN),
* STD – odchyleniu standardowym,
* SUM – sumie wartości,
* LOW OUT – ilości dolnych wartości odstających (ilość obserwacji mniejszych od Q25 – 1.5\*IQR)
* UPP OUT – ilości górnych wartości odstających (ilości obserwacji większych od 0.75 + 1.5\*IQR)
  + 1. Metoda check\_category\_data

W metodzie tej dodano jeden dodatkowy argument *cat\_dist* przyjmujący wartość *True* lub *False*. Określa on czy użytkownika potrzebuje wyświetlić podstawowe statystyki analogicznie do metody *check\_numeric\_data* (*False*), czy chce zobaczyć rozkład każdej ze zmiennej kategorycznej (*True*). Przykład wywołania tej metody przedstawiono na Rys. 15. Wyniki jej działania zaprezentowano na Rys. 16   
i Rys. 17. Do pokazania rezultatów użyto przykładowej ramki danych o nazwie *ramka\_danych* przechowującej 4 kolumny kategoryczne (*f*, *g*, *h*, *i*):

*Rys. 15 – Przykład wywołania metody check\_category\_data*

*Rys. 16 – Przykładowy wynik wywołania metody check\_category\_data   
z argumentem cat\_dist utawionym na wartość False*

*Rys. 17 – Przykładowy wynik wywołania metody check\_category\_data   
z argumentem cat\_dist ustawionym na wartość True*

W przypadku przedstawionym na Rys. 16 cechy opisujące zmienną kategoryczną to:

* NAME – nazwa,
* TYPE – typ,
* NaN – ilość brakujących wartości kategorii,
* UNIQUE – ilość unikatowych kategorii,
* MODE – kategoria modalna, najczęściej występująca,
* FREQ – ilość wystąpień kategorii modalnej,
* FIRST – kategoria występująca pod pierwszym indeksem (indeks 0),
* LAST – kategoria występująca pod ostatnim indeksem (indeks -1).

W przypadku przedstawienia rozkładu kategorii na Rys. 17, statystyki opisujące to:

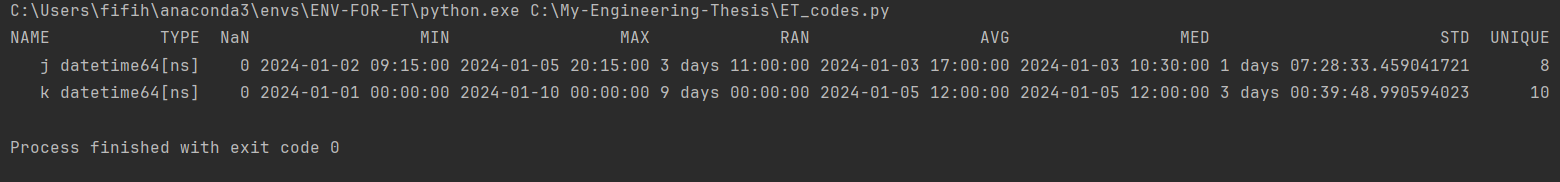
* CATEGORY – nazwa kategorii,
* AMOUNT – ilość wystąpień,
* PERCENTAGE – procentowy udział kategorii w zbiorze danych
* RANK – pozycja kategorii (1-występuje najczęściej), podsumowanie każdej kolumny posortowano względem tego pola
  + 1. Metoda *check\_time\_series\_data*

Metoda ta podsumowywuje kolumny ramki danych w postaci szeregów czasowych. Statystyki wyświetlające się w terminalu w przypadku tej metody to:

* NAME – nazwa kolumny,
* TYPE – typ zmiennej,
* NaN – ilość brakujących obserwacji,
* MIN – najwcześniejszy moment czasu,
* MAX – najpóźniejszy moment czasu,
* AVG – średni moment czasu,
* MED – mediana momentów czasu,
* STD – odchylenie standardowe momentów czasu,
* UNIQUE – ilość unikatowych momentów czasu.

Przykład wywołania tej metody na prostej, dziesięcioelementowej ramce danych *ramka\_testowa* posiadającej dwie kolumny *j* oraz *k* o typie datetime przedstawiono na Rys. 18. Ponadto na Rys. 19 zaprezentowano output po wywołaniu tej metody.

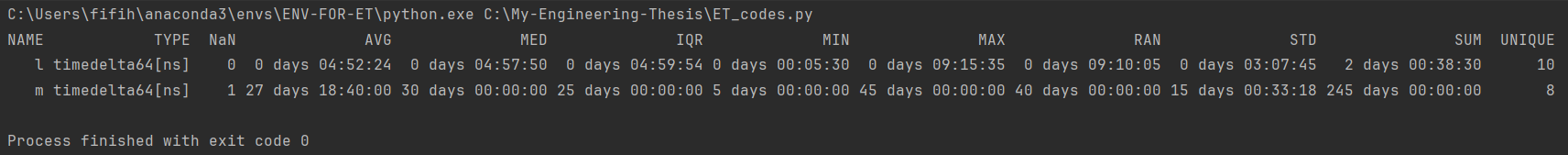
*  
Rys. 18 – Przykład wywołania metody check\_time\_series\_data*

*  
Rys. 19 - Przykładowy wynik wywołania metody check\_time\_series\_data*

* + 1. Metoda *check\_time\_interval\_data*

Ostatnią zaimplementowaną metodą podsumowywującą konkretny rodzaj zmiennych w tym podmodule jest *check\_time\_interval\_data*. Służy ona do podsumowywania zmiennych w postaci interwałów czasowych. Poniżej na Rys. 20 zaprezentowano przykład wywołania tej metody na testowej ramce danych   
o nazwie *ramka\_restowa* zbudowanej z dwóch kolumn *l* oraz *m* w postaci interwałów czasowych.

*  
Rys. 20 - Przykład wywołania metody check\_time\_interval\_data*

*  
Rys. 21 – Przykładowy wynik wywołania metody check\_time\_interval\_data*

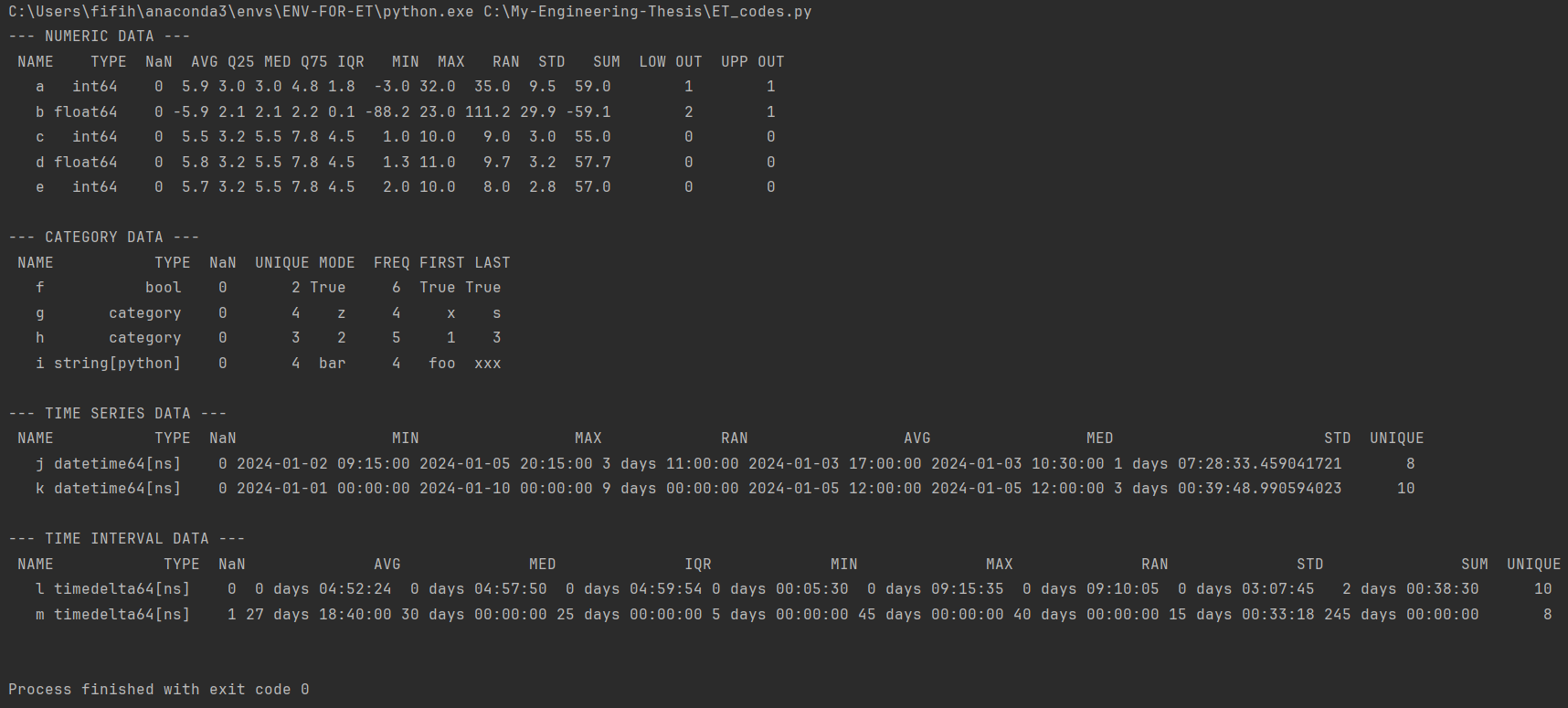
Przykładowy wynik zapytania z Rys. 20 przedstawiono na Rys. 21. Wyświetlone na nim statystyki to:

* NAME – nazwa kolumny,
* TYPE – typ zmiennych,
* NaN – ilość brakujących obserwacji,
* AVG – wartość średnia interwałów czasowych,
* MED. – mediana, wartość środkowa interwałów czasowych,
* IQR – Rozstęp międzykwartylowy interwałów czasowych,
* MIN – najkrótszy interwał czasowy,
* MAX – najdłuższy interwał czasowy,
* RAN – różnica między MAX i MIN, zakres,
* STD – odchylenie standardowe interwałów czasowych,
* SUM – suma wszystkich interwałów czasowych w danej kolumnie,
* UNIQUE – ilość unikalnych wartości interwałów czasowych.
  + 1. Metoda *check\_data*

Ostatnią metodą w tym pakiecie jest metoda *check\_data*. Służy ona wyświetlaniu wyników czterech poprzednich metod. Jest ona niejako podsumowaniem statystycznym danych wewnątrz ramki danych. Poniżej na Rys. 22 zaprezentowano sposób wywoływania tej metody.

*  
Rys. 22 - Przykład wywołania metody check\_data*

W przypadku nieistnienia jednego z czterech rodzajów typów zmiennych metoda zwróci odpowiednią informację. Na Rys. 23 zaprezentowano wynik działania tej metody na testowej ramce danych *ramka\_testowa*.

  
Rys. 23 - *Przykładowy wynik wywołania metody check\_data*

## PODMODUŁ „transformations”

Podmoduł ten złożony jest z siedmiu metod transformujących dane wchodzących   
w skład ramki danych. Transformacja danych jest bardzo szerokim pojęciem związanym z procesami przygotowującymi zestaw danych przede wszystkim pod stworzenie modelu uczenia maszynowego na jego podstawie. Polega na przekształceniu danych w taki sposób, aby pozbyć się negatywnego wpływu wartości, które mogłyby zbyt natrętnie lub niepoprawnie oddziałowywać na model. Wybrane metody transformacji danych to standaryzacja, normalizacja min-max, transformacja logarytmiczna, normalizacja boxa-koxa, transformacja pierwiastkowa, binaryzacja oraz kodowanie one-hot. Dla każdej spośród podanych typów transformacji stworzono osobną metodę. W każdym przypadku metoda ta na wejściu przyjmuje ramkę danych (argument *df*) oraz nazwę kolumny/kolumn (argument *columns*) w której znajdują się dane, które użytkownik chce przetransformować. Argument *df* zawsze musi przyjmować ramkę danych   
z pakietu Pandas, natomiast argument *columns* może być w postaci stringu –   
w przypadku gdy użytkownik chce zadziałać funkcją transformującą tylko na jednej kolumnie, jak i w postaci listy stringów – gdy kolumn do transformacji jest więcej niż jedna.

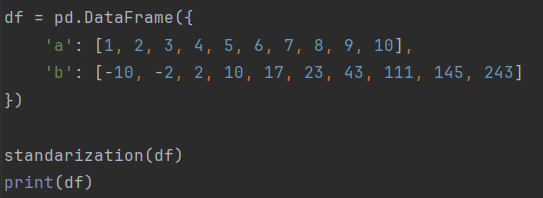
* + 1. Metoda *standarization*

Standaryzacja danych jest procesem przekształcania danych w taki sposób, aby dane wyjściowe charakteryzowały się; średnią równą zero oraz odchyleniem standardowym równym jeden. Jest to proces niezwykle potrzebny chociażby w przypadku tworzenia modeli regresji liniowej czy regresji logistycznej, gdy ma się do czynienia z różną skalą zmiennych numerycznych. W procesie standaryzacyjnym każda z obserwacji (k) otrzymuje nową wartość zgodnie ze wzorem:

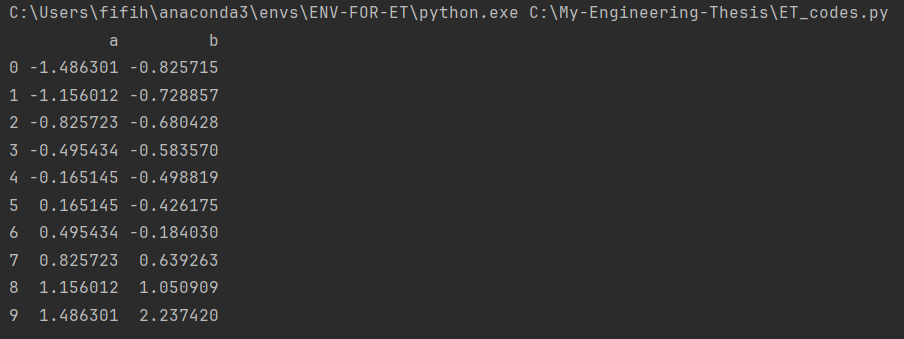
*yk = (xk – xśr) / xstd*

gdzie:yk – wartość ustandaryzowana  
xk – wartość zaobserwowana  
xśr – średnia z całej populacji zmiennej x

xstd – odchylenie standardowe całej populacji zmiennej x

*  
Rys. 24 - Przykład wywołania metody standarization*

Na rysunku powyżej Rys. 24 zaprezentowano wywołanie tej metody na przykładowej 10-cio elementowej ramce danych. Poniżej na Rys. 25 przedstawiono wynik działania tej metody. Dane nie są postaci takiej, jak początkowo przy tworzeniu ramki, a zostały ustandaryzowane.

*Rys. 25 - Przykładowy wynik wywołania metody standarization*

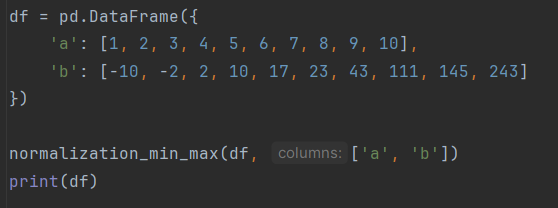
* + 1. Metoda *normalization\_min\_max*

Jest to pierwsza z metod normalizujących dane wewnątrz tego podmodułu. Używa do tego jednego z najbardziej popularnych algorytmów min-max. Każdą   
wartość k oblicza się na nowo korzystając ze wzoru:

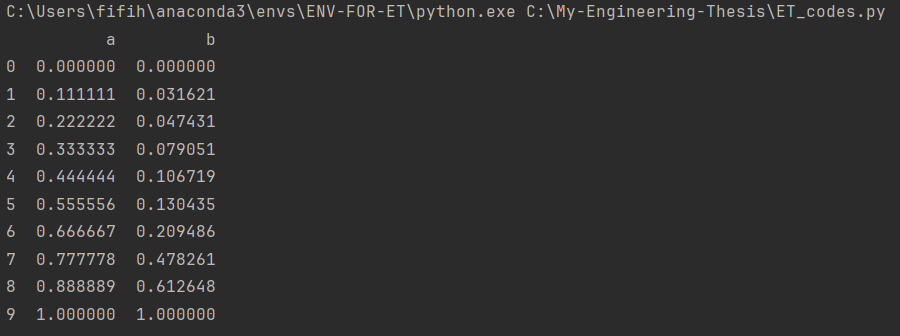
*yk = (xk – xmin) / (xmax – xmin)*

gdzie:  
yk – wartość znormalizowana  
xk – wartość zaobserwowana  
xmin – wartość minimalna występująca w populacjixmax – wartość maksymalna występująca w populacji

Normalizacja min-max ma bardzo podobne zastosowanie co standaryzacja. Podczas wykonywania tej operacja na danych zostają one obliczone z powyższego wzoru, przez co wartość minimalna w nowym rozkładzie osiąga wartość 0, a wartość maksymalna 1. Poniżej na Rys. 26 pokazano przykładowe wywołanie metody *normalization\_min\_max* na przykładowej ramce danych.

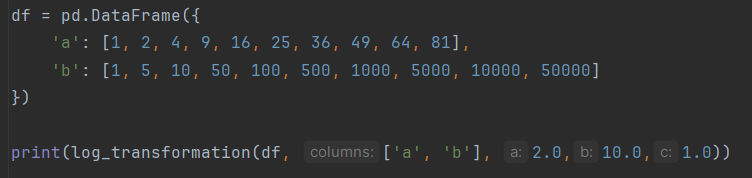
  
*Rys. 26 - Przykład wywołania metody normalization\_min\_max*

Wynik operacji z Rys. 26 zaprezentowano na Rys. 27. Przedstawia on dane znormalizowane za pomocą algorytmu min-max.

 *Rys. 27 - Przykładowy wynik wywołania metody normalization\_min\_max*

* + 1. Metoda *log\_transformation*

Metoda ta odpowiedzialna jest za przeprowadzanie transformacji logarytmicznej na zmiennej. Transformacja logarytmiczna jest kolejnym przykładem przekształcania danych. Polega ona na wyciągnięciu logarytmu z każdej obserwacji w danej kolumnie ramki danych. Jest to niezwykle przydatne narzędzie, w przypadku gdy zakres danych ma bardzo dużą wartość, a wartości odstające mogą wypaczyć wynik prowadzonych badań. Poprzez przeprowadzenie transformacji logarytmicznej wpływ tychże wartości zmniejsza się, często pozwalając na zbudowanie modelu (np. regresyjnego) bez negatywnego wpływu wartości znacznie odstających.

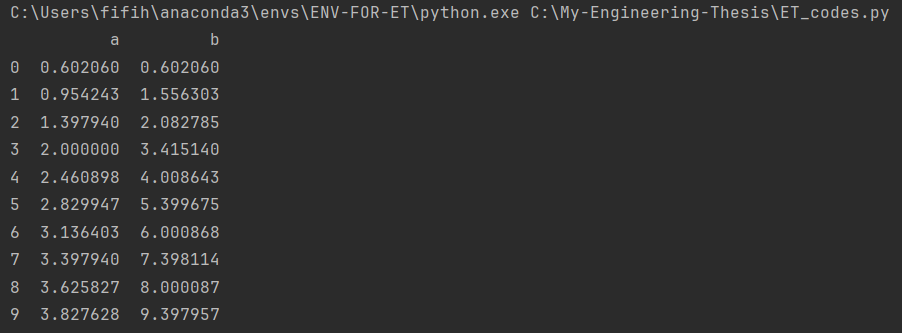
*   
Rys. 28 - Przykład wywołania metody log\_transformation*

Na rysunku powyżej Rys. 28 zaprezentowano przykład wywołania tej metody. Jak widać przyjmuje ona oprócz dwóch podstawowych argumentów; trzy dodatkowe. Są to argumenty *a*, *b* oraz *c*, każdy z nich musi być podany jako obiekt typu float. Pierwszy z nich, argument *a* określa współczynnik skalujący, drugi argument *b* jest podstawą logarytmu, trzeci natomiast jest przesunięciem (tzw. offsetem). Poniżej przedstawiono wzór, który pokazuje w jaki sposób obliczana jest nowa wartość dzięki tej funkcji transformacyjnej:

*yk = a \* log(xk + c)*

gdzie:  
yk – wartość znormalizowana  
xk – wartość zaobserwowana

Poniżej na Rys. 29 zaprezentowano wynik działania tej funkcji:

* Rys. 29 - Przykładowy wynik wywołania metody normalization\_min\_max*

Wartości przed wywołaniem funkcji znacznie różnią się od tych po zadziałaniu metody log\_transformation. Podstawową różnicą jest zakres danych; wynoszący niemal 50000 w przypadku drugiej kolumny przed procesem wywołania metody oraz niecałe 9 tuż po jej wywołaniu.

* + 1. Metoda *transformation\_box\_kox*

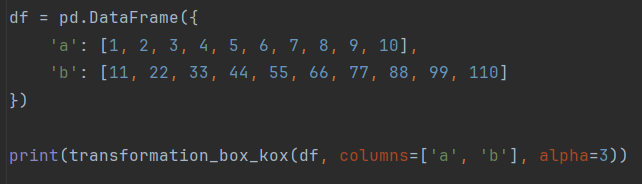
Kolejna metodą wewnątrz podmodułu „transformations” jest metoda do przekształcania danych używając do tego transformacji Boxa-Koxa. Transformacja ta podobnie do poprzednich działa na zmiennych numerycznych, a jej głównymi atutami jest możliwość dostosowania wariancji danych, pozbycie się zbyt dużej skośności danych, także przybliżenie rozkładu danych do rozkładu normalnego. Jej wynik zależy przede wszystkim od parametru λ. Każdy rekord danych oblicza się na nowo korzystając z formuły:

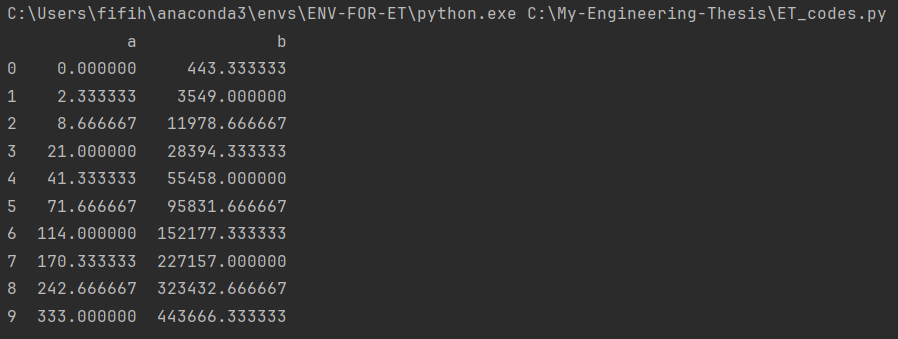
*{yk = (xk*λ*- 1) /* λ*, dla* λ ≠ *0} lub {yk = ln(xk), dla* λ *=0}*

gdzie:  
yk – wartość znormalizowana  
xk – wartość zaobserwowana

λ – parametr alpha

Zaimplementowana metoda *transformation\_box\_kox* posiada jeden dodatkowy parametr *alpha,* który odpowiada λ w powyższym wzorze. Dzięki dodaniu tej opcji użytkownik jest w stanie samodzielnie wybrać, jaką wartość ma ten parametr przyjąć. Przykładowe wywołanie tej metody przedstawiono na Rys. 30. Parametr *alpha* ustawiono na wartość 3.

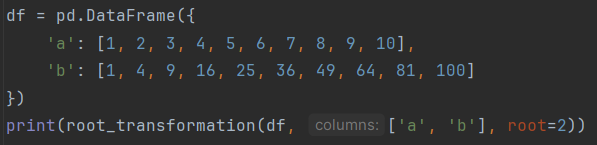
  
*Rys. 30 - Przykład wywołania metody transformation\_box\_kox*

  
*Rys. 31 - Przykładowy wynik wywołania metody normalization\_min\_max*

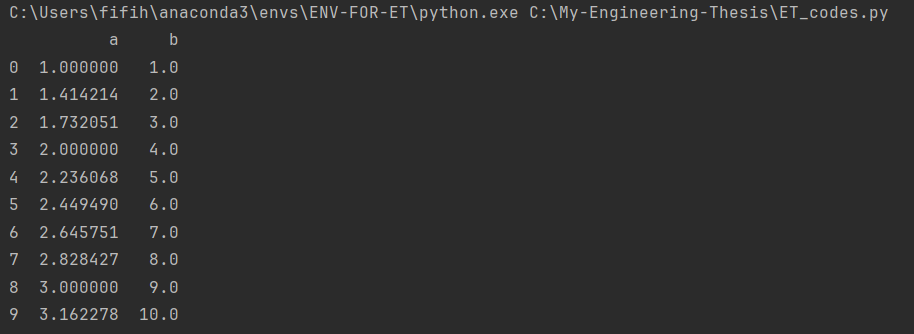
Jak zaprezentowano na Rys. 31 dane zostały przekształcone zgodnie ze wzorem podanym powyżej.

* + 1. *Metoda root\_transformation*

Metoda *root\_transformation* jest kolejną metodą wewnątrz podmodułu „transformations”, która transformuje dane numeryczne. Sposób w jaki dane są przekształcane polega na wyciągnięciu pierwiastka z każdej wartości wewnątrz kolumny numerycznej. Użytkownik jest w stanie dostosować indeks pierwiastka według własnej potrzeby edytując parametr *root* podczas wywoływania funkcji.

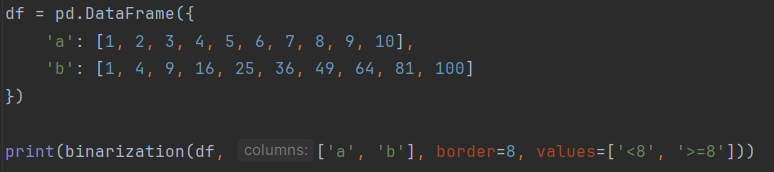
*  
Rys. 32 – Przykładowe wywołanie metody root\_transformation*

Powyżej na Rys. 32 przedstawiono przykład wywołania metody podając argument *root* równy 2. Oznacza to, że transformacja jaką ta metoda wykona na danych to pierwiastkowanie kwadratowe. Domyślnie argument *root* przyjmuje wartość 1, czyli nie podając żadnej wartości tego argumentu dane się nie zmienią. Poniżej na   
Rys. 33 zaprezentowano rezultat wywołania metody *root\_transformation*.

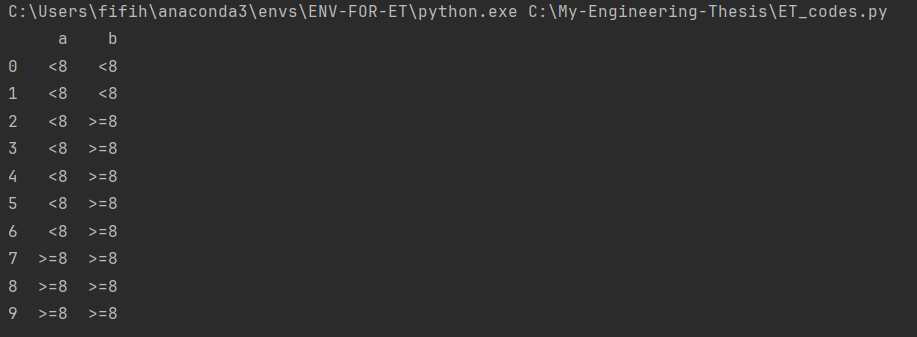
  
*Rys. 33 - Przykładowy wynik wywołania metody root\_transformation*

* + 1. Metoda *binarization*

Metoda *binarization* została utworzono do przeprowadzania binaryzacji danych. Binaryzacja danych jest procesem uzyskiwania tylko i wyłącznie 2 wartości z całej gamy zmiennych. Metodę tą zaimplementowano w taki sposób, aby użytkownik był w stanie samodzielnie ustalić wartość graniczną. Wartość ta oddziela dwie grupy danych; tą z wartościami niższymi oraz tą z wartościami wyższymi (bądź takimi samymi). Aby ustalić wartość graniczną użytkownik przy wywoływaniu funkcji musi zmienić parametr *border*. Domyślnie przyjmuje wartość 0 (ujemne wartości trafiają do jednej grupy danych, a nieujemne do drugiej). Po określeniu wartości granicznej wartości mniejsze trafiają do jednej grupy otrzymując nową, określoną wartość, podobnie jak wartości większe bądź równe (wartości granicznej) trafiają do drugiej grupy i również przyjmują nową wartość. To jakie wartości przyjmują obie grupy użytkownik jest w stanie zoptymalizować samodzielnie ustawiając parametr *values* zgodnie z własnymi oczekiwaniami. Parametr ten jest dwuelementową listą, z czego element o indeksie 0 jest wartością, którą przyjmują dane mniejsze niż wartość graniczna, natomiast element listy o indeksie 1 jest wartością, którą przyjmują dane większe bądź równe wartości granicznej. Domyślnie (nie podając żadnej wartości) argument *values* jest listą o zerowym elemencie równym 0, oraz pierwszym elemencie równym 1. Dodatkowo użytkownik jako elementy listy *values* może podać zmienne o bardzo różnym typie. Przykład wywołania metody binarization przedstawioni poniżej na Rys. 34.

  
*Rys. 34 – Przykładowe wywołanie metody binarization*

Przykładowo ustawiono wartość graniczną równą 8, zastępując wartości krótką informacją; czy wartość jest mniejsza od 8 czy nie. Na Rys. 35 zaprezentowano wynik działania tej metody.

  
*Rys. 35 - Przykładowy wynik wywołania metody root\_transformation*

Binaryzacja jest bardzo przydatną transformacją, służącą głównie do uproszczenia danych, czyli do redukcji ich złożoności. Przydatna jest przy tworzeniu modeli uczenia maszynowego, niekiedy zwiększając ich wydajność. Ponadto używana jest w takich dziedzinach jak analiza i przetwarzanie obrazów, czy przy tworzeniu tzw. Drzew decyzyjnych.

* + 1. Metoda *one\_hot\_encoding*

aaaaaaaaa

# PRZYKŁADY UŻYCIA

# PODSUMOWANIE, DYSKUSJA, WNIOSKI

# LITERATURA

https://www.tiobe.com/tiobe-index/