Beeg Data - Beeg Meme – Final Report

Kacper Grzymkowski, Jakub Fołtyn, Illia Tesliuk, Mikołaj Malec

Introduction

This document represents the finalization of the project titled “Beeg Meme”, realized during the Big Data Analytics course. In it, we will present the project in its final version, providing descriptions of its modules in separate sections. What is more, we will also provide the project’s results, as well as their assessment. We will also describe the benefits of the project from the user’s point of view and present the impact of deploying the project on society from the ethical and economic perspective. Finally, a detailed contribution of all the group’s members will be provided.

Changes in the project structure

Unfortunately, due to some changes in the project team’s structure, we were forced to change the plan for the implementation of the presentation layer. As such, we provide an additional section called “Presentation module”, in which we will present the serving layer as it was ultimately realized.
We also changed the *EasyOCR* model to *TesseractOCR*, due to memory leakage issues.

Project description

This project represents a platform for the analysis of humorous images posted on the internet, known as “memes”. It allows for combining images from different platforms in a real-time manner (they are processed as soon as they are posted on the platform). Such swift processing allows for a plethora of possibilities, such as a prediction of a given image's popularity or the detection of a new, rising trend in memes.

Benefits from the user’s perspective

The benefits have not changed in a significant way since the project was first proposed. They include:

- **Meme creator support**: Meme creators can use our app to find the most promising memes and trends, allowing them to discover which memes are most likely to yield the highest reach and engagement.
- **Comprehensive trend analysis**: Researchers can delve into sentiment analysis and meme effectiveness. They will gain insights into not only what memes are trending but also how they resonate with users on a daily basis.

- **Targeted advertisement creation**: By analyzing the current meme trends, as well as the sentiment they possess, advertising developers can create highly targeted and engaging advertisements. This ensures that their ads resonate with the intended audience, leading to higher conversion rates. What is more, our application allows to detect the most popular users on platforms (meaning: users that provide the most popular content). Such users may be approached for the role of counselors for current trends, to better suite a marketing campaign to the users of a given platform.

Data sources description

In this section, we will describe all the different data sources in detail. It is worth noting that the data sources themselves and the way they are extracted have not changed. As such, this description will be similar in structure and content to the one that was provided during one of the previous deliverables.
Each source will be described in its separate sub-section.

Reddit.com

Data from the Reddit API is acquired through an official API channel, which is supported by the PRAW (The Python Reddit API Wrapper) library. The script that downloads post metadata in JSON format from the API is invoked once every 12 seconds. The API returns a lot of metadata, the majority of which is technical in nature and matters little for the project. However, all the data returned from the API is passed to further processing in NiFi, in particular to be saved in the master log. A trimmed down (in terms of fields) message is then passed to Apache Kafka for further processing.
Important features of the post (submission, new entry) include:

- Post url (Usually a direct link to an image/video. Extensions inclue **.jpeg**, **.jpg**, **.png** and **.gif**)
- Post title
- Author name
- Post scores (ups/downs/score, however reddit deprecated the downs field a long time ago)
- Number of comments
- Self text (Usually treated as an extension to post title or the body of a “self” post, which do not contain images)
- Awards (paid interactions with the post)
- Date of creation (in seconds since UNIX epoch, UTC timezone)

We are acquiring data from a subreddit dedicated to posting humorous images, called **r/memes**. This subreddit is fairly active (as for Reddit’s standards), which results in 1 new image per minute or two on average.
It is also worth noting that posts acquired from this source are not filtered by the “newest added” category. As such, those posts usually have already gained some popularity on the website – as such, their score values are significantly higher than those from other data sources.

Tumblr.com

Data from the Tumblr API is acquired through an official API channel, which is supported by the official Python API Wrapper pytumblr. The Python script receives information about a predefined number of posts from a particular tag category in a JSON format. Tumblr posts from the #memes category can be represented as images, videos, or texts. Since our models operate only on images, the script filters the received data by the availability of image fields. Each Tumblr image post contains an XML string with URIs of the meme image in different resolutions. The script uses an xml module to parse the string and retrieve a link to the largest resolution image. To compensate for non-image posts filtered out previously, the script sends a new API request with a condition for posts to be older than the previous batch. This way, the predefined number of records is obtained. The maximal number of posts which can be received from a single tag category is limited to 20. To obtain the larger number of posts, the script repeats requests following an above-mentioned approach. Each meme post contains a huge set of meta-data, however, only the following fields can be used in our project:

- blog (author) name
- posted date
- total number of interactions, which is a sum of:
  - number of likes
  - number of replies
  - number of reposts
- description (often empty)
- link to the image

We are acquiring data from the tag #memes, as it is used to post humorous images. Unfortunately, the amount of new images posted in this tag is relatively low (one post every 10-15 minutes).

Imgur.com

Data is acquired through the use of a dedicated Imgur API Python library – imgur-python. This library enables users to both acquire posts from the website, as well as post their own images, comments, and also browse other users' accounts and comment sections.
Imgur API enables users to get posts from multiple post flows: **top** posts, meaning the posts with highest scores (in the last week, day etc.), **hot** posts, meaning posts that are quickly gaining attention, "trending", and finally a category called **added by users**, which simply means newest posts.
In general, new posts come in two different variations: a single image or a gallery of multiple images. The script that we have prepared "unpacks" the gallery so that each image in the gallery is treated as a single image. What is more, we omit the gif-type images (so animated images) in our data acquisition. These are the reasons why sometimes, when passing the parameter **n** (which corresponds to the number of posts that we want to retrieve), the number of actually retrieved images may vary. We have since updated the script, so that it always provides at least **n** images.
Due to internal limitations, the API always returns 60 posts. We are acquiring posts from the "added by users" category, as this is the most frequent category, so the category in which we can obtain new images the fastest. API returns information about posts in the form of JSON

files. Those files include various information about the post. Some of the most important ones are:

- poster name and id
- posted date
- number of comments
- number of ups/downs
- number of views
- post topic (often empty)
- post title
- post description (often empty)
- link to the image (may be gif)
- Image width and height
- author id and name

When investigating, we have noticed that the API returns new posts with a slight delay (meaning that the "newest" posts returned by API have actually been posted around 15 minutes ago). In general, it is hard to measure the number of new posts, as it varies with regard to e.g. the time of the day, but according to our tests, on average around 1-3 new posts are added every minute.

EDA

We have conducted an Exploratory Data Analysis on a data extract taken for days 01-04-2024 and 01-05-2024. The data was saved in the form of *.json* files, where each file contained a single record. It is worth noting that both unique (so images that were newly-added) and repeating images were stored in these records. The EDA was conducted using *Python* language.
Our first step was to load and combine all the records. It resulted in a single pandas dataframe. The dataframe's head is as follows:

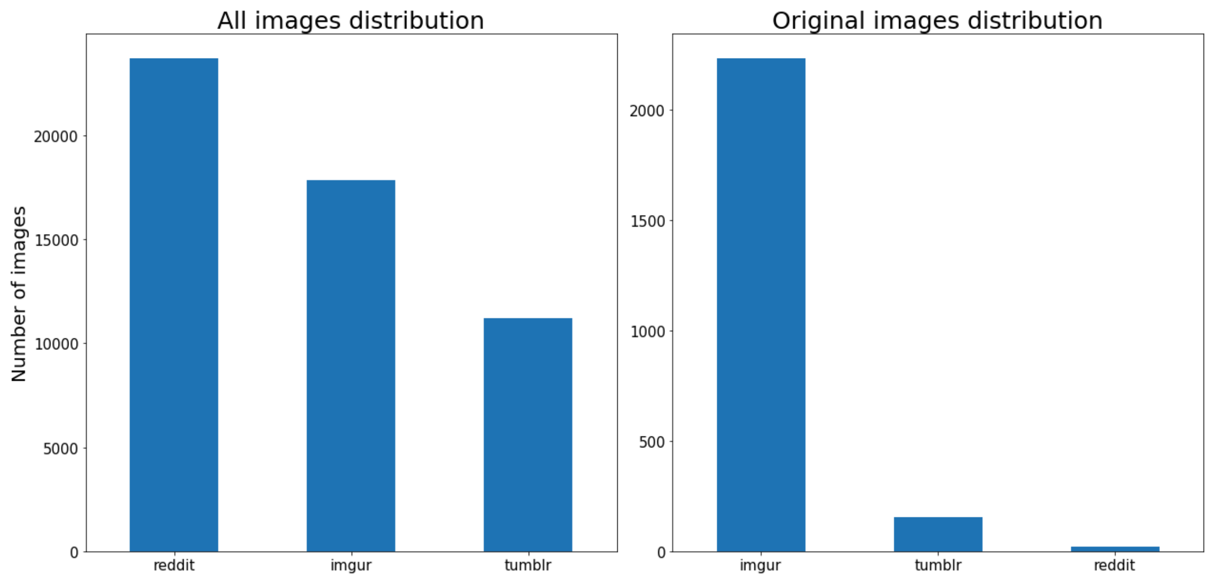
	global_id	author	created_time	desc	score	url	source
0	02Zyk8U-imgur	AzrielStrife	2024-01-04T21:13:08	The Fly https://youtu.be/ECFAMTEnvqw?si=s5tePl...	3	https://i.imgur.com/02Zyk8U.jpg	imgur
0	02Zyk8U-imgur	AzrielStrife	2024-01-04T21:13:08	The Fly https://youtu.be/ECFAMTEnvqw?si=s5tePl...	3	https://i.imgur.com/02Zyk8U.jpg	imgur
0	02Zyk8U-imgur	AzrielStrife	2024-01-04T21:13:08	The Fly https://youtu.be/ECFAMTEnvqw?si=s5tePl...	3	https://i.imgur.com/02Zyk8U.jpg	imgur
0	02Zyk8U-imgur	AzrielStrife	2024-01-04T21:13:08	The Fly https://youtu.be/ECFAMTEnvqw?si=s5tePl...	3	https://i.imgur.com/02Zyk8U.jpg	imgur
0	02Zyk8U-imgur	AzrielStrife	2024-01-04T21:13:08	The Fly https://youtu.be/ECFAMTEnvqw?si=s5tePl...	3	https://i.imgur.com/02Zyk8U.jpg	imgur

This dataframe contained 52,764 records. As we can see, there are numerous repeating posts (as the sources return the latest posts, sometimes no new posts have been added since the last query). To combat this, we also created a dataframe with only unique posts:

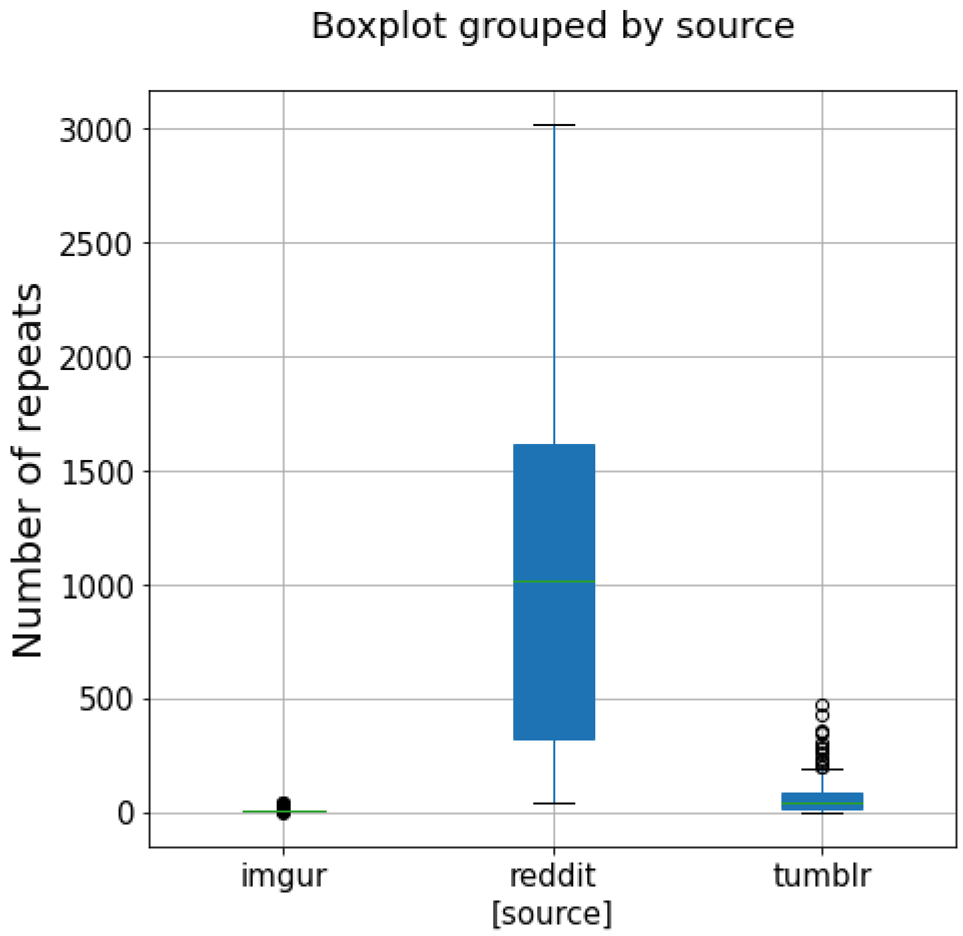
	level_0	index	global_id	author	created_time	desc	score	url	source
0	0	0	02Zyk8U-imgur	AzrielStrife	2024-01-04T21:13:08	The Fly https://youtu.be/ECFAMTEnvqw?si=s5tePl...	3	https://i.imgur.com/02Zyk8U.jpg	imgur
1	5	0	04FNenk-imgur	Tinkywink	2024-01-05T00:33:49	None	6	https://i.imgur.com/04FNenk.jpg	imgur
2	10	0	0at1r90-imgur	linexnewt	2024-01-04T23:16:24	Music by Queen	4	https://i.imgur.com/0at1r90.jpg	imgur
3	15	0	0BoiFPH-imgur	jensyao	2024-01-04T01:45:11	None	-3	https://i.imgur.com/0BoiFPH.jpg	imgur
4	21	0	0BTHRI5-imgur	VanessaBludgeons	2024-01-05T06:54:57	Poster for the series V	3	https://i.imgur.com/0BTHRI5.jpg	imgur

It contained only 2,408 records, which presents the scale of the repeating posts.

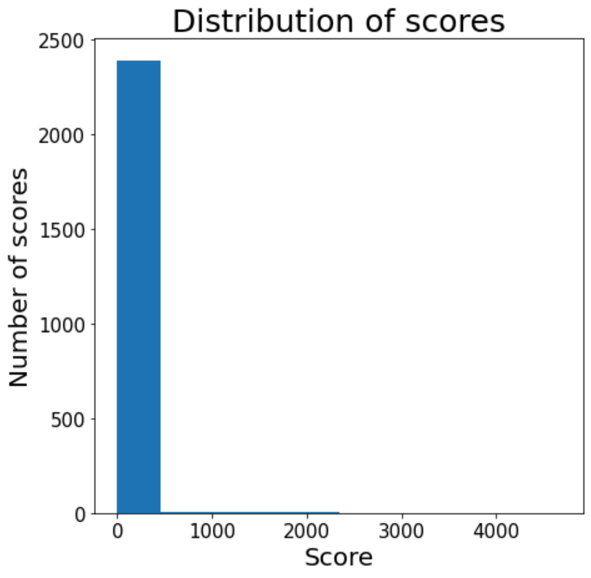
To further investigate how many repeated posts are in the data, we drew their distributions as histograms:
On the left, we have the number of posts in total per source, on the right – number of original posts. As we can see, while Reddit dominates in the number of posts downloaded, it also contains the least original posts. Imgur clearly dominates in that regard – it may be because we are downloading all possible posts from Imgur, while in other sources, we only confine ourselves to the “meme” subsections of the sites.

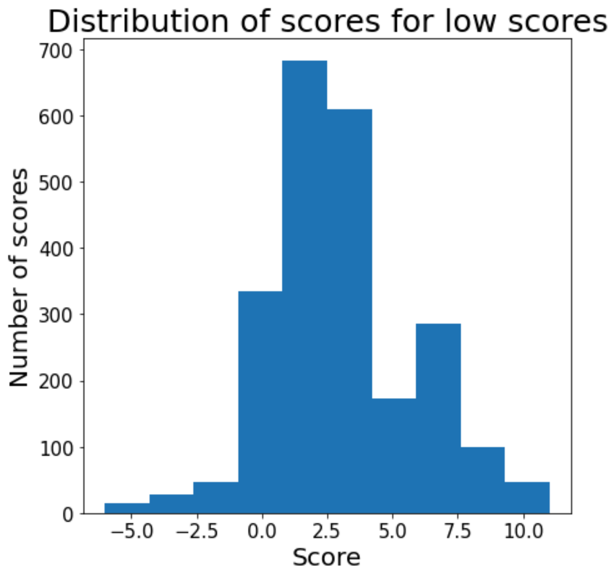


The distribution of repeats per source was further investigated:

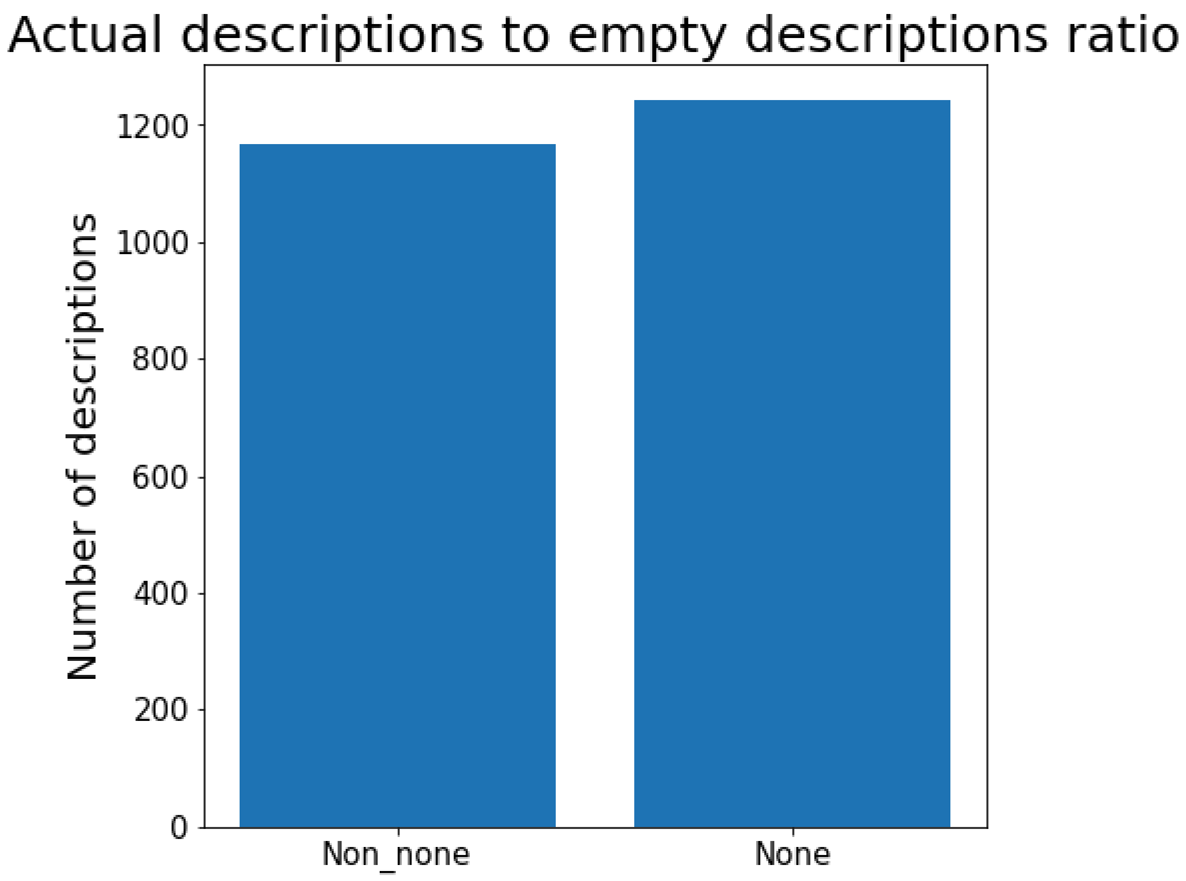


And it confirms our previous findings, with Reddit having the highest amounts of repeats per post (ranging all the way up to 3,000 repeats). It is worth noting here that despite these large amounts of repeats, the notion of collecting this data is still valid – as even if the image itself is not changing, its statistics are – in particular, its score may be updated with each query.
Next step was to investigate some of the field present in the data. First, we researched the score statistics:

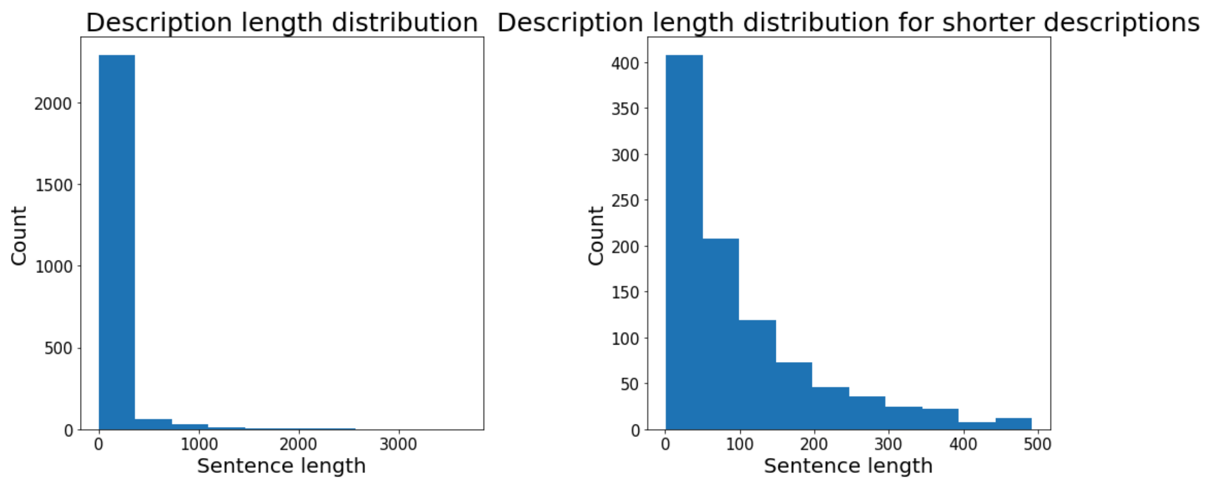




On the left we have the total distribution of scores, and on the right – the distribution of only lower scores (those lower than 10). As already mentioned, Reddit scews heavily the first diagram, that is why we decided to investigate also only the lower-scoring posts (as those come from Imgur and Tumblr. As we can see, the lower-scoring posts have a clear tendency to distribute themselves in a normal distribution fashion, which is not a surprising outcome in the domain of statistics.
Next, we investigated the descriptions. A significant number of them is left empty in posts – we decided to check their number in relation to the non-empty ones in original posts:

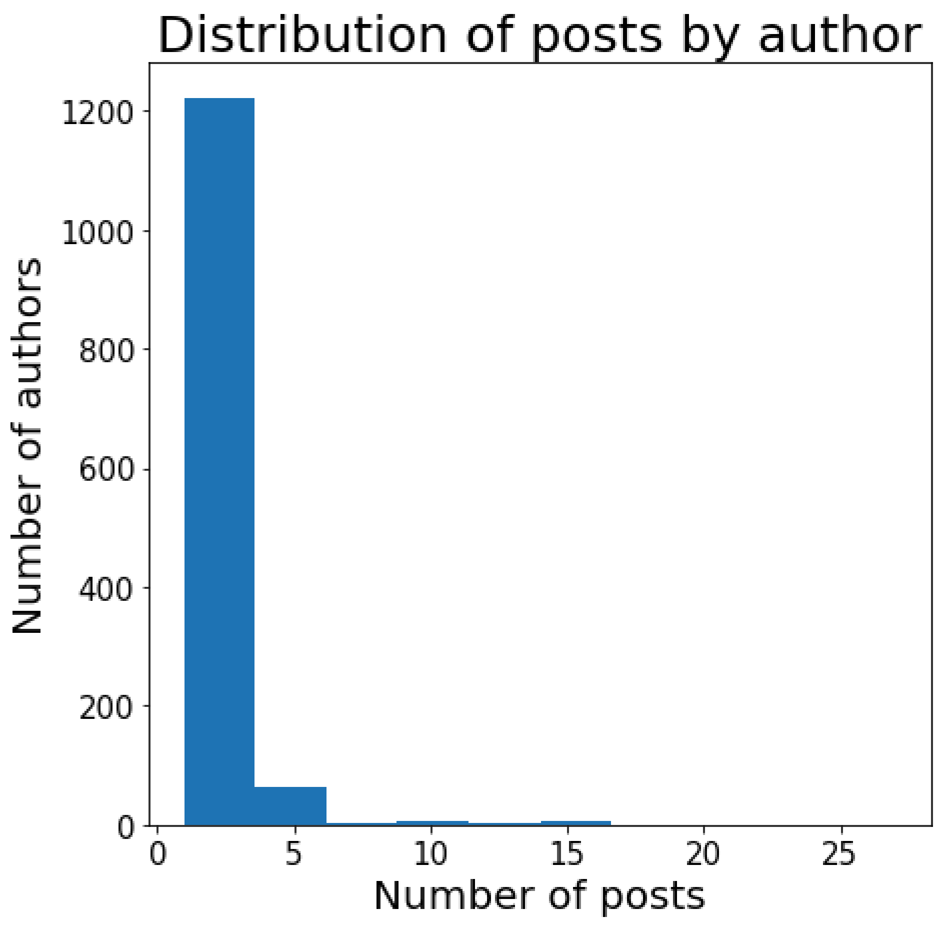


As we can see, the majority of descriptions is left empty, but the difference is not that significant. We will now investigate the non-empty ones, taking their length distribution:



On the left is the full distribution on description lenghts, on the right – the same distribution, but restricted only to lengths below 500 characters. As we can see, both distributions are rather skewed, with the vast majority of descriptions being a few characters long. The longest description contained 3,017 characters.

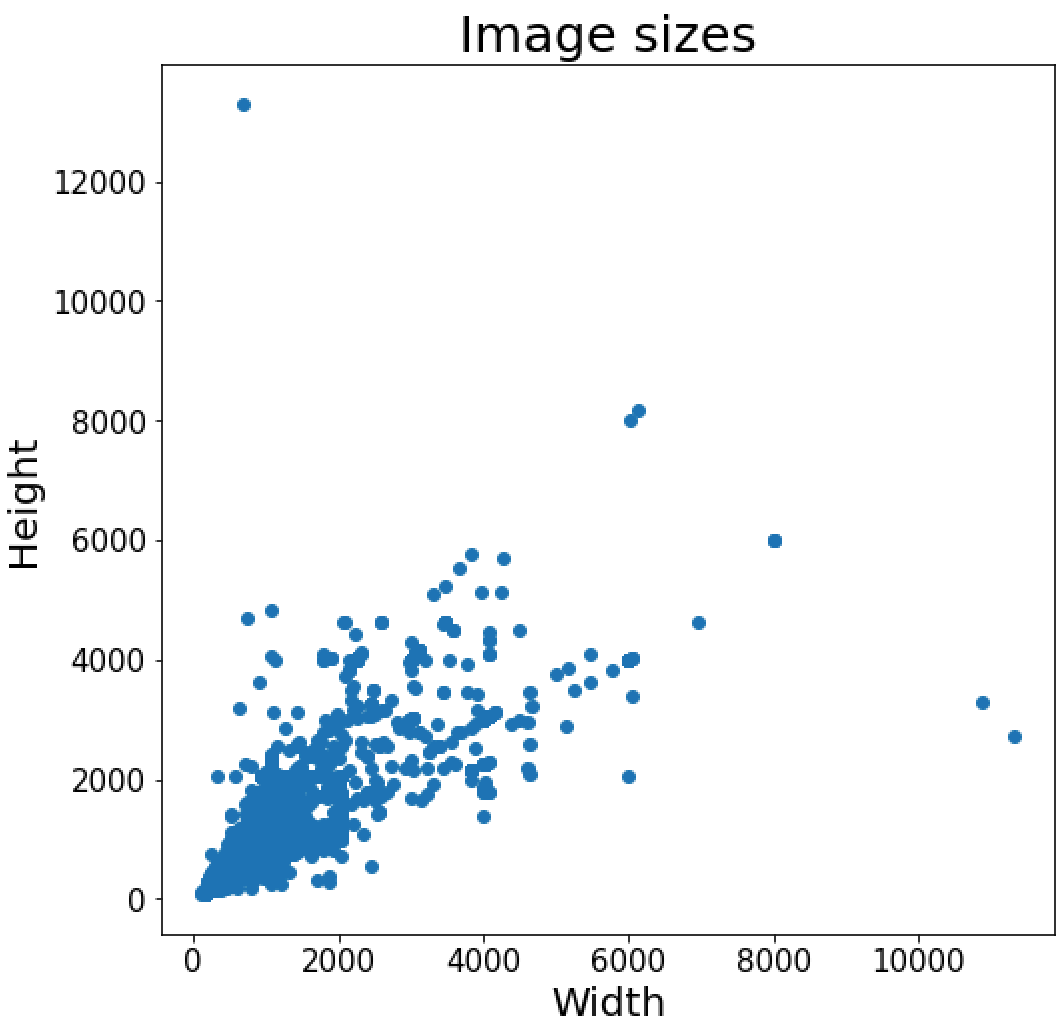
Next step was investigating the number of posts by author:



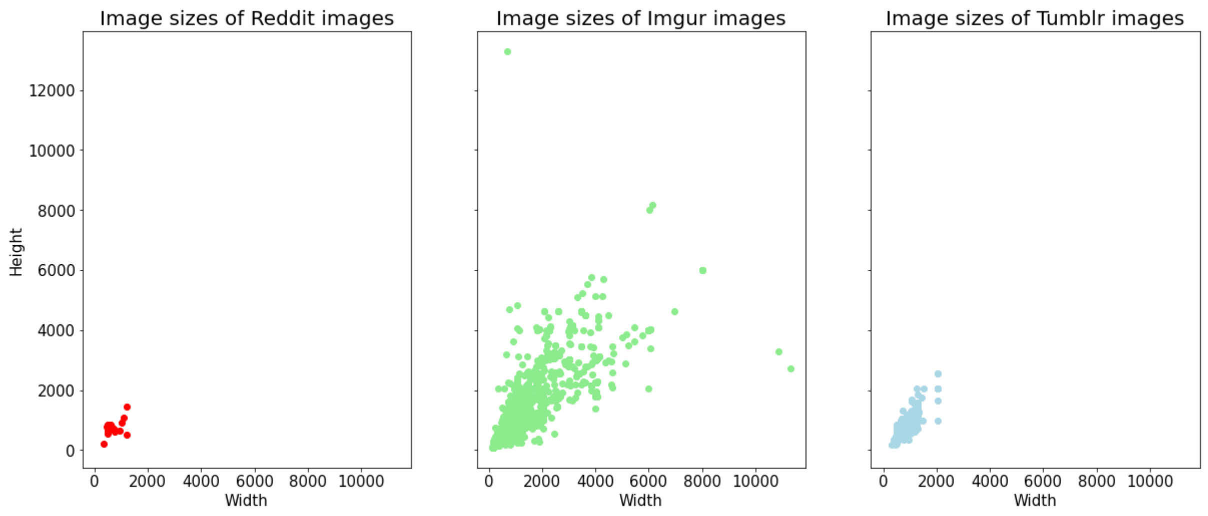
Again, the distribution of authors per number of posts is skewed towards left, meaning that most authors post only 1-2 posts, which is understandable in such a short period of time. The record taker author is named “Crazyzaku”, who made 27 posts in only 2 days.
The images associated with posts come with various different formats. We decided to perform a quick check of distributions of these formats:

```
{'jpg': 1945, 'png': 435, 'none': 7, 'jpeg': 12, 'gif': 8, 'webp': 1}
```

As we can see, **.jpg** files clearly dominate the number of files downloaded. The “none” field means that the post did not contain any easily-downloadable medium.
Finally, we decided to investigate the distribution of image sizes among posts. The results of overall distribution are presented here:



And with regard to the source:

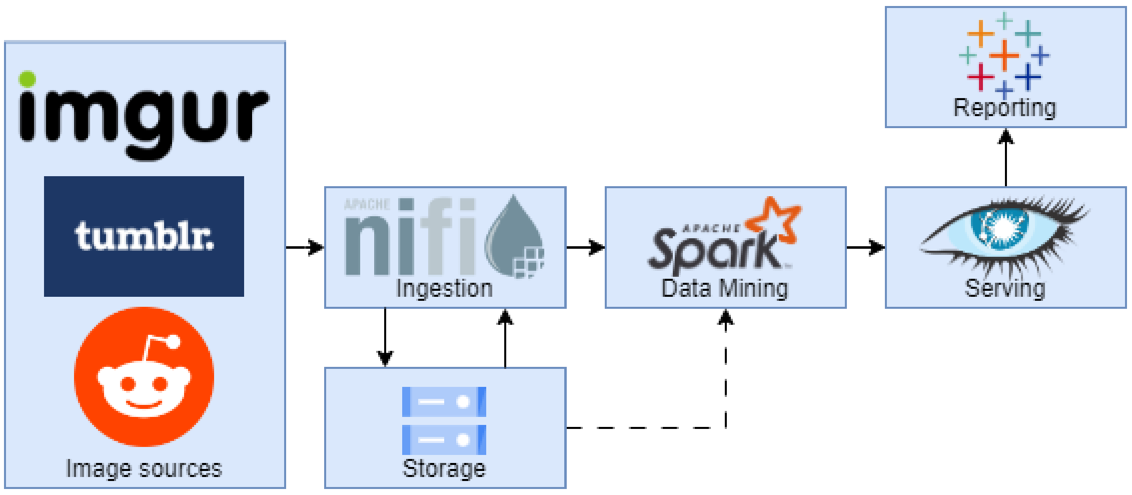


As we can see, images from Imgur dominate the original plot’s shape, which is understandable. What is more, all the outliers of image size also come from Imgur. Other than that, the majority of images are rather smaller in size, and have below 2,000 pixels in width and height.

System architecture

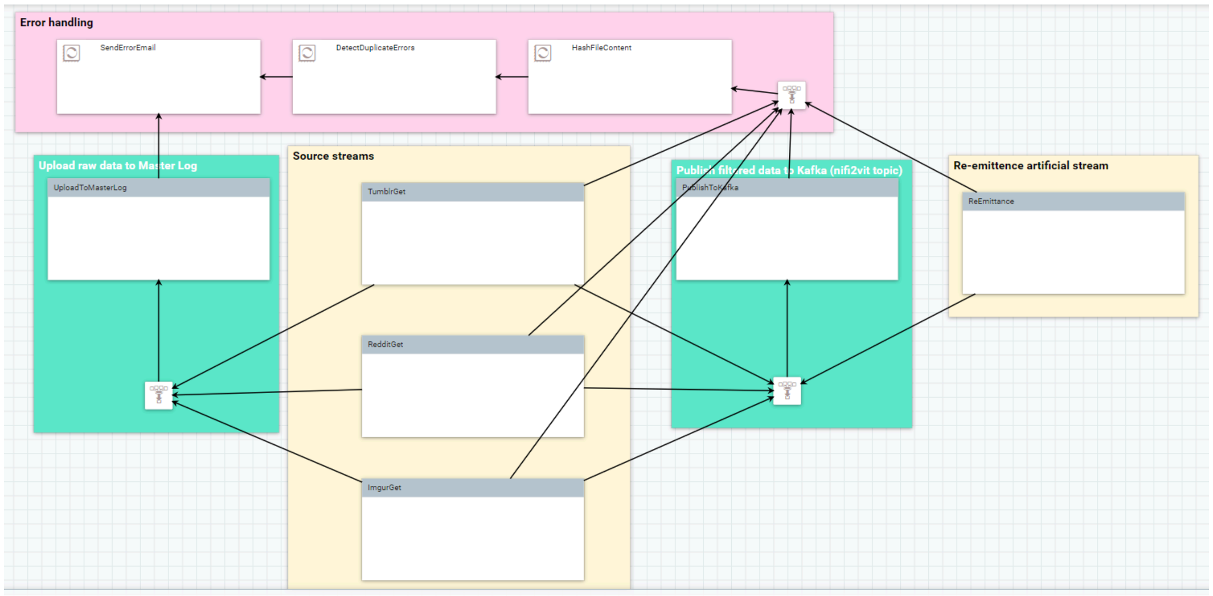
In this section, we will provide the various diagrams representing our solution, along with their descriptions.

Overview diagram



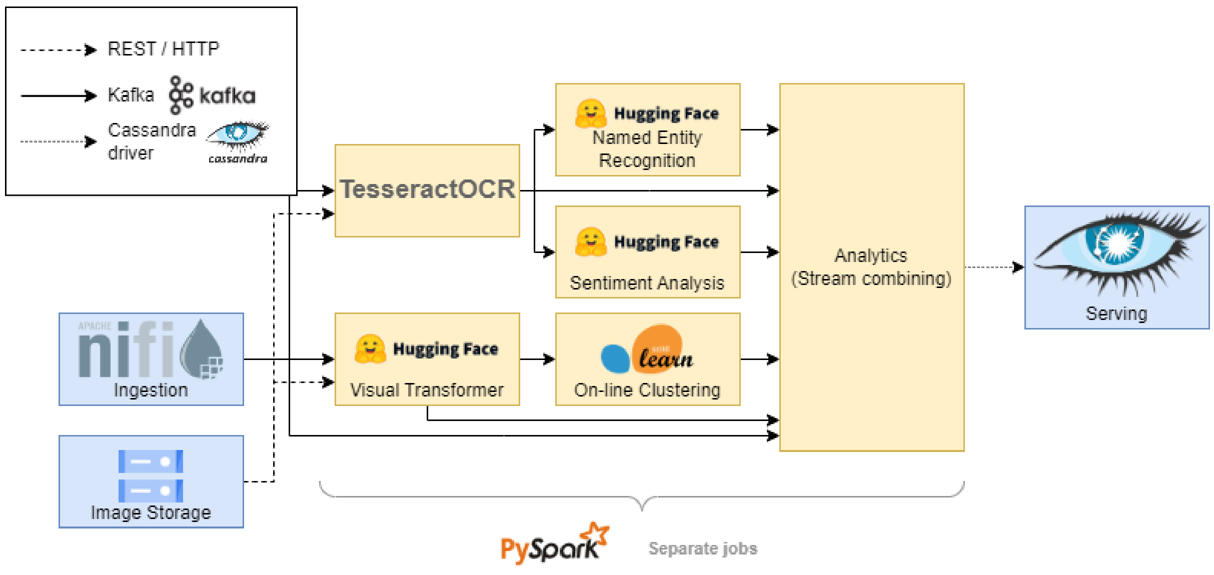
The overview of the system is fairly simple. The Ingestion module handles taking the images from their sources, running their API calls. It also handles saving the images and metadata to storage, and allows re-emission of past events, as per the Kappa architecture. It pushes the images and their metadata to the Data Mining module which analyzes the images using a variety of techniques to extract interesting information. The images are not sent directly, but accessed from the storage instead. The results of these analyses are saved in the Serving module and visualized using the reporting module.

Ingestion diagram



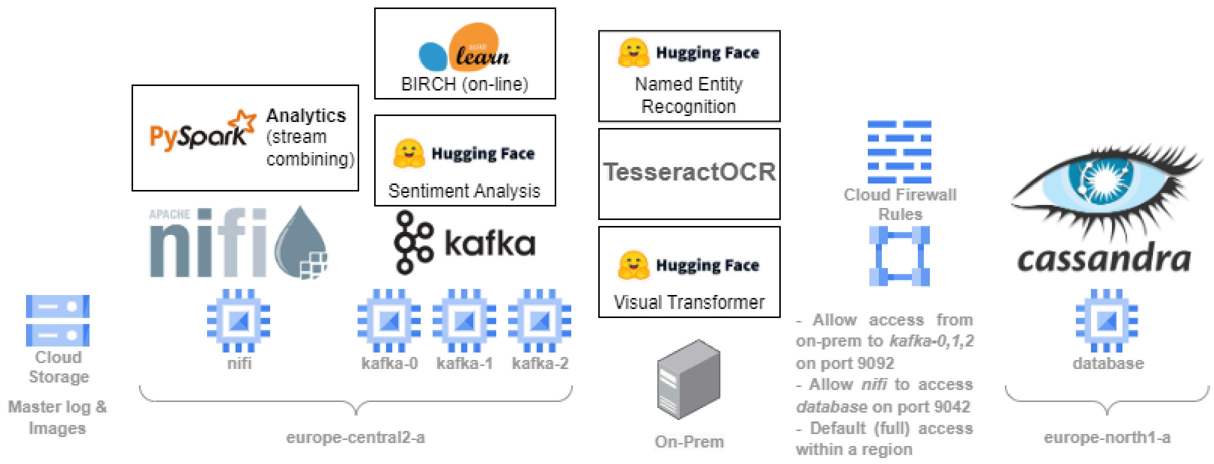
The ingestion layer is divided into several parts. In the center, on the yellow background we can see the 3 source streams, that acquire the data and perform the preliminary pre-processing. The raw, unprocessed data is sent to the left, merged and uploaded directly to the master log. On the right, processed and unified data is being published on multiple Kafka topics (to be later consumed by various models). What is more, on the farthest left, there is also a re-emittance path, on which we can take historical data straight from the master log and publish it to the Kafka. This path is often used to artificially enrich our data streams. Finally, at the top, on the pink background, there is an error-handling path. On it, errors from the whole workflow are de-duplicated and then sent in a form of emails to an email account set up precisely for this very purpose.

Data-Mining diagram



A detailed description of the components is contained in the Data-mining module section of the document.

Infrastructure diagram



Due to constraints in the free trial version of GCP, we were limited to 8vCPUs per region, and 12vCPUs in total. Unfortunately, for this purpose, the shared-core machines count as the maximum number of vCPU they can offer, that being 2 vCPUs. This means that we were limited to 4 compute engine instances per region, and we were forced to make some *creative* decisions on the placement of services.

The NiFi ingestion module is run on the **nifi** instance in GCP, alongside the **analytics** module which combines the streams together. This was done as the Spark driver for Cassandra is not updated to Spark 3.5.1, so we did this to avoid having to downgrade the spark version everywhere else.
The master log and image storage is stored on Google Cloud Storage in separate buckets.
The instances **kafka-0,1,2** are configured to run Kafka in a cluster setting. We initially wanted to set up a separate Spark cluster, but we were unable to manage this due to the limitations in the GCP free trial. As a result, Spark is also configured in a cluster setting alongside Kafka on these instances. On this cluster we run the **on-line clustering** module, as well as **sentiment analysis**. We initially wanted to run **Visual Transformer** on this cluster in a distributed manner to demonstrate scalability, but the small size of the instances made this extremely inefficient and slow, so we decided to move it on-prem.
On-prem required some set-up from the GCP VPC Firewall, and setting some insecure rules (allowing traffic from a select IP address), but we did not have time to set up a proper solution (VPN). We run the **Named Entity Recognition, TesseractOCR** and **Visual Transformer** modules on-prem.
The **database** instance was also configured to have firewall access, as it doesn't have it by default across regions. We put it in a separate region due to the above-mentioned free trial constraints. As the name implies, this is where **Cassandra** is installed.

Data ingestion and transformation

Please note that this section is the combination of sections created during previous deliverables, with an addition of description of all the modifications that have been made since then.
We used Apache Nifi as the ingestion tool in our project.

Data acquisition

We acquire data from all the sources with the use of Python scripts. We have created three scripts in total, each of them uses their data source's respective API library:

- **praw** – for accessing data from *reddit* data source
- **imgur-python** – for accessing data from *imgur* data source
- **pytumblr** – for accessing data from *tumblr* data source

The above libraries have been described in greater detail in their respective data source's sections.
In addition to source-specific libraries, we are also using some more "common" Python libraries, such as:

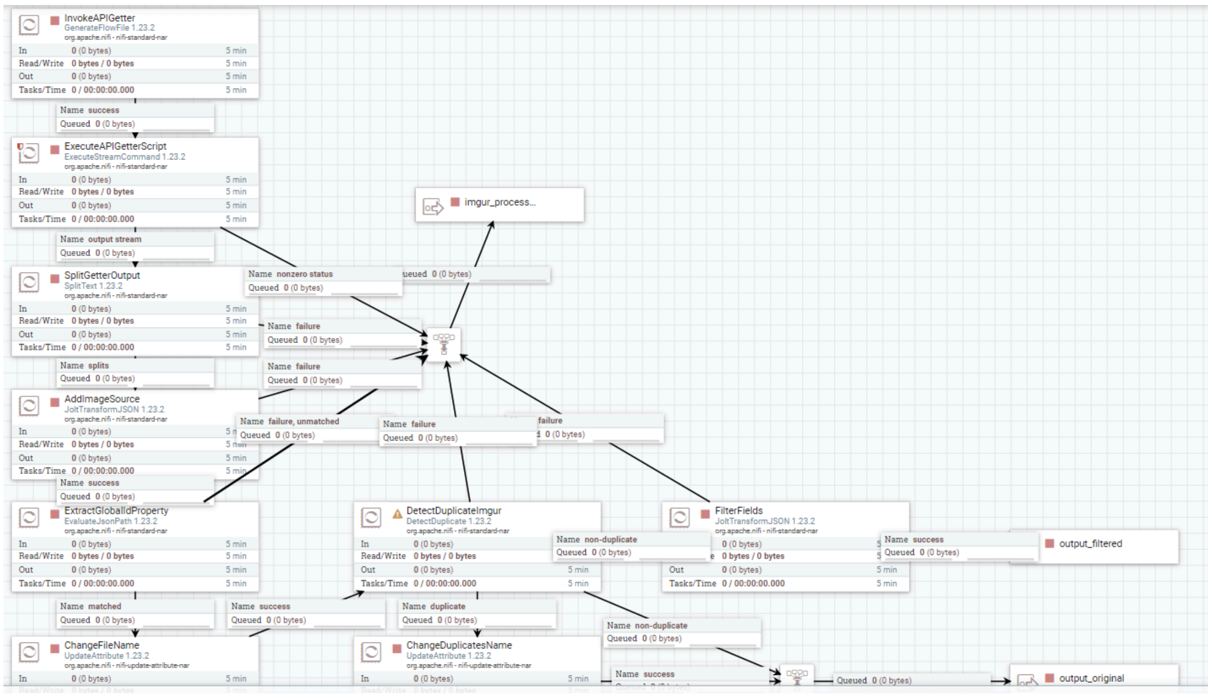
- **datetime** – for retrieving dates in UTC ISO format
- **json** – for converting printed data into json objects

- **requests** – for retrieving website data

Each data acquisition script is invoked by the Nifi *ExecuteStreamCommand* processor. It is achieved by sending an empty flow file to this processor every 12 seconds. Each time, each scripts prints out **n** latest images from its source (where **n** is a parameter). The **n** parameter is set to 2 for each source, as not to produce too many repeated posts. All the collected images are then further processed using Nifi.
As an additional processing step during the data acquisition part, we have also created two new Python scripts: *loadImage.py* and *saveToBucket.py*. Using this scripts, after having obtained the posts from the source, but before the posts are further processed in Nifi, we are able to retrieve the images and then save them in our created Google Cloud Storage Bucket – *images_beeg_meme*. Each image is saved under its original URL path. This way, we are able to easily retrieve the images later on, even if they are no longer available in the original source. The saving process is de-duplicated, meaning that only the original images are saved, and duplicates are ignored.

Data transformation and storage

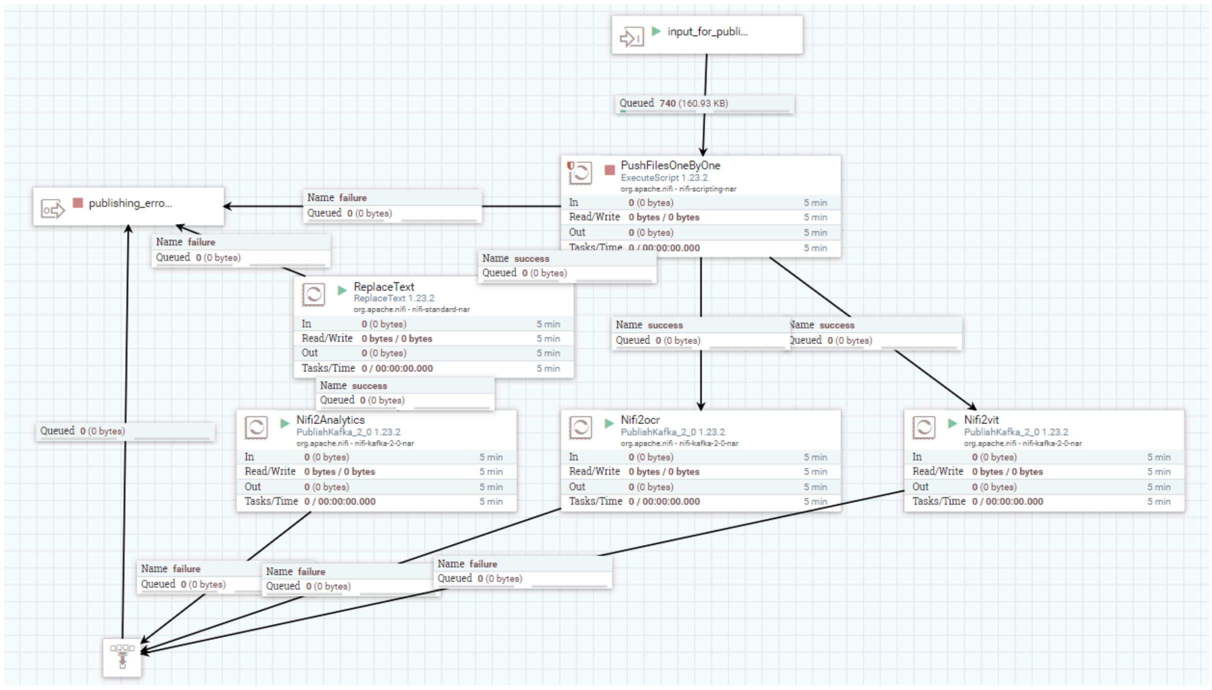
As previously described, data is acquired via Python scripts and transformed using Nifi. In general, there are three distinct source flows – one for Reddit data, one for Tumblr data, and one for Imgur data. Inside these flows, the operations performed are quite similar.



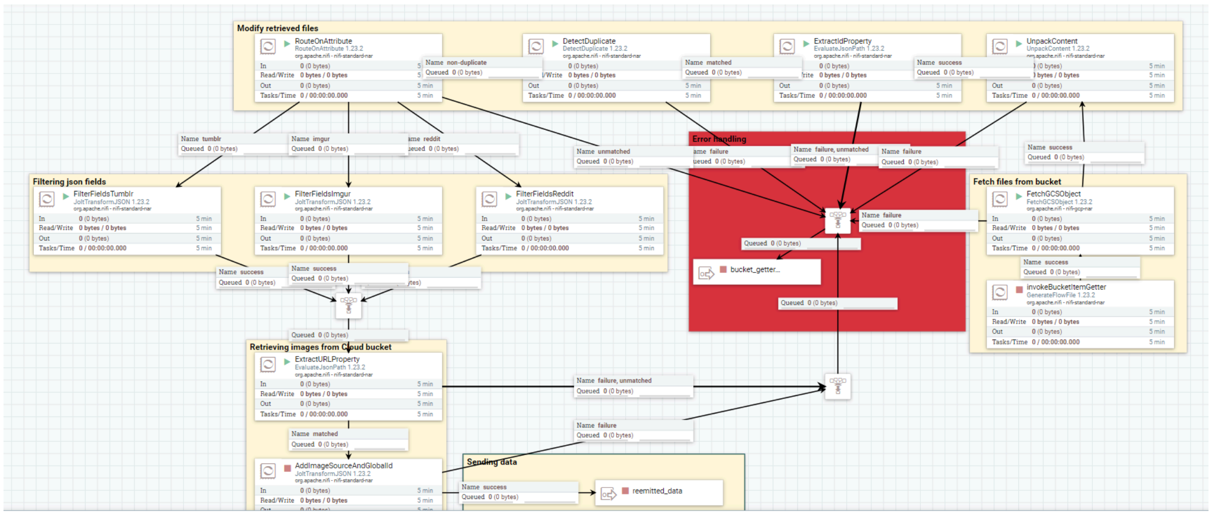
In the image, we can see an example flow for Imgur data source. In general, the API invoking script is started by passing an empty flowfile into an ExecuteStreamCommand processor. This processor then produces a single output flowfile with a number of concatenated JSON files. Those files are then split, so that each object is a single flow file, the source of the image is added to the file (in case of of reddit in tumblr, this is also the place in which global_id is added, which is of the form [image_id]-[source]), and then the filename is changed to the global id of the file. The files are then checked for duplication, and non-duplicates are passed to a filtering processor, which chooses only the fields in the JSON file, which are consistent across all data sources. They are:

- **Author**
- **Image description**
- **Date of image creation**
- **Score (number of upvotes-downvotes)**
- **Image source**
- **Image global id**

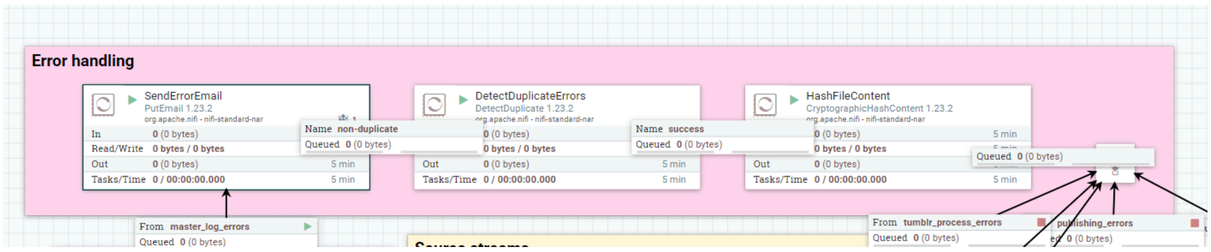
Duplicated images have their filename modified (a timestamp is added), and are passed with the non-duplicated ones in their original form to be posted to master log. Master log is a cloud storage bucket. The files are merged into a TAR file and then uploaded into the bucket. We are merging 10,000 files at once (meaning that each file on the bucket contains around 10,000 records).
The deduplicated files, on the other hand, are then merged into one stream with all the fields consistent. Those files are then sent to be published by a KafkaPublisher processor. The publishing process is as follows:



As we can see, there are three topics to which the files are published simultaneously: Nifi2Analytics (for analytics purposes), Nifi2ocr (for the OCR model) and Nifi2vit (for the VIT) model. For the purpose of the analytics module (in which the stream merging between results of different models is performed), the ingestion time is added to each record (meaning the time they are published in UTC format. What is more, there is also another processor (*PushFilesOneByOne*), that ensures that when using the re-emittance path, the files are sent one-by-one, rather than being published all at once.
We have mentioned the re-emittance path. Its flow is as follows:

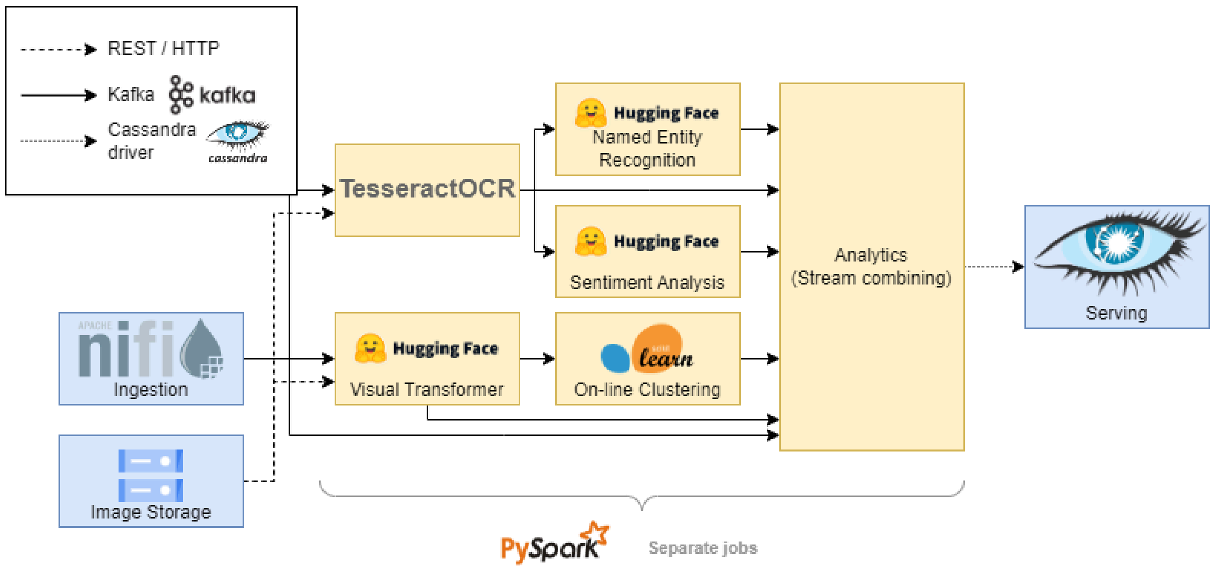


The name of the file we want to retrieve from the Master Log is passed as an argument in the first processor on the path (the right-hand side). The data is then unpacked and the files inside are processed as to bring them into accordance with the rest of the data that is processed in Spark (meaning that the fields are unified, just as it was described above). Finally, the only major change is that the URL address is modified – instead of the original address, the address to the *images_beeg_meme* bucket resource is passed. The idea here is that images from possibly long ago could have been deleted in the original resource, so the copy that we have saved should be retrieved instead.
Finally, there is an error-handling path:



It is located at the top of the main workflow. In general, all the errors encountered during any stage of preprocessing are sent there. Here, they are de-duplicated)so that only original errors are passed), and then sent to a dedicated email address.

Data-mining module



We run a number of machine learning models in order to mine useful information out of the images we process. In particular, we used a pre-trained MobileViT2 model from HuggingFace - a lightweight version of the ViT transformer designed for mobile phones. This original ViT model is too heavy-weight and slow for our system, therefore a faster transformer was required to achieve acceptable inference times. Currently, MobileViT2 requires approximately 1 second per image with part of the time being taken by image downloading. MobileViT2 is used for image processing and produces more compact image representations which are next used by the clustering algorithm.

Since we are implementing a Kappa architecture, a standard image classification model or even K-Means clustering algorithm are not applicable. With streaming architecture in mind, we've selected an online BIRCH clustering algorithm and used its implementation from scikit-learn library. We've selected BIRCH due to its iterative nature that allows it to process images one by one as they come from the ViT module, and not the whole dataset at once. We've already tested it and received non-error cluster id as outputs for input memes, however it's difficult to evaluate the clustering performance, as it requires a larger amount of collected data to draw conclusions.

On another branch of our system we perform Optical Character Recognition on the input meme images. Initially, we've selected an EasyOCR python module, that provides a reader capable of detecting english words on images. The algorithm performed quite well and correctly detects the majority of letters, however, sometimes it can make mistakes in a couple of letters. For example '2020's' was interpreted as '2020$' by EasyOCR. However, OCR operation with this module was taking too much computational resources and was significantly slowing the system down. Its inference time was usually taking around 10 seconds on CPU and was even slower on the cloud.

Instead, we've selected a Tesseract OCR module. Substitution of the OCR module led to much faster results and provided more flexibility in language choice.Tesseract provides support for tens of languages and only requires downloading the pre-trained data files for the selected languages. While the goal of our project is to extract information from memes written in English, the incorporation of the Tesseract module provides more flexibility for expanding the set of supported languages in the future works.

Next, the extracted text is sent to Named Entity Recognition (NER) and Sentiment Analysis modules. We use a WikiNEuRal multilingual pipeline for the task of NER and roBERTa-base model fine-tuned onTweetEval benchmark for the task of Sentiment Analysis. Both these pre-trained models were taken from the HuggingFace. While the choice of OCR implementations is quite limited, HuggingFace offers a wide range of models for different NLP tasks or Visual Transformers, therefore we'll be able to test other pre-trained NER and Sentiment Analysis models as well.

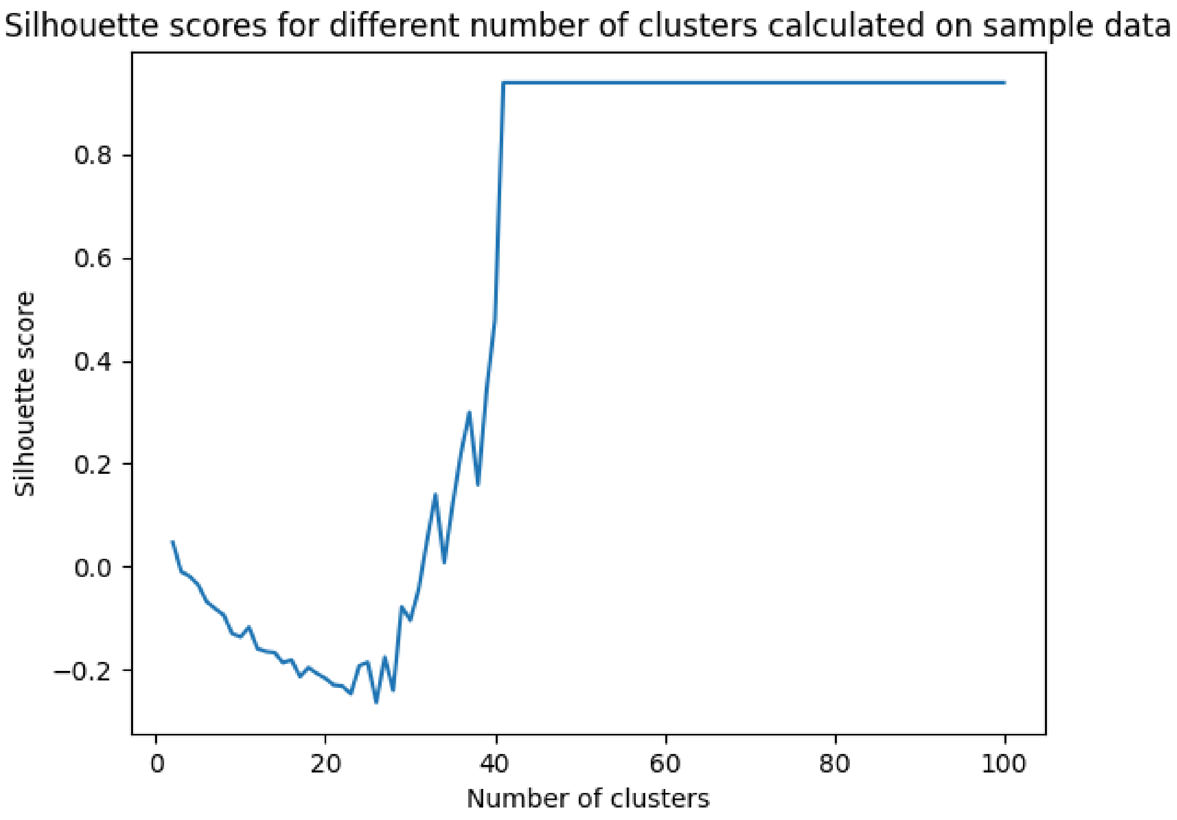
All three HuggingFace models we used, achieved SOTA results on their respective benchmarks. It is difficult to evaluate their performance on our data, as we do not possess stationary dataset with NER, OCR or Sentiment Analysis labels. Contrary to these three tasks, clustering is an unsupervised task and possesses a list of evaluation metrics that can be applied to our system. Sklearn offers several evaluation metrics that do not require ground-truth labels, namely silhouette coefficient, homogeneity, completeness, V-measure such, Calinski-Harabasz score, Davies-Bouldin index, contingency or pair confusion matrix. Either of these metrics can be used to evaluate the clustering results as they assess how well the resulting clusters are formed. We will apply some of these metrics when we'll collect enough data to perform a sensible analysis.

Every module is implemented in a separate Spark job. We've selected a Python implementation of Spark library (pyspark). Each module receives Kafka messages from the corresponding sources, extracts and processes the data, and finally sends the results using a Kafka topics to the corresponding jobs

The final Analytics job collects the mining information from all 5 modules and uses a left join to combine these results with the original image metadata ingested into the system via Apache NIFI. The join operation collects the data from different streams by the image's global ID. We also check the processing timestamps of each of the 5 streams and compare it with the NIFI's ingestion timestamp. In order for the model stream to be joined with the basic NIFI stream on the same ID, the difference between its processing time and the ingestion time should not exceed 50 minutes. Additionally, we use watermarking with a 5-minute threshold. This helps to prevent records with data collected from only a part of sources from staying in memory indefinitely while waiting for late data from the remaining streams.

Model improvements

To improve the performance of our machine learning model we decided to use a separate processing workflow. A sample of data was used and we decided to optimize the number of clusters based on the silhouette score they achieve. Silhouette score is a method of evaluating clustering without "ground-truth" labels, which we do not have. We ran the clustering algorithm on the embeddings generated ahead of time while changing the *n_clusters* parameter from 2 to 100. The results are visualized in the plot below.



For a low number of clusters the silhouette score was around 0.05. Increasing the parameter made results progressively worse, even dipping into the negatives, until around 25 clusters when the trend reversed, quickly climbing to levels higher than ever before suddenly plateauing at around 0.9. Investigating this sudden leveling we found that the clustering actually only used 40 of the clusters, despite being allowed to create more clusters. Based on this insight, we set the number of clusters parameter of the clustering algorithm to 40.

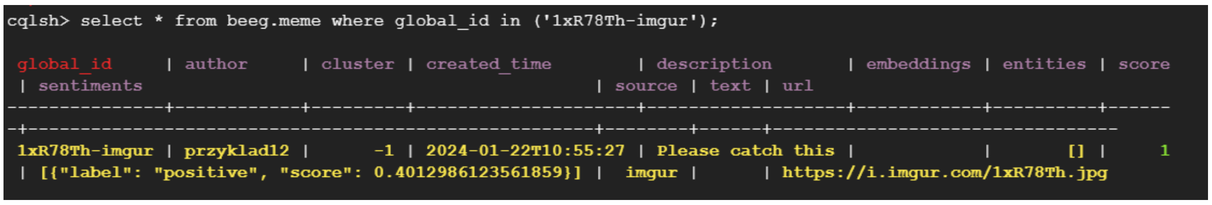
Serving & Presentation module

We have decided to drop the idea of setting up a MongoDB server. Instead, we opted to use the Cassandra database (in version *4.0.11*), as our data is rather structured (meaning that the number of fields is consistent) and is suited for the tabular format. While we do not really utilize the wide column format, we still benefit from the availability over consistency properties of Cassandra, as data consistency is not really important in our project. Another aspect that encouraged us to use Cassandra was the seeming ease with which one can connect to it from the Spark level and the connection support between it and the Tableau software.
The columns in the table are as follows:

- **global_id** – text column, also the **primary key** of the table
- author – text column,
- created_time – text column,
- description – text column,
- score – int column,
- url – text column,

- source – text column,
- embeddings – text column,
- cluster – text column,
- text – text column,
- entities – text column,
- sentiments – text column,

Example row from the table:



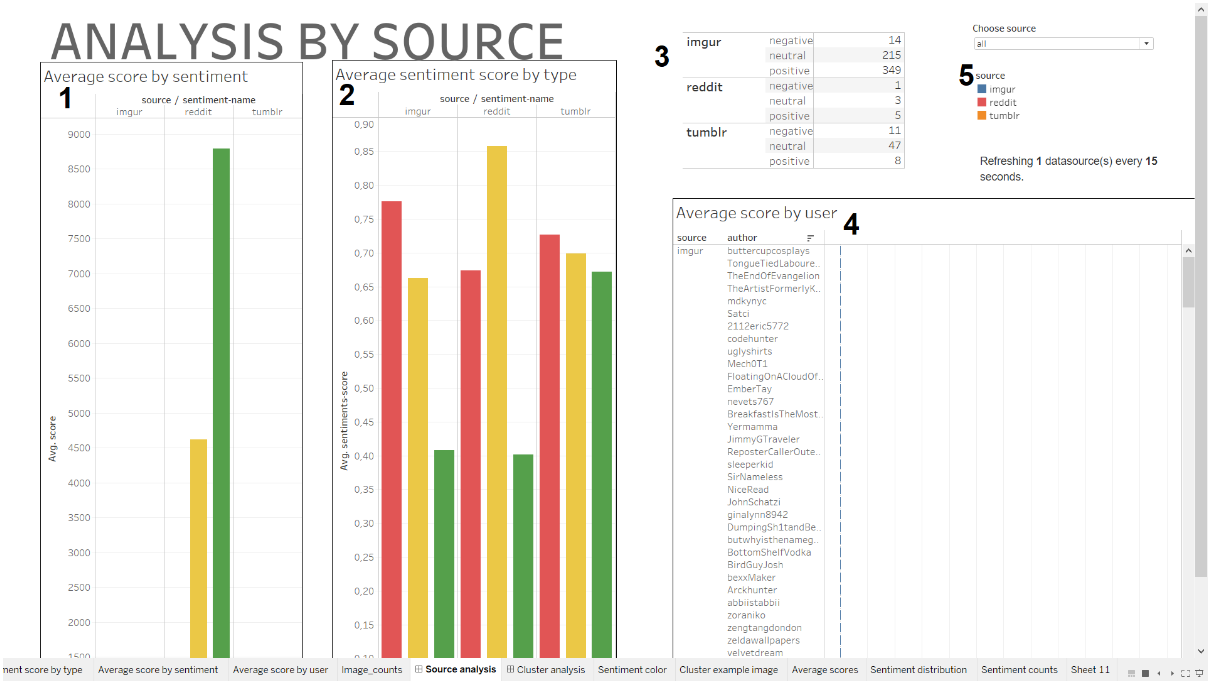
```
cqlsh> select * from beeg.meme where global_id in ('1xR78Th-imgur');

 global_id      | author      | cluster | created_time         | description        | embeddings | entities | score
 | sentiments                                              | source | text | url
----------------+-------------+---------+----------------------+--------------------+------------+----------+------
-+-------------------------------------------------------+--------+------+--------------------------------
 1xR78Th-imgur | przyklad12 |      -1 | 2024-01-22T10:55:27 | Please catch this |            |       [] |      1
 | [{"label": "positive", "score": 0.4012986123561859}] |  imgur |      | https://i.imgur.com/1xR78Th.jpg
```

In terms of the presentation module – as already mentioned, we decided to use Tableau for our presentation, as it supports live data source refreshes and allows us to create informative dashboards in an easy and approachable manner. We used Tableau in version *2023.3*. We created two dashboards in total, each dashboard is being refreshed every 15 seconds.

Project results

Results presentation

The final results of the project are presented in the form of Tableau Dashboards. We have created two dashboards. Each dashboard is refreshed every 15 seconds with new data. The first dashboard is dedicated to the analysis based on different sources. It is as follows:



imgur	negative	14
	neutral	215
	positive	349
reddit	negative	1
	neutral	3
	positive	5
tumblr	negative	11
	neutral	47
	positive	8

source
🟦 imgur
🟥 reddit
🟧 tumblr

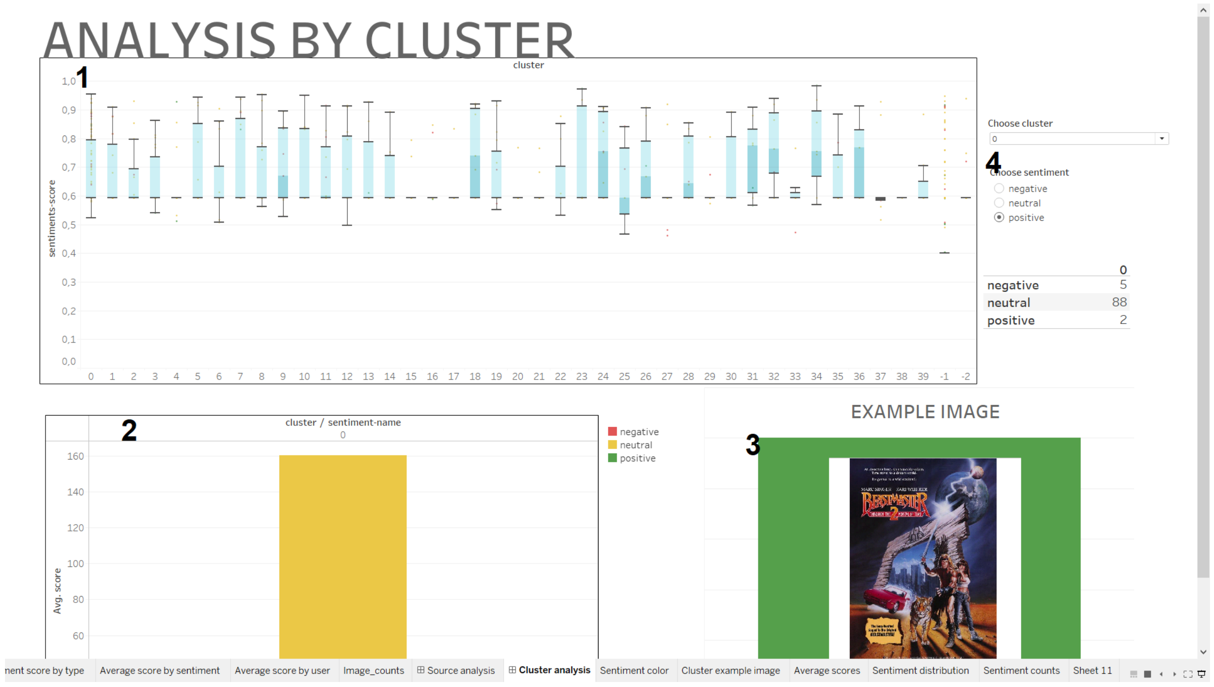
Refreshing **1** datasource(s) every **15** seconds.

Average score by user

source	author
imgur	buttercupcosplays
	TongueTiedLaboure..
	TheEndOfEvangelion
	TheArtistFormerlyK..
	mdkynyc
	Satci
	2112eric5772
	codehunter
	uglyshirts
	Mech0T1
	FloatingOnACloudOf..
	EmberTay
	nevets767
	BreakfastIsTheMost..
	Yermamma
	JimmyGTraveler
	ReposterCallerOute..
	sleeperkid
	SirNameless
	NiceRead
	JohnSchatzi
	ginalynn8942
	DumpingSh1tandBe..
	butwhyisthenameg..
	BottomShelfVodka
	BirdGuyJosh
	bexxMaker
	Arckhunter
	abbiistabbii
	zoraniko
	zengtangdondon
	zeldawallpapers
	velvetdream

There are 5 main components. Plots **1** and **2** are similar in structure – plot **1** represents the average score values per sentiment name and per source, while plot **2** – average sentiment values. It is worth noting that there are 3 sentiment names: *positive*, *neutral* and *negative*

(colored accordingly – green, yellow and red). Component **3** is the table representing the count of posts for each source and sentiment. It is the component that is updated the most frequently. It provides a much-needed context for plots **1** and **2**, as users can check which sources are the most numerous and if this is reflected on the plots.
Component number **4** is a straightforward ranking of scores per author. This way we are able to quickly retrieve the highest-scoring users for each source.
Finally, component **5** allows user to switch dashboard to a specific data source. By default all sources (Imgur, Reddit, Tumblr) are displayed, but it is possible to switch dashboard to just one source. It affects all the remaining components.
The second dashboard is dedicated to the clustering analysis. It is as follows:



Choose cluster: 0

Choose sentiment: negative neutral [✓ positive]

	0
negative	5
neutral	88
positive	2

In general, this dashboard is dedicated to analyzing sentiments across clusters. The main part is plot number **1**. It presents the distribution of sentiment scores among different clusters. Plot number **2** allows for a more in-depth analysis of sentiment scores among sentiment types inside a chosen cluster. In component **3** we are presented with an example image from the chosen cluster and sentiment type. Finally, component **4** allows us to choose a cluster for more in-depth analysis as well as the sentiment types. The choice affects components **2** and **3**.

Near-real-time changes

Link with movie documenting the changes on the clusters dashboard:
https://drive.google.com/file/d/1dM2Xe6wOEXaQCeaAzZ7WKJgOznnxy6Hf/view?usp=sharing
In this movie we can clearly see the number of clusters increasing, which is reflected on plot **1**.

Link with the movie documenting the changes on the sources dashboard:
https://drive.google.com/file/d/1CTcRusEmiCY6u2XMeM-wDw8_Rq8-1tyc/view?usp=sharing

In it, we can clearly see the numbers in the table **3** increase and the plot **1** for the neutral sentiment rises slightly (in the first 6 seconds we can see the changes, later on the capabilities of the dashboard are shown)

Results assessment

Despite numerous difficulties during the development of this project, overall we consider our results to be rather satisfactory. In assessment, we will focus on the presentation layer.
First of all, we consider plot number **1** from the first dashboard (tackling the analysis of sources) to yield probably the most promising results of all. As it turns out, for all the different sources (exept Tumblr, as became apparent with more data arriving), there is an interesting phenomenon present: posts classified as "postive" have, on average, the best scores of all. Next come the posts labeled as "neutral", and posts labeled as "negative" have the lowest scores. This is especially interesting, as it gives a possibility to predict the score of the post based on its sentiment (whether the post will be more or less popular). It also shows that people tend to like more posts that are associated with positivity.
Other plots also may be considered useful: as an example, **4** from the first dashboard can be used to quickly find the best-scoring users – which may be beneficial e.g. from a marketing perspective (contacting users for advice on how to create a new marketing campaign).
FInally, the whole second dashboard provides an interesting area of analysis. While the clusters themselves may have a non-obvious interpretation (as was apparent from analysing the example images from them), the clusters themselves are rather distinguishable in terms of sentiment inside them. It is highly possible that given enough time (and data), we would be able to distinguish a few "more interesting" clusters, and by further analysing images inside them, draw some potentially useful conclusions (e.g. a template or a topic which is associated with more positive sentiment, and thus, as plot **1** from first dashboard shows us, with possibly more popularity).

Key Performance Measures

- Cost: ~ 60 PLN / day, not including on-prem costs
- Time to process a new observation 2 minutes +/- 1minute
- Startup time of the Spark Analytics module: ~40 minutes
- Scores of the clustering module:
  - Silhouette score: -0.0971
  - Davies-Bouldin score: 5.5847
  - Calinski-Harabasz score: 1.0433

The societal impact

There are two types of potential users of our project: meme-creators, seeking inspiration or marketing agents. While the first group is not really concerning, as their impact may be negligible, the second one may be more “interesting”.
One obvious negative outcome may be that a company using our solution may create an ineffective marketing campaign and lose a significant amount of money. This, while undoubtedly negative, is not too much of a wide-reaching issue. However, there is another, perhaps more interesting possibility. Hypothetically, a company may be able to consistently create very effective marketing campaigns based on the current trends in meme culture. In fact, it may happen that these campaigns are so effective, that company-sponsored images start to blend in with others, creating a type of subliminal message campaign. This, naturally, may be further affected by the company’s own ethical policies. As a result, users may become more and more distrustful of their favorite entertainment-providing platforms, which may significantly affect their well-being. What is more, such an effective campaign would also create a significant economic effect on a possibly global scale.
Finally, there is a non-negligible privacy concern. While it may not be as impactful in this project, as it was only analysis-based, if it were to be used for commercial purposes, it would need to be granted approval not only by the administrators of platforms from which the data is collected but also, possibly, from the users posting the images. The latter part may depend on the individual platform’s rules and regulations, though.

Tests

The test document is located here.

Work division

A detailed description of team members' contributions throughout the whole project (software and reports).

Jakub Fołtyn:
- Imgur data acquisition
- NiFi and Cassandra set-up
- Nifi workflows implementation and testing
- Storage set-up for Master Log
- Presentation layer and Tableau dashboards
- EDA
- Ingestion layer, presentation layer, EDA sections

Kacper Grzymkowski:
- Reddit data acquisition
- Infrastructure management
- Kafka, Spark set-up
- ViT and Clustering module implementation and testing
- Resolving issues with everything falling apart

- Model improvement via Silhouette score
- Project architecture section

Illia Tesliuk:

- Tumblr data acquisition
- Pubsub/Pubsub lite/Kafka feasibility assessment
- OCR, NER, Sentiment Analysis, Stream Combining module implementation and testing

Mikołaj Malec

- Not applicable