

**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](#_Toc182677488)

[1.1. Cel 3](#_Toc182677489)

[2. TECHNOLOGIA I METODY 4](#_Toc182677490)

[2.1. Środowisko programistyczne 4](#_Toc182677491)

[2.2. Środowisko wirtualne 4](#_Toc182677492)

[2.3. Tworzenie pustego pakietu 5](#_Toc182677493)

[2.4. System kontroli wersji 5](#_Toc182677494)

[3. OGÓLNY OPIS PAKIETU 6](#_Toc182677495)

[3.1. STANDARDY TWORZENIA POSZCZEGÓLNYCH PODMODUŁÓW 6](#_Toc182677496)

[3.2. WYKORZYSTANE PAKIETY 6](#_Toc182677497)

[4. SZCZEGÓŁOWY OPIS POSZCZEGÓLNYCH PODMODUŁÓW 7](#_Toc182677498)

[4.1. PODMODUŁ ‘\_errors’ 7](#_Toc182677499)

[4.2. PODMODUŁ ‘check’ 9](#_Toc182677501)

[5. PRZYKŁADY UŻYCIA 14](#_Toc182677502)

[6. PODSUMOWANIE, DYSKUSJA, WNIOSKI 15](#_Toc182677503)

[7. LITERATURA 16](#_Toc182677504)

# 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 wymienionyh 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:

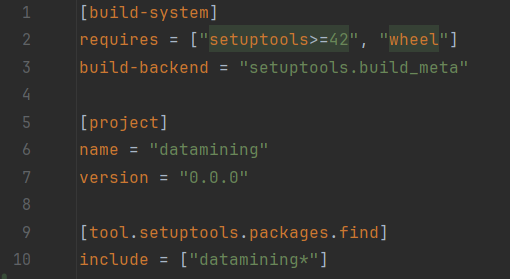
*conda activate base* – aktywowanie domyślnego środowiska Condy

*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,
* **normalizations** – rozbudowany podmoduł przechowujący funkcje normalizacyjne 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 ofichalnych dokumentacji + wykorzystana wersja Pythona

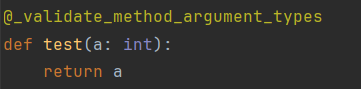
# SZCZEGÓŁOWY OPIS POSZCZEGÓLNYCH PODMODUŁÓW

## 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*

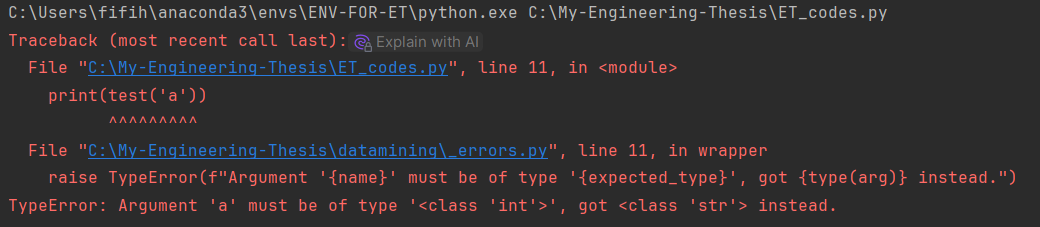
* Wywołanie funkcji z argumentem ‘a’ o typie integer (zgodnym z definicją funkcji) powoduje wykonanie się funkcji (dekorator nie podnosi błędu):

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

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

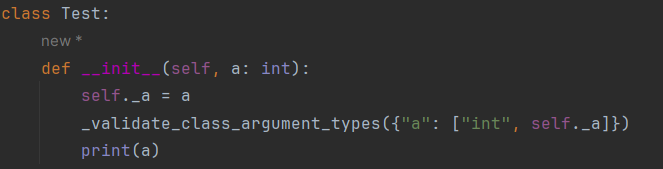
* Wywołanie funkcji z argumentem ‘a’ o innym typie niż integer (niezgodnym   
  z definicją funkcji) powoduje podniesienie błędu przez dekorator:

 *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*

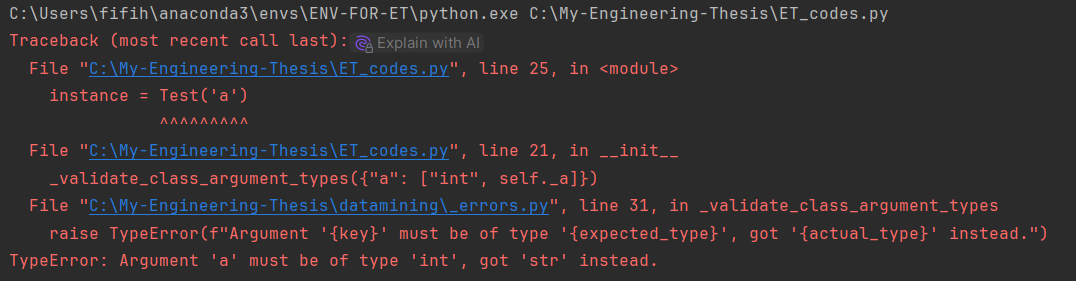
* Utworzenie instancji klasy ‘Test’ z argumentem ‘a’ podanym do konstruktora klasy o typie integer (zgodnym z definicją konstruktora) powoduje stworzenie się instancji (metoda nie podnosi błędu):

*  
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:

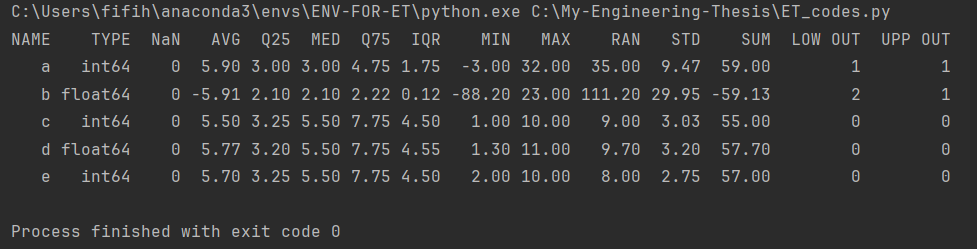
* zmienne numeryczne, np. 1; 1.12; 1+2i; -4
* zmienne kategoryczne, np. ‘a’; ‘man’; True
* zmienne w postaci szeregów czasowych, np. 7 maj 2024 01:02:03.45
* zmienne w postaci interwałów czasowych, np. 2 lata 3 dni 4 godziny; 3 minuty 3 sekundy

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.

* + 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 wchodzą cztery kolumny numeryczne (‘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:

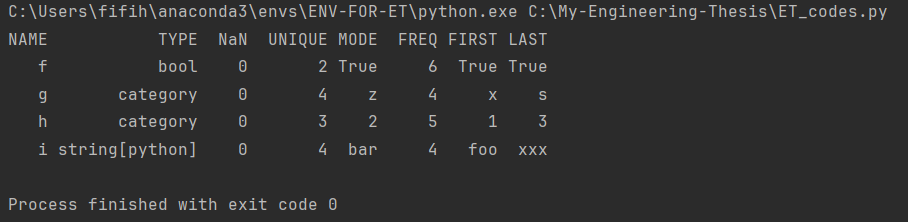
* NAME – nazwie kolumny numerycznej,
* TYPE – typie zmiennej,
* NaN – ilości brakujących obserwacji,
* AVG – wartości średniej,
* Q25 – dolnym kwartylu (25% obserwacji jest mniejszych od tej wartości),
* Q50 – medianie,
* Q75 – górnym kwartylu (25% obserwacji jest większych od tej wartości),
* IQR – 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)

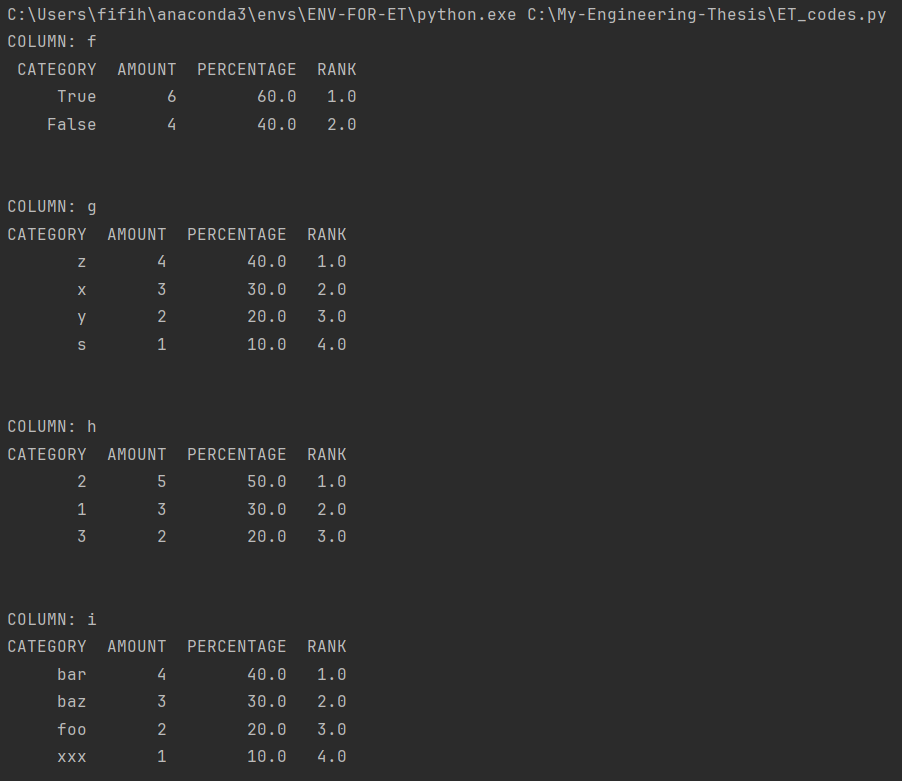
Ponadto zaimplementowano piątą metodę wewnątrz tego podmodułu, podsumowywującą działanie wyżej opisanych czterech metod

* + 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

Time series data opis

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

Time interval data opis

* + 1. Metoda check\_data

check\_data opis

# PRZYKŁADY UŻYCIA

# PODSUMOWANIE, DYSKUSJA, WNIOSKI

# LITERATURA

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