

**Usługi sieciowe w biznesie**

Dokumentacja projektu

**Gabriel Lichacz**

Rzeszów, 2022

Spis treści

[1. Wstęp 3](#_Toc103016156)

[1.1. Apache Druid 3](#_Toc103016157)

[1.2. MySQL 4](#_Toc103016158)

[1.3. PyDruid 5](#_Toc103016159)

[1.4. SQLAlchemy 5](#_Toc103016160)

[1.5. Dane - mniejsze 5](#_Toc103016161)

[1.6. Dane - większe 7](#_Toc103016162)

[2. Budowa systemu 7](#_Toc103016163)

[3. Testy wydajności 8](#_Toc103016164)

[3.1. Wczytanie danych 8](#_Toc103016165)

[3.2. Połączenie z bazą danych 8](#_Toc103016166)

[3.3. Kwerendy 9](#_Toc103016167)

[3.3.1. Baza mniejsza 9](#_Toc103016168)

[3.3.2. Baza większa 9](#_Toc103016169)

[4. Analiza 10](#_Toc103016170)

[4.1. Klasteryzacja 10](#_Toc103016171)

[4.2. Model Support Vector Machines (SVM) 11](#_Toc103016172)

[4.3. Szybkość 12](#_Toc103016173)

[5. Wnioski 13](#_Toc103016174)

[6. Spis ilustracji 14](#_Toc103016175)

[7. Spis tabel 14](#_Toc103016176)

[8. Źródła 15](#_Toc103016177)

Wersja online dokumentu oraz dane i kod źródłowy w języku Python dostępne na stronie: *https://github.com/gabriellichacz/Druid\_MySQL\_comparison*

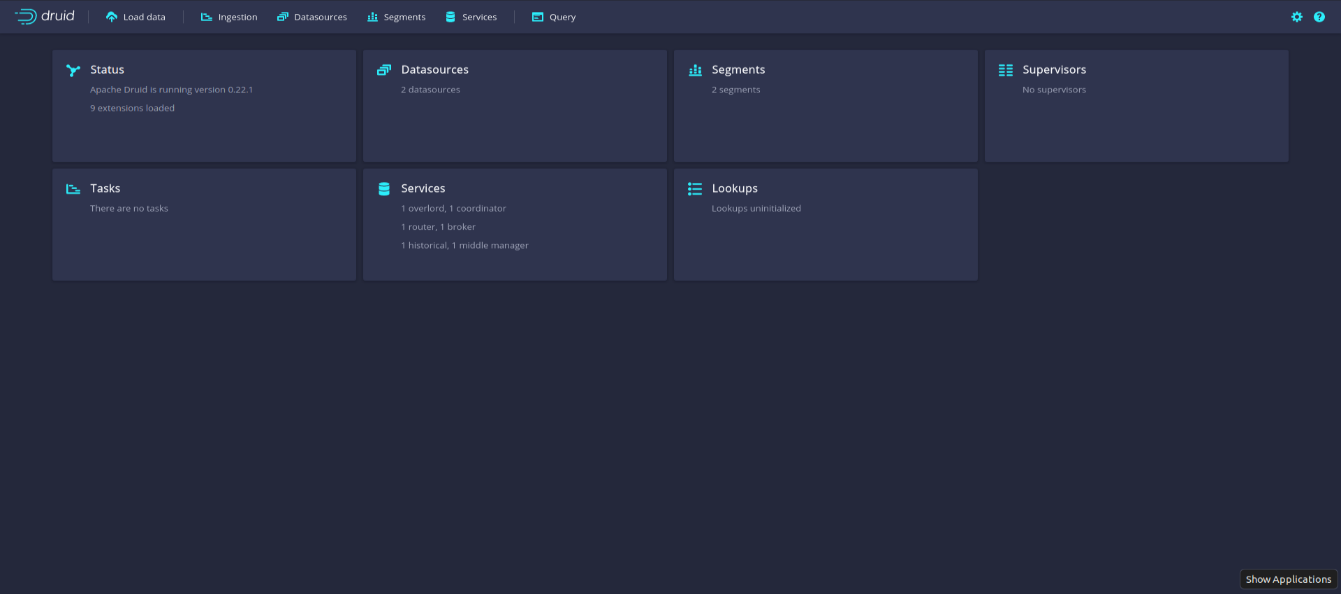
# Wstęp

Tematem projektu jest porównanie szybkości działania połączeń z bazami danych Apache Druid oraz MySQL przy wykorzystaniu języka Python. Sprawdzana będzie szybkość połączeń, wykonywania zapytań oraz przeprowadzania analizy.

## Apache Druid

Apache Druid jest bazą danych ukierunkowaną na wysoką wydajność w czasie rzeczywistym. Jego głównym założeniem jest zredukowanie czasu do wglądu i działania.

Druid jest zaprojektowany do pracy, gdzie wymagane są szybkie kwerendy i przyjmowanie danych. Najlepiej działa zasilając UI, wykonując tymczasowe kwerendy czy obsługując wiele transakcji użytkowników. Druid jest open source’owym odpowiednikiem hurtowni danych.

  
rys. ‑ Interface Apache Druid’a

Druid jest bazą danych zorientowaną kolumnowo. Oznacza to, że każda kolumna jest kompresowana i przechowywana indywidualnie. Dzięki temu Druid w trakcie wykonywania kwerendy odczytuje pojedynczą tabelę, co zwiększa prędkość działania.

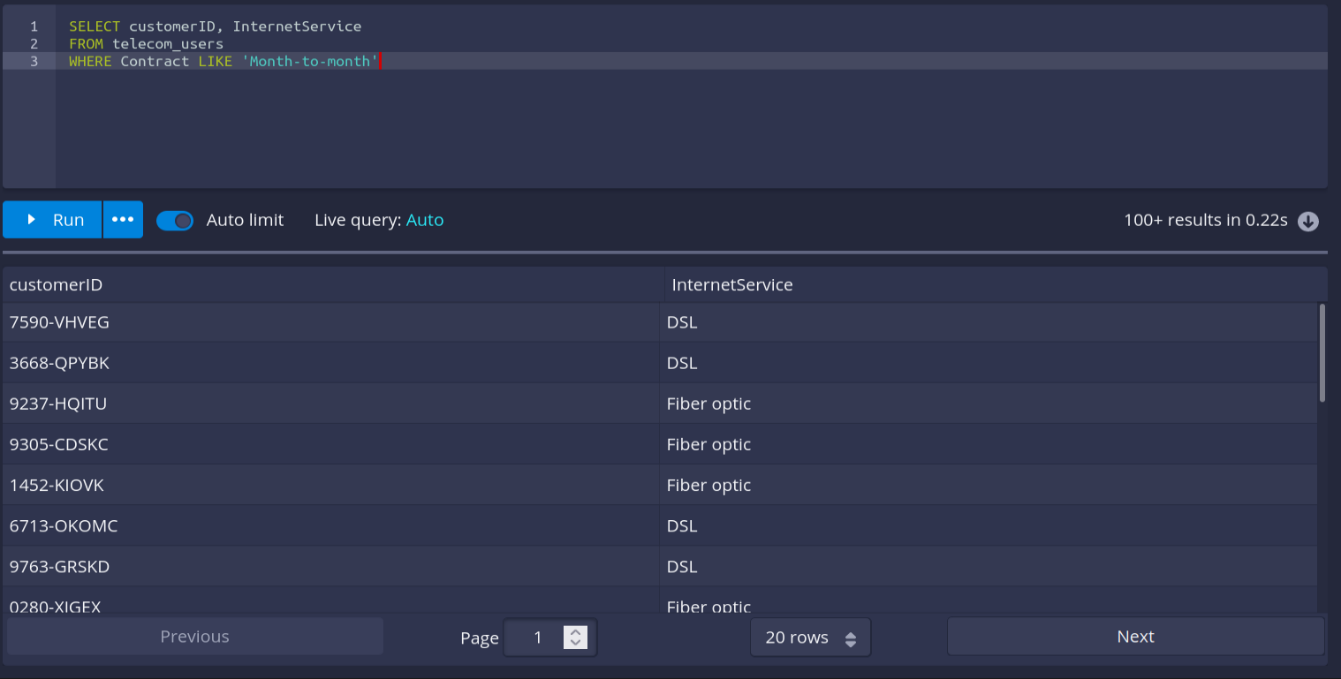
|  |  |  |  |
| --- | --- | --- | --- |
| No | Imię | Nazwisko | Nr telefonu |
| 1 | Zbigniew | Wysocki | 567 541 584 |
| 2 | Jan | Kowalski | 987 487 544 |
| 3 | Karol | Konieczny | 547 541 698 |

rys. ‑ Baza danych zorientowana wierszowo

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No |  | Imię |  | Nazwisko |  | Nr telefonu |
| 1 |  | Zbigniew |  | Wysocki |  | 567 541 584 |
| 2 |  | Jan |  | Kowalski |  | 987 487 544 |
| 3 |  | Karol |  | Konieczny |  | 547 541 698 |

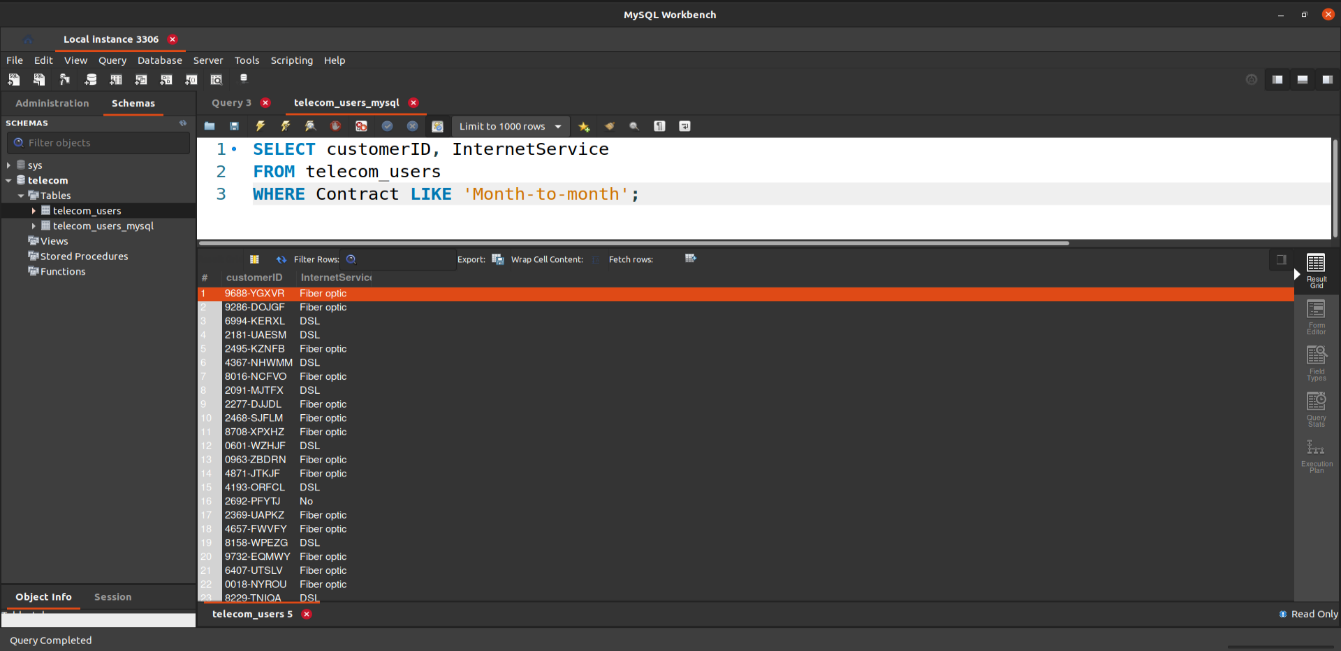
rys. ‑ Baza danych zorientowana kolumnowo

Druid obsługuje kwerendy w języku Druid SQL. Jest to język bliźniaczo podobny do standardowego SQL. Nie obsługuje jednak złączeń typu JOIN.

  
rys. ‑ Przykładowe zapytanie w Apache Druid

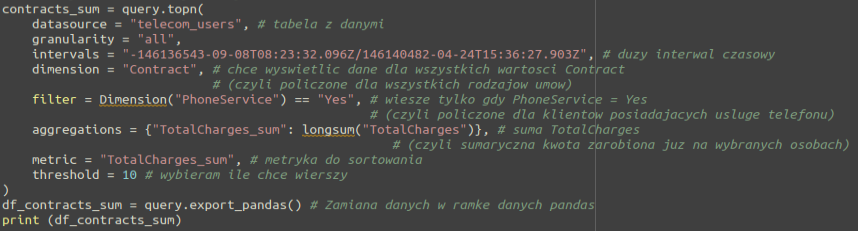
## MySQL

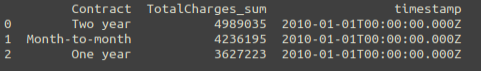
MySQL to jeden z najpowszechniej używanych systemów zarządzania relacyjnymi bazami danych. W przeciwieństwie do Apache Druid jest zorientowany wierszowo oraz posiada znacznie bardziej rozbudowany język SQL.

  
rys. ‑ Przykładowe zapytanie w MySQL

## PyDruid

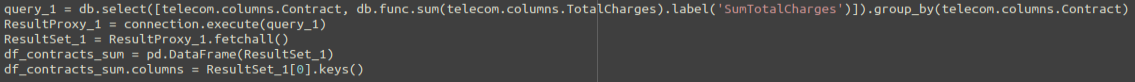
PyDruid to biblioteka przeznaczona dla języka Python. Umożliwia budowanie kwerend Apache Druid w języku Python. Dane otrzymywane są w formacie .json, można je później przekonwertować do ramki danych z biblioteki pandas lub wyeksportować w celu użycia z innym językiem.

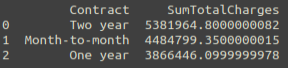
  
rys. ‑ Przykładowa kwerenda PyDruid

  
rys. ‑ Wynik przykładowej kwerendy PyDruid przekonwertowany na ramkę danych

## SQLAlchemy

SQLAlchemy to biblioteka dla języka Python służąca do pracy z bazami danych. Posiada funkcje umożliwiające połączenie się z najpopularniejszymi rodzajami baz danych (SQLite, MySQL, itd.). Otrzymane dane można przekonwertować na ramkę danych pandas.

  
rys. ‑ Przykładowa kwerenda MySQL w SQLAlchemy

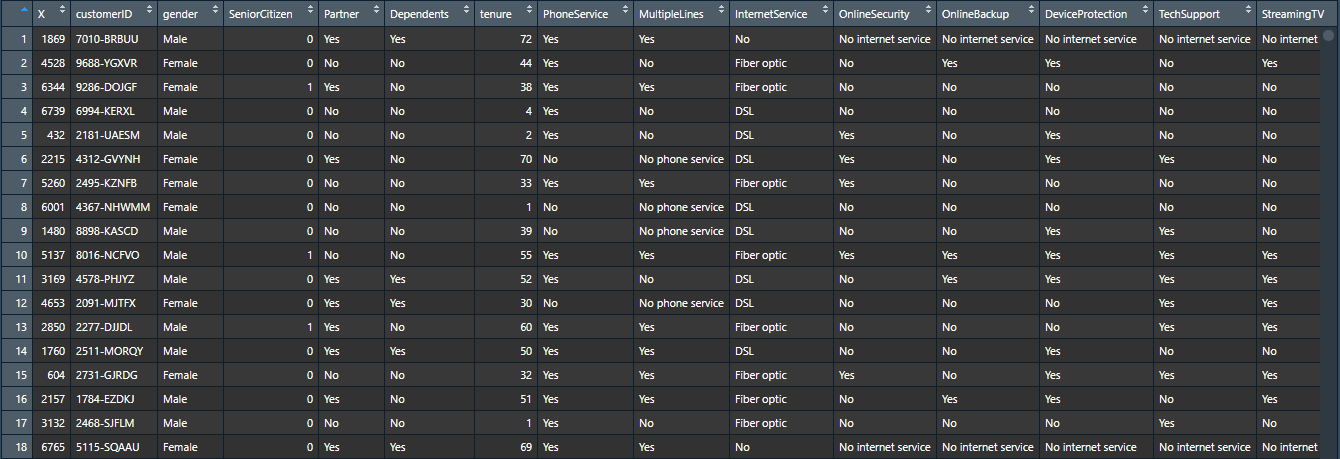
  
rys. ‑ Wynik przykładowej kwerendy MySQL w SQLAlchemy przekonwertowany na ramkę danych

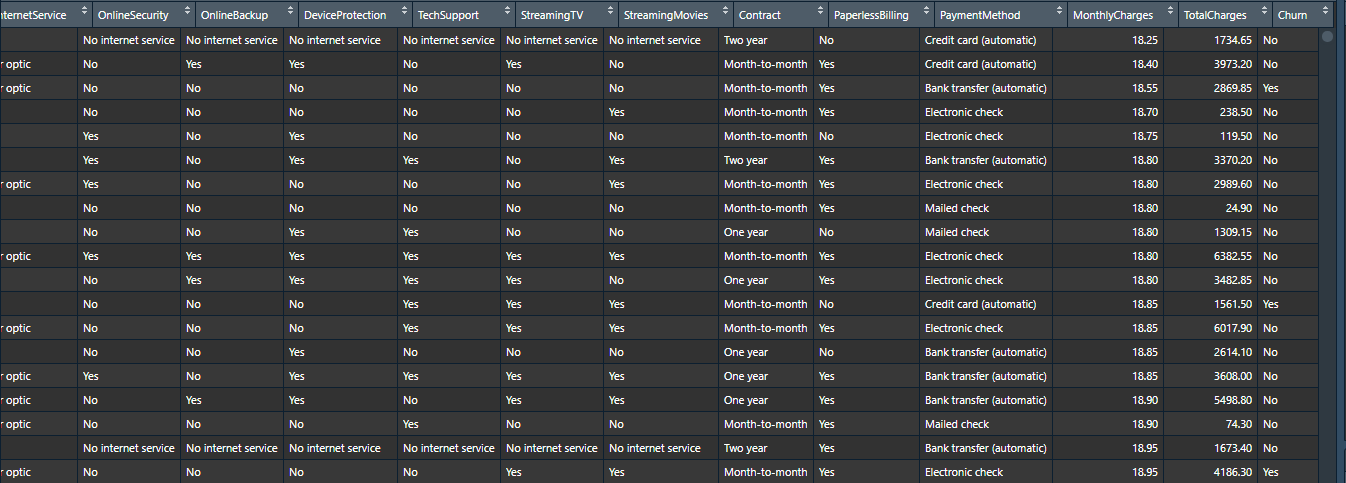
## Dane - mniejsze

Wykorzystane w projekcie dane to baza klientów firmy oferującej usługi telekomunikacyjne. Występuje w niej 5986 wierszy. Na tych danych przeprowadzona zostanie przykładowa analiza.

Baza danych zawiera kolumny:

1. X – ID w zbiorze danych
2. customerID – ID klienta
3. gender – płeć klienta
4. SeniorCitizen – czy klient jest na emeryturze
5. Partner – czy klient jest w związku małżeńskim
6. Dependents – czy klient utrzymuje inne osoby (np. dzieci)
7. tenure – ile miesięcy klient ma już podpisaną umowę z firmą
8. PhoneService – abonament komórkowy
9. MultipleLines – dodatkowa do abonamentu komórkowego
10. InternetService – usługa internetu
11. OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies – dodatkowe usługi związane z połączeniem internetowym
12. Contract – rodzaj umowy
13. PaperlessBilling – czy rozliczenie jest przesyłane formą elektroniczną
14. PaymentMethod – metoda płatności za abonament
15. MonthlyCharges – miesięczna kwota abonamentu
16. TotalCharges – kwota jaką klient już zapłacił przez cały czas trwania jego umów
17. Churn – wskaźnik rezygnacji

  
rys. ‑ Analizowane dane, cz.1

  
rys. ‑ Analizowane dane, cz.2

## Dane - większe

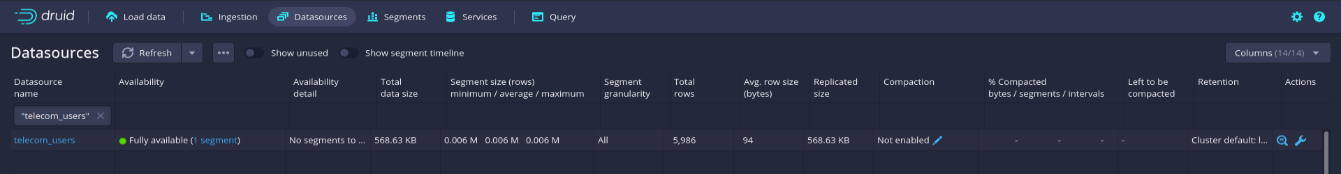
Zbiór danych zawiera informacje o klientach firmy telekomunikacyjnej. Zawiera kolumny:

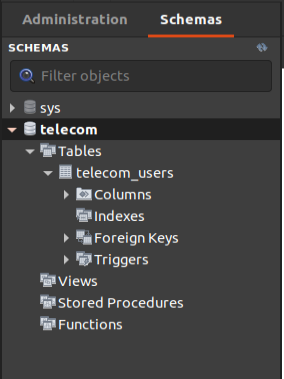
1. msisdn – unikalny numer ID przypisany każdemu numerowi telefonu
2. gender – płeć użytkownika
3. year\_of\_birth – rok urodzenia użytkownika
4. system\_status – status usługi używanej przez klienta
5. mobile\_type – typ umowy (sprzedaż przedpłacona lub abonament)
6. value\_segment – klasyfikacja klienta (jak dobrze spełnia cele biznesowe firmy)

  
rys. ‑ Analizowane dane - prawie 5 mln rekordów

# Budowa systemu

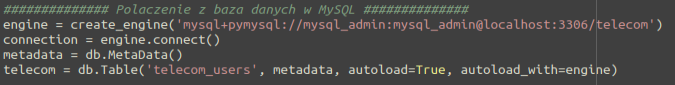
Dane, zaimportowane z pliku .csv, znajdują się w bazie danych Apache Druid na serwerze lokalnym. Drugi zestaw danych znajduje się w bazie MySQL.

  
rys. ‑ Baza telecom\_users w Apache Druid

  
rys. ‑ Baza telecom\_users w MySQL

Za pomocą biblioteki PyDruid utworzone jest połączenie między bazą danych a środowiskiem programistycznym. W podobny sposób stworzone jest połączenie z bazą MySQL przy użyciu funkcji z biblioteki SQLAlchemy.

  
rys. ‑ Połączenie z bazą danych Apache Druid za pomocą biblioteki PyDruid

  
rys. ‑ Połączenie z bazą danych MySQL za pomocą biblioteki SQLAlchemy

# Testy wydajności

W ramach testów sprawdzony został czas połączenie z bazami danych, dwóch typów kwerend oraz wykonania przykładowej analizy. Czas mierzony w sekundach.

## Wczytanie danych

MySQL znacznie gorzej radzi sobie z wczytywaniem danych. Około 4,5 mln rekordów wczytywało się około 8 godzin. Apache Druid z tym samym zbiorem poradził sobie w czasie krótszym niż 5 minut.

## Połączenie z bazą danych

Połączenie z bazą danych Apache Druid było szybsze. Jest to jednak niewielka różnica, zaważając na to, że oba połączenia wykonały się w mniej niż sekundę. Liczba danych w bazie nie wpływa na prędkość łączenia się z bazą danych.

|  |  |  |
| --- | --- | --- |
| **Nazwa bazy i biblioteki** | Apache Druid - PyDruid | MySQL - SQLAlchemy |
| **~Czas [s]** | 0.000056 | 0.011643 |

Tabela Porównanie czasów połączenia baz danych

## Kwerendy

### Baza mniejsza

|  |  |  |
| --- | --- | --- |
| **Kwerenda wczytująca** | **Nazwa bazy i biblioteki** | |
| Apache Druid – PyDruid | MySQL – SQLAlchemy |
| Filtrowane dane | 1.3624 s | 0.0785 s |
| Całość danych | 0.4759 s | 0.2784 s |

Tabela Porównanie czasu wykonania kwerend na mniejszej bazie danych

### Baza większa

Ze względu na znaczne wydłużenie czasu wykonywania zapytania wybierającego wszystkie dane przy użyciu języka Python, ten test przeprowadzony został również bezpośrednio na bazach danych przy użyciu komend w terminalu Linux. W przypadku wykorzystania języka Python (biblioteka PyDruid oraz SQLAlchemy) wczytanie prawie 5 milionów rekordów zajmuje co najmniej 6 godzin. Po tym czasie wykonywanie kwerendy zostaje przerwane.

Zapytanie SELECT \* FROM data w dsql dla Apache Druid przerywa się po wybraniu około 1.15 mln rekordów. Z tego względu w MySQL wybrane zostało tyle samo rekordów.

|  |  |  |
| --- | --- | --- |
| **Kwerenda wczytująca** | **Nazwa bazy danych** | |
| Apache Druid – dsql | MySQL – mysql |
| 1.15 mln rekordów | 303 s | 25 s |

Tabela Porównanie czasu wykonania kwerendy w terminalu na większej bazie danych

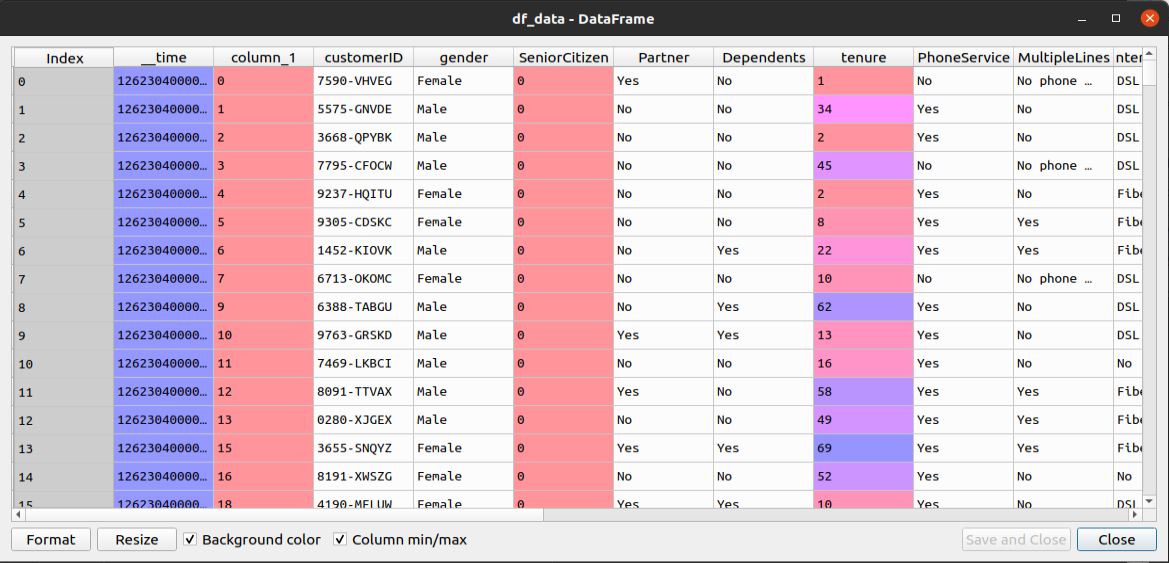
W przypadku danych filtrowanych wykonane zostały 2 kwerendy:

1. SELECT \* FROM data WHERE gender LIKE ‘male’;
2. SELECT count(mobile\_type), mobile\_type FROM data   
   WHERE system\_status LIKE ‘ACTIVE’ GROUP BY mobile\_type

|  |  |  |
| --- | --- | --- |
| **Kwerenda wczytująca** | **Nazwa bazy i biblioteki** | |
| Apache Druid – PyDruid | MySQL – SQLAlchemy |
| Filtrowane dane 1 | 7.24 s | 27.93 s |
| Filtrowane dane 2 | 0.33 s | 13.12 s |

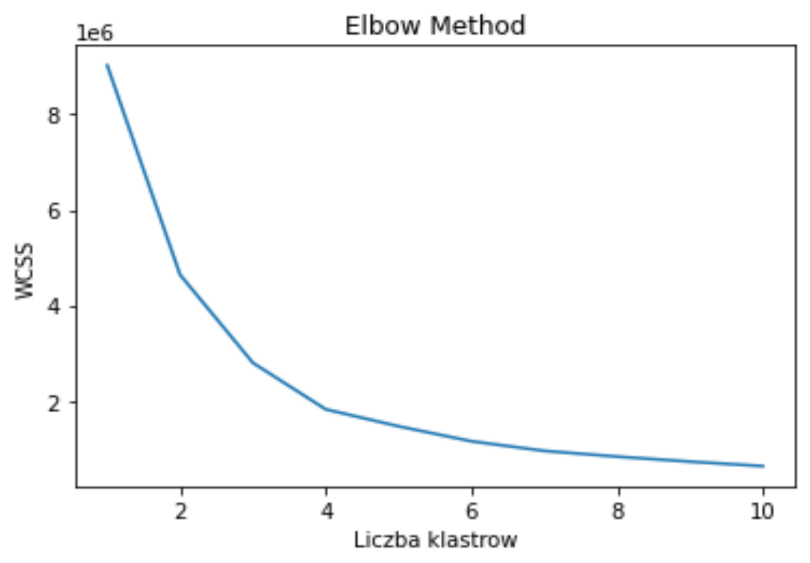
Tabela Porównanie czasu wykonania kwerend z użyciem Pythona na większej bazie danych

# Analiza

  
rys. ‑ Wczytane dane w postaci ramki danych pandas

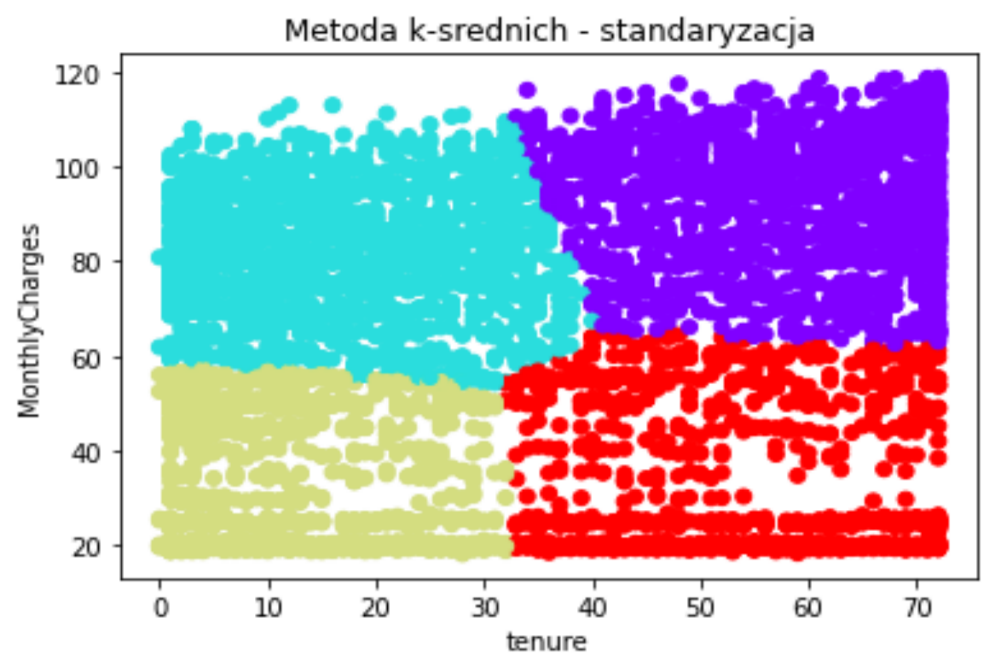
## Klasteryzacja

Metoda łokciowa to badanie optymalnej liczby klastrów. Dla analizowanych danych najlepszą liczbą okazała się 4.

  
rys. ‑ Metoda łokciowa

Dane ustandaryzowane oraz w pierwotnej formie zostały sklastrowane ze względu na miesięczny abonament i długość trwania umowy. Dzięki podziałowi firma może zaplanować działanie na przyszłość i skierować odpowiednie oferty dla danej grupy klientów. Oba sposoby klasteryzacji zwróciły podobne wyniki. Klienci podzieleni zostali na 4 grupy:

Grupa I - płacący krótko i mało.  
Grupa II - płacący długo i mało.  
Grupa III - płacący krótko i dużo.  
Grupa IV - płacący długo i dużo.

  
rys. ‑ Klasteryzacja danych ze standaryzacją

Celem firmy w stosunku do grupy I powinno być przekonanie klientów, by pozostali jak najdłużej i zdecydowali się na droższą usługę lub więcej usług. Rozwiązaniem może być czasowa zniżka na droższą usługę oraz program lojalnościowy.

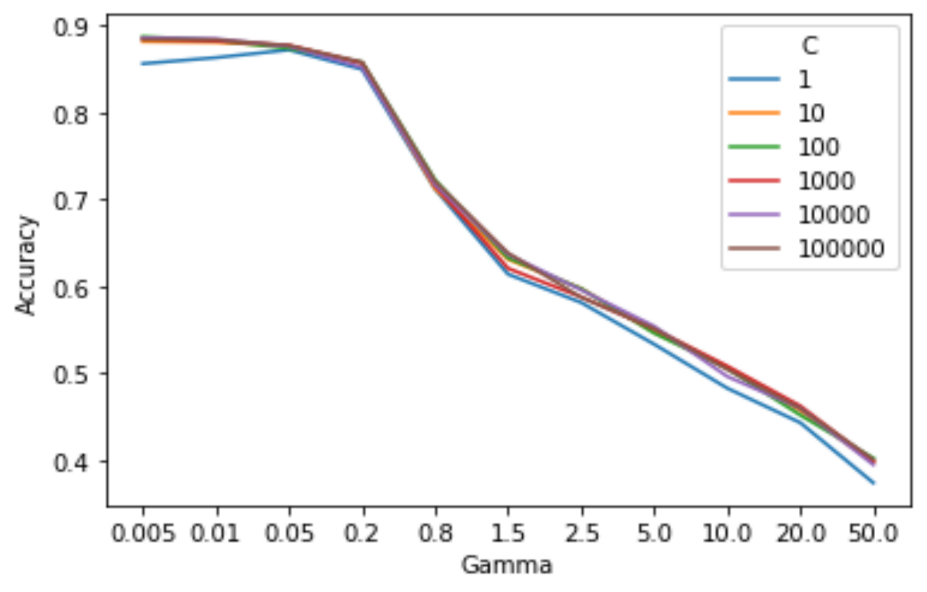
Celem firmy w stosunku do grupy II powinno być przekonanie klientów, by zdecydowali się na droższą usługę lub więcej usług, można zastosować czasową zniżkę na droższą usługę.

Celem firmy w stosunku do grupy III powinno być przekonanie klientów, by pozostali jak najdłużej. Można im zaoferować program lojalnościowy z wyszczególnionymi jasno zniżkami na przyszłość

Celem firmy w stosunku do grupy IV powinno być utrzymanie jak najdłużej lojalnych klientów. Można zaoferować im np. program lojalnościowy ze zniżkami lub nowymi usługami.

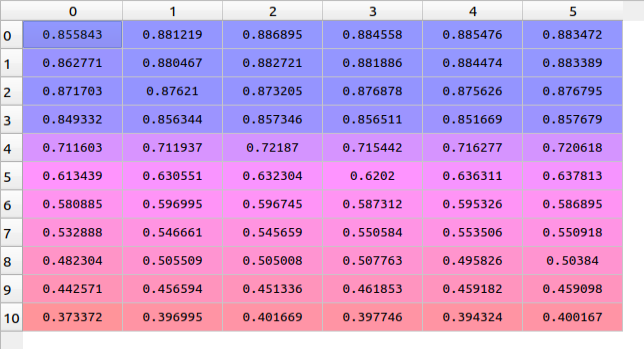
## Model Support Vector Machines (SVM)

Używane w analizie jądro modelu SVM to radialna funkcja bazowa. Model badany dla zmiennych parametrów gamma oraz C. Gamma zmieniała się w przedziale od 0.005 do 50, a C w przedziale od 1 do 100000. Zbiór danych testowych wynosił 20% całego zbioru danych.

  
rys. ‑ Dokładności modelu SVM - wykres

Najlepsza dokładność modelu została uzyskana dla małych wartości gamma. Parametr C nie wpływał znacznie na wyniki. Oznacza to, że im zasięg oddziaływania pojedynczej próbki testowej był większy, tym lepsza dokładność była uzyskiwana.

Najlepsza dokładność wyniosła około 88.69% dla C = 100 oraz gamma = 0.005. Taki model można następnie wykorzystać do dokładniejszej klasyfikacji danych.



rys. ‑ Dokładności stworzonego modelu SVM

## Szybkość

Powyższa analiza w połączeniu z bazą danych Apache Druid wykonywała się średnio około 723 sekundy. W przypadku bazy MySQL czas wydłużył się do średnio 807 sekund.

# Wnioski

W testach na mniejszej bazie danych (prawie 6 tys. rekordów) czas wykonywania kwerend był krótszy w przypadku bazy MySQL. Przy większym zestawie danych (prawie 5 mln rekordów) Apache Druid poradził sobie znacznie lepiej – różnice wynosiły kolejno około 20 i 13 sekund. Również przykładowa analiza została wykonana szybciej, gdy dane wczytano z Druida. W przypadku wczytywania wszystkich rekordów z baz danych (co ostatecznie okazało się niemożliwe) górował MySQL. Udało mu się wczytać ponad 1.15 mln rekordów w czasie około 25 sekund. Apache Druid dokonał tego w 303 sekundy. W przypadku wczytywania zestawu danych do bazy, Druid poradził sobie znacznie lepiej od MySQLa. Wczytanie około 4.5 mln rekordów zajęło mu mniej niż 5 minut, podczas gdy MySQL wykonał zadanie w 8 godzin.

# Spis ilustracji

[rys. 1‑1 Interface Apache Druid’a 3](#_Toc102679723)

[rys. 1‑2 Baza danych zorientowana wierszowo 3](#_Toc102679724)

[rys. 1‑3 Baza danych zorientowana kolumnowo 3](#_Toc102679725)

[rys. 1‑4 Przykładowe zapytanie w Apache Druid 4](#_Toc102679726)

[rys. 1‑5 Przykładowe zapytanie w MySQL 4](#_Toc102679727)

[rys. 1‑6 Przykładowa kwerenda PyDruid 5](#_Toc102679728)

[rys. 1‑7 Wynik przykładowej kwerendy PyDruid przekonwertowany na ramkę danych 5](#_Toc102679729)

[rys. 1‑8 Przykładowa kwerenda MySQL w SQLAlchemy 5](#_Toc102679730)

[rys. 1‑9 Wynik przykładowej kwerendy MySQL w SQLAlchemy przekonwertowany na ramkę danych 5](#_Toc102679731)

[rys. 1‑10 Analizowane dane, cz.1 6](#_Toc102679732)

[rys. 1‑11 Analizowane dane, cz.2 6](#_Toc102679733)

[rys. 1‑12 Analizowane dane - prawie 5 mln rekordów 7](#_Toc102679734)

[rys. 2‑1 Baza telecom\_users w Apache Druid 7](#_Toc102679735)

[rys. 2‑2 Baza telecom\_users w MySQL 7](#_Toc102679736)

[rys. 2‑3 Połączenie z bazą danych Apache Druid za pomocą biblioteki PyDruid 8](#_Toc102679737)

[rys. 2‑4 Połączenie z bazą danych MySQL za pomocą biblioteki SQLAlchemy 8](#_Toc102679738)

[rys. 4‑1 Wczytane dane w postaci ramki danych pandas 10](#_Toc102679739)

[rys. 4‑2 Metoda łokciowa 10](#_Toc102679740)

[rys. 4‑3 Klasteryzacja danych ze standaryzacją 11](#_Toc102679741)

[rys. 4‑4 Dokładności modelu SVM - wykres 12](#_Toc102679742)

[rys. 4‑5 Dokładności stworzonego modelu SVM 12](#_Toc102679743)

# Spis tabel

[Tabela 1 Porównanie czasów połączenia baz danych 8](#_Toc102679639)

[Tabela 2 Porównanie czasu wykonania kwerend na mniejszej bazie danych 9](#_Toc102679640)

[Tabela 3 Porównanie czasu wykonania kwerendy w terminalu na większej bazie danych 9](#_Toc102679641)

[Tabela 4 Porównanie czasu wykonania kwerend z użyciem Pythona na większej bazie danych 9](#_Toc102679642)

# Źródła

1. https://druid.apache.org/docs/latest/design/index.html dostęp 20.03.2022
2. [https://github.com/druid-io/pydruid dostęp 20.03.2022](https://github.com/druid-io/pydruid%20dostęp%2020.03.2022)
3. https://pythonhosted.org/pydruid/ dostęp 20.03.2022
4. [https://scikit-learn.org/stable/user\_guide.html dostęp 22.04.2022](https://scikit-learn.org/stable/user_guide.html%20dostęp%2022.04.2022)
5. https://dev.mysql.com/doc/refman/8.0/en/ dostęp 28.04.2022
6. <https://www.kaggle.com/datasets/krishnacheedella/telecom-iot-crm-dataset>  
   dostęp 28.04.2022
7. <https://matplotlib.org/stable/users/index.html> dostęp 28.04.2022
8. <https://numpy.org/doc/stable/> dostęp 28.04.2022
9. https://github.com/gabriellichacz/Druid\_MySQL\_comparison