

**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](#_Toc186895190)

[1.1. Cel 3](#_Toc186895191)

[2. TECHNOLOGIA I METODY 4](#_Toc186895192)

[2.1. Środowisko programistyczne 4](#_Toc186895193)

[2.2. Środowisko wirtualne 4](#_Toc186895194)

[2.3. Tworzenie pustego pakietu 5](#_Toc186895195)

[2.4. System kontroli wersji 6](#_Toc186895196)

[3. OGÓLNY OPIS PAKIETU 7](#_Toc186895197)

[3.1. STANDARDY TWORZENIA POSZCZEGÓLNYCH PODMODUŁÓW 7](#_Toc186895198)

[3.2. WYKORZYSTANE PAKIETY 7](#_Toc186895199)

[4. SZCZEGÓŁOWY OPIS POSZCZEGÓLNYCH PODMODUŁÓW 8](#_Toc186895200)

[4.1. PODMODUŁ „\_errors” 8](#_Toc186895201)

[4.2. PODMODUŁ „check” 10](#_Toc186895203)

[4.3. PODMODUŁ „transformations” 16](#_Toc186895204)

[4.4. PODMODUŁ „preprocessing” 23](#_Toc186895205)

[4.5. PODMODUŁ „regression” 26](#_Toc186895206)

[4.6. PODMODUŁ „classification” 32](#_Toc186895207)

[5. PRZYKŁADY UŻYCIA 35](#_Toc186895208)

[6. PODSUMOWANIE, DYSKUSJA, WNIOSKI 36](#_Toc186895209)

[7. BIBLIOGRAFIA 37](#_Toc186895210)

# WSTĘP

# 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.

Język Python jest jednym z najczęściej aktualnie wykorzystywanych języków programowania. W wielu rankingach znajduje się nawet na 1 miejscu; chociażby według TIOBE INDEX [1] 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.

## Środowisko programistyczne

Do utworzenia pakietu, jego implementacji, prezentacji przykładów i case study posłużono się środowiskiem PyCharm od firmy JetBrains [2]. Jest to jedno z dwóch głównych narzędzi wykorzystywanych do tworzenia oprogramowania w języku Python obok programu Microsoft Studio Code [3]. 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 [4].

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 utworzenia było ułatwienie zarządzania pakietami wewnątrz środowiska wirtualnego. Poniżej na Rys. 2.1 zaprezentowano jego zawartość:

  
*Rys. 2.1 – zawartość pliku environemnt.yml*

W kolejnym etapie zainstalowano narzędzie conda-lock. Jego zadaniem 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:

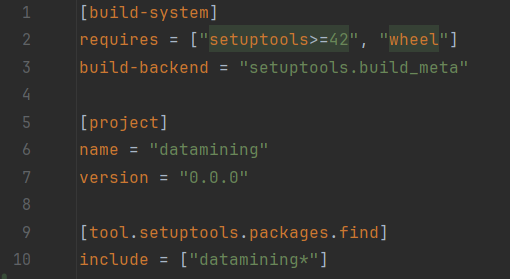
*conda activate base* – aktywowanie domyślnego środowiska Condy

*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. 2.2:

  
*Rys. 2.2 – 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

Każdą zaimplementowaną metodę bądź klasę obiektów opisano za pomocą komentarzy: *”””opis metody/klasy”””*. W skład każdego opisu wchodzi kilka składowych; nazwy i typy parametrów wejściowych, nazwy i typy parametrów wyjściowych oraz krótkie streszczenie na czym polega działanie opisywanej metody/klasy.

Ponadto, dodawano komentarze w najistotniejszych miejscach kodu bądź w miejscach, które mogą na pierwszy rzut oka nie jasne dla osoby tworzącej, rozwijającej metodę/klasę.

Każdą metodę/klasę przetestowano według wcześniej określonego scenariusza (scenariusz ten często zmieniał się wraz z trwaniem implementacji metody/klasy). Testy te polegały na wykorzystaniu metody/klasy dla różnych typów i wartości parametrów wejściowych oraz weryfikacji, czy na wyjściu otrzymano oczekiwane rezultaty. Dużą uwagę zwrócono na potencjalne błędy podczas wywoływania metod lub tworzenia instancji klas z niepoprawnymi/niezgodnymi typami parametrów wejściowych. Duży nacisk postawiono na dokładne opisywanie błędów w wywoływaniu metod/klas, tak aby użytkownik przekazujący niepoprawny typ lub wartość jako parametr wejściowy miał jasną i klarowną odpowiedź, dlaczego próba kompilacji nie przebiegła pomyślnie.

## WYKORZYSTANE PAKIETY

Do implementacji pakietu posłużono się językiem Python w wersji 3.12. Była to najnowsza wersja tego języka na czas pisania pracy. Wykorzystano kilka innych pakietów w najnowszych wersjach na czas pisania pracy. Tworząc środowisko wirtualne założono minimalne wersje pakietów (Rys. 2.2). Poniżej przedstawiono listę zaimportowanych pakietów oraz ich dokładnych wersji używanych do implementacji pakietu:

* Pandas w wersji 2.2.2 [5],
* Pip w wersji 24.0 [6],
* Scikit-learn w wersji 1.5.1 [7],
* Matplotlib w wersji 3.9.1 [8],
* Numpy w wersji 2.0.0 [9].

Do poprawnego działania opisywanego pakietu konieczne jest wcześniejsze zaimportowanie do środowiska wyżej wymienionych pakietów.

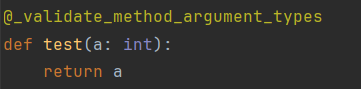
# SZCZEGÓŁOWY OPIS POSZCZEGÓLNYCH PODMODUŁÓW

## 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. 4.1 – Definicja funkcji test z nałożonym dekoratorem*

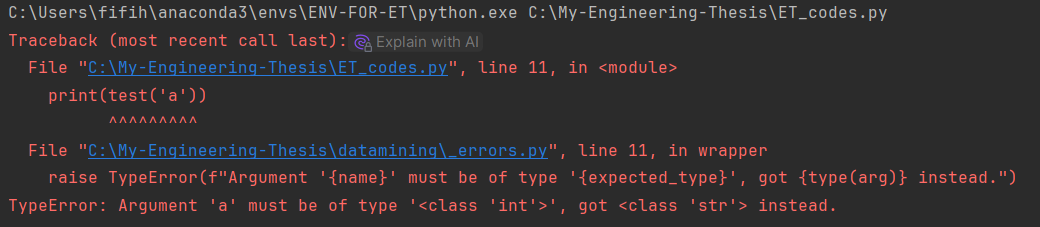
* Wywołanie funkcji z argumentem *a* o typie integer (zgodnym z definicją funkcji) powoduje wykonanie się funkcji (dekorator nie podnosi błędu):

  
*Rys. 4.2 – Wywołanie funkcji test z poprawnym typem argumentu a*

*Rys. 4.3 – Wynik działania funkcji test z podanym poprawnym typem argumentu a*

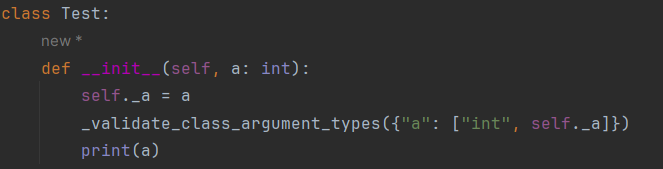
* Wywołanie funkcji z argumentem *a*, o innym typie niż integer (niezgodnym   
  z definicją funkcji) powoduje podniesienie błędu przez dekorator:

 *Rys. 4.4 – Wywołanie funkcji test z niepoprawnym typem argumentu a*

*Rys. 4.5 – 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. 4.6 - Definicja klasy Test z użytą funkcją sprawdzającą typy argumentów konstruktora*

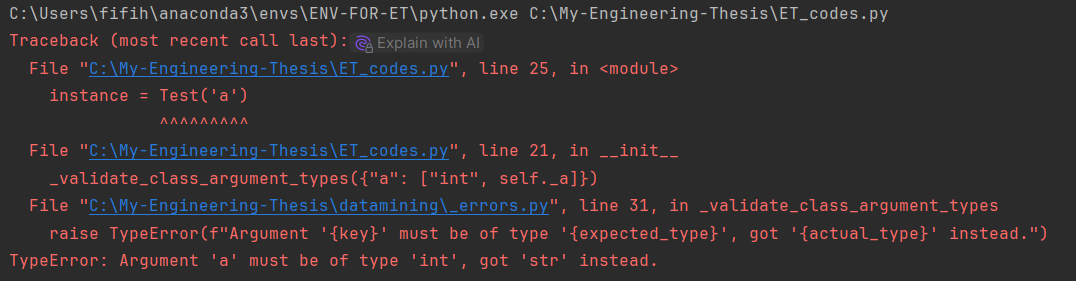
* Utworzenie instancji klasy *Test* z argumentem *a* podanym do konstruktora klasy o typie integer (zgodnym z definicją konstruktora) powoduje stworzenie się instancji (metoda nie podnosi błędu):

*  
Rys. 4.7 – Utworzenie instacji klasy Test z poprawnym typem argumentu a konstruktora*

  
*Rys. 4.8 – 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. 4.9 – Utworzenie instancji klasy Test z niepoprawnym typem argumentu a konstruktora*

 *Rys. 4.10 – 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:

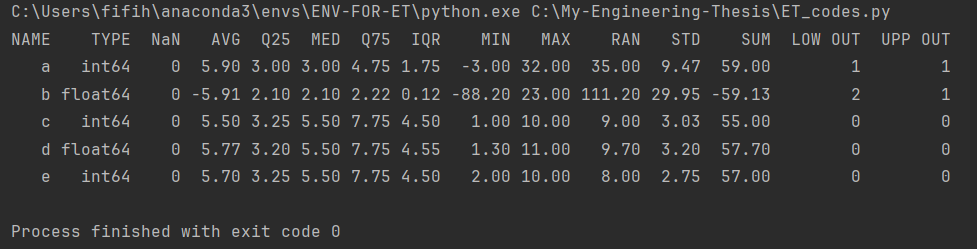
* zmienne numeryczne, np. 1, 1.12, 1+2i, -4,
* zmienne kategoryczne, np. ‘a’, ‘man’, True,
* zmienne w postaci szeregów czasowych, np. 7 maj 2024 01:02:03.45,
* zmienne w postaci interwałów czasowych, np. 2 lata 3 dni 4 godziny, 3 minuty 3 sekundy.

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. 4.11. Wynik działania tej metody zaprezentowano na Rys. 4.12

  
*Rys. 4.11 – Przykładowe wywołanie metody check\_numeric\_data*

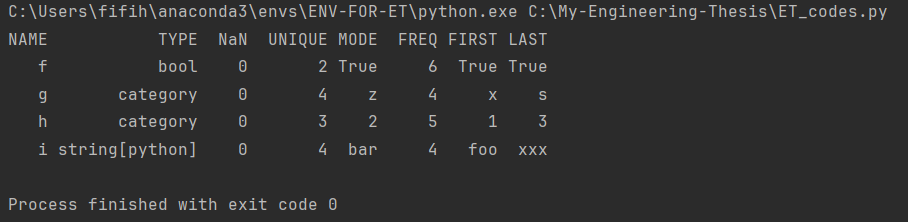
*Rys. 4.12 – Przykładowy wynik działania metody check\_numeric\_data*

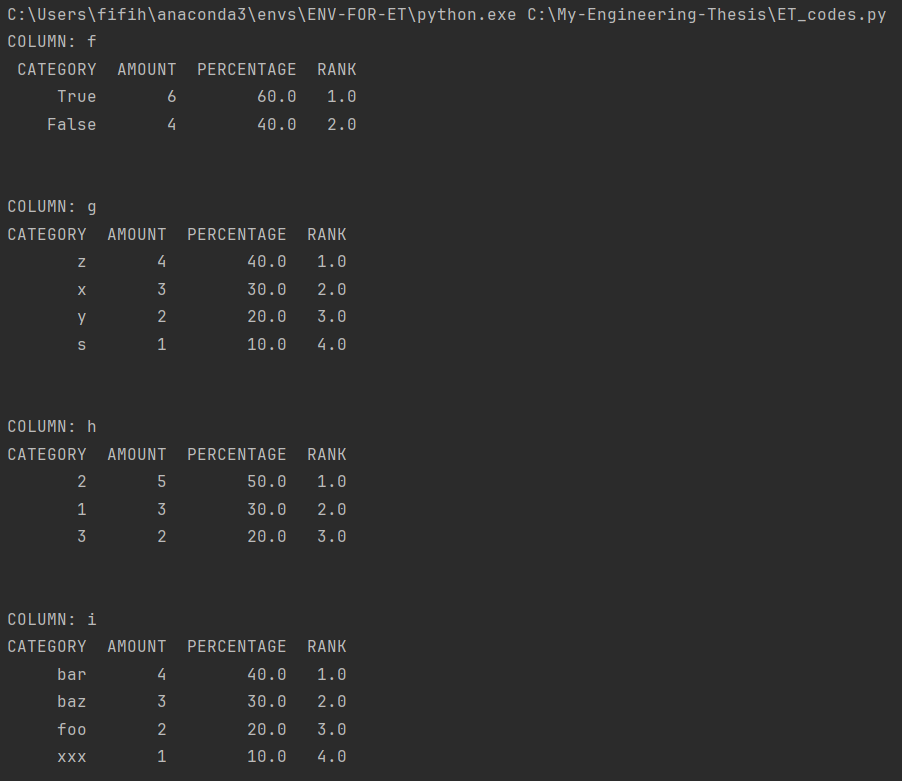
Jak przedstawiono na Rys. 4.12, użytkownik dzięki uruchomieniu tej metody jest   
w stanie dowiedzieć się o NAME, TYPE, NaN,...

* 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 użytkownik 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. 4.13. Wyniki jej działania zaprezentowano na Rys. 4.14   
i Rys. 4.15. Do pokazania rezultatów użyto przykładowej ramki danych o nazwie *ramka\_danych* przechowującej 4 kolumny kategoryczne (*f*, *g*, *h*, *i*):

*Rys. 4.13 – Przykład wywołania metody check\_category\_data*

*Rys. 4.14 – Przykładowy wynik wywołania metody check\_category\_data z argumentem cat\_dist utawionym na wartość False*

*Rys. 4.15 – Przykładowy wynik wywołania metody check\_category\_data z argumentem cat\_dist ustawionym na wartość True*

W przypadku przedstawionym na Rys. 4.14 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:

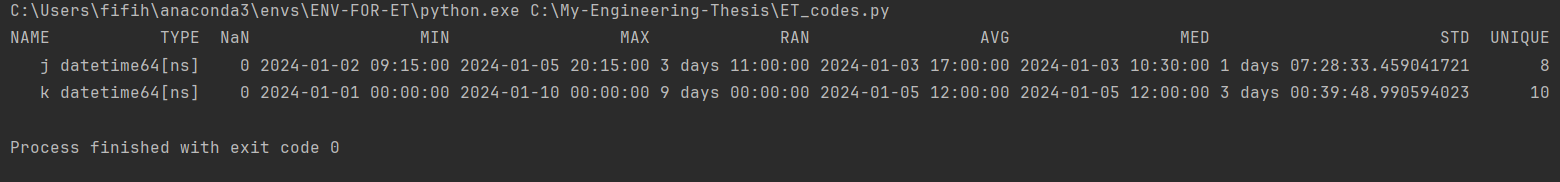
* CATEGORY – nazwa kategorii,
* AMOUNT – ilość wystąpień,
* PERCENTAGE – procentowy udział kategorii w zbiorze danych,
* RANK – pozycja kategorii (1-występuje najczęściej), podsumowanie każdej kolumny posortowano względem tego pola.
  + 1. Metoda *check\_time\_series\_data*

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. 4.16. Ponadto na Rys. 4.17 zaprezentowano output po wywołaniu tej metody.

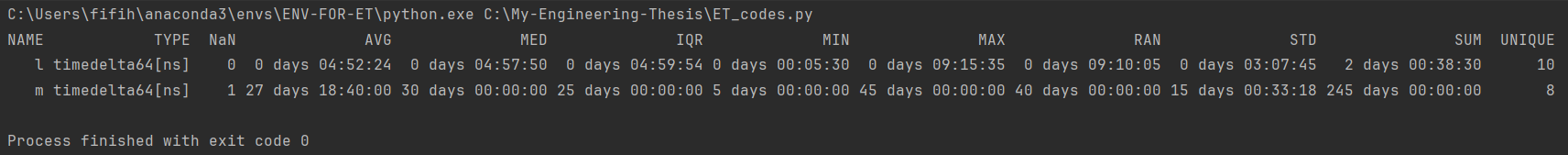
*  
Rys. 4.16 – Przykład wywołania metody check\_time\_series\_data*

*  
Rys. 4.17 - 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. 4.18 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. 4.18 - Przykład wywołania metody check\_time\_interval\_data*

*  
Rys. 4.19 – Przykładowy wynik wywołania metody check\_time\_interval\_data*

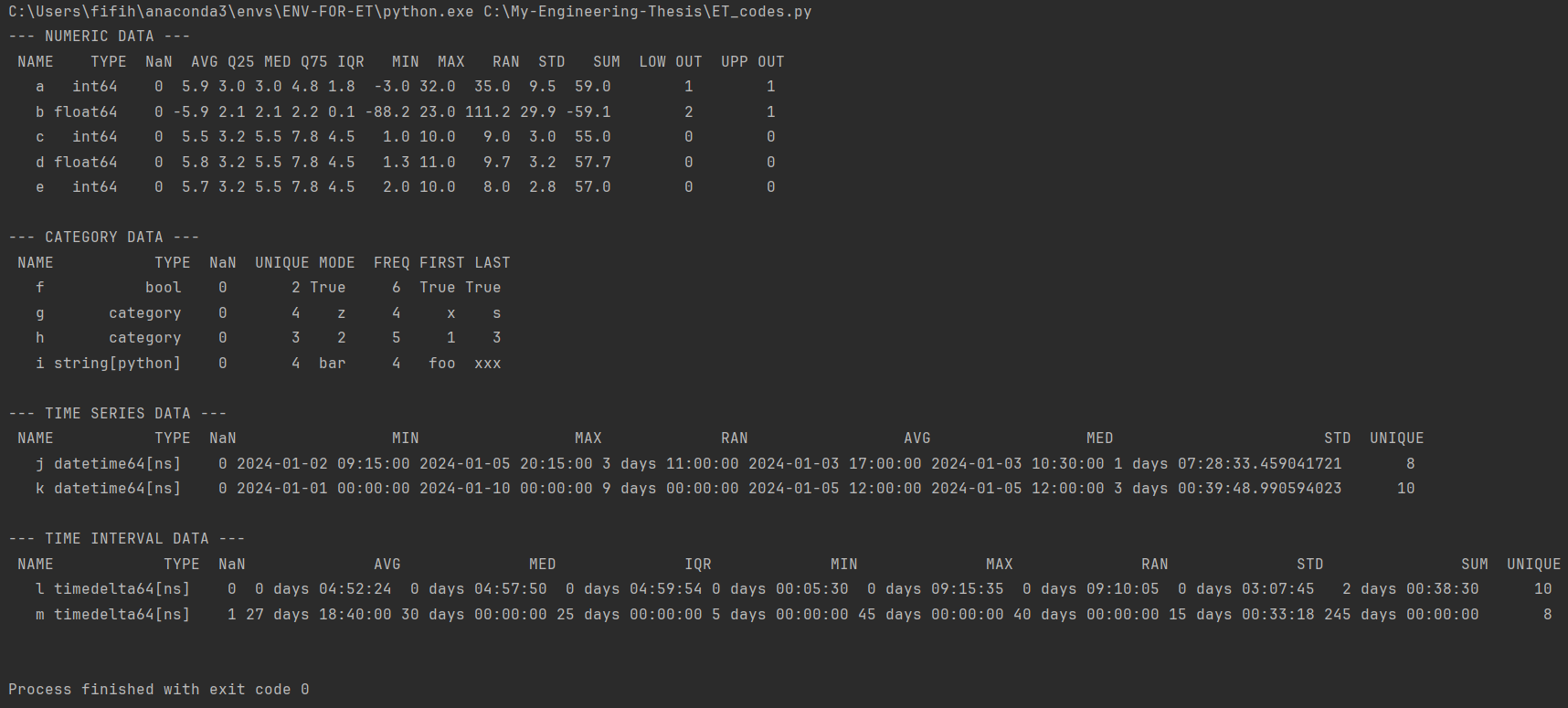
Przykładowy wynik zapytania z Rys. 4.18 przedstawiono na Rys. 4.19. Wyświetlone na nim statystyki to:

* 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. 4.20 zaprezentowano sposób wywoływania tej metody.

*  
Rys. 4.20 - Przykład wywołania metody check\_data*

W przypadku nieistnienia jednego z czterech rodzajów typów zmiennych metoda zwróci odpowiednią informację. Na Rys. 4.21 zaprezentowano wynik działania tej metody na testowej ramce danych *ramka\_testowa*.

  
Rys. 4.21 - *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 silnie lub niepoprawnie oddziaływać na model. Wybrane metody transformacji danych to standaryzacja, normalizacja min-max, transformacja logarytmiczna, normalizacja boxa-coxa, 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.

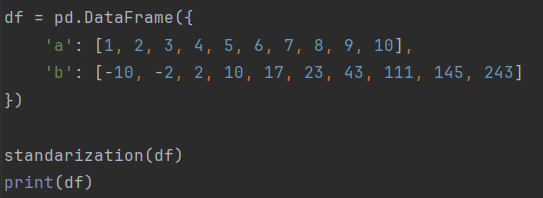
* + 1. Metoda *standarization*

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 [10]:

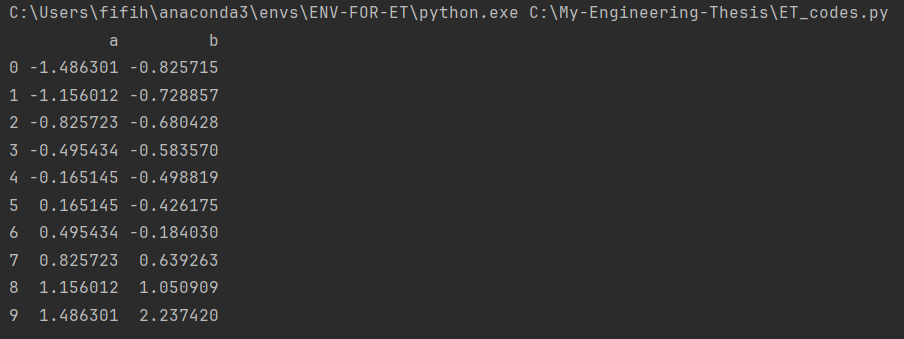
*yk = (xk – xśr) / xstd*

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

xstd – odchylenie standardowe całej populacji zmiennej x

*  
Rys. 4.22 - Przykład wywołania metody standarization*

Na rysunku powyżej Rys. 4.22 zaprezentowano wywołanie tej metody na przykładowej 10-cio elementowej ramce danych. Poniżej na Rys. 4.23 przedstawiono wynik działania tej metody. Dane nie są postaci takiej, jak początkowo przy tworzeniu ramki, a zostały ustandaryzowane.

*Rys. 4.23 - Przykładowy wynik wywołania metody standarization*

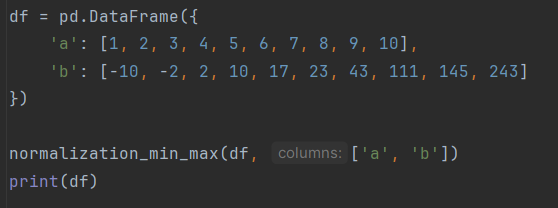
* + 1. Metoda *normalization\_min\_max*

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 [11]:

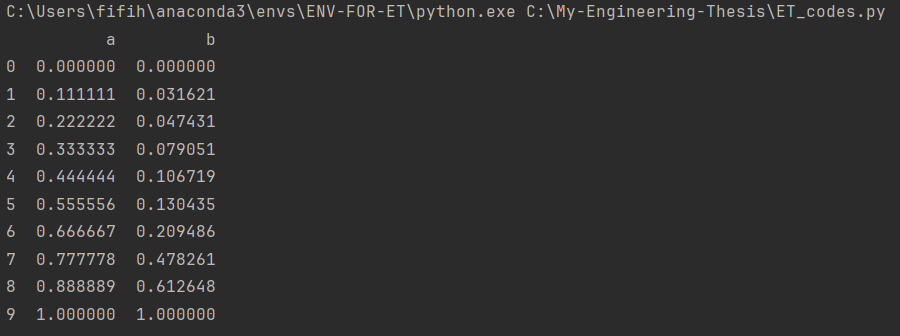
*yk = (xk – xmin) / (xmax – xmin)*

gdzie:  
yk – wartość znormalizowana,  
xk – wartość zaobserwowana,  
xmin – wartość minimalna występująca w populacji,xmax – 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. 4.24 pokazano przykładowe wywołanie metody *normalization\_min\_max* na przykładowej ramce danych.

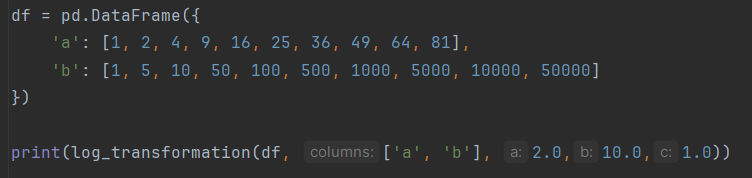
  
*Rys. 4.24 - Przykład wywołania metody normalization\_min\_max*

Wynik operacji z Rys. 4.24 zaprezentowano na Rys. 4.25. Przedstawia on dane znormalizowane za pomocą algorytmu min-max.

 *Rys. 4.25 - 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.

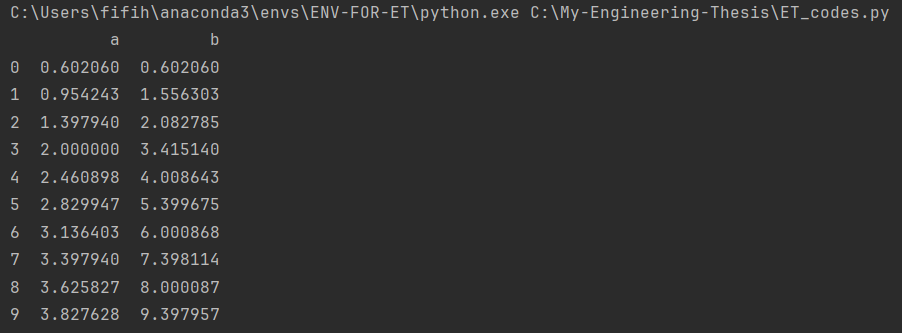
*   
Rys. 4.26 - Przykład wywołania metody log\_transformation*

Na rysunku powyżej Rys. 4.26 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. 4.27 zaprezentowano wynik działania tej funkcji:

* Rys. 4.27 - Przykładowy wynik wywołania metody log\_transformation*

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\_cox*

Kolejna metodą wewnątrz podmodułu „transformations” jest metoda do przekształcania danych używając do tego transformacji Boxa-Coxa. 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 [12]:

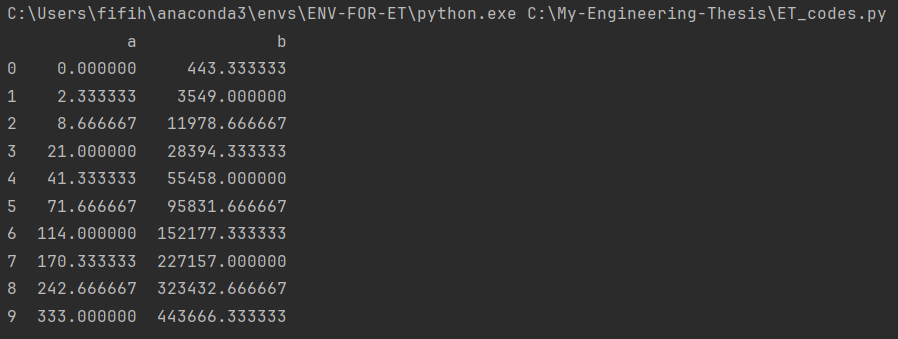
*{yk = (xk*λ*- 1) /* λ*, dla* λ ≠ *0} lub {yk = ln(xk), dla* λ *=0}*

gdzie:  
yk – wartość znormalizowana,  
xk – wartość zaobserwowana,  
λ – parametr alpha.

Zaimplementowana metoda *transformation\_box\_cox* 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. 4.28. Parametr *alpha* ustawiono na wartość 3.

Obraz zawierający tekst, zrzut ekranu, Czcionka

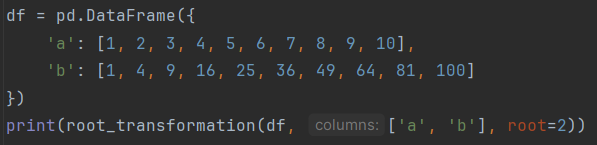
Opis wygenerowany automatycznie  
*Rys. 4.28 - Przykład wywołania metody transformation\_box\_cox*

  
*Rys. 4.29 - Przykładowy wynik wywołania metody transformation\_box\_cox*

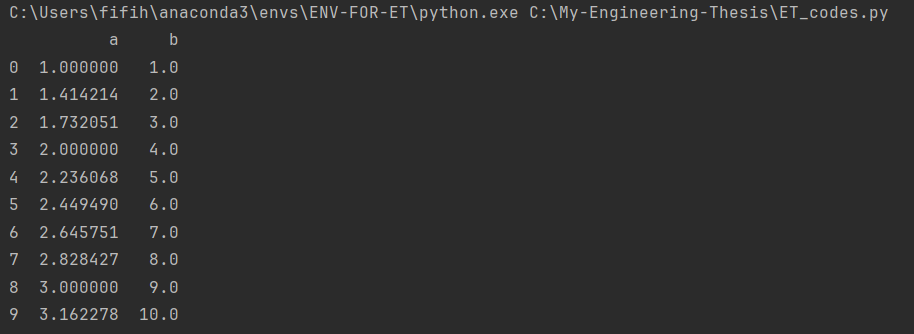
Jak zaprezentowano na Rys. 4.29 dane zostały przekształcone zgodnie ze wzorem podanym powyżej.

* + 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.

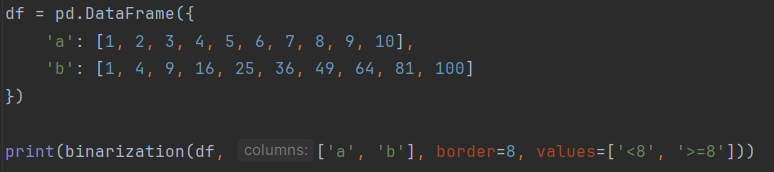
*  
Rys. 4.30 – Przykładowe wywołanie metody root\_transformation*

Powyżej na Rys. 4.30 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. 4.31 zaprezentowano rezultat wywołania metody *root\_transformation*.

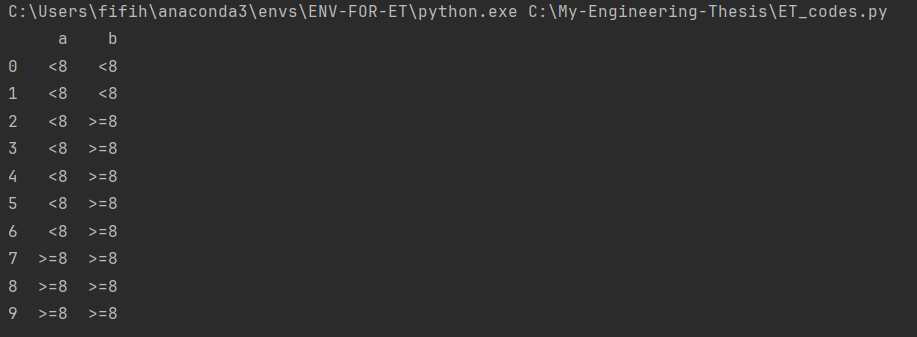
  
*Rys. 4.31 - Przykładowy wynik wywołania metody root\_transformation*

* + 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. 4.32.

  
*Rys. 4.32 – Przykładowe wywołanie metody binarization*

Przykładowo ustawiono wartość graniczną równą 8, zastępując wartości krótką informacją; czy wartość jest mniejsza od 8 czy nie. Na Rys. 4.33 zaprezentowano wynik działania tej metody.

  
*Rys. 4.33 - 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.

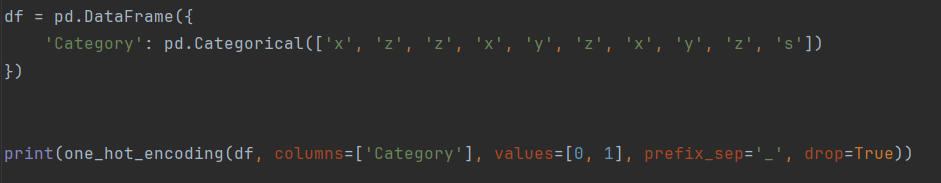
* + 1. Metoda *one\_hot\_encoding*

Ostatnią metodą wchodzącą w skład podmodułu „transformations” jest *one\_hot\_encoding* umożliwiająca przeprowadzanie procesu kodowania one-hot. Jest to bardzo istotna transformacja pozwalająca na zamianę zmiennych kategorycznych na zmienne numeryczne. Polega na utworzeniu n-nowych kolumn w ramce danych. Liczba n określa ilość kategorii wewnątrz kolumny z danymi kategorycznymi. Każda nowo utworzona kolumna odpowiada jednej kategorii i składa się zaledwie z dwóch wartości; jednej stwierdzającej, że rekord należy do tej kategorii oraz drugiej określającej, że rekord należy do innej kategorii. Najczęstszymi wartościami określającymi przynależność rekordu do kategorii są 0 (nie należy) oraz 1 (należy).

Metoda *one\_hot\_encoding*, poza standardowymi argumentami *df* i *columns* przyjmuje trzy dodatkowe argumenty.

Pierwszym z nich jest argument *values*, będący dwuelementową listą. Element o indeksie 0 wewnątrz listy określa wartość numeryczną, jaką przyjmuje rekord nie należący do danej kategorii, natomiast element element o indeksie 1 określa przynależność do danej kategorii. Domyślnie lista ta wygląda następująco: *[0, 1]*.

Drugi z parametrów, które użytkownik samodzielnie może ustawić to *prefix\_sep*. Przyjmuje on wartości o typie tekstowym. Pozwala na dodanie do nowo utworzonych kolumn z flagą przynależności do kategorii separator ułatwiający nazewnictwo nowo utworzonych kolumn. Poniżej na Rys. 4.34 zaprezentowano wywołanie funkcji dla separatora „\_”, natomiast na Rys. 4.35 wyświetlono wynik tej operacji. Przekazując odpowiedni separator, trafia on do nazwy nowo utworzonej kolumny pomiędzy nazwę kolumny kategorycznej i nazwę kategorii.

  
*Rys. 4.34 – Przykładowe wywołanie metody one\_hot\_encoding*

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Rys. 4.35 – Przykładowy wynik wywołania metody binarization

Ostatni parametr o nazwie *drop*, określa czy użytkownik chce usunąć starą kolumną kategoryczną. Przyjmuje on wartości *True* albo *False* (*True* – usuwa kolumnę, *False* – zostawia kolumnę).

Kodowanie one-hot jest bardzo przydatnym narzędziem przy tworzeniu modeli uczenia maszynowego. Pozwala nie tylko pozbyć się negatywnego wpływu danych kategorycznych na model, ale także poprawić jego jakość. Przykładem, w którym warto stosować ten rodzaj transformacji może być chociażby ramka danych w skład której wchodzi kolumna z wartościami 1-6 określająca 6 różnych zjawisk, przy czym zjawiska te nie są w żaden sposób uporządkowane. Stworzenie binarnych kolumn dla każdej z kategorii nie pozwala modelowi nauczyć się opierając się na porządku tych kategorii.

## PODMODUŁ „preprocessing”

Preprocessing jest jednym z pierwszych procesów wchodzących w skład szeroko pojętej analizy danych. Jak podaje portal GeekForGeeks jest to technika używana do transformacji danych tabelarycznych w użyteczny i wydajny format, w której skład wchodzi czyszczenie danych, transformacja danych i redukcja danych. Proces ten w zależności od problemu i danych badawczych może się różnić [13].

Wewnątrz podmodułu „preprocessing” zaimplementowano dwie funkcje optymalizujące radzenie sobie z danymi brakującymi. Pierwsza z nich wspomaga analityka w kontekście danych numerycznych, druga zaś w kontekście danych kategorycznych.

Obie funkcje przyjmują dokładnie te same argumenty:

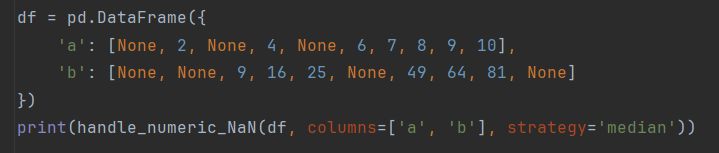
* *df* – badana ramka danych z pakietu *pandas,*
* *columns* – lista stringów z nazwami kolumn,
* *strategy* – strategia, którą użytkownik może użyć do poradzenia sobie   
  z wartościami brakującymi.
  + 1. Metoda *handle\_numeric\_NaN*

Metoda *handle\_numeric\_NaN* wspomaga radzenie sobie z brakującymi danymi numerycznymi. Użytkownik za pomocą parametru *strategy* określa scenariusz, według którego brakujące wartości zostaną edytowane. Zaimplementowano   
6 różnych scenariuszy:

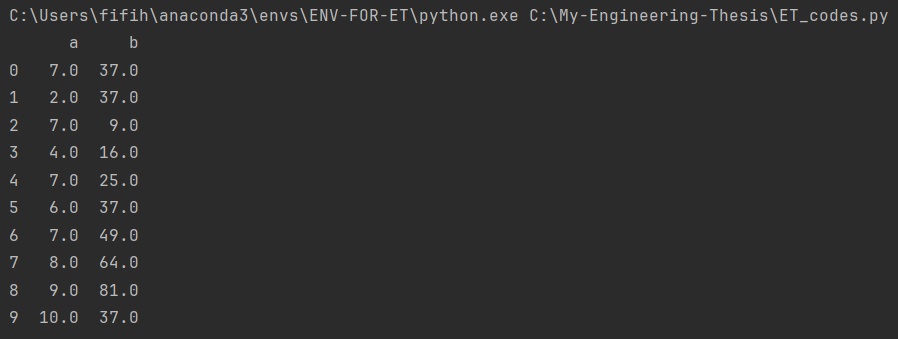
* Przekazanie parametru w postaci wartości numerycznej – w tym przypadku wartości brakujące zostaną zastąpione przekazaną wartością,
* *‘mean’* – w tym przypadku wartości brakujące zostaną zastąpione wartością średnią z reszty rekordów,
* *‘median’* – w tym przypadku wartości brakujące zostaną zastąpione medianą z reszty rekordów,
* *‘min’* – w tym przypadku wartości brakujące zostaną zastąpione najniższą wartością z reszty rekordów,
* *‘max’* – w tym przypadku wartości brakujące zostaną zastąpione najwyższą wartością z reszty rekordów,
* *‘drop’ –* w tym przypadku metoda usunie wiersze z brakującymi wartościami.

W przypadku podania innej wartości parametru niż podane powyżej, metoda podniesie błąd.

Poniżej na Rys. 4.36 zaprezentowano przykład wywołania tej metody, dla parametru *strategy* ustawionym na ‘*median’*.

  
*Rys. 4.36 – Przykładowe wywołanie metody handle\_numeric\_NaN*

Na Rys. 4.37 przedstawiającym wynik powyższego wywołania metody warto skupić się na pierwszym rekordzie, który w przypadku obu pól *a* i *b* ma wartość *None* (wartość brakującą). Zgodnie z przyjętą strategią przyjął on medianę z kolumny *a* (w przypadku kolumny *a*) oraz medianę z kolumny *b* (w przypadku kolumny *b*).

  
*Rys. 4.37 - Przykładowy wynik wywołania metody handle\_numeric\_NaN*

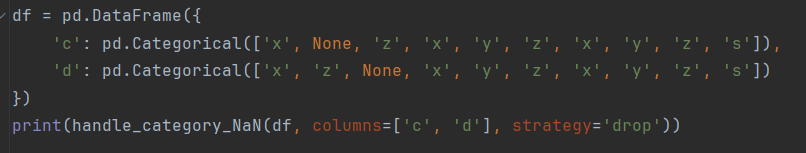
* + 1. Metoda *handle\_category\_NaN*

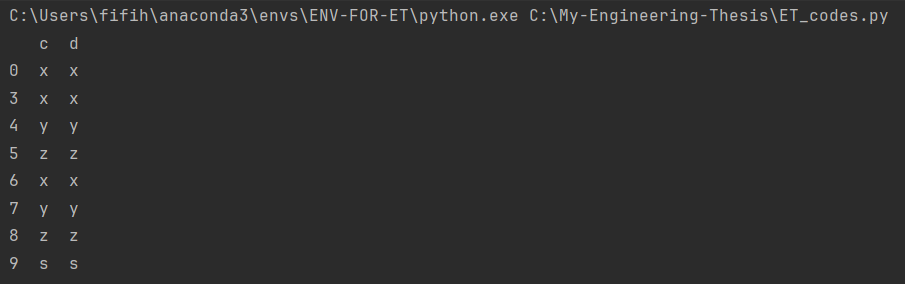
Metoda *handle\_numeric\_NaN* wspomaga radzenie sobie z brakującymi danymi kategorycznymi. Podobnie jak w przypadku *handle\_numeric\_NaN*, użytkownik za pomocą parametru *strategy* określa scenariusz, według którego brakujące wartości zostaną edytowane. Zaimplementowano 5 różnych scenariuszy:

* Przekazanie parametru w postaci wartości tekstowej innej niż poniższe – w tym przypadku kategorie brakujące zostaną zastąpione przekazaną,
* *‘mode’* – w tym przypadku kategorie brakujące zostaną zastąpione najczęściej występującą kategorią,
* *‘next’* – w tym przypadku kategorie brakujące zostaną zastąpione kategorią, do której należy następny rekord (z indeksem o 1 wyższym),
* *‘prev’* – w tym przypadku kategorie brakujące zostaną zastąpione kategorią, do której należy poprzedni rekord (z indeksem o 1 niższym),
* *‘drop’ –* w tym przypadku metoda usunie wiersze z brakującymi kategoriami.

W przypadku podania innej wartości parametru niż podane powyżej, metoda podniesie błąd.

Poniżej, na Rys. 4.38 zaprezentowano w jaki sposób użytkownik może funkcję *handle\_category\_NaN* wywołać, a na Rys. 4.39 jak wygląda przykładowy output tej metody. W przykładzie wykorzystano opcję *‘drop’,* usuwającą rekordy   
z brakującymi kategoriami.

  
*Rys. 4.38 – Przykładowe wywołanie metody handle\_category\_NaN*

  
*Rys. 4.39 - Przykładowy wynik wywołania metody handle\_category\_NaN*

Metoda *handle\_category\_NaN* usunęła drugi i trzeci rekord z ramki danych *df*,   
z powodu braku wartości w kolumnach *c* i *d* (rekordy usunięte to te o indeksach 1   
i 2).

## PODMODUŁ „regression”

Wewnątrz podmodułu nazwanego „regression” zaimplementowano dwie klasy obiektów ułatwiających i usprawniających tworzenie modelów regresji liniowej.

Regresja ogólnie jest narzędziem statystycznym pozwalającym na określenie, bądź przewidywanie ciągłej wartości liczbowej jakiejś cechy na podstawie innej cechy bądź innych cech.

Według portalu naukowiec.org regresja liniowa jest najprostszym wariantem regresji, który zakłada, że zmienność zmiennej objaśnianej względem zmiennej objaśniającej (bądź grupy zmiennych objaśniających) jest określona funkcją liniową [14].

Do utworzenia modelu regresji liniowej dane powinny zostać podzielone na dwie części (nie jest to konieczne, aby obie części były sobie równe, nie jest to również zalecane): zbiór danych treningowy i zbiór danych testowy. Pierwszy z nich służyć ma do nauczenia modelu (czyli jego utworzenia), drugi do ocenienia (sprawdzenia jak dobrze model przewiduje zmienną zależną). Opisany podział danych na dwa mniejsze zbiory danych; treningowy i testowy jest najpopularniejszą metodą służącą do nauki modelu. Drugą, nieco bardziej zaawansowaną jest metoda krosswalidacji. Polega ona na podziale danych na k-części. Każda część powinna być takiej samej wielkości jak reszta (posiadać tyle samo rekordów). Następnie po przeprowadzeniu podziału danych następuje k-krotne uczenie modelu za pomocą   
k-1 części danych oraz jego ocena/przetestowanie za pomocą 1, pozostałej części danych. Formułę tą powtarza się k-krotnie zmieniając za każdym razem część danych, która jest odpowiedzialna za ocenę modelu. Na końcu wyniki z k-krotnego uczenia i testowania modelu uśrednia się i otrzymuje się jeden model.

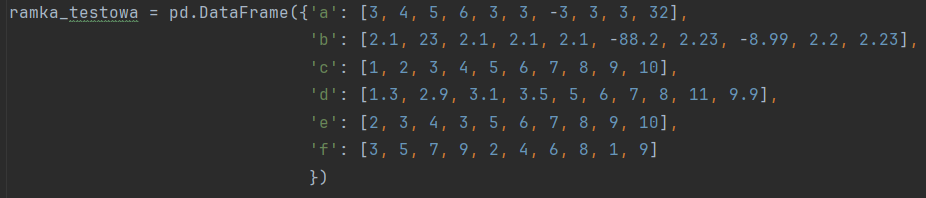
* + 1. Klasa obiektów *BestSimpleLinearRegression*

Jak wskazuje nazwa klasy *BestSimpleLinearRegression* jest to klasa tworząca modele prostej regresji liniowej. Utworzenie instancji tej klasy wiąże się   
z szeregiem operacji w tle, w taki sposób, aby model wyjściowy był jak najlepszy (posiadał jak najwyższą wartość współczynnika R-squared). Aby osiągnąć ten cel, wewnątrz klasy tworzy się określona liczba modelów prostej regresji liniowej (liczba ta jest równa liczbie cech, kolumn wewnątrz badanej ramki danych nie wliczając cechy będącej zmienną zależną, badaną), za każdym razem porównując czy utworzony model nie jest lepszy od modelu, który w danym momencie jest uznany za najlepszy.

Konstruktor tej klasy przyjmuje dokładnie 6 argumentów:

* df (pandas Data Frame) – wskazuje na badaną ramkę danych,
* response (string) – wskazuje na nazwę zmiennej odpowiedzi/zależnej,
* set\_seed (int) – ziarno (kilkukrotne stworzenie instancji klasy   
  z niezmiennym parametrem spowoduje utworzenie dokładnie tych samych instancji, zmiana parametru poskutkuje nieco zmienionym procesem uczenia i oceniania modelu, a więc również innym modelem w postaci instancji klasy),
* divide\_method (string) – przyjmuje wartości crossvalidation lub train\_test, określa, którą metodę ma przyjąć klasa do podziału danych i uczenia modelu (krosswalidacja, bądź prosty podział na zbiory treningowy i testowy),
* k (int) – ilość równych sobie przedziałów (niezbędne do uczenia modelu metodą krosswalidacji). Domyślnie przyjmuje wartość 5,
* test\_size (float) – zakładając, że ilość wszystkich rekordów w ramce danych jest równa 1, określa jaka część danych powinna służyć do testowania modelu, podczas procesu uczenia metodą podziału na zbiory treningowy   
  i testowy.

Poniżej na Rys. 4.40 zaprezentowano przykładową, 10-cio elementową ramkę danych *ramka\_testowa* na podstawie której zaprezentowane zostaną przykłady działania klasy *BestSimpleLinearRegression*.

  
*Rys. 4.40 – ramka danych ramka\_testowa, na podstawie której zaprezentowany zostanie przykład działania klasy BestSimpleLinearRegression*

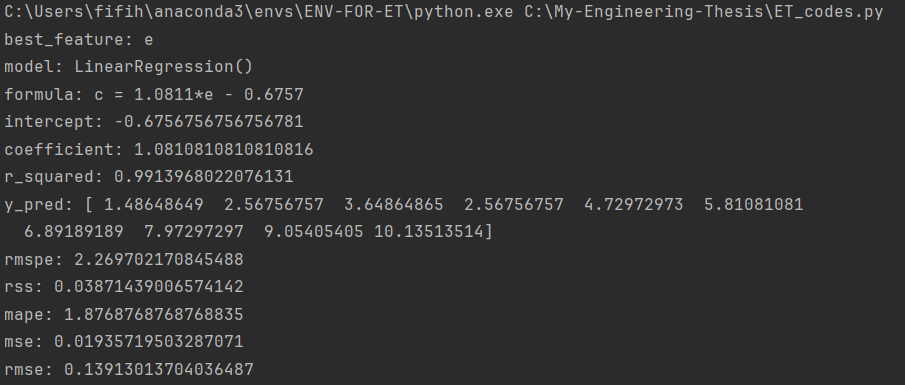
W celu zaprezentowania działania klasy *BestSimpleLinearRegression* utworzono instancję tej klasy o nazwie *instance* wskazując na kolumnę *c*, jako tą, która przechowuje wartości zmiennej zależnej. Ziarno ustawiono na 12 oraz posłużono się metodą podziału danych na zbiór uczący i testowy (wielkość zbioru testowego ustawiono na 20% całego zbioru danych). Następnie wypisano do terminala wszystkie możliwe wartości parametrów. Poniżej na Rys. 4.41 pokazano utworzenie instancji oraz sposób wypisania wszystkich parametrów:

*  
Rys. 4.41 – przykład utworzenia instancji klasy BestSimpleLinearRegression oraz wypisania wszystkich parametrów tej instancji*

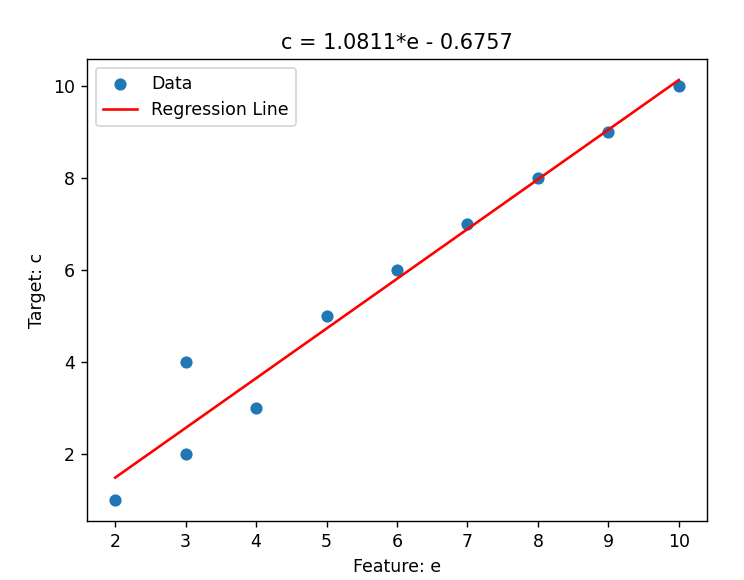
Wywołanie tychże parametrów instancji, ma na celu bliższe zapoznanie się   
z utworzonym modelem przez użytkownika. Przedstawione parametry określają:

* *best\_feature –* nazwę zmiennej niezależnej, która najlepiej koreluje się ze zmienną zależną,
* *model* – obiekt klasy LinearRegression z pakietu scikit-learn,
* *formula* – wzór funkcji regresji (funkcja liniowa),
* *intercept* – wyraz wolny z prostej regresji (przecięcie z osią X),
* *coefficient* – współczynnik kierunkowy prostej regresji,
* *r\_squared* – wartość parametru r-squared,
* *y\_pred* – wartości przewidziane przez model,
* *rmspe* – Root Mean Squared Percentage Error,
* *rss* – Residual Sum of Squares,
* *mape* – Mean Absolute Percentage Error,
* *mse* – Mean Squared Error,
* *rmse* – Root Mean Squared Error.

Poniżej na Rys. 4.42 przedstawiono wynik kompilacji przedstawionego na Rys. 4.41 tworzenia instancji i wypisywania parametrów:

  
*Rys. 4.42 – wynik stworzenia instancji klasy BestSimpleLinearRegression na podstawie ramki danych ramka\_testowa i wypisania wartości parametrów*

Ponadto do klasy *BestSimpleLinearRegression* dodano również metodę pozwalającą na wyświetlenie modelu na prostym wykresie (przy pomocy funkcji z pakietu *matplotlib*). Metodę tę nazwano *plot\_model*. Poniżej na Rys. 4.43 przedstawiono wykres liniowy modelu modelu. Tytuł wykresu przedstawia wzór funkcji liniowej (współczynnik kierunkowy i wyraz wolny zaokrąglono do 4 miejsc po przecinku). Model regresji liniowej zaprezentowano kolorem czerwonym, natomiast dane kolorem niebieskim. Oś X wskazuje na zmienną niezależną, która najlepiej opisuje zmienną zależną, przedstawioną na osi Y. Do wykresu dodano również legendę.

  
*Rys. 4.43 – Model prostej regresji liniowej stworzony za pomocą klasy BestSimpleLinearRegression przedstawiony na wykresie za pomocą metody plot\_model, wchodzącej w skład tejże klasy*

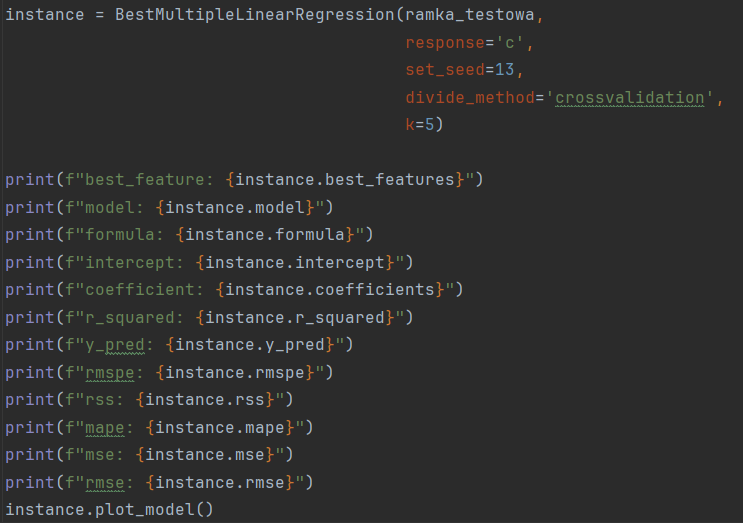
* + 1. Klasa obiektów *BestMultipleLinearRegression*

Poza powyżej opisaną klasą, w skład podmodułu „regression” wchodzi klasa obiektów *BestMultipleLinearRegression* tworząca modele wielorakiej regresji liniowej. Zasada działania tej klasy jest analogiczna do *BestSimpleLinearRegression* z tą różnicą, że zamiast tworzenia modelów regresji liniowej dla jednej zmiennej niezależnej tworzą się modele opisujące zmienną zależną co najmniej dwoma zmiennymi niezależnymi. Algorytm działa w sposób następujący; wylicza się wartość współczynnika r-squared dla każdej kombinacji dwóch zmiennych niezależnych, następnie dla każdej kombinacji trzech zmiennych niezależnych i tak dalej, aż do momentu, kiedy utworzy się model dla kombinacji wszystkich zmiennych niezależnych. Kombinacja, która osiągnie najlepszy wynik (r-squared) będzie ostatecznie zwrócona jako obiekt klasy.

Konstruktor klasy *BestMultipleLinearRegression* przyjmuje dokładnie takie same argumenty jak w przypadku konstruktora klasy *BestSimpleLinearRegression*:

* df (pandas Data Frame) – wskazuje na badaną ramkę danych,
* response (string) – wskazuje na nazwę zmiennej odpowiedzi/zależnej,
* set\_seed (int) – ziarno (kilkukrotne stworzenie instancji klasy   
  z niezmiennym parametrem spowoduje utworzenie dokładnie tych samych instancji, zmiana parametru poskutkuje nieco zmienionym procesem uczenia i oceniania modelu, a więc również innym modelem w postaci instancji klasy),
* divide\_method (string) – przyjmuje wartości crossvalidation lub train\_test, określa, którą metodę ma przyjąć klasa do podziału danych i uczenia modelu (krosswalidacja, bądź prosty podział na zbiory treningowy i testowy),
* k (int) – ilość równych sobie przedziałów (niezbędne do uczenia modelu metodą krosswalidacji). Domyślnie przyjmuje wartość 5,
* test\_size (float) – zakładając, że ilość wszystkich rekordów w ramce danych jest równa 1, określa jaka część danych powinna służyć do testowania modelu, podczas procesu uczenia metodą podziału na zbiory treningowy   
  i testowy.

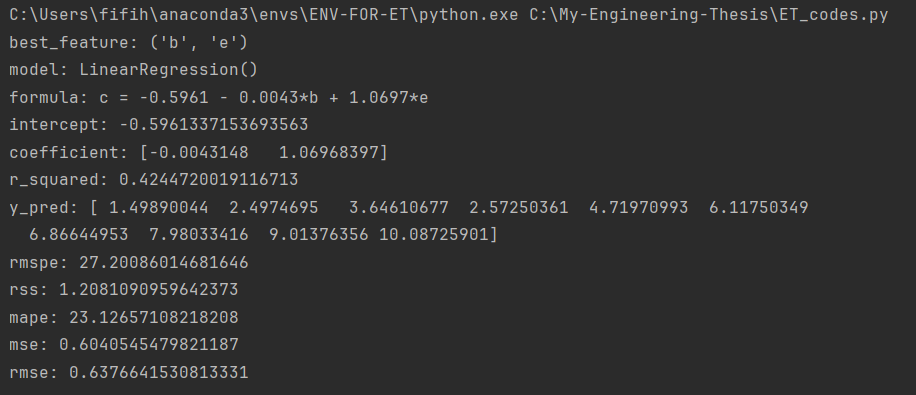
W celu zaprezentowania działania tej klasy posłużono się ponownie ramką danych   
o nazwie *ramka\_danych*, której definicja znajduje się na Rys. 4.40. Na początku utworzono instancję przekazując do konstruktora kolumnę *c* jako tą wskazującą na zmienną zależną. Tym razem jako metodę uczenia modelu wybrano krosswalidację   
z argumentem *k* ustawionym na wartość 5. Zmieniono również ziarno. Poniżej na   
Rys. 4.44 zaprezentowano tworzenie instancji klasy, wypisywanie parametrów tejże instancji i wykonywanie metody tworzącej wykres modelu.

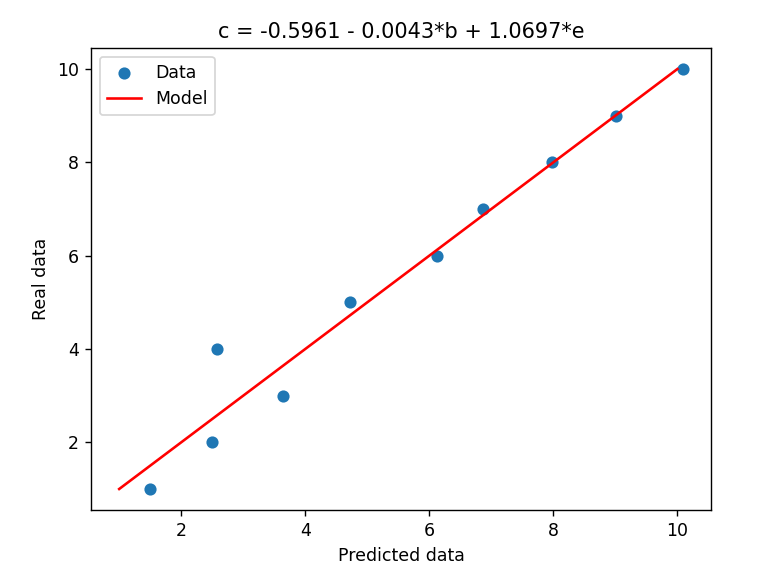
  
*Rys. 4.44 – Przykład utworzenia instancji klasy BestMultipleLinearRegression, wypisania wszystkich parametrów tej instancji oraz wywołania metody tworzącej wykres modelu regresji*

Parametry utworzonej instancji, które użytkownik jest w stanie wyświetlić to:

* *best\_features –* nazwy zmiennych niezależnych, które najlepiej opisują zmienną zależną,
* *model* – obiekt klasy LinearRegression z pakietu scikit-learn,
* *formula* – wzór funkcji regresji (funkcja liniowa),
* *intercept* – wyraz wolny z prostej regresji (przecięcie z osią X),
* *coefficients* – współczynniki kierunkowe prostej regresji (każdy kolejny ma to samo położenie w liście, co nazwa zmiennej do której się odnosi w liście *best\_features*),
* *r\_squared* – wartość parametru r-squared,
* *y\_pred* – wartości przewidziane przez model,
* *rmspe* – Root Mean Squared Percentage Error,
* *rss* – Residual Sum of Squares,
* *mape* – Mean Absolute Percentage Error,
* *mse* – Mean Squared Error,
* *rmse* – Root Mean Squared Error.

Poniżej przedstawiono wynik wyświetlenia parametrów instancji na Rys. 4.45 oraz wynik działania metody tworzącej wykres na Rys. 4.46:

 *Rys. 4.45 – wynik stworzenia instancji klasy BestMultipleLinearRegression na podstawie ramki danych ramka\_testowa i wypisania wartości parametrów*

  
Rys. 4.46 – Model prostej regresji liniowej stworzony za pomocą klasy *BestMultipleLinearRegression przedstawiony na wykresie za pomocą metody plot\_model, wchodzącej w skład tejże klasy*

W tytule wykresu dodano wzór funkcji regresji. Ponownie czerwonym kolorem zaprezentowano tę linię za pomocą obliczonego wzoru, a kolorem niebieskim dane. Dane zostały zaprezentowane nieco inaczej niż w przypadku poprzedniej klasy. Oś X wskazuje na wartości przewidziane przez model, natomiast oś Y reprezentuje dane rzeczywiste.

## PODMODUŁ „classification”

Klasyfikacja jest jednym z głównych typów uczenia maszynowego nadzorowanego. Polega na przypisaniu obiektów do odpowiednich, wcześniej zdefiniowanych klas na podstawie określonych cech. Podobnie jak regresja, klasyfikacja jest rodzajem uczenia maszynowego nadzorowanego. Uczenie modelu klasyfikacyjnego odbywa się na podstawie wcześniej określonych cech obiektu i przypisanej do niego klasy [15].

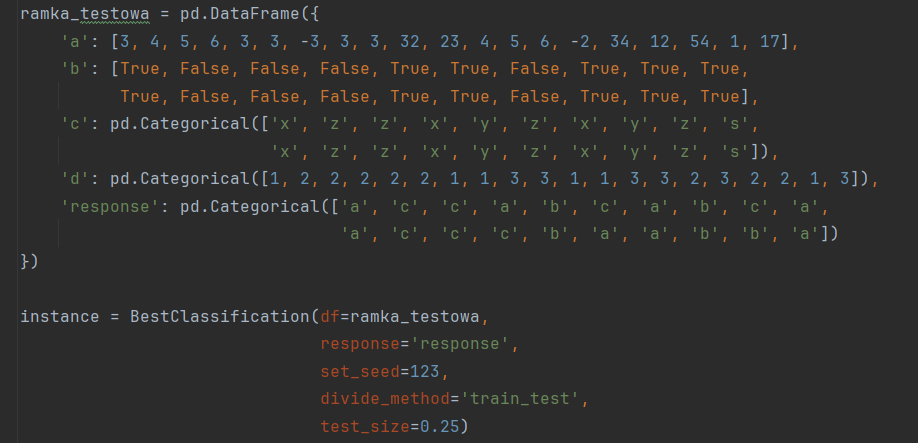
Istnieje wiele różnych algorytmów klasyfikujących. W tej pracy skupiono się na klasyfikatorach: regresja logistyczna, losowy las decyzyjny (random forest) i SVC (Support Vector Classifier).

Wewnątrz tego podmodułu zaimplementowano jedną klasę obiektów o nazwie *BestClassification*, której działanie opiera się na wybraniu takiego klasyfikatora, który najlepiej opisuje daną zmienną zależną na podstawie zmiennych niezależnych. Klasa ta bada trzy wyżej wymienione klasyfikatory, porównując je między sobą miarą dokładności (accurancy). Jest to miara określająca współczynnik poprawnie sklasyfikowanych obiektów (ilość poprawnie sklasyfikowanych obiektów podzielona przez całkowitą ilość obiektów).

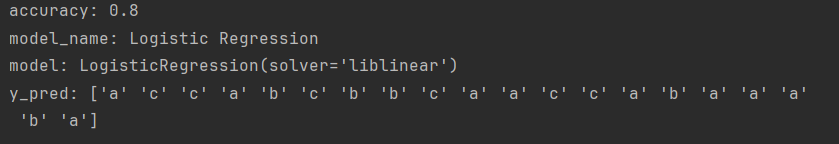
Konstruktor klasy *BestClassification* przyjmuje dokładnie takie same argumenty, co konstruktory klas w podmodule „regression”:

* df (pandas Data Frame) – wskazuje na badaną ramkę danych,
* response (string) – wskazuje na nazwę pola przechowującego klasy obiektów,
* set\_seed (int) – ziarno (kilkukrotne stworzenie instancji klasy   
  z niezmiennym parametrem spowoduje utworzenie dokładnie tych samych instancji, zmiana parametru poskutkuje nieco zmienionym procesem uczenia i oceniania modelu, a więc również innym modelem w postaci instancji klasy),
* divide\_method (string) – przyjmuje wartości crossvalidation lub train\_test, określa, którą metodę ma przyjąć klasa do podziału danych i uczenia modelu (krosswalidacja, bądź prosty podział na zbiory treningowy i testowy),
* k (int) – ilość równych sobie przedziałów (niezbędne do uczenia modelu metodą krosswalidacji). Domyślnie przyjmuje wartość 5,
* test\_size (float) – zakładając, że ilość wszystkich rekordów w ramce danych jest równa 1, określa jaka część danych powinna służyć do testowania modelu, podczas procesu uczenia metodą podziału na zbiory treningowy   
  i testowy.

W celu przedstawienia tworzenia przykładowej instancji klasy *BestClassification* utworzono 20-sto elementową ramkę danych *ramka\_testowa* składającej się z pięciu kolumn, z czego założono, że kolumna o nazwie *response* przechowuje klasy obiektów, a reszta kolumn to cechy opisujące dany obiekt. Poniżej na Rys. 4.47 zaprezentowano deklarację zmiennej *ramka\_testowa* oraz utworzenie instancji klasy dla ziarna równego 123, metody podziału danych na zbiór treningowy (75% rekordów) i testowy (25% rekordów).

  
*Rys. 4.47 – Przykład tworzenia instancji klasy BestClassification przy wykorzystaniu utworzonej ramki danych ramka\_testowa*

Każda utworzona instancja klasy *BestClassification* posiada cztery atrybuty. Poniżej na Rys. 4.48 przedstawiono ich wartości dla instancji utworzonej na Rys. 4.47.

*Rys. 4.48 – wartości atrubytów dla instancji klasy BestClassification utworzonej na podstawie ramki ramka\_testowa na Rys. 4.47*

Atrybuty tej klasy to:

* *accurancy* – dokładność, miara obrazująca ile obiektów zostało poprawnie przypisanych do klasy w porównaniu do całkowitej liczby obiektów,
* *model\_name* – nazwa modelu, który wykazał najwyższą dokładność,
* *model* – obiekt klasy z pakietu scikit-learn. Ten atrybut może posłużyć do dalszego budowania, analizowania, oceniania utworzonego modelu,
* *y\_pred* – wartości klas przewidziane przez model z najwyższą dokładnością.

# PRZYKŁADY UŻYCIA

# PODSUMOWANIE, DYSKUSJA, WNIOSKI

# BIBLIOGRAFIA

[1] „Tiobe Index”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://www.tiobe.com/tiobe-index/

[2] „PyCharm”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://www.jetbrains.com/pycharm/

[3] „Microsoft Studio Code”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://code.visualstudio.com/

[4] „Anaconda”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://docs.anaconda.com/working-with-conda/environments/

[5] „Pandas”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://pandas.pydata.org/docs/

[6] „Pip”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://pip.pypa.io/en/stable/

[7] „Scikit-learn”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://scikit-learn.org/stable/

[8] „MatplotLib”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://matplotlib.org/stable/index.html

[9] „Numpy”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://numpy.org/doc/

[10] „Standaryzacja”. Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://pogotowiestatystyczne.pl/slowniki/standaryzacja/

[11] „Normalizacja minmax”, Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://pqstat.pl/?mod\_f=funkcje

[12] „Normalizacja BoxKox”, Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://statorials.org/pl/przemiana-box-coxa-w-sluze-powietrzna/

[13] „Preprocessing”, Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

[14] „Regresja liniowa”, Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://www.naukowiec.org/wiedza/statystyka/regresja-liniowa\_765.html

[15] „Klasyfikacja”, Dostęp: 4 styczeń 2025. [Online]. Dostępne na: https://prompterai.pl/portal/wprowadzenie-do-uczenia-maszynowego-klasyfikacja-regresja-klasteryzacja/