Wieloskalowa analiza danych z forum internetowego przy użyciu usług chmury AWS

Michał Kamiński

21.01.2023

Spis treści

Streszczenie

Słowa kluczowe: Big Data, Spark, AWS, EMR, S3

Summary

Keywords: Big Data, Spark, AWS, EMR, S3

Cześć I

Wstęp

Technologie big data

Formaty danych

Chmury

Cel pracy

Celem niniejszej pracy jest utworzenie infrastruktury w chmurze obliczeniowej AWS pozwalające na wielkoskalową analizy danych w sytemie rozproszonym (ang. *Big Data*).

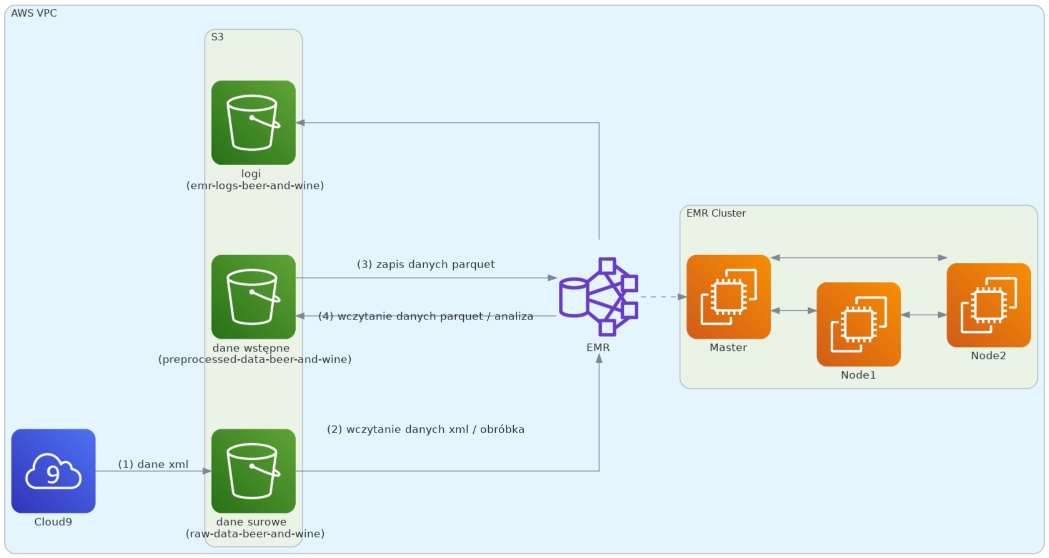
Do stworzenia przykładowego projektu wykorzystano dane ze strony *Stack Exchange* zawierającej zestawy danych pochodzące z forów społecznościoowych. Analizę ograniczono do danych pochodzących z forum o nazwie *Beer, Wine and Spirits*.

W niniejszej pracy ...

Cześć II

Wyniki i Dyskusja

1 Schemat infrastruktury



Rysunek 1.1: Schemat rozwiązania

W celu rozwiązania postawionego problemu analitycznego stworzono infrastrukturę wyłącznie w obrębie chmury AWS, której ogólny schemat przedstawiono na Rysunek 1.1

1.1 Ekstrakcja

Do etapu ekstrakcji danych wykorzystano usługę `Cloud9`, która zapewnia dostęp do terminala maszyny wirtualnej z systemem linux (platforma `Amazon Linux 2`, typ instancji `t2.micro`). Z użyciem tej usługi dane zostały pobrane ze źródła w binarnym formacie `7z` a następnie pliki zostały wyekstrahowane w formacie `xml` przy pomocy programu `p7zip`. Dane w formacie `xml` zostały następnie skopiowane do serwisu `S3`, gdzie utworzono koszyk danych (ang. *bucket*) o nazwie `raw-data-beer-and-wine`, którego przeznaczeniem jest przetwymywanie danych nieprzetworzonych.

Powyższe operacje zostały wykonane przy użyciu poniższych poleceń:

```
# instalacja programu p7zip
sudo yum install p7zip.x86_64

# pobranie danych
wget https://archive.org/download/stackexchange/beer.stackexchange.com.7z

# ekstrakcja danych do folderu raw-data
7za e  beer.stackexchange.com.7z -oraw-data

# zapis danych do koszyka S3 przy użyciu programu `AWS CLI`
aws s3 cp $(pwd)/raw-data s3://raw-data-beer-and-wine/ --recursive  --include "*.xml"
```

1.2 Przygotowanie danych wstępnych

W celu przygotowania danych do analizy, dane surowe zostały wstępnie przetworzone oraz zapisane w formacie `parquet`, co pozwoli na wydajniejsze wczytywanie danych podczas uruchomień programu. Podczas etapu wstępnego przetwarzania danych, oprócz zmiany formatu plików, zdefiniowane zostały także schematy danych, które zapewnią, że kolumny danych będą posiadały odpowiednie typy oraz, że krytyczne dane nie będą zawierały pustych wartości. Dodatkowo kolumny z wartościami tekstowymi, niesłownikowanymi zostały oczyszczone z tagów `html` oraz poddane standardowej procedurze oczyszania tekstu.

Powyższe czynności zostały wykonane w notatniku typu Jupiter (ang. *`Jupyter Notebook`*) w serwisie `AWS EMR`. Stworzono klaster `EMR` (wersja 6.8.0) z instalacją `Hadoop` 3.2.1, `Jupyter Hub` oraz `Spark` 3.3.0, składający się z 1 instancji typu *master* oraz 2 instancji typu *core*, każda typu `m4.large`. W celu ograniczenia kosztów jako opcję zakupu wybrano typ `spot` z limitem maksymalnym ceny odpowiadającej typowi `on-demand`. Wielkość dysków `EBS` stworzonych instancji wynosiła 32 GiB dla każdej istancji w klastrze.

Polecenie programu AWS CLI odpowiadające za utworzenie klastra zajduje się w sekcji Sekcja 4.1.1.

Dostęp do `Jupyter Notebook` w utworzonym klastrze jest możliwy poprzez połącznie przez przeglądarkę z środowiskiem graficznym `Jupyter Hub` wykorzystując adres DNS instancji *master* i port 9443.

1.3 Budowa infrastruktury

Wszystkie serwisy AWS na potrzeby tego projektu zostały utworzone w sposób programatyczny przy użyciu programu `AWS CLI` (poza `Cloud9`, który został utworzony z poziomu konsoli zarządzającej). Wykorzystane polecenia dostępne są w sekcji Rozdział 4.

2 Wstępna obróbka danych

2.1 Konfiguracja aplikacji

W celu przygotowania danych do analizy zostały one wstępnie przetworzone. Pierwszym etapem wstępnego przetwarzania jest wczytanie danych do środowiska analitycznego. Dane surowe, przechowywane w koszyku `raw-data-beer-and-wine` znajowały się w mało przyjaznym dla analiz formacie `xml`. Wczytanie tego typu danych wymagało załadowania dodatkowego pakietu `jar` o nazwie `spark-xml_2.12:0.14.0` pobranego z repozytorium `maven`.

W serwisie `EMR` można dodać tego typu pakiety wykorzystując specjalne polecenia typy `Sparkmagic` rozpoczynające się od znaków `%%`. W tym przypadku użyto `%%configure`:

```
%%configure -f
{
    "conf": {
        "spark.jars.packages": "com.databricks:spark-xml_2.12:0.14.0"
    }
}
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Preprocessing").getOrCreate()
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
2	application_1674145937048_0003	pyspark	idle	Link	Link	None	

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
SparkSession available as 'spark'.
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

2.2 Schematy danych

2.2.1 Users

```
from pyspark.sql.types import *

users_schema = StructType([
    StructField('_AboutMe', StringType(), True),
    StructField('_AccountId', IntegerType(), True),
    StructField('_CreationDate', TimestampType(), True),
    StructField("_DisplayName", StringType(), True),
    StructField("_DownVotes", IntegerType(), True),
    StructField("_Id", IntegerType(), True),
    StructField("_LastAccessDate", TimestampType()),
    StructField("_Location", StringType(), True),
    StructField("_ProfileImageUrl", StringType(), True),
    StructField("_Reputation", IntegerType(), True),
    StructField("_UpVotes", IntegerType(), True),
    StructField("_Views", IntegerType(), True),
    StructField("_WebsiteUrl", StringType(), True)
])

users = spark.read.format('xml').options(rowTag='row').schema(users_schema).load("s3://raw-d
users.show(2, vertical=True, truncate=50)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
-RECORD 0-----------------------------------------------------------------
 _AboutMe          | <p>Hi, I'm not really a person.</p>\n\n<p>I'm a...
 _AccountId        | -1
 _CreationDate     | 2014-01-21 17:45:53.587
 _DisplayName      | Community
 _DownVotes        | 503
 _Id               | -1
 _LastAccessDate   | 2014-01-21 17:45:53.587
 _Location         | on the server farm
 _ProfileImageUrl  | null
 _Reputation       | 1
 _UpVotes          | 2
 _Views            | 5
 _WebsiteUrl       | http://meta.stackexchange.com/
-RECORD 1-----------------------------------------------------------------
 _AboutMe          | <p>Dev #2 who helped create Stack Overflow curr...
 _AccountId        | 2
 _CreationDate     | 2014-01-21 20:21:18.797
 _DisplayName      | Geoff Dalgas
 _DownVotes        | 0
 _Id               | 1
 _LastAccessDate  | 2016-05-06 20:34:57.983
 _Location        | Corvallis, OR
 _ProfileImageUrl | https://i.stack.imgur.com/nDllk.png?s=128&g=1
 _Reputation      | 101
 _UpVotes         | 0
 _Views           | 42
 _WebsiteUrl      | http://stackoverflow.com
only showing top 2 rows
```

2.2.2 Tags

```
tags_schema = StructType([
    StructField('_Count', IntegerType(), True),
    StructField('_ExcerptPostId', IntegerType(), True),
    StructField('_Id', IntegerType(), True),
    StructField("_TagName", StringType(), True),
    StructField("_WikiPostId", IntegerType(), True)
])

tags = spark.read.format('xml').options(rowTag='row').schema(tags_schema).load("s3://raw-dat
tags.show(n=5)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'

+------+--------------+---+-----------+-----------+
|_Count|_ExcerptPostId|_Id|   _TagName|_WikiPostId|
+------+--------------+---+-----------+-----------+
|    17|          5062|  1|       hops|       5061|
|    85|          7872|  2|    history|       7871|
|    69|          4880|  4|    brewing|       4879|
|    37|          5109|  5|    serving|       5108|
|    31|           304|  6|temperature|        303|
+------+--------------+---+-----------+-----------+
only showing top 5 rows
```

2.2.3 Votes

```
votes_schema = StructType([
    StructField('_BountyAmount', IntegerType(), True),
    StructField('_CreationDate', TimestampType(), True),
    StructField('_Id', IntegerType(), True),
    StructField("_PostId", StringType(), True),
    StructField("_UserId", IntegerType(), True),
    StructField("_VoteTypeId", IntegerType(), True)
])

votes = spark.read.format('xml').options(rowTag='row').schema(votes_schema).load("s3://raw-d
votes.show(n=5)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
+-------------+-------------------+---+-------+-------+-----------+
|_BountyAmount|      _CreationDate|_Id|_PostId|_UserId|_VoteTypeId|
+-------------+-------------------+---+-------+-------+-----------+
|         null|2014-01-21 00:00:00|  1|      1|   null|          2|
|         null|2014-01-21 00:00:00|  2|      1|   null|          2|
|         null|2014-01-21 00:00:00|  3|      4|   null|          2|
|         null|2014-01-21 00:00:00|  4|      1|   null|          2|
|         null|2014-01-21 00:00:00|  5|      4|   null|          2|
+-------------+-------------------+---+-------+-------+-----------+
only showing top 5 rows
```

2.2.4 Posts

```
posts_schema = StructType([
    StructField('_AcceptedAnswerId', IntegerType(), True),
    StructField('_AnswerCount', IntegerType(), True),
    StructField('_Body', StringType(), True),
    StructField("_ClosedDate", TimestampType(), True),
    StructField("_CommentCount", IntegerType(), True),
    StructField("_CommunityOwnedDate", TimestampType(), True),
    StructField("_ContentLicense", StringType(), True),
    StructField("_CreationDate", TimestampType(), True),
    StructField("_FavoriteCount", IntegerType(), True),
    StructField("_Id", IntegerType(), True),
    StructField("_LastActivityDate", TimestampType(), True),
    StructField("_LastEditDate", TimestampType(), True),
    StructField("_LastEditorDisplayName", StringType(), True),
    StructField("_LastEditorUserId", IntegerType(), True),
    StructField("_OwnerDisplayName", StringType(), True),
    StructField("_OwnerUserId", IntegerType(), True),
    StructField("_ParentId", IntegerType(), True),
    StructField("_PostTypeId", IntegerType(), True),
    StructField("_Score", IntegerType(), True),
    StructField("_Tags", StringType(), True),
    StructField("_Title", StringType(), True),
    StructField("_ViewCount", IntegerType(), True),
])

posts = spark.read.format('xml').options(rowTag='row').schema(posts_schema).load("s3://raw-d
posts.show(n=1,vertical=True, truncate=50)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
-RECORD 0------------------------------------------------------------------
 _AcceptedAnswerId      | 4
 _AnswerCount           | 1
 _Body                  | <p>I was offered a beer the other day that was ...
 _ClosedDate            | null
 _CommentCount          | 0
 _CommunityOwnedDate    | null
 _ContentLicense        | CC BY-SA 3.0
 _CreationDate          | 2014-01-21 20:26:05.383
 _FavoriteCount         | null
 _Id                    | 1
 _LastActivityDate      | 2014-01-21 22:04:34.977
 _LastEditDate          | 2014-01-21 22:04:34.977
 _LastEditorDisplayName | null
 _LastEditorUserId      | 8
 _OwnerDisplayName      | null
 _OwnerUserId           | 7
 _ParentId              | null
 _PostTypeId            | 1
 _Score                 | 21
 _Tags                  | <hops>
 _Title                 | What is a citra hop, and how does it differ fro...
 _ViewCount             | 2441
only showing top 1 row
```

2.2.5 Post links

```
links_schema = StructType([
    StructField("_CreationDate", TimestampType()),
    StructField("_Id", IntegerType()),
    StructField("_LinkTypeId", IntegerType()),
    StructField("_PostId", IntegerType()),
    StructField("_RelatedPostId", IntegerType())
])

links = spark.read.format('xml').options(rowTag='row').schema(links_schema).load("s3://raw-d
links.show(n=5, truncate=False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
+-----------------------+---+-----------+-------+--------------+
|_CreationDate          |_Id|_LinkTypeId|_PostId|_RelatedPostId|
+-----------------------+---+-----------+-------+--------------+
|2014-01-21 21:04:25.23 |25 |3          |29     |25            |
|2014-01-21 21:42:09.103|89 |1          |83     |50            |
|2014-01-21 21:50:41.313|95 |1          |86     |2             |
|2014-01-21 22:07:35.783|101|3          |47     |99            |
|2014-01-21 22:13:51.38 |102|1          |74     |3             |
+-----------------------+---+-----------+-------+--------------+
only showing top 5 rows
```

2.2.6 Post History

```
history_schema = StructType([
    StructField("_Comment", StringType()),
    StructField("_ContentLicense", StringType()),
    StructField("_CreationDate", TimestampType()),
    StructField("_Id", IntegerType()),
    StructField("_PostHistoryTypeId", IntegerType()),
    StructField("_PostId", IntegerType()),
    StructField("_RevisionGUID", StringType()),
    StructField("_Text", StringType()),
    StructField("_UserDisplayName", StringType()),
    StructField("_UserId", IntegerType()),
])

history = spark.read.format('xml').options(rowTag='row').schema(history_schema).load("s3://r
history.show(n=2,vertical=True, truncate=50)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'

-RECORD 0----------------------------------------------------------------
 _Comment           | null
 _ContentLicense    | CC BY-SA 3.0
 _CreationDate      | 2014-01-21 20:26:05.383
 _Id                | 1
 _PostHistoryTypeId | 2
 _PostId            | 1
 _RevisionGUID      | a17002a0-00b0-417b-a404-0d8864bbbca5
 _Text              | I was offered a beer the other day that was rep...
 _UserDisplayName   | null
 _UserId            | 7
-RECORD 1----------------------------------------------------------------
 _Comment           | null
 _ContentLicense    | CC BY-SA 3.0
 _CreationDate      | 2014-01-21 20:26:05.383
 _Id                | 2
 _PostHistoryTypeId | 1
 _PostId            | 1
 _RevisionGUID      | a17002a0-00b0-417b-a404-0d8864bbbca5
 _Text              | What is a citra hop, and how does it differ fro...
 _UserDisplayName   | null
 _UserId            | 7
only showing top 2 rows
```

2.2.7 Badges

```
badges_schema = StructType([
    StructField("_Class", IntegerType()),
    StructField("_Date", TimestampType()),
    StructField("_Id", IntegerType()),
    StructField("_Name", StringType()),
    StructField("_TagBased", BooleanType()),
    StructField("_UserId", IntegerType()),
])

badges = spark.read.format('xml').options(rowTag='row').schema(badges_schema).load("s3://raw
badges.show(n=5,truncate=False)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
+------+----------------------+---+--------------+---------+-------+
|_Class|_Date                 |_Id|_Name         |_TagBased|_UserId|
+------+----------------------+---+--------------+---------+-------+
|3     |2014-01-21 20:52:16.97|1  |Autobiographer|false    |1      |
|3     |2014-01-21 20:52:16.97|2  |Autobiographer|false    |2      |
|3     |2014-01-21 20:52:16.97|3  |Autobiographer|false    |6      |
|3     |2014-01-21 20:52:16.97|4  |Autobiographer|false    |7      |
|3     |2014-01-21 20:52:16.97|5  |Autobiographer|false    |9      |
+------+----------------------+---+--------------+---------+-------+
only showing top 5 rows
```

2.3 Czyszczenie kolumn tekstowych

```
from pyspark.sql.functions import regexp_replace, trim, udf, col

from bs4 import BeautifulSoup
from html import unescape

def tags_remove(s):
    if s is not None:
        soup = BeautifulSoup(unescape(s), 'lxml')
        return soup.text
    else:
        return None
udf_tags_remove = udf(lambda m: tags_remove(m))
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
users_clean = users.withColumn("_AboutMe_clean", regexp_replace("_AboutMe", "\n|\t|\r", " ")
    .withColumn("_AboutMe_clean", udf_tags_remove(col('_AboutMe_clean'))) \
    .withColumn("_AboutMe_clean", regexp_replace("_AboutMe_clean", "\s{2,}", " ")) \
    .withColumn("_AboutMe_clean", trim("_AboutMe_clean"))


history_clean = history.withColumn("_Text_clean", regexp_replace("_Text", "\n|\t|\r", " "))
    .withColumn("_Text_clean", udf_tags_remove(col('_Text_clean'))) \
    .withColumn("_Text_clean", regexp_replace("_Text_clean", "\s{2,}", " ")) \
    .withColumn("_Text_clean", trim("_Text_clean"))

posts_clean = posts.withColumn("_Body_clean", regexp_replace("_Body", "\n|\t|\r", " ")) \
    .withColumn("_Body_clean", udf_tags_remove(col('_Body_clean'))) \
    .withColumn("_Body_clean", regexp_replace("_Body_clean", "\s{2,}", " ")) \
    .withColumn("_Body_clean", trim("_Body_clean"))
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px
```

```
users_clean.show(1, vertical=True, truncate=50)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px
```

```
-RECORD 0-------------------------------------------------------------
 _AboutMe         | <p>Hi, I'm not really a person.</p>\n\n<p>I'm a...
 _AccountId       | -1
 _CreationDate    | 2014-01-21 17:45:53.587
 _DisplayName     | Community
 _DownVotes       | 503
 _Id              | -1
 _LastAccessDate  | 2014-01-21 17:45:53.587
 _Location        | on the server farm
 _ProfileImageUrl | null
 _Reputation      | 1
 _UpVotes         | 2
 _Views           | 5
 _WebsiteUrl      | http://meta.stackexchange.com/
 _AboutMe_clean   | Hi, I'm not really a person. I'm a background p...
only showing top 1 row
```

```
history_clean.show(1, vertical=True, truncate=50)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px
```

```
-RECORD 0---------------------------------------------------------------
 _Comment           | null
 _ContentLicense    | CC BY-SA 3.0
 _CreationDate      | 2014-01-21 20:26:05.383
 _Id                | 1
 _PostHistoryTypeId | 2
 _PostId            | 1
 _RevisionGUID      | a17002a0-00b0-417b-a404-0d8864bbbca5
 _Text              | I was offered a beer the other day that was rep...
 _UserDisplayName   | null
 _UserId            | 7
 _Text_clean        | I was offered a beer the other day that was rep...
only showing top 1 row
```

```
posts_clean.show(1, vertical=True, truncate=50)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'

-RECORD 0-------------------------------------------------------------------
 _AcceptedAnswerId      | 4
 _AnswerCount           | 1
 _Body                  | <p>I was offered a beer the other day that was ...
 _ClosedDate            | null
 _CommentCount          | 0
 _CommunityOwnedDate    | null
 _ContentLicense        | CC BY-SA 3.0
 _CreationDate          | 2014-01-21 20:26:05.383
 _FavoriteCount         | null
 _Id                    | 1
 _LastActivityDate      | 2014-01-21 22:04:34.977
 _LastEditDate          | 2014-01-21 22:04:34.977
 _LastEditorDisplayName | null
 _LastEditorUserId      | 8
 _OwnerDisplayName      | null
 _OwnerUserId           | 7
 _ParentId              | null
 _PostTypeId            | 1
 _Score                 | 21
 _Tags                  | <hops>
 _Title                 | What is a citra hop, and how does it differ fro...
 _ViewCount             | 2441
 _Body_clean            | I was offered a beer the other day that was rep...
only showing top 1 row
```

2.4 Zapis danych jako plik w formacie `parquet`

```
users_clean.select(
    col("_AboutMe").alias("about_me"),
    col("_AboutMe_clean").alias("about_me_clean"),
    col("_CreationDate").alias("creation_date"),
    col("_DisplayName").alias("display_name"),
    col("_DownVotes").alias("down_votes"),
    col("_Id").alias("id"),
    col("_LastAccessDate").alias("last_access_date"),
    col("_Location").alias("location"),
    col("_ProfileImageUrl").alias("profile_image_url"),
    col("_Reputation").alias("reputatio"),
    col("_UpVotes").alias("up_votes"),
    col("_Views").alias("views"),
    col("_WebsiteUrl").alias("website_url")
).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
tags.select(
    col("_Count").alias("count"),
    col("_ExcerptPostId").alias("excerpt_post_id"),
    col("_Id").alias("id"),
    col("_TagName").alias("tag_name"),
    col("_WikiPostId").alias("wiki_post_id"),
).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
votes.select(
    col("_BountyAmount").alias("bounty_amount"),
    col("_CreationDate").alias("creation_date"),
    col("_Id").alias("id"),
    col("_PostId").alias("post_id"),
    col("_UserId").alias("user_id"),
    col("_VoteTypeId").alias("vote_type_id"),
).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
posts_clean.select(
    col("_AcceptedAnswerId").alias("accepted_answer_id"),
    col("_AnswerCount").alias("answer_count"),
    col("_Body").alias("body"),
    col("_Body_clean").alias("body_clean"),
    col("_ClosedDate").alias("closed_date"),
    col("_CommentCount").alias("comment_count"),
    col("_CommunityOwnedDate").alias("community_owned_date"),
    col("_ContentLicense").alias("content_licence"),
    col("_CreationDate").alias("creation_date"),
    col("_FavoriteCount").alias("favourite_count"),
    col("_Id").alias("id"),
    col("_LastActivityDate").alias("last_activity_date"),
    col("_LastEditDate").alias("last_edit_date"),
    col("_LastEditorDisplayName").alias("last_editor_display_name"),
    col("_LastEditorUserId").alias("last_editor_user_id"),
    col("_OwnerUserId").alias("owner_user_id"),
    col("_PostTypeId").alias("post_type_id"),
    col("_ParentId").alias("parent_id"),
    col("_Score").alias("score"),
    col("_Tags").alias("tags"),
    col("_Title").alias("title"),
    col("_ViewCount").alias("view_count"),
).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
    links.select(
        col("_CreationDate").alias("creation_date"),
        col("_Id").alias("id"),
        col("_LinkTypeId").alias("link_type_id"),
        col("_PostId").alias("post_id"),
        col("_RelatedPostId").alias("related_post_id"),
    ).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
    history_clean.select(
        col("_Comment").alias("comment"),
        col("_ContentLicense").alias("content_license"),
        col("_CreationDate").alias("creation_date"),
        col("_Id").alias("id"),
        col("_PostHistoryTypeId").alias("post_history_type_id"),
        col("_PostId").alias("post_id"),
        col("_RevisionGUID").alias("revision_guid"),
        col("_Text").alias("text"),
        col("_Text_clean").alias("text_clean"),
        col("_UserDisplayName").alias("user_distplay_name"),
        col("_UserId").alias("user_id"),
).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

```
badges.select(
    col("_Class").alias("class"),
    col("_Date").alias("date"),
    col("_Id").alias("id"),
    col("_Name").alias("name"),
    col("_TagBased").alias("tag_based"),
    col("_UserId").alias("user_id"),
).write.mode('overwrite').format('parquet').option('path', "s3://preprocessed-data-beer-and-
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px'
```

3 Eksploracja i analiza danych

3.1 Questions/Answers over time

```
from pyspark.sql import (
    SparkSession,
    functions as f
    )
import matplotlib
```

```
spark = SparkSession.builder.master("local[12]").appName("Analytics").getOrCreate()
```

```
posts = spark.read.format('parquet').load("outputs/posts")
posts.show(1, vertical=True)
```

```
-RECORD 0----------------------------------------
 accepted_answer_id       | 4
 answer_count             | 1
 body                     | <p>I was offered ...
 body_clean               | I was offered a b...
 closed_date              | null
 comment_count            | 0
 community_owned_date     | null
 content_licence          | CC BY-SA 3.0
 creation_date            | 2014-01-21 20:26:...
 favourite_count          | null
 id                       | 1
 last_activity_date       | 2014-01-21 22:04:...
 last_edit_date           | 2014-01-21 22:04:...
 last_editor_display_name | null
 last_editor_user_id      | 8
 owner_user_id            | 7
 post_type_id             | 1
 parent_id                | null
 score                    | 21
 tags                     | <hops>
 title                    | What is a citra h...
 view_count               | 2434
only showing top 1 row
```

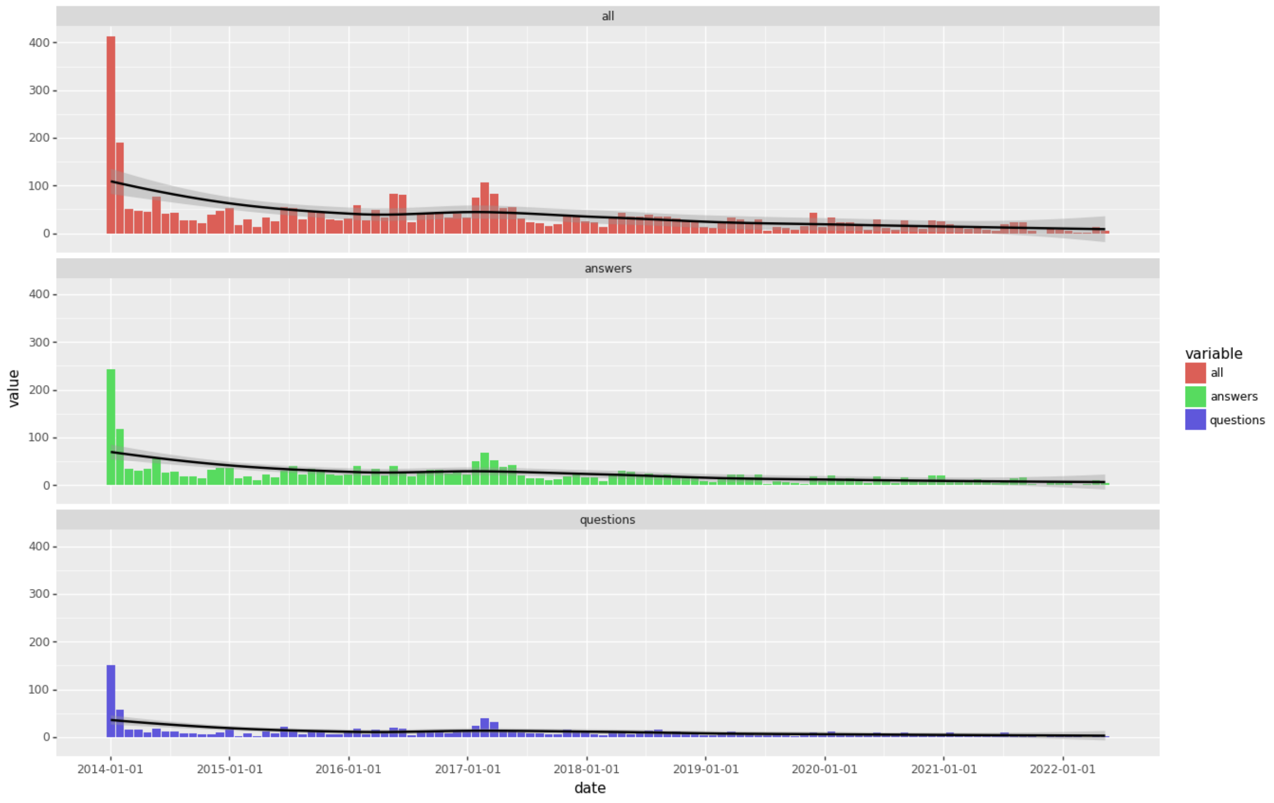
```
posts_grouped = (
    posts
    .filter(f.col('owner_user_id').isNotNull())
    .groupBy(
        f.window('creation_date', '4 weeks')
    )
    .agg(
        f.sum(f.lit(1)).alias('all'),
        f.sum(f.when(f.col('post_type_id') == 1, f.lit(1)).otherwise(f.lit(0))).alias('questions'),
        f.sum(f.when(f.col('post_type_id') == 2, f.lit(1)).otherwise(f.lit(0))).alias('answers')
    )
    # window struct has nested columns 'start' and 'end'
    .withColumn('date', f.col('window.start').cast('date'))
    .orderBy('date')
).toPandas()
```

```
posts_grouped.head()
```

	window	all	questions	answers	date
0	(2014-01-02 00:00:00, 2014-01-30 00:00:00)	413	150	243	2014-01-02
1	(2014-01-30 00:00:00, 2014-02-27 00:00:00)	190	58	118	2014-01-30
2	(2014-02-27 00:00:00, 2014-03-27 00:00:00)	50	16	34	2014-02-27
3	(2014-03-27 00:00:00, 2014-04-24 00:00:00)	47	16	31	2014-03-27
4	(2014-04-24 00:00:00, 2014-05-22 00:00:00)	44	10	34	2014-04-24

```
# posts_grouped.plot(
#     x='date',
#     figsize=(12, 6),
#     title='Number of questions/answers per month (4 weeks)',
#     legend=True,
#     xlabel='Date',
#     ylabel='Count',
#     kind='line'
# )
```

```
from plotnine import aes, facet_wrap, ggplot, scale_x_datetime, options, stat_smooth, geom_col

options.figure_size = (15, 10)

posts_long = posts_grouped.melt(id_vars=('date'), value_vars=('all', 'questions', 'answers'))
posts_long.head()

(ggplot(posts_long, aes(x='date', y='value', group='variable'))
+ geom_col(aes(fill='variable'))
+ scale_x_datetime()
+ stat_smooth(method='loess')
+ facet_wrap('variable', ncol=1)
)
```



```
<ggplot: (8785994108971)>
```

3.2 Forum retention - time from account creation to last comments

```
users = spark.read.format('parquet').load("outputs/users").select(f.col('id'), f.col('creation_date')
```

```
users.show()
```

```
+---+--------------------+----------------+
| id|       creation_date|    display_name|
+---+--------------------+----------------+
| -1|2014-01-21 17:45:...|       Community|
|  1|2014-01-21 20:21:...|    Geoff Dalgas|
|  2|2014-01-21 20:22:...|  Kasra Rahjerdi|
|  3|2014-01-21 20:22:...|       Adam Lear|
|  4|2014-01-21 20:22:...|   Arie Litovsky|
|  5|2014-01-21 20:22:...|    Brian Nickel|
|  6|2014-01-21 20:23:...|        Jeremy T|
|  7|2014-01-21 20:24:...|      Tom Medley|
|  8|2014-01-21 20:25:...|LessPop_MoreFizz|
|  9|2014-01-21 20:25:...|     Nick Craver|
| 10|2014-01-21 20:28:...|          ChrisG|
| 11|2014-01-21 20:28:...|        hairboat|
| 12|2014-01-21 20:29:...|        nhaarman|
| 13|2014-01-21 20:29:...|           Shog9|
| 14|2014-01-21 20:32:...|     Ben Collins|
| 15|2014-01-21 20:33:...|             Ana|
| 16|2014-01-21 20:34:...|      Grace Note|
| 17|2014-01-21 20:34:...|     Jon Ericson|
| 18|2014-01-21 20:36:...|         awesame|
| 19|2014-01-21 20:36:...|   Steve Robbins|
+---+--------------------+----------------+
only showing top 20 rows
```

```
posts_by_user = posts.select(f.col('owner_user_id'), f.col('last_activity_date'), f.col('id').alias('
```

```
posts_and_users_joined = (users
    .filter(f.col('id') != -1) # remove bots
    .join(posts_by_user, users.id  == posts_by_user.owner_user_id, how="left" )
    .filter(f.col('post_id').isNotNull()) # remove users that never posted
)

posts_and_users_joined.show()
```

```
+---+--------------------+--------------+-------------+--------------------+-------+
| id|       creation_date|  display_name|owner_user_id|  last_activity_date|post_id|
+---+--------------------+--------------+-------------+--------------------+-------+
|  2|2014-01-21 20:22:...|Kasra Rahjerdi|            2|2014-01-22 00:26:...|     39|
|  2|2014-01-21 20:22:...|Kasra Rahjerdi|            2|2014-01-22 05:50:...|     28|
|  4|2014-01-21 20:22:...| Arie Litovsky|            4|2016-03-06 04:31:...|     85|
|  5|2014-01-21 20:22:...|  Brian Nickel|            5|2014-02-04 23:20:...|    533|
|  5|2014-01-21 20:22:...|  Brian Nickel|            5|2014-01-23 16:41:...|    267|
|  5|2014-01-21 20:22:...|  Brian Nickel|            5|2014-01-22 18:36:...|    217|
|  5|2014-01-21 20:22:...|  Brian Nickel|            5|2018-08-09 15:38:...|     50|
|  5|2014-01-21 20:22:...|  Brian Nickel|            5|2014-01-21 20:59:...|     32|
|  5|2014-01-21 20:22:...|  Brian Nickel|            5|2014-01-21 20:45:...|     17|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2014-01-29 20:34:...|    426|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2014-11-19 15:11:...|     82|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2014-01-21 21:47:...|     70|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2020-08-28 07:35:...|     59|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2022-01-14 10:04:...|     38|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2014-01-22 06:24:...|     35|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2014-01-22 17:04:...|     10|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2017-08-24 06:53:...|      8|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2021-01-15 06:17:...|      7|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2017-06-07 11:10:...|      5|
|  7|2014-01-21 20:24:...|    Tom Medley|            7|2015-01-29 14:50:...|      3|
+---+--------------------+--------------+-------------+--------------------+-------+
only showing top 20 rows
```

```
posts_and_users_joined.select(f.col('post_id')).count() == posts_and_users_joined.select(f.col('post_
```

```
True
```

```
user_last_post = (posts_and_users_joined
    .groupBy(f.col('id'), f.col('creation_date'))
    .agg(
        f.max(f.col('last_activity_date'))
    )
)

# time from account creation to last activity
user_last_post = user_last_post.withColumn('diff',f.datediff(f.col('max(last_activity_date)'), f.col(
user_last_post.show()
```

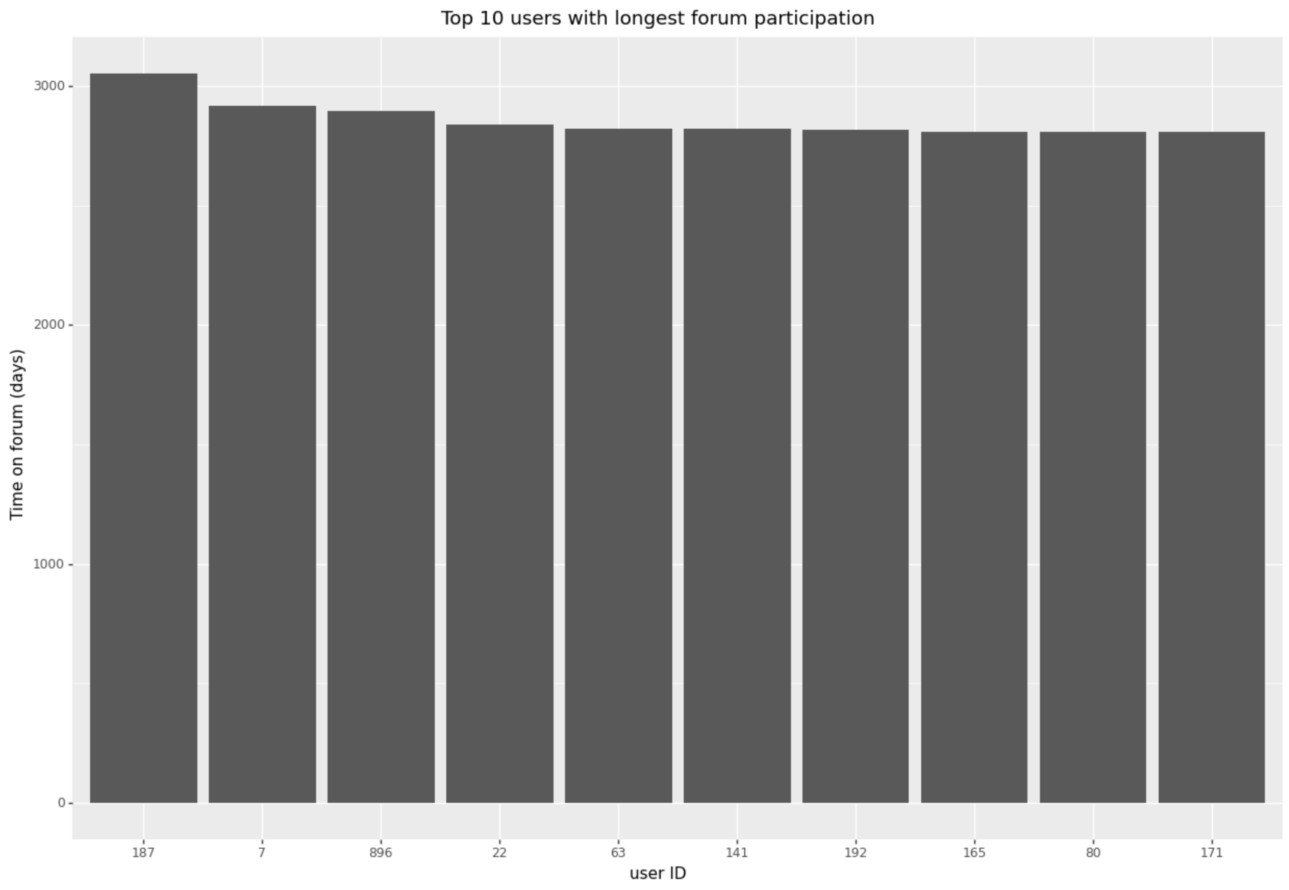
```
+-----+--------------------+-----------------------+----+
|   id|       creation_date|max(last_activity_date)|diff|
+-----+--------------------+-----------------------+----+
| 6696|2017-04-27 18:46:...|   2017-09-18 21:40:...| 144|
| 7212|2017-10-24 01:20:...|   2017-10-24 01:20:...|   0|
| 7311|2017-11-28 23:29:...|   2017-12-23 14:32:...|  25|
|10039|2020-01-17 20:47:...|   2020-03-25 21:19:...|  68|
|  149|2014-01-22 16:41:...|   2014-01-23 08:52:...|   1|
|  736|2014-04-03 13:25:...|   2017-01-27 14:13:...|1030|
| 4197|2015-06-10 19:13:...|   2015-06-10 19:18:...|   0|
| 5654|2016-07-13 09:17:...|   2016-09-02 12:08:...|  51|
| 7154|2017-10-01 06:24:...|   2017-10-01 06:24:...|   0|
| 7286|2017-11-23 11:43:...|   2017-11-24 22:07:...|   1|
| 7936|2018-07-20 09:31:...|   2018-07-24 11:49:...|   4|
|11698|2020-11-12 20:24:...|   2020-11-12 20:24:...|   0|
| 1295|2014-09-10 17:56:...|   2016-10-08 14:17:...| 759|
| 5893|2016-09-11 03:06:...|   2016-09-11 03:07:...|   0|
| 6636|2017-04-06 13:23:...|   2017-04-06 13:23:...|   0|
| 6699|2017-04-28 07:43:...|   2017-11-16 18:08:...| 202|
| 7208|2017-10-22 23:11:...|   2017-10-22 23:48:...|   0|
| 8088|2018-09-15 08:09:...|   2018-09-15 08:19:...|   0|
|  740|2014-04-04 15:58:...|   2016-06-21 13:46:...| 809|
| 1077|2014-07-09 23:08:...|   2018-10-09 20:22:...|1553|
+-----+--------------------+-----------------------+----+
only showing top 20 rows
```

```
user_last_post.select(f.col('id')).count() == user_last_post.select(f.col('id')).distinct().count()
```

```
True
```

```
user_last_post_df = user_last_post.orderBy(f.col('diff').desc()).limit(10).withColumn('id_cat', f.col

import pandas as pd
# add sorted categories for pretty plotting
user_last_post_df['id_cat'] = pd.Categorical(user_last_post_df.id_cat, categories=user_last_post_df.i

from plotnine import labs
(ggplot(user_last_post_df, aes(x='id_cat', y='diff'))
     + geom_col()
     + labs(x='user ID', y='Time on forum (days)', title='Top 10 users with longest forum participati
)
```

```
/config/workspace/.venv/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning
/config/workspace/.venv/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning
```



```
<ggplot: (8785942620109)>
```

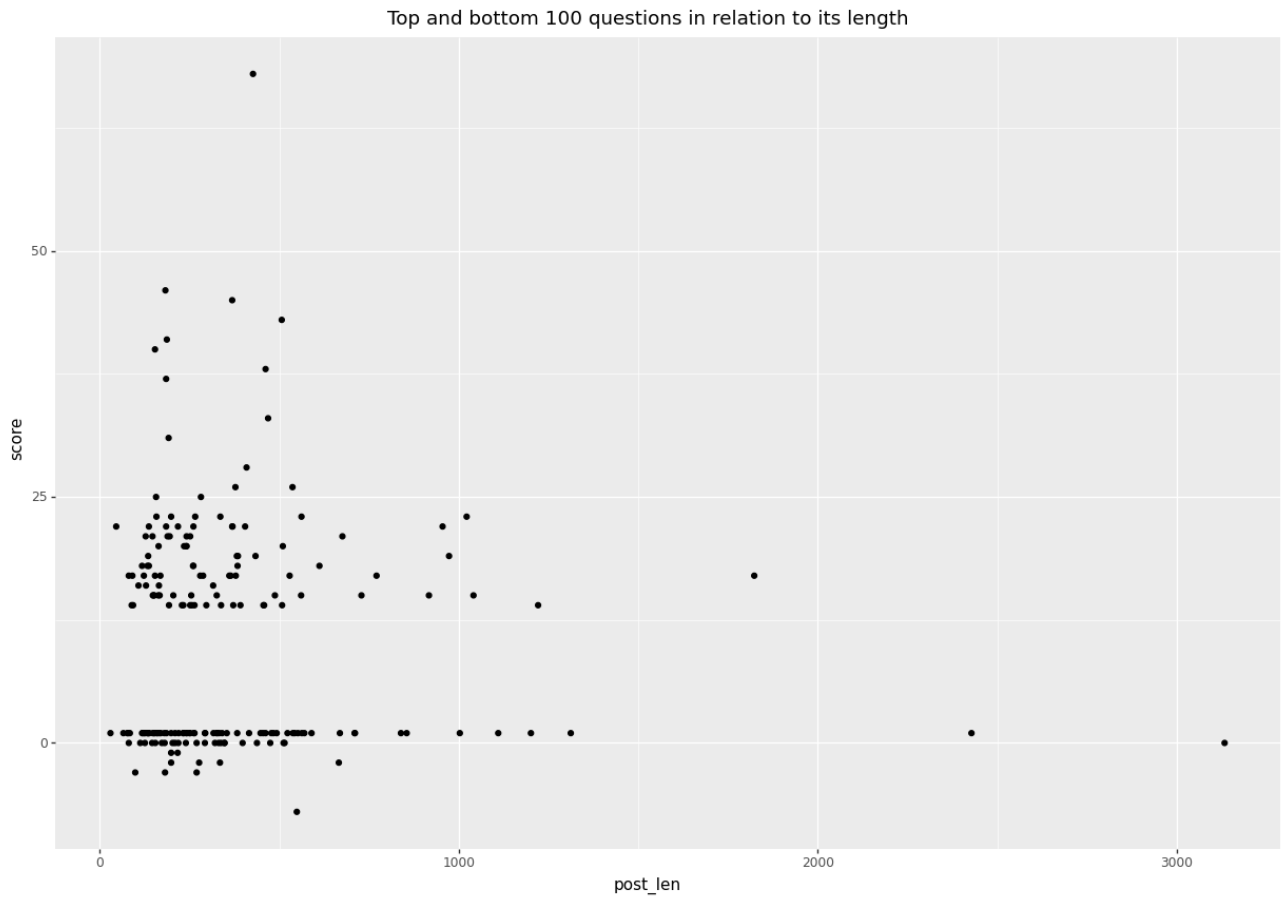
3.3 porównanie najwyżej i najniżej ocenianych pytań (długość, tagi, liczba odpowiedzi)

```
#users2 = spark.read.format('parquet').load("outputs/users").select(f.col('id'), f.col('creation_date
#posts.show(1, vertical=True)




#1 - Question 2 - Answer 3 - Wiki 4 - TagWikiExcerpt 5 - TagWiki 6 - ModeratorNomination 7 -  WikiPla
questions = posts.select(f.col('id'), f.col('body_clean'), f.col('answer_count'), f.col('view_count')
    .filter(f.col('post_type_id') == 1)\
    .drop(f.col('post_type_id'))
```

```
n_questions = 100
top_questions = questions.orderBy(f.col('score'), ascending=False).limit(n_questions).withColumn('typ
bottom_questions = questions.orderBy(f.col('score'), ascending=True).limit(n_questions).withColumn('t
edge_questions = top_questions.unionAll(bottom_questions)
```

```
from pyspark.sql.functions import length
edge_questions = edge_questions.withColumn('post_len', f.length(f.col('body_clean')))
edge_questions_pd = edge_questions.toPandas()
```

```
from plotnine import ggplot, aes, geom_point, ggtitle

(ggplot(edge_questions_pd, aes(x = 'post_len', y = 'score') ) \
    + geom_point() \
    + ggtitle(f'Top and bottom {n_questions} questions in relation to its length'))
```

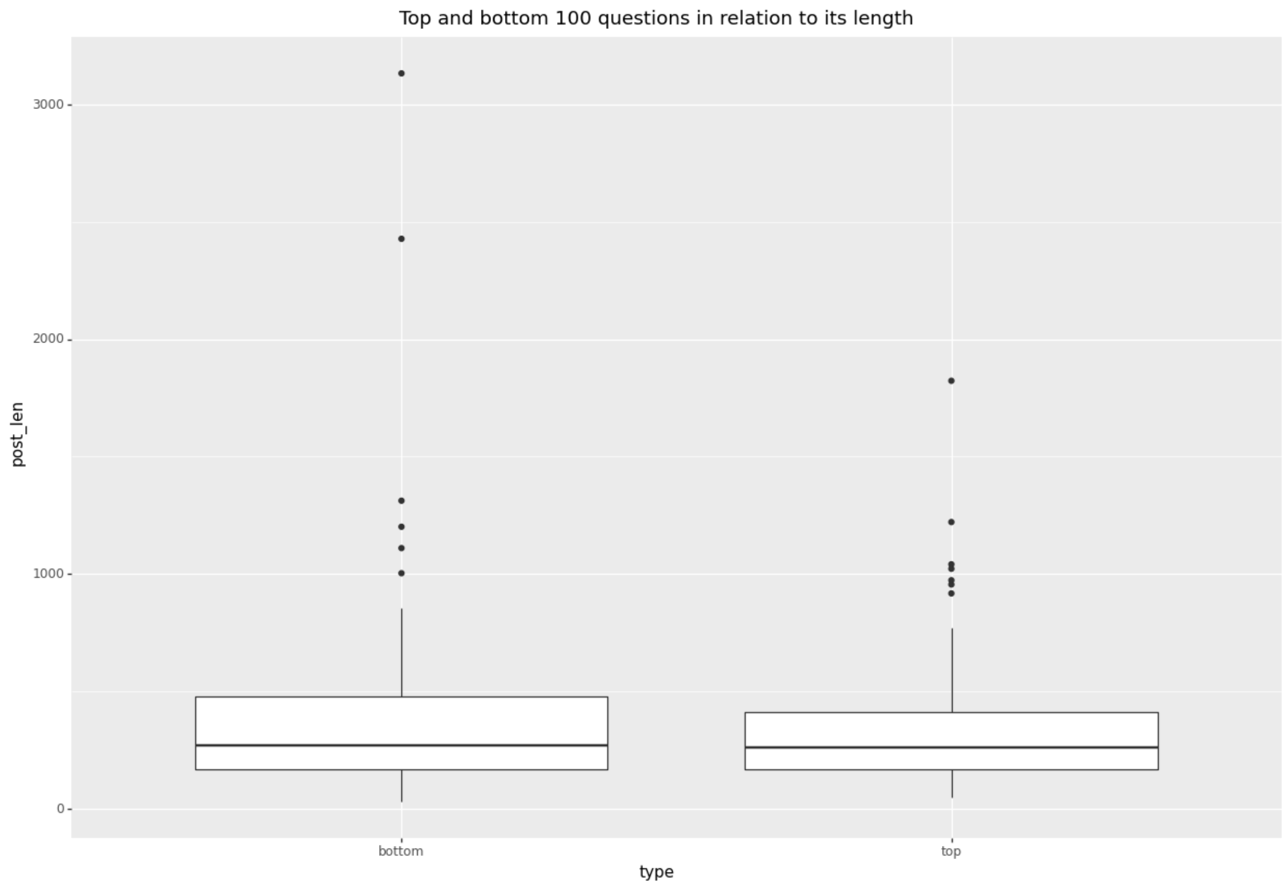


```
<ggplot: (8786045558433)>
```

```
edge_questions.groupby('type')\
    .agg(
        f.max(f.col('post_len')),
        f.min(f.col('post_len')),
        f.mean(f.col('post_len')),
        f.stddev(f.col('post_len')),
        f.percentile_approx(f.col('post_len'), 0.5)
    ).show()
```

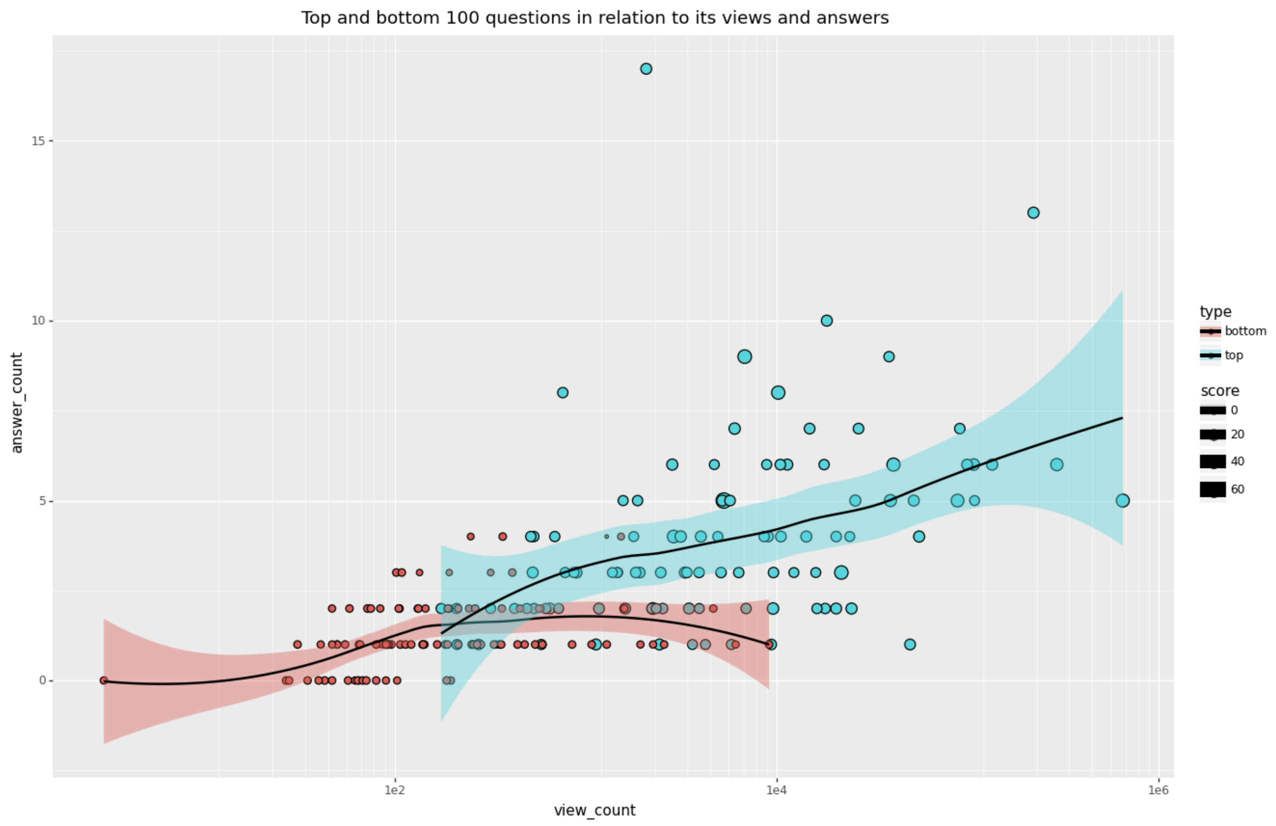
```
+------+-------------+-------------+-------------+--------------------+------------------------------
|  type|max(post_len)|min(post_len)|avg(post_len)|stddev_samp(post_len)|percentile_approx(post_len, 0.5
+------+-------------+-------------+-------------+--------------------+------------------------------
|   top|         1823|           46|       349.08|   275.08858738430234|
|bottom|         3133|           30|       389.26|   424.65483595146975|
+------+-------------+-------------+-------------+--------------------+------------------------------
```

```python
from plotnine import ggplot, aes, geom_boxplot, ggtitle

(ggplot(edge_questions_pd, aes(x = 'type', y = 'post_len') ) \
    + geom_boxplot() \
    + ggtitle(f'Top and bottom {n_questions} questions in relation to its length'))
```



```
<ggplot: (8785942555621)>
```

```python
from plotnine import ggplot, aes, geom_point, ggtitle, scale_x_log10, geom_smooth

(ggplot(edge_questions_pd, aes(x = 'view_count', y = 'answer_count', fill = 'type', size = 'score') )
    + geom_point() \
    + scale_x_log10()\
    + geom_smooth()\
    + ggtitle(f'Top and bottom {n_questions} questions in relation to its views and answers'))
```



```
<ggplot: (8785942667757)>
```

```
edge_questions.groupby('type')\
    .agg(
        f.max(f.col('answer_count')),
        f.min(f.col('answer_count')),
        f.mean(f.col('answer_count')),
        f.stddev(f.col('answer_count')),
        f.percentile_approx(f.col('answer_count'), 0.5)
    ).show()
```

```
+------+-----------------+-----------------+-----------------+-------------------------+----------------
|  type|max(answer_count)|min(answer_count)|avg(answer_count)|stddev_samp(answer_count)|percentile_appro
+------+-----------------+-----------------+-----------------+-------------------------+----------------
|   top|               17|                1|             3.96|        2.581675910150324|
|bottom|                4|                0|             1.28|       0.9648363026488436|
+------+-----------------+-----------------+-----------------+-------------------------+----------------
```

```
edge_questions.groupby('type')\
    .agg(
        f.max(f.col('view_count')),
        f.min(f.col('view_count')),
        f.mean(f.col('view_count')),
        f.stddev(f.col('view_count')),
        f.percentile_approx(f.col('view_count'), 0.5)
    ).show()
```

```
+------+---------------+---------------+---------------+----------------------+------------------------
|  type|max(view_count)|min(view_count)|avg(view_count)|stddev_samp(view_count)|percentile_approx(view_c
+------+---------------+---------------+---------------+----------------------+------------------------
|   top|         648941|            175|       26047.65|      76276.20019090576|
|bottom|           9124|              3|         495.56|     1220.7562924910794|
+------+---------------+---------------+---------------+----------------------+------------------------
```

```python
#https://gist.github.com/dannymeijer/be3534470b205280e52dbbcbb19a9670


from pyspark.sql import DataFrame
from pyspark.sql import functions as f


def regexp_extract_all(
    df: DataFrame,
    regex: str,
    no_of_extracts: int,
    input_column_name: str,
    output_column_name: str = "output",
    empty_array_replace: bool = True,
):
    """Pyspark implementation for extracting all matches of a reg_exp_extract

    Background
    ----------
    The regular implementation of regexp_extract (as part of pyspark.sql.functions module)
    is not capable of returning more than 1 match on a regexp string at a time. This
    function can be used to circumvent this limitation.

    How it works
    ------------
    You can specify a `no_of_extracts` which will essentially run the regexp_extract
    function that number of times on the `input_column` of the `df` (`DataFrame`).
    In between extracts, a set of interim columns are created where every
    intermediate match is stored. A distinct array is created from these matches,
    after which the interim columns are dropped. The resulting array is stored in
    the defined `output_column`. Empty strings/values in the resulting array can
    optionally be dropped or kept depending on how `empty_array_replace` is set
    (default is True).

    Usage example
    -------------
    In the below example, we are extracting all email-addresses from a body of text.
    The returned DataFrame will have a new ArrayType column added named `email_addresses`

    > # Assuming `df` is a valid DataFrame containing a column named `text`
    > email_regex = r"[\w.-]+@[\w.-]+\.[a-zA-Z]{1,}"
    > df = regexp_extract_all(df, email_regex, 6, "text", "email_addresses", True)

    Parameters
    ----------
    df: DataFrame
        Input DataFrame
```

```
regex: str
    Regexp string to extract from input DataFrame

no_of_extracts: int
    Max number of occurrences to extract

input_column_name: str
    Name of the input column

output_column_name: str
    Name of the output column (default: output)

empty_array_replace: bool
    If set to True, will replace empty arrays with null values (default: True)
"""
repeats = range(0, no_of_extracts)

# A set of interim columns are created that will be dropped afterwards
match_columns = [f"___{r}___" for r in repeats]

# Apply regexp_extract an r number of times
for r in repeats:
    df = df.withColumn(
        match_columns[r],
        f.regexp_extract(
            f.col(input_column_name),
            # the input regex string is amended with ".*?"
            # and repeated an r number of times
            # r needs to be +1 as matching groups are 1-indexed
            "".join([f"{regex}.*?" for i in range(0, r + 1)]),
            r + 1,
        ),
    )

# Create a distinct array with all empty strings removed
df = df.withColumn(
    output_column_name,
    f.array_remove(f.array_distinct(f.array(match_columns)), ""),
)

# Replace empty string with None if empty_array_replace was set
if empty_array_replace:
    df = df.withColumn(
        output_column_name,
        f.when(f.size(output_column_name) == 0, f.lit(None)).otherwise(
            f.col(output_column_name)
        ),
    )

# Drop interim columns
for c in match_columns:
    df = df.drop(c)

return df
```

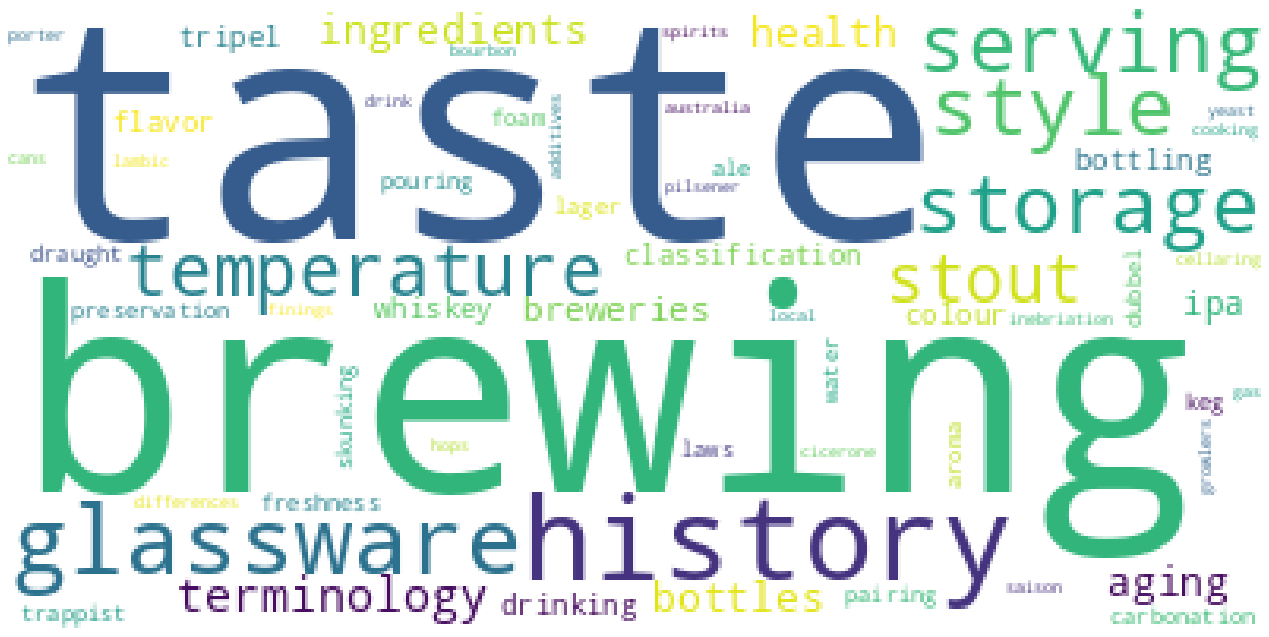
```
#edge_questions.select(f.col('tags')).withColumn('tags_split', f.regexp_extract(f.col('tags'), r'<(\w

edge_questions = regexp_extract_all(edge_questions, r'<(\w+)>', 99, "tags", "tags_split", True)
```

```
import pyspark.rdd as rdd
h = edge_questions.filter(f.col('type') == 'top').select(f.col('tags_split')).rdd
l = edge_questions.filter(f.col('type') == 'bottom').select(f.col('tags_split')).rdd
```

```
h.flatMap(lambda x: [y if y is not None else "" for y in x])\
    .flatMap(lambda x: [x[y] for y in range(0, len(x))])\
    .map(lambda x: (x, 1))\
    .aggregateByKey(0, (lambda acc,x: acc + x ), (lambda acc1,acc2: acc1+acc2))\
    .filter(lambda x: x[1] > 1)\
    .sortBy(lambda x: x[1], ascending=False)\
    .collect()
```

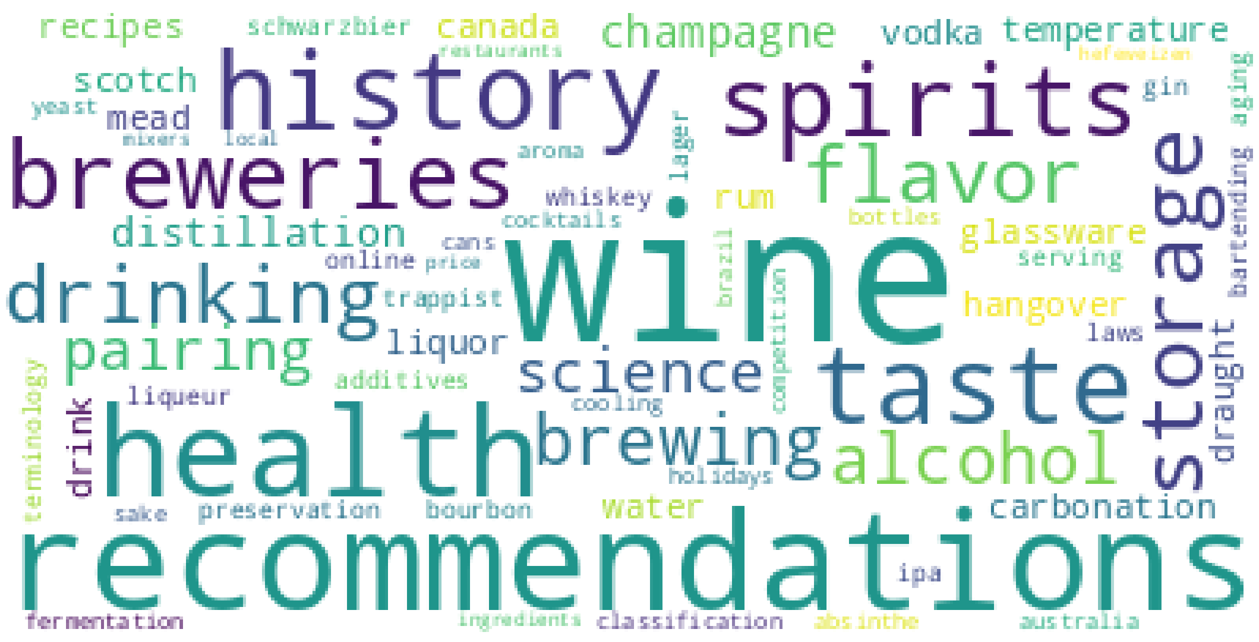
```
[('taste', 17),
 ('brewing', 14),
 ('history', 12),
 ('glassware', 8),
 ('storage', 8),
 ('serving', 8),
 ('style', 8),
 ('temperature', 6),
 ('stout', 5),
 ('terminology', 5),
 ('aging', 4),
 ('health', 4),
 ('bottles', 4),
 ('ingredients', 4),
 ('breweries', 3),
 ('ipa', 3),
 ('classification', 3),
 ('whiskey', 3),
 ('tripel', 3),
 ('drinking', 3),
 ('bottling', 3),
 ('flavor', 3),
 ('colour', 3),
 ('aroma', 2),
 ('freshness', 2),
 ('ale', 2),
 ('lager', 2),
 ('preservation', 2),
 ('foam', 2),
 ('dubbel', 2),
 ('skunking', 2),
 ('laws', 2),
 ('draught', 2),
 ('pouring', 2),
 ('pairing', 2),
 ('keg', 2),
 ('water', 2),
 ('trappist', 2),
 ('carbonation', 2)]
```

```
tags_rdd = h.flatMap(lambda x: [y if y is not None else "" for y in x])\
    .flatMap(lambda x: [x[y] for y in range(0, len(x))])

x = tags_rdd.collect()
tags_str = ''
for y in range(len(x)):
    tags_str += f"{x[y]} "

from wordcloud import WordCloud
import matplotlib.pyplot as plt
wc = WordCloud(background_color ='white').generate(tags_str)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wc)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



```
tags_rdd = l.flatMap(lambda x: [y if y is not None else "" for y in x])\
    .flatMap(lambda x: [x[y] for y in range(0, len(x))])

x = tags_rdd.collect()
tags_str = ''
for y in range(len(x)):
    tags_str += f"{x[y]} "

from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```python
wc = WordCloud(background_color ='white').generate(tags_str)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wc)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



```python
l.flatMap(lambda x: [y if y is not None else "" for y in x])\
    .flatMap(lambda x: [x[y] for y in range(0, len(x))])\
    .map(lambda x: (x, 1))\
    .aggregateByKey(0, (lambda acc,x: acc + x ), (lambda acc1,acc2: acc1+acc2))\
    .filter(lambda x: x[1] > 1)\
    .sortBy(lambda x: x[1], ascending=False)\
    .collect()
```

```
[('wine', 20),
 ('recommendations', 14),
 ('health', 11),
 ('taste', 8),
 ('history', 8),
 ('spirits', 7),
 ('breweries', 6),
 ('storage', 5),
 ('drinking', 4),
 ('flavor', 4),
 ('brewing', 4),
 ('alcohol', 4),
 ('pairing', 3),
 ('science', 3),
 ('champagne', 3),
 ('distillation', 3),
 ('temperature', 2),
 ('glassware', 2),
 ('scotch', 2),
 ('hangover', 2),
 ('water', 2),
 ('rum', 2),
 ('liquor', 2),
 ('draught', 2),
 ('vodka', 2),
 ('mead', 2),
 ('carbonation', 2),
 ('canada', 2),
 ('drink', 2),
 ('recipes', 2)]
```

3.4 procent przypadków kiedy najwyżej oceniana odpowiedź to nie zaakceptowana odpowiedź

```
#1 - Question 2 - Answer 3 - Wiki 4 - TagWikiExcerpt 5 - TagWiki 6 - ModeratorNomination 7 -  WikiPla
posts_tmp = posts.select(f.col('id'), f.col("parent_id"), f.col('accepted_answer_id'), f.col('answer_
questions = posts_tmp.filter('post_type_id == 1 and answer_count > 0')\
    .select(f.col('id').alias('q_id'), f.col('accepted_answer_id'))

answers = posts_tmp.filter(f.col('post_type_id') == 2)\
    .select(f.col('id').alias('a_id'), f.col('parent_id'), f.col('score'))
```

```
from pyspark.sql import Window

window_partition_agg = Window.partitionBy("q_id")

questions.join(answers, on=questions.q_id == answers.parent_id)\
    .sort(['q_id', 'a_id'])\
    .withColumn("max_score", f.max(f.col("score")).over(window_partition_agg))\
    .filter(f.col("score") == f.col("max_score"))\
    .filter(f.col("accepted_answer_id").isNotNull())\
    .withColumn("is_accepted_best", f.col("accepted_answer_id") == f.col("a_id"))\
    .agg(
        f.sum(f.col("is_accepted_best").cast("integer")).alias("sum"),
        f.count(f.col("q_id")).alias("count")
    )\
    .withColumn("percent", (f.col("count") - f.col("sum")) / f.col("count") * 100).show()
```

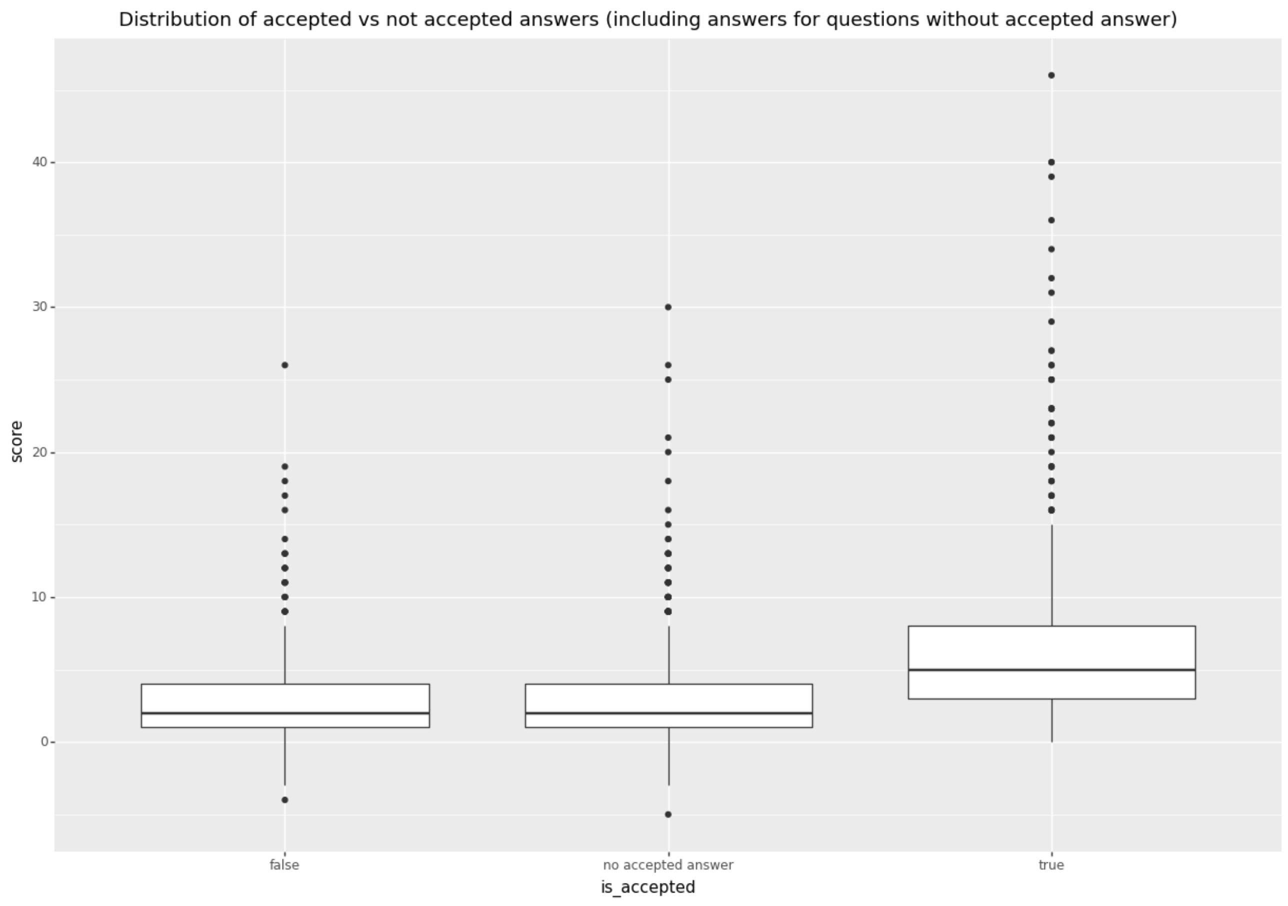
```
+---+-----+------------------+
|sum|count|           percent|
+---+-----+------------------+
|641|  735|12.789115646258503|
+---+-----+------------------+
```

3.5 rozkład ocen odpowiedzi zaakceptowanych vs pozostałych (średnia, odchylenie, minimum, maksimum)

```
window_partition_agg = Window.partitionBy("q_id")

questions.join(answers, on=questions.q_id == answers.parent_id)\
    .sort(['q_id', 'a_id'])\
    .withColumn("is_accepted", f.col("accepted_answer_id") == f.col("a_id"))\
    .groupBy(f.col("is_accepted")).agg(
        f.avg(f.col("score")).alias("avg_score"),
        f.stddev(f.col("score")).alias("std_score"),
        f.min(f.col("score")).alias("min_score"),
        f.max(f.col("score")).alias("max_score"),
        f.count("a_id")
    ).show()
```

```
+-----------+------------------+------------------+---------+---------+-----------+
|is_accepted|         avg_score|         std_score|min_score|max_score|count(a_id)|
+-----------+------------------+------------------+---------+---------+-----------+
|       null|2.7551686615886832|3.1818372333580007|       -5|       30|        919|
|       true| 6.395043731778426| 5.915949387154137|        0|       46|        686|
|      false|2.5841694537346713|2.7353292123298076|       -4|       26|        897|
+-----------+------------------+------------------+---------+---------+-----------+
```

```
accepted_df = questions.join(answers, on=questions.q_id == answers.parent_id)\
    .sort(['q_id', 'a_id'])\
    .withColumn("is_accepted", (f.col("accepted_answer_id") == f.col("a_id")).cast("string"))\
    .withColumn("is_accepted", f.when(f.col("is_accepted").isNull(), "no accepted answer").otherwise(

(ggplot(accepted_df, aes(x="is_accepted", y="score"))\
    +geom_boxplot()\
    +ggtitle("Distribution of accepted vs not accepted answers (including answers for questions witho
```



```
<ggplot: (8785942528877)>
```

3.6 top N tagów które wygenerowały najwięcej wyświetleń

```
tags_views = posts.select(['tags', 'view_count'])
tags_views_agg = regexp_extract_all(tags_views, r'<(\w+)>', 99, "tags", "tags_split", True)\
    .select([f.explode(f.col('tags_split')).alias("tag"), f.col("view_count")])\
    .filter(f.col("view_count").isNotNull())\
    .groupBy('tag')\
    .agg(
        f.sum("view_count").alias("sum_views")
    )
```

```
tag_top_views.head(20)
```

	tag	sum_views
0	taste	1330670
1	health	1286001
2	preservation	682216
3	storage	542860
4	whiskey	464756
5	bourbon	330268

	tag	sum_views
6	brewing	307892
7	ipa	291935
8	spirits	255328
9	drinking	225924
10	temperature	218203
11	drink	204991
12	tequila	196689
13	alcohol	188615
14	recommendations	185154
15	wine	181081
16	style	168681
17	flavor	168594
18	history	167414
19	pairing	164589

3.7 liczba postów w czasie dla każdego z top N tagów (lineplot/barplot)

```
#1 - Question 2 - Answer 3 - Wiki 4 - TagWikiExcerpt 5 - TagWiki 6 - ModeratorNomination 7 -  WikiPla
posts_tmp = posts.select(f.col('id'), f.col('creation_date'), f.col('tags'))

posts_tags_time = regexp_extract_all(posts_tmp, r'<(\w+)>', 99, "tags", "tags_split", True).withColum

top_posts_tags_time = posts_tags_time.join(tag_top_views, on="tag", how="inner").select(f.col('id'),
```

```
(ggplot(top_posts_tags_time_agg_pd, aes("date", "count"))\
    + scale_x_datetime()\
    + geom_col() \
    + facet_wrap("tag", ncol=3) \
    + ylim(0, 15))
```

3.8 najczęściej pojawiające się słowa w tytułach (z pominięciem stopwords

```
from bs4 import BeautifulSoup
from html import unescape
from pyspark.sql.functions import udf, regexp_replace
from pyspark.sql.types import *
# remove html tags
def tags_remove(s):
    soup = BeautifulSoup(unescape(s), 'lxml')
    return soup.text

udf_tags_remove = udf(lambda m: tags_remove(m))

titles = posts.filter(f.col("title").isNotNull()).select(f.col("title"))\
    .withColumn("title_clean", f.lower(f.col("title")))\
    .withColumn("title_clean", regexp_replace('title_clean', "[^a-zA-Z\\s]", " "))
```

```
from pyspark.ml.feature import Tokenizer, StopWordsRemover
from nltk.stem.snowball import SnowballStemmer
udf_filter_length = udf(lambda row: [x for x in row if len(x) > 1], ArrayType(StringType()))

stemmer = SnowballStemmer(language='english')
stemmer_udf = udf(lambda token: stemmer.stem(token), StringType())

tokenizer = Tokenizer(inputCol='title_clean', outputCol='words_token')
title_tokens = tokenizer.transform(titles).withColumn('words_token', udf_filter_length(f.col('words_t




remover = StopWordsRemover(inputCol='words_token', outputCol='words_no_stop')
title_tokens_no_stop = remover.transform(title_tokens)
exploded = title_tokens_no_stop.withColumn("words", f.explode(f.col("words_no_stop")))

title_stem = exploded.withColumn('words_stem', stemmer_udf("words"))

word_lookup = title_stem.select([f.col("words"), f.col("words_stem")]).distinct()
word_lookup.show() # TODO aggregate this
title_stem.groupBy("words_stem").agg(f.count("title").alias('count')).orderBy('count', ascending=Fals
```

```
+---------------+-----------+
|          words| words_stem|
+---------------+-----------+
|         opened|       open|
|antidepressants|antidepress|
|          taken|      taken|
|    alternative|     altern|
|       learning|      learn|
|         sherry|     sherri|
|      regionali|  regionali|
|      archetype|   archetyp|
|      inhibitor|  inhibitor|
|        outside|     outsid|
|            bay|        bay|
|        sangria|    sangria|
|         invest|     invest|
|        togther|    togther|
|           fake|       fake|
|         kahlua|     kahlua|
|       imported|     import|
|         tables|       tabl|
|         desire|      desir|
|        bavaria|    bavaria|
+---------------+-----------+
only showing top 20 rows

+----------+-----+
|words_stem|count|
+----------+-----+
|      beer|  476|
|      wine|  147|
|     drink|  104|
|   alcohol|   88|
|    differ|   72|
|     bottl|   68|
|       use|   50|
|      tast|   47|
|      brew|   43|
|      make|   41|
|      good|   33|
|  cocktail|   29|
|       age|   27|
| recommend|   26|
|       ale|   26|
|      like|   24|
|      made|   23|
|   whiskey|   23|
|    spirit|   23|
|       one|   22|
+----------+-----+
only showing top 20 rows
```

3.9 procent użytkowników którzy nigdy nic nie zapostowali

```
# users.show(2)
# posts.show(2)

users_posts = (users.join(
    (posts.select(f.col('id').alias('post_id'), f.col('owner_user_id'))), on=[users.id == posts.owner
    .filter("id IS NOT NULL and NOT id = -1")
    )

users_posts.select([f.col('id'), f.col('post_id')])\
    .groupBy("id")\
        .agg(
            f.count(f.col('post_id')).alias('post_count')
        )\
    .agg(
        f.sum(f.when(f.col('post_count') == 0, f.lit(1)).otherwise(f.lit(0))).alias("not_posted"),
        f.count('id').alias('all')
    ) \
    .withColumn('% not posted', (f.col("not_posted") / f.col('all') * 100)).show()

# (users.join(
#     (posts.select(f.col('id').alias('post_id'), f.col('owner_user_id'))), on=[users.id == posts.own
#     .filter(f.col('id').isNotNull() & f.col('post_id').isNull())\
#     .select([f.col('id'), f.col('display_name')]).distinct()\
#     .show()
#     )
```

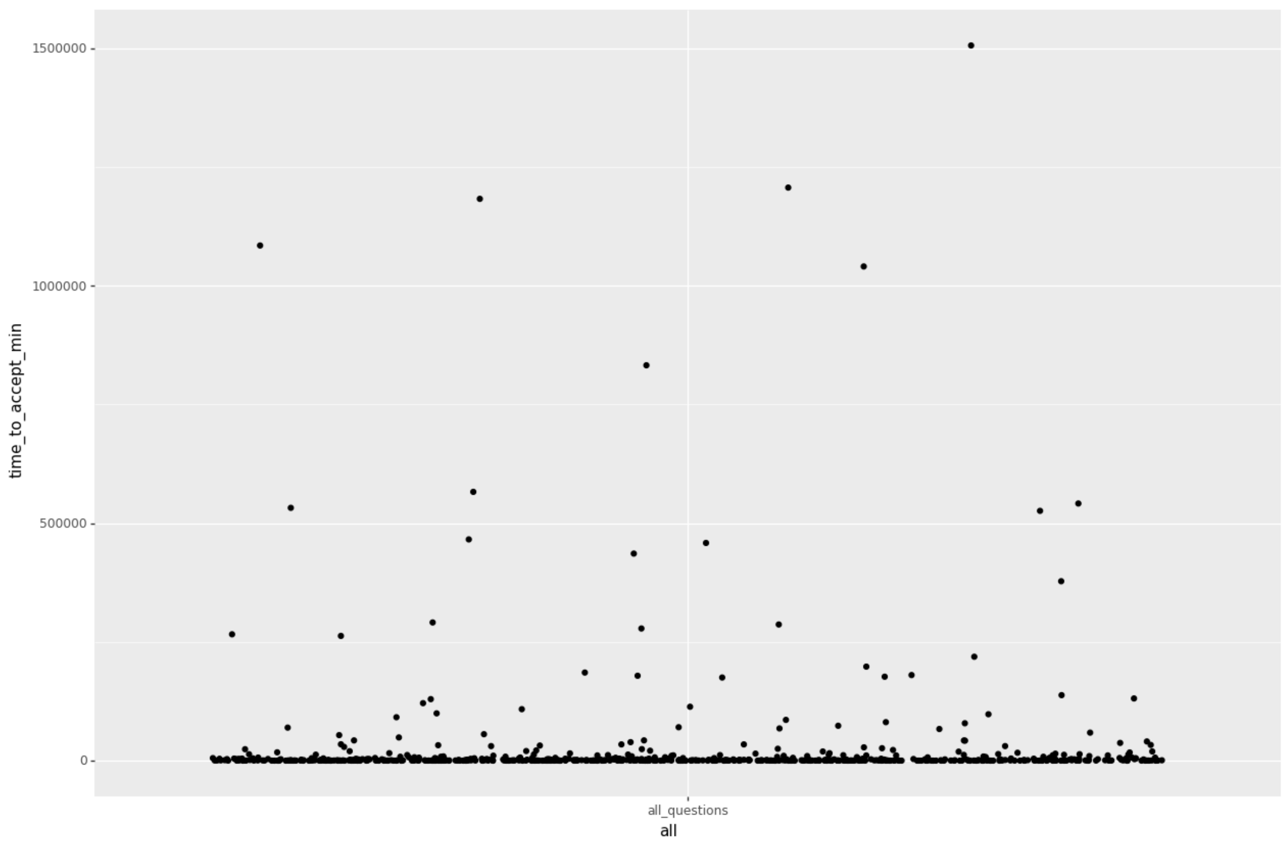
```
+----------+----+-----------------+
|not_posted| all|     % not posted|
+----------+----+-----------------+
|      7691|8947|85.96177489661339|
+----------+----+-----------------+
```

3.10 średni czas od pojawienia się pytania do pojawienia się zaakceptowanej odpowiedzi

```
# keep only questions with answers
questions = posts.filter(f.col('post_type_id') == 1).filter(f.col('answer_count') > 0).select([f.col(
answers  = posts.filter(f.col('post_type_id') == 2).select([f.col('id').alias('a_id'), f.col('parent_
#posts.show(1, vertical=True)
time_to_accept = questions.join(answers, on=[questions.accepted_answer_id==answers.a_id])\
    .withColumn('time_to_accept_sec', f.unix_timestamp('a_creation_date') - f.unix_timestamp('q_creat
    .withColumn('time_to_accept_min', f.round(f.col('time_to_accept_sec') / 60, 2))

time_to_accept.agg(
        f.avg('time_to_accept_min'),
        f.stddev('time_to_accept_min'),
        f.percentile_approx("time_to_accept_min", [0.25, 0.5, 0.75], 1000000).alias("quantiles")
    ).show(truncate=False)


time_to_accept_pd = time_to_accept.withColumn('all', f.lit("all_questions")).toPandas()

from plotnine import geom_jitter
(ggplot(time_to_accept_pd, aes(x='all', y="time_to_accept_min"))\
    +geom_jitter())
```

```
+-----------------------+------------------------------+------------------------+
|avg(time_to_accept_min)|stddev_samp(time_to_accept_min)|quantiles               |
+-----------------------+------------------------------+------------------------+
|25244.9435714286       |123338.6325101642             |[141.53, 753.25, 3605.8]|
+-----------------------+------------------------------+------------------------+
```

```
/config/workspace/.venv/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning
/config/workspace/.venv/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning
```



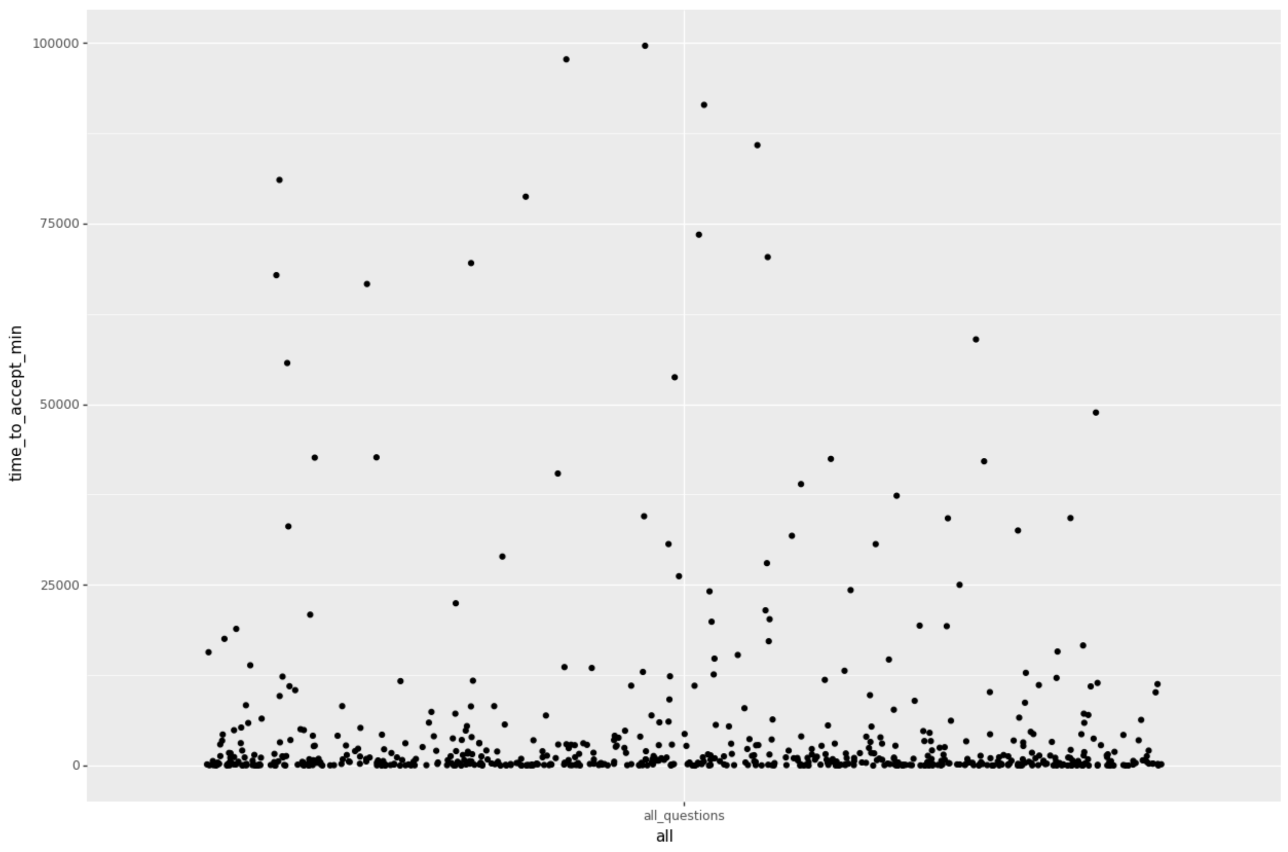
```
<ggplot: (8785939115177)>
```

3.10.1 remove outliers

```
no_outliers = time_to_accept.filter(f.col('time_to_accept_min') < 100000)

no_outliers.agg(
        f.avg('time_to_accept_min'),
        f.stddev('time_to_accept_min'),
        f.percentile_approx("time_to_accept_min", [0.25, 0.5, 0.75], 1000000).alias("quantiles")
    ).show(truncate=False)

no_outliers_pd = no_outliers.withColumn('all', f.lit("all_questions")).toPandas()
(ggplot(no_outliers_pd, aes(x='all', y="time_to_accept_min"))\
    +geom_jitter())
```

```
+-----------------------+-------------------------------+-------------------------+
|avg(time_to_accept_min)|stddev_samp(time_to_accept_min)|quantiles                |
+-----------------------+-------------------------------+-------------------------+
|4769.308792048925      |12694.627623679486             |[127.72, 644.93, 2922.82]|
+-----------------------+-------------------------------+-------------------------+
```

```
/config/workspace/.venv/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning
/config/workspace/.venv/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning
```


100000
75000
50000
25000
0
time_to_accept_min
all_questions
all

```
<ggplot: (8785939033767)>
```

References

4 Załączniki

4.1 Polecenia budujące infrastrukturę

4.1.1 EMR

```
aws emr create-cluster --name="MyEMRCluster" \
    --release-label emr-6.8.0 \
    --applications Name=JupyterHub Name=Hadoop Name=Spark \
    --log-uri s3://emr-logs-beer-and-wine/MyJupyterClusterLogs \
    --use-default-roles \
    --instance-groups InstanceGroupType=MASTER,InstanceCount=1,InstanceType=m4.large InstanceGroupTyp
    --ebs-root-volume-size 32 \
    --configurations file://emr-configurations.json \
    --bootstrap-actions Path=s3://misc-beer-and-wine/install_python_libraries.sh,Name=InstallJupyterL
```

Plik `install-my-jupyter-libraries.sh` dostępny jest pod poniższym adresem

4.1.2 S3

Koszyki S3 zostały utworzone przy pomocy poniższego polecenia:

```
aws s3api create-bucket --acl private --bucket <nazwa koszka>
```