UNIWERSYTET RZESZOWSKI
Kolegium Nauk Przyrodniczych

Paweł Mokrzycki

106462

Informatyka

Optymalizacja efektywności metod wykrywania nietypowych przypadków w dużych danych za pomocą narzędzia ClickHouse

Praca magisterska

Praca wykonana pod kierunkiem

Dr hab. Jana Bazana

Rzeszów, 2023

Coś się kończy, coś zaczyna… Serdeczne podziękowania dr hab. Janowi Bazanowi za pomoc w realizacji pracy. Rodzinie i przyjaciołom za wszelaką pomoc, wsparcie i motywacje w pięcioletnim okresie studiowania.

Spis treści

1 Wstęp

1.1 Cel pracy

Wraz z postępem cyfryzacji coraz bardziej na znaczeniu zyskują działania związane z jak najbardziej efektywnym analizowaniem i przetwarzaniem dużych zbiorów danych. Przy dzisiejszej technologii niewielkie dane jesteśmy w stanie analizować błyskawicznie. Problem pojawia się, gdy chcemy to robić, gdy jest ich dużo więcej. Dzisiaj nikogo już nie dziwią petabajty danych z miliardami rekordów. Przetwarzanie ich nie jest proste i wymaga zwiększenia nacisku na przyjrzenie się problemowi ich przetwarzania również pod kątem wydajnościowym. Zwłaszcza jeśli chcemy przetwarzać dane w czasie rzeczywistym.

Celem niniejszej pracy jest skupienie się na jednej z kluczowych części analizy dużych danych, czyli wykrywaniu nietypowych przypadków. Głównym założeniem pracy jest zbadanie aktualnego stanu wiedzy na temat metod, które to umożliwiają. Następnie zakłada się próbę przełożenia ich implementacji w taki sposób, aby można było je wykonać w narzędziu ClickHouse, dzięki któremu powinny działać bardziej wydajnie. Porównywane one będą z istniejącymi już implementacjami z wykorzystaniem języka Python. Jest to obecnie jeden z najpopularniejszych języków [1]. Jest on kojarzony z efektywną eksploracją danych. Zawiera wiele bibliotek z gotowymi implementacjami pod kątem eksploracji danych. Powinien więc być idealną bazą porównawczą.

1.2 Wprowadzenie

Wchodząc na oficjalną stronę projektu ClickHouse na samym wstępie ukazana jest informacja: „ClickHouse is the fastest and most resource efficient open-source database for real-time apps and analytics” [2]. Możemy to przetłumaczyć jako „ClickHouse to najszybsza i najbardziej zasobooszczędna baza danych typu open-source dla aplikacji i analiz w czasie rzeczywistym.” W skrócie jest to narzędzie, które potrafi błyskawicznie zwrócić rezultaty zapytań o dane. Celem jest jak najlepsze wykorzystanie właściwości tego narzędzia, aby spełnić postawione założenie stanowiące temat pracy.

W ostatnich latach obserwujemy niesamowity postęp cyfryzacji. Do świata cyfrowego przenosimy coraz więcej działalności. Duża część z nich wymaga gromadzenia i przetwarzania dużej ilości danych.

Przykładów, gdzie wykorzystywana jest analiza dużych danych jest mnóstwo. Wiele naukowych badań się na tym skupia. Choćby analiza danych w celu prognozowania zmian klimatycznych. Banki ciągle analizują dane finansowe w celu np. wykrycia oszustw. Dzisiejsza medycyna przetwarza multum danych w celu znalezienia kluczowych czynników powodujących określone choroby, czy wystawić prawidłową diagnozę. Wartościowe informację mogę prowadzić do innowacji, usprawnień, przełomowych odkryć.

Im więcej danych, tym coraz większa potrzeba sprawnego zarządzania nimi. Ciężko wyobrazić sobie ręczne przeszukiwanie milionowych zbiorów danych. Jednym z kluczowych aspektów analizy danych jest identyfikacja odstających wartości od pozostałych danych. Operując na dużych danych istnieję ryzyko, że jakieś przypadki zostały wprowadzone błędnie. Wykrycie ich może poprawić jakość danych. Czasem możemy chcieć się ich pozbyć, gdyż w znaczny sposób wpływają na końcowy rezultat analizy. Znalezienie jakiś odstających wzorców może również doprowadzić do kluczowych wniosków, pozyskać nową wiedzę. Pojawienie się nietypowych danych może oznaczać próbę oszustwa.

Jak widać zastosowań wykorzystania odnalezionych nietypowych przypadków jest wiele. Warto więc pochylić się nad możliwościami jak najbardziej efektywnego wyszukiwania ich.

1.3 Zakres prac

Zakres prac obejmuję przegląd literatury pod kątem znalezienia metod umożliwiających wykrywanie takich przypadków. Ważny jest też wybór takich metod, które będzie można efektywnie odwzorować. Temu etapowi poświęcony jest rozdział: Część teoretyczna.

Następnym krokiem jest przełożenie ich na autorską implementację, która pozwoli na wykorzystywanie metod w Clickhouse. Wymaga to analizy tego narzędzia i skupieniu się na wykorzystaniu go do stworzenia jak najszybszych w działaniu metod. Poświęcony temu został rozdział: Proponowane metody z użyciem ClickHouse. Zamieszczone tam zostaną także implementacje gotowych już rozwiązań. Jak zostało już wspomniane w celu pracy, wykorzystany zostanie do tego język Python. Rozdział zawiera również sposób przygotowania danych do części badawczej wraz z ich opisem.

Rozdział: Część eksperymentalna stanowi główną część pracy. Zostaną porównane w niej istniejące już rozwiązania z tymi autorskimi na dużych zbiorach danych. Analizie zostaną poddane czasy wykonania poszczególnych metod oraz końcowy rezultat w kontekście wykrywania nietypowych danych. Dodatkowo porównane zostaną wyniki w celu zweryfikowania poprawności autorskich implementacji.

Praca kończy się analizą i podsumowaniem otrzymanych wyników wraz z oceną w jaki sposób udało się spełnić założenia pracy.

2 Część teoretyczna

Niniejszy rozdział obejmuje przegląd literatury pod kątem metod wykrywania nietypowych przypadków wraz z opisem w jaki sposób tego dokonują. Zawiera również krótki opis narzędzia ClickHouse oraz przedstawia powody jego wyboru.

2.1 ClickHouse

Jednym z wyzwań przy przetwarzaniu dużych danych jest wybór odpowiednich narzędzi. Chcemy je przechowywać w taki sposób, aby mieć do nich możliwe jak najlepszy dostęp i móc je analizować w jak najszybszy sposób.

ClickHouse jest to nowoczesna baza danych. Udostępniona została po raz pierwszy w 2016 roku na licencji open-source. Jak zostało napisane we wstępie, autorzy projektu wskazują na błyskawiczne rezultaty zapytań analitycznych. Głównym celem, który przyświecał autorom było stworzenie narzędzia, które pozwoli błyskawicznie operować na dużych danych i pod tym kątem były implementowane główne mechaniki tego narzędzia.

ClickHouse jest bardzo wydajnym, zorientowanym na kolumny systemem zarządzania bazą danych SQL do przetwarzania analitycznego ogromnych zbiorów danych [3]. Podejście kolumnowe zostało zaprezentowane na rysunku 1. Zostało one oznaczone kolorem niebieskim. Zakłada pobieranie tylko potrzebnych danych z kolumn, które zostały określone w zapytaniu. Jest to bardziej wydajne podejście w porównaniu z podejściem wierszowym oznaczonym kolorem czerwonym. Dokumentacja wskazuję na ponad 100 razy szybszy czas wydobywanie danych.

Id	Imię	Nazwisko	Wiek
1	Jan	Kowalski	24
2	Carl	Johnson	45
3	Tytus	Bomba	34

Rysunek 1. Kolumnowy system zarządzania bazą danych

Szybkość tego narzędzia potwierdzają liczne testy wydajnościowe. Jeden z nich został przestawia rysunek 2. Widać na nim, że ClickHouse jest w stanie przeszukiwać dane błyskawicznie. Różnica czasowa w zapytaniach jest kilkukrotna w porównaniu do innej popularnej bazy PostgreSQL.

Q#	Query Contains Aggregates and Group By
Q1	SELECT DayOfWeek, count(*) AS c FROM ontime WHERE Year >= 2000 AND Year <= 2008 GROUP BY DayOfWeek ORDER BY c DESC;
Q2	SELECT DayOfWeek, count(*) AS c FROM ontime WHERE DepDelay>10 AND Year >= 2000 AND Year <= 2008 GROUP BY DayOfWeek ORDER BY c DESC;
Q3	SELECT Origin, count(*) AS c FROM ontime WHERE DepDelay>10 AND Year >= 2000 AND Year <= 2008 GROUP BY Origin ORDER BY c DESC LIMIT 10;
Q4	SELECT Carrier, count(*) FROM ontime WHERE DepDelay>10 AND Year = 2007 GROUP BY Carrier ORDER BY count(*) DESC;
Q5	SELECT a.Carrier, c, c2, c*1000/c2 as c3 FROM (SELECT Carrier, count(*) AS c FROM ontime WHERE DepDelay>10 AND Year=2007 GROUP BY Carrier) a INNER JOIN (SELECT Carrier,count(*) AS c2 FROM ontime WHERE Year=2007 GROUP BY Carrier)b on a.Carrier=b.Carrier ORDER BY c3 DESC;
Q6	SELECT a.Carrier, c, c2, c*1000/c2 as c3 FROM (SELECT Carrier, count(*) AS c FROM ontime WHERE DepDelay>10 AND Year >= 2000 AND Year <= 2008 GROUP BY Carrier) a INNER JOIN (SELECT Carrier, count(*) AS c2 FROM ontime WHERE Year >= 2000 AND Year <= 2008 GROUP BY Carrier) b on a.Carrier=b.Carrier ORDER BY c3 DESC;
Q7	SELECT Carrier, avg(DepDelay) * 1000 AS c3 FROM ontime WHERE Year >= 2000 AND Year <= 2008 GROUP BY Carrier;
Q8	SELECT Year, avg(DepDelay) FROM ontime GROUP BY Year;
Q9	select Year, count(*) as c1 from ontime group by Year;
Q10	SELECT avg(cnt) FROM (SELECT Year,Month,count(*) AS cnt FROM ontime WHERE DepDel15=1 GROUP BY Year,Month) a;
Q11	select avg(c1) from (select Year,Month,count(*) as c1 from ontime group by Year,Month) a;
Q12	SELECT OriginCityName, DestCityName, count(*) AS c FROM ontime GROUP BY OriginCityName, DestCityName ORDER BY c DESC LIMIT 10;
Q13	SELECT OriginCityName, count(*) AS c FROM ontime GROUP BY OriginCityName ORDER BY c DESC LIMIT 10;
	Query Contains Joins
Q14	SELECT a.Year, c1/c2 FROM (select Year, count(*)*1000 as c1 from ontime WHERE DepDelay>10 GROUP BY Year) a INNER JOIN (select Year, count(*) as c2 from ontime GROUP BY Year) b on a.Year=b.Year ORDER BY a.Year;
Q15	SELECT a."Year", c1/c2 FROM (select "Year", count(*)*1000 as c1 FROM fontime WHERE "DepDelay">10 GROUP BY "Year") a INNER JOIN (select "Year", count(*) as c2 FROM fontime GROUP BY "Year") b on a."Year"=b."Year";

Q#	PostgreSQL	PostgreSQL (Indexed)	ClickHouse
Q1	27920	19634	23
Q2	35124	17301	50
Q3	34046	15618	67
Q4	31632	7667	25
Q5	47220	8976	27
Q6	58233	24368	55
Q7	30566	13256	52
Q8	38309	60511	112
Q9	20674	37979	31
Q10	34990	20102	56
Q11	30489	51658	37
Q12	39357	33742	186
Q13	29912	30709	101
Q14	54126	39913	124
Q15	97258	30211	245

Rysunek 2. Test wydajnościowy bazy ClickHouse, źródło: https://www.percona.com/blog/benchmark-clickhouse-database-and-clickhousedb_fdw, data dostępu: 13.07.2023

Podsumowując powyższe informacje, jest to narzędzie stworzone do pracowania z dużymi danymi. Wszelkie informacje na temat tej bazy wskazują, że powinno to być narzędzie idealne do stworzenia metod, które błyskawicznie będą w stanie znaleźć nietypowe przypadki.

2.2 Z-score

2.2.1 Teoria

Z-score został zaproponowany jako składowa modułu odpowiedzialnego za detekcję anomalii w książce „Intelligent Data Mining and Analysis in Power and Energy Systems: Models and Applications for Smarter Efficient Power Systems" [4]. Porusza ona głównie temat eksploracji danych i inteligentnych modeli analizy danych w odniesieniu do systemów zasilania i energii. Jak napisali autorzy: „Oprogramowanie w centrum kontroli pomaga w monitorowaniu systemu energetycznego przy użyciu pomiarów w czasie rzeczywistym z różnych czujników, takich jak inteligentne urządzenia elektroniczne (IED), zdalne jednostki terminalowe (RTU) i inne urządzenia pomiarowe i sensory w sieci. Aplikacje wykorzystujące te pomiary zapewniają wsparcie decyzyjne dla działań korygujących mających na celu niezawodne działanie systemu energetycznego. Jednakże, jeśli pomiary są błędne, aplikacje mogą dostarczać fałszywe wyniki, wpływając na regularną pracę sieci. Dynamiczne zachowanie sieci energetycznej (np. zmiany obciążenia, generacji, rozproszonych źródeł energii (DER), przełączania, sieci, sterowania) powoduje zmienność definicji anomalii danych w czasie i wymaga oceny w czasie."

Kolejnym materiałem, który wskazuję ten sposób wykrywania tych przypadków jest publikacja: „Simple Anomaly Detection Using Plain SQL" [5]. Przedstawia on implementacje metody w sql-u do wykrycia anomalii w ruchu aplikacji webowej i wykrycia potencjalnych ataków DOS, czy wykrycia błędów w logice biznesowej.

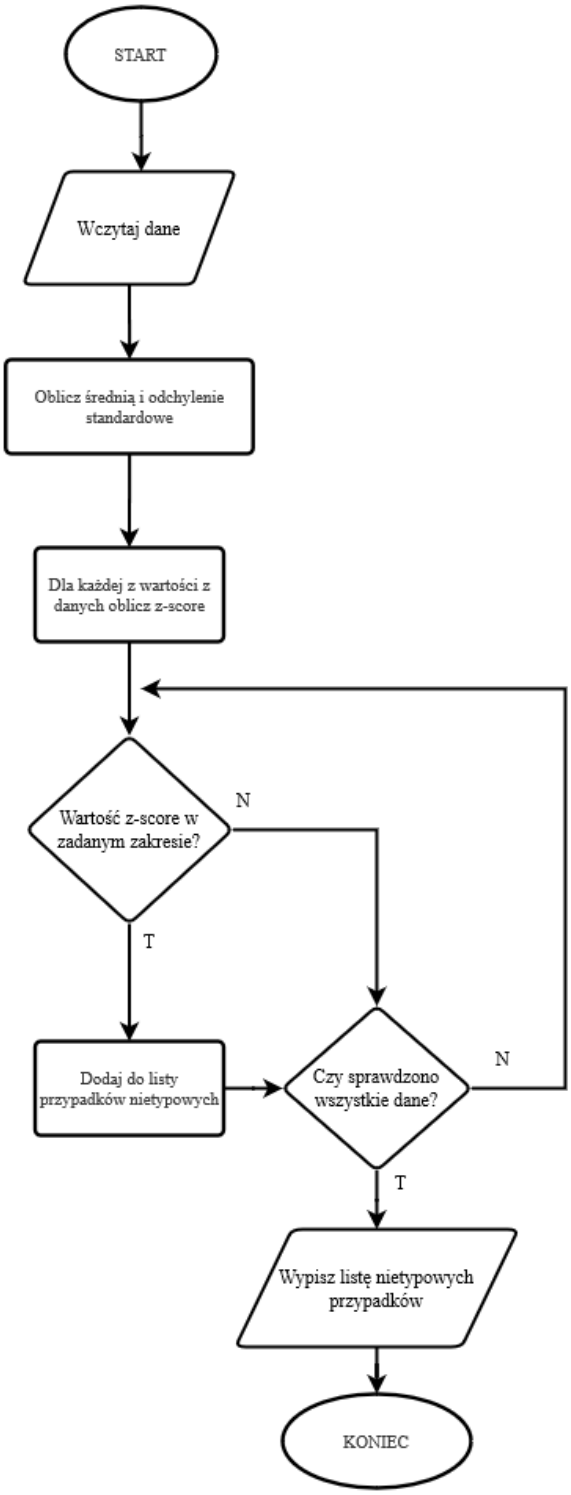
Wartość z-score oznacza miarę, która pokazuję, jak bardzo dana wartość różni się od średniej [6]. Im bardziej wartość jest większa bądź mniejsza od zera, tym dalej od niej się znajduję .Wzór do obliczenia z-score wygląda następująco:

$$z = (X - \mu) / \sigma$$

, gdzie:

- z – wartość z-score
- X – wartość, dla której obliczamy z-score
- μ – średnia
- σ – odchylenie standardowe

Wykorzystując powyższą wiedzę i mając obliczone wartości z-score dla każdej z danych, możemy ustalić dwie wartości wyznaczające zakres powyżej i poniżej którego możemy uznać, że przypadek może być wartością nietypową. Realizację tej metody przedstawia Rysunek 3.



Rysunek 3. Z-score - schemat blokowy

2.2.2 Przykład

Przyjmując zbiór danych: [11, 9, 13, 31, 9, 1]

Średnia: 12.33

Odchylenie standardowe: 9.14

Otrzymamy następujące wyniki z-score.

Liczba	Z-score
11	-0.15
9	-0.36
13	0.07
31	2.04
9	-0.36
1	-1.24

Tabela 1. Wartości z-score dla przykładowych danych

Dla powyższych wyników ustalamy próg anomalii na wartości powyżej 1 i poniżej -1. Warunek ten spełniają liczby: 31 i 1. Jak można zauważyć są to wartości, które faktycznie odstają od pozostałych.

Wykorzystując tę metodę trzeba jednak zwrócić uwagę na to, że w pewnych przypadkach może dawać błędne wyniki lub w ogóle nie wykrywać anomalii. Bardzo wiele będzie zależeć od sposobu rozkładu danych, dla których jest wykonywana. Metoda operuje na średniej oraz odchyleniu standardowym. Wobec tego może być wrażliwa na pojedyncze, skrajne wartości w zbiorze danych. Jeśli dane zawierają odosobnione wartości odstające, mogą one spowodować przesunięcie średniej i odchylenia standardowego, co wpłynie na wyniki końcowe. W części badawczej zostanie zasymulowany taki przypadek. Najlepszy rezultat będzie dawać dla danych o w miarę znormalizowanym rozkładzie danych.

Metoda ta operuję na pojedynczej kolumnie danych oraz wykorzystuje proste analityczne operacje. Stąd zdecydowałem się ją wybrać, gdyż idealnie wpasowuję się w założenia narzędzia ClickHouse.

2.3 IQR

2.3.1 Teoria

IQR jako sposób na znalezienie wartości odstających został przedstawiony w książce SQL for Data Analysis [7]. Pozycja ta skupia się wokół możliwości języka sql w badaniu i analizowaniu danych. W kontekście pracy niezwykle interesujący jest rozdział dotyczący wykrywania anomalii. Między innymi został tam przedstawiony sposób implementacji wykorzystania metody IQR za pomocą języka SQL do detekcji odstających wyników magnitudy trzęsień ziemi w Japonii. Prezentuję go rysunek 4.

```
SELECT ntile_25, median, ntile_75
,(ntile_75 - ntile_25) * 1.5 as iqr
,ntile_25 - (ntile_75 - ntile_25) * 1.5 as lower_whisker
,ntile_75 + (ntile_75 - ntile_25) * 1.5 as upper_whisker
FROM
(

     SELECT
     percentile_cont(0.25) within group (order by mag) as ntile_25
     ,percentile_cont(0.5) within group (order by mag) as median
     ,percentile_cont(0.75) within group (order by mag) as ntile_75
     FROM earthquakes
     WHERE place like '%Japan%'
 ) a
 ;

 ntile_25  median  ntile_75  iqr   lower_whisker  upper_whisker
 --------  ------  --------  ----  -------------  -------------
 4.3       4.5     4.7       0.60  3.70           5.30
```

Rysunek 4. Implementacja IQR w SQL. Źródło: Książka "SQL for Data Analysis", Cathy Tanimura

IQR (ang. Interquartile Range) możemy tłumaczyć jako rozstęp kwartylowy. W statystyce kwartyl dzieli liczbę punktów na cztery mniej więcej równe części. IQR jest to różnica między trzecim kwartylem, a pierwszym [8]. Wzór przedstawia się następująco:

$$\mathrm{IQR} = \mathrm{Q3} - \mathrm{Q1}$$

Aby obliczyć IQR należy wykonać następujące czynności

1) Uporządkuj zbiór danych od najmniejszej do największej wartości.
2) Oblicz pierwszy kwartyl (Q1), który jest medianą dolnej połowy danych.
3) Oblicz trzeci kwartyl (Q3), który jest medianą górnej połowy danych.
4) IQR jest różnicą między trzecim a pierwszym kwartylem: IQR = Q3 - Q1.

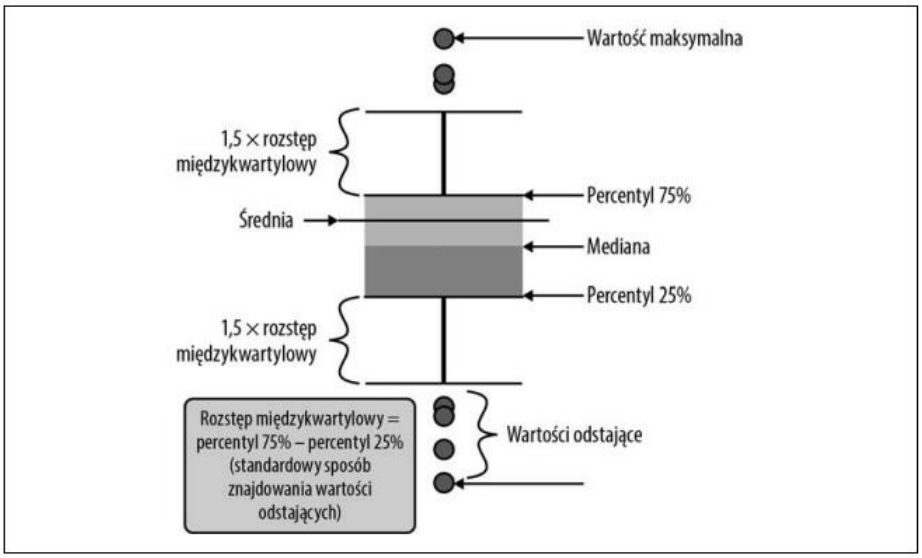
Mając wartość IQR, Q3, Q1 możemy wyszukać przypadki odstające, które znajdują się poza zdefiniowanym przez nas zakresem. Wymieniona wyżej pozycja oraz inne źródła [9], przyjmują za współczynnik wartość 1.5 (wartość sugerowana,

gdyż 1.5 * IQR jest zbliżoną wartością do 3 odchyleń standardowych powyżej średniej), dla którego określamy dolną i górną granicę odstępstwa:

- Q1 – 1.5 * IQR – Dolna granica odstępstwa.
- Q3 + 1.5 * IQR – Górna granica odstępstwa.

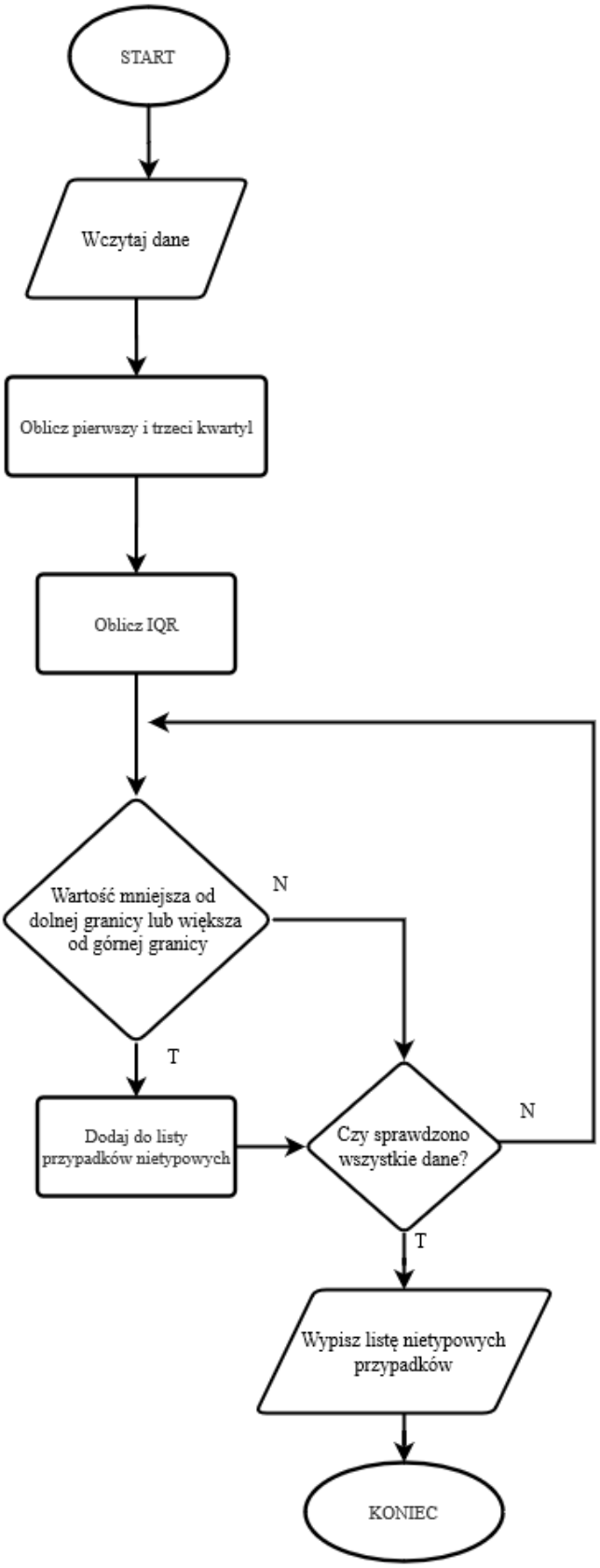
Wartości spoza tych granic mogę być przypadkami odstającymi i warto się im przyjrzeć. Oczywiście nic nie stoi na przeszkodzie w doborze innego współczynnika.

Metodę tę dobrze ilustruje poniższy rysunek.



Rysunek 5. IQR jako metoda do wykrywanie nietypowych przypadków, Książka "SQL for Data Analysis", Cathy Tanimura

Rysunek 6 przedstawiona schemat blokowy realizujący wykrywanie nietypowych przypadków metodą iqr.



Rysunek 6. IQR - schemat blokowy

2.3.2 Przykład

Przyjmując zbiór danych: [11, 9, 13, 31, 9, 1, 12]

1) Sortowanie: 1, 9, 9 ,11, 12, 13, 31
2) Q1 = 9
3) Q3 = 13
4) IQR = 4
5) Dolny zakres: 9 – 1.5 * 4 = 3
6) Górny zakres: 13 + 1.5 * 4 = 19

Poniżej wartości 3 i powyżej wartości 19 znajdują się dwie wartości: 1 i 31. Jak widać obie są danymi odstającymi i metoda poprawnie je zidentyfikowała. Można zauważyć, że podobnie jak z-score będzie bardzo efektywna dla danych zbliżonych do rozkładu normalnego. Dla takich danych otrzymamy przypadki, które znajdują się poza wyznaczonym przez nas progiem. Może jednak mieć problemy w przypadku danych o nietypowym rozkładzie danych.

Metoda ta również jak z-score operuje na pojedynczej kolumnie. Operacje, które wykonuje również powinny w narzędziu ClickHouse działać błyskawicznie. Stąd wybór tej metody.

2.4 K-means

2.4.1 Teoria

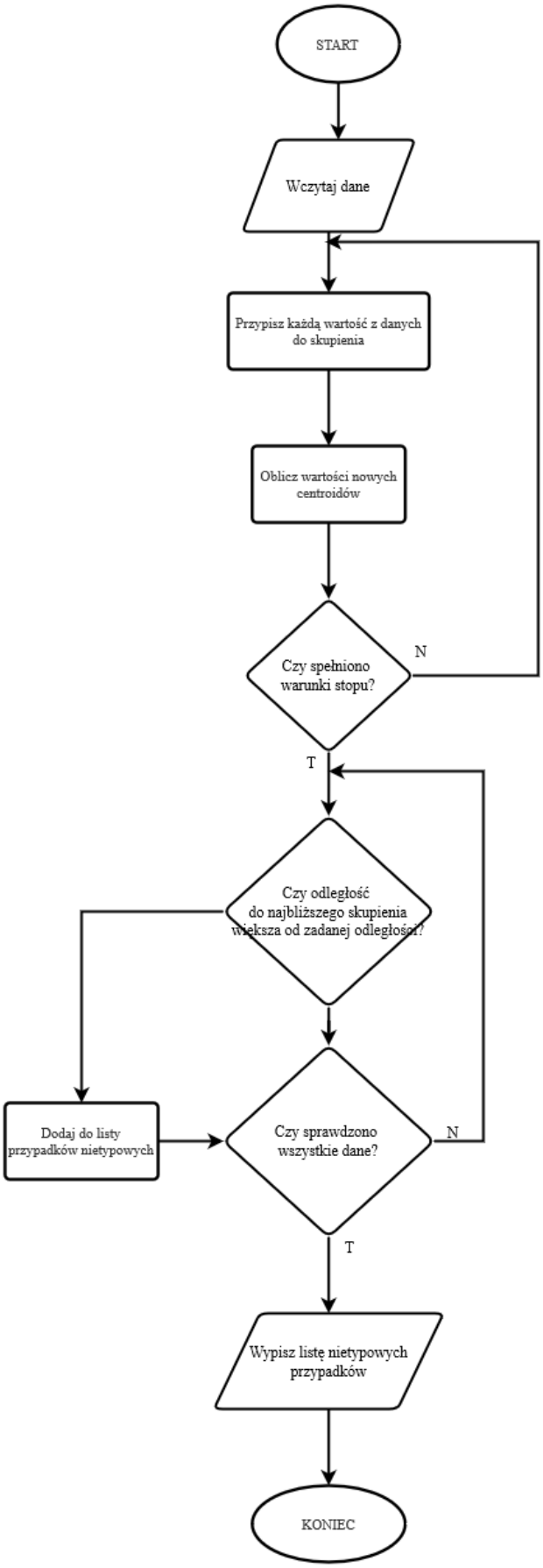
Na użyciu metody k-means do detekcji przypadków odstających opiera się publikacja: „A Data-Driven Heart Disease Prediction Model Through K-Means Clustering-Based Anomaly Detection” [10]. Jak wskazuję nazwa tego dokumentu wykorzystuję grupowanie k-mans do detekcji anomalii, co następnie przysłużyło się do opracowania badania przewidywania chorób serca. K-means jest metodą grupowania, mającą za zadanie podział danych na założoną na wejściu liczbę skupień [11]. Istnieją też publikacje opisujące metodykę sposobu implementacji k-means w języku sql [12] [13]. Artykuły te w znaczy sposób pomogły w implementacji tej metody w ClickHouse.

W pracy k-means będzie operował na dwuwymiarowych danych. Mając takie punkty możemy umieścić je w przestrzeni dwuwymiarowej. Algorytm k-means składa się z następujących kroków.

1) Inicjalizacja: Wybierz liczbę skupień "k" oraz początkowe punkty centralne (centroidy) losowo lub na podstawie innych kryteriów. Każdy punkt danych zostaje przypisany do jednego ze skupień, do którego ma najmniejszy dystans.
2) Kroki iteracyjne:
    - Przypisanie do punktów skupień: Dla każdego punktu danych, oblicz odległość od wszystkich centroidów i przypisz punkt do skupienia, którego centroid jest najbliższy. W przypadku pracy wykorzystana zostanie metryka euklidesowa, która umożliwia zwrócenie odległości między punktami w przestrzeni dwuwymiarowej.
    - Dla każdego skupienia, oblicz nowy centroid, który jest średnią arytmetyczną punktów przypisanych do tego skupienia.
3) Warunki stopu: Algorytm k-średnich kontynuuje iteracje przypisania i aktualizacji, dopóki nie zostaną spełnione pewne warunki stopu, na przykład:
    - Żaden punkt nie zmienia skupienia (przypisania punktów nie zmieniają się.
    - Maksymalna liczba iteracji została osiągnięta.
    - Minimalna zmiana w centroidach jest mniejsza niż określona wartość.
4) Zakończenie: Po zakończeniu iteracji lub spełnieniu warunku stopu, wynikiem algorytmu k-średnich są ostateczne skupienia oraz ich centroidy.

Założenia k-means można wykorzystać do detekcji przypadków odstających. Mając dane zebrane w skupienia oraz dane na temat odległości do najbliższego skupienia możemy odfiltrować według tego, czy odległość znajduję się dalej od wybranej przez nas wartości.

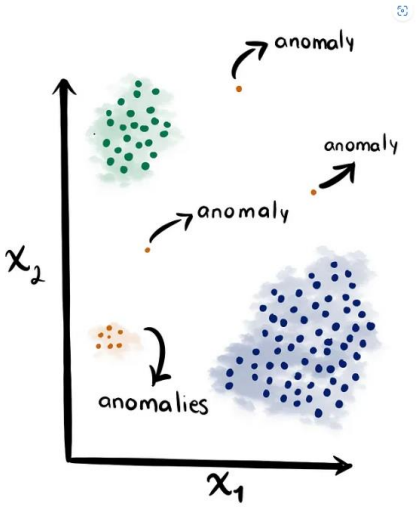
Na rysunku 7 przedstawiony został schemat blokowy realizujący wykrywanie nietypowych przypadków metodą k-means.



Rysunek 7. K-means - schemat blokowy

2.4.2 Przykład

Na rysunku 8 zostało przedstawione grupowanie danych metodą k-means. Jak widać nastąpił podział na dwa skupienia. Centroidów możemy spodziewać się pośrodku grupy danych oznaczonej kolorem zielonym i niebieskim. Przypadki oznaczone kolorem pomarańczowym będą miały znaczną odległość do tych centroidów. Możemy uznać je więc za nietypowe przypadki.



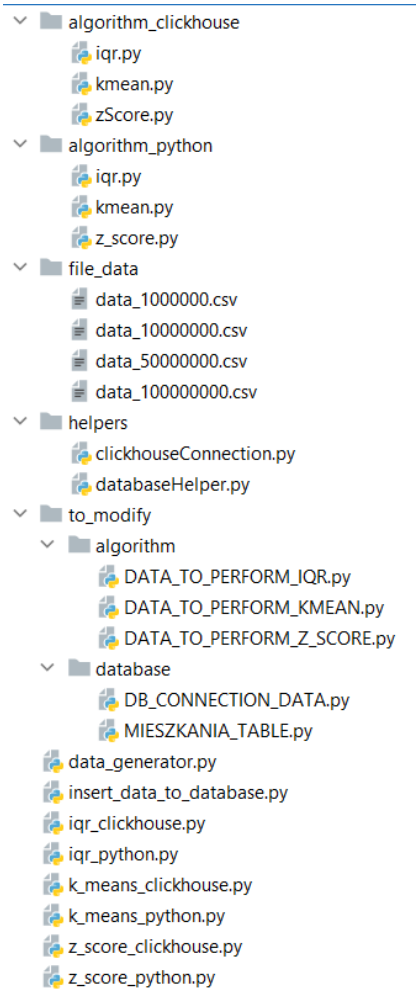
Rysunek 8. K-means jako metoda do wykrywania nietypowych przypadków, źródło: https://towardsdatascience.com/unsupervised-anomaly-detection-on-spotify-data-k-means-vs-local-outlier-factor-f96ae783d7a7, Data dostępu: 15.07.2023

3 Proponowane metody z użyciem ClickHouse

Poniższy rozdział przedstawia realizację logiki algorytmów przedstawionych w rozdziale z częścią teoretyczną w narzędziu ClickHouse. Przedstawia również sposób ich implementacji w języku Python z wykorzystaniem istniejących już rozwiązań, które w części eksperymentalnej pozwolą na porównanie efektywności jak i poprawności autorskich rozwiązań. Została też dokonana przybliżona (brak dokładnych wartości w dokumentacji) analiza złożoności czasowej, wskazująca miejsca, w których spodziewane jest zwiększenie efektywności dzięki narzędziu ClickHouse, co następnie zostanie zweryfikowane w części eksperymentalnej. Zamieszczony też został sposób generowania danych do części eksperymentalnej wraz z ich opisem.

3.1 Struktura plików projektu

Rysunek 9 przedstawia strukturę projektową wykonanych skryptów. Następnie po krótce została ona opisana.



Rysunek 9. Struktura plików projektu

- algorith_clickhouse: Folder ze skryptami implementującymi metody wykrywania nietypowych przypadków w narzędziu clickHouse
- algorith_python: Jak wyżej, tylko implementację w języku Python zamiast ClickHouse.
- file_data – Folder z wygenerowanymi pliki z danymi, na których przeprowadzana będzie część badawcza.
- helpers – Folder zawierający skrypty umożliwiający łączenie się z bazą danych oraz tworzenie tabeli.
- to_modify – Folder do modyfikacji. Korzystają z niego skrypty wykonywalne. W plikach tych zawarte są dane, które następnie są wykorzystywane przez metody wykrywania odstających danych. Domyślne wartości zostały przedstawione w podrozdziałach ze szczegółami implementacji.
- folder główny - skrypty wykonywalne – skrypty przeznaczone do wykonania. Nazwa pliku wskazuję ich przeznaczenie. Prosty kod wywołujący metody z pozostałych folderów implementujących logikę.

3.2 Przygotowanie danych do części badawczej

Praca polega na badaniu jak sprawdzą się metody służące do wykrywania nietypowych przypadków w dużych zbiorach danych. Wobec tego zdecydowałem się napisać prosty skrypt przedstawiony na rysunku 10, który pozwala wygenerować wskazaną liczbę danych. Tak wygenerowane dane posłużą mi do przeprowadzenia części badawczej.

```python
def generate_data_file():
    with open(new_file_name, 'w', newline='') as new_csv:
        header = ['id', 'miasto', 'cena', 'liczba_metrow', 'cena_m']
        data = generate_random_data(num_rows_to_generate)
        # Dodanie odstajacych przypadków, celem jest ich wykrycie
        data.append(["Krakow", 600000, 50, 12000])  # anomalia: liczba_metrow
        data.append(["Krakow", 240000, 20, 12000])  # anomalia: liczba_metrow
        data.append(["Krakow", 960000, 80, 12000])  # anomalia: liczba_metrow
        data.append(["Krakow", 9018000, 68, 13500])  # drobna anomalia: cena_m
        data.append(["Krakow", 560000, 35, 16000])  # wieksza anomalia: cena_m
        random.shuffle(data)  # losowe przemieszanie danych
        # Przypisanie id
        ids = list(range(1, len(data) + 1))
        for row, id_value in zip(data, ids):
            row.insert(0, id_value)
        writer = csv.writer(new_csv)
        writer.writerow(header)
        writer.writerows(data)


def generate_random_data(num_rows_to_generate: int):
    data = []
    for _ in range(num_rows_to_generate):
        miasto = "Warszawa" if random.random() > 0.9 else "Krakow"
        case = random.randint(1, 2)
        if case == 1:
            cena = 396000 + random.randint(-1000, 1000)
            liczba_metrow = round(33.0 + random.uniform(-2, 2), 1)
        else:
            cena = 780000 + random.randint(-1000, 1000)
            liczba_metrow = round(65.0 + random.uniform(-2, 2), 1)
        cena_m = int(cena // liczba_metrow)
        data.append([miasto, cena, liczba_metrow, cena_m])
    return data


num_rows_to_generate = 1000000
new_file_name = 'file_data/data_1000000.csv'
generate_data_file()
```

Rysunek 10. Skrypt generujący dane

Skrypt generuję plik o podanej nazwie ze wskazaną liczbą rekordów, do których dokłada pięć zdefiniowanych anomalii. Funkcja generate_random_data() tworzy losowe dane, jednak w ograniczonym zakresie. Wskutek tego działania wynikowy zbiór tych danych nie będzie zawierał przypadków odstających.

Schemat generowania losowych danych został przedstawiony poniżej:

- miasto – Losowo. Jedna wartość z dwóch, wraz ze zdefiniowanym procentem szans: Kraków (95%), Warszawa (5%).
- cena i liczba_metrow – Losowo (równe szanse) na wartości z dwóch przedziałów:

- I przedział: Cena z zakresu <395000, 397000> zaokrąglona do liczby całkowitej. Liczba metrów z zakresu <31.00, 35.00>, zaokrąglona do dwóch miejsc po przecinku.
- II przedział: Cena z zakresu <779000, 781000>. Liczba metrów z zakresu <63.00, 67.00>, zaokrąglona do dwóch miejsc po przecinku.

- cena_m – Wartość zaokrąglona do liczby całkowitej poprzez podzielenie ceny przez liczbę metrów. Jest to wartość zależna od powyżej wymienionych wartości ceny i liczby metrów. Będzie osiągać wartość z dwóch przedziałów: <11285, 12806> dla danych z przedziału pierwszego lub <11626 ,12396> dla danych z przedziału drugiego. Łącząc te przedziały uzyskamy końcową wartość z przedziału: <11285, 12806>.

Do tak wygenerowanego losowego zestawu danych zostało dołożonych pięć przypadków odstających. Następnie dane zostały przemieszane i zostały do nich dołożone kolejne numery id. Przykładowa struktura pliku z dodanymi dziesięcioma rekordami i pięcioma anomaliami została przedstawiona na rysunku 11. Kolorem czerwonym zostały zaznaczone dodane nietypowe przypadki. Pozostałe dane zostały wygenerowane losowo zgodnie z wcześniejszym schematem.

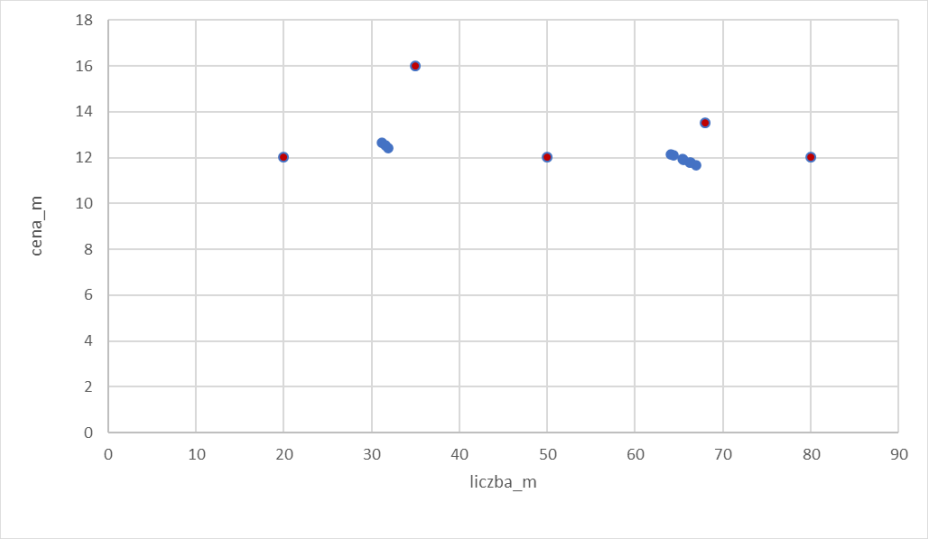
```
id,miasto,cena,liczba_metrow,cena_m
1,Krakow,600000,50,12000
2,Krakow,395075,31.2,12662
3,Krakow,779203,64.1,12156
4,Krakow,780503,66.2,11790
5,Krakow,395732,31.9,12405
6,Krakow,779187,64.4,12099
7,Krakow,779608,65.5,11902
8,Krakow,240000,20,12000
9,Krakow,560000,35,16000
10,Krakow,780300,66.9,11663
11,Krakow,780593,65.4,11935
12,Warszawa,780865,66.3,11777
13,Krakow,960000,80,12000
14,Krakow,396240,31.6,12539
15,Krakow,9018000,68,13500
```

Rysunek 11. Przykładowy plik z wygenerowanymi danymi

Część badawcza będzie skupiona wokół dwóch kolumn. Liczba_metrow jako liczba całkowita i cena_m jako liczba dziesiętna. Dodatkowo zdecydowałem się wprowadzić kolumnę miasto. Pozwoli to dodatkowo na wykonanie operacji

filtrowania w części badawczej i sprawdzenia jak taka operacja wpływa na czasy wykonania. Prawdopodobieństwo Krakowa wynoszące 95% nadal pozwoli zachować duży zbiór danych.

Wizualizacja tych dwóch kolumn na przestrzeni dwuwymiarowej została przedstawiona na rysunku 12. Dla czytelności kolumna cena_m została podzielona przez tysiąc.



Rysunek 12. Rozkład wygenerowanych danych

W kolorze niebieskim zostały zaprezentowane losowo generowane dane. Jak widać zgodnie z założeniami będą one znajdować się blisko siebie. Ceny za metr są bardzo podobne dla wszystkich losowych przypadków. Rozbieżność pojawia się w liczbie_metrow, co dzieli dane na dwie grupy. Zgodnie z teorią przypadek ten pozwoli sprawdzić jak z takim rozkładem poradzą sobie metody z-score oraz iqr, które w założeniach nie są przystosowane do takiego rozkładu danych. Pozwoli to też na grupowanie danych metodą k-means. Kolorem czerwonym zostały zaznaczone anomalie. Kolejno od lewej:

- Stosunkowo duża anomalia ze względu na liczbę metrów (20.00).
- Stosunkowo duża anomalia ze względu na cenę za metr (1600).
- Anomalia ze względu na liczbę metrów (50.00). Wartość znajduję się pośrodku dwóch grup. Może być to trudniejszy przypadek do znalezienia dla algorytmów.
- Drobna anomalia. Lekko umieszczona poza maksymalnymi zakresami losowania danych (68.00, 13500).

- Stosunkowa duża anomalia ze względu na liczbę metrów (80.00).

Powyższy skrypt został wykorzystany do utworzenia czterech zbiorów liczących kolejno rekordów w milionach: 1, 10, 50, 100.

Skrypt insert_data_to_database.py pozwala na wstawienie danych do narzędzia ClickHouse. Plik MIESZKANIA_TABLE pozwala wybrać lokalizację pliku z danymi. Określa również nazwę tabeli do jakiej zostaną wprowadzone dane. Zawiera także schemat tej tabeli.

3.3 Z-score

Plik z_score_python.py jest skryptem uruchomieniowym wypisujący nietypowe przypadki zwrócone poprzez metodę zScore_outlier, którą przedstawia rysunek 13.

```python
import time

import numpy as np
import pandas as pd
from scipy.stats import zscore
from to_modify.algorithm import DATA_TO_PERFORM_Z_SCORE as Z_SCORE_DATA

def zScore_outlier():
    filename = Z_SCORE_DATA.FILE_LOCATION  # plik z danymi
    column_name = Z_SCORE_DATA.COLUMN_NAME  # kolumna dla której wykonać metodę
    # Poniżej Zakresy dla wykrycia przypadków nietypowych
    zScoreResultMinBetweenValue = Z_SCORE_DATA.ZSCORE_RESULT_MIN_BETWEEN_VALUE
    zScoreResultMaxBetweenValue = Z_SCORE_DATA.ZSCORE_RESULT_MAX_BETWEEN_VALUE
    print('Rozpoczęto wykonywanie funkcji zScore')
    start_time = time.time()  # Rozpoczęcie odliczania czasy
    data = pd.read_csv(filename)  # Import danych z pliku
    z_scores = zscore(data[column_name])  # Obliczenie z-score dla każdej wartości
    # Poniżej indetyfikacja przypadków odstających
    outlier_indices = np.where((z_scores > zScoreResultMaxBetweenValue) | (z_scores < zScoreResultMinBetweenValue))
    outliers_case = [(i + 1, data.iloc[i][column_name], z_scores[i]) for i in outlier_indices[0]]
    run_time = time.time() - start_time  # Zakończenie mierzenia czasu
    print(f'Zakończono wykonywanie funkcji zScore w czasie {run_time} sekund.')
    return outliers_case  # Zwrócenie odstających przypadków
```

Rysunek 13. Implementacja z-score - Python

Powyższy skrypt wykorzystuje metodę z-score z biblioteki scipy [14]. Po jej zastosowaniu następuję filtrowanie danych w celu pozostawiania wyłącznie przypadków odstających. Następnie funkcja zwraca listę nietypowych przypadków.

Opierając się na implementacji w Python oraz części teoretycznej utworzyłem skrypt realizujący metodę z-score w narzędziu ClickHouse. Implementacja została przedstawiona na rysunku 14.

```python
import time

from clickhouse_connect.driver import Client


def zScore(client: Client, colName: str, tableName: str, zScoreFrom: int, zScoreTo: int):
    print('Rozpoczęto wykonywanie funkcji zScore')
    start_time = time.time()
    ZSCORE_QUERY = 'WITH series AS (SELECT id, {} AS data_set from {}), stats AS (SELECT ' \
                   'avg(data_set) series_avg, stddevSamp(data_set) as series_stddev FROM series ), ' \
                   'zscores AS (SELECT id, data_set, (data_set - series_avg) / series_stddev AS zscore ' \
                   'FROM series, stats ) SELECT * FROM zscores where zscore NOT BETWEEN {} AND {}' \
        .format(colName, tableName, zScoreFrom, zScoreTo)
    result = client.query(ZSCORE_QUERY)
    run_time = time.time() - start_time
    print('Zakończono wykonywanie funkcji zScore w czasie %s sekund.' % run_time)
    return result.result_set
```

Rysunek 14. Implementacja z-score - ClickHouse

Metoda ta generuje dynamicznie SQL na podstawie przekazanych danych w skrypcie wykonywalnym z_score_clickhouse.py. Czyni to metodę uniwersalną, mogącą przetwarzać wskazane dane bez potrzeby modyfikacji kodu SQL. Uruchamiając skrypt z domyślnie przypisanymi danymi z pliku DATA_TO_PERFORM_Z_SCORE.py, zostanie wygenerowany SQL przedstawiony na rysunku 15.

```sql
WITH series AS (SELECT id, liczba_metrow AS data_set from MIESZKANIA),
     stats AS (SELECT avg(data_set) series_avg,stddevSamp(data_set) as series_stddev FROM series),
     zscores AS (SELECT id, data_set, (data_set - series_avg) / series_stddev AS zscore FROM series,stats)
SELECT * FROM zscores where zscore NOT BETWEEN -3 AND 3
```

Rysunek 15. Z-score - przykład SQL dla ClickHouse

W powyższym SQL-u można wyróżnić cztery etapy.

1) With series AS… – Utworzenie zbioru z danymi, na których metoda będzię operować. Kolumna id pozwala zidentyfikować przypadek w zbiorze. Kolumna liczba_metrow jest kolumną dla której będzie wykonywana metoda z-score. Został do niej dodany alias data_set. Do tego aliasu będą odwoływać się kolejne kroki, co czyni SQL bardziej uniwersalnym. Wystarczy podać nazwę kolumny w jednym miejscu.
2) stats AS… – Obliczenie średniej oraz odchylenia standardowego zbioru danych data_set z pierwszego kroku. Wartości te posłużą do obliczenia z-score w kroku 3.
3) zscore AS… – zawiera id, wartość ze zbioru danych data_set z pierwszego kroku, rezultat z-score dla wartości.
4) SELECT * FROM zscores where… - Zwrócenie wszystkich wartości z kroku 3 spełniających warunki filtru where. W przedstawionym przypadku zwraca

wartości z wynikiem z-score większym od 3 lub mniejszym od -3 wraz z numerem id.

Domyślne dane umieszczone w pliku DATA_TO_PERFORM_Z_SCORE.py dla jakich ma zostać wykonana metoda przedstawia rysunek 16. Wartości zostały stosownie pokomentowane, zgodnie z ich przeznaczeniem.

```
# Nazwa tabeli (dla z_score_clickhouse) (+ opcjonalnie klauzula np. WHERE)
TABLE_NAME = "MIESZKANIA"
# TABLE_NAME = "MIESZKANIA WHERE miasto = 'Krakow'"
# Ścieżka do pliku (dla z_score_python)
FILE_LOCATION = 'file_data/data_10000000.csv'
# Nazwa kolumny dla której zostanie wykonane zSore
COLUMN_NAME = 'cena_m'
# COLUMN_NAME = 'liczba_metrow'
# Pokaż w końcowym wyniku rezultaty zScore mniejsze od zadanej wartości
ZSCORE_RESULT_MIN_BETWEEN_VALUE = -3
# Pokaż w końcowym wyniku rezultaty zScore większe od zadanej wartości
ZSCORE_RESULT_MAX_BETWEEN_VALUE = 3
```

Rysunek 16. Domyślne dane dla wykonania metody z-score

Opierając się na przedstawionej implementacji poniżej została dokonana analiza złożoności czasowej skryptu Python.

1) Obliczenie z-score – Funkcja zscore z pakiety scipy. Wymaga obliczenia wartości średniej oraz odchylenia standardowego. Operacje te pod względem czasowym są rzędu O(N), gdzie N oznacza ilość danych.
2) Identyfikacja wartości odstających. Weryfikacja czy wartość z-score danej liczby mieści się w zadanym zakresie. Wymaga przejrzenia całości zbioru danych, co charakteryzuję się rzędem złożoności O(N), gdzie N oznacza ilość danych.

W implementacji Python poszczególne kroki metody z-score charakteryzują się liniową złożonością. W narzędziu ClickHouse obliczenie średniej i odchylenia standardowego powinno być bardzo szybkie, podobnie jak identyfikacja przypadków odstających. Spodziewana jest natychmiastowa odpowiedź w tym narzędziu dla dowolnej ilości danych. W skrypcie Python spodziewany jest wzrost czasowy wraz ze zwiększeniem rozmiaru danych.

3.4 IQR

Plik iqr_python.py jest skryptem uruchomieniowy wypisujący nietypowe przypadki z wykorzystaniem metody iqr_outlier. Przedstawia go rysunek 17.

```python
import time
import pandas as pd
from to_modify.algorithm import DATA_TO_PERFORM_IQR as IQR_DATA

def iqr_outlier():
    filename = IQR_DATA.FILE_LOCATION  # plik z danymi
    column_name = IQR_DATA.COLUMN_NAME  # kolumna dla której wykonać metodę
    # Współczynnik do wykrycia nietypowych przypadków
    outlier_coefficient = IQR_DATA.OUTLIER_COEFFICIENT
    print('Rozpoczęto wykonywanie funkcji iqr')
    start_time = time.time()  # Rozpoczęcie odliczania czasu
    # Import danych z pliku
    df = pd.read_csv(filename)
    column_data = df[column_name]
    q1 = column_data.quantile(0.25)  # Obliczenie 1 kwartylu
    q3 = column_data.quantile(0.75)  # Obliczenie 3 kwartylu
    iqr = q3 - q1  # Obliczenie IQR

    fence_low = q1 - outlier_coefficient * iqr  # dolna granica odstępstwa
    fence_high = q3 + outlier_coefficient * iqr  # górna granica odstępstwa
    # Detekcja przypadków odstających
    outliers = df[(column_data < fence_low) | (column_data > fence_high)]
    outliers_case = list(zip(outliers.index + 1, outliers[column_name]))
    run_time = time.time() - start_time  # Zakończenie mierzenia czasu
    print('Zakończono wykonywanie funkcji IQR w czasie %s sekund.' % run_time)
    return outliers_case  # Zwrócenie odstających przypadków
```

Rysunek 17. Implementacja IQR - Python

Do obliczenia wartości kwartyli została wykorzystana biblioteka pandas [15]. Następie zgodnie z teorią zostały poczynione dalsze kroki. Został obliczony wskaźnik iqr oraz ustalone zostały wartości powyżej i poniżej których zostaną zwrócone nietypowe przypadki.

Wzorując się na powyższej implementacji, a także części teoretycznej została napisana funkcja przystosowana do narzędzia ClickHouse. Przedstawia ją rysunek 18.

```python
import time
from clickhouse_connect.driver import Client

def iqr(client: Client, idColName: str, colName: str, tableName: str, outlierCoefficient: float):
    print('Rozpoczęto wykonywanie funkcji iqr')
    start_time = time.time()
    IQR_QUERY = 'WITH iqr_data_helper AS (SELECT quantileExact(0.25)(x) AS q1, ' \
                'quantileExact(0.75)(x) AS q3, (q3 - q1) AS iqr FROM (SELECT {} AS x FROM {})), outliers AS ' \
                '(SELECT {} as id, {} as data FROM {} WHERE data < (SELECT q1 - {} * iqr FROM iqr_data_helper) ' \
                'OR data > (SELECT q3 + {} * iqr ' \
                'FROM iqr_data_helper)) SELECT id, data FROM outliers' \
        .format(colName, tableName, idColName, colName, tableName, outlierCoefficient, outlierCoefficient)
    result = client.query(IQR_QUERY)
    run_time = time.time() - start_time
    print('Zakończono wykonywanie funkcji iqr w czasie %s sekund.' % run_time)
    return result.result_set
```

Rysunek 18. Implementacja IQR - ClickHouse

Generuje ona dynamicznie SQL na podstawie przekazanych danych z pliku DATA_TO_PERFORM_IQR.py, czyniąc metodę uniwersalną, adaptującą się do przekazanych danych. Uruchamiając skrypt z domyślnie przypisanymi danymi, zostanie wygenerowany SQL zaprezentowany na rysunku 16.

```sql
WITH iqr_data_helper AS (SELECT quantileExact(0.25)(x) AS q1,
                                quantileExact(0.75)(x) AS q3,
                                (q3 - q1)              AS iqr
                         FROM (SELECT liczba_metrow AS x FROM MIESZKANIA)),
     outliers AS
         (SELECT id as id, liczba_metrow as data
          FROM MIESZKANIA
          WHERE data < (SELECT q1 - 1.5 * iqr FROM iqr_data_helper)
             OR data > (SELECT q3 + 1.5 * iqr
                        FROM iqr_data_helper))
SELECT id, data
FROM outliers
```

Rysunek 19. IQR - przykład SQL dla ClickHouse

W powyższym SQL-u można wyróżnić trzy etapy.

1) WITH iqr_data_helper… – Obliczenie kwartylu pierwszego i trzeciego oraz wartości iqr na podstawie tych wyników. Do obliczenia wartości kwartyli została wykorzystana istniejąca funkcja Clickhouse quantileExact.
2) outliers AS… – Zawiera przypadki odstające zidentyfikowane poprzez klauzulę WHERE dla wskazanego w części SELECT zbioru danych wraz z numerem id. Zgodne z teorią został podstawiony współczynnik 1.5, który można modyfikować.
3) Zwrócenie wartości nietypowych z kroku 2.

Domyślne dane znajdujące się w pliku DATA_TO_PERFORM_Z_IQR.py dla jakich ma zostać wykonana metoda przedstawia rysunek 20. Wartości zostały stosownie pokomentowane, zgodnie z ich przeznaczeniem.

```
# Nazwa tabeli (dla clickhouse) (+ opcjonalnie klauzula np. WHERE)
TABLE_NAME = 'MIESZKANIA'
# TABLE_NAME = "MIESZKANIA WHERE miasto = 'Krakow'"
# Ścieżka do pliku (dla iqr-numpy)
FILE_LOCATION = 'file_data/data_10000000.csv'
# Nazwa kolumny dla której zostanie wykonane iqr
COLUMN_NAME = 'cena_m'
# COLUMN_NAME = 'liczba_metrow'
# Nazwa kolumny z id w tabeli
ID_COLUMN_NAME = 'id'
# współczynnik obliczający przypadek odstający
OUTLIER_COEFFICIENT = 1.5
```

Rysunek 20. Domyślne dane dla wykonania metody IQR

Opierając się na przedstawionej implementacji poniżej została dokonana analiza złożoności czasowej skryptu Python.

1) Obliczenie kwartyli – Funkcja quantile z pakietu Pandas. Obliczenia te wymagają posortowania danych, co jest głównym kosztem czasowym tego kroku. Operacja ta charakteryzuję się złożonością rzędu O(N * log(N)), gdzie N oznacza ilość danych.
2) Obliczenie iqr, górnej i dolnej granicy. Podstawowe operacje arytmetyczne ze złożonością rzędu O(1).
3) Identyfikacja wartości odstających. Przegląd zbioru danych pod kątem wartości odstających. Wymaga przejrzenia całości zbioru, co charakteryzuję się rzędem złożoności O(N), gdzie N oznacza ilość danych.

Największą operacją pod kątem czasowym jest wyznaczenie kwartyli, które wymagają posortowania danych. Wykonanie tej operacji w ClickHouse według zapewnień twórców powinno być szybkie, tym samym znacznie przyspieszając wykonanie algorytmu. Również identyfikacja wartości odstających powinna być szybsza. Z racji, że wykorzystywane są tu proste metody analityczne, spodziewana jest natychmiastowa odpowiedź w narzędziu ClickHouse. W skrypcie Pythonowym w raz ze wzrostem danych spodziewany jest przyrost czasowy wraz ze wzrostem całkowitej liczby danych.

3.5 K-means

Plik kmean_python.py jest skryptem uruchomieniowy wypisujący nietypowe przypadki z wykorzystaniem metody kmean_outlier. Jej kod przedstawia rysunek 21.

```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
from to_modify.algorithm import DATA_TO_PERFORM_KMEAN as KMEAN_DATA

def kmean_outlier():
    filename = KMEAN_DATA.FILE_LOCATION  # plik z danymi
    column_name_1 = KMEAN_DATA.COLUMN_FIRST_DATA_POINT_NAME_PYTHON  # 1 kolumna do klasteryzacji
    column_name_2 = KMEAN_DATA.COLUMN_SECOND_DATA_POINT_NAME_PYTHON  # 2 kolumna do klasteryzacji

    df = pd.read_csv(filename)  # Odczyt danych
    df[column_name_2] = df[column_name_2] / 1000  # Standaryzacja
    features = [column_name_1, column_name_2]  # kolumny do kmean
    # Wykonanie metody grupującej
    kmeans = KMeans(n_clusters=2, max_iter=100, n_init=1, init=KMEAN_DATA.INITIAL_CENTROIDS)
    kmeans.fit(df[features])
    # Tablica wyników z dystansem do najbliższego centroidu z użyciem metryki euklidesowej
    distances = pairwise_distances(df[features], kmeans.cluster_centers_, metric="euclidean")
    return distances, df
```

Rysunek 21. Implementacja k-means - Python

Do dokonania grupowania została wykorzystana metoda k-means z pakietu scikit-learn [16]. W celu standaryzacji danych kolumna cena_m została podzielona przez tysiąc. Metoda ta wypisuję liczba iteracji. Wartość ta jest warunkiem zatrzymania w implementacji w skrypcie realizującym metodę w narzędziu ClickHouse. W części badawczej ta wartość jest przepisywana do wykonania metody w części ClickHousowej. Dzięki temu porównanie wyników czasowych odbędzie się na takich samych warunkach.

Bazując na powyższym skrypcie oraz teorii został utworzony skrypt dostosowujący k-means do narzędzia ClickHouse. Wywołuję go skrypt k_means_clickhouse.py. Skrypt został przedstawiony na rysunku 22.

```python
import math
import time
from typing import List, Tuple
from clickhouse_connect.driver import Client

def kmean(client: Client, tableName: str, id_col_name: str, column_first_data_point_name: str,
          column_second_data_point_name: str, number_of_iteration: int, beginCentroid: List[Tuple[int, int]],
          outlier_distance: float):
    print('Rozpoczęto wykonywanie funkcji kmean')
    start_time = time.time()
    client.command('DROP TABLE IF EXISTS km_clusters'.format(tableName))
    client.query(CREATE_KM_CLUSTERS_TABLE)
    for counter, (x, y) in enumerate(beginCentroid, start=1):
        client.query(INSERT_KM_CLUSTERS.format(counter, x, y))

    for _ in range(number_of_iteration - 1):
        result = client.query(CLUSTERING_AND_NEW_CENTROID
                              .format(id_col_name, column_first_data_point_name, column_second_data_point_name,
                                      tableName))
        client.command(TRUNCATE_KM_CLUSTERS)
        for cluster_id, x, y in result.result_set:
            client.query(INSERT_KM_CLUSTERS.format(cluster_id, round(x, 1), math.ceil(y)))

    outliers_result = client.query(LAST_ITERATION_RETURN_OUTLIERS
                                   .format(id_col_name, column_first_data_point_name, column_second_data_point_name,
                                           tableName, outlier_distance))
    run_time = time.time() - start_time
    print('Zakończono wykonywanie funkcji knn w czasie %s sekund.' % run_time)
    return outliers_result.result_set
```

Rysunek 22. Implementacja k-means - ClickHouse

Kody Sql przedstawione na powyższym rysunku jako stałe, zostały przedstawione na rysunkach 23, 24, 25. W dalszej części wynikowy sql został dokładniej przedstawiony.

```
CREATE_KM_CLUSTERS_TABLE = '''
CREATE TABLE km_clusters (
    id Int32,
    x Float32,
    y Float32
) ENGINE = MergeTree()
ORDER BY id;
'''
```

```
INSERT_KM_CLUSTERS = '''
INSERT INTO km_clusters (id, x, y) values ({}, {}, {})
```

```
TRUNCATE_KM_CLUSTERS = '''
TRUNCATE TABLE km_clusters
'''
```

Rysunek 23. Kody SQL tabeli pomocniczej km_kluster

```
CLUSTERING_AND_NEW_CENTROID = '''
WITH closest_cluster AS (
    SELECT
        d.{} AS d_id,
        c.id AS c_id,
        d.{} AS d_x,
        {} AS d_y,
        c.x AS c_x,
        c.y AS c_y,
        ROW_NUMBER() OVER(PARTITION BY d_id ORDER BY L2Distance((d_x, d_y), (c_x, c_y))) AS rn
    FROM {} AS d
    JOIN km_clusters AS c
    ON 1=1
),
cluster_stats AS (
    SELECT
        c_id AS cluster_id,
        AVG(d_x) AS avg_d_x,
        AVG(d_y) AS avg_d_y
    FROM closest_cluster
    WHERE rn = 1
    GROUP BY c_id
)
SELECT * FROM cluster_stats
ORDER BY cluster_id
'''
```

Rysunek 24. SQL dla ClickHouse wykonujący grupowanie i zwracający punkty nowych centroidów

```
LAST_ITERATION_RETURN_OUTLIERS = '''
WITH closest_cluster AS (
    SELECT
        d.{} AS d_id,
        c.id AS c_id,
        d.{} AS d_x,
        {} AS d_y,
        c.x AS c_x,
        c.y AS c_y,
        L2Distance((d_x, d_y), (c.x, c.y)) AS distance,
        ROW_NUMBER() OVER(PARTITION BY d.id ORDER BY L2Distance((d_x, d_y), (c.x, c.y))) AS rn
    FROM {} AS d
    JOIN km_clusters AS c
    ON 1=1
),
cluster_stats AS (
    SELECT
    d_id,
    d_x,
    d_y,
    distance
    FROM closest_cluster
    WHERE rn = 1
)
SELECT * FROM cluster_stats
WHERE distance > {}
'''
```

Rysunek 25. SQL dla ClickHouse - ostatnia iteracja zwracająca nietypowe przypadki

Podstawiając do generowanego SQL-a domyślne dane z pliku DATA_TO_PERFOM_KMEANS.py powstaną zapytania zaprezentowane na rysunkach 26, 27.

```sql
WITH closest_cluster AS (
    SELECT
        d.id AS d_id,
        c.id AS c_id,
        d.liczba_metrow AS d_x,
        d.cena_m / 1000 AS d_y,
        c.x AS c_x,
        c.y AS c_y,
        ROW_NUMBER() OVER(PARTITION BY d_id ORDER BY L2Distance((d_x, d_y), (c_x, c_y))) AS rn
    FROM MIESZKANIA AS d
    JOIN km_clusters AS c
    ON 1=1
),
cluster_stats AS (
    SELECT
        c_id AS cluster_id,
        AVG(d_x) AS avg_d_x,
        AVG(d_y) AS avg_d_y
    FROM closest_cluster
    WHERE rn = 1
    GROUP BY c_id
)
SELECT * FROM cluster_stats
ORDER BY cluster_id
```

Rysunek 26. Przykład SQL dla ClickHouse wykonującego grupowanie i zwracającego nowe centroidy

```sql
WITH closest_cluster AS (
    SELECT
        d.id AS d_id,
        c.id AS c_id,
        d.liczba_metrow AS d_x,
        d.cena_m / 1000 AS d_y,
        c.x AS c_x,
        c.y AS c_y,
        L2Distance((d_x, d_y), (c.x, c.y)) AS distance,
        ROW_NUMBER() OVER(PARTITION BY d.id ORDER BY L2Distance((d_x, d_y), (c.x, c.y))) AS rn
    FROM MIESZKANIA AS d
    JOIN km_clusters AS c
    ON 1=1
),
cluster_stats AS (
    SELECT
    d_id,
    d_x,
    d_y,
    distance
    FROM closest_cluster
    WHERE rn = 1
)
SELECT * FROM cluster_stats
WHERE distance > 3
```

Rysunek 27. Przykład SQL dla ClickHouse wykonującego ostatnią iteracją ze zwróceniem odstających przypadków

Implementacja metody k-means w ClickHouse składa się z następujących kroków. Na początek tworzona jest tabela km_clusters i zostaję ona wypełniona początkowymi centroidami wraz z inkrementowaną wartością id o jeden, oznaczającą numer skupienia. Następnie wykonany zostaję SQL przedstawiony na rysunku 26, składający się z poniższych kroków.

1) WITH closest_cluster… - Najpierw poprzez funkcję CROSS JOIN tworzony jest iloczyn kartezjański tabeli ze zbiorem danych z tabelę z skupieniami. Dzięki temu w kroku ROW_NUMBER() OVER(PARTITION BY d_id ORDER BY L2Distance((d_x, d_y), (c_x, c_y))) AS rn, zostaję obliczona odległość do każdego skupienia. Klauzula ORDER BY sprawia, że dodana kolumna rn będzie miała wartość 1 dla danej z wartością skupienia do którego ma najbliżej.
2) cluster_stats AS… - Obliczenie nowych wartości centroidów. Klauzula WHERE rn=1, jak zostało napisane w 1 kroku pooduję, że zapytanie korzysta tylko z danych, które są najbliżej punktów skupień. Następnie klauzula GROUP_BY dokonuję grupowania danych ze względu na skupienia, dla których oblicza wartości średniej kolumn, które stają się nowymi centroidami.
3) Zwrócenie nowych wartości centroidów z kroku 2.

Powyższy krok iterowany jest zadaną ilość razy. Ostatnia iteracja przedstawiona na rysunku 27 została zmodyfikowana względem powyżej implementacji. Nie zwraca już nowych centroidów, tylko przypadki odstające. Składa się z poniższych kroków.

1) WITH clocest_cluster… Krok pierwszy analogiczniny jak wyżej.
2) cluster_stats AS… - Różnica w tym kroku polega na braku klauzli GROUP BY. Etap ten zatrzymuję się na odfiltrowaniu danych zostawiając wartości w raz z informacją o dystansie do najbliższego punktu skupienia, jak zostało opisane wcześniej.
3) Zwrócenie danych z kroku 2.

Powyższym sposobem udało się zaimplementować algorytm k-means w narzędziu ClickHouse.

Domyślne dane umieszczone w pliku DATA_TO_PERFORM_K_MEANS.py dla jakich ma zostać wykonana metoda przedstawia rysunek 28. Wartości zostały stosownie pokomentowane, zgodnie z ich przeznaczeniem.

```python
# Nazwa tabeli (dla clickhouse)
TABLE_NAME = 'MIESZKANIA'
# Ścieżka do pliku (dla kmean python)
FILE_LOCATION = 'file_data/data_10000000.csv'
# Nazwa kolumny z id
ID_COLUMN_NAME = 'id'
# Nazwa pierwszej kolumny dla kmens
COLUMN_FIRST_DATA_POINT_NAME_PYTHON = 'liczba_metrow'
COLUMN_FIRST_DATA_POINT_NAME_CLICKHOUSE = 'liczba_metrow'
# Nazwa drugiej kolumny dla kmens
COLUMN_SECOND_DATA_POINT_NAME_PYTHON = 'cena_m'
COLUMN_SECOND_DATA_POINT_NAME_CLICKHOUSE = 'cena_m / 1000'
# Liczba iteracji algorytmu kmeans
NUMBER_OF_ITERATION = 2
# inicjalne centroidy, na ich podstawie wyliczana jest liczba klastrów (tutaj 2)
INITIAL_CENTROIDS = [(0.01, 0.01), (99.01, 22.01)]
# Powyżej tego dystansu od centroidu, wartość jest uznawana za nietypową
OUTLIER_DISTANCE = 3
```

Rysunek 28. Domyślne dane dla wykonania metody k-means

Opierając się na przedstawionej implementacji poniżej została dokonana analiza złożoności czasowej skryptu Python.

1) Operacja standaryzacji kolumny cena_m wykonywana jest na każdej wartości. Złożoność rzędu O(N), gdzie N to ilość danych do przetworzenia.
2) Inicjalizacja i dopasowanie modelu K-means za pomocą funkcji kmeans.fit(df[features]) z pakietu scikitlearn. Wymaga przejrzenia całości zbioru danych i dopasowania do odpowiedniego skupienia. Złożoność takiej operacji jest rzędu O(N * K), gdzie N to ilość danych, a K to liczba skupień.
3) Obliczenie dystansów do najbliższego z centroidu w celu detekcji przypadków odstających. Funkcja pairwise_distances z pakietu scikitlearn. Wymaga obliczenia odległości każdej wartości z danych do wyznaczonych punktów skupień. Złożoność takiej operacji jest rzędu O(N * K), gdzie N to ilość danych, a K to liczba skupień.

W przypadku implementacji ClickHouse najbardziej złożonym czasowo krokiem jest operacja złączenia danych z tabelą km_cluster. Charakteryzuję się ona złożonością czasową rzędu O(N*M), gdzie N to liczba wierszy w pierwszej tabeli (dane), a M to liczba wierszy w drugiej tabeli (skupienia). Dla pozostałych kroków

jak odfiltrowanie danych, czy obliczenie średniej w celu uzyskania nowych punków skupień spodziewana jest natychmiastowa odpowiedź. Z racji wymagającej operacji złączenia danych, w tym przypadku nie można oczekiwać natychmiastowego rezultatu wykorzystując narzędzie ClickHouse. Natomiast w ogólnym rozrachunku spodziewane jest uzyskanie wyniku szybciej od skryptu wykonanego w Python.

4 Część eksperymentalna

Poniższy rozdział przedstawia część eksperymentalną pracy. Schemat w kolejnych podrozdziałach wygląda następująco. Na początku zostaje wykonany skrypt na jednomilionowym zbiorze danych zarówno w implementacji Python, jak i ClickHouse. Następnie przedstawione zostało porównanie otrzymanych wyników obydwu implementacji. Pozwala to zweryfikować poprawność autorskiej implementacji. Została także dokonana analiza odnalezienia odstających przypadków. W dalszej części zostało zamieszczone porównanie czasowe na różnych wielkościowo danych. Na końcu rozdziału zostało zamieszczone podsumowanie części badawczej.

4.1 Narzędzia, sposób przeprowadzenia eksperymentów

Do przeprowadzenia części doświadczalnej został wykorzystany laptop z następującymi parametrami:

- Procesor: AMD Ryzen 5 3550H
- Pamięć RAM: 13.9 GB
- Dysk SSD 512GB: Kioxia KXG60ZNV512G
- System operacyjny: Windows 10 Home w wersji 22H2
- Python w wersji: 3.9
- Docker w wersji: 20.10.24 z wykorzystaniem WSL2
- Baza danych ClikHouse w wersji: 23.3.1.2823

Aktualnie baza danych ClickHouse powiązana jest z systemem Linux. Nie ma możliwości uruchomienia jest natywnie z wykorzystaniem systemu Windows.

W przypadku realizowanej pracy baza danych została uruchomiona przy pomocy narzędzia Docker. Wykorzystane do tego zostało następującego polecenie:

```
docker run -d --name clickhouse-mgr -p 8123:8123 clickhouse/clickhouse-server.
```

Dzięki temu utworzony został kontener o nazwie clickhouse-mgr z systemem Ubuntu, który jest oparty o system Linux. Następnie uruchamia on bazę danych ClickHouse. Dla niej został udostępniony port 8123 umożliwiający komunikację z kontenerem. Wykorzystuje on dostępną moc procesora oraz maksymalnie może skorzystać z 6.715 GiB pamięci RAM.

Rysunek 29. Metryki uruchomionego kontenera z bazą ClickHouse

Domyślnie używana będzie najnowsza stabilna wersja obrazu udostępniona w repozytorium docker hub [17]. Do tego polecenia został również dostosowany plik DB_CONNECTION_DATA.py, przetrzymujący stałe potrzebne do ustanowienia połączenia z bazą danych ClickHouse.

Przy domyślnym stanie projektu, aby przeprowadzić eksperymenty należy:

1) Upewnić się, że na komputerze jest zainstalowany Python oraz jest uruchomiona instancja bazy ClickHouse.
2) Należy zainstalować dodatkowe pakiety: scikit-learn, pandas, numpy, scipy, clickhouse-connect, wymagane przez skrypty. Przykładowo korzystając z repozytorium PIP należy w terminalu wpisać polecenia:
    - pip install scikit-learn
    - pip install pandas
    - pip install numpy
    - pip install scipy
    - pip install clickhouse-connect
3) Bądąc w katalogu głównym projektu należy dodać dane do narzędzia ClickHouse wywołując skrypt insert_data_to_database.py, przykładowo będąc w katalogu głównym projektu wpisując w terminalu:
   python .\insert_data_to_database.py
4) Teraz analogicznie jak wyżej można uruchamiać kolejne skrypty realizujące wyszukiwanie nietypowych przypadków
    - python .\z_score_python.py – Eksperyment wyszukujący nietypowe przypadki metodą z-score z wykorzystaniem Python’a.
    - python .\z_score_clickhouse.py – Eksperyment wyszukujący nietypowe przypadki metodą z-score z wykorzystaniem ClickHouse.
    - python .\iqr_clickhouse.py – Eksperyment wyszukujący nietypowe przypadki metodą iqr z wykorzystaniem Python’a.
    - python .\iqr_clickhouse.py – Eksperyment wyszukujący nietypowe przypadki metodą iqr z wykorzystaniem ClickHouse.

- python .\k_means_clickhouse.py – Eksperyment wyszukujący nietypowe przypadki metodą k-means z wykorzystaniem Python'a.
- python .\k_means_clickhouse.py – Eksperyment wyszukujący nietypowe przypadki metodą k-means z wykorzystaniem ClickHouse.

Modyfikując pliki w katalogu to_modify można dostosować dane oraz progi wskazujące na przypadek odstający, zgodnie z tym jak przedstawiono to w rozdziale 3.

4.2 Z-score

4.2.1 Analiza poprawności

W celach porównawczych poprawność w skryptach został zamieniony warunek zwracający nietypowe przypadki na rzecz wypisania 5 rezultatów z największą i najmniejszą wartością z-score. Domyślnie skrypt będzie zwracał przypadki odstające spoza ustanowionego zakresu, jak przedstawiono w rozdziale z implementacją. Jako że metoda ta działa na pojedynczych kolumnach, została wykonana dla danych: liczba_metrow oraz cena_m. Poniżej zostały przedstawione wyniki.

Id	Wartość	Z-score
193556	16000	11.95
1714	13500	4.46
3624	12806	2.39
66286	12806	2.39
70903	12806	2.39
35890	11285	-2.16
48533	11285	-2.16
101797	11285	-2.16
105824	11285	-2.16
127347	11285	-2.16

Tabela 2. Metoda z-score. Wyniki dla danych: cena_m - Python

Id	Wartość	Z-score
193556	16000	11.95

1714	13500	4.46
3624	12806	2.39
66286	12806	2.39
70903	12806	2.39
35890	11285	-2.16
48533	11285	-2.16
105824	11285	-2.16
127347	11285	-2.16
101797	11285	-2.16

Tabela 3. Metoda z-score. Wyniki dla danych: cena_m. - ClickHouse

Id	Wartość	Z-score
802779	80.00	1.93
1714	68.00	1.18
238	67.00	1.12
393	67.00	1.12
622	67.00	1.12
960331	20.00	-1.81
667	31.00	-1.12
905	31.00	-1.12
959	31.00	-1.12
1159	31.00	-1.12

Tabela 4. Metoda z-score. Wyniki dla danych: liczba_metrow - Python

Id	Wartość	Z-score
802779	80.00	1.93
1714	68.00	1.18
35541	67.00	1.12
34869	67.00	1.12
34949	67.00	1.12
960331	20.00	-1.81
35323	31.00	-1.12
35190	31.00	-1.12

35188	31.00	-1.12
34377	31.00	-1.12

Tabela 5. Metoda z-score. Wyniki dla danych: liczba_metrow - ClickHouse

Na podstawie otrzymanych wyników można stwierdzić poprawność autorskiego algorytmu. Implementacja wykorzystująca ClickHouse zwróciła przypadki z taką samą wartością z-score jak implementacja Python.

Poniżej została przedstawiona analiza wykrycia nietypowych przypadków. Jak zostało przedstawione w części implementacyjnej, za wartości odstające uznajemy dane nie mieszczące się w przedziale <-3;3>.

cena_m	z-score	Czy wykryto
13500	± 4.46	Tak, > 3, odstaje od reszty
16000	± 11.94	Tak, > 3, odstaje od reszty

Tabela 6. Metoda z-score. Analiza wykrycia nietypowych przypadków dla cena_m

liczba_metrow	z-score	Czy wykryto
80.00	± 1.93	Nie, < 3, nie odstaje znacząco od reszty
50.00	± -0.062	Nie, > -3, nie odstaje znacząco od reszty
20.00	± -1.8	Nie, > -3, nie odstaje znacząco od reszty

Tabela 7. Metoda z-score. Analiza wykrycia nietypowych przypadków dla liczba_metrow

Wskazaną metodą udało się znaleźć dwa przypadki odstające, które odstają od zadanego progu. Obydwa dotyczą kolumny cena_m. Jak opisano w rozdziale z częścią teoretyczną, dane te mają zbliżone do siebie wartości. Zgodnie z przypuszczeniami dla danych z taki rozkładem algorytm z łatwością znajduje odstające przypadki. Gorzej wygląda kwestia w przypadku kolumny liczba_m. Tutaj metoda nie potrafiła jednoznacznie zidentyfikować odstających przypadków. Stało się tak, gdyż wartości w tej kolumnie oscylują wokół dwóch przedziałów. Metoda ta nie jest najlepsza dla danych o takim rozkładzie. Zwłaszcza widać to po odstającej wartości 50. Jest to wartość pośrodku dwóch przedziałów w jakim znajdują się dane. Z racji swojej specyfiki z-score potraktował ją jako bardzo małą szansę na przypadek odstający. Pozostałe dwa nietypowe przypadki, co oczywiste

mają największą i najmniejszą wartość z-score, jednak po wyniku z-score ciężko stwierdzić jednoznacznie, aby znacząco odbiegały od reszty. W sytuacji, gdybyśmy chcieli znaleźć takie przypadki musielibyśmy najpierw przefiltrować dane. Zamiast wykonywać metodę na danych skupionych wokół dwóch grup, trzeba by podzielić je na dwie części np. według wartości liczba metrów >= 50. Taki zabieg ujednolici dane i spowoduję, że uda się wykryć wszystkie przypadki. Niemniej wymaga to oczywiście wiedzy na temat rozkładu danych. Wykonując metodę na całym zbiorze nie otrzymamy wszystkich przypadków.

4.2.2 Porównanie czasowe

Poniżej zostały zamieszczony rezultaty czasowe wykonania metody dla różnych zbiorów danych z przywróconą wersją skryptów zwracających tylko odstające przypadki.

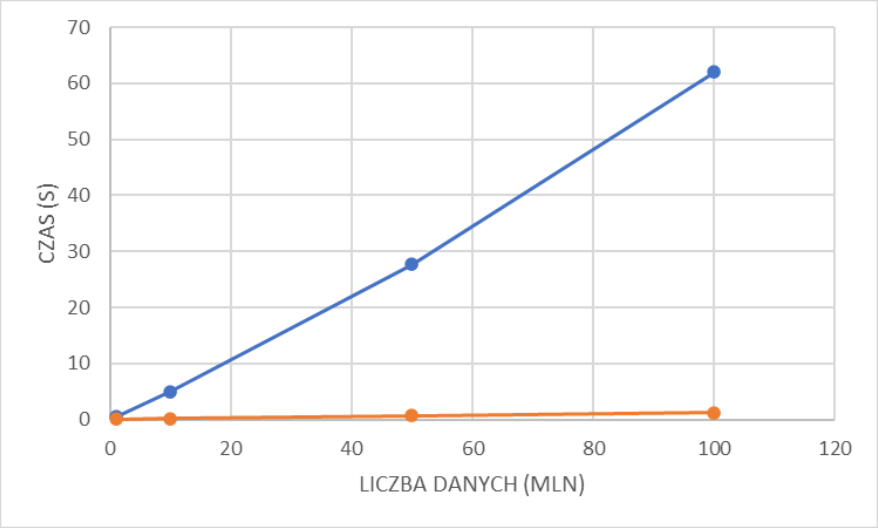
Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.41	0.58	0.42	0.47
10	liczba_metrow	4.99	5.01	4.84	4.95
50	liczba_metrow	27.59	27.44	27.91	27.65
100	liczba_metrow	62.76	61.21	62.19	62.05
1	cena_m	0.45	0.40	0.44	0.43
10	cena_m	5.69	5.57	5.71	5.66
50	cena_m	28.08	27.61	28.19	27.96
100	cena_m	62.60	63.11	62.23	62.65

Tabela 8.Metoda z-score. Czasy wykonania - Python

Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.09	0.09	0.10	0.09
10	liczba_metrow	0.19	0.20	0.19	0.19
50	liczba_metrow	0.70	0.73	0.72	0.72
100	liczba_metrow	1.20	1.22	1.22	1.21

1	cena_m	0.09	0.11	0.10	0.1
10	cena_m	0.20	0.20	0.19	0.20
50	cena_m	0.69	0.69	0.70	0.69
100	cena_m	1.26	1.24	1.27	1.26

Tabela 9. Metoda z-score. Czasy wykonania - ClickHouse



Rysunek 30. Metoda z-score. Porównanie czasów - Python vs ClickHouse

Jak można zaobserwować wykonanie tej metody w ClickHouse jest znacznie szybsze od implementacji z wykorzystaniem Pythona. Dopiero 100 milionowy zestaw danych spowodował przekroczenie 1 sekundy w implementacji ClickHouse. Metoda ta w tym narzędziu działa błyskawicznie, czego nie można powiedzieć o implementacji Python. Czasy tam rosną praktycznie liniowo, dla 100 milionowego zbioru danych czas wykonania przekroczył 1 minutę. Nie zaobserwowano większych różnic pomiędzy czasem wykonania na danych przechowujących liczbę całkowitą, a danych z liczbą dziesiętną.

Poniżej został przedstawione czasy z drugą próbą z drobną modyfikacją. Często na zbiorze chcemy wykonywać jakieś operacje. Do obydwu skryptów została dodana operacja filtrowania danych według miasta ‘Kraków’. Czyli dodano linijki:

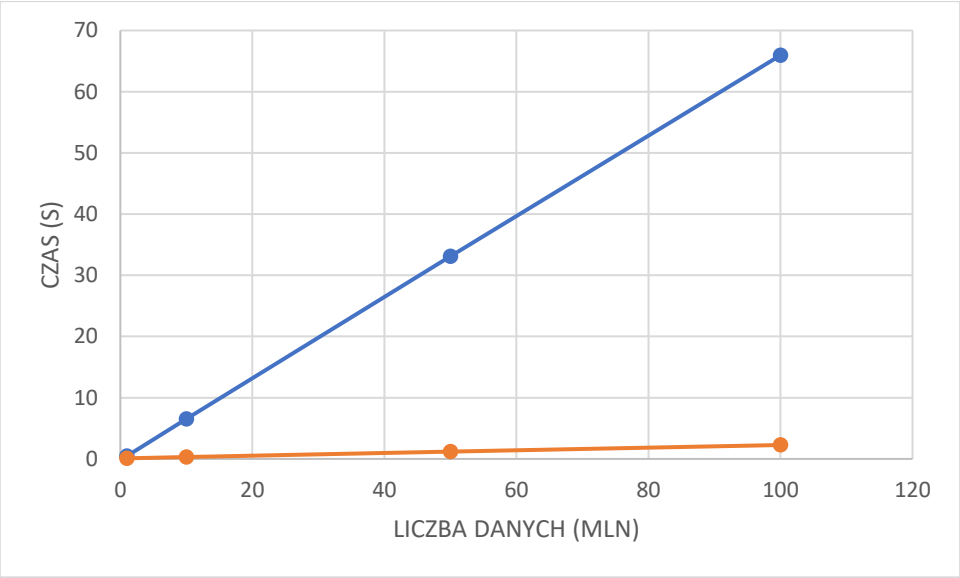
- Python: filtered_data = data[data['miasto'] == 'Krakow']
- ClickHouse: WHERE miasto = 'Krakow'"

Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.49	0.44	0.48	0.47
10	liczba_metrow	6.84	6.34	6.46	6.55
50	liczba_metrow	33.19	32.93	33.11	33.08
100	liczba_metrow	65.88	65.74	66.23	65.95
1	cena_m	0.53	0.52	0.49	0.51
10	cena_m	6.25	6.11	6.07	6.14
50	cena_m	31.87	32.17	32.98	32.34
100	cena_m	66.54	67.01	66.42	66.66

Tabela 10. Metoda z-score z filtrowaniem – Czasy wykonania - Python

Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.10	0.09	0.11	0.10
10	liczba_metrow	0.30	0.31	0.30	0.30
50	liczba_metrow	1.20	1.19	1.20	1.20
100	liczba_metrow	2.28	2.26	2.28	2.27
1	cena_m	0.09	0.11	0.10	0.10
10	cena_m	0.30	0.30	0.31	0.30
50	cena_m	1.23	1.22	1.23	1.23
100	cena_m	2.37	2.37	2.36	2.37

Tabela 11. Metoda z-score z filtrowaniem – Czasy wykonania - ClickHouse



Rysunek 31. Metoda z-score z filtrowaniem. Porównanie czasów - Python vs ClickHouse

Dodanie filtrowania zwiększyło czasy obydwu metoda. Korzystniej znowu wypadła implementacja ClickHouse, dla której czasy wykonania zostały zwiększone w znacznie mniejszym stopniu (1 sekunda vs 4 sekundy).

4.3 IQR

4.3.1 Analiza poprawności

Zgodnie ze schematem opisanym na początek należy zweryfikować poprawność algorytmów. Poniżej zostały przedstawione rezultaty dokonanych obliczeń pomocniczych w celu identyfikacji wartości odstających metodą IQR z domyślnie ustalonymi parametrami przedstawionymi w rozdziale Implementacja.

Implementacja	Q1	Q3	IQR	Dolny zakres	Górny zakres
Python	33.00	65.00	32.00	-15.00	113.00
Clickhouse	33.00	65.00	32.00	-15.00	113.00

Tabela 12. Metoda IQR. Pomocnicze wyliczenia - Python

Implementacja	Q1	Q3	IQR	Dolny zakres	Górny zakres
Python	11759	12249	490	11024	12984
Clickhouse	11759	12249	490	11024	12984

Tabela 13. Metoda IQR. Pomocnicze wyliczenia - ClickHouse

Na podstawie otrzymanych wyników można stwierdzić poprawność autorskiego algorytmu w ClickHouse. Funkcje do obliczenia wartości kwartyli oraz IQR zwróciły takie same wartości. Poniżej została przedstawiona analiza wykrycia nietypowych przypadków.

cena_m	Czy wykryto
13500	Tak, wartość większa od górnego przedziału
16000	Tak, wartość większa od górnego przedziału

Tabela 14. Metoda IQR. Analiza wykrycia nietypowych przypadków dla cena_m

liczba_metrow	Czy wykryto
80	Nie, wartość mniejsza od górnego zakresu i większa od dolnego zakresu
50	Nie, wartość mniejsza od górnego zakresu i większa od dolnego zakresu
20	Nie, wartość mniejsza od górnego zakresu i większa od dolnego zakresu

Tabela 15. Metoda IQR. Analiza wykrycia nietypowych przypadków dla liczba_metrow

Rezultat poszukiwania odstających przypadków za pomocą metody IQR jest podobny jak dla metody z-score. Wytłumaczenie tego jest analogiczne jak w powyższym rozdziale. W przypadku tej metody również dużą rolę odgrywa rozkład. Rozkład danych kolumny cena_m pozwala tą metodą znaleźć odstające przypadki. Z tego samego powodu zawodzi przy kolumnie liczba_metrow.

4.3.2 Porównanie czasowe

Poniżej zostały zamieszczony rezultaty czasowe wykonania metody dla różnych zbiorów danych.

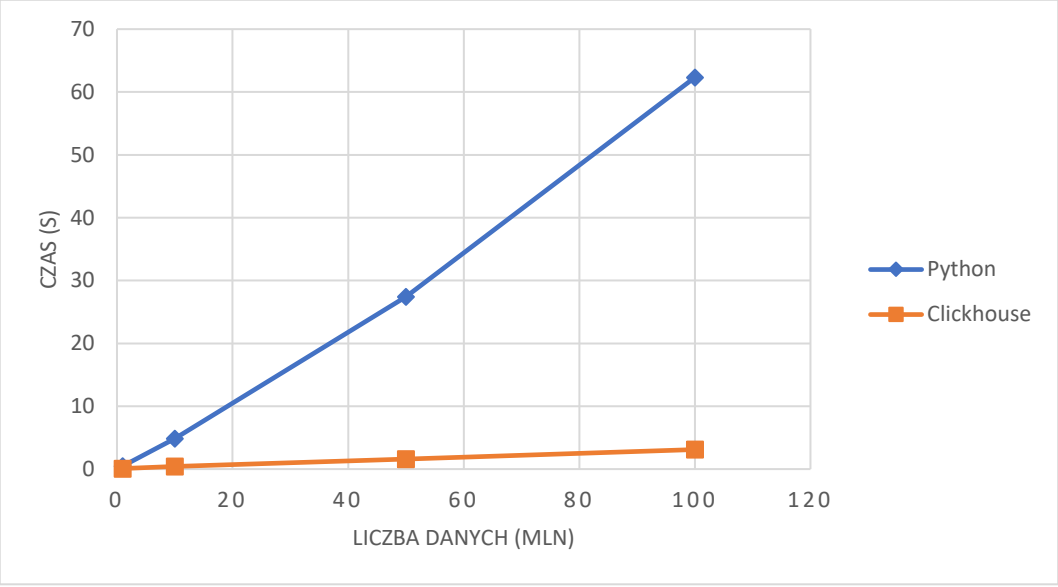
Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.49	0.51	0.43	0.48
10	liczba_metrow	4.95	5.61	4.01	4.86
50	liczba_metrow	27.49	27.73	27.11	27.44
100	liczba_metrow	62.72	61.91	62.33	62.32
1	cena_m	0.45	0.49	0.41	0.45

10	cena_m	5.14	4.92	5.23	5.10
50	cena_m	28.90	27.52	28.07	28.16
100	cena_m	63.42	62.01	62.98	62.80

Tabela 16. Metoda IQR. Czasy wykonania - Python

Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.09	0.09	0.10	0.09
10	liczba_metrow	0.43	0.45	0.44	0.44
50	liczba_metrow	1.57	1.58	1.56	1.57
100	liczba_metrow	3.06	3.16	3.12	3.11
1	cena_m	0.09	0.10	0.09	0.09
10	cena_m	0.40	0.38	0.38	0.39
50	cena_m	1.33	1.34	1.34	1.34
100	cena_m	2.74	2.83	2.91	2.83

Tabela 17. Metoda IQR. Czasy wykonania - ClickHouse



Rysunek 32. Metoda IQR. Porównanie czasów – Python vs ClickHouse

Wykrywanie odstających przypadków jest wykonywane zdecydowanie szybciej w ClickHouse. Tutaj również jak przy metodzie z-score nawet dla 100 milionowego zbioru odpowiedź zostaje zwrócona błyskawicznie, około 22 razy szybciej od implementacji w Python. Tutaj również nie zaobserwowano większych

różnic pomiędzy czasem wykonania na danych przechowujących liczbę całkowitą, a danych z liczbą dziesiętną.

Analogicznie jak przy metodzie z-score do implementacji została dołożona część z filtracją danych ze względu na miasto. Poniżej zostały zamieszczone wyniki badające wpływ na czasy wykonania algorytmu.

- Python: filtered_data = data[data['miasto'] == 'Krakow']
- ClickHouse: MIESZKANIA where miasto = 'Krakow'

Poniżej zostały zamieszczone wyniki weryfikujące jak wpłynęło to na ostateczne czasy.

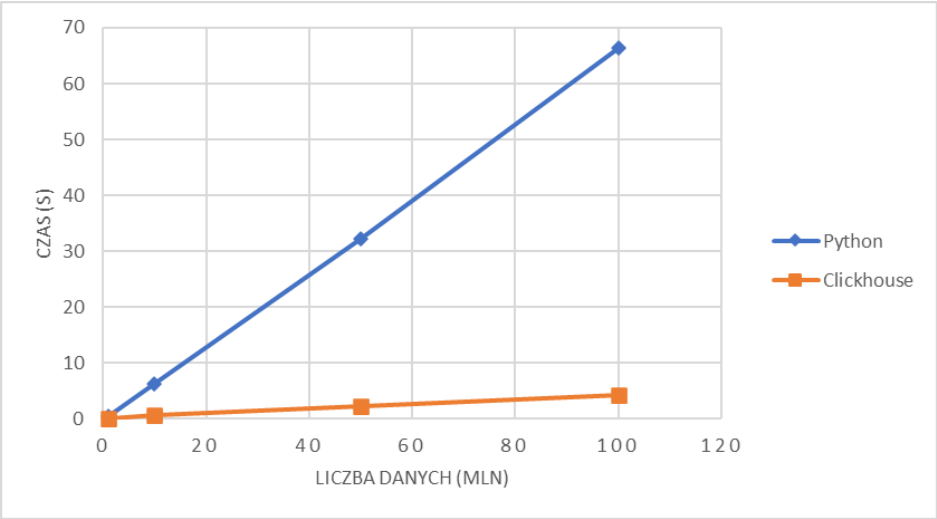
Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.51	0.51	0.52	0.51
10	liczba_metrow	5.96	6.11	6.61	6.23
50	liczba_metrow	33.49	31.73	31.11	32.11
100	liczba_metrow	65.72	66.91	66.33	66.32
1	cena_m	0.45	0.49	0.41	0.45
10	cena_m	5.14	4.92	5.23	5.10
50	cena_m	31.90	31.52	32.07	31.83
100	cena_m	66.42	67.01	66.98	66.80

Tabela 18. Metoda IQR z filtrowaniem. Czasy wykonania - Python

Liczba danych (mln)	Kolumna	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	liczba_metrow	0.09	0.10	0.10	0.10
10	liczba_metrow	0.56	0.57	0.58	0.57
50	liczba_metrow	2.20	2.28	2.25	2.24
100	liczba_metrow	4.21	4.11	4.06	4.13
1	cena_m	0.10	0.10	0.11	0.1
10	cena_m	0.50	0.51	0.5	0.5
50	cena_m	2.06	2.05	2.08	2.06

100	cena_m	3.91	3.87	4.03	3.94

Tabela 19. Metoda IQR z filtrowaniem. Czasy wykonania - ClickHouse



Rysunek 33. Metoda IQR z filtrowaniem. Porównianie czasów – ClickHouse vs Python

Impakt filtrowania danych jest mniejszy dla ClickHouse. Dla 100 milionowego zestawu danych zwiększył czas około 1 sekundę, natomiast dla implementacji Python o około 4 sekundy. Nie są to jednak wyniki znacznie wpływające na końcowy czas.

4.4 K-means

4.4.1 Analiza poprawności

Zgodnie ze schematem opisanym na początku rozdziału, poniżej został zamieszczony rezultat wykonania metody k-means. W celach porównawczych poprawności metody w skrypcie został dostosowany etap filtrowania do zwrócenia po dziesięć rezultatów z największym dystansem do najbliższego skupienia, zamiast wyłącznie rezultatów powyżej wskazanego progu.

Id	liczba_metrow	cena_m	Dystans
802779	50.00	12000	15.002
769045	80.00	12000	14.997
960331	20.00	12000	13.00
193556	35.00	16000	4.46
1714	68.00	13500	3.35
3624	35.00	11285	2.12
70903	35.00	2.12	2.12

Tabela 20 Metoda k-means. Wyniki - Python

Id	liczba_metrow	cena_m	Dystans
769045	80.00	12000	15.01
802779	50.00	12000	14.99
960331	20.00	12000	13.01
193556	35.00	16000	4.46
1714	68.00	13500	3.36
3624	35.00	11285	2.15
70903	35.00	2.12	2.12

Tabela 21. Metoda k-means. Wyniki - ClickHouse

Na podstawie otrzymanych wyników można stwierdzić poprawność autorskiego algorytmu. Poniżej została przedstawiona analiza wykrycia nietypowych przypadków. Przypominając w implementacji został ustalony próg 3. Powyżej tej odległości wartość uznawana jest za odstającą.

cena_m	k-means	Czy wykryto
13500	± 3.35	Tak, > 3, odstaje od reszty
16000	± 4.45	Tak, > 3, odstaje od reszty

Tabela 22. Metoda k-means. Analiza wykrycia nietypowych przypadków

liczba_metrow	k-means	Czy wykryto
80	± 15	Tak, > 3, odstaje od reszty
50	± 14.99	Tak, > 3, odstaje od reszty
20	± 13	Tak, > 3, odstaje od reszty

Tabela 23. Metoda k-means. Analiza wykrycia nietypowych przypadków

Wskazaną metodą udało się znaleźć wszystkie przypadki odstające. Każdy z nich miał odległość do najbliższego skupienia większą od zadanej minimalnej odległości. Metoda ta w porównaniu do poprzednich brała pod uwagę dwie zmienne. Dzięki temu jest bardziej odporna na nietypowe rozkłady danych. Wymaga jednak większej znajomości danych, gdyż wymaga wskazania liczby skupień oraz może wystąpić konieczność standaryzacji zmiennych.

4.4.2 Porównanie czasowe

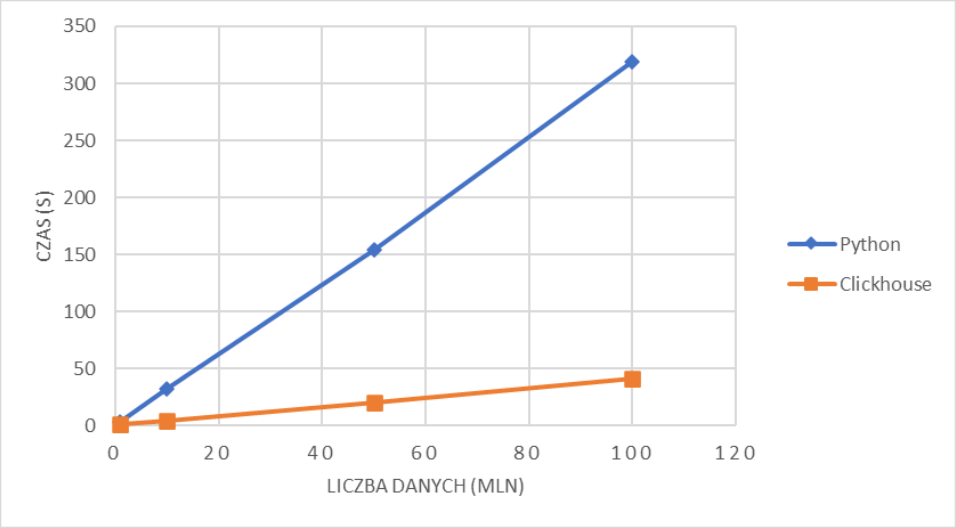
Poniżej został przedstawione czasy działania z docelowym ustawionym progiem detekcji odstających przypadków.

Liczba danych (mln)	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	3.35	3.36	3.35	3.35
10	31.57	32.14	31.97	31.89
50	154.49	153.21	154.76	154.15
100	318.43	320.11	318.98	319.17

Tabela 24. Metoda k-means. Czasy wykonania - Python

Liczba danych (mln)	Czas 1 (s)	Czas 2 (s)	Czas 3 (s)	Średni czas (s)
1	0.89	0.85	0.87	0.87
10	4.30	4.32	4.31	4.31
50	19.78	19.62	19.67	19.69
100	40.87	41.08	41.12	41.02

Tabela 25. Metoda k-means. Czasy wykonania - ClickHouse



Rysunek 34. Metoda k-means. Porównanie czasów - Python vs ClickHouse

Wykrywanie nietypowych przypadków z użyciem k-means przebiegło sprawniej w ClickHouse. W przypadku dużych danych musimy jednak zaczekać trochę dłużej na odpowiedź w porównaniu do metod IQR, czy z-score. Wynika to oczywiście z większego skomplikowania tej metody oraz faktu, że jest to algorytm iteracyjny. Dochodzi też klauzula JOIN łącząca całość danych z wszystkimi dodanymi centroidami.

4.5 Analiza wyników części badawczej

Szczegółowe zestawienie porównawcze czasów wykonywania poszczególnych przedstawia tabela 26.

Liczba danych (mln)	Metoda	Czas (ClickHouse)	Czas (Python)	Różnica (stosunek)
1	Z-score	0.09	0.47	5.22
10	Z-score	0.19	4.95	26.05
50	Z-score	0.72	27.65	38.4
100	Z-score	1.22	62.05	50.86
1	IQR	0.09	0.48	5.33
10	IQR	0.44	4.86	11.05
50	IQR	1.57	27.44	17.48
100	IQR	3.11	62.32	20.04
1	K-means	0.87	3.35	3.85
10	K-means	4.31	31.97	7.42
50	K-means	19.67	154.76	7.87
100	K-means	41.12	318.98	7.76

Tabela 26. Porównanie średnich czasów wszystkich metod

Implementacja każdej z metod wypadła zdecydowanie lepiej w narzędziu ClickHouse. Pod względem czasowym najlepiej wypadła metoda z-score. Wykorzystując ją dane zwracane są prawie natychmiastowo. Wraz z przyrostem danych widać coraz większy uzysk czasu w porównaniu z implementacją w Pythonie. Metoda IQR również działa szybko, jednak wolniej od z-score. Widać też znacznie mniejszy stosunek korzyści czasowej w porównaniu z jej odpowiednikiem w Python. W metodzie k-means uzysk czasowy stabilizuje się. Działa jednak wydajniej i przy dużych danych, wraz ze zwiększaniem ich rozmiaru, różnica czasowa jest coraz bardziej widoczna.

Część doświadczalna potwierdziła założenia zawarte w części teoretycznej. Metody z-score i iqr są stosunkowo proste i do obliczenia ich wykorzystywane są funkcję analityczne, które są wysoce zoptymalizowane w ClickHouse. Metoda k-means jest trochę bardziej skomplikowana. Jest to algorytm iteracyjny. Oblicza się

w nim również odległość do każdego skupienia. Chcąc to zrobić wykonywana jest operacja JOIN, która powiela ilość danych o tyle razy, ile została wskazana liczba skupień. Ciężko, więc oczekiwać natychmiastowych odpowiedzi korzystając z tej metody.

Potwierdziły się również przypuszczenia, że metody z-score i iqr będą mieć problemy z rozkładem danych, który nie przypomina rozkładu normalnego. Wykonywane na całym zbiorze nie potrafiły znaleźć wszystkich nieprawidłowych przypadków. W tym przypadku odpowiednie filtrowanie danych, które spowodowałoby ustandaryzowanie danych pozwoliłoby tym metodom na znalezienie ich. Test filtrowania danych nie wykazał dużego impaktu na końcowy czas. Wobec tego, że metody te działają błyskawicznie, nic nie stoi na przeszkodzie, aby próbować odpowiednio wyselekcjonować dane. Z wykryciem wszystkich przypadków poradziła sobie za to metoda k-means. Jest ona bardziej odporna na nietypowe rozkłady danych. Dodatkowo operuję na ilości zmiennych większej niż pojedyncze, więc może pozwolić na odkrycie jakiś nieznanych korelacji.

5 Podsumowanie

W ramach pracy udało się spełnić założenia stanowiące temat pracy. Udało się wyselekcjonować metody, których przełożenie na narzędzie ClickHouse znacznie zoptymalizowało czasy przeszukiwań dużych zbiorów danych w kontekście wykrycia nietypowych przypadków.

Metody IQR i Z-score wykorzystują do wyliczenia swoich wartości popularne metody analityczne. Potwierdziło się, że ich natywna implementacja w ClickHouse pozwala na niemal natychmiastową odpowiedź zwrotną, dzięki czemu obie metody działają błyskawicznie na wielomilionowych danych. K-means jako bardziej złożona metoda, nie pozwoliła na już na tak szybkie wykonanie, nadal jednak w widoczny sposób wykonywała się optymalnej.

Idealnych metod oczywiście nie ma. Chcąc analizować duże dane nie można ograniczać się do jednej wybranej. Narzędzie ClickHouse zdecydowanie potwierdziło, że jest narzędziem, które potrafi zoptymalizować pracę nad dużymi zbiorami i warto przenosić do tego narzędzia metody służące eksploracji danych.

Choć praca skupiała się głównie wokół wyszukiwanie nietypowych danych, część implementacyjna może przysłużyć się także do innych zastosowań. Przykładowo z-score używane jest do standaryzacji danych. Jest to operacja używana bardzo często w analizie danych, a jak pokazał ClickHouse z jego wykorzystaniem operacja ta jest bardzo szybka. K-means pokazało, że również metody grupowania wykonywana są wydajniej w ClickHouse. Można więc oczekiwać, że inne algorytmy tego typu również będą wykonywać się wydajniej.

6 Bibliografia

[1] Ranking popularności języków programowania, https://www.tiobe.com/tiobe-index, (dostęp 15.08.2023)

[2] Oficjalna strona projektu ClickHouse, https://clickhouse.com, (dostęp 15.08.2023)

[3] Wstęp oficjalnej dokumentacji projektu ClickHouse, https://clickhouse.com/docs/en/intro, (dostęp 15.08.2023)

[4] Zita A. Vale, Tiago Pinto, Michael Negnevitsky, Ganesh Kumar Venayagamoorthy, *Intelligent Data Mining and Analysis in Power and Energy Systems: Models and Applications for Smarter Efficient Power Systems* (2023), IEEE Press Series on Power and Energy Systems

[5] Artykuł dotyczący użycia języka SQL do detekcji nietypowych przypadków metodą z-score, Haki Benita*, Simple Anomaly Detection Using Plain SQL*, https://hakibenita.com/sql-anomaly-detection, (dostęp 15.08.2023)

[6] Peter J. Rousseeuw and Mia Hubert, WIREs Data Mining Knowl Discov 2018, 8:e1236. doi: 10.1002/widm.1236, (dostęp 15.08.2023)

[7] Cathy Tanimura, *SQL for Data Analysis* (2021), O'Reilly Media

[8] Alabrah A. An Improved CCF Detector to Handle the Problem of Class Imbalance with Outlier Normalization Using IQR Method. Sensors. 2023; 23(9):4406. https://doi.org/10.3390/s23094406, (dostęp 15.08.2023)

[9] Artykuł dotyczący wykorzystania metody IQR do detekcji przypadków nietypowych z użyciem SQL, Rob Solgadao, *Anomaly Detection With SQL* , https://towardsdatascience.com/anomaly-detection-with-sql-7700c7516d1d, (dostęp 15.08.2023)

[10] Ripan, R.C., Sarker, I.H., Hossain, S.M.M. *et al.* A Data-Driven Heart Disease Prediction Model Through *K*-Means Clustering-Based Anomaly Detection. *SN COMPUT. SCI.* **2**, 112 (2021). https://doi.org/10.1007/s42979-021-00518-7, (dostęp 15.08.2023)

[11] Angelin and A. Geetha, "Outlier Detection using Clustering Techniques – K-means and K-median," *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2020, pp. 373-378, doi: 10.1109/ICICCS48265.2020.9120990

[12] Ordonez, Carlos. (2006). Integrating K-means clustering with a relational DBMS using SQL. Knowledge and Data Engineering, IEEE Transactions on. 18. 188- 201. 10.1109/TKDE.2006.31

[13] Ordonez, Carlos. (2004). Programming the K-means clustering algorithm in SQL. KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 823-828. 10.1145/1014052.1016921

[14] Dokumentacja metody z-score użytej w implementacji Python, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html, (dostęp 15.08.2023)

[15] Dokumentacja projektu pandas użytego w implementacji Python do obliczenia kwartyli w metodzie iqr oraz do wydajnego procesowania plików csv, https://pandas.pydata.org, (dostęp 15.08.2023)

[16] Dokumentacja metody k-means użytej w implementacji Python, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, (dostęp 15.08.2023)

[17] Repozytorium obrazów Docker, skąd została pobrana i uruchomiona instancja bazy ClickHouse, https://hub.docker.com/r/clickhouse/clickhouse-server, (dostęp 15.08.2023)

7 Wykaz rysunków

8 Wykaz tabel

9 Streszczenie

Tytuł: Optymalizacja efektywności metod wykrywania nietypowych przypadków w dużych danych za pomocą narzędzia ClickHouse

Praca skupia się wokół wykorzystania narzędzia ClickHouse przystosowanego do szybkiego przeszukiwania dużych danych w celu optymalizacji metod wykrywania nietypowych przypadków. Pierwsza część pracy poświęcona jest analizie metod do tego służących. Kolejnym etapem jest implementacja ich w sposób zrozumiały dla ClickHouse w jak najbardziej optymalny sposób. Autorską implementację uzupełnia implementacja z wykorzystaniem gotowych rozwiązań w języku Python. Część badawcza zawiera porównanie wykonania tych metod w obydwu środowiskach. Ostatnia część poświęcona jest wnioskom i podsumowaniu całości prac.

Title: Optimizing the effectiveness of methods for detecting unusual cases in big data using the ClickHouse tool.

The thesis focuses on the use of the ClickHouse tool adapted for fast searching of big data to optimize methods for detecting unusual cases. The first part of the work is dedicated to analyzing the methods for doing this. The next step is to implement them in a way that ClickHouse understands in the most optimal way possible. The author's implementation is supplemented by an implementation using off-the-shelf solutions in Python. The research part includes a comparison of the execution of these methods in both environments. The last part is devoted to conclusions and a summary of the entire work.