



**WYDZIAŁ  
MATEMATYKI  
I FIZYKI STOSOWANEJ**  
POLITECHNIKI RZESZOWSKIEJ

**Usługi sieciowe w biznesie**

Dokumentacja projektu

**Gabriel Lichacz**

Rzeszów, 2022

# Spis treści

<b>1. Wstęp.....</b>	<b>3</b>
1.1. Apache Druid .....	3
1.2. MySQL.....	4
1.3. PyDruid .....	5
1.4. SQLAlchemy .....	5
1.5. Dane - mniejsze.....	5
1.6. Dane - większe .....	7
<b>2. Budowa systemu.....</b>	<b>7</b>
<b>3. Testy wydajności .....</b>	<b>8</b>
3.1. Wczytanie danych .....	8
3.2. Połączenie z bazą danych.....	8
3.3. Kwerendy .....	9
3.3.1. Baza mniejsza .....	9
3.3.2. Baza większa.....	9
<b>4. Analiza .....</b>	<b>10</b>
4.1. Klasteryzacja.....	10
4.2. Model Support Vector Machines (SVM).....	11
4.3. Szybkość .....	12
<b>5. Wnioski .....</b>	<b>13</b>
<b>6. Spis ilustracji .....</b>	<b>14</b>
<b>7. Spis tabel.....</b>	<b>14</b>
<b>8. Źródła.....</b>	<b>15</b>

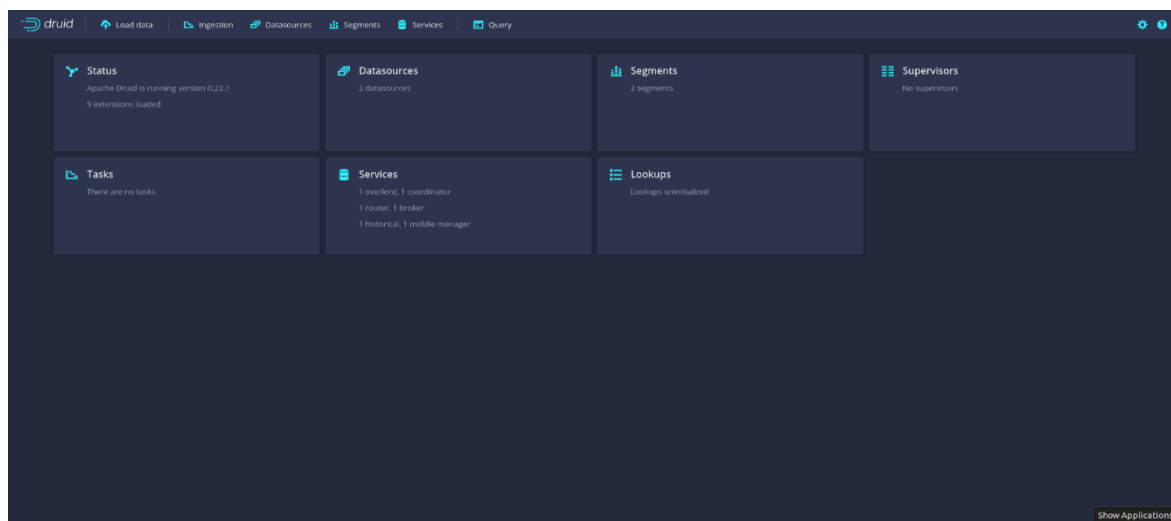
# 1. Wstęp

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.

## 1.1. 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. 1-1 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.

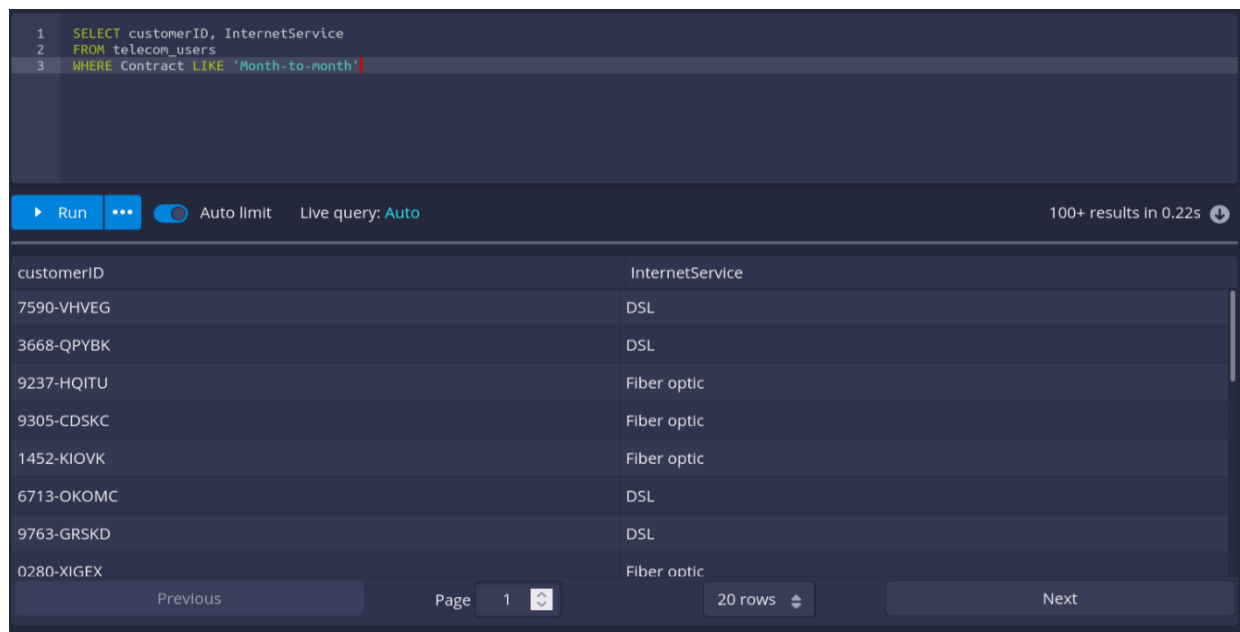
No	Imię	Nazwisko	Nr telefonu
1	Zbigniew	Wysocki	567 541 584
2	Jan	Kowalski	987 487 544
3	Karol	Konieczny	547 541 698

rys. 1-2 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. 1-3 Baza danych zorientowana kolumnowo

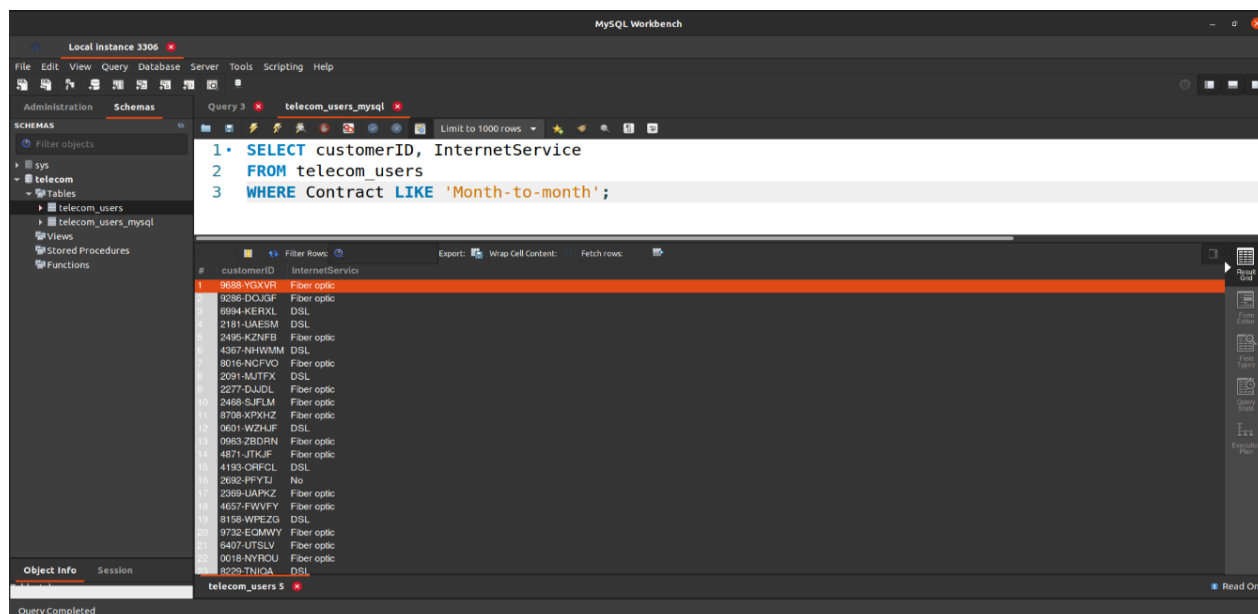
Druid obsługuje kwerendy w języku Druid SQL. Jest to język bliźniaczo podobny do standardowego SQL. Nie obsługuje jednak złączeń typu JOIN.



rys. 1-4 Przykładowe zapytanie w Apache Druid

## 1.2. 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. 1-5 Przykładowe zapytanie w MySQL

### 1.3. 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.

```
contracts_sum = query.topn(  
    datasource = "telecom_users", # tabela z danymi  
    granularity = "all",  
    intervals = "-146136543-09-08T08:23:32.096Z/146140482-04-24T15:36:27.903Z", # duży interwał czasowy  
    dimension = "Contract", # chce wyswietlic dane dla wszystkich wartosci Contract  
    # (czyli policzone dla wszystkich rodzajow umow)  
    filter = Dimension("PhoneService") == "Yes", # wiesz tylko gdy PhoneService = Yes  
    # (czyli policzone dla klientow posiadajacych usluge telefonu)  
    aggregations = {"TotalCharges_sum": longsum("TotalCharges")}, # suma TotalCharges  
    # (czyli sumaryczna kwota zarobiona juz na wybranych osobach)  
    metric = "TotalCharges_sum", # metryka do sortowania  
    threshold = 10 # wybieram ile chce wierszy  
)  
df_contracts_sum = query.export_pandas() # Zamiana danych w ramke danych pandas  
print(df_contracts_sum)
```

rys. 1-6 Przykładowa kwerenda PyDruid

	Contract	TotalCharges_sum	timestamp
0	Two year	4989035	2010-01-01T00:00:00.000Z
1	Month-to-month	4236195	2010-01-01T00:00:00.000Z
2	One year	3627223	2010-01-01T00:00:00.000Z

rys. 1-7 Wynik przykładowej kwerendy PyDruid przekonwertowany na ramkę danych

### 1.4. 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.

```
query_1 = db.select([telecom.columns.Contract, db.func.sum(telecom.columns.TotalCharges).label("SumTotalCharges")]).group_by(telecom.columns.Contract)  
ResultProxy_1 = connection.execute(query_1)  
ResultSet_1 = ResultProxy_1.fetchall()  
df_contracts_sum = pd.DataFrame(ResultSet_1)  
df_contracts_sum.columns = ResultSet_1[0].keys()
```

rys. 1-8 Przykładowa kwerenda MySQL w SQLAlchemy

	Contract	SumTotalCharges
0	Two year	5381964.8000000082
1	Month-to-month	4484799.3500000015
2	One year	3866446.0999999978

rys. 1-9 Wynik przykładowej kwerendy MySQL w SQLAlchemy przekonwertowany na ramkę danych

### 1.5. 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 przysyłane 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

X	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV
1	1869	7010-BRBUU	Male	0	Yes	Yes	72	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service
2	4528	9688-YGXVR	Female	0	No	No	44	Yes	No	Fiber optic	No	Yes	Yes	No
3	6344	9286-DOJGF	Female	1	Yes	No	38	Yes	Yes	Fiber optic	No	No	No	No
4	6739	6994-KERXL	Male	0	No	No	4	Yes	No	DSL	No	No	No	No
5	432	2181-UAESM	Male	0	No	No	2	Yes	No	DSL	Yes	No	Yes	No
6	2215	4312-GVYNH	Female	0	Yes	No	70	No	No phone service	DSL	Yes	No	Yes	No
7	5260	2495-KZNF8	Female	0	No	No	33	Yes	Yes	Fiber optic	Yes	No	No	No
8	6001	4367-NHWMW	Female	0	No	No	1	No	No phone service	DSL	No	No	No	No
9	1480	8898-KASCD	Male	0	No	No	39	No	No phone service	DSL	No	No	Yes	No
10	5137	8016-NCFOV	Male	1	No	No	55	Yes	Yes	Fiber optic	Yes	Yes	Yes	Yes
11	3169	4578-PHYFZ	Male	0	Yes	Yes	52	Yes	No	DSL	No	Yes	Yes	Yes
12	4653	2091-MJTFX	Female	0	Yes	Yes	30	No	No phone service	DSL	No	No	No	Yes
13	2850	2277-DIIDL	Male	1	Yes	No	60	Yes	Yes	Fiber optic	No	No	No	Yes
14	1760	2511-MORQY	Male	0	Yes	Yes	50	Yes	Yes	DSL	No	No	Yes	No
15	604	2731-GJRDG	Female	0	No	No	32	Yes	Yes	Fiber optic	Yes	No	Yes	Yes
16	2157	1784-EZDKJ	Male	0	Yes	No	51	Yes	Yes	Fiber optic	No	Yes	Yes	Yes
17	3132	2468-SJFLM	Male	0	No	No	1	Yes	No	Fiber optic	No	No	No	No
18	6765	5115-SQAUA	Female	0	Yes	Yes	69	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service

rys. 1-10 Analizowane dane, cz.1

InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Credit card (automatic)	18.25	1734.65	No
Fiber optic	No	Yes	Yes	No	Yes	No	Month-to-month	Yes	Credit card (automatic)	18.40	3973.20	No
Fiber optic	No	No	No	No	No	No	Month-to-month	Yes	Bank transfer (automatic)	18.55	2869.85	Yes
	No	No	No	No	No	Yes	Month-to-month	Yes	Electronic check	18.70	238.50	No
	Yes	No	Yes	No	No	No	Month-to-month	No	Electronic check	18.75	119.50	No
	Yes	No	Yes	Yes	No	Yes	Two year	Yes	Bank transfer (automatic)	18.80	3370.20	No
Fiber optic	Yes	No	No	No	No	Yes	Month-to-month	Yes	Electronic check	18.80	2989.60	No
	No	No	No	No	No	No	Month-to-month	Yes	Mailed check	18.80	24.90	No
	No	No	Yes	Yes	No	No	One year	No	Mailed check	18.80	1309.15	No
Fiber optic	Yes	Yes	Yes	Yes	Yes	Yes	Month-to-month	Yes	Electronic check	18.80	6382.55	No
	No	Yes	Yes	Yes	Yes	No	One year	Yes	Electronic check	18.80	3482.85	No
	No	No	No	Yes	Yes	Yes	Month-to-month	No	Credit card (automatic)	18.85	1561.50	Yes
Fiber optic	No	No	No	Yes	Yes	Yes	Month-to-month	Yes	Electronic check	18.85	6017.90	No
	No	No	Yes	No	No	No	One year	No	Bank transfer (automatic)	18.85	2614.10	No
Fiber optic	Yes	No	Yes	Yes	Yes	Yes	One year	Yes	Bank transfer (automatic)	18.85	3608.00	No
Fiber optic	No	Yes	Yes	No	Yes	Yes	One year	Yes	Bank transfer (automatic)	18.90	5498.80	No
Fiber optic	No	No	No	Yes	No	No	Month-to-month	Yes	Mailed check	18.90	74.30	No
	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	Yes	Bank transfer (automatic)	18.95	1673.40	No
Fiber optic	No	No	No	No	Yes	Yes	Month-to-month	Yes	Electronic check	18.95	4186.30	Yes

rys. 1-11 Analizowane dane, cz.2

## 1.6. Dane - większe

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

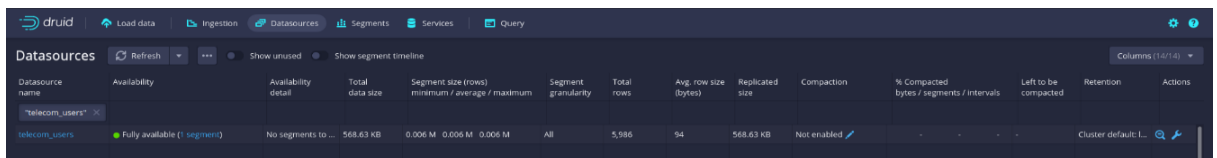
1. msisdn – unikalny numer ID przypisany każdemu numerowi telefonu
2. gender – płeć użytkownika
3. year\_of\_birth – rok urodzenia użytkownika
4. system\_status – status usługi używanej przez klienta
5. mobile\_type – typ umowy (sprzedaż przedpłacona lub abonament)
6. value\_segment – klasyfikacja klienta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

gender	mobile_type	msisdn	system_status	value_segment	year_of_birth
MALE	Prepaid	aeef4233d9ad34e41f7ecf48d646	ACTIVE	Tier_3	1985
MALE	Prepaid	2397d009e705bc8a2654cbb3f48	ACTIVE	Tier_3	1958
MALE	Prepaid	6f05d738919f9283322bae17dc3f	ACTIVE	Tier_3	1976
MALE	Prepaid	45352d9d126f86f40c7eee79a82c	ACTIVE	Tier_3	1996
FEMALE	Prepaid	959b00a279e2785cf81728338c	ACTIVE	Tier_3	1960
null	Prepaid	b74761390712bde1309ac38b46c	ACTIVE	Tier_3	1994
Male	Prepaid	b74761390712bde1309ac38b46c	ACTIVE	Tier_3	1994
Male	Prepaid	1311649d0d7feb132cfaf6a0233	ACTIVE	Tier_3	1992
MALE	Prepaid	8e6e340aa4b1a449305da80327f	ACTIVE	Tier_3	1963
Male	Postpaid	fc91bb6f212bf1aca7f30acb5a8f1	ACTIVE	Tier_3	1978
null	Prepaid	32d4f8f1e7193e334981a42bb0cc	SUSPEND	Tier_3	1969
null	Prepaid	32d4f8f1e7193e334981a42bb0cc	ACTIVE	Tier_3	1969
Male	Prepaid	e78cfd27d4cfd96c79884afdf309f	ACTIVE	Tier_3	1981

rys. 1-12 Analizowane dane - prawie 5 mln rekordów

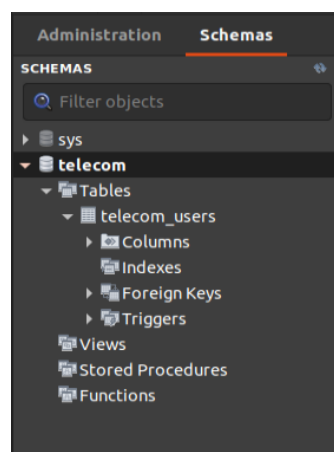
## 2. Budowa systemu

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



Datasource name	Availability	Availability detail	Total data size	Segment size (rows) minimum / average / maximum	Segment granularity	Total rows	Avg. row size (bytes)	Replicated size	Compaction	% Compacted bytes / segments / intervals	Left to be compacted	Retention	Actions
telecom_users	Fully available (1 segment)	No segments to ...	568.63 KB	0.006 M 0.006 M 0.006 M	All	5,586	94	568.63 KB	Not enabled	-	-	-	Cluster default: L...

rys. 2-1 Baza telecom\_users w Apache Druid



rys. 2-2 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.

```
##### Połączenie z baza danych w Apache Druid #####
query = PyDruid('http://localhost:8888', 'druid/v2/')
```

rys. 2-3 Połączenie z bazą danych Apache Druid za pomocą biblioteki PyDruid

```
##### Połączenie z baza danych w MySQL #####
engine = create_engine('mysql+pymysql://mysql_admin:mysql_admin@localhost:3306/telecom')
connection = engine.connect()
metadata = db.MetaData()
telecom = db.Table('telecom_users', metadata, autoload=True, autoload_with=engine)
```

rys. 2-4 Połączenie z bazą danych MySQL za pomocą biblioteki SQLAlchemy

### 3. Testy wydajności

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

#### 3.1. 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.

#### 3.2. Połączenie z bazą danych

Połączenie z bazą danych Apache Druid było szybsze. Jest to jednak niewielka różnica, zważając na to, że oba połączenia wykonały się w mniej niż sekundę. Ilość 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 1 Porównanie czasów połączenia baz danych



### 3.3. Kwerendy

#### 3.3.1. 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 2 Porównanie czasu wykonania kwerend na mniejszej bazie danych

#### 3.3.2. 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 3 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 4 Porównanie czasu wykonania kwerend z użyciem Pythona na większej bazie danych

## 4. Analiza

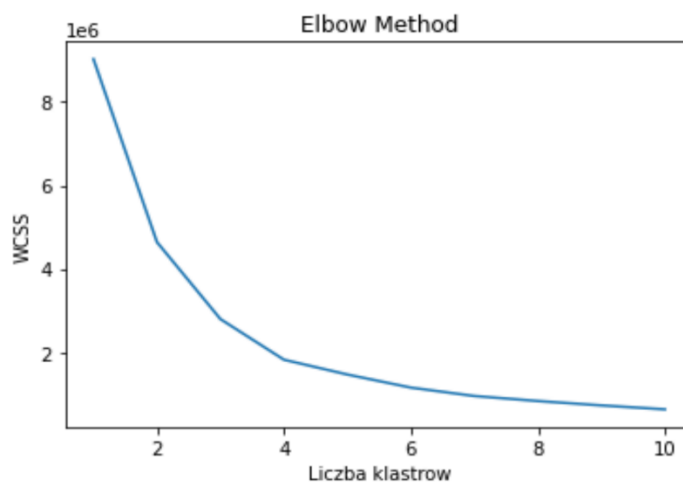
df_data - DataFrame												
Index	_time	column_1	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	nter	
0	12623040000...	0	7590-VHVEG	Female	0	Yes	No	1	No	No phone ...	DSL	
1	12623040000...	1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	
2	12623040000...	2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	
3	12623040000...	3	7795-CFOCW	Male	0	No	No	45	No	No phone ...	DSL	
4	12623040000...	4	9237-HQITU	Female	0	No	No	2	Yes	No	Fibre	
5	12623040000...	5	9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fibre	
6	12623040000...	6	1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	Fibre	
7	12623040000...	7	6713-OKOMC	Female	0	No	No	10	No	No phone ...	DSL	
8	12623040000...	9	6388-TABGU	Male	0	No	Yes	62	Yes	No	DSL	
9	12623040000...	10	9763-GRSKD	Male	0	Yes	Yes	13	Yes	No	DSL	
10	12623040000...	11	7469-LKBCI	Male	0	No	No	16	Yes	No	No	
11	12623040000...	12	8091-TTVAX	Male	0	Yes	No	58	Yes	Yes	Fibre	
12	12623040000...	13	0280-XJGEX	Male	0	No	No	49	Yes	Yes	Fibre	
13	12623040000...	15	3655-SNQYZ	Female	0	Yes	Yes	69	Yes	Yes	Fibre	
14	12623040000...	16	8191-XWSZG	Female	0	No	No	52	Yes	No	No	
15	12623040000...	18	4190-MFLIUN	Female	0	Yes	Yes	10	Yes	No	DSL	

Format    Resize    ☒ Background color    ☒ Column min/max    Save and Close    Close

rys. 4-1 Wczytane dane w postaci ramki danych pandas

### 4.1. Klasteryzacja

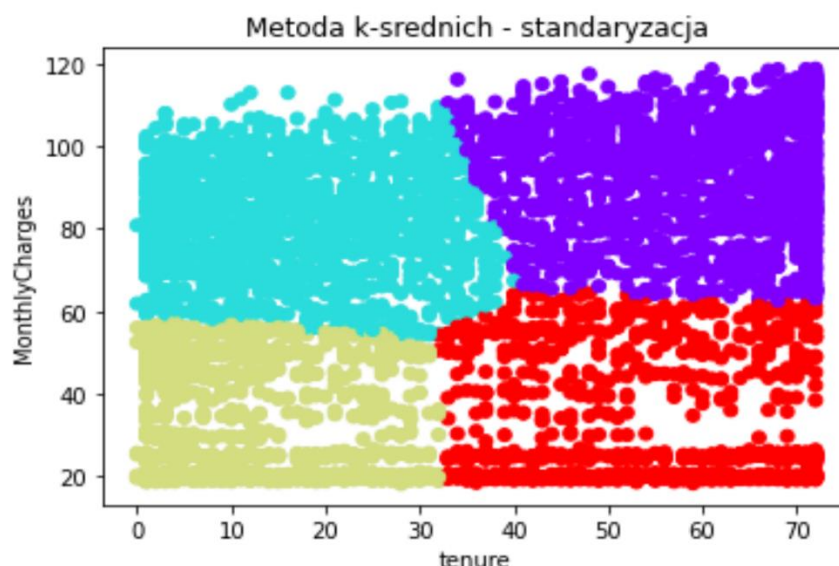
Metoda łokciowa to badanie optymalnej liczby klastrów. Dla analizowanych danych najlepszą liczbą okazała się 4.



rys. 4-2 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. 4-3 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.

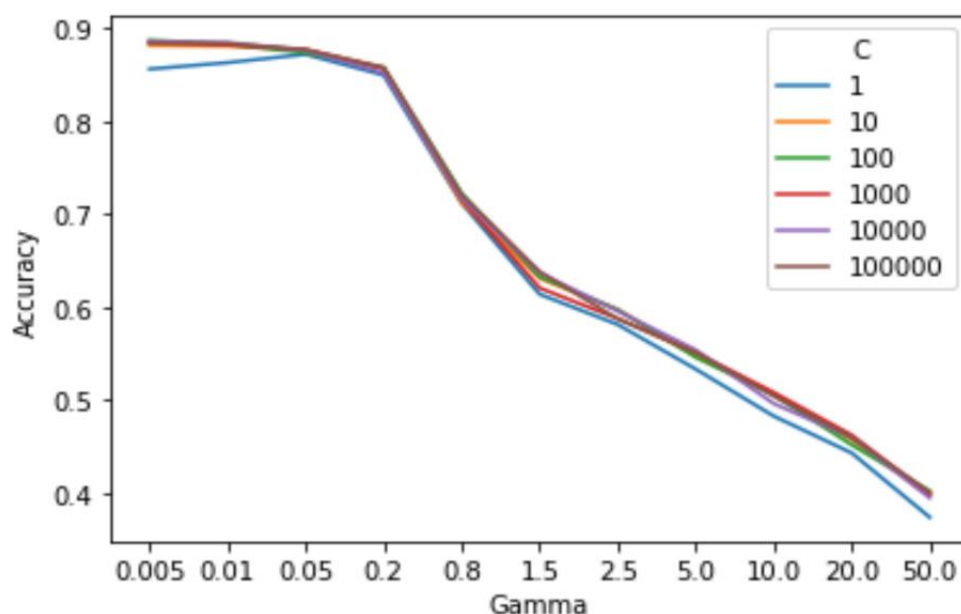
Celem firmy w stosunku do grupy II powinno być przekonanie klientów by zdecydowali się na droższą usługę lub więcej usług, można zastosować czasową zniżkę na droższą usługę.

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.

## 4.2. 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. 4-4 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  $\text{gamma} = 0.005$ . Taki model można następnie wykorzystać do dokładniejszej klasyfikacji danych.

	0	1	2	3	4	5
0	0.855843	0.881219	0.886895	0.884558	0.885476	0.883472
1	0.862771	0.880467	0.882721	0.881886	0.884474	0.883389
2	0.871703	0.87621	0.873205	0.876878	0.875626	0.876795
3	0.849332	0.856344	0.857346	0.856511	0.851669	0.857679
4	0.711603	0.711937	0.72187	0.715442	0.716277	0.720618
5	0.613439	0.630551	0.632304	0.6202	0.636311	0.637813
6	0.580885	0.596995	0.596745	0.587312	0.595326	0.586895
7	0.532888	0.546661	0.545659	0.550584	0.553506	0.550918
8	0.482304	0.505509	0.505008	0.507763	0.495826	0.50384
9	0.442571	0.456594	0.451336	0.461853	0.459182	0.459098
10	0.373372	0.396995	0.401669	0.397746	0.394324	0.400167

rys. 4-5 Dokładności stworzonego modelu SVM

### 4.3. 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.

## 5. 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.

## 6. Spis ilustracji

rys. 1-1 Interface Apache Druid'a.....	3
rys. 1-2 Baza danych zorientowana wierszowo .....	3
rys. 1-3 Baza danych zorientowana kolumnowo .....	3
rys. 1-4 Przykładowe zapytanie w Apache Druid.....	4
rys. 1-5 Przykładowe zapytanie w MySQL .....	4
rys. 1-6 Przykładowa kwerenda PyDruid .....	5
rys. 1-7 Wynik przykładowej kwerendy PyDruid przekonwertowany na ramkę danych .....	5
rys. 1-8 Przykładowa kwerenda MySQL w SQLAlchemy .....	5
rys. 1-9 Wynik przykładowej kwerendy MySQL w SQLAlchemy przekonwertowany na ramkę danych .....	5
rys. 1-10 Analizowane dane, cz.1 .....	6
rys. 1-11 Analizowane dane, cz.2 .....	6
rys. 1-12 Analizowane dane - prawie 5 mln rekordów .....	7
rys. 2-1 Baza telecom_users w Apache Druid .....	7
rys. 2-2 Baza telecom_users w MySQL.....	7
rys. 2-3 Połączenie z bazą danych Apache Druid za pomocą biblioteki PyDruid.....	8
rys. 2-4 Połączenie z bazą danych MySQL za pomocą biblioteki SQLAlchemy.....	8
rys. 4-1 Wczytane dane w postaci ramki danych pandas.....	10
rys. 4-2 Metoda łokciowa.....	10
rys. 4-3 Klasteryzacja danych ze standaryzacją.....	11
rys. 4-4 Dokładności modelu SVM - wykres.....	12
rys. 4-5 Dokładności stworzonego modelu SVM .....	12

## 7. Spis tabel

Tabela 1 Porównanie czasów połączenia baz danych.....	8
Tabela 2 Porównanie czasu wykonania kwerend na mniejszej bazie danych .....	9
Tabela 3 Porównanie czasu wykonania kwerendy w terminalu na większej bazie danych ..	9
Tabela 4 Porównanie czasu wykonania kwerend z użyciem Pythona na większej bazie danych .....	9

## 8. 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
- [3] <https://pythonhosted.org/pydruid/> dostęp 20.03.2022
- [4] [https://scikit-learn.org/stable/user\\_guide.html](https://scikit-learn.org/stable/user_guide.html) dostęp 22.04.2022
- [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/>