МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ДАТА-МАЙНИНГ В РОЗРОБЦІ ДОДАТКІВ ДЛЯ ОНЛАЙН ТОРГІВЛІ»**

Виконав: студент 2 курсу, групи 8.1222
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
А.В. Пономаренко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент кандидат фізико-математичних наук, доцент кафедри програмної інженерії математики, Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., доцент

_____ Шило Г.М.
(підпис)

" 23 " травня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Пономаренку Антону Валерійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Дата-майнинг в розробці додатків для онлайн торгівлі

керівник роботи Борю Сергій Юрійович, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » квітня 2023 року № 642-с

2. Строк подання студентом роботи 11.12.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Програмна реалізація системи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка архітектури системи.
4. Розробка веб-додатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 23.06.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	30.06.2023	
2.	Збір вихідних даних.	09.07.2023	
3.	Обробка методичних та теоретичних	27.08.2023	
	джерел.		
4.	Розробка першого та другого розділу.	10.09.2023	
5.	Розробка третього розділу.	12.10.2023	
6.	Оформлення та нормоконтроль	27.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	11.12.2023	

Студент _____ А.В. Пономаренко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Дата-майнинг в розробці додатків для онлайн торгівлі»: 99 с., 5 рис., 11 табл., 12 джерел, 2 додатків.

АНАЛІТИЧНИЙ ПАРАЛІЧ, ВЕБ-АНАЛІТИКА, ВЕЛИКІ ДАННІ, ВІЗУАЛІЗАЦІЯ ДАНИХ, ІНТЕГРАЦІЯ ДАНИХ, ОНЛАЙН-ТОРГІВЛЯ, РОЗГОРТАННЯ ДОДАТКУ, СПАРК, ТЕХНОЛОГІЇ АНАЛІЗУ.

Об’єкт дослідження: Процес створення та впровадження веб-додатку для онлайн-торгівлі, який використовує різні технології аналізу даних для подолання аналітичного паралічу та покращення процесів управління бізнесом.

Мета роботи: Вивчення проблем аналітичного паралічу в бізнесі та розробка та реалізація веб-додатку для збору, аналізу та візуалізації даних онлайн-торгівлі з використанням Apache Spark[1] та інших технологій аналізу. Мета полягає в пошуку способів подолання аналітичного паралічу та підвищенні якості управлінських рішень у бізнесі.

Метод дослідження: Базується на зборі аналізі та порівнянні існуючих методів обробки даних, використанні відповідних інструментів та технологій для розробки веб-додатку та інтеграції його з Apache Spark[1]. В роботі використовуються методи аналізу, програмування, а також використання сучасних технологій для обробки й візуалізації великих обсягів даних.

У роботі здійснено вивчення та розробку веб-додатку для онлайн-торгівлі, використовуючи різні технології аналізу даних з метою подолання аналітичного паралічу та підвищення ефективності управлінських рішень у сфері бізнесу. Результатом реалізація веб-додатку, спрямованого на оптимізацію та покращення процесів онлайн-торгівлі, що може бути корисним для підвищення продуктивності та прийняття обґрунтованих управлінських рішень у бізнесі.

SUMMARY

Master’s Qualifying Theses «Data Mining in Developing Applications for Online Trading»: 99 pages, 5 figures, 11 tables, 12 references, 2 appendices.

ANALYTICS TECHNOLOGIES, APPLICATION DEPLOYMENT, BIG DATA, DATA INTEGRATION, DATA VISUALIZATION, ONLINE TRADING, SPARK, WEB ANALYTICS, ANALYTIC PARALYSIS.

Research Object: The process of creating and implementing a web application for online trading that utilizes various data analysis technologies to overcome analytic paralysis and enhance business management processes.

Objective: Studying the issues of analytic paralysis in business and developing and implementing a web application for the collection, analysis, and visualization of online trading data using Apache Spark[1] and other analytics technologies. The goal is to find ways to overcome analytic paralysis and improve the quality of managerial decisions in business.

Research Method: Based on collecting and analyzing existing data processing methods, employing suitable tools and technologies to develop a web application and integrating it with Apache Spark[1]. The work involves analysis, programming methods, and using modern technologies for processing and visualizing large volumes of data.

The work involved studying and developing a web application for online trading, utilizing various data analysis technologies to overcome analytical paralysis and enhance the efficiency of managerial decision-making in the business sphere. The outcome was the implementation of a web application aimed at optimizing and improving online trading processes, which could be beneficial for increasing productivity and making informed managerial decisions in business.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Основні проблеми організації онлайн торгівлі 9
1.1 Аналітика даних в бізнесі: поняття і підходи 9
1.2 Проблема "аналітичного паралічу" та її вплив на бізнес 10
1.3 Огляд сучасних технологій та інструментів аналізу даних 11
1.4 Огляд сучасних технологій та інструментів аналізу даних 13
2 Розробка Архітектури системи 15
2.1 Генерація даних 15
2.1.1 Мета генерації даних 15
2.1.2 Алгоритми генерації 16
2.1.3 Структура та формат даних 17
2.2 Обробка та агрегація даних 19
2.2.1 Мета обробки даних 19
2.2.2 Модуль DataGenerationJob 22
2.4 Розгортання додатку та інтеграція з Apache Spark 24
2.3.2 Налаштування середовища Spark 24
2.3.2 Інтеграція з Apache Spark 24
2.5 Вибір та обґрунтування використаних технологій 25
2.5.1 Обґрунтування вибору Spark для обробки даних 25
2.5.2 Обґрунтування вибору PostgreSQL для зберігання даних 26
2.5.3 Обґрунтування вибору Java для розробки 26
2.5.4 Обґрунтування вибору Docker для розгортання 26
2.5.5 Обґрунтування вибору Spring та Hibernate 27
3 Розробка веб-додатку 28

3.1 Проектування бази даних та моделі даних 28
3.2 Реалізація функціональності для обробки та аналізу даних 31
3.2.1 Прогнозування доходу 31
3.2.2 Щотижневі, квартальні та повні прогнози 46
3.2.3 Щотижневий рейтинг продажів 49
3.2.4 Профілювання клієнтів 50
3.2.5 Хмара слів 52
3.2.6 Топ клаймбер та фоллер, мувер та шейкер (*ref*. Top Climber and Faller / Movers and Shakers) 53
3.2.7 Можливості підписки 54
3.3 Візуалізація даних 55
Висновки 57
Перелік посилань 59
Додаток А Код реалізації DataProcessingJob 60
Додаток Б Код реалізації ForecastCalculationHolder 90

ВСТУП

Сьогоднішній бізнесовий світ характеризується безпрецедентним обсягом даних, які накопичуються в різних компаніях і магазинах. Велика кількість організацій розуміє, що ці дані мають великий потенціал і можуть стати джерелом цінної інформації та інсайтів для прийняття рішень. Проте, одна проблема, яка стикається багатьох підприємств, полягає в складності обробки та аналізу цих даних.

Цей явище, що іноді називають "аналітичним паралічем", виникає не через недостатність даних або некомпетентність у використанні аналітичних інструментів[3, 7]. Просто обсяг та складність даних надто великі, щоб людина можна було швидко і ефективно їх проаналізувати та використати для прийняття рішень. Числа без контексту, запутані зв'язки між різними наборами даних та відсутність чіткої структури можуть стати перешкодою в отриманні цінних інсайтів.

Саме тому розробка ефективного та проникливого аналітичного інструменту має вирішальне значення для успіху бізнесу. Цей інструмент повинен забезпечувати обробку, аналіз та використання даних швидко, ефективно та зрозуміло для користувача. Він має допомагати здійснювати стратегічне планування, приймати обґрунтовані рішення та відслідковувати результати.

У цій дипломній роботі розробляється веб-додаток, який вирішує проблему аналітичного паралічу шляхом забезпечення зручного та ефективного інструменту для обробки та аналізу даних в бізнесі. Розробка виконується з використанням сучасних технологій та інструментів, такі як Java 8[6], Spring[11], Hibernate[5] та PostgreSQL[8], для створення потужного та функціонального додатку.

Основним інструментом обробки даних, який буде використовуватись, є Apache Spark [1]. Вибір Spark обумовлений його здатністю виконувати

розподілені та ітераційні обчислення в пам'яті, що робить його особливо корисним при роботі з великими обсягами даних та алгоритмами машинного навчання. Використання Spark дозволить нам здійснювати швидку та ефективну аналітику без необхідності використовувати повільні дискові операції для збереження проміжних результатів.

1 ОСНОВНІ ПРОБЛЕМИ ОРГАНІЗАЦІЇ ОНЛАЙН ТОРГІВЛІ

1.1 Аналітика даних в бізнесі: поняття і підходи

Аналітика даних в бізнесі відіграє ключову роль у прийнятті обґрунтованих рішень та впровадженні ефективних стратегій. Цей розділ присвячений огляду понять і підходів, пов'язаних з аналітикою даних, і їх значенню для бізнес-середовища.

Аналітика даних – це процес збору, обробки, аналізу та використання даних для отримання цінної інформації та висновків. У бізнесі аналітика даних дозволяє підприємствам розкрити приховані залежності, тренди та шаблони у своїх даних, що дозволяє зробити осмислені рішення.

Одним з основних підходів до аналітики даних є дескриптивний аналіз, який зосереджується на описі поточного стану справ у бізнесі на основі наявних даних. Цей підхід дозволяє відповісти на питання "Що відбувається?", надаючи бізнесу загальне уявлення про свої показники та результати.

Інший важливий підхід – прогнозування і передбачення. Використовуючи статистичні моделі та алгоритми машинного навчання, аналітики можуть розробити прогнози майбутніх подій та трендів у бізнесі. Це дозволяє підприємствам готуватися до майбутніх змін, виявляти можливості та знижувати ризики.

Також важливим елементом аналітики даних є виявлення взаємозв'язків та кореляцій між різними змінними. Це дозволяє зрозуміти, які фактори впливають на показники бізнесу та які змінні можуть бути ключовими для досягнення успіху.

В цьому розділі будуть розглянуті основні поняття аналітики даних в бізнесі, а також різні підходи і методи, які використовуються для отримання цінної інформації з даних. Детальний огляд літератури та проведених досліджень

дозволить визначити найефективніші інструменти та підходи для досліджуваної проблематики.

Цей розділ допоможе розуміти сутність аналітики даних в бізнесі та її значення для прийняття обґрунтованих рішень. Він також послужить основою для подальшого дослідження та розробки аналітичного додатку для отримання цінної інформації з даних бізнесу.

1.2 Проблема "аналітичного паралічу" та її вплив на бізнес

Аналітика даних є незамінною складовою успішної стратегії розвитку бізнесу. Однак, навіть при наявності великої кількості даних та інструментів для їх аналізу, можуть виникати складнощі, пов'язані з ефективним використанням цих ресурсів. Одна з найпоширеніших проблем у цьому контексті відома як "аналітичний параліч".

"Аналітичний параліч" виникає, коли організації зіштовхуються з надмірним обсягом даних та складнощами їх інтерпретації. Нерозбірливість та надмірність інформації можуть призвести до того, що співробітники не можуть виявити ключові висновки або прийняти обґрунтовані рішення на основі даних. Ця проблема особливо поширена на інформаційних панелях керування, де дані представлені без належного контексту та зрозумілого формату.

Наслідки "аналітичного паралічу" для бізнесу можуть бути серйозними. Перш за все, це призводить до втрати часу та ресурсів. Замість того, щоб використовувати дані для отримання цінних інсайтів та прийняття управлінських рішень, співробітники витрачають час на невпорядковану та безплідну обробку даних.

Крім того, "аналітичний параліч" може призвести до неправильних або необґрунтованих рішень. Коли дані не представлені в зрозумілому та логічному форматі, важко зрозуміти їх суть та знайти зв'язки між ними. Це може призвести

до прийняття рішень на основі неповної або спотвореної інформації, що може негативно вплинути на стратегію бізнесу та призвести до фінансових втрат.

Нарешті, "аналітичний параліч" може призвести до втрати конкурентної переваги. У сучасному бізнес-середовищі, де швидкість та точність прийняття рішень мають вирішальне значення, організації, які не можуть належним чином аналізувати та використовувати свої дані, відстають від конкурентів. Прийняття інформованих рішень та розробка ефективних стратегій потребують доступу до чіткої, зрозумілої та релевантної аналітичної інформації.

Отже, вирішення проблеми "аналітичного паралічу" стає нагальною задачею для організацій, що прагнуть досягти успіху на ринку. Налагодження ефективних процесів аналітики даних, використання спеціалізованих інструментів та забезпечення належного навчання та підтримки співробітників можуть допомогти подолати цю проблему і забезпечити віддачу від аналітики даних, необхідну для успішного розвитку бізнесу.

1.3 Огляд сучасних технологій та інструментів аналізу даних

У сучасному світі, де обсяги даних зростають експоненційно, існує широкий спектр технологій та інструментів, які допомагають організаціям здійснювати аналіз даних та отримувати цінні інсайти. В цьому розділі ми розглянемо деякі з найбільш поширених технологій та інструментів аналізу даних.

Apache Hadoop [4]: Hadoop є відкритим фреймворком для обробки великих обсягів даних паралельно на кластері комп'ютерів. Він базується на розподіленій системі файлів Hadoop (HDFS) та фреймворку MapReduce, який дозволяє розподілено обробляти дані на вузлах кластера. Hadoop є потужним інструментом для обробки та аналізу структурованих і неструктурованих даних.

Apache Spark [1]: Spark є іншим розподіленим фреймворком для аналізу даних, який працює в пам'яті. Він надає широкий спектр функцій для обробки

даних, включаючи можливості розподіленого обчислення, потокової обробки даних та машинного навчання. Spark дозволяє швидко та ефективно аналізувати великі обсяги даних і використовувати їх для отримання цінних висновків.

Tableau [12]: Tableau є популярним інструментом візуалізації даних, який дозволяє легко створювати інтерактивні графіки, діаграми та звіти. Він дозволяє користувачам взаємодіяти з даними та отримувати візуальне представлення складних аналітичних результатів. Tableau забезпечує швидкий та зрозумілий спосіб представлення даних для прийняття рішень.

Python [9] та R [10]: Python та R є двома популярними мовами програмування для аналізу даних та статистики. Вони мають широкий спектр бібліотек та інструментів, які дозволяють проводити обробку даних, виконувати статистичний аналіз та розробляти моделі машинного навчання. Python та R надають гнучкість та потужність для виконання складних аналітичних завдань.

Google Analytics [2]: Google Analytics є одним з найпоширеніших інструментів для аналізу веб-трафіку та поведінки користувачів на веб-сайтах. Він надає детальну інформацію про відвідуваність, джерела трафіку, конверсії та інші метрики, що допомагають організаціям розуміти ефективність їхніх веб-страниць та маркетингових кампаній.

Ці технології та інструменти представляють лише невелику частину доступних варіантів для аналізу даних. Вибір певних технологій залежатиме від потреб організації, типу даних, розмірів обсягів даних та інших факторів. Ефективне використання сучасних технологій та інструментів аналізу даних може значно покращити процеси прийняття рішень і допомогти організаціям досягти успіху в конкурентному бізнес-середовищі.

1.4 Огляд сучасних технологій та інструментів аналізу даних

Метою роботи є дослідження застосування аналізу даних та розробка системи для використання даних у сфері онлайн-торгівлі для досягнення наступних цілей:

Аналіз застосування аналітики даних у бізнесі з метою виявлення ключових аспектів та переваг цього підходу.

Розгляд основних технологій та інструментів аналізу даних для визначення їхнього впливу на ефективність онлайн-торгівлі.

Для досягнення вище зазначеної мети, в роботі вирішуються наступні завдання:

- аналіз і вивчення принципів та можливостей застосування аналітики даних у сфері бізнесу з підкресленням її важливості для оптимізації процесів та прийняття обгрунтованих рішень;
- розробка веб-додатку для обробки та аналізу даних у сфері онлайн-торгівлі з використанням технологій Java 8 [6], Spring [11], Hibernate [5], та PostgreSQL [8];
- використання Apache Spark [1] для обробки даних про клієнтів, замовлення та товари для отримання детальних інсайтів щодо доходів, нових клієнтів, покупок та інших ключових показників;
- розробка вимог до веб-системи, яка буде використовувати дані для оптимізації процесів онлайн-торгівлі, визначення основних завдань та функціональності системи;
- розробка архітектури системи, визначення її цілей та завдань для досягнення оптимальної ефективності в обробці даних;
- вибір та обґрунтування використаних технологій для реалізації системи, враховуючи їхні переваги в контексті конкретних вимог;
- розробка структурної схеми системи та інформаційної моделі, що відображає потік даних та їх обробку;

- розробка інтерфейсу системи та алгоритмів обробки даних для досягнення максимальної продуктивності та зручності використання;
- розробка та реалізація веб-додатку, який використовуватиме дані для оптимізації онлайн-торгівлі;
- розробка інструкцій щодо експлуатації системи для забезпечення її коректної роботи та використання;
- тестування розробленої системи та оцінка її результатів для перевірки відповідності поставленим цілям та завданням.

2 РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ

2.1 Генерація даних

2.1.1 Мета генерації даних

Генерація вхідних даних є ключовим етапом у підготовці даних для подальшого використання в системі. Метою процесу генерації є створення адекватних та реалістичних наборів вхідних даних, які відображають різні сценарії та умови, що можуть виникнути в реальному середовищі.

Основні аспекти мети генерації даних включають:

Репрезентативність: Забезпечення того, що створені дані відображають широкий спектр сценаріїв та реальних умов, що можуть мати місце у виробничому середовищі.

Масштабованість: Здатність генерувати дані в різних масштабах від дрібних до великих обсягів для охоплення різних вимог та умов використання.

Адаптованість: Здатність адаптувати генерацію даних відповідно до конкретних вимог та структури системи обробки.

Якість: Забезпечення точності, послідовності та правдоподібності створених наборів даних для забезпечення надійності результатів обробки та аналізу.

Ефективність: Максимізація швидкості та ефективності процесу генерації даних для забезпечення оптимальних умов використання в робочій системі.

Виконання цих критеріїв визначає успішність генерації вхідних даних, яка відіграє важливу роль у подальшій аналітиці та обробці даних у проекті.

2.1.2 Алгоритми генерації

Генерація користувачів є важливим етапом у підготовці вхідних даних для системи. У цьому проекті використовується алгоритм, який базується на унікальних ідентифікаторах та випадкових значеннях. Кожен користувач отримує унікальний ідентифікатор, електронну адресу, ім'я, прізвище, адресу та інші атрибути. Такий підхід гарантує унікальність та реалістичність даних. Імена та прізвища генеруються шляхом конкатенації унікального ідентифікатора та числа. Email формується на основі унікального ідентифікатора. Генерується випадковим чином. Генерація користувачів може бути обмежена різними параметрами, такими як кількість користувачів та регіон, щоб створювані дані були адекватні та реалістичні. Генерація відбувається з використанням стрімів Java та функціонального програмування для ефективної обробки та створення списку користувачів.

Генерація продуктів передбачає створення унікальних товарів з випадковими параметрами, такими як ціна, категорія, доступність та інші. Використовуються функціональні можливості Java для лаконічного та ефективного коду. Кожен продукт отримує унікальний ідентифікатор, випадковий URL-адрес, зображення та інші характеристики. Ціна визначається випадковим чином в заданому діапазоні. Випадковим чином обирається доступність продукту на складі. URL та інші параметри генеруються на основі унікального ідентифікатора. Це забезпечує реалістичність та різноманітність сценаріїв. Параметри, які обмежують генерацію продуктів, включають максимальну та мінімальну ціну, максимальний залишок товарів на складі та кількість категорій товарів.

Генерація замовлень передбачає створення великої кількості унікальних замовлень, кожен з яких може містити випадкову кількість товарів та кількість одиниць кожного товару. Для реалістичності використовуються випадкові елементи з списку користувачів та продуктів. Випадковим чином обираються дані про спосіб оплати та статус замовлення. Кількість товарів та час замовлення

генеруються випадковим чином. Для генерації замовлень можна обмежити кількість замовлень, максимальну кількість товарів у замовленні та максимальну кількість одиниць кожного товару. Використовуються функціональні можливості Java для зручного створення та обробки списків.

2.1.3 Структура та формат даних

Система використовує ряд сутностей, які відображають основні об'єкти в додатку, такі як Замовлення, Продукти та Користувачі. Ці об'єкти зберігаються в базі даних та взаємодіють між собою за допомогою унікальних ідентифікаторів та відповідних зв'язків (Таблиця 1 – Сутності та атрибути).

Таблиця 1 – Сутності та атрибути

Об’єкт	Атрибут	Тип даних	Опис
Замовлення (Order)	id	String	Унікальний ідентифікатор замовлення
	orderId	String	Унікальний ідентифікатор замовлення
	customerId	String	Унікальний ідентифікатор користувача
	productExId	String	Унікальний ідентифікатор продукту
	qty	Long	Кількість одиниць товару в замовленні
	timestamp	Date	Дата та час розміщення замовлення
	customerEmailAddress	String	Електронна пошта користувача
	customerMobilePhoneNumber	String	Номер мобільного телефону користувача
	paymentMethod	String	Спосіб оплати

Продовження таблиці 1

	currency	String	Валюта оплати
	status	String	Статус замовлення
	discountCode	String	Код знижки
	discountAmount	BigDecimal	Сума знижки
	lineValueExcludingTax	BigDecimal	Загальна вартість замовлення без податку
	lineValueIncludingTax	BigDecimal	Загальна вартість замовлення з податком.
	created	Date	Дата створення запису
	updated	Date	Дата останнього оновлення запису
	billingAddress	String	Адреса для виставлення рахунку
	postalAddress	String	Поштова адреса доставки
Продукт (Product)	product_id	String	Унікальний ідентифікатор продукту
	name	String	Назва продукту
	url	String	URL продукту
	image	String	URL-адреса зображення продукту
	available	Boolean	Доступність продукту
	price	BigDecimal	Ціна продукту
	category	String	Категорія продукту
	stock	Long	Кількість товару на складі
	created	Date	Дата створення запису
	updated	Date	Дата останнього оновлення запису

Кінець таблиці 1

Користувач (User)	user_id	String	Унікальний ідентифікатор користувача
	email	String	Електронна пошта користувача
	firstName	String	Ім'я користувача
	lastName	String	Прізвище користувача
	postcode	String	Поштовий індекс користувача
	created	Date	Дата створення запису
	updated	Date	Дата останнього оновлення запису
	mobileNumber	String	Номер мобільного телефону користувача
	country	String	Країна проживання користувача

Ці дані можна представити у формі об'єктів Java за допомогою Hibernate JPA. Репозиторії, які ви використовуєте (OrderRepository, ProductRepository, UserRepository), дозволяють вам взаємодіяти з цими об'єктами та виконувати операції збереження, оновлення, видалення та вибірки даних, що робить їх доступними для подальшої обробки в системі.

2.2 Обробка та агрегація даних

2.2.1 Мета обробки даних

Генеровані дані у проекті створюють фундаментальну основу для різноманітних операцій, що включають агрегацію, аналіз та оптимізацію. Ці дані широко використовуються в різних аспектах бізнесу, забезпечуючи докладну

інформацію та контекст для прийняття обґрунтованих стратегічних рішень. Нижче розглянуті ключові області використання згенерованих даних (рис. 1).

Рисунок 1

а) аналітика та звітність:

1) призначення: Дані про замовлення, продукти та користувачів використовуються для створення аналітичних звітів, які

допомагають у розумінні ринкових тенденцій та ефективності продажів;

2) значущість: Ці аналітичні звіти надають бізнес-аналітикам та керівникам важливий інструментарій для прийняття рішень, спрямованих на покращення стратегій продажів та збільшення прибутковості;

б) прогнозування та планування:

1) призначення: Історичні дані про продажі використовуються для розроблення моделей прогнозування, що дозволяють ефективно планувати обсяги виробництва та запасів товарів;

2) значущість: Прогнози стають основою для стратегій планування, забезпечуючи підприємству гнучкість у реагуванні на ринкові зміни;

в) оптимізація запасів:

1) призначення: Дані про залишки товарів та їх доступність використовуються для точного управління запасами та забезпечення належного рівня наявності товарів;

2) значущість: Ефективне управління запасами сприяє уникненню зайвих витрат та покращує обслуговування клієнтів;

г) системи бізнес-аналітики:

1) призначення: Інтеграція даних у системи бізнес-аналітики дозволяє отримати глибше розуміння факторів, що впливають на продажі та споживацькі вподобання;

2) значущість: Бізнес-аналітика допомагає ідентифікувати можливості для росту та розвитку бізнесу;

д) управління клієнтським досвідом:

1) призначення: Дані про користувачів використовуються для персоналізації обслуговування та покращення взаємодії з клієнтами;

2) значущість: Покращений клієнтський досвід сприяє збереженню клієнтів та підвищує їхню лояльність;

е) операційне управління:

1) призначення: Дані про статус замовлень та їх обсяг використовуються для ефективного управління операціями та плануванням виробничих процесів;

2) значущість: Це допомагає уникати затримок, покращує процес доставки та забезпечує ефективне виробниче середовище.

Отже згенеровані дані відіграють ключову роль у створенні інформаційної основи для бізнес-процесів. Вони надають можливість для детального аналізу, прогнозування та прийняття стратегічних рішень. Комбінація інформації про користувачів, продукти та замовлення робить дані витоком цінного знання, яке підтримує різні аспекти діяльності підприємства. Забезпечення цих даних відіграє важливу роль у підтримці конкурентоспроможності та ефективності бізнесу в умовах швидкозмінюючогося ринкового середовища.

2.2.2 Модуль DataGenerationJob

Модуль DataGenerationJob є ключовим елементом у процесі обробки та аналізу вихідних даних, згенерованих в результаті роботи системи. Для досягнення ефективної взаємодії із згенерованими даними, модуль виконує ряд технічних операцій, включаючи аналіз та агрегацію. Нижче подано докладний огляд цих технічних деталей.

Модуль DataGenerationJob починає свою роботу з детального аналізу згенерованих даних. Цей аналіз включає в себе:

– структурний аналіз: Перевірка структури даних для визначення наявності усіх необхідних полів та їхніх типів. Це допомагає у визначенні консистентності та правильності формату даних;

– валідація значень: Перевірка значень у кожному полі на відповідність заданим обмеженням та правилам генерації. Це сприяє уникненню помилок та забезпеченню якості вхідних даних.

Далі, модуль виконує процес агрегації, об'єднуючи дані з різних джерел та джерел генерації. Цей етап включає в себе:

– об’єднання таблиць: З’єднання таблиць з користувачами, продуктами та замовленнями на основі унікальних ідентифікаторів. Це формує комплексний набір даних для подальших операцій;
– створення звітів: Формування звітів та агрегованих даних для полегшення подальшого аналізу та використання результатів генерації.

Модуль також враховує аспекти продуктивності та оптимізації шляхом:

– індексація: Застосування відповідних індексів для прискорення доступу до даних та оптимізації швидкості обробки;
– паралельна обробка: Використання механізмів паралельної обробки для збільшення ефективності використання ресурсів.

Модуль обладнаний механізмами моніторингу та логування для виявлення та вирішення можливих проблем. Це включає в себе:

– лог-файли: Запис подій та помилок для подальшого аналізу та виправлення;
– моніторинг продуктивності: Постійний моніторинг роботи модуля для виявлення потенційних та поточних проблем з продуктивністю.

Модуль має вбудовані механізми для підтримки та розширення функціональності. Це включає в себе:

– документація: Повна та зрозуміла документація для спрощення використання та розробки;
– модульність: Можливість розширення функціональності шляхом додавання нових компонентів чи покращення існуючих.

Модуль DataGenerationJob зберігає оброблені дані у визначених форматах та місцях для подальшого використання в системі. Це включає в себе збереження у базі даних, файловій системі або інших призначених сховищах.

Такий підхід до взаємодії згенерованих даних забезпечує їхню готовність для подальших етапів обробки та використання в різних бізнес-сценаріях.

2.4 Розгортання додатку та інтеграція з Apache Spark

2.3.2 Налаштування середовища Spark

Для розгортання Apache Spark та інтеграції з додатком використовується docker-compose файл, який містить налаштування контейнерів середовища, зокрема Spark Master та Spark Worker. Цей файл конфігурації забезпечує створення окремих контейнерів зі Spark, PostgreSQL та Redis, і об'єднує їх у спільну мережу для спрощення спільної роботи.

Процес розгортання Apache Spark починається з запуску контейнера spark-master, який ініціює основу для роботи та керування задачами. Після цього запускається контейнер spark-worker, який приєднується до spark-master, отримуючи вказівки щодо виконання задач. Дані контейнери налаштовані для взаємодії та розподілу завдань, а також забезпечують спільний доступ до даних у спільній мережі.

Це середовище дозволяє додатку інтегруватись з Apache Spark, використовуючи можливості розподіленого обчислення та обробки великих обсягів даних.

2.3.2 Інтеграція з Apache Spark

У конфігураційних файлах або параметрах додатку визначаються адреса та порт, за якими Spark Master буде доступний для додатку. Це передбачає налаштування параметрів специфікації, які вказують Spark на адресу та порт сервера Spark Master.

spark.master: spark://spark-master:7077

У цьому прикладі, spark-master – це назва контейнера зі Spark Master, доступного для використання додатком у спільній мережі. 7077 – це порт, який призначений для комунікації інших служб з Spark Master. Ці дані використовуються для встановлення зв'язку та взаємодії додатку з інфраструктурою Spark.

Для ефективної роботи з Spark необхідно налагодити доступ та використання ресурсів. Це може включати встановлення параметрів для використання пам'яті, процесорного часу та інших обчислювальних ресурсів. Налагодження цих параметрів дозволяє оптимізувати роботу додатку та Spark для обробки великих обсягів даних.

2.5 Вибір та обґрунтування використаних технологій

2.5.1 Обґрунтування вибору Spark для обробки даних

- масштабованість: Spark може обробляти великі обсяги даних, що ідеально відповідає потребам проекту, який вимагає ефективну обробку великих обсягів даних;
- швидкодія обробки даних: Його здатність працювати в оперативній пам'яті та виконувати різні операції з даними в пам'яті, що робить обробку даних ефективною та швидкою;
- модульність та багатофункціональність: Spark надає багато модулів, таких як SQL та MLlib, що забезпечують різноманітні функції для обробки та аналізу даних;
- підтримка розподіленого обчислення: Spark дозволяє розподілене обчислення, що дозволяє працювати з даними на кластерах з багатьох машин.

2.5.2 Обґрунтування вибору PostgreSQL для зберігання даних

– підтримка SQL: PostgreSQL має повну підтримку мови SQL, що робить його дружнім для розробників та аналітиків, які звикли до SQL-запитів;
– висока надійність та стабільність: Відомий своєю надійністю та можливістю працювати у великих завданнях обробки даних;
– масштабованість: PostgreSQL підтримує широкі можливості масштабування, що дозволяє зберігати значні обсяги даних.

2.5.3 Обґрунтування вибору Java для розробки

– універсальність: Java є широко використовуваною мовою програмування, що дозволяє створювати різноманітні додатки;
– крос-платформеність: Програми, написані на Java, можуть працювати на будь-якій платформі;
– велика спільнота розробників: Це сприяє доступності багатьох бібліотек та фреймворків для розробки програм.

2.5.4 Обґрунтування вибору Docker для розгортання

– контейнеризація: Docker надає зручний спосіб пакувати, розповсюджувати та запускати програми у контейнерах;
– уніфікований середовище розробки та розгортання: Це спрощує розробку та деплоймент додатків, сприяючи консистентності середовищ у різних стадіях розробки;
– ізоляція середовищ: Docker надає ізольоване середовище для додатків, що дозволяє запускати декілька екземплярів програм без взаємного впливу на одне одного.

2.5.5 Обґрунтування вибору Spring та Hibernate

Spring: Цей фреймворк надає модульну архітектуру, вбудовану підтримку перевірки та управління життєвим циклом об'єктів;

– Hibernate: Як ORM-фреймворк, Hibernate дозволяє спростити взаємодію з базою даних за допомогою об'єктно-орієнтованої парадигми. Це полегшує розробку та зменшує кількість SQL-коду.

3 РОЗРОБКА ВЕБ-ДОДАТКУ

Код містить 57 класів і складається з 3351 рядків коду. Фрагмент основного класу з реалізацією алгоритмів обробки даних наведено у додатку А

3.1 Проектування бази даних та моделі даних

В контексті проекту було розроблено оптимальну модель бази даних, яка враховує специфіку додатку та забезпечує ефективне зберігання та обробку вхідних та оброблених даних. Основні складові цієї моделі включають:

– сутності: Сутності були визначені відповідно до логіки додатку та його функціоналу. Наприклад, сутності User, Order та Product відображають основні об'єкти в системі.

```
@Entity
@Table(name = "user")
public class User {
   // поля сутності User
}
@Entity
@Table(name = "order")
public class Order {
   // поля сутності Order
}
@Entity
@Table(name = "product")
public class Product {
   // поля сутності Product
}
```

– зв'язки між сутностями: Були встановлені зв'язки між сутностями для відображення їхніх взаємозв'язків у базі даних. Наприклад, зв'язок багато-до-багатьох між Order та Product:

```
@Entity
@Table(name = "order")
public class Order {
    @ManyToMany
    @JoinTable(
        name = "order_products",
        joinColumns = @JoinColumn(name = "order_id"),
        inverseJoinColumns = @JoinColumn(name = "product_id")
    )
    private List<Product> products;
}

@Entity
@Table(name = "product")
public class Product {
    @ManyToMany(mappedBy = "products")
    private List<Order> orders;
}
```

Для забезпечення ефективного зберігання даних та уникнення дублювання інформації, в базі даних застосовані принципи нормалізації. Це дозволяє оптимізувати структуру бази даних та забезпечити консистентність даних. Наприклад:

– розбиття таблиць: Таблиці були розбиті на менші, невеликі частини для збереження даних без зайвого дублювання.

– унікальність даних: Застосування унікальних ключів та зв'язків між таблицями допомагає уникнути повторення інформації.

```
CREATE TABLE IF NOT EXISTS user (
    user_id INT PRIMARY KEY,
    email VARCHAR(255) UNIQUE,
    -- інші поля користувача
);

CREATE TABLE IF NOT EXISTS product (
    product_id INT PRIMARY KEY,
    name VARCHAR(255),
    -- інші поля продукту
);

CREATE TABLE IF NOT EXISTS order (
    order_id INT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    -- інші поля замовлення
    FOREIGN KEY (customer_id) REFERENCES user(user_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);
```

Ці підходи дозволяють підтримувати оптимальну структуру бази даних, що сприяє ефективній роботі додатку та забезпечує консистентність та цілісність даних.

3.2 Реалізація функціональності для обробки та аналізу даних

3.2.1 Прогнозування доходу

Метою цього алгоритму є створення перспективного прогнозу на 12 місяців для будь-якого бізнесу, який відповідає критеріям прийнятності.

Це:

- вони мають дані про замовлення та здійснили принаймні деякі продажі.
- вони мають принаймні дані за день/місяць/тиждень/квартал.

Фрагмент коду з використанням Java 8 та Apache Spark приведений в Додатку Б.

Алгоритм працюватиме таким чином:

Крок 1 Зведені дані про замовлення щотижня:

Цей крок підсумовує загальні продажі, здійснені щотижня. Тиждень починається в понеділок о 00:00 і закінчується в неділю о 23:59:59. Тижні позначені від 1 до 53, відповідно до їхньої позиції у фінансовому році магазину. Фінансовий рік починається з 1-го числа місяця для фінансового року. Це не обов'язково понеділок. Номер тижня можна розрахувати за такою формулою (Надалі приклади алгоритмів будуть представлені у вигляді SQL коду для кращого розуміння):

```
 select case when
date_diff('day',cast(date_trunc('year',cast(d.calendar_week_start as date)) as date),
cast(case when cast(extract(year from cast(d.calendar_week_start as date))as
bigint)  = cast(extract(year from date_trunc('week',cast(d.calendar_week_start as
date)))as bigint)  +1
then  date_trunc('month',cast(d.calendar_week_start as date))
else  date_trunc('week',cast(d.calendar_week_start as date))
end as date)) = 0 then 1
```

```
else (date_diff('day',cast(date_trunc('year',cast(d.calendar_week_start as date)) as
date), cast(case when cast(extract(year from cast(d.calendar_week_start as date))as
bigint)  = cast(extract(year from date_trunc('week',cast(d.calendar_week_start as
date)))as bigint)  +1
then  date_trunc('month',cast(d.calendar_week_start as date))
else  date_trunc('week',cast(d.calendar_week_start as date))
end as date)) / 7 ) +2 end as week_number
```

де дата [d.calendar_week_start] — це дата, скорочена до початку календарного тижня (понеділок): date_trunc('week', [date]).

Дані замовлення потрібно буде правильно об’єднати в таблицю дат, щоб гарантувати, що жоден тиждень не буде пропущено, якщо відсутні дані замовлення. У наступних прикладах календарний рік використовується як фінансовий рік.

```
create table mv_analytics_agg_order_history as
SELECT distinct
cast(d.calendar_week_start as date) as week_start
,cast(extract(year from cast(d.calendar_week_start as date))as bigint) as year, case
when extract(month from cast(d.calendar_week_start as date)) in (1,2,3) then 1
when extract(month from cast(d.calendar_week_start as date)) in (4,5,6) then 2
when extract(month from cast(d.calendar_week_start as date)) in (7,8,9) then 3
when extract(month from cast(d.calendar_week_start as date)) in (10,11,12) then 4
end as quarter, case
when date_diff('day',cast(date_trunc('year',cast(d.calendar_week_start as date)) as
date), cast(case when cast(extract(year from cast(d.calendar_week_start as date))as
bigint)  = cast(extract(year from date_trunc('week',cast(d.calendar_week_start as
date)))as bigint)  +1
then  date_trunc('month',cast(d.calendar_week_start as date))
else  date_trunc('week',cast(d.calendar_week_start as date)) end as date)) = 0 then 1
```

```
else (date_diff('day',cast(date_trunc('year',cast(d.calendar_week_start as date)) as
date), cast(case when cast(extract(year from cast(d.calendar_week_start as date))as
bigint)  = cast(extract(year from date_trunc('week',cast(d.calendar_week_start as
date)))as bigint)  +1
then  date_trunc('month',cast(d.calendar_week_start as date))
 else  date_trunc('week',cast(d.calendar_week_start as date)) end as date)) / 7 ) +2
end as week_number,
row_number() over( order by d.calendar_week_start asc) as row_number,
count(distinct case when d.calendar_week_start < cast(getdate()as date) then
o.order_external_id else null end) as number_of_orders,
case when d.calendar_week_start < cast(getdate()as date) then
case when sum(o.line_value_including_tax * o.line_item_qty) is null then 0
                    else sum(o.line_value_including_tax * o.line_item_qty)  end
          else 0 end    as total_value
      from analytics_datasets.dates d
      left join (select * from order_coredata_table where line_item_timestamp <
date_trunc('day',cast(getdate() as date))) o on cast(
date_trunc('day',o.line_item_timestamp) as date) = cast(d.the_date as date)
      where d.calendar_week_start > (select max(calendar_week_start) from
(select d.calendar_week_start, count(distinct o.order_external_id) count
from analytics_datasets.dates d
left join order_coredata_table o on cast( date_trunc('day',o.line_item_timestamp) as
date) = cast(d.the_date as date)
where d.the_date < cast(getdate() as date)
group by d.calendar_week_start order by calendar_week_start ) a where count = 0)
and d.the_date < date_trunc('week',dateadd('year',1,getdate()))
group by d.calendar_week_start
```

Це має призвести до такої таблиці (наприклад, Таблиця 2 – Mv_analytics_agg_order_history):

Таблиця 2 – Mv_analytics_agg_order_history

Номер року	Дата початку тижня	Номер тижня	Абсолютний номер рядка	Кількість продажів	Загальна вартість продажів
2022	1 січня 2022 року	1	1	90	421.06
2022	8 січня 2022 року			50	219.48
...	...	..	..	..	..
2022	31 грудня 2022 року	52	52	105	1328.72

Номер року: це рік початку фінансового року.

Дата початку тижня: це дата понеділка цього тижня.

Абсолютний номер рядка: це просто номер рядка таблиці. Тобто не скидається кожен рік

Загальна вартість продажів: це сума вартості всіх здійснених продажів.

Кількість продажів: це кількість розміщених замовлень.

Крок 2 Розрахунок тенденції та сезонності:

На цьому кроці ми намагаємося порівняти значення трендів із історичними показниками продажів за тиждень за тижнем і 12-місячним прогнозом.

Використовуючи 12-місячний огляд, обчислюється:

Використовуючи таблицю сукупної вартості замовлення по тижнях, обчислюється 4-тижневе ковзне середнє для кожного історичного тижня, використовуючи значення між 2-тижневим лагом і 1-тижневим випередженням, розділеним на змінну цілу різницю в році між кожним тижнем і поточним дата , k, тобто

```
select *, case when week_start< date_trunc('week',cast(getdate() as date))
then cast(AVG ( ALL cast( total_value as float) ) OVER ( partition by cast(cast(
```

```
datediff('week',getdate(),week_start) as float) / 52  as integer) ORDER BY
week_start , week_number  rows between 2 preceding and 1 following) as float )
                else null end as moving_average_total_value,
    cast(cast( datediff('week',getdate(),week_start) as float) / 52  as integer) as k
    from mv_analytics_agg_order_history ) a
    order by week_start;
```

Використовуючи цей стовпець ковзного середнього, обчислюється центроване ковзне середнє для кожного історичного тижня, використовуючи поточне та 1 наступне ковзне середнє:

```
    create table mv_analytics_agg_order_history_moving_avgs as
    SELECT
        *
        ,(moving_average_total_value + (lead(moving_average_total_value)
over (partition by k order by week_start) ))/2 as
centred_moving_average_total_value
    from (
     select *, case when week_start< date_trunc('week',cast(getdate() as date))
then cast(AVG ( ALL cast( total_value as float) ) OVER ( partition by cast(cast(
datediff('week',getdate(),week_start) as float) / 52  as integer) ORDER BY
week_start , week_number  rows between 2 preceding and 1 following) as float )
                else null end as moving_average_total_value
         ,cast(cast( datediff('week',getdate(),week_start) as float) / 52  as
integer) as k
    from mv_analytics_agg_order_history ) a
    order by week_start;
```

Таблиця проміжних результатів (Таблиця 3 – Mv_analytics_forecasting_history).

Таблиця 3 – Mv_analytics_forecasting_history

Номер року	Дата початку тижня	Номер тижня	Абсолютний номер рядка	Різниця цілого року до поточної дати, k	Загальна вартість продажів	Загальне значення ковзного середнього	Центроване ковзне середнє значення
2022	1 січня 2022 року	1	1	-1	421.06	320.27878…	302.2373…
2022	8 січня 2022 року			-1	219.48	284.196…	259.27458…
...	...	..	..	...	..	...	...
2022	31 грудня 2022 року	52	52	0	1328.72	1538.73342…	1219.34581…

Використовуючи центровані ковзні середні значення, ми тепер обчислюємо нахил і перетин ліній тренду цих значень:

```
create table mv_analytics_forecasting_trend as with trend_line as (
    SELECT k, slope_no_orders, slope_value, y_bar_no_orders_max –
(x_bar_max * slope_no_orders) as intercept_no_orders, y_bar_value_max –
(x_bar_max * slope_value) as intercept_value
    FROM (select distinct k, sum((x-x_bar) * (y_no_orders-y_bar_no_orders))
over (partition by k) / sum((x-x_bar) * (x-x_bar)) over (partition by k) as
slope_no_orders, max(x_bar)over (partition by k) as x_bar_max,
max(y_bar_no_orders) over (partition by k) as y_bar_no_orders_max, sum((x-
x_bar) * (y_value-y_bar_value)) over (partition by k) / sum((x-x_bar) * (x-x_bar))
```

over (partition by k) as slope_value, max(y_bar_value) over (partition by k) as y_bar_value_max

from (SELECT row_number as x, k, avg(row_number) over (partition by k order by week_start rows between unbounded PRECEDING and unbounded following) as x_bar, centred_moving_average_no_orders as y_no_orders, avg(centred_moving_average_no_orders) over(partition by k order by week_start rows between UNBOUNDED PRECEDING and unbounded following) as y_bar_no_orders, centred_moving_average_total_value as y_value, avg(centred_moving_average_total_value) over(partition by k order by week_start rows between UNBOUNDED PRECEDING and unbounded following) as y_bar_value from mv_analytics_agg_order_history_moving_avgs where week_start < date_trunc('week',cast(getdate() as date)))))

Це дає значення нахилу та перетину ліній тренду, які виглядатимуть, наприклад (Рисунок 2).

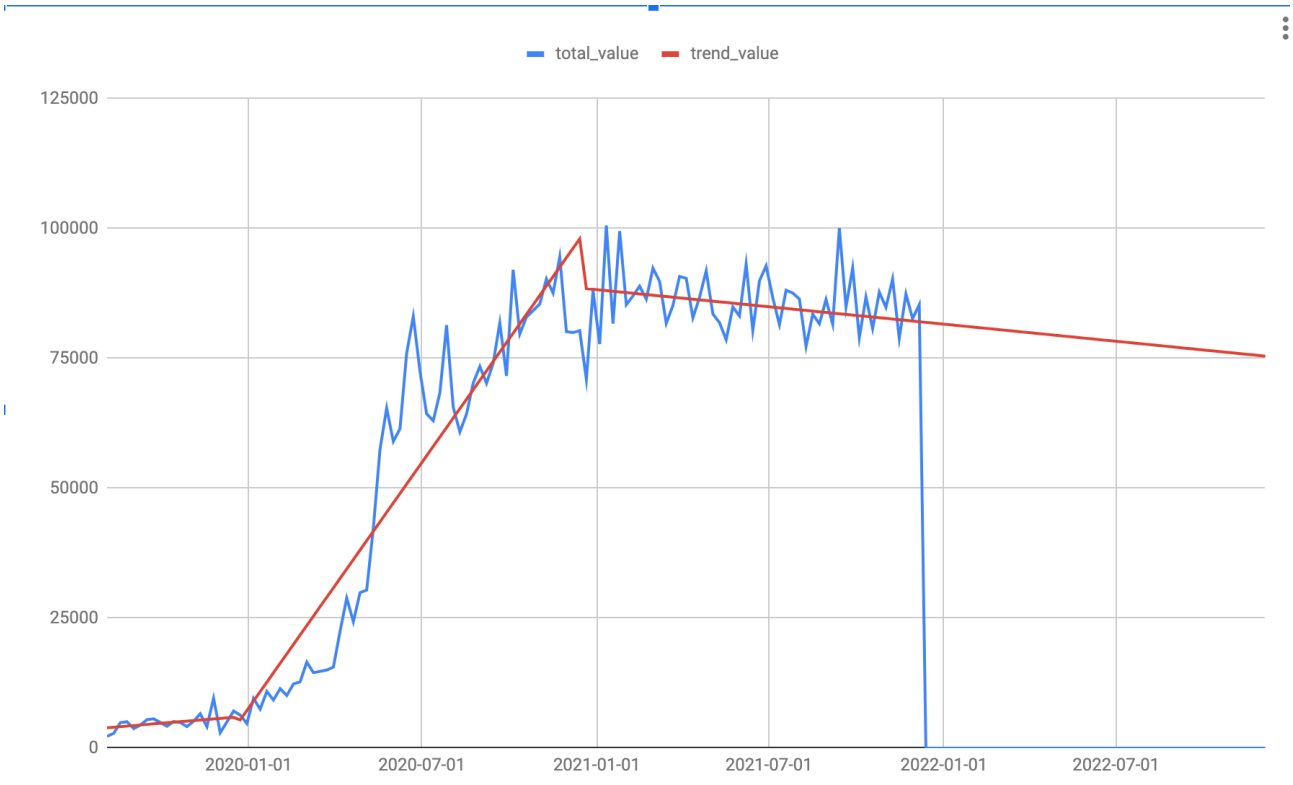


Рисунок 2

Який може розрахуватись для кожного історичного тижня, а тепер прогнозованого тижня. Різниця в історичних даних між фактичним значенням і лінією тренду вказується як значення сезонності (може бути позитивним або негативним).

Це має призвести до таблиці, яка містить таку інформацію (Таблиця *4* – Сезонність тижнівТаблиця 4).

Таблиця 4 – Сезонність тижнів

Номер року	Дата початку	Номер	Абсолютний номер	Різниця цілого року до поточної	Загальна вартість	Загальне значення ковзного	Центроване ковзне середнє значення	Сезонність продажів
2022	1 січня 2022 року	1	1	-1	421.06	320.27878…	302.2373…	888.403
2022	8 січня 2022 року			-1	219.48	284.196…	259.27458…	-214.29197000781323
…	…	..	..	…	..	…	…	…
2022	31 грудня 2022 року	52	52	0	1328.72	1538.73342…	1219.34581…	102.2938576

Інтеграція з Apache Spark: Забезпечення взаємодії із Apache Spark для ефективної обробки та аналізу даних на розподіленому кластері.

Крок 3 Розрахувати середню сезонність:

Використовуючи розраховані значення сезонності для всіх історичних даних порядку, потім обчислюється середнє значення сезонності для кожного тижня року. Наприклад:

```
create table mv_analytics_average_seasonality as
select week_number, avg(seasonality_value) as avg_seasonality_value
from mv_analytics_forecasting_trend
where week_start < date_trunc('week',cast(getdate() as date))
group by week_number;
```

У результаті має вийти наступна таблиця (Таблиця 5).

Таблиця 5 – Середня сезонність

Номер тижня	Середня вартість сезонних продажів
1	177.8939852546423
2	-852.6013331661761
...	...
53	-139.52665158699438

Крок 4 Створення потижневих прогнозів:

Тепер, використовуючи тренд і сезонність, спрогнозовується майбутні загальні тижневі продажі, гарантуючи суто позитивні прогнози.

```
create table mv_analytics_order_predictions as
SELECT ft.*, s.avg_seasonality_value, case when ft.trend_value +
s.avg_seasonality_value >= 0 then ft.trend_value + s.avg_seasonality_value
    else 0 end as prediction_value
    FROM mv_analytics_forecasting_trend ft
    left join mv_analytics_average_seasonality  s  on ft.week_number=
s.week_number
    order by week_start;
```

Це має призвести до таблиці, яка містить таку інформацію (Таблиця 6).

Таблиця 6 – Прогнозована вартість продажів

Номер року	Дата початку тижня	Номер тижня	Загальна вартість продажів	Тенденція загальної вартості продажів	Сезонність продажів	Прогнозована вартість продажів
2022	1 січня 2022 року	1	421.06	1,100	888.40	700
2022	8 січня 2022 року		219.48	1,200	-214.291	867
...	...	..	..	...	...	...
2022	31 грудня 2022 року	52	1328.72	3,500	102.29	1420

Крок 5 Обчислення похибки:

Використовуючи історичні дані, обчислюється верхня та нижня межі, які з часом збільшуються, для прогнозу.

Якщо σ^h позначає стандартне відхилення h-крокового розподілу прогнозу, а σ є залишковим стандартним відхиленням, тоді використовуються наступні вирази.

Сезонні прогнози:

$$\sigma^h = \sigma^{\sqrt{k}+1}$$

де k — ціла частина $\frac{(h-1)}{m}$, а m — сезонний період.

Прогнози тенденцій:

$$\sigma^h = \sigma^{\sqrt{h}\times(1+\frac{h}{T})}$$

де T – поточний час

Це дає:

$$\sigma^h = \sigma^{\sqrt{h}\times(1+\frac{h}{T})\times\sqrt{k}+1}.$$

Отже, нижня та верхня межі для передбачення y на момент часу $T + h$ відповідно дорівнюють $y^{\pm c\sigma^h}$, де множник c залежить від ймовірності покриття, наприклад припущення нормального розподілу залишків дає $c = 1{,}28$ для 80% інтервалів.

Ці значення можуть бути обчислені за допомогою таблиці виведення прогнозів (mv_analytics_order_predictions), наприклад:

```
create table mv_analytics_predictions_error as
with residual_stddev as (
SELECT stddev_samp(residual) as residual_sample_stddev
from (SELECT p.week_start, p.year, p.week_number, p.total_value,
p.prediction_value, p.prediction_value – p.total_value as residual
FROM mv_analytics_order_predictions p
where p.week_start < date_trunc('week',getdate()))), multipliers as (
SELECT *, stddev_multiplier_naive_seasonal * stddev_multiplier_drift as
stddev_multiplier
from (SELECT p.week_start, p.year, p.week_number, p.row_number,
p.total_value, p.prediction_value, datediff('week',getdate(),week_start) +1 as step,
cast(cast( datediff('week',getdate(),week_start) as float) / 52  as integer) as k, case
when cast(cast( datediff('week',getdate(),week_start) as float) / 52  as integer)  >=0
then sqrt(cast(cast(cast( datediff('week',getdate(),week_start) as float) / 52  as
integer) + 1 as float))
else null end as stddev_multiplier_naive_seasonal, case when
cast(datediff('week',getdate(),week_start) +1 as float) * cast( 1  +
(cast(datediff('week',getdate(),week_start) +1 as float) / (select
```

```
cast(max(row_number) as float) from  mv_analytics_order_predictions where
week_start < date_trunc('week',getdate()))  ) as float) >= 0 then
                        sqrt(cast(datediff('week',getdate(),week_start) +1 as float) *
cast( 1  + (cast(datediff('week',getdate(),week_start) +1 as float) / (select
cast(max(row_number) as float) from  mv_analytics_order_predictions where
week_start < date_trunc('week',getdate()))  ) as float))
        else null end as stddev_multiplier_drift
        from mv_analytics_order_predictions p))
        SELECT p.week_start, p.year, p.week_number, p.total_value,
p.prediction_value, case when p.prediction_value – (1.28 * (select
residual_sample_stddev from residual_stddev) *
m.stddev_multiplier_naive_seasonal ) >= 0
                then p.prediction_value – (1.28 * (select residual_sample_stddev
from residual_stddev) * m.stddev_multiplier_naive_seasonal )
                else 0 end as lower_bound, case when p.prediction_value + (1.28 *
(select residual_sample_stddev from residual_stddev) *
m.stddev_multiplier_naive_seasonal ) >= 0
                then p.prediction_value + (1.28 * (select residual_sample_stddev
from residual_stddev) * m.stddev_multiplier_naive_seasonal )
                else 0 end as upper_bound, getdate() as forecast_run_timestamp,
case when extract (dow from getdate()) = 1 then 'Mon' when extract (dow from
getdate()) = 5 then 'Fri'
        else null end as forecast_run_day
        from  mv_analytics_order_predictions p
        left join multipliers m on p.row_number = m.row_number
        order by week_start;
```

Тут спочатку обчислюється вибіркове стандартне відхилення залишків історичних даних. (Залишок — це різниця між прогнозованим і фактичним значенням).

Потім для кожного кроку прогнозу обчислюється множник

$$\sqrt{h} \times (1 + \frac{h}{T}) \times \sqrt{k} + 1,$$

а потім верхня та нижня межі.

Це має призвести до такої таблиці (Таблиця 7):

Таблиця 7 – Mv_analytics_predictions_error

Номер року	Дата початку тижня	Номер тижня	Загальна вартість продажів	Тенденція загальної вартості продажів	Прогнозована вартість продажів	Нижня межа передбачення	Верхня межа передбачення
2022	1 січня 2022 року	1	421.06	1,100	700	350	1200
2022	8 січня 2022 року		219.48	1,200	867	390	1320
...	...	..	..	...	...	...	...
2022	31 грудня 2022 року	52	1328.72	3,500	1420	690	1730

Це дасть змогу побудувати графік, що відображатиме майбутній прогноз прогнозу з верхньою та нижньою межами, наприклад (Рисунок *3*).

Крок 6 Створення поденних прогнозів на наступний тиждень:

Використовуючи дані про замовлення за останні 6 тижнів, розраховується середній розподіл продажів за тиждень.

Потім цей приблизний розподіл використовується для розподілу прогнозованої вартості продажів на наступний тиждень за днями.

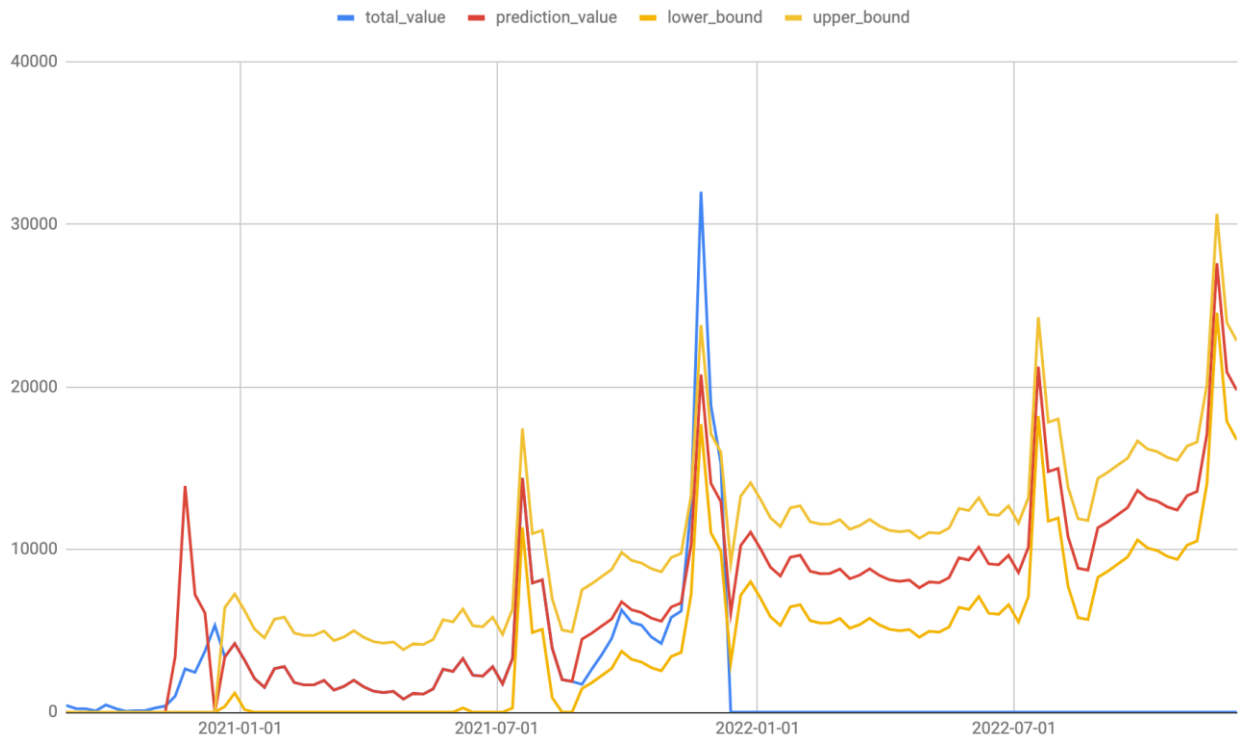


Рисунок 3

```
create table mv_analytics_forecast_week_breakdown as
with breakdown as (
SELECT distinct
    cast(d.the_date as date) as date
    ,d.day_name
    ,case when date_diff('day',cast(date_trunc('year',cast(d.the_date as date)) as date), cast(case when cast(extract(year from cast(d.the_date as date))as bigint)  = cast(extract(year from date_trunc('week',cast(d.the_date as date)))as bigint)  +1
                then  date_trunc('month',cast(d.the_date as date))
                else  date_trunc('week',cast(d.the_date as date))
                end as date))
        = 0 then 1
```

```
                else (date_diff('day',cast(date_trunc('year',cast(d.the_date as date))
as date), cast(case when cast(extract(year from cast(d.the_date as date))as bigint)  =
cast(extract(year from date_trunc('week',cast(d.the_date as date)))as bigint)  +1
                        then  date_trunc('month',cast(d.the_date as date))
                        else  date_trunc('week',cast(d.the_date as date))
                        end as date))
                        / 7 ) +2
                end  as week_number
        ,count(distinct o.order _id ) as number_of_orders
        ,cast(sum(o.value_including_tax * o. qty) as float)as total_value
    from analytics_datasets.dates d
    left join order _table o on  cast( date_trunc('day',o.timestamp) as date) =
cast(d.the_date as date)
    where
    ( d.the_date >= dateadd('week',-6,getdate())
    and d.the_date <= getdate())
    group by d.the_date, d.day_name
    )
    , proportions as (
    select distinct
        day_name
        ,cast(sum(total_value) over (partition by day_name) as float) /
cast((sum(total_value) over () ) as float) as proportion_total_value
    from breakdown
    )

    SELECT
        case when day_name= 'Monday' then  week_start
        when day_name= 'Tuesday' then  dateadd('day',1,week_start)
        when day_name= 'Wednesday' then  dateadd('day',2,week_start)
```

```
when day_name= 'Thursday' then  dateadd('day',3,week_start)
when day_name= 'Friday' then  dateadd('day',4,week_start)
when day_name= 'Saturday' then  dateadd('day',5,week_start)
when day_name= 'Sunday' then  dateadd('day',6,week_start)
end as date
 ,day_name
 ,cast(proportion_total_value *  prediction_value  as decimal(18,2)) as
prediction_day_value
 from (select cast(week_start as date) as week_start,prediction_value from
mv_analytics_order_predictions where cast(week_start as date)  =
cast(date_trunc('week',getdate()) as date)) dates
 cross join proportions p
 order by DATE
```

У результаті має вийти наступна таблиця (Таблиця 8).

Таблиця 8 – Прогнозована вартість продажів

Дата	Назва дня	Прогнозна вартість продажів
2022-12-13	Понеділок	1305.26
2022-12-14	Вівторок	737.06
...	...	...
2022-12-19	Неділя	969.53

3.2.2 Щотижневі, квартальні та повні прогнози

Щоб розрахувати прогноз за весь рік і порівняти його із загальним обсягом продажів за попередній рік, спочатку об’єднуються та підсумовуються загальні продажі за минулі тижні поточного року та прогнозовані значення тижня для решти року.

Потім це можна порівняти з агрегованими сумами за попередні роки.

```
SELECT *, full_year_total – lag(full_year_total) over (order by year) as
absolute_growth, ((full_year_total – lag(full_year_total) over (order by year))/
lag(full_year_total) over (order by year)) as percentage_growth
    from (SELECT year, cast(sum( combined_year_forecast) as decimal(18,2))
as full_year_total
        from (SELECT *, case when week_start < date_trunc('week',getdate())
then total_value
    when week_start >= date_trunc('week',getdate()) then prediction_value
    else NULL
    end as combined_year_forecast
    from mv_analytics_predictions_error)
        where year <= extract(year from getdate()) group by year)
    order by year;
```

Результатом цього має бути, наприклад (Таблиця 9).

Таблиця 9 – Прогнозування за роками

Рік	Усього за рік	Абсолютне річне зростання	Річний приріст у відсотках
2020	109323.43		
2021	2818597.94	2709274.51	24.782194539633452774
2022	4430671.92	1612073.98	0.571941800255484469

Щоб обчислити квартальні прогнози, загальну суму об’єднаних загальних продажів за історичні тижні в поточному році та прогнозовані значення тижня

для решти року агрегується за кварталами (наведені за фінансовим роком організації) і підсумовується. Наприклад:

```
SELECT *, quarter_total – lag(quarter_total) over (order by quarter) as
absolute_growth, ((quarter_total – lag(quarter_total) over (order by quarter))/
lag(quarter_total) over (order by quarter)) as percentage_growth
from (SELECT quarter, cast(sum(combined_year_forecast) as decimal(18,2))
as quarter_total, case when quarter < case when extract(month from getdate()) in
(1,2,3) then 1
when  extract(month from getdate()) in (4,5,6) then 2
when extract(month from getdate()) in (7,8,9) then 3
when  extract(month from getdate()) in (10,11,12) then 4
end then 'historical'
else 'forecast' end as historical_or_forecast
from (SELECT *, case when extract(month from week_start) in (1,2,3)
then 1
when  extract(month from week_start) in (4,5,6) then 2
when extract(month from week_start) in (7,8,9) then 3
when  extract(month from week_start) in (10,11,12) then 4
end as quarter, case when week_start < date_trunc('week',getdate()) then
total_value
when week_start >= date_trunc('week',getdate()) then prediction_value
else NULL
end as combined_year_forecast
from mv_analytics_predictions_error
where year = extract(year from getdate())
order by week_start)
group by  quarter
) order by quarter;
```

Це призводить до вихідної таблиці (Таблиця 10).

Таблиця 10 – Квартальні прогнози

Квартал	Загальний обсяг продажів за квартал	Історичний або прогноз	Абсолютне річне зростання	Річний приріст у відсотках
1	1146717.38	історичний		
2	1119910.71	історичний	-26806.67	-0.023376876000606182
3	1117678.16	історичний	-2232.55	-0.001993507143082862
4	1046365.67	прогноз	-71312.49	-0.063804136604047089

3.2.3 Щотижневий рейтинг продажів

Для обчислення рейтингу продажів за останні 20 тижнів, по-перше, дані про замовлення агрегуються за тижнями.

Потім останні 20 тижнів упорядковуються за підсумковим загальним значенням desc, щоб отримати ранг 1 як тиждень із найвищим загальним обсягом продажів, а 20 – з найнижчим.

```
with totals as (
```

```
select DISTINCT d.calendar_week_start, count(distinct o.order_id) as
total_orders, cast(sum(o.line_value_including_tax * o.line_item_qty) as
decimal(18,2))as total_value
from analytics_datasets.dates d
left join order o on cast(date_trunc('day',o.line_item_timestamp) as date) =
cast(d.the_date as date)
where d.calendar_week_start >= date_trunc('week',dateadd('week',-
21,getdate()))
and d.calendar_week_start < date_trunc('week',getdate())
and o.line_item_timestamp >= date_trunc('week',dateadd('week',-
21,getdate()))
and o.line_item_timestamp < date_trunc('week',getdate())
group by d.calendar_week_start
order by d.calendar_week_start)
SELECT *, rank() over ( order by total_value DESC) as RANK
from totals order by calendar_week_start
```

Вихідна таблиця для цього виглядає так (Таблиця 11).

Таблиця 11 – Рейтинг продажів за 20 тижнів

Початок тижня	Всього замовлень	Загальна вартість	Ранг
2022-07-26	2380	87562.34	5
2022-08-02	2435	86394.57	8
2022-08-09	2237	77255.86	20
...	...	...	...
2022-12-06	2168	85295.05	10

3.2.4 Профілювання клієнтів

Для профілювання клієнтів спочатку групуються дані за користувачами, визначається найстарший та найновіший час замовлень для кожного користувача. Далі обчислюються різні метрики, такі як загальна кількість замовлень, сума вартості замовлень, кількість замовлень за останні квартали тощо.

Для агрегації та визначення різних характеристик для окремих сегментів користувачі фільтрубться за умовами сегменту, визначається їхня активність та лояльність, розраховується середню вартість та кількість замовлень, формується список найпопулярніших товарів у сегменті та аналізується перше замовлення для кожного користувача у сегменті. Для цього використовується агрегацію даних, фільтрація за певними умовами та розрахунки характеристик для кожного сегменту, щоб підготувати дані для подальшої обробки або аналізу веб-додатком. Це допомагає зрозуміти та класифікувати користувачів замовлень для вживання подальших заходів або вироблення стратегій бізнесу.

Користувачі розбиваються на три категорії: активні, лояльні та користувачі під ризиком в залежності від загальної вартості їхніх замовлень: високоцінний, середньоцінний та низькоцінний сегменти. Виконуються різні обчислення, такі як кількість замовлень, середній час між замовленнями, середній обсяг замовлення та інші метрики для кожного сегменту. Крім того, класифікуються користувачі як активні, лояльні чи під ризиком на основі їхньої активності та інших показників.

Також корисним буде пошук топ п’яти найпопулярніших продуктів, які купують клієнти в сегменті. Для кожного сегмента обчислюється загальна кількість клієнтів у цьому сегменті, які зробили принаймні одне успішне замовлення для кожного продукту на момент запуску додатку. Загальні витрати на кожен продукт за сегментами також розраховуються за сумою вартості продукту включаючи податок помножений на кількість позицій продукту.

Продукти впорядковано для кожного сегмента за загальною кількістю окремих клієнтів за спаданням, а потім за загальними витратами за спаданням. 1 – рейтинг, наданий продукту з найбільшою кількістю клієнтів, які придбали

тощо. П’ять найкращих продуктів зберігаються, а також надаються їхні ідентифікатори та назви. У разі нічиєї в будь-якому з місць перемагає продукт з найбільшим доходом.

Для кожного з п’яти найпопулярніших продуктів у кожному сегменті кількість клієнтів, які здійснили покупку, надається як відсоток від загальної кількості клієнтів у цьому сегменті.

Провідний продукт для сегмента визначається як найпоширеніший продукт, який купують у першому порядку клієнти в сегменті. Таким чином, для кожного сегмента підраховується кількість окремих клієнтів на продукт, з їхнім провідним продуктом як цей продукт. Крім того, загальні витрати на продукт у цих покупках першого замовлення підсумовуються. Потім продукти впорядковуються за цією загальною кількістю клієнтів за спаданням, а потім за спаданням загальних витрат. Найкращий продукт – це той, який має найбільшу кількість клієнтів і є основним продуктом, і в разі нічиєї після того, як усі продукти першої покупки обчислюються для всіх клієнтів у сегменті, виграє продукт, який приніс найбільший дохід.

3.2.5 Хмара слів

Створення хмари слів – це складний процес, що включає декілька етапів. Перш за все, назви товарів проходять через процес "санітарізації" для видалення небажаних символів або знаків, а також для стандартизації формату. Після цього вони розбиваються на окремі слова або токени, які стають частинами хмари слів.

Створення хмари слів включає аналіз тексту, виділення унікальних слів та їхньої частоти вживання. Однак це лише перший крок. Для виявлення повторюваних слів, застосовується Stanford NLP (Natural Language Processing). Цей інструмент допомагає виявити не лише частоту, а й контекст, в якому використовується кожне слово. На основі цього аналізу виокремлюються ключові слова чи фрази, які найчастіше зустрічаються у назвах товарів, що становить основу для створення хмари слів.

Такий аналіз допомагає візуалізувати частотність слів у формі хмари (Рисунок *24*), де більш часті слова або фрази відображаються більшими за розміром, надаючи швидкий узагальнений огляд тематики або популярних позицій товарів.

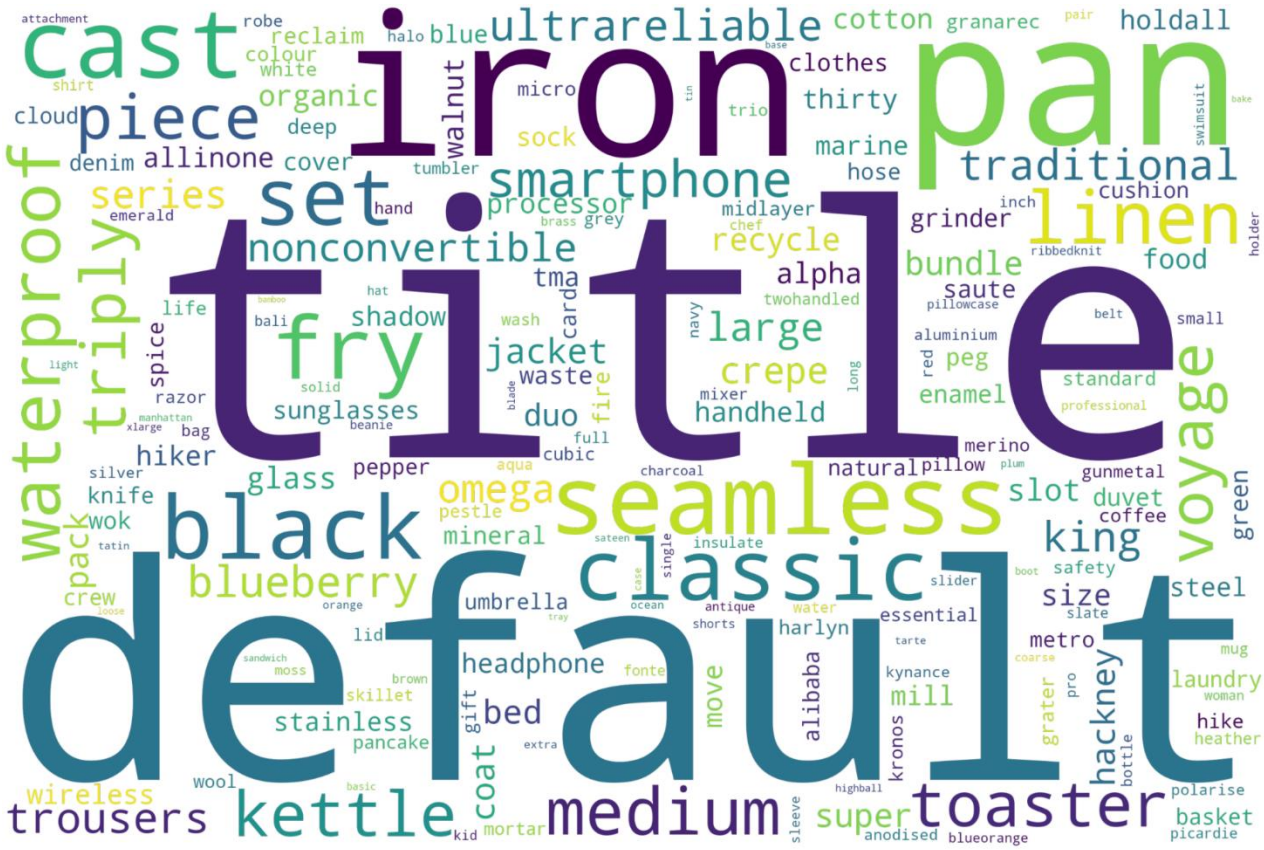


Рисунок 4

3.2.6 Топ клаймбер та фоллер, мувер та шейкер (*ref*. Top Climber and Faller / Movers and Shakers)

За останні 2 тижні, тобто за останні 14 днів розраховується загальна кількість проданих одиниць товару за тиждень для замовлень зі статусом “сплачено”. Також розраховується загальна вартість цих замовлень на один продукт за тиждень. Це не обмежується продуктами, які “доступні” лише зараз.

Для кожного тижня продукти ранжуються за кількістю проданих одиниць за спаданням, потім за вартістю за спаданням, а потім за кількістю окремих замовлень за спаданням.

Для всіх цих продуктів дані агрегуються, щоб підсумувати загальну кількість проданих товарів і загальну вартість усіх їхніх замовлень. Ці значення надають загальні підсумки.

Список продуктів обмежується 10 найкращими продуктами за поточний тиждень. Для цих продуктів також наведено їхню позицію за останній тиждень.

Для продуктів, які були за межами топ-10, їхні рядки незалежно агрегуються, щоб підсумувати загальну кількість проданих товарів і підсумувати загальну вартість цих замовлень. Ці значення надають загальні значення “Інші”.

Зміна позиції для 10 найпопулярніших продуктів обчислюється як позиція за останній тиждень мінус позиція поточного тижня. Так, наприклад, для продукту, який минулого тижня займав позицію 9, а цього тижня – 4, його зміна позиції становитиме +5.

Продукти, які є новими цього тижня, не мають попередньої позиції минулого тижня, тому їх позиція змінилася на “Новий”.

Ігноруючи нові продукти, решта 10 найпопулярніших продуктів потім знову ранжуються, цього разу за зміною позиції за спаданням, потім за кількістю проданих товарів за поточний тиждень за спаданням, потім за значенням доходу за поточний тиждень за спаданням, а потім за спаданням кількість окремих замовлень на поточному тижні за спаданням.

Дані муверс (*ref.* Movers) та шейкерс (*ref.* Shakers)— це вихідні дані, що містять для 10 найпопулярніших продуктів назву продукту, позиційні зміни порівняно з попереднім тижнем, загальну кількість продуктів, замовлених цього поточного тижня, і загальну вартість цих замовлень.

3.2.7 Можливості підписки

Це призначено для того, щоб висвітлити для користувачів продукти, які купує одна особа кілька разів. Знову в списку топ 10. Набір даних для цього охоплює три місяці.

Це товари, які один і той самий клієнт купував більше одного разу за останні три місяці.

Для цього фільтруються замовлення за останні три місяці, і для кожного клієнта кожному з його замовлень надається порядковий номер замовлення для кожного продукту.

Для кожного клієнта та кожного продукту розраховується загальна кількість замовлень і загальна сума витрат.

Замовлення також призначаються як «повторні» або «перші» замовлення. Замовлення з порядковим номером 1 є «першим» замовленням, усі наступні замовлення є «повторними». Потім кількість повторних замовлень і перших замовлень розраховується для кожного клієнта та продукту. Вартість цих повторних замовлень і перших замовлень розраховується для клієнта та продукту. Потім вони підсумовуються для всіх клієнтів, щоб отримати кількість усіх замовлень, кількість повторних замовлень, кількість перших замовлень, загальну вартість усіх замовлень, загальну вартість повторних замовлень і загальну вартість перших замовлень, усіх за продукт.

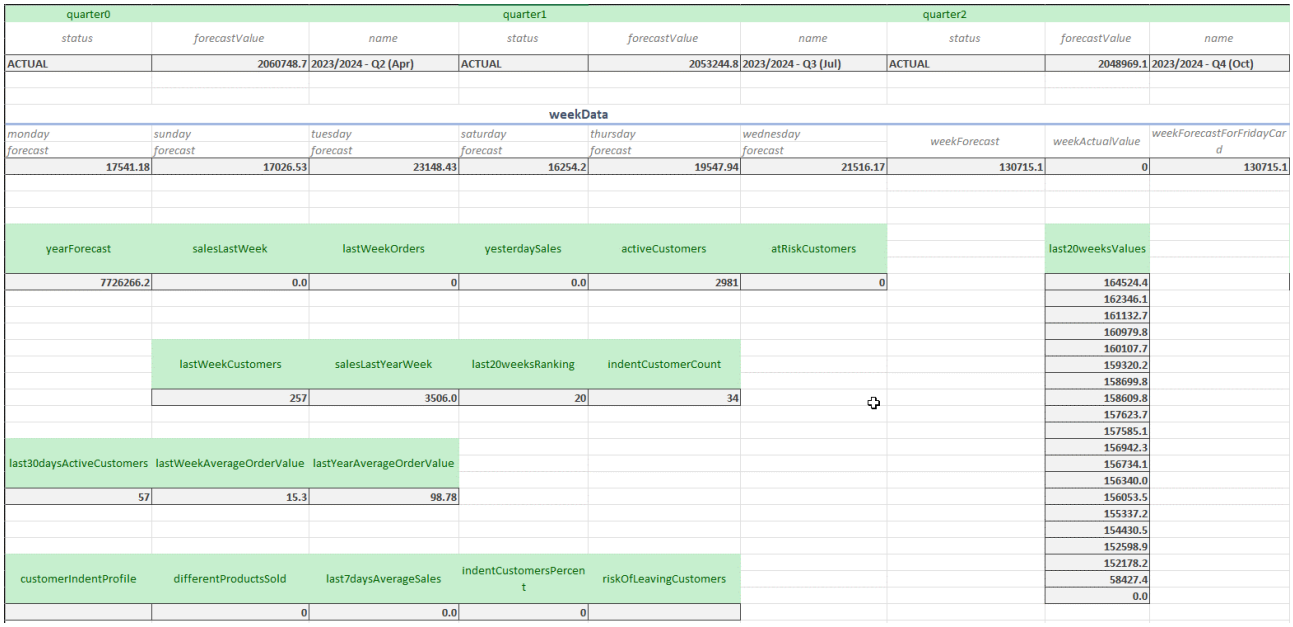
Потім для кожного продукту рейтинг повторень в відсотковому відношені обчислюється як кількість повторних замовлень, поділена на кількість усіх замовлень (* 100). Продукти впорядковуються за цією частотою повторень і загальною вартістю повторних замовлень, обидва за спаданням.

Десять найкращих продуктів відображаються як десять найкращих можливостей підписки.

3.3 Візуалізація даних

Результатом роботи програми є JSON-файл, що містить значний обсяг даних, що може бути імпортований та оброблений в різних аналітичних інструментах, таких як Excel. Цей файл містить ключову інформацію про онлайн-торгівлю, включаючи дані про замовлення, користувачів та товари.

Даний JSON-файл легко імпортується та обробляється в Excel, що дозволяє проводити різноманітний аналіз та візуалізацію даних. Після імпортування файлу, можна проводити такі операції (рис. 5).



quarter0			quarter1			quarter2		
status	*forecastValue*	*name*	*status*	*forecastValue*	*name*	*status*	*forecastValue*	*name*
ACTUAL	2060748.7	2023/2024 - Q2 (Apr)	ACTUAL	2053244.8	2023/2024 - Q3 (Jul)	ACTUAL	2048969.1	2023/2024 - Q4 (Oct)

weekData

monday forecast	*sunday forecast*	*tuesday forecast*	*saturday forecast*	*thursday forecast*	*wednesday forecast*	*weekForecast*	*weekActualValue*	*weekForecastForFridayCard*
17541.18	17026.53	23148.43	16254.2	19547.94	21516.17	130715.1	0	130715.1

yearForecast	salesLastWeek	lastWeekOrders	yesterdaySales	activeCustomers	atRiskCustomers	last20weeksValues
7726266.2	0.0	0	0.0	2981	0	164524.4
						162346.1
						161132.7
						160979.8
						160107.7
						159320.2
						158699.8
						158609.8
						157623.7
						157585.1
						156942.3
						156734.1
						156340.0
						156053.5
						155337.2
						154430.5
						152598.9
						152178.2
						58427.4
						0.0

lastWeekCustomers	salesLastYearWeek	last20weeksRanking	indentCustomerCount
257	3506.0	20	34

last30daysActiveCustomers	lastWeekAverageOrderValue	lastYearAverageOrderValue
57	15.3	98.78

customerIndentProfile	differentProductsSold	last7daysAverageSales	indentCustomersPercent	riskOfLeavingCustomers
	0	0.0	0	

Рисунок 5

В Excel є змога створювати різноманітні звіти за допомогою вбудованих функцій аналізу даних. Легко проводити фільтрацію та сортування даних для отримання потрібної інформації.

Ці інструменти у поєднанні з отриманими даними забезпечують зручний та швидкий аналіз даних онлайн-торгівлі для прийняття ефективних управлінських рішень.

ВИСНОВКИ

У результаті цієї роботи була розроблена та впроваджена система, яка забезпечує необхідність збору та аналізу даних онлайн-торгівлі. Вона спрощує процеси прийняття управлінських рішень та оптимізує бізнес-процеси завдяки можливості швидкого доступу до даних.

Ця робота спрямована на вивчення аналітичного паралічу в бізнесі та розробку веб-додатку для збору, аналізу та візуалізації даних онлайн-торгівлі. Основним результатом стала створена система, яка інтегрує Apache Spark для обробки великих обсягів даних, що дозволяє зручний доступ до інструментів аналізу та оптимізації даних у сфері онлайн-торгівлі.

У ході розробки проекту було здійснено:

Розгортання середовища: Співставлення вимог до середовища та розгортання його через Docker, у тому числі налаштування та інтеграція із Apache Spark для обробки великих обсягів даних.

Проектування бази даних та моделі даних: Розроблення оптимальної моделі бази даних, відповідної вимогам та забезпечення нормалізації та оптимізації його структури.

Створення функціональності обробки та аналізу даних: Реалізація алгоритмів обробки, агрегації та аналізу великих обсягів даних, інтеграція з Apache Spark для виконання обчислень.

Розробка інтерфейсу користувача та візуалізація даних: Створення інтерфейсу для зручного доступу до функцій обробки даних та візуалізації результатів у вигляді графіків та діаграм.

Тестування та оцінка результатів: Проведення тестувань та оцінка ефективності додатку, підтвердження відповідності результатів вимогам.

Загальним результатом роботи є розроблена система для обробки та аналізу даних, здатна працювати з великими обсягами інформації та надати користувачам зручний інструмент для її використання та аналізу.

Отримані результати можуть бути використані для поліпшення аналітичних інструментів в різних сферах бізнесу, а також для подальшого вдосконалення аналітичних моделей та управлінських рішень в онлайн-торгівлі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Apache Spark. Apache Spark – Unified Analytics Engine for Big Data. URL : https://spark.apache.org/ (дата звернення : 13.07.2023)

2. Google Analytics. Google Analytics Solutions. URL : https://analytics.google.com/ (дата звернення : 09.07.2023)

3. Google Sheets. URL : https://docs.google.com/spreadsheets (дата звернення : 25.08.2023)

4. Hadoop. Apache Hadoop – Open Source Ecosystem. URL : https://hadoop.apache.org/ (дата звернення : 01.08.2023)

5. Hibernate. Official Hibernate ORM Framework. URL : https://hibernate.org/ (дата звернення : 02.07.2023)

6. Java Programming Language. Official Java Website. URL : https://www.java.com/ (дата звернення : 02.07.2023)

7. Microsoft Exel. URL : https://www.microsoft.com/uk-ua/microsoft-365/excel (дата звернення : 10.08.2023)

8. PostgreSQL. Official PostgreSQL Database Management System. URL : https://www.postgresql.org/ (дата звернення : 02.07.2023)

9. Python. Official Website of Python Programming Language. URL : https://www.python.org/ (дата звернення : 07.07.2023)

10. R Programming. Comprehensive R Archive Network. URL : https://cran.r-project.org/ (дата звернення : 07.07.2023)

11. Spring Framework. Official Spring Framework. URL : https://spring.io/ (дата звернення : 02.07.2023)

12. Tableau. Tableau Software – Data Visualization. URL : https://www.tableau.com/ (дата звернення : 10.07.2023)

ДОДАТОК А

Код реалізації DataProcessingJob

```
package dpl.processing.job;

import dpl.processing.model.*;
import dpl.processing.job.context.ProcessJobContext;
import dpl.processing.service.AggregatedDataService;
import dpl.processing.service.spark.IPostgresSparkDataService;
import dpl.processing.utils.ProductUtils;
import dpl.processing.utils.ScalaUtils;
import dpl.processing.vo.YearChartPoint;
import dpl.processing.vo.holder.ForecastCalculationHolder;
import dpl.processing.vo.holder.ProductDataHolder;
import dpl.processing.vo.wrapper.row.EmptyRowWrapper;
import dpl.processing.vo.wrapper.row.RowWrapper;
import dpl.processing.vo.wrapper.row.SimpleRowWrapper;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang.StringUtils;
import org.apache.spark.sql.Column;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.expressions.Window;
import org.apache.spark.sql.expressions.WindowSpec;
import org.apache.spark.sql.types.DataTypes;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Component;
import scala.collection.Seq;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.nio.charset.StandardCharsets;
```

```
import java.sql.Timestamp;
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.LongStream;

import static dpl.processing.constants.PostgresConstants.*;
import static dpl.processing.utils.DateUtils.*;
import static dpl.processing.utils.ProductUtils.FIRST_ORDER_ID_FIELD;
import static dpl.processing.utils.ScalaUtils.toSeq;
import static java.time.temporal.TemporalAdjusters.firstDayOfYear;
import static org.apache.spark.sql.functions.*;

@Slf4j
@Component
public class DataProcessingJob {
    protected static final String WEEK_DATE_FIELD = "week_date";
    protected static final String DAY_DATE_FIELD = "day_date";

    protected static final String OLDEST_ORDER_TIMESTAMP_FIELD =
"oldest_order_timestamp_5y_cut";
    protected static final String OLDEST_ORDER_TIMESTAMP_FIELD_W_CUT
= "oldest_order_timestamp";
    protected static final String JOB_DATE_FIELD = "job_date";
    public static final String TOTAL_VALUE_FIELD = "total_value";
    protected static final String CUSTOMER_TOTAL_VALUE_FIELD =
"customer_total_value";
    protected static final String TOTAL_VALUE_YEAR_FIELD =
"year_total_value";
    protected static final String TOTAL_VALUE_QUARTER_FIELD =
"quarter_total_value";
    protected static final String TOTAL_VALUE_60_DAYS_FIELD =
"d60_total_value";
    protected static final String TOTAL_VALUE_90_DAYS_FIELD =
"d90_total_value";
    protected static final String TOTAL_VALUE_30_DAYS_FIELD =
"d30_total_value";
    protected static final String TOTAL_VALUE_WEEK_FIELD =
"last_week_total_value";

    protected static final String AVG_BY_DAY_WEEK_FIELD =
"avg_by_day_week_value";
```

```
    protected static final String YESTERDAY_TOTAL_VALUE_FIELD =
"yesterday_total_value";
    protected static final String LAST_WEEK_ORDER_NUMBER_FIELD =
"last_week_num_of_orders";
    protected static final String YESTERDAY_ORDER_NUMBER_FIELD =
"yesterday_num_of_orders";
    protected static final String LAST_WEEK_ANG_ORDER_VALUE_FIELD =
"last_week_avg_order_value";
    protected static final String YESTERDAY_ANG_ORDER_VALUE_FIELD =
"yesterday_avg_order_value";

    protected static final String QUARTER2_LOWER_BOUND =
"q2_lower_bound";
    protected static final String QUARTER3_LOWER_BOUND =
"q3_lower_bound";
    protected static final String QUARTER4_LOWER_BOUND =
"q4_lower_bound";

    protected static final String ORDERS_COUNT_QUARTER1 =
"orders_count_quarter_1";
    protected static final String ORDERS_COUNT_QUARTER2 =
"orders_count_quarter_2";
    protected static final String ORDERS_COUNT_QUARTER3 =
"orders_count_quarter_3";
    protected static final String ORDERS_COUNT_QUARTER4 =
"orders_count_quarter_4";

    protected static final String ENGAGEMENT_FIRST_TIMESTAMP_FIELD =
"engagement_first_timestamp";
    protected static final String ENGAGEMENT_LAST_TIMESTAMP_FIELD =
"engagement_last_timestamp";
    protected static final String ENGAGEMENT_EVENTS_COUNT =
"engagement_events_count";

    protected static final String LAST_30_DAYS_ACTIVE_CUSTOMERS =
"last_30_days_active_customers";
    protected static final String LAST_30_DAYS_LOST_CUSTOMERS =
"last_30_days_lost_customers";
    protected static final String RISK_OF_LEAVING_CUSTOMERS =
"risk_of_leaving_customers";
    protected static final String LOYAL_RISK_OF_LEAVING_CUSTOMERS =
"loyal_risk_of_leaving_customers";

    protected static final String IS_LOYAL_FIELD = "is_loyal";
    public static final String IS_ACTIVE_FIELD = "is_active";
```

```
    protected static final String IS_AT_RISK_FIELD = "is_at_risk";

    public static final String ORDER_NUMBER_FIELD = "num_of_orders";
    public static final String AVG_ORDER_VALUE_FIELD = "avg_order_value";
    public static final String ITEMS_NUMBER_FIELD = "num_of_items";
    public static final String DIFF_PRODUCTS_NUMBER_FIELD =
"num_of_diff_products";
    public static final String CUSTOMER_NUMBER_FIELD =
"num_of_customers";
    public static final String VALUE_FIELD = "value";

    private static final String WEEK_DAY_NUMBER = "day_of_week";
    private static final String WEEK_NUMBER_FIELD = "week_number";
    private static final String FORECAST_FIELD = "forecast";

    private static final String SPENT_RANK_FIELD = "spent_rank";
    private static final String ACTIVE_CUSTOMERS_VALUE =
"active_customers_last_year_value";
    private static final String TOTAL_CUSTOMERS_VALUE =
"total_customers_last_year_value";
    private static final String ACTIVE_CUSTOMERS = "active_customers";
    private static final String AT_RISK_CUSTOMERS_VALUE =
"at_risk_customers_last_year_value";
    private static final String AVG_CUSTOMER_SPEND_VALUE =
"avg_customer_spend_value";
    private static final String PRODUCT_IDS = "product_ids";
    private static final String FIRST_PRODUCT_IDS = "first_product_ids";
    private static final String ONE_ORDER_CUSTOMERS =
"one_order_customers";
    private static final String TOTAL_ORDER_NUMBER = "total_order_number";
    private static final String ORDER_FREQUENCY_FIELD = "order_frequency";
    private static final String ORDER_AVERAGE_VALUE_FIELD =
"order_average_value";

    protected static final String
REPEAT_COUNT_WEEK_BEFORE_LAST_FIELD =
"repeat_count_week_before_last";
    protected static final String
REPEAT_COUNT_PERCENT_WEEK_BEFORE_LAST_FIELD =
"repeat_count_percent_week_before_last";
    protected static final String COUNT_CUSTOMERS_FIELD =
"count_customers";
    protected static final String REPEAT_COUNT_FIELD = "repeat_count";
    protected static final String REPEAT_COUNT_PERCENT_FIELD =
"repeat_count_percent";
```

```
    protected static final String
REPEAT_VALUE_WEEK_BEFORE_LAST_FIELD =
"week_before_last_repeat_value";
    protected static final String REPEAT_VALUE_FIELD = "repeat_value";
    protected static final String TOTAL_ORDER_QTY_FIELD = "total_order_qty";
    protected static final String
TOTAL_ORDER_QTY_WEEK_BEFORE_LAST_FIELD =
"week_before_last_total_order_qty";
    protected static final String PRODUCT_NAME_ARR_FIELD =
"product_name_arr";
    protected static final String CHANGE_FIELD = "change";
    protected static final String PRODUCT_POSITION_FIELD =
"product_position";
    protected static final String
PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD =
"week_before_last_product_position";
    protected static final String TOTAL_VALUE_WEEK_BEFORE_LAST_FIELD
= "week_before_last_total_value";
    protected static final String FIRST_ORDER_VALUE_FIELD =
"first_order_value";
    protected static final String TOTAL_FIRST_ORDER_VALUE_FIELD =
"total_first_order_value";


    protected static final String LEFT_JOIN = "left";
    protected static final Seq<String> USER_ID_TO_JOIN =
toSeq(Collections.singletonList(USER_ID_FIELD));
    protected static final Seq<String> ORDER_CUSTOMER_ID_TO_JOIN =
toSeq(Collections.singletonList(ORDER_CUSTOMER_ID_FIELD));
    public static final List<String> ORDER_PAID_STATUSES =
Arrays.asList("PAID", "PARTIALLY_PAID", "PARTIALLY_REFUNDED");
    protected static final Column ORDER_PRODUCT_JOIN_COLUMN =
col(ORDER_PRODUCT_ID_FIELD)
            .equalTo(col(PRODUCT_ID_FIELD));

    protected static final Seq<String> PRODUCT_JOIN_COLUMN =
ScalaUtils.toSeq(Collections.singletonList(
            ORDER_PRODUCT_ID_FIELD));


    @Value("classpath:stopwords")
    private Resource stopWords;

    @Autowired
    protected IPostgresSparkDataService sparkDataService;
```

```
    @Autowired
    protected AggregatedDataService dataService;

//    public final void startJob() {
//        String sessionName = sparkDataService.getInfoSession().getName();
//        Dataset<Row> orders = sparkDataService.loadBaseTable(sessionName,
"testshop.orders");
//
//        System.out.println(orders.collectAsList());
//
//    }

    public final void startJob(ProcessJobContext context) {
        String sessionName = sparkDataService.getInfoSession().getName();

        LocalDateTime jobDate = context.getJobEntryTimestamp();

        log.trace("{} job: using '{}' spark session", getJobName(), sessionName);

        Dataset<Row> ordersDataSet =
sparkDataService.loadPurchaseDataForOrg(sessionName, context)
                .where(col(ORDER_TIMESTAMP).lt(Timestamp.valueOf(jobDate))

.and(upper(col(ORDER_PAYMENT_STATUS)).isin(ORDER_PAID_STATUSES
.toArray())))
                .withColumn(WEEK_DATE_FIELD, date_trunc("WEEK",
col(ORDER_TIMESTAMP)))
                .withColumn(TOTAL_VALUE_FIELD,
coalesce(col(LINE_VALUE_INCLUDING_TAX),
col(LINE_VALUE_EXCLUDING_TAX), lit(0))
                        .multiply(coalesce(col(ORDER_LINE_ITEM_QTY),
lit(1))).cast(DataTypes.DoubleType)
                );

        if (log.isTraceEnabled()) {
            log.trace("{} job: loaded ORDER dataset with {} rows", getJobName(),
ordersDataSet.count());
        }

        AggregatedData aggregatedData = startProcessingData(context, sessionName,
jobDate, ordersDataSet);

        ResultData resultData = new ResultData(new Date(), aggregatedData);

        dataService.saveData(resultData);
```

```
    }

    private AggregatedData startProcessingData(ProcessJobContext context, String
sparkSession, LocalDateTime jobDate, Dataset<Row> ordersDataSet) {
        AggregatedData aggregatedData = new AggregatedData();
        Column timestampColumn = col(ORDER_TIMESTAMP);

        log.trace("{} job: generating card header", getJobName());

        Dataset<Row> avgByDayOfWeekData = ordersDataSet

.filter(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusDays(7))))
                .withColumn(DAY_DATE_FIELD, date_trunc("DAY",
col(ORDER_TIMESTAMP)))
                .groupBy(DAY_DATE_FIELD)
                .agg(

sum(TOTAL_VALUE_FIELD).cast(DataTypes.DoubleType).as(AVG_BY_DAY_
WEEK_FIELD)
                )

.select(avg(col(AVG_BY_DAY_WEEK_FIELD)).as(AVG_BY_DAY_WEEK_FI
ELD));

        BigDecimal last7daysAverageSales = BigDecimal.valueOf(new
SimpleRowWrapper(avgByDayOfWeekData
                .collectAsList().get(0)).getDouble(AVG_BY_DAY_WEEK_FIELD,
0D));
        SimpleRowWrapper headerData = new
SimpleRowWrapper(ordersDataSet.agg(

                        sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE_FIELD),

sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusYear
s(1))), col(TOTAL_VALUE_FIELD)))
                                .as(TOTAL_VALUE_YEAR_FIELD),

sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusDays
(7))), col(TOTAL_VALUE_FIELD)))
                                .as(TOTAL_VALUE_WEEK_FIELD),

sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusDays
(1))), col(TOTAL_VALUE_FIELD)))
                                .as(YESTERDAY_TOTAL_VALUE_FIELD),
```

```
countDistinct(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusYears(1))), col(ORDER_ID_FIELD)))
                        .alias(ORDER_NUMBER_FIELD),

countDistinct(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusDays(7))), col(ORDER_ID_FIELD)))
                        .alias(LAST_WEEK_ORDER_NUMBER_FIELD),

countDistinct(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusDays(1))), col(ORDER_ID_FIELD)))
                        .alias(YESTERDAY_ORDER_NUMBER_FIELD)
            ).withColumn(AVG_ORDER_VALUE_FIELD,
col(TOTAL_VALUE_FIELD).divide(col(ORDER_NUMBER_FIELD)).cast(Data
Types.DoubleType))
            .withColumn(LAST_WEEK_ANG_ORDER_VALUE_FIELD,
col(TOTAL_VALUE_WEEK_FIELD).divide(col(LAST_WEEK_ORDER_NUM
BER_FIELD)).cast(DataTypes.DoubleType))
            .withColumn(YESTERDAY_ANG_ORDER_VALUE_FIELD,
col(YESTERDAY_TOTAL_VALUE_FIELD).divide(col(YESTERDAY_ORDER
_NUMBER_FIELD)).cast(DataTypes.DoubleType))
            .collectAsList().get(0));

        LocalDateTime lastWeek = jobDate.minusDays(7);
        LocalDate yearStart = LocalDate.now().with(firstDayOfYear());

        ForecastCalculationHolder forecastCalculationHolder = new
ForecastCalculationHolder(
            getWeekData(sparkSession, jobDate, ordersDataSet)
                .sort(col(WEEK_DATE_FIELD).desc())
                .limit(52),
            jobDate, yearStart);

        List<YearChartPoint> yearData = forecastCalculationHolder.getYearData();
        List<YearChartPoint> predictions =
forecastCalculationHolder.getPredictions();
        Dataset<Row> weekData = forecastCalculationHolder.getWeekData();

        // day-by-day predictions
        log.trace("{} job : Starting calculating predictions for last 6 weeks",
getJobName());

        Dataset<Row> last6weeksOrders =
ordersDataSet.where(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.
minusWeeks(6))));
```

```
        List<Row> last6WeeksData =
last6weeksOrders.select(sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE
_FIELD)).collectAsList();

        BigDecimal last6weeksTotal = last6WeeksData.isEmpty() ?
BigDecimal.ZERO : BigDecimal.valueOf(new
SimpleRowWrapper(last6WeeksData.get(0))
                .getDouble(TOTAL_VALUE_FIELD, 0.0));

        log.trace("{} job : Total for last 6 weeks is {}", getJobName(),
last6weeksTotal);

        Map<Integer, DayData> dataByDays = last6weeksOrders
                .groupBy(dayofweek(timestampColumn).as(WEEK_DAY_NUMBER))
                .agg(sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE_FIELD))
                .orderBy(WEEK_DAY_NUMBER)
                .collectAsList()
                .stream()
                .map(SimpleRowWrapper::new)
                .collect(HashMap::new, (map, wr) ->
map.put(wr.getInt(WEEK_DAY_NUMBER),
                                new DayData(BigDecimal.ZERO.compareTo(last6weeksTotal)
== 0
                                                ? BigDecimal.ZERO
                                                :
BigDecimal.valueOf(wr.getDouble(TOTAL_VALUE_FIELD, 0.0))
                                                .multiply(predictions.get(0).getForecast())
                                                .divide(last6weeksTotal, 2, RoundingMode.HALF_UP))),
                        HashMap::putAll);

        WeekData shopifyWeekData = new WeekData(
                dataByDays.getOrDefault(DayOfWeek.MONDAY.getValue(),
DayData.EMPTY_FORECAST),
                dataByDays.getOrDefault(DayOfWeek.TUESDAY.getValue(),
DayData.EMPTY_FORECAST),
                dataByDays.getOrDefault(DayOfWeek.WEDNESDAY.getValue(),
DayData.EMPTY_FORECAST),
                dataByDays.getOrDefault(DayOfWeek.THURSDAY.getValue(),
DayData.EMPTY_FORECAST),
                dataByDays.getOrDefault(DayOfWeek.FRIDAY.getValue(),
DayData.EMPTY_FORECAST),
                dataByDays.getOrDefault(DayOfWeek.SATURDAY.getValue(),
DayData.EMPTY_FORECAST),
```

```
            dataByDays.getOrDefault(DayOfWeek.SUNDAY.getValue(),
DayData.EMPTY_FORECAST)
        );


        if (log.isTraceEnabled()) {
            log.trace("{} job: Loaded predictions for each days in current week",
getJobName());
            StringBuilder valuesForDays = new StringBuilder();
            shopifyWeekData.getDaysList().forEach(dayData ->
valuesForDays.append(dayData.getForecast()).append(" "));
            log.trace("{} job: Predictions for days: {}", getJobName(), valuesForDays);
        }

        log.trace("{} job : Loaded predictions for last 6 weeks", getJobName());

        // new customers
        long newCustomersCount =
ordersDataSet.groupBy(ORDER_CUSTOMER_ID_FIELD)
                .agg(countDistinct(ID_FIELD).as(ORDER_NUMBER_FIELD),

countDistinct(when(timestampColumn.$greater(Timestamp.valueOf(lastWeek)),
ID_FIELD)).as(LAST_WEEK_ORDER_NUMBER_FIELD)
                )

.filter(col(ORDER_NUMBER_FIELD).equalTo(col(LAST_WEEK_ORDER_NU
MBER_FIELD)))
                .count();

        log.trace("{} job : Find customer count: {}", getJobName(),
newCustomersCount);

        // construction of week stats
        Row lastWeekInfoRow =
weekData.orderBy(col(WEEK_NUMBER_FIELD).desc()).collectAsList().get(0);

        RowWrapper<String> lastWeekInfo = new EmptyRowWrapper();

        if (lastWeekInfoRow != null) {
            lastWeekInfo = new SimpleRowWrapper(lastWeekInfoRow);
        }

        List<Row> weekYearAgo = weekData
```

```
.filter(col(WEEK_DATE_FIELD).equalTo(Timestamp.valueOf(jobDate.minusWe
eks(52))))
            .collectAsList();

      RowWrapper<String> weekYearAgoInfo = weekYearAgo.isEmpty() ? new
EmptyRowWrapper() : new SimpleRowWrapper(weekYearAgo.get(0));

      List<Row> last52Weeks = weekData
            .orderBy(col(WEEK_NUMBER_FIELD).desc())
            .limit(52)

.agg(sum(TOTAL_VALUE_FIELD).divide(sum(ORDER_NUMBER_FIELD)).as(
AVG_ORDER_VALUE_FIELD))
            .collectAsList();

      BigDecimal lastYearAverageOrderValue = last52Weeks.isEmpty() ?
BigDecimal.ZERO :
            round(BigDecimal.valueOf(new
SimpleRowWrapper(last52Weeks.get(0)).getDouble(AVG_ORDER_VALUE_FIE
LD, 0.0)));

      List<BigDecimal> last20weeks =
forecastCalculationHolder.getLast20weeksInfo();
      BigDecimal current = last20weeks.isEmpty() ? BigDecimal.ZERO :
last20weeks.get(0);
      last20weeks.sort(Collections.reverseOrder());

      List<QuarterData> quarterData = forecastCalculationHolder.getQuarterData();



aggregatedData.setTotalSales(toBigDecimal(headerData.getDouble(TOTAL_VAL
UE_YEAR_FIELD, 0.0)));

aggregatedData.setYesterdaySales(toBigDecimal(headerData.getDouble(YESTER
DAY_TOTAL_VALUE_FIELD, 0.0)));

aggregatedData.setYesterdayAverageOrderValue(toBigDecimal(headerData.getDo
uble(YESTERDAY_ANG_ORDER_VALUE_FIELD, 0.0)));
      aggregatedData.setLast7daysAverageSales(last7daysAverageSales);

aggregatedData.setLast7daysAverageOrderValue(toBigDecimal(headerData.getDo
uble(LAST_WEEK_ANG_ORDER_VALUE_FIELD, 0.0)));
      aggregatedData.setLast20weeksRanking(last20weeks.indexOf(current) + 1);
```

```
        aggregatedData.setLast20weeksValues(last20weeks);
        aggregatedData.setWeekData(shopifyWeekData);

aggregatedData.setLastWeekOrders(lastWeekInfo.getLong(ORDER_NUMBER_FI
ELD, 0L));

aggregatedData.setLastWeekCustomers(lastWeekInfo.getLong(CUSTOMER_NU
MBER_FIELD, 0L));
        aggregatedData.setLastWeekNewCustomers(newCustomersCount);

aggregatedData.setLastWeekAverageOrderValue(round(BigDecimal.valueOf(last
WeekInfo.getDouble(AVG_ORDER_VALUE_FIELD, 0.0))));

aggregatedData.setSalesLastWeek(round(BigDecimal.valueOf(lastWeekInfo.getDo
uble(TOTAL_VALUE_FIELD, 0.0))));

aggregatedData.setSalesLastYearWeek(round(BigDecimal.valueOf(weekYearAgoI
nfo.getDouble(TOTAL_VALUE_FIELD, 0.0))));

aggregatedData.setForecastLastYearWeek(round(BigDecimal.valueOf(weekYearA
goInfo.getDouble(FORECAST_FIELD, 0.0))));
        aggregatedData.setQuarter0(quarterData.get(0));
        aggregatedData.setQuarter1(quarterData.get(1));
        aggregatedData.setQuarter2(quarterData.get(2));
        aggregatedData.setQuarter3(quarterData.get(3));
        aggregatedData.setYearData(new YearData(yearData));

aggregatedData.setYearForecast(forecastCalculationHolder.getYearForecast());

aggregatedData.setLastYearAverageOrderValue(lastYearAverageOrderValue);


        Dataset<Row> customerStatsDataset = createCustomerStatsDataset(jobDate,
ordersDataSet)
                .withColumn(SPENT_RANK_FIELD,
percent_rank().over(Window.orderBy(CUSTOMER_TOTAL_VALUE_FIELD)))
                .cache();

        log.trace("{} job: Constructed CUSTOMER dataset", getJobName());

        SimpleRowWrapper additionalCustomersInfo = new SimpleRowWrapper(
                customerStatsDataset
                        .agg(
```

```
countDistinct(when(col(IS_ACTIVE_FIELD).equalTo(lit(true)),
col(USER_ID_FIELD)))
                        .alias(ACTIVE_CUSTOMERS),

countDistinct(when(col(IS_AT_RISK_FIELD).equalTo(lit(true)),
col(USER_ID_FIELD)))
                        .alias(RISK_OF_LEAVING_CUSTOMERS),
                    sum(when(col(IS_ACTIVE_FIELD).equalTo(lit(true)),
col(TOTAL_VALUE_YEAR_FIELD)).otherwise(lit(0)))
                        .alias(ACTIVE_CUSTOMERS_VALUE),

sum(col(TOTAL_VALUE_YEAR_FIELD)).alias(TOTAL_CUSTOMERS_VALU
E),
                    sum(when(col(IS_AT_RISK_FIELD).equalTo(lit(true)),
col(TOTAL_VALUE_YEAR_FIELD)).otherwise(lit(0)))
                        .alias(AT_RISK_CUSTOMERS_VALUE)
                ).collectAsList().get(0)
    );

    log.trace("{} job: Aggregated and loaded CUSTOMER data", getJobName());

    log.trace("{} job: Constructed CUSTOMER with ENGAGEMENT dataset",
getJobName());

    Dataset<Row> productsDataset =
sparkDataService.loadProductDataForOrg(sparkSession, context);

    log.trace("{} job: Constructed PRODUCT dataset", getJobName());

    if (log.isTraceEnabled()) {
       log.trace("{} job: Size of PRODUCT dataset is {} rows", getJobName(),
productsDataset.count());
    }

    log.trace("{} job: Building Shopify Card Stage", getJobName());

    SegmentData highValueSegmentData =
aggregateSegmentData(ordersDataSet, productsDataset, customerStatsDataset,
            jobDate, additionalCustomersInfo, SegmentType.HIGH_VALUE);

    SegmentData averageValueSegmentData =
aggregateSegmentData(ordersDataSet, productsDataset, customerStatsDataset,
            jobDate, additionalCustomersInfo, SegmentType.AVERAGE_VALUE);
```

```
        SegmentData lowValueSegmentData = aggregateSegmentData(ordersDataSet,
productsDataset, customerStatsDataset,
                jobDate, additionalCustomersInfo, SegmentType.LOW_VALUE);

        customerStatsDataset.unpersist();


aggregatedData.setActiveCustomers(additionalCustomersInfo.getLong(ACTIVE_C
USTOMERS, 0L));

aggregatedData.setAtRiskCustomers(additionalCustomersInfo.getLong(RISK_OF_
LEAVING_CUSTOMERS, 0L));

aggregatedData.setActiveCustomersLastYearSpend(BigDecimal.valueOf(additiona
lCustomersInfo.getDouble(ACTIVE_CUSTOMERS_VALUE, 0.0)));

aggregatedData.setAtRiskCustomersLastYearSpend(BigDecimal.valueOf(additiona
lCustomersInfo.getDouble(AT_RISK_CUSTOMERS_VALUE, 0.0)));
        aggregatedData.setHighValueSegmentData(highValueSegmentData);
        aggregatedData.setAverageValueSegmentData(averageValueSegmentData);
        aggregatedData.setLowValueSegmentData(lowValueSegmentData);

        // movers and shakers
        WindowSpec windowSpecTheWeekBeforeLast =
Window.orderBy(col(TOTAL_ORDER_QTY_WEEK_BEFORE_LAST_FIELD).
desc(), col(TOTAL_VALUE_WEEK_BEFORE_LAST_FIELD).desc());

        Dataset<Row> ordersTheWeekBeforeLast = ordersDataSet

.filter(timestampColumn.between(Timestamp.valueOf(jobDate.minusDays(14)),
Timestamp.valueOf(jobDate.minusDays(7))))
                .join(productsDataset, ORDER_PRODUCT_JOIN_COLUMN, "left")
                .groupBy(col(ORDER_PRODUCT_ID_FIELD))
                .agg(

sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE_WEEK_BEFORE_LAS
T_FIELD),

sum(col(ORDER_LINE_ITEM_QTY)).as(TOTAL_ORDER_QTY_WEEK_BEFO
RE_LAST_FIELD)

).withColumn(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD,
row_number().over(windowSpecTheWeekBeforeLast));
```

```
        WindowSpec windowSpecLastWeek =
Window.orderBy(col(TOTAL_ORDER_QTY_FIELD).desc(),
col(TOTAL_VALUE_FIELD).desc());

        Dataset<Row> ordersLastWeek = ordersDataSet

.filter(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusDays(7))))
                .join(productsDataset, ORDER_PRODUCT_JOIN_COLUMN, "left")
                .groupBy(col(ORDER_PRODUCT_ID_FIELD))
                .agg(
                        first(col(PRODUCT_NAME), true).as(PRODUCT_NAME),
                        sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE_FIELD),

sum(col(ORDER_LINE_ITEM_QTY)).as(TOTAL_ORDER_QTY_FIELD)
                ).withColumn(PRODUCT_NAME_ARR_FIELD,
split(col(PRODUCT_NAME), " "))
                .withColumn(PRODUCT_POSITION_FIELD,
row_number().over(windowSpecLastWeek));

        Dataset<Row> top10LastWeek = ordersLastWeek
                .sort(col(TOTAL_ORDER_QTY_FIELD).desc(),
col(TOTAL_VALUE_FIELD).desc())
                .limit(10);

        Dataset<Row> moversAndShakersTop10 = top10LastWeek
                .join(ordersTheWeekBeforeLast, PRODUCT_JOIN_COLUMN, "left")
                .groupBy(col(ORDER_PRODUCT_ID_FIELD))
                .agg(
                        first(col(PRODUCT_NAME), true).as(PRODUCT_NAME),
                        first(col(TOTAL_VALUE_FIELD),
true).as(TOTAL_VALUE_FIELD),
                        first(col(TOTAL_ORDER_QTY_FIELD),
true).as(TOTAL_ORDER_QTY_FIELD),
                        first(col(PRODUCT_POSITION_FIELD),
true).as(PRODUCT_POSITION_FIELD),

first(col(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD),
true).as(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD)
                ).withColumn(CHANGE_FIELD,
col(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD).minus(col(PR
ODUCT_POSITION_FIELD)))
                .sort(col(TOTAL_ORDER_QTY_FIELD).desc(),
col(TOTAL_VALUE_FIELD).desc());

        Dataset<Row> otherProducts = ordersLastWeek
```

```
            .except(top10LastWeek)
            .agg(

sum(col(TOTAL_ORDER_QTY_FIELD)).as(TOTAL_ORDER_QTY_FIELD),
                    sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE_FIELD)
            ).withColumn(PRODUCT_NAME, lit("Other"));

        // customer subscription opportunities
        WindowSpec windowSpecTheWeekBeforeLastSubscriptionOpportunities =
Window.orderBy(col(REPEAT_COUNT_PERCENT_WEEK_BEFORE_LAST_FI
ELD).desc(),
            col(REPEAT_VALUE_WEEK_BEFORE_LAST_FIELD).desc());

        Dataset<Row> productsTotalFirstValueWeekBeforeLast = ordersDataSet

.filter(timestampColumn.between(Timestamp.valueOf(jobDate.minusMonths(3).mi
nusWeeks(1)),
                    Timestamp.valueOf(jobDate.minusWeeks(1))))
            .join(productsDataset, ORDER_PRODUCT_JOIN_COLUMN,
LEFT_JOIN)
            .groupBy(col(ORDER_PRODUCT_ID_FIELD),
col(ORDER_CUSTOMER_ID_FIELD))
            .agg(

first(col(TOTAL_VALUE_FIELD)).as(FIRST_ORDER_VALUE_FIELD)
            ).groupBy(col(ORDER_PRODUCT_ID_FIELD))

.agg(sum(col(FIRST_ORDER_VALUE_FIELD)).as(TOTAL_FIRST_ORDER_V
ALUE_FIELD));

        Dataset<Row> subscriptionOrderTheWeekBeforeLast = ordersDataSet

.filter(timestampColumn.between(Timestamp.valueOf(jobDate.minusMonths(3).mi
nusWeeks(1)),
                    Timestamp.valueOf(jobDate.minusWeeks(1))))
            .join(productsDataset, ORDER_PRODUCT_JOIN_COLUMN,
LEFT_JOIN)
            .join(productsTotalFirstValueWeekBeforeLast,
PRODUCT_JOIN_COLUMN, LEFT_JOIN)
            .groupBy(col(ORDER_PRODUCT_ID_FIELD))
            .agg(
                sum(col(TOTAL_VALUE_FIELD)).as(TOTAL_VALUE_FIELD),

countDistinct(col(ORDER_CUSTOMER_ID_FIELD)).as(COUNT_CUSTOMERS
_FIELD),
```

```
countDistinct(col(ORDER_ID_FIELD)).as(ORDER_NUMBER_FIELD),

first(col(TOTAL_FIRST_ORDER_VALUE_FIELD)).as(TOTAL_FIRST_ORDER
_VALUE_FIELD)
        ).withColumn(REPEAT_COUNT_WEEK_BEFORE_LAST_FIELD,
col(ORDER_NUMBER_FIELD).minus(col(COUNT_CUSTOMERS_FIELD)))
        .withColumn(REPEAT_VALUE_WEEK_BEFORE_LAST_FIELD,
col(TOTAL_VALUE_FIELD).minus(col(TOTAL_FIRST_ORDER_VALUE_FIE
LD)))

.withColumn(REPEAT_COUNT_PERCENT_WEEK_BEFORE_LAST_FIELD,
col(REPEAT_COUNT_WEEK_BEFORE_LAST_FIELD).multiply(lit(100D).divi
de(col(ORDER_NUMBER_FIELD))))

.withColumn(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD,
row_number().over(windowSpecTheWeekBeforeLastSubscriptionOpportunities));

    WindowSpec windowSpecLastWeekSubscriptionOpportunities =
Window.orderBy(col(REPEAT_COUNT_PERCENT_FIELD).desc(),
col(REPEAT_VALUE_FIELD).desc());

    Dataset<Row> productsTotalFirstValueLastWeek = ordersDataSet

.filter(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusMonths(3)
)))
        .join(productsDataset, ORDER_PRODUCT_JOIN_COLUMN,
LEFT_JOIN)
        .groupBy(col(ORDER_PRODUCT_ID_FIELD),
col(ORDER_CUSTOMER_ID_FIELD))
        .agg(

first(col(TOTAL_VALUE_FIELD)).as(FIRST_ORDER_VALUE_FIELD)
        ).groupBy(col(ORDER_PRODUCT_ID_FIELD))

.agg(sum(col(FIRST_ORDER_VALUE_FIELD)).as(TOTAL_FIRST_ORDER_V
ALUE_FIELD));

    Dataset<Row> subscriptionOrderLastWeek = ordersDataSet

.filter(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.minusMonths(3)
)))
        .join(productsDataset, ORDER_PRODUCT_JOIN_COLUMN)
        .join(productsTotalFirstValueLastWeek, PRODUCT_JOIN_COLUMN,
LEFT_JOIN)
```

```
            .groupBy(col(ORDER_PRODUCT_ID_FIELD))
            .agg(
                  first(col(PRODUCT_NAME), true).as(PRODUCT_NAME),

countDistinct(col(ORDER_CUSTOMER_ID_FIELD)).as(COUNT_CUSTOMERS
_FIELD),

countDistinct(col(ORDER_ID_FIELD)).as(ORDER_NUMBER_FIELD),

sum(col(TOTAL_VALUE_FIELD)).as(REPEAT_VALUE_FIELD),

first(col(TOTAL_FIRST_ORDER_VALUE_FIELD)).as(TOTAL_FIRST_ORDER
_VALUE_FIELD)
            ).withColumn(REPEAT_COUNT_FIELD,
col(ORDER_NUMBER_FIELD).minus(col(COUNT_CUSTOMERS_FIELD)))
            .withColumn(REPEAT_VALUE_FIELD,
col(REPEAT_VALUE_FIELD).minus(col(TOTAL_FIRST_ORDER_VALUE_FI
ELD)))
            .withColumn(REPEAT_COUNT_PERCENT_FIELD,
col(REPEAT_COUNT_FIELD).multiply(lit(100D).divide(col(ORDER_NUMBER
_FIELD))))
            .withColumn(PRODUCT_POSITION_FIELD,
row_number().over(windowSpecLastWeekSubscriptionOpportunities));


      Dataset<Row> top10LastWeekSubscriptionOpportunities =
subscriptionOrderLastWeek.limit(10);

      Dataset<Row> resultSubscriptionDataSet =
top10LastWeekSubscriptionOpportunities
            .join(subscriptionOrderTheWeekBeforeLast,
PRODUCT_JOIN_COLUMN, "left")
            .groupBy(col(ORDER_PRODUCT_ID_FIELD))
            .agg(
                  first(col(PRODUCT_NAME), true).as(PRODUCT_NAME),
                  first(col(REPEAT_COUNT_WEEK_BEFORE_LAST_FIELD),
true)
                        .alias(REPEAT_COUNT_WEEK_BEFORE_LAST_FIELD),
                  first(col(PRODUCT_POSITION_FIELD),
true).as(PRODUCT_POSITION_FIELD),

first(col(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD),
true).as(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD),
                  first(col(REPEAT_COUNT_PERCENT_FIELD),
true).as(REPEAT_COUNT_PERCENT_FIELD),
```

```
                first(col(REPEAT_VALUE_FIELD),
true).as(REPEAT_VALUE_FIELD)
        ).withColumn(CHANGE_FIELD,
col(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIELD).minus(col(PR
ODUCT_POSITION_FIELD)))
        .sort(col(REPEAT_COUNT_PERCENT_FIELD).desc(),
col(REPEAT_VALUE_FIELD).desc());

    // word cloud
    Map<String, Double> wordCloudWeights = new HashMap<>();

    StanfordCoreNLP pipeline = getPipeline();

    List<Row> orderLastWeekList = ordersLastWeek.collectAsList();

    if (!orderLastWeekList.isEmpty()) {
       orderLastWeekList
            .stream()
            .map(SimpleRowWrapper::new)
            .forEach(s -> {
               Double weight = s.getDouble(TOTAL_VALUE_FIELD);
               s.getList(PRODUCT_NAME_ARR_FIELD, String.class,
Collections.emptyList())
                    .stream()
                    .filter(word ->
pipeline.processToCoreDocument(word).tokens().size() > 0)
                    .map(this::prepareWord)
                    .filter(this::isFullWord)
                    .forEach(word ->

wordCloudWeights.compute(pipeline.processToCoreDocument(word)
                                .tokens().get(0).lemma(), (k, v) ->
Optional.ofNullable(v).orElse(0D) + weight));
            });
    }

    List<String> stopWordsList = getStopWords();

    List<WordWithValue> words = wordCloudWeights.entrySet()
        .stream()
        .filter(e -> !stopWordsList.contains(e.getKey()))
        .map(e -> new WordWithValue(e.getKey(),
round(BigDecimal.valueOf(e.getValue()))))
        .collect(Collectors.toList());
```

```
        // building climbers and shakers
        ProductSoldInfo topClimber = new ProductSoldInfo();
        ProductSoldInfo topFaller = new ProductSoldInfo();

        if (!moversAndShakersTop10.collectAsList().isEmpty()) {
            topClimber = buildMoversAndShakersProducts(new
SimpleRowWrapper(moversAndShakersTop10
                    .sort(col(CHANGE_FIELD).desc(),
col(TOTAL_VALUE_FIELD).desc()).collectAsList().get(0)));

            topFaller = buildMoversAndShakersProducts(new
SimpleRowWrapper(moversAndShakersTop10
                    .sort(col(CHANGE_FIELD).asc_nulls_last(),
col(TOTAL_VALUE_FIELD)).collectAsList().get(0)));
        }

        List<ProductSoldInfo> moversAndShakers =
moversAndShakersTop10.collectAsList()
                .stream()
                .map(SimpleRowWrapper::new)
                .map(this::buildMoversAndShakersProducts)
                .collect(Collectors.toList());

        moversAndShakers.add(buildMoversAndShakersProducts(new
SimpleRowWrapper(otherProducts.collectAsList().get(0))));
        moversAndShakers.add(
                moversAndShakers.stream().reduce(new ProductSoldInfo("Total",
BigDecimal.ZERO, BigDecimal.valueOf(0L), 0, 0),
                        (p1, p2) -> new ProductSoldInfo(
                                p1.getName(),
                                p1.getTotalValue().add(p2.getTotalValue()),
                                p1.getLastWeekValue().add(p2.getLastWeekValue()),
                                p1.getWeekBeforeLastPosition() +
p2.getWeekBeforeLastPosition(),
                                p1.getLastWeekPosition() + p2.getLastWeekPosition()
                        )
                )
        );

        List<ProductSoldInfo> subscriptionOpportunities =
resultSubscriptionDataSet.collectAsList()
                .stream()
                .map(SimpleRowWrapper::new)
                .map(this::buildSubscriptionOpportunitiesProducts)
                .collect(Collectors.toList());
```

```
        aggregatedData.setWordCloud(new WordCloud(words));
        aggregatedData.setDifferentProductsSold(ordersLastWeek.count());
        aggregatedData.setTopClimber(topClimber);
        aggregatedData.setTopFaller(topFaller);
        aggregatedData.setMoversAndShakers(moversAndShakers);
        aggregatedData.setSubscriptionOpportunities(subscriptionOpportunities);

        return aggregatedData;
    }

    protected Dataset<Row> createCustomerStatsDataset(LocalDateTime jobDate,
Dataset<Row> ordersDataSet) {
        log.trace("{} job: loading user stats dataset", getJobName());

        Column timestampColumn = col(ORDER_TIMESTAMP);

        Dataset<Row> dates =
ordersDataSet.groupBy(ORDER_CUSTOMER_ID_FIELD)
                .agg(

min(timestampColumn).as(OLDEST_ORDER_TIMESTAMP_FIELD_W_CUT),

when(min(timestampColumn).gt(Timestamp.valueOf(jobDate.minusYears(5))),
min(timestampColumn))

.otherwise(Timestamp.valueOf(jobDate.minusYears(5))).as(OLDEST_ORDER_TI
MESTAMP_FIELD),
                        lit(Timestamp.valueOf(jobDate)).as(JOB_DATE_FIELD)
                )
                .withColumn(QUARTER2_LOWER_BOUND, expr("date_sub(job_date,
cast(datediff(job_date, oldest_order_timestamp) / 4 as int))").cast("timestamp"))
                .withColumn(QUARTER3_LOWER_BOUND, expr("date_sub(job_date,
cast(datediff(job_date, oldest_order_timestamp) / 2 as int))").cast("timestamp"))
                .withColumn(QUARTER4_LOWER_BOUND, expr("date_sub(job_date,
cast(datediff(job_date, oldest_order_timestamp) * 3 / 4 as
int))").cast("timestamp"));


        if (log.isTraceEnabled()) {
            log.trace("{} job: loaded CUSTOMER data with {} rows", getJobName(),
dates.count());
        }
```

```
        WindowSpec window =
Window.partitionBy(ID_FIELD).orderBy(col(ORDER_TIMESTAMP).asc_nulls_l
ast());

        return ordersDataSet
                .withColumn(FIRST_ORDER_ID_FIELD, first(col(ID_FIELD),
true).over(window))
                .join(dates, ORDER_CUSTOMER_ID_TO_JOIN, LEFT_JOIN)
                .groupBy(col(ID_FIELD))
                .agg(

first(FIRST_ORDER_ID_FIELD).alias(FIRST_ORDER_ID_FIELD),
                        countDistinct(col(ID_FIELD)).alias(ORDER_NUMBER_FIELD),
                        coalesce(sum(col(TOTAL_VALUE_FIELD)),
lit(0)).as(CUSTOMER_TOTAL_VALUE_FIELD),

coalesce(sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusYears(1))), col(TOTAL_VALUE_FIELD))), lit(0))
                                .as(TOTAL_VALUE_YEAR_FIELD),

coalesce(sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusDays(90))), col(TOTAL_VALUE_FIELD))), lit(0))
                                .as(TOTAL_VALUE_QUARTER_FIELD),

coalesce(sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusDays(90))), col(TOTAL_VALUE_FIELD))), lit(0))
                                .as(TOTAL_VALUE_90_DAYS_FIELD),

coalesce(sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusDays(60))), col(TOTAL_VALUE_FIELD))), lit(0))
                                .as(TOTAL_VALUE_60_DAYS_FIELD),

coalesce(sum(when(timestampColumn.$greater$eq(Timestamp.valueOf(jobDate.m
inusDays(30))), col(TOTAL_VALUE_FIELD))), lit(0))
                                .as(TOTAL_VALUE_30_DAYS_FIELD),

countDistinct(when(timestampColumn.lt(col(QUARTER2_LOWER_BOUND)),
when(timestampColumn.$greater$eq(col(OLDEST_ORDER_TIMESTAMP_FIEL
D)), col(ID_FIELD)))).as(ORDERS_COUNT_QUARTER1),

countDistinct(when(timestampColumn.lt(col(QUARTER3_LOWER_BOUND)),
when(timestampColumn.$greater$eq(col(QUARTER2_LOWER_BOUND)),
col(ID_FIELD)))).as(ORDERS_COUNT_QUARTER2),

countDistinct(when(timestampColumn.lt(col(QUARTER4_LOWER_BOUND)),
```

```
when(timestampColumn.$greater$eq(col(QUARTER3_LOWER_BOUND)),
col(ID_FIELD))).as(ORDERS_COUNT_QUARTER3),

countDistinct(when(timestampColumn.$less$eq(col(JOB_DATE_FIELD)),
when(timestampColumn.$greater$eq(col(QUARTER4_LOWER_BOUND)),
col(ID_FIELD)))).as(ORDERS_COUNT_QUARTER4),
                first(col(OLDEST_ORDER_TIMESTAMP_FIELD_W_CUT),
true).as(OLDEST_ORDER_TIMESTAMP_FIELD_W_CUT),
                first(col(OLDEST_ORDER_TIMESTAMP_FIELD),
true).as(OLDEST_ORDER_TIMESTAMP_FIELD),
                first(col(JOB_DATE_FIELD), true).as(JOB_DATE_FIELD),
                first(col(ORDER_CUSTOMER_ID_FIELD),
true).as(USER_ID_FIELD)
            )
            .withColumn(IS_LOYAL_FIELD,
col(ORDERS_COUNT_QUARTER1).plus(col(ORDERS_COUNT_QUARTER3))
.$greater(lit(0))

.and(col(ORDERS_COUNT_QUARTER2).plus(col(ORDERS_COUNT_QUART
ER4)).$greater(lit(0))))
            .withColumn(IS_ACTIVE_FIELD,
col(TOTAL_VALUE_90_DAYS_FIELD).$greater(lit(0)))
            .withColumn(IS_AT_RISK_FIELD,
col(CUSTOMER_TOTAL_VALUE_FIELD).$greater(col(TOTAL_VALUE_YEA
R_FIELD))

.and(col(TOTAL_VALUE_YEAR_FIELD).$greater(col(TOTAL_VALUE_60_D
AYS_FIELD)))
                .and(col(TOTAL_VALUE_60_DAYS_FIELD).equalTo(lit(0)))
            );

    }

    private SegmentData aggregateSegmentData(Dataset<Row> ordersDataSet,
                                Dataset<Row> productsDataset, Dataset<Row>
segmentData,
                                LocalDateTime jobDate, SimpleRowWrapper
additionalCustomersInfo,
                                SegmentType segmentType) {
        log.trace("{} job: Aggregating data for {} Segment", getJobName(),
segmentType);
        Dataset<Row> filteredSegment = segmentData

.filter(segmentType.getBuildFilterColumn().apply(SPENT_RANK_FIELD))
            .filter(col(IS_ACTIVE_FIELD).equalTo(lit(true)));
```

```
        long segmentCustomers = filteredSegment.count();

        log.trace("{} job: {} Segment has {} customers", getJobName(),
segmentType, segmentCustomers);
        long totalCustomers = segmentData.count();

        log.trace("{} job: Segment {} has {}/{} CUSTOMERS", getJobName(),
segmentType, segmentCustomers, totalCustomers);

        SimpleRowWrapper productIdsWrapper =
ProductUtils.getProductIdsWrapper(ordersDataSet, productsDataset,
filteredSegment);

        List<ProductDataHolder> products =
ProductUtils.sortProductInfo(productIdsWrapper.getListOfStructs(PRODUCT_ID
S));
        List<ProductDataHolder> firstProducts =
ProductUtils.sortProductInfo(productIdsWrapper.getListOfStructs(FIRST_PRODU
CT_IDS));

        log.trace("{} job: Segment {} customers bought {} products, {} in the first
order", getJobName(), segmentType,
                products.size(), firstProducts.size());

        List<ProductData> top5Products =
ProductUtils.findTopProduct(segmentCustomers, products, 5);

        ProductData leadingPurchase =
ProductUtils.getLeadingPurchase(segmentCustomers, firstProducts);

        if (leadingPurchase == null) {
            log.trace("{} job: Segment {} CUSTOMERS don't have any purchases",
getJobName(), segmentType);
        } else {
            log.trace("{} job: Segment {} leading product is '{}'", getJobName(),
segmentType, leadingPurchase.getName());
        }

        Dataset<Row> aggregatedDataset = filteredSegment
                .withColumn(ORDER_FREQUENCY_FIELD,
unix_timestamp(col(JOB_DATE_FIELD)).minus(unix_timestamp(col(OLDEST_
ORDER_TIMESTAMP_FIELD_W_CUT)))
                        .divide(col(ORDER_NUMBER_FIELD)))
```

```
            .withColumn(ORDER_AVERAGE_VALUE_FIELD,
col(CUSTOMER_TOTAL_VALUE_FIELD).divide(col(ORDER_NUMBER_FIE
LD)))
            .select(
                  countDistinct(when(col(IS_ACTIVE_FIELD).equalTo(lit(true)),
col(USER_ID_FIELD)))
                        .alias(ACTIVE_CUSTOMERS),

countDistinct(when(col(ORDER_NUMBER_FIELD).equalTo(lit(1)),
col(USER_ID_FIELD)))
                        .alias(ONE_ORDER_CUSTOMERS),

sum(ORDER_NUMBER_FIELD).alias(TOTAL_ORDER_NUMBER),

sum(CUSTOMER_TOTAL_VALUE_FIELD).alias(CUSTOMER_TOTAL_VAL
UE_FIELD),
                  sum(when(col(IS_ACTIVE_FIELD).equalTo(lit(true)),
col(TOTAL_VALUE_YEAR_FIELD)).otherwise(lit(0)))
                        .alias(ACTIVE_CUSTOMERS_VALUE),

avg(col(ORDER_AVERAGE_VALUE_FIELD)).as(AVG_ORDER_VALUE_FIE
LD),

avg(col(CUSTOMER_TOTAL_VALUE_FIELD)).as(AVG_CUSTOMER_SPEND
_VALUE),

avg(col(ORDER_FREQUENCY_FIELD)).as(ORDER_FREQUENCY_FIELD)
            );

      // segment history
      List<ChartPoint> segmentChartDataHistory =
getSegmentChartDataHistory(ordersDataSet, jobDate, segmentType,
segmentData);
      ChartPoint currentPoint = new ChartPoint(asDate(jobDate),
BigDecimal.valueOf(filteredSegment.count()));
      segmentChartDataHistory.add(currentPoint);
      segmentChartDataHistory.sort(Comparator.comparing(ChartPoint::getDate));

      log.trace("{} job: Built Segment {} history has {} history points",
getJobName(), segmentType, segmentChartDataHistory.size());

      SimpleRowWrapper aggregatedSegmentInfo = new
SimpleRowWrapper(aggregatedDataset.collectAsList().get(0));

      log.trace("{} job: Building {} Segment Entity", getJobName(), segmentType);
```

```
        return SegmentData.builder()
                .numberOfCustomers(segmentCustomers)

.averageSpend(toBigDecimal(aggregatedSegmentInfo.getDouble(AVG_CUSTOM
ER_SPEND_VALUE, 0.0)))

.activeCustomers(aggregatedSegmentInfo.getLong(ACTIVE_CUSTOMERS, 0L))

.allActiveCustomers(additionalCustomersInfo.getLong(ACTIVE_CUSTOMERS,
0L))

.totalRevenueCustomers(BigDecimal.valueOf(additionalCustomersInfo.getDouble(
TOTAL_CUSTOMERS_VALUE, 0.0)))

.revenueCustomers(BigDecimal.valueOf(aggregatedSegmentInfo.getDouble(ACTI
VE_CUSTOMERS_VALUE, 0.0)))

.oneOrderCustomers(aggregatedSegmentInfo.getLong(ONE_ORDER_CUSTOME
RS, 0L))

.totalOrderNumber(aggregatedSegmentInfo.getLong(TOTAL_ORDER_NUMBER,
0L))

.averageOrderValue(toBigDecimal(aggregatedSegmentInfo.getDouble(AVG_ORD
ER_VALUE_FIELD, 0.0)))

.averagePurchaseFrequency(toBigDecimal(aggregatedSegmentInfo.getDouble(OR
DER_FREQUENCY_FIELD, 0.0)))

.averageVisitFrequencySeconds(toBigDecimal(aggregatedSegmentInfo.getDouble(
ENGAGEMENT_GAP_SECONDS_FIELD, 0.0)))
                .mostPopularProducts(top5Products)
                .leadingPurchase(leadingPurchase)
                .chartData(segmentChartDataHistory)
                .build();
    }

    private List<ChartPoint> getSegmentChartDataHistory(Dataset<Row>
ordersDataSet, LocalDateTime jobDate,
                                        SegmentType segmentType, Dataset<Row>
segmentData) {

        return LongStream.range(1, 10)
                .mapToObj(jobDate::minusWeeks)
```

```
            .map(day -> new ChartPoint(
                asDate(day),

BigDecimal.valueOf(ordersDataSet.where(col(ORDER_TIMESTAMP).lt(Timesta
mp.valueOf(day)))
                    .join(segmentData, segmentData.col(USER_ID_FIELD)
                        .equalTo(ordersDataSet.col(USER_ID_FIELD)),
LEFT_JOIN)
                    .groupBy(ordersDataSet.col(USER_ID_FIELD))
                    .agg(
                        coalesce(sum(col(TOTAL_VALUE_FIELD)),
lit(0)).as(CUSTOMER_TOTAL_VALUE_FIELD),
                        first(col(IS_ACTIVE_FIELD)).as(IS_ACTIVE_FIELD)
                    )
                    .withColumn(SPENT_RANK_FIELD,
percent_rank().over(Window.orderBy(CUSTOMER_TOTAL_VALUE_FIELD)))

.where(segmentType.getBuildFilterColumn().apply(SPENT_RANK_FIELD))
                    .filter(col(IS_ACTIVE_FIELD).equalTo(lit(true)))
                    .count()))
            .collect(Collectors.toList());
    }


    public static BigDecimal toBigDecimal(double value) {
        return round(BigDecimal.valueOf(value));
    }

    public static BigDecimal round(BigDecimal value) {
        return value.setScale(2, RoundingMode.HALF_UP);
    }

    protected Dataset<Row> getWeekData(String sparkSession, LocalDateTime
jobDate, Dataset<Row> ordersDataSet) {
        // data for all weeks
        Dataset<Row> weekData = ordersDataSet.groupBy(WEEK_DATE_FIELD)
            .agg(
                countDistinct(col(ID_FIELD)).alias(ORDER_NUMBER_FIELD),

countDistinct(USER_ID_FIELD).alias(CUSTOMER_NUMBER_FIELD),

sum(col(ORDER_LINE_ITEM_QTY)).alias(ITEMS_NUMBER_FIELD),
                sum(TOTAL_VALUE_FIELD).alias(TOTAL_VALUE_FIELD),
```

```
countDistinct(col(ORDER_PRODUCT_ID_FIELD)).alias(DIFF_PRODUCTS_NU
MBER_FIELD)
                );
        // handling the case when there were no sales in some week, and it disappears
from the weekData
        List<Row> weekDataList = weekData.collectAsList();

        Timestamp startDateTime = Timestamp.valueOf(jobDate.minusYears(1));

        if (!weekDataList.isEmpty()) {
           startDateTime = new
SimpleRowWrapper(weekData.sort(col(WEEK_DATE_FIELD)).collectAsList().g
et(0)).getTimestamp(WEEK_DATE_FIELD);
        }

        Timestamp endDateTime = Timestamp.valueOf(jobDate.minusWeeks(1));

        Dataset<Timestamp> startWeeksDataset =
sparkDataService.createOneColumnDataset(sparkSession,
                generateWeekTimeKeys(startDateTime.toLocalDateTime(),
endDateTime.toLocalDateTime()), Encoders.TIMESTAMP());

        Dataset<Row> resultStartWeeksDataset =
startWeeksDataset.withColumnRenamed(VALUE_FIELD,
WEEK_DATE_FIELD);

        return weekData
                .join(resultStartWeeksDataset,
toSeq(Collections.singletonList(WEEK_DATE_FIELD)), "right")
                .withColumn(ORDER_NUMBER_FIELD,
coalesce(col(ORDER_NUMBER_FIELD), lit(0L)))
                .withColumn(CUSTOMER_NUMBER_FIELD,
coalesce(col(CUSTOMER_NUMBER_FIELD), lit(0L)))
                .withColumn(ITEMS_NUMBER_FIELD,
coalesce(col(ITEMS_NUMBER_FIELD), lit(0L)))
                .withColumn(TOTAL_VALUE_FIELD,
coalesce(col(TOTAL_VALUE_FIELD), lit(0D)))
                .withColumn(DIFF_PRODUCTS_NUMBER_FIELD,
coalesce(col(DIFF_PRODUCTS_NUMBER_FIELD), lit(0L)));
    }

    /**
     * Remove all non-letter characters and do lowercase.
     *
```

```
     * @param s src string.
     * @return processed string.
     */
    protected String prepareWord(String s) {
        if (s != null) {
            return s.replaceAll("[^\\p{L}]", "").toLowerCase();
        }
        return null;
    }

    /**
     * Check that word isn't a number, or only one letter/symbol.
     *
     * @param word word to check.
     * @return true if word is correct, false if word is a number or only one symbol.
     */
    protected boolean isFullWord(String word) {
        if (word == null || word.length() <= 3) {
            return false;
        }
        return !StringUtils.isNumeric(word);
    }

    private StanfordCoreNLP getPipeline() {
        Properties props = new Properties();
        props.setProperty("annotators", "tokenize,ssplit,pos,lemma");
        return new StanfordCoreNLP(props);
    }

    private List<String> getStopWords() {
        try (InputStream resource = stopWords.getInputStream()) {
            return new BufferedReader(new InputStreamReader(resource,
StandardCharsets.UTF_8))
                    .lines().collect(Collectors.toList());
        } catch (IOException e) {
            log.error("Can't read stop words file!", e);
            return Collections.emptyList();
        }
    }

    private ProductSoldInfo buildMoversAndShakersProducts(SimpleRowWrapper
productWrapper) {
        return new ProductSoldInfo(
                productWrapper.getString(PRODUCT_NAME, "Unknown"),
```

```
round(BigDecimal.valueOf(productWrapper.getDouble(TOTAL_VALUE_FIELD,
0D))),

toBigDecimal(productWrapper.getLong(TOTAL_ORDER_QTY_FIELD, 0L)),
            productWrapper.getInt(PRODUCT_POSITION_FIELD, 0),

productWrapper.getInt(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIE
LD, 0)
        );
    }

    private ProductSoldInfo
buildSubscriptionOpportunitiesProducts(SimpleRowWrapper productWrapper) {
        return new ProductSoldInfo(
            productWrapper.getString(PRODUCT_NAME, "Unknown"),

round(BigDecimal.valueOf(productWrapper.getDouble(REPEAT_VALUE_FIELD
, 0D))),

toBigDecimal(productWrapper.getDouble(REPEAT_COUNT_PERCENT_FIELD,
0D)),
            productWrapper.getInt(PRODUCT_POSITION_FIELD, 0),

productWrapper.getInt(PRODUCT_POSITION_IN_WEEK_BEFORE_LAST_FIE
LD, 0)
        );
    }

    private String getJobName() {
        return "Data Processing Job";
    }
}
```

ДОДАТОК Б

Код реалізації ForecastCalculationHolder

```
package dpl.processing.vo.holder;

import dpl.processing.model.QuarterData;
import dpl.processing.vo.YearChartPoint;
import dpl.processing.vo.wrapper.row.SimpleRowWrapper;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.expressions.Window;
import org.apache.spark.sql.expressions.WindowSpec;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.temporal.WeekFields;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.LongStream;
import java.util.stream.Stream;

import static dpl.processing.job.DataProcessingJob.*;
```

```
import static dpl.processing.utils.DateUtils.asDate;
import static dpl.processing.utils.DateUtils.setYearStartRelatively;
import static org.apache.spark.sql.functions.*;

@Slf4j
@NoArgsConstructor
@Getter
public class ForecastCalculationHolder {
    protected static final String WEEK_DATE_FIELD = "week_date";

    private static final double C_CONSTANT_FOR_PREDICTION_INTERVALS
= 1.28;
    private static final int GRAPH_WEEKS_COUNT = 52;
    private static final int WEEKS_FOR_FORECAST = 52;

    private static final String CUSTOMER_NUMBER_FIELD =
"num_of_customers";
    private static final String YEAR_WEEK_NUMBER = "week_of_year";
    private static final String WEEK_NUMBER_FIELD = "week_number";
    private static final String WEEK_TREND_VALUE = "week_trend_value";
    private static final String ACTUAL_DIFF_VALUE = "actual_diff";
    private static final String ACTUAL_MULTIPLY_VALUE = "actual_multiply";
    private static final String WEEK_DIFF_AVG = "avg_week_diff";
    private static final String STD_DEV_FIELD = "std_dev";
    private static final String FORECAST_FIELD = "forecast";
    private static final String WEEK_SEASONALITY_VALUE_FIELD =
"week_seasonality_value";

    private static final String ROLLING_AVG_VALUE_FIELD =
"rolling_4w_value";
```

```
    private static final String CENTRED_AVG_VALUE_FIELD =
"centred_average";
    private static final String CURRENT_DAY_FIELD = "current_day";
    private static final String XY_VALUE_FIELD = "xy";
    private static final String X_SUM_FIELD = "x_sum";
    private static final String X_POW_2_FIELD = "x_pov_2";
    private static final String XY_SUM_FIELD = "xy_sum";
    private static final String X_POW_2_SUM_FIELD = "x_pov_2_sum";
    private static final String Y_SUM_FIELD = "y_sum";
    private static final String M_VALUE_FIELD = "m";
    private static final String B_VALUE_FIELD = "b";

    private List<YearChartPoint> yearData;
    private List<BigDecimal> last20weeksInfo;
    private List<YearChartPoint> predictions;
    private List<YearChartPoint> existingData;
    private List<QuarterData> quarterData;
    private Dataset<Row> weekData;

    public ForecastCalculationHolder(Dataset<Row> weekData, LocalDateTime
currentDay,
                                     LocalDate configuredYearStart, BigDecimal weekForecast)
{
        // window over the all order dataset...
        WindowSpec weekWindow =
Window.partitionBy().orderBy(WEEK_DATE_FIELD);
        this.weekData = weekData;

        this.weekData = this.weekData
                .withColumn(AVG_ORDER_VALUE_FIELD,
col(TOTAL_VALUE_FIELD).divide(col(ORDER_NUMBER_FIELD)))
```

```
            .withColumn(WEEK_NUMBER_FIELD,
row_number().over(weekWindow))
            .withColumn(ROLLING_AVG_VALUE_FIELD,
avg(TOTAL_VALUE_FIELD).over(weekWindow.rowsBetween(-2, 1)))
            .withColumn(CENTRED_AVG_VALUE_FIELD,
avg(ROLLING_AVG_VALUE_FIELD).over(weekWindow.rowsBetween(0, 1)))
            .withColumn(XY_VALUE_FIELD,
col(CENTRED_AVG_VALUE_FIELD).multiply(col(WEEK_NUMBER_FIELD))
)
            .withColumn(X_POW_2_FIELD, pow(col(WEEK_NUMBER_FIELD),
2))
            .withColumn(YEAR_WEEK_NUMBER,
weekofyear(col(WEEK_DATE_FIELD)));

        // weeks info for aggregations and statistics
        long weeksCount = this.weekData.count();

        log.trace("DataProcessing job: Loaded ORDER data for {} weeks",
weeksCount);

        Dataset<Row> weekAggregationData = this.weekData
            .agg(
                sum(XY_VALUE_FIELD).as(XY_SUM_FIELD),
                sum(WEEK_NUMBER_FIELD).as(X_SUM_FIELD),
                sum(CENTRED_AVG_VALUE_FIELD).as(Y_SUM_FIELD),
                sum(X_POW_2_FIELD).as(X_POW_2_SUM_FIELD),

countDistinct(CENTRED_AVG_VALUE_FIELD).as(CENTRED_AVG_VALUE_
FIELD))
```

```
            .withColumn(M_VALUE_FIELD,
col(CENTRED_AVG_VALUE_FIELD).multiply(col(XY_SUM_FIELD)).minus(c
ol(X_SUM_FIELD).multiply(col(Y_SUM_FIELD)))

.divide(col(CENTRED_AVG_VALUE_FIELD).multiply(col(X_POW_2_SUM_FI
ELD)).minus(pow(col(X_SUM_FIELD), 2))))
            .withColumn(B_VALUE_FIELD,
col(Y_SUM_FIELD).minus(col(M_VALUE_FIELD).multiply(col(X_SUM_FIEL
D))).divide(col(CENTRED_AVG_VALUE_FIELD)));

        SimpleRowWrapper weeksInfo = new
SimpleRowWrapper(weekAggregationData.collectAsList().get(0));
        log.trace("DataProcessing job : Loaded weeks info");

        double m = weeksInfo.getDouble(M_VALUE_FIELD, 0.0);
        log.trace("DataProcessing job : m value is – {}", m);

        double b = weeksInfo.getDouble(B_VALUE_FIELD, 0.0);
        log.trace("DataProcessing job : b value is – {}", m);

        WeekFields weekFields = WeekFields.of(DayOfWeek.MONDAY, 3);

        this.weekData = this.weekData.withColumn(WEEK_TREND_VALUE,
lit(m).multiply(col(WEEK_NUMBER_FIELD)).plus(lit(b)))
            .withColumn(ACTUAL_DIFF_VALUE,
col(TOTAL_VALUE_FIELD).minus(col(WEEK_TREND_VALUE)))
            .withColumn(ACTUAL_MULTIPLY_VALUE,
col(TOTAL_VALUE_FIELD).divide(col(WEEK_TREND_VALUE)));

        // pairs week_no -> diff to the prediction value
```

```
Map<Integer, Double> weekSeasons =
this.weekData.groupBy(YEAR_WEEK_NUMBER)
        .agg(coalesce(avg(ACTUAL_DIFF_VALUE),
lit(0.0)).as(WEEK_DIFF_AVG))
        .collectAsList()
        .stream().collect(HashMap::new, (map, row) -> {
           SimpleRowWrapper rowWrapper = new SimpleRowWrapper(row);
           map.put(rowWrapper.getInt(YEAR_WEEK_NUMBER),
rowWrapper.getDouble(WEEK_DIFF_AVG, 0.0));
        }, HashMap::putAll);

    log.trace("DataProcessing job : Loaded week seasons info");

    WindowSpec weekSeasonalityWindow =
Window.partitionBy(YEAR_WEEK_NUMBER);

    this.weekData = this.weekData
        .withColumn(WEEK_SEASONALITY_VALUE_FIELD,
coalesce(avg(ACTUAL_DIFF_VALUE).over(weekSeasonalityWindow), lit(0.0)))
        .withColumn(FORECAST_FIELD,

lit(col(WEEK_SEASONALITY_VALUE_FIELD).plus(col(WEEK_TREND_VAL
UE))));

    log.trace("DataProcessing job : Constructed week data");

    double standardDeviation = new
SimpleRowWrapper(this.weekData.select(stddev(col(WEEK_TREND_VALUE).m
inus(col(TOTAL_VALUE_FIELD))).as(STD_DEV_FIELD))
        .collectAsList().get(0)).getDouble(STD_DEV_FIELD, 0.0);
```

```
log.trace("DataProcessing job : Calculated standard deviation: {}",
standardDeviation);

// calculated predictions for the next 'WEEKS_FOR_FORECAST' (52) weeks
this.predictions = LongStream.iterate(1, l -> ++l)
        .limit(WEEKS_FOR_FORECAST)
        .mapToObj(h -> {
            // ternary operator to eliminate duplicate week in year data for
FridayJob
            LocalDateTime dateTime = currentDay
                .plusWeeks(h – 1);
            Double trendDiff =
weekSeasons.getOrDefault(dateTime.get(weekFields.weekOfWeekBasedYear()),
1.0);

            double deviation = 0;
            if(!Double.valueOf(standardDeviation).isNaN()){
               deviation = C_CONSTANT_FOR_PREDICTION_INTERVALS *
standardDeviation * Math.sqrt(h * (1 + (double) h / (weeksCount)));
            }

            BigDecimal forecast = toBigDecimal(Math.max(trendDiff + (m *
(weeksCount + h) + b), 0D));

            return new YearChartPoint(asDate(dateTime),
                forecast.setScale(2, RoundingMode.HALF_UP),
                BigDecimal.valueOf(Math.max(forecast.doubleValue() –
deviation, 0.0)).setScale(2, RoundingMode.HALF_UP),
                forecast.add(BigDecimal.valueOf(deviation).setScale(2,
RoundingMode.HALF_UP))
            );
```

```
            }).collect(Collectors.toList());

        log.trace("DataProcessing job : Loaded predictions info");


        // existing data points
        this.existingData =
this.weekData.orderBy(col(WEEK_NUMBER_FIELD).desc())
                .collectAsList()
                .stream()
                .map(SimpleRowWrapper::new)
                .map(wr -> new
YearChartPoint(asDate(wr.getTimestamp(WEEK_DATE_FIELD).toLocalDateTim
e()),
                        BigDecimal.valueOf(wr.getDouble(TOTAL_VALUE_FIELD,
0.0)).setScale(2, RoundingMode.HALF_UP)))
                .collect(Collectors.toList());

        log.trace("DataProcessing job : Loaded existing data info");

        if (!existingData.isEmpty()) {
            log.trace("DataProcessing job : Actual value for last week is {}",
existingData.get(existingData.size() – 1).getForecast());
        } else {
            log.trace("DataProcessing job : Existing data is empty");
        }

        log.trace("DataProcessing job : Prediction value for current week is {}",
predictions.get(0).getForecast());

        // final year chart data
```

```
        this.yearData =
Stream.concat(this.existingData.stream().limit(GRAPH_WEEKS_COUNT / 2),
                this.predictions.stream())
            .sorted(Comparator.comparing(YearChartPoint::getDate))
            .limit(GRAPH_WEEKS_COUNT)
            .collect(Collectors.toList());

        log.trace("DataProcessing job : Loaded year data info");

        this.last20weeksInfo = this.existingData.subList(0,
Math.min(this.existingData.size(), 20))
            .stream().map(YearChartPoint::getActual).collect(Collectors.toList());

        log.trace("DataProcessing job : Loaded info for last 20 weeks");

        // final year chart data
        List<YearChartPoint> allGeneratedData =
Stream.concat(this.existingData.stream(),
                this.predictions.stream())
            .sorted(Comparator.comparing(YearChartPoint::getDate).reversed())
            .collect(Collectors.toList());

        this.quarterData = IntStream.range(1, 5)
            .mapToObj(i -> {
               YearQuarterHolder yearQuarter = new
YearQuarterHolder(setYearStartRelatively(configuredYearStart,
currentDay.toLocalDate()),
                    i, asDate(currentDay));

               return new QuarterData(allGeneratedData.stream()
                    .filter(w -> yearQuarter.isInQ(w.getDate()))
```

```
                    .map(y ->
Optional.ofNullable(y.getForecast()).orElseGet(y::getActual))
                    .reduce(BigDecimal.ZERO, BigDecimal::add),
yearQuarter.getCalendarQ(), yearQuarter.getStatus());
            })
            .collect(Collectors.toList());

      if (log.isTraceEnabled()) {
         log.trace("DataProcessing job: Loaded quarter data:");
         this.quarterData.forEach(q -> log.trace("DataProcessing job: Value for {}
quarter: {}", q.getName(), q.getForecastValue()));
      }
   }

   public BigDecimal getYearForecast(){
      return this.quarterData.stream()
            .map(QuarterData::getForecastValue)
            .reduce(BigDecimal.ZERO, BigDecimal::add);
   }

   public ForecastCalculationHolder(Dataset<Row> ordersDataSet, LocalDateTime
currentDay,
                                    LocalDate configuredYearStart) {
      this(ordersDataSet, currentDay, configuredYearStart, BigDecimal.ZERO);
   }
}
```