**Big Data Engineer Course**

**Final project**

Naya college

January 2023

**Telegram Shopping Bot**

Shira Shimla

Daniel Premisler

Liri Roeznthal

**Table of contents**

Background ……………………………………………………………………………………………………………… 3

Objectives …………………………………………………………………………………………………………………. 4

High Level Architecture Requirements Overview……………….……………………. 5

` Proposed High-Level Technological Components Overview………….…6-8

Architecture………………………………………………………………………..……………….……………………. 9

Suggested Data ProcessingModel……………………………………………………………...9-14

Project Perquisites………………………………………………………………….………………………………15

Alternatives………………………………………………………………………..……………….………………..16-19

Appendix………………………………………………………………………..……………….…………………...20-21

**Background**

Israel 2023 is an expensive country to live in. Israel’s annual inflation rate rose to 5.3 percent in November of 2022, which was the highest rating since October 2008. The inflation hasn’t passed the food industry, and it has indeed become one of the industries that their price increase was one of the more dramatic ones. Unlike many other products, food is essential, and cannot be given up.

In such reality, tools for prices’ comparisons are becoming more and more essential in general, and in the food industry particularly. As we could not find such a convenient tool dedicated to the Israeli market, we’ve decided to create one.

We have chosen to daily scrape prices data reported by three of Israel’s top retailers to the official website of Israel’s Consumer Protection And Fair Trade Authority. Then, through Telegram Bot, we allow each person to send his list of products, and get the cheapest retailer-branch combination to buy his products in. In addition, we aggregate and visualise all of the historical pricing data, in order to identify prices’ trends in a convenient way.

**Objectives**

On the business side, we aimed to develop a program that will encourage smart consumerism, and increase the chance for money savings among the people, by giving them the access to their most essential and bought products’ prices, data which without our program, is indeed existing today, however sporadically and not comparably.

On the technological side, we aimed to experience streaming technologies, scrape in different techniques, examine many platforms and technologies in order to decide which one fits best to our needs, and finally:

1. Creating a Telegram Bot that will allow receiving inputs from end users.
2. Creating a dashboard containing aggregated historical prices data, all at one place.
3. Creating an archive for all of the prices data, in order to conduct historical pricing analysis.

**High Level Architecture Requirements Overview**

# The main requirements to support the solution are::

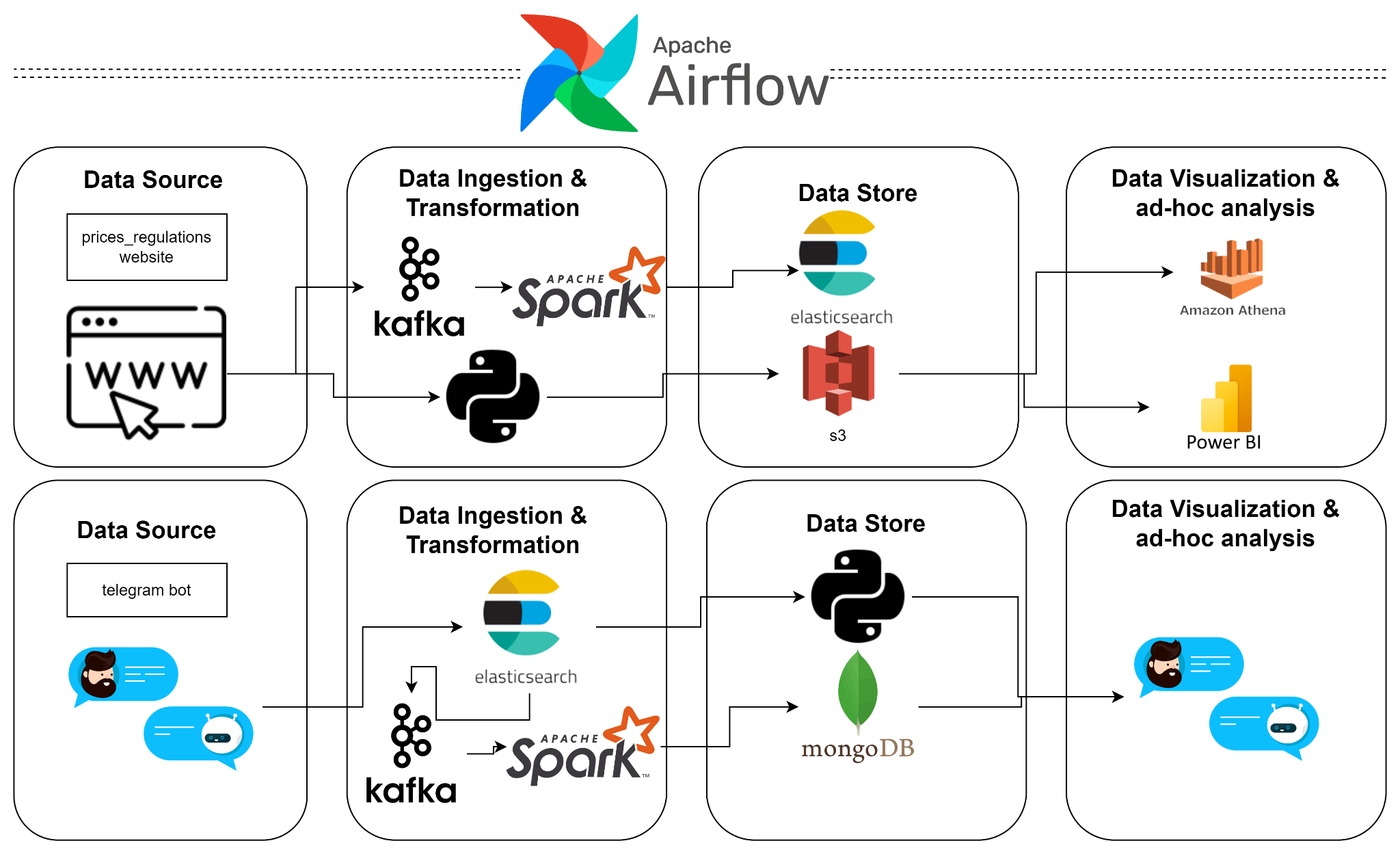
1. **Scalability** - Currently the data flow is limited in quantity, but the system was planned to expand and handle thousands of API requests and data flow per seconds from different data sources by using a big data architecture methodology of pipelining and data treatment and analysis.
2. **High availability** - is intended to become a major attribute in the system, as one of the advantages of this system is to allow users to interact with it's search engine to compare shopping lists and products.
3. **Low latency** - to get an immediate response per users query of the telegram bot.
4. **Data lake** - need to be connected to an analysis layer in order to allow users self-analysis and reporting creation over the ingested data.

# **Proposed High-Level Technological Components Overview**

We’ve decided to chooses the following components in our pipeline’s architecture:

1. **Storage** - Amazon S3 delivers easy to use data management capabilities to organise, store, and retrieve data effortlessly. The advantages of using S3 in our solution was due to the following reasons:
   1. Cost-effective - S3 is relatively cheap, it also supports all kinds of data types such as text files, binary files such as parquet, avro and many more.
   2. In-Depth Analytics - We wanted the ability to perform ad-hoc and query-in-place analytics, using Amazon Athena with S3 lets us do this without any difficulty.
   3. Accessibility - while one could argue we could have used HDFS for the analytics using Hive/Impala, one cannot argue with the fact that S3 provides an easy access to the data to other application, such as BI applications like Power Bi and Tableau or Data warehouse solutions such as RedShift or even act as a storage solution for a website or a web application. Configuring HDFS to some of these solutions would prove more difficult this is why we chose S3.
2. **Search Engine** - ElasticSearch is our component for fetching the right products and stores when a customer looks to compare between shopping lists. ElasticSearch allows users to quickly retrieve data records in any form, perform various kinds of searches and analyse billions of data records in just a couple of seconds. Our solution requires the customer to perform full text search of product names, ElasticSearch allows us to find the right products at ease.
3. **Visualisation** - Power Bi was our choice to show insights and various visualisations on our historic data. It has the ability to ingest data from more than 100 sources including python and R scripts. Using python in power bi we managed to create a data model of our historic data which is stored in S3 parquet files with ease.
4. **Orchestration** - Apache Airflow is an integrated application which makes it trivial to handle ETL and various work flows in our solution. Airflow is an easy to use UI that uses DAG’s to organise tasks with dependencies and relationships.
5. **Broadcasting** - Kafka technology was our choice to safely move the data from our python scraping process to the many other elements of our solution such as ElasticSearch, MongoDB and S3. The Kafka technology was established on three basic guidelines: fault tolerant, distributed and scalable which provides the ability to handle huge masses of data to lots of consumers and producers.
6. **Computing** - Apache Spark works great with kafka, it can work in batches as well as in a streaming mode. Spark can use the [Dataset/DataFrame API](https://spark.apache.org/docs/2.2.0/sql-programming-guide.html) in Scala, Java, Python or R to express streaming aggregations, event-time windows, stream-to-batch joins, etc.
7. **Database** - We used MongoDB in order to store the interactions between the telegram bot and the user, its schemaless NoSQL structure is perfect for storing various different interactions as well as easy aggregation options. MongoDB also works great with other tools we are using such as Apache Spark.
8. **Telegram Bot** - The Bot API is an HTTP based interface created for developers keen on building bots for Telegram. We use this technology for the user to interact with our database. Telegram is the 5th worldwide popular chat messenger, it supports easy bot building and customization via python library without the need to create a business account, this was the main reason to use it on behalf of other messengers.

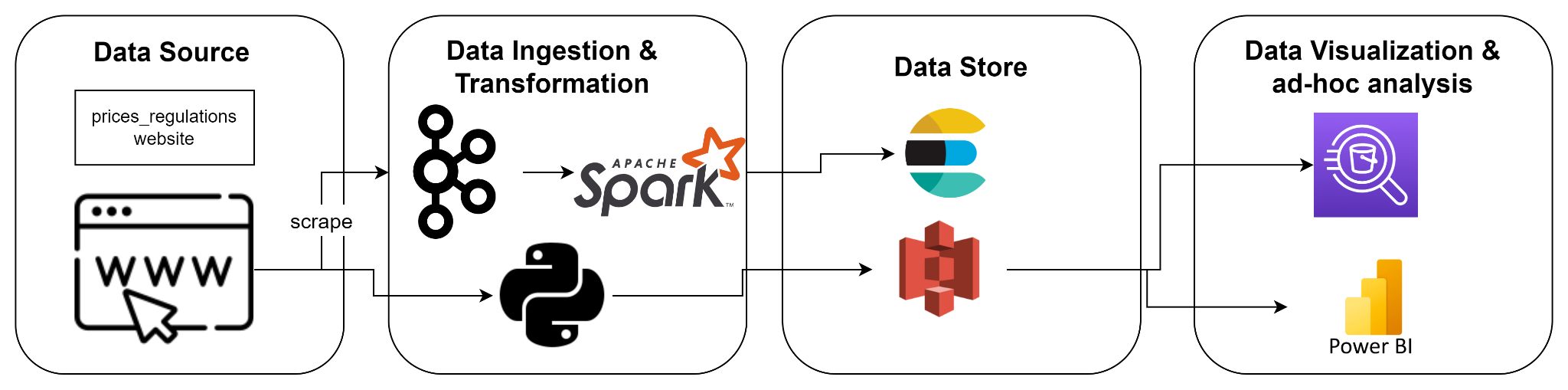
**Architecture**



**Suggested Data Processing Model**

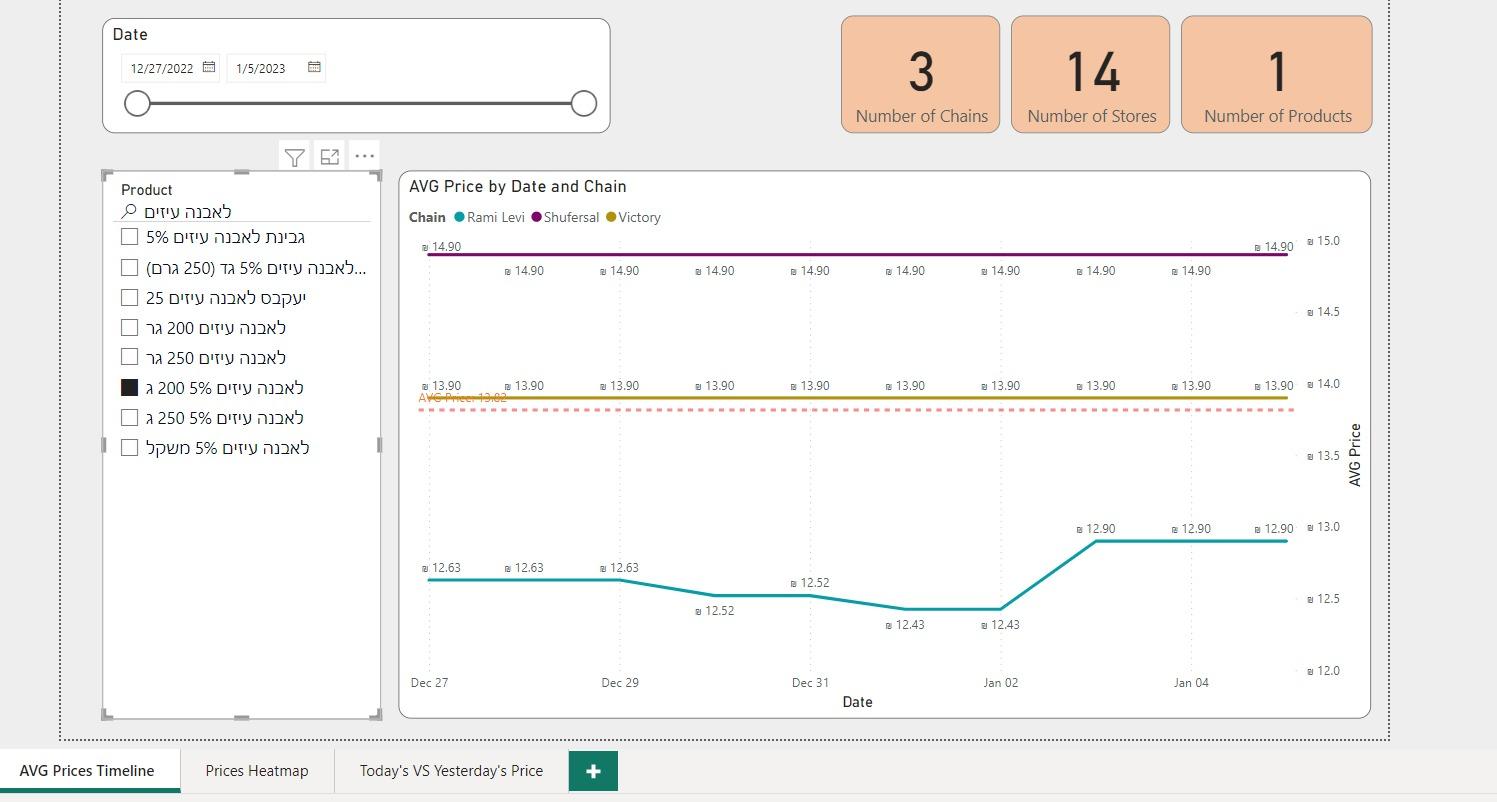
Data processing pipelines are described below and scheduled by apache Airflow

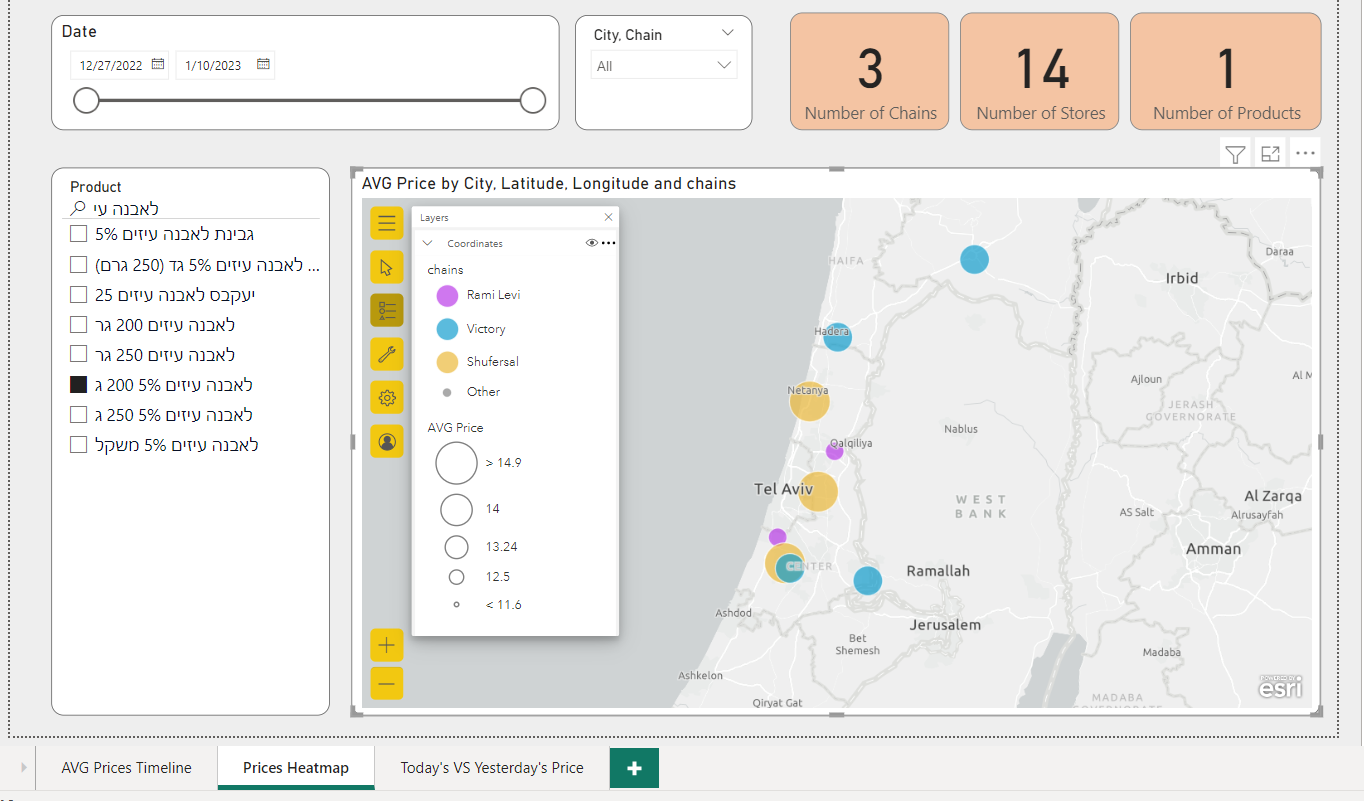
1. Data pipeline - Scrape for data about products and stores from selected retail chains for archiving/analysis/visualisation purposes.
2. Data pipeline - Allow users to compare shopping lists between different retail chains within nearby stores.

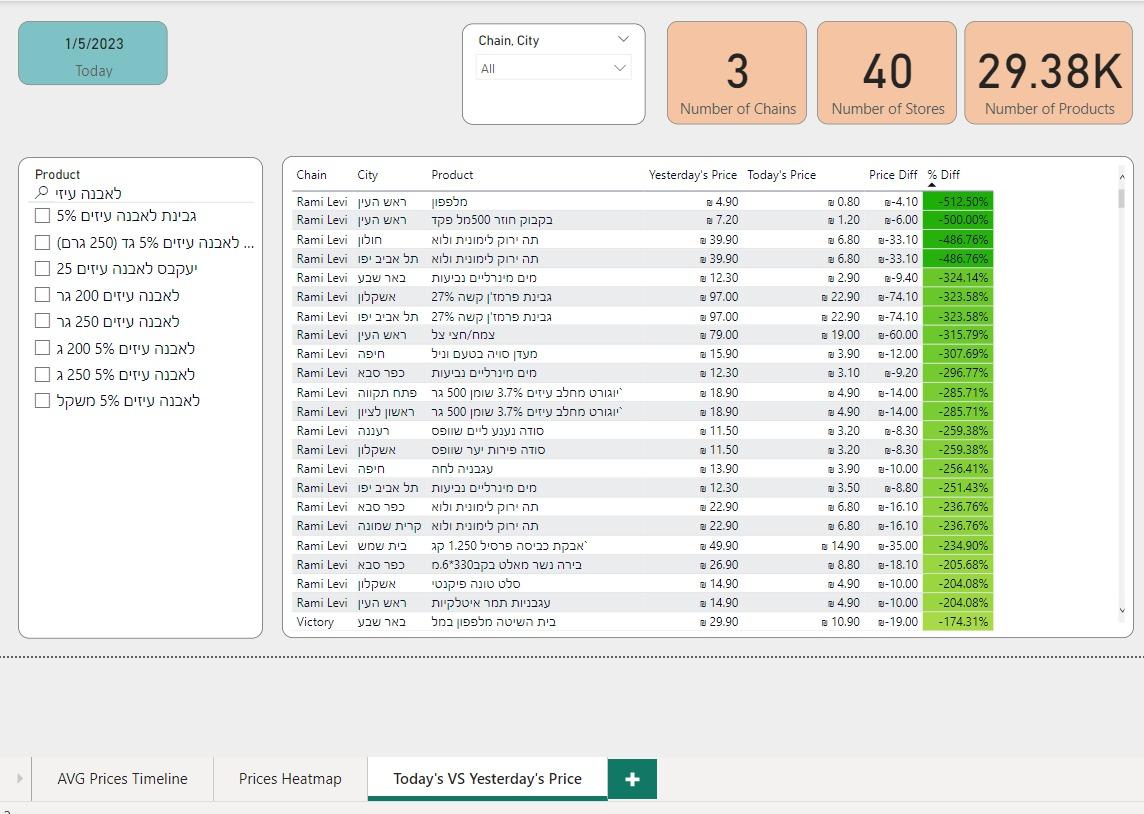
**Part 1 - Data pipeline -** Scrape for data about products and stores from selected retail chains for archiving/analysis/visualisation purposes.

1. Air flow triggers a python script which downloads and parses files involving information about retail chains stores and products in each store.
2. Each retail chain has its own website which requires a different scraping solution. In our case we used selenium for one retail chain and beautifulsoup for the other two retail chains.
3. We filter, transform and combine the data for the purpose of having all needed information for each product under the same document (json form) as well as each store.
4. A kafka producer sends each product and store via Spark into the Elasticsearch index, one for all the products and another for all the stores.
5. A python script sends the same data as a json into S3
6. Power bi visualises the data from S3 using a python script
7. We connect Amazon Athena to the S3 in order to perform ad-hoc queries on the products and stores

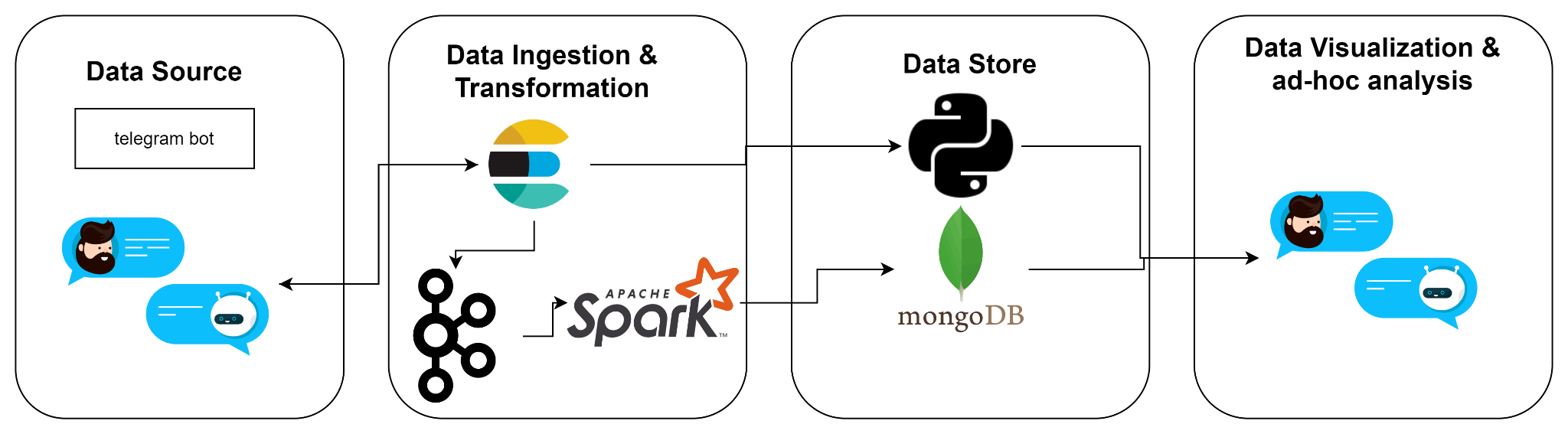
Examples of the visualisations from the power bi on the data stored on s3:





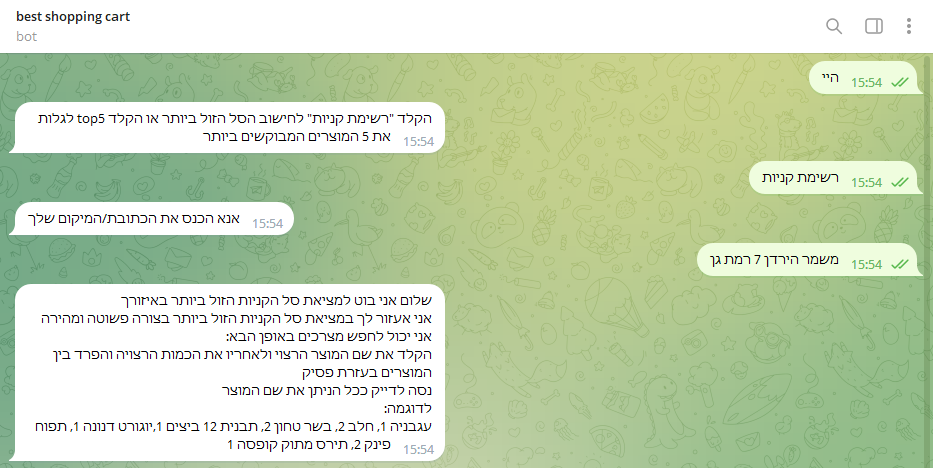


**Part 2 - Data pipeline -** Allow users to compare shopping lists between different retail chains within nearby stores.



1. A telegram bot handler allows a user to query for products using full text search on the Elasticsearch index from the previous pipeline
2. A python script transforms and sends a response regarding which store and chain has the lowest price along with the total price back to the bot handler. The bot do this calculations according to the user reported location (example below)
3. A Kafka producer sends via Spark information about the response along with metadata about the user interactions with the bot handler to MongoDB for future use
4. A user can also request to see the 5 most common products that were requested in the shopping bot, using spark to query the mongoDB.

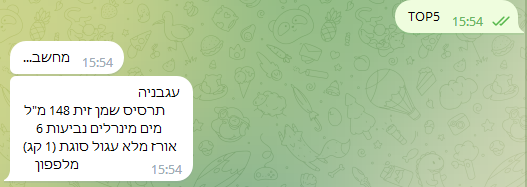
Example of the telegram bot showing the top products users are asking about:

****

Example of the telegram bot calculating the lowest price per shopping list

****

Example of the telegram bot showing the top products users are asking about:



**Project Perquisites**

The full working solution requires the below components:

1. Install Linux machine with 16 core 64 GB, 100GB SSD
2. Install Docker for Linux
3. The following applications and frameworks are needed:
   1. Airflow
   2. Kafka
   3. Spark
   4. Python 3.6
   5. ElasticSearch 7
   6. MongoDB
4. Install Conda with all required libraries:
   1. PyArrow
   2. PySpark
   3. Python for Kafka
   4. Telegram
   5. Pandas
   6. Numpy
   7. Geopy
   8. pyTelegramBotAPI
5. An APi is needed for two geocoding services, there are free options:
   1. [HERE Technologies | The world's #1 location platform](https://www.here.com/)
   2. [Openrouteservice](https://openrouteservice.org/)
6. Amazon account with S3 and Athena configured to work with the Linux machine
7. Power bi for desktop

**Alternatives**

**Storage:**

HDFS vs S3:

Hadoop Distributed File System (HDFS) and Amazon Simple Storage Service (S3) are both distributed file storage systems that can be used to store large volumes of data. However, they are designed for different use cases and have some significant differences.

In terms of scalability, both HDFS and S3 are highly scalable and can handle large volumes of data. HDFS is designed to scale out by adding more nodes to the cluster, while S3 is designed to scale up by increasing the capacity of the storage nodes.

In terms of durability, both HDFS and S3 are designed to be highly durable, with multiple copies of data stored across different nodes to protect against data loss. HDFS stores multiple copies of data on different nodes within the same cluster, while S3 stores data across multiple Availability Zones within an AWS region.

In terms of price, S3 is generally more expensive than HDFS, as it is a fully managed service offered by Amazon Web Services (AWS) and charges fees based on the volume of data stored and the number of requests made. HDFS, on the other hand, is free to use, as it is an open source technology that can be installed and run on your own infrastructure.

In terms of overall performance, S3 is generally considered to be faster and more performant than HDFS, as it is optimized for reading and writing large volumes of data in parallel. HDFS, on the other hand, is designed for high throughput of data at the expense of latency, and it may not be as performant for small files or for applications that require low latency.

Overall, HDFS and S3 are both distributed file storage systems that can be used to store large volumes of data. HDFS is a good choice for storing data that will be processed by Hadoop or other big data frameworks, while S3 is a good choice for storing data that will be accessed directly by applications or users.

**Stream platforms:**

Kafka vs Kinesis

Apache Kafka and Amazon Kinesis are both real-time data streaming platforms that can be used to ingest, process, and publish large volumes of data.

In terms of performance, both Kafka and Kinesis are capable of handling high-throughput, low-latency data streams. Kafka is generally considered to be more performant than Kinesis, as it is designed to handle larger volumes of data with higher levels of concurrency.

In terms of cost, Kafka is an open source technology, so it is free to use. Kinesis, on the other hand, is a cloud-based service offered by Amazon Web Services (AWS), and it charges fees based on the volume of data processed and the number of requests made.

In terms of scalability, both Kafka and Kinesis are highly scalable and can be easily distributed across multiple servers to support large volumes of data. Kafka supports horizontal scaling using partitioning, while Kinesis supports both horizontal and vertical scaling using a combination of shards and replica streams.

In terms of ease of use, Kinesis is generally considered to be easier to use than Kafka, as it is a fully managed service offered by AWS. Kafka, on the other hand, requires more setup and configuration, as it is a standalone technology that must be installed and managed on your own infrastructure.

Overall, both Kafka and Kinesis are powerful real-time data streaming platforms that can be used to ingest, process, and publish large volumes of data. Kafka is generally more performant and flexible, but it requires more setup and maintenance, while Kinesis is easier to use but may be more expensive and less customizable.

**Search Engine**

ElasticSearch vs Solr

Elasticsearch and Solr are both highly scalable, open source search engines that are built on top of Apache Lucene. Both Elasticsearch and Solr can be used to index, search, and analyse large volumes of data quickly and in near real time.

In terms of installation and configuration, Elasticsearch is generally easier to install and configure than Solr. Elasticsearch is written in Java and can be installed as a standalone application, or it can be run as a service. Solr, on the other hand, is a standalone server that requires a servlet container (such as Tomcat) to be installed and configured in order to run.

Both Elasticsearch and Solr support powerful full-text search, but Elasticsearch is generally considered to be more powerful and easier to use, thanks to its flexible search API and rich query DSL (domain specific language). Solr also has a flexible search API, but its query language is more limited and can be more difficult to use for some types of queries.

In terms of scalability, both Elasticsearch and Solr are highly scalable and can be easily distributed across multiple servers to support large volumes of search traffic. Both technologies support sharding and replicas to allow for horizontal scaling, and both can be run in a cluster to provide automatic failover and improved availability.

Overall, Elasticsearch and Solr are both powerful search engine technologies that can be used to index and search large volumes of data. Elasticsearch is generally considered to be more user-friendly and easier to use, while Solr is more powerful and flexible, but can be more difficult to set up and configure.

**Orchestration**

Apache Airflow vs Perfect

Apache Airflow and Prefect are both open source workflow management platforms that can be used to define, schedule, and execute complex pipelines.

In terms of workflow management, both Apache Airflow and Prefect provide a way to define and organise tasks into directed acyclic graphs (DAGs) and to schedule and execute these tasks on a regular basis. Both platforms also provide a way to monitor the execution of tasks and to retry failed tasks.

In terms of purpose, Apache Airflow is primarily used for scheduling and orchestration of batch jobs, while Prefect is designed for both batch and real-time data pipelines. Prefect also provides additional features such as dynamic task generation and the ability to build "flows" which are collections of tasks that can be reused and shared across different pipelines.

In terms of suitable platform, both Apache Airflow and Prefect can be used on a variety of platforms, including on-premises servers, cloud environments, and local development environments.

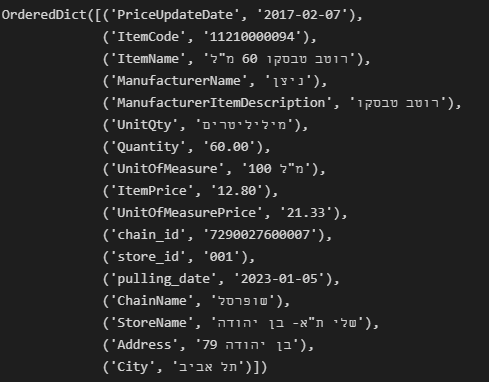
In terms of ease of use, both Apache Airflow and Prefect are relatively easy to use, but Apache Airflow can be more difficult to set up and configure due to its more complex architecture. Prefect, on the other hand, is designed to be more lightweight and easy to use, and it provides a user-friendly web interface for managing and monitoring pipelines.

**Appendix**

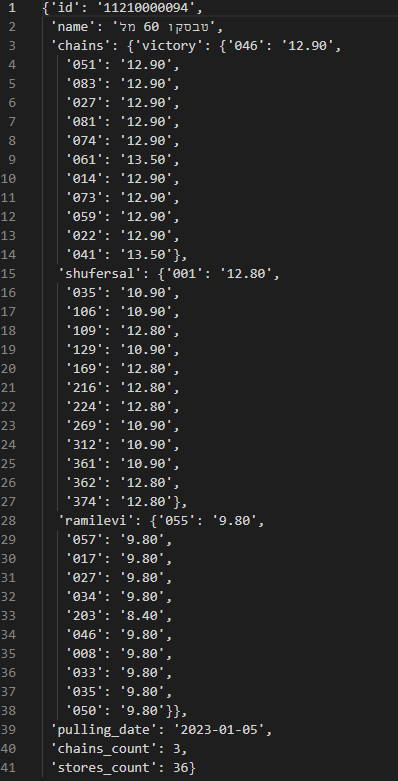
Product consolidation example:

Each product can appear several times for each store and in more than one retail chain

For instance the following product information appears in 36 stores, in 3 different retailers:



The difference between each store is the store id, retailer id and the product price, our python scraping script, consolidated it to the following json format:



This condense format stores only the most important information