Analiza wpływu czynników treściowych na interakcje społeczne dla witryny mathematica.meta.stackexchange.com

Sylwia Snarska

STRESZCZENIE

Praca opisuje analizę zbioru danych pochodzących z platformy Stack Exchange dla witryny 'mathematica.meta.stackexchange.com'. Temat został wybrany z uwagi na zainteresowanie matematyką oraz chęć zgłębienia wiedzy dotyczącej interakcji społeczności związanej z programem Mathematica. W ramach pracy, przeprowadzono analizę różnych aspektów platformy, takich jak liczba postów w czasie, aktywność użytkowników, oceny pytań i odpowiedzi, a także interakcje między użytkownikami. Dodatkowo, zbadano zależności między liczbą odpowiedzi na post a jego oceną, a także między oceną posta a liczbą słów w treści. Wykorzystano również techniki przewidywania ocen i ulubionych postów na podstawie różnych cech, takich jak liczba wyświetleń, ocena, długość treści i liczba komentarzy. Analiza danych stanowi cenne źródło informacji dla badaczy społeczności online oraz użytkowników zainteresowanych platformą Stack Exchange i programem Mathematica."

Słowa kluczowe: Analiza danych, społeczność użytkowników, badanie zależności

---

THESIS TITLE IN ENGLISH

The paper describes an analysis of a dataset obtained from the Stack Exchange platform for the 'mathematica.meta.stackexchange.com' website. The choice of this topic stems from my interest in mathematics and the desire to gain knowledge about the community interaction related to the Mathematica program. The study examines various aspects of the platform, such as the number of posts over time, forum activity of the top 10 users, a comparison of the highest and lowest rated questions, the percentage of cases where the highest rated answer is not accepted, the distribution of ratings for accepted vs non-accepted answers, the average time from question appearance to the appearance of an accepted answer, the average number of comments per post in the top 10 tags. The data analysis serves as a valuable resource for researchers studying online communities and users interested in the Stack Exchange platform and the Mathematica program.

Keywords: Data analysis, user community, exploring dependencies

Spis treści

1. Wstęp

Jako inżynier matematyki stosowanej, zdecydowałam się na analizę danych pochodzących z platformy Stack Exchange dla witryny "mathematica.meta.stackexchange.com". Wybór tego tematu wynika z mojego zainteresowania matematyką oraz chęci zgłębienia wiedzy na temat interakcji społeczności związanej z programem Mathematica - to wszechstronne i potężne środowisko obliczeniowe oraz zaawansowany system do przeprowadzania obliczeń matematycznych, symulacji, analizy danych, programowania i wizualizacji. Zatem zbiór ten stanowi cenne źródło danych, które mogę wykorzystać do badania wzorców, trendów i dynamiki tej społeczności.

Głównym celem mojego projektu było przeprowadzenie wszechstronnej analizy danych wybranego zbioru. Spodziewałam się odkrycia interesujących informacji na temat aktywności użytkowników, dynamiki forum, popularności tagów, wzorców oceniania postów oraz zależności między różnymi cechami postów a ich ocenami. Projekt miał na celu uzyskanie nowych spostrzeżeń i wglądu w społeczność związaną z Mathematica, a także dostarczenie wartościowych informacji i wniosków dla czytelników.

Po przeczytaniu mojej pracy dowiesz się o wynikach i wnioskach płynących z analizy danych. Przedstawiam także różne pytania badawcze dotyczące aktywności użytkowników, czasu trwania, oceniania postów, popularności tagów, a także zawieram predykcje dotyczące pewnych zmiennych na podstawie istniejących danych. Celem mojej pracy jest zrozumienie dynamiki i charakterystyki społeczności związanej z Mathematica, dostarczenie wiedzy o wzorcach i tendencjach w interakcjach użytkowników oraz zapewnienie wartościowych informacji dla czytelników zainteresowanych tym obszarem badawczym. Otwierając kolejny rozdział, będę szczegółowo analizować pytania badawcze, a także przedstawiać metody analizy danych oraz wyniki uzyskane w trakcie badania.

1. Zbieranie surowych danych

Proces analizy danych zaczyna się od zbierania surowych danych, które będą podstawą naszej dalszej pracy. W tym rozdziale skupimy się na krokach związanych z pozyskaniem, rozpakowaniem i przeniesieniem danych do odpowiednich zasobów w chmurze. Wykorzystamy narzędzia takie jak Cloud9 i S3, aby efektywnie zarządzać danymi i zapewnić ich dostępność oraz bezpieczeństwo.

Pierwszym krokiem było pobranie danych pochodzących z Stack Exchange Data Dump z witryny „mathematica.meta.stackexchange.com”. Folder zawiera 8 plików w formacie xml. Pierwszy z plików o nazwie „Badges” zawiera informacje o odznakach w trzech rangach – Gold, Siver, Bronze, które są przyznawane użytkownikom forum. Ponadto plik zawiera wszelkie inne niezbędne informacje typu id użytkownika, pozwalające na dalsze połączenie przyznanej odznaki z odpowiednią osobą. Kolejnym plikiem jest „Comments” zawierający treści komentarzy wraz z innymi identyfikującymi je treściami typu id komentarza, liczba punktów oddanych przez użytkowników za lub przeciw, data utworzenia. Kolejny plik „PostHistory” zawiera historię zmian postów, „PostLinks” przechowuje dane o linkach umieszczonych w poście, zaś „Tags’’ informację o przypisanych tagach. Następnym plikiem jest „Posts” – przechowuje zarówno treści postów jak i inne niezbędne dane typu identyfikator, data utworzenia, ocena, tytuł, przypisane tagi i wiele innych informacji. Z kolei plik „Users’’ zawiera informację o użytkownikach, ich reputacji na forum, własnoręczne opisy siebie w sekcji „AboutMe” czy datę ostatniego logowania. Ostatnim plikiem jest „Votes’’ mówiący o oddanych głosach przez użytkowników forum na posty.

Dane były spakowane programem 7-zip. Po rozpakowaniu pliku dane zostały przeniesione do AWS S3 (Simple Storage Service), usługi do przechowywania obiektów w chmurze, która zapewnia skalowalne i wydajne przechowywanie danych. W tym celu należy utworzyć bucket (pojemnik) na S3, który będzie służył jako miejsce

przechowywania danych. Po zakończeniu tych kroków będziemy mieli surowe dane, gotowe do wczytania i dalszej analizy przy użyciu Sparka. W kolejnych rozdziałach będziemy skupiać się na przetwarzaniu i analizie tych danych, odpowiadając na różne pytania analityczne i tworząc wizualizacje.

2. Przetwarzanie i eksploracja danych

W tym rozdziale przedstawię, jakie przetwarzanie wymagały dane do dalszej analizy. Pierwszym krokiem jest pobranie danych z wcześniej utworzonego bucketu na S3. W tym celu korzystam z usługi Cloud9, która jest zintegrowanym środowiskiem programistycznym dostarczanym przez Amazon Web Services *(AWS)*. AWS Cloud9 to usługa chmurowa, która umożliwia tworzenie, edycję, debugowanie i wdrażanie aplikacji w trybie online. Dostarcza kompletną infrastrukturę deweloperską w chmurze, eliminując potrzebę konfiguracji lokalnego środowiska programistycznego. Konfiguruję własne środowisko w chmurze i uruchamiam w terminalu Jupyter Notebooka. W tym celu korzystam również z usługi AWS EC2 *(Elastic Compute Cloud)*, która umożliwia uruchamianie i zarządzanie wirtualnymi maszynami *(instancjami)* w chmurze. EC2 pozwala na elastyczne skalowanie zasobów obliczeniowych, umożliwiając dostosowanie mocy obliczeniowej do bieżących potrzeb. Tworzę więc i konfiguruję nową instancję EC2, z której odczytuję publiczny adres DNS instancji. Następnie w połączeniu z uruchomieniem środowiska w AWS Cloud9 uruchamiam Jupyter Notebook.

Do korzystania z niezbędnych bibliotek w Sparku wykorzystuję usługę EMR *(Elastic MapReduce)*. EMR to usługa umożliwiająca łatwe i skalowalne przetwarzanie dużych zbiorów danych. Zapewnia zarządzanie i uruchamianie klastrów Apache Hadoop, Apache Spark, Apache Hive, Apache HBase, Apache Flink i innych frameworków do przetwarzania i analizy danych. Tworzę i konfiguruję klaster Sparka oraz klucz dostępu. Po wykonaniu tych czynności mogę przystąpić do wczytywania plików.

Aby wczytać plik XML z S3 do DataFrame w Sparku, korzystam z biblioteki spark-xml. Wymaga ona dodatkowej biblioteki com.databricks:spark-xml. Dodatkowo, importuję bibliotekę SparkSession z modułu pyspark.sql. Wbudowana w niej funkcja spark.read.format("xml") posłuży mi do wczytania pierwszego z plików - Comments.

```
from pyspark.sql import SparkSession


spark = SparkSession.builder \
    .appName("Read XML from S3") \
    .getOrCreate()

xml_path = "s3://123project456/folder/Comments.xml"

Comments = spark.read.format("xml") \
    .option("rowTag", "root_element") \
    .load(xml_path)
```

Rysunek 1. Wgranie pliku xml

Po wczytaniu pliku z komentarzami do Dataframe, przeglądam kilka wierszy aby zobaczyć strukturę danych.

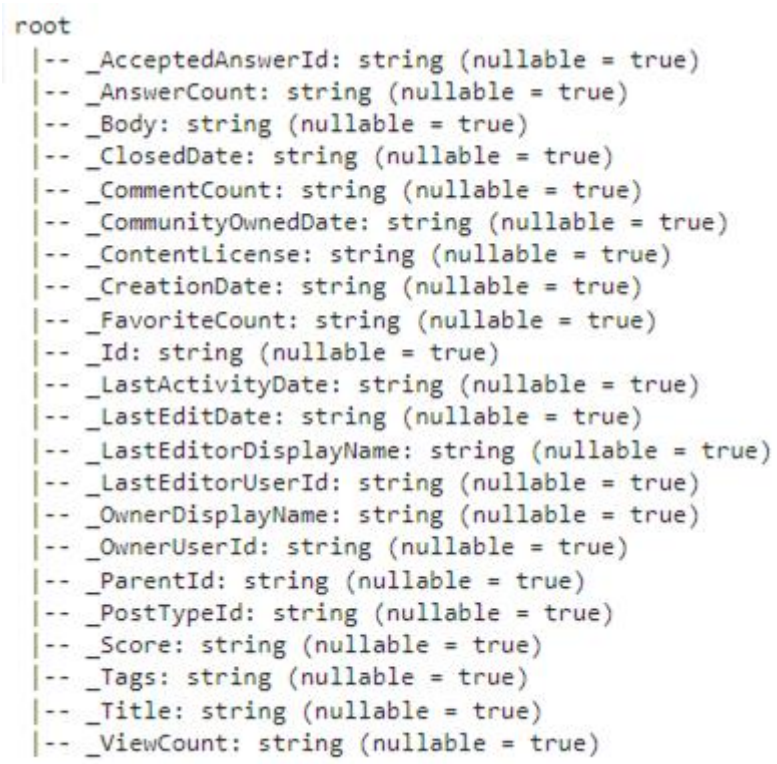
_CreationDate	_Id	_PostId	_Score	Text	_UserDisplayName	UserId
null	1	3	3	It looks like LaT...	null	9
null	3	2	3	Could we make a m...	null	39
null	4	3	0	Seems like it is.	null	52
null	5	7	1	I must second thi...	null	52
null	7	7	9	And please...let'...	null	91
null	10	7	6	Agreed. Even vete...	null	89
null	11	12	3	`?Global\`*` It w...	null	50
null	14	7	1	Agree with both t...	null	12
null	15	16	2	Yeah, it would be...	null	50
null	16	19	0	It's not terribly...	null	95
null	17	13	1	Actually, all plo...	null	52
null	18	20	0	Thanks. Would nic...	null	95
null	19	20	0	Ah, true, but ala...	null	52
null	20	21	1	It could be usefu...	null	95
null	21	22	0	I agree with this...	null	95

Rysunek 2. Przykładowe komentarze

Kolumny przechowują dane takie jak treść komentarza, ocena, identyfikator autora komentarza itp.

Kolejny plik zawierający treść i informację o postach wczytuję analogicznie jak plik Comments. Również korzystam z metody show() na utworzonym Dataframe, aby zobaczyć

jak wyglądają dane w pliku oraz z metody printSchema() do wyświetlenia struktury Dataframe.



Rysunek 3. Struktura dataframe Posts

Jak widać na powyższym Rysunku, dataframe przechowuje różne dane związane z postami, takie jak identyfikator, treść, liczba odpowiedzi, liczba komentarzy, ocena, tagi i wiele innych.

Kolejny plik – Badges zawiera informację o odznakach. Przyjrzyjmy się jego strukturze przy ponownym skorzystaniu z metod printSchema() oraz show().

```
root
 |-- _Class: integer (nullable = true)
 |-- _Date: timestamp (nullable = true)
 |-- _Id: integer (nullable = true)
 |-- _Name: string (nullable = true)
 |-- _TagBased: boolean (nullable = true)
 |-- _UserId: integer (nullable = true)

+------+--------------------+---+-------------+---------+-------+
|_Class|               _Date|_Id|        _Name|_TagBased|_UserId|
+------+--------------------+---+-------------+---------+-------+
|     3|2012-01-17 20:56:...|  1|Autobiographer|    false|      3|
|     3|2012-01-17 21:06:...|  2|Autobiographer|    false|      1|
|     3|2012-01-17 21:11:...|  3|Autobiographer|    false|      8|
|     3|2012-01-17 21:21:...|  4|Autobiographer|    false|     39|
|     3|2012-01-17 21:26:...|  5|Autobiographer|    false|     35|
+------+--------------------+---+-------------+---------+-------+
```

Rysunek 4. Schemat Badges

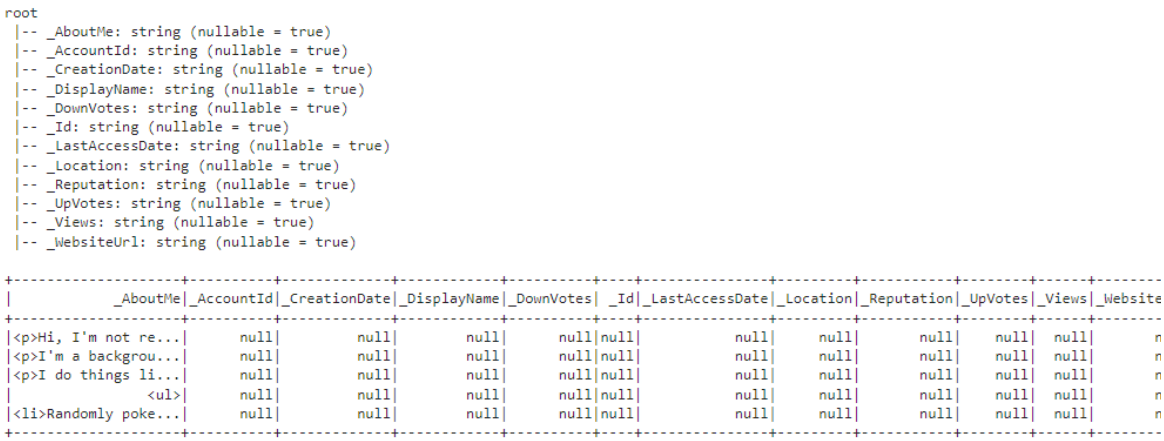
Z powyższego Rysunku możemy odczytać jakie są przykładowe odznaki przyznawane użytkownikom.

Plik Tags zawiera informację o użytych tagach na forum oraz informacje takie jak liczbę wystąpień, identyfikator excerpt posta, informacje o moderatorze, wymagane tagi itp.

```
+------+-------------+---+---------------+-----------+----------------+-----------+
|_Count|_ExcerptPostId|_Id|_IsModeratorOnly|_IsRequired|        _TagName|_WikiPostId|
+------+-------------+---+---------------+-----------+----------------+-----------+
|    30|         null|  1|           null|       true|             bug|       null|
|   102|         null|  2|           null|       true| feature-request|       null|
|   713|         null|  3|           null|       true|      discussion|       null|
|     3|         null|  4|           true|       null|             faq|       null|
|    62|         null|  5|           true|       null|status-completed|       null|
|     2|         null|  6|           true|       null| status-declined|       null|
|     8|         2684|  7|           true|       null| status-bydesign|       2683|
|     1|         2680|  8|           true|       null|  status-norepro|       2679|
|     1|         2682|  9|           true|       null|status-reproduced|       2681|
|     0|         2676| 10|           true|       null|  status-planned|       2675|
|     0|         2674| 11|           true|       null| status-deferred|       2673|
|     0|         2565| 12|           true|       null|   status-review|       2564|
|   163|         2678| 13|           null|       true|         support|       2677|
|     4|         null| 14|           null|       null|      reputation|       null|
|     6|         null| 15|           null|       null|  site-promotion|       null|
|    22|         null| 16|           null|       null|       questions|       null|
|    15|         null| 18|           null|       null|        comments|       null|
|     3|         null| 19|           null|       null|          badges|       null|
|     9|         null| 20|           null|       null|         answers|       null|
+------+-------------+---+---------------+-----------+----------------+-----------+
```

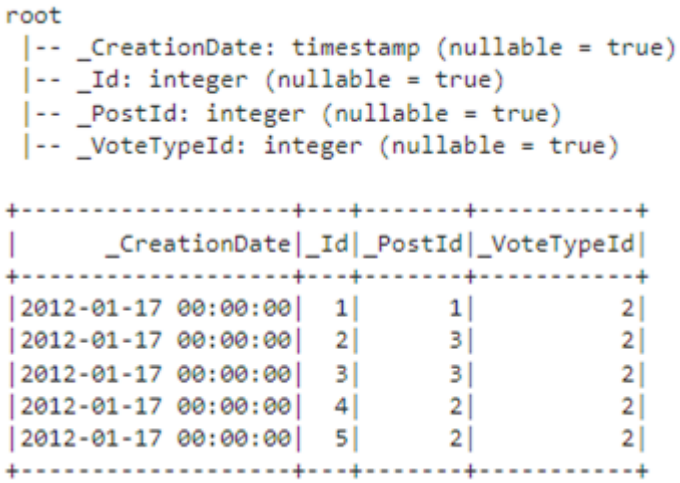
Rysunek 5. Dataframe Tags

Kolejny dataframe Users przechowuje dane związane z użytkownikami, takie jak informacje osobiste, identyfikator, datę utworzenia, nazwę wyświetlaną, reputację, liczbę upvote'ów i wiele innych. Jak widać wiele kolumn zawiera puste rekordy.



Rysunek 6. Dataframe Users

Dataframe Votes składa się z czterech kolumn – przechowuje informację o głosach oddanych na poszczególne posty, data oddania głosu itp.



Rysunek 7. Dataframe Votes

DataFrame o nazwie "PostLink" zawiera informacje o linkach między postami. Te informacje umożliwiają śledzenie powiązań między różnymi postami w zbiorze danych i analizowanie relacji między nimi.

```
root
 |-- _CreationDate: timestamp (nullable = true)
 |-- _Id: integer (nullable = true)
 |-- _LinkTypeId: integer (nullable = true)
 |-- _PostId: integer (nullable = true)
 |-- _RelatedPostId: integer (nullable = true)

+--------------------+---+-----------+-------+--------------+
|       _CreationDate|_Id|_LinkTypeId|_PostId|_RelatedPostId|
+--------------------+---+-----------+-------+--------------+
|2012-01-17 22:17:...| 12|          1|      2|             5|
|2012-01-18 13:47:...|145|          1|     39|             8|
|2012-01-19 12:09:...|441|          1|     69|             2|
|2012-01-20 02:42:...|468|          1|     69|             3|
|2012-01-20 19:34:...|570|          1|     86|            11|
+--------------------+---+-----------+-------+--------------+
only showing top 5 rows
```

Rysunek 8. Dataframe PostLinks

Ostatni dataframe Posthistory przechowują różne informacje związane z historią postów, takie jak komentarze, data utworzenia, identyfikator.

```
+-----------------+--------------------+--------------------+---------+-----------------+-------+--------------------+--------------------+---------------+-------+
|         _Comment|     _ContentLicense|       _CreationDate|      _Id|_PostHistoryTypeId|_PostId|       _RevisionGUID|               _Text|_UserDisplayName|_UserId|
+-----------------+--------------------+--------------------+---------+-----------------+-------+--------------------+--------------------+---------------+-------+
|             null|        CC BY-SA 3.0|2012-01-17T21:17:...|        1|                2|      1|bc214fb9-0b32-4eb...|This question is ...|           null|   null|
|Is this easy to s...| can we have some...| e.g. to treat `(...| not `//`?"|             null|      8|                null|                null|           null|   null|
|             null|        CC BY-SA 3.0|2012-01-17T21:17:...|        2|                1|      1|bc214fb9-0b32-4eb...|Can we have code ...|           null|      8|
|             null|        CC BY-SA 3.0|2012-01-17T21:17:...|        3|                3|      1|bc214fb9-0b32-4eb...|<feature-request>...|           null|      8|
|             null|        CC BY-SA 3.0|2012-01-17T21:25:...|        4|                2|      2|4f9c85c0-8500-49b...|What are best pra...|           null|   null|
+-----------------+--------------------+--------------------+---------+-----------------+-------+--------------------+--------------------+---------------+-------+
```

Rysunek 9. Dataframe Posthistory

3. Czyszczenie danych

W tym rozdziale skupimy się na etapie czyszczenia danych, który jest istotnym krokiem w procesie analizy danych. Czyszczenie danych ma na celu standaryzację i przygotowanie danych do dalszej analizy. W ramach tego rozdziału przeprowadzimy szereg operacji na danych, takich jak:

1. Przeanalizowanie struktury danych i identyfikacja niezbędnych operacji czyszczenia.
2. Oczyszczenie danych poprzez usunięcie zbędnych kolumn, wierszy zawierających braki danych lub duplikaty.
3. Standaryzacja formatów danych, np. konwersja dat do odpowiednich typów, normalizacja tekstu.
4. Zastosowanie filtrów i reguł czyszczenia, aby usunąć niepożądane znaki, słowa kluczowe, znaczniki HTML itp.
5. Ujednolicenie nazw kolumn i wartości, aby zapewnić spójność danych.

Czyszczenie danych jest niezwykle istotnym etapem, ponieważ poprawnie oczyszczone dane są niezbędne do uzyskania dokładnych i wiarygodnych wyników analizy. Dzięki zastosowaniu narzędzi i technik dostępnych w środowisku Spark, możemy skutecznie przetworzyć duże zbiory danych w sposób efektywny i skalowalny.

3.1. Pliki zawierające kolumny z tekstem

Czyszczenie danych rozpocznę od pliku Comments. Nazwy kolumn w pliku zawierają znak „_” przed każdą nazwą kolumny,co jest zbędne i pogarsza czytelność. Metoda withColumnRenamed() pozwoli na zmianę nazw odpowiednich kolumn.

```
Comments = Comments.withColumnRenamed("Id", "comment_id") \
    .withColumnRenamed("_PostId", "post_id") \
    .withColumnRenamed("_Score", "score") \
    .withColumnRenamed("Text", "comment_text") \
    .withColumnRenamed("_CreationDate", "creation_date") \
    .withColumnRenamed("_UserDisplayName", "user_display_name") \
    .withColumnRenamed("UserId", "user_id")\
.withColumnRenamed("_ContentLicense", "ContentLicense")
```

Rysunek 10. Zmiana nazw kolumn

Należy również przyjrzeć się typom kolumn. Uruchamiając w poprzednim rozdziale metodę printSchema() można było zauważyć, że nie wszystkie kolumny mają odpowiedni typ danych. Przykładowo, kolumna „Id" nie powinna mieć typu string, jeśli przechowuje wyłącznie liczby całkowite. Analogiczna sytuacja występuje dla kolumn „Score", „PostId'', „User_Id''. Z kolei kolumna „CreationDate'' przechowująca datę utworzenia komentarza również nie powinna być typu string. Oprócz daty widnieje również godzina publikacji komentarza. W takim razie odpowiedni będzie typ danych timestamp. Aby przekonwertować na tej typ kolumny niezbędne będzie zaimportowanie biblioteki pyspark.sql.functions, z której wykorzystam funkcję to_timestamp.

```
from pyspark.sql.functions import to_timestamp

Comments = Comments.withColumn('_CreationDate', to_timestamp(Comments['_CreationDate'], 'yyyy-MM-dd HH:mm:ss'))
Comments = Comments.withColumn('_Score', Comments['_Score'].cast('int'))
Comments = Comments.withColumn('_Id', Comments['_Id'].cast('long'))
Comments = Comments.withColumn('_PostId', Comments['_PostId'].cast('long'))
Comments = Comments.withColumn('_UserId', Comments['_UserId'].cast('long'))
Comments = Comments.withColumnRenamed('_UserId', 'UserId')
```

Rysunek 11. Comments - konwersja typów

Przyglądając się dalej danym, widać że nie wszystkie kolumny będą użyteczne w dalszej analizie. Przykładowo kolumna „ContentLicense" nie będzie potrzebna. Usuwam ją więc za pomocą metody drop.

```
#usuwam kolumne
Comments = Comments.drop("_ContentLicense")
```

Rysunek 12. Usuwanie kolumny

Należy również upewnić się, czy zestaw danych przypadkiem nie zawiera powtarzających się rekordów. Wykorzystam gotową metodę dropDuplicates()

```
# Usuwanie duplikatów
Comments = Comments.dropDuplicates()
```

Rysunek 13. Usuwanie duplikatów

Białe znaki na początku i końcu wartości tekstowych w kolumnach również nie będą potrzebne. Za pomocą poniższej pętli można je z łatwością usunąć.

```
Comments = Comments.select(*(udf(lambda x: x.strip() if isinstance(x, str) else x, StringType())(c).alias(c) for c in Comments.columns))
```

Rysunek 14. Usuwanie białych znaków

Treść komentarzy zawiera również różne niechciane znaki. Nie są one wartościowe w dalszej analizie, więc importuję metodę regexp_replace z modułu "pyspark.sql.functions", który zawiera różne funkcje do manipulacji i transformacji danych w DataFrame. Zastosowanie tej funkcji powoduje usunięcie wszystkich tagów HTML z zawartości komentarzy w kolumnie "comment_text".

```
from pyspark.sql.functions import regexp_replace

Comments = Comments.withColumn("comment_text", regexp_replace(col("comment_text"), "<.*?>", ""))
```

Rysunek 15. Usuwanie znaczników HTML

Kolumnę z treścią komentarza również warto dalej przetworzyć. W tym celu zastosuję tokenizację tekstu - to proces podziału tekstu na mniejsze jednostki zwane tokenami. Tokeny mogą być pojedynczymi słowami, znakami interpunkcyjnymi, fragmentami zdania lub innymi ustalonymi jednostkami, które stanowią podstawowe elementy analizowanego tekstu.

Tokenizacja jest ważnym etapem w przetwarzaniu języka naturalnego i analizie tekstu. Podzielony tekst można dalej przetwarzać, analizować lub wykorzystać do budowy modeli predykcyjnych.
Importuję z biblioteki pyspark.ml.feature metodę tokenizer, która odpowiednio podzieli treść komentarzy.

```
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType
from pyspark.ml.feature import StopWordsRemover
from pyspark.ml.feature import Tokenizer
from nltk.stem.porter import *
```

```
# Oczyszczenie i tokenizacja tekstu
tokenizer = Tokenizer(inputCol="comment_text", outputCol="words")
Comments = tokenizer.transform(Comments)
```

Rysunek 16. Tokenizacja tekstu

Ostatnim krokiem będzie usunięcie z tekstu zbędnych słów, które nie mają wartości w analizie tekstu - stop words, przykładowo "a", "an", "the", "in", "on", "is", "are", "of", "to" itp. W dalszej analizie chcę skupić się na istotnych słowach kluczowych. Usuwanie stop words pomaga zmniejszyć rozmiar danych tekstowych, poprawia efektywność analizy tekstu i redukuje wpływ słów o niskiej wartości semantycznej na wyniki analizy. Importuję więc z biblioteki pyspark.ml.feature metodę StopWordsRemover, która usunie zbędne słowa.

```
remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
Comments = remover.transform(Comments)
```

Rysunek 17. Usuwanie stopwords

Oczyszczony DataFrame jest już gotowy do dalszej analizy. W tym celu zapisuję dataframe w formacie Parquet – jest to format kolumnowy, który zapewnia wysoką wydajność, kompresję danych oraz optymalizację operacji odczytu i zapisu. Parquet został stworzony jako projekt open source przez Apache Software Foundation i jest szeroko wykorzystywany w ekosystemie narzędzi Big Data, takich jak Apache Spark, Apache Hadoop, Apache Hive i inne.

```
Comments.write.mode("overwrite").parquet("s3://myproject/Comments.parquet")
```

Rysunek 18. Zapis w formacie parquet

Dane są już przechowywane w utworzonym wcześniej buckecie na S3.

Kolejny dataframe Posts zawiera 22 kolumny. Większość z nich posiada nieodpowiedni typ danych oraz zbędne znaki w nazwie. Czyszczenie przebiega w sposób analogiczny do pliku Comments. Tym razem kolumna „Body" zawierająca treść postu podlega tokenizacji oraz usunięciu stop words, aby w dalszych krokach była poddana analizie.

Z kolei plik PostHistory zawiera zdecydowanie mniej kolumn – pięć. Po podstawowym oczyszczeniu danych, kolumna „Comments" zawierająca komentarz użytkownika,który edytował post zostaje poddana tokenizacji i usunięciu zbędnych słów.

3.2. Reszta plików

Plik zawierający informację o odznakach nie wymaga zaawansowanego czyszczenia ze względu na mniejszą ilość kolumn. Usuwam więc duplikaty z danych oraz zmieniam nazwy kolumn na czytelniejsze, usuwając zbędne znaki.

```
Badges = Badges.dropDuplicates()
```

```
Badges = Badges.withColumnRenamed("_Id", "id") \
    .withColumnRenamed("_Name", "name") \
    .withColumnRenamed("_Date", "date") \
    .withColumnRenamed("_UserId", "user_id")\
    .withColumnRenamed("_Class", "class")\
    .withColumnRenamed("_TagBased", "TagBased")
```

Rysunek 19. Zmiana nazw kolumn

Typy kolumn również należy zmienić, ponieważ wszystkie posiadają typ string, który nie jest odpowiedni do przechowywania dat lub identyfikatorów użytkowników, które są liczbami całkowitymi.

```
from pyspark.sql.functions import to_timestamp

Badges = Badges.withColumn('date', to_timestamp(Badges['date'], 'yyyy-MM-dd HH:mm:ss'))
Badges = Badges.withColumn('id', Badges['id'].cast('long'))
Badges = Badges.withColumn('TagBased', Badges['TagBased'].cast('boolean'))
Badges = Badges.withColumn('user_id', Badges['user_id'].cast('long'))
Badges = Badges.withColumn('class', Badges['class'].cast('int'))
```

Rysunek 20. Zmiana typów kolumn

Dataframe Tags, Votes, Postlinks również zawierają tylko kilka kolumn, ich czyszczenie przebiega analogicznie jak dla danych z pliku Badges – czyli zmiana typów kolumn na odpowiednie oraz wyczyszczenie nazw.

Z pliku Users usuwam dodatkowo kolumnie WebsiteUrl, która nie będzie uwzględniana w dalszej analizie.

```
Users = Users.drop("WebsiteUrl")
```

Rysunek 21. Usuwanie kolumny

4. Dane w akcji: od czystych danych do przydatnych wyników.

W tym rozdziale skupimy się na analizie oczyszczonych danych. Przeprowadzę różnorodne analizy, które pomogą nam wyciągnąć przydatne wnioski i odkryć interesujące wzorce. Poniżej przedstawiam pytania, na których skoncentrowałam analizę:

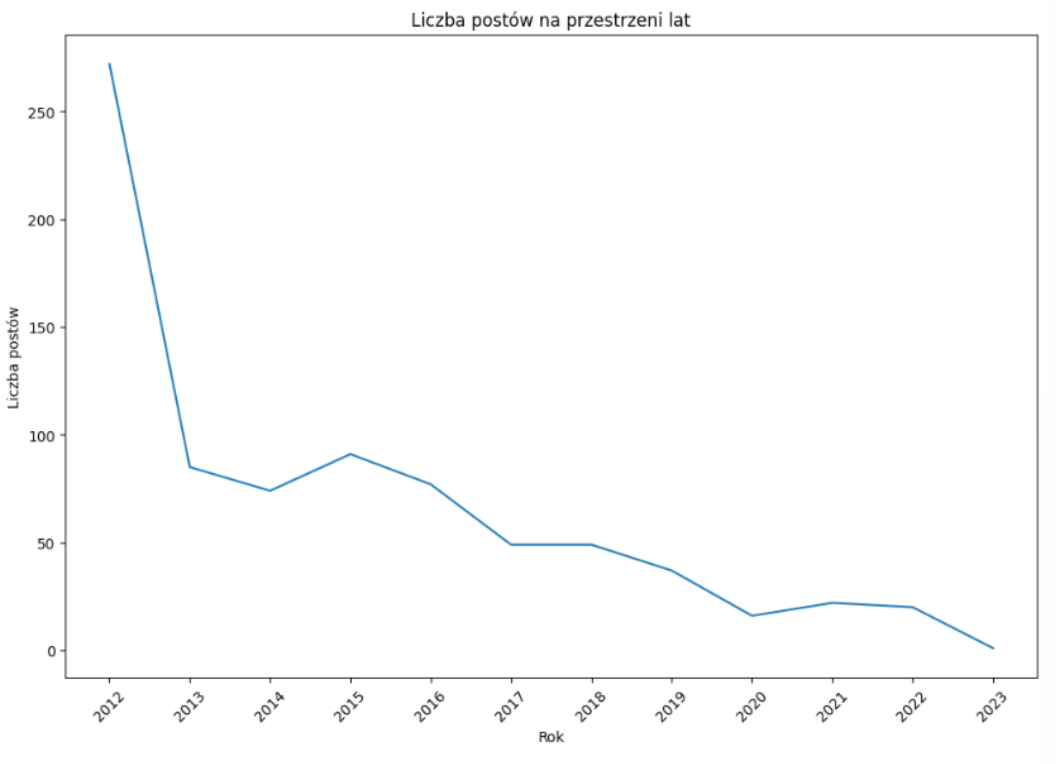
4.1. Pytania analityczne

- **Liczba postów na przestrzeni czasu.**

Rozważymy liczbę postów w zależności od czasu, aby zobaczyć, jak zmieniała się aktywność na forum w ciągu określonego okresu. Wykorzystamy wykresy typu lineplot i barplot, aby wizualnie przedstawić te zmiany. Przydatna będzie biblioteka matplotlib. Importuję więc z niej moduł pyplot który dostarcza interfejs podobny do funkcji dostępnych w MATLAB-ie. Umożliwia on łatwe tworzenie różnych typów wykresów, takich jak wykresy liniowe, słupkowe, punktowe, histogramy itp.

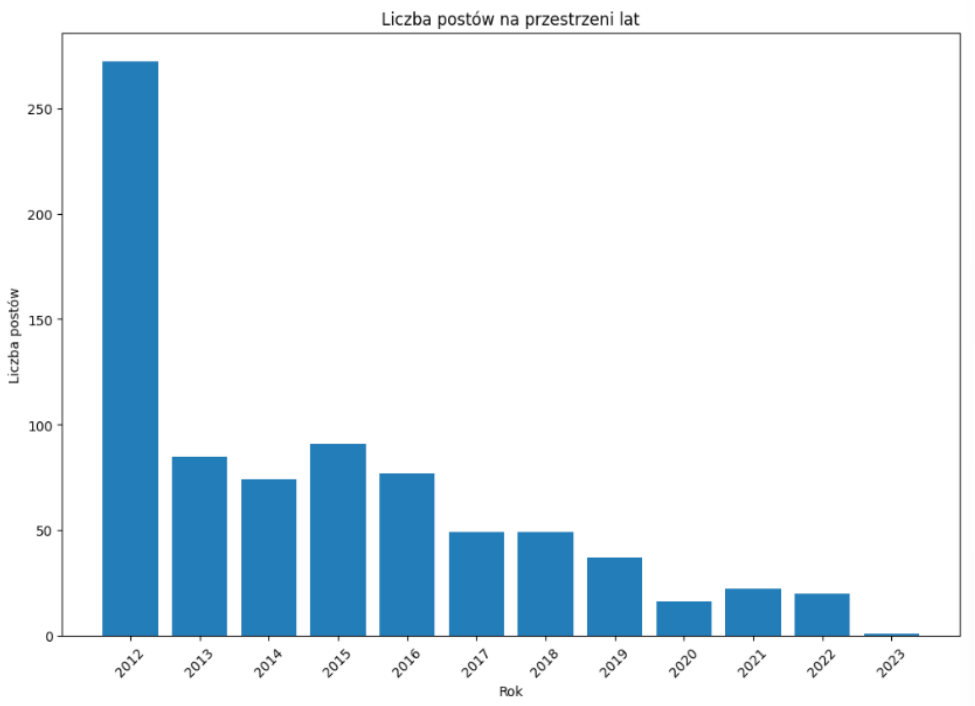
```
# Grupowanie postów na podstawie roku
post_count = Posts.groupBy(F.year('CreDate').alias('year')).count().orderBy('year')

# Konwersja do list
years = [str(row.year) for row in post_count.select('year').collect()]
counts = [row['count'] for row in post_count.select('count').collect()]

# Lineplot
plt.figure(figsize=(12, 8))
plt.plot(years, counts)
plt.title('Liczba postów na przestrzeni lat')
plt.xlabel('Rok')
plt.ylabel('Liczba postów')
plt.xticks(rotation=45)
plt.show()
```

Rysunek 22. Kod tworzący wykres typu Lineplot



Rysunek 23. Liczba postów na przestrzeni lat - wykres Lineplot

Forum osiągało szczytową liczbę postów w 2012 roku. Od tamtej pory liczba postów sukcesywnie zmniejsza się, chociaż wyjątkiem są lata 2014-2015, wtedy zauważalny jest niewielki wzrost. Im dalej przesuwamy się na wykresie w prawo, liczba zbliża się do zera, przy czym w 2023 roku posty nie są już praktycznie publikowane. Aby łatwiej odczytać dokładne wartości, możemy posłużyć się wykresem typu barplot.



Rysunek 24. Liczba postów na przestrzeni lat - wykres Barplot

Teraz zdecydowanie dokładniej możemy odczytać liczbę postów w poszczególnych latach. Jak widać w obecnym roku pojawiają się jeszcze nieliczne publikacje na forum. W porównaniu do roku 2012 jest to około pięćdziesięciokrotny spadek.

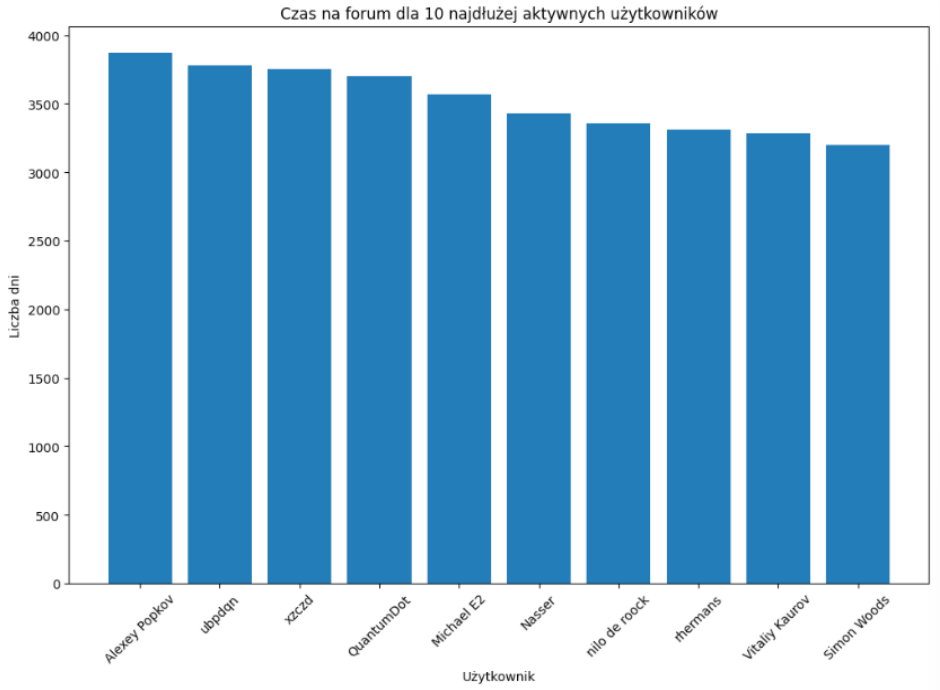
- **Czas na forum - 10 najdłużej aktywnych użytkowników.**

Skoncentrujemy się teraz na analizie czasu spędzonego na forum przez najbardziej aktywnych użytkowników. Wykluczymy boty i wybierzemy 10 najdłużej aktywnych użytkowników, aby zrozumieć ich zaangażowanie i wkład w społeczność. Za pomocą biblioteki pyspark.sql.functions importuję niezbędne funkcje jak max,min,datediff. Analizę rozpoczynam od wyeliminowania botów – są nimi użytkownicy o identyfikatorze -1. Aby obliczyć czas potrzebuję połączyć dataframe Posts oraz Users. Czas na forum wyliczam za pomocą daty utworzenia otatniego postu oraz daty utworzenia konta.

```python
from pyspark.sql.functions import max, min, datediff
import matplotlib.pyplot as plt

# Filtrowanie użytkowników, pomijając boty (przyjmując, że boty mają ID równą -1)
filtered_users = Users.filter(Users.Id != "-1")

# Obliczanie czasu od pojawienia się użytkownika do ostatniego posta/komentarza
user_activity = Posts.join(filtered_users, Posts.OwnerUserId == filtered_users.Id, "inner") \
    .groupBy(filtered_users.DisplayName) \
    .agg(max(Posts.CreDate).alias("last_activity"), min(filtered_users.CreationDate).alias("user_creation")) \
    .withColumn("time_on_forum", datediff("last_activity", "user_creation")) \
    .orderBy("time_on_forum", ascending=False) \
    .limit(10)

# Wyświetlanie wyników
user_activity.show()

# Przygotowanie danych do wykresu
user_names = [row.DisplayName for row in user_activity.select("DisplayName").collect()]
time_on_forum = [row.time_on_forum for row in user_activity.select("time_on_forum").collect()]
```

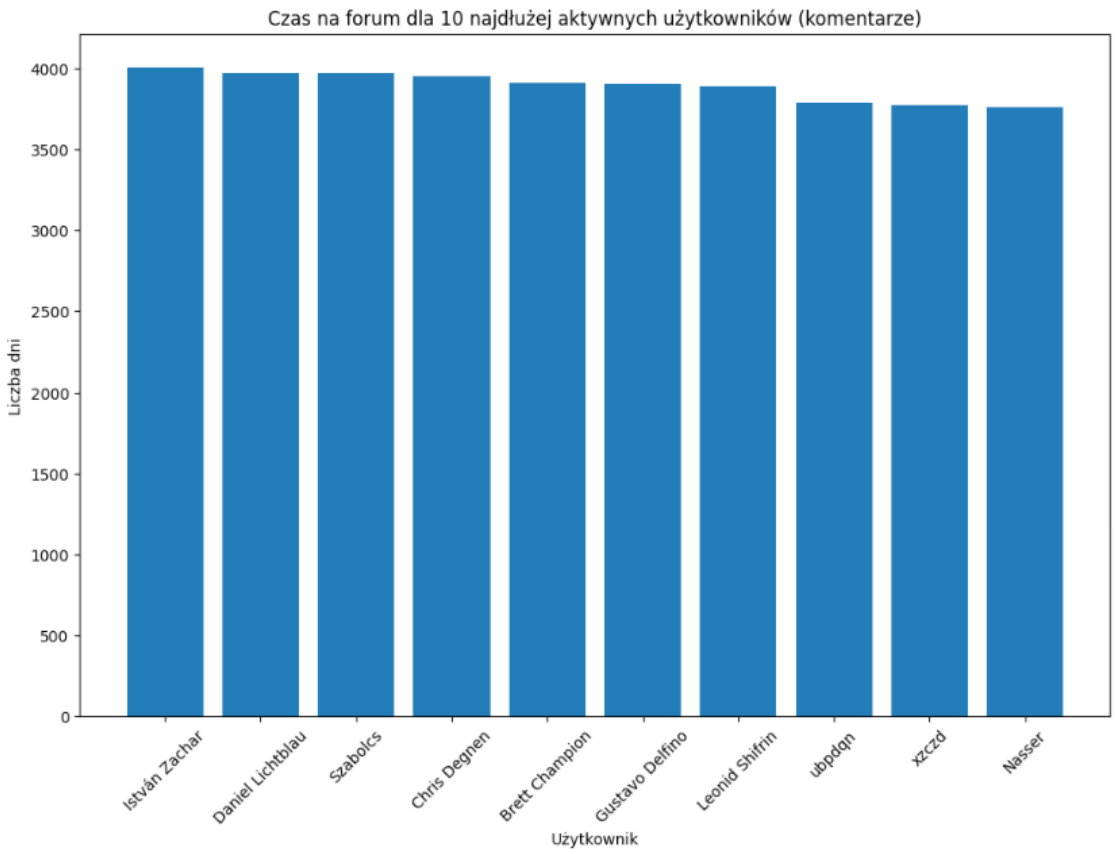
Rysunek 25. Generowanie top 10 najdłużej aktywnych użytkowników

Następnie generuję wykres barplot z otrzymanych wyników, który przedstawia 10 najdłużej aktywnych użytkowników. Na poziomej osi widoczne są nazwy top 10 użytkowników, a na pionowej czas podany w dniach.



Rysunek 26. Wykres top 10 najdłużej aktywnych użytkowników (posty)

Top 1 użytkownik jest aktywny od ponad dziesięciu lat. Spoglądając na wszystkich użytkowników, widać że ich czas znacznie od siebie nie odbiega. Średnio każdy z nich jest



Rysunek 27. Wykres top 10 najdłużej aktywnych użytkowników (komentarze)

aktywny od około 10 lat. Podobną analizę wykonuję biorąc pod uwagę datę najnowszego komentarza.

Uśredniając, każdy z top 10 użytkowników jest aktywny od około dziesięciu lat. Danie nieznacznie od siebie odbiegają, wyniki poszczególnych osób są do siebie bardzo zbliżone.

- **Porównanie najwyżej i najniżej ocenianych pytań.**

Przeanalizujemy najwyżej i najniżej oceniane pytania, zwracając uwagę na długość pytania, tagi, liczby odpowiedzi i inne czynniki, aby zidentyfikować różnice między tymi dwoma grupami.

```
from pyspark.sql.functions import col, desc

# Najwyżej ocenione pytania
#1 = Question
top_questions = Posts.filter(Posts.PostTypeId == 1).orderBy(desc("Score")).limit(1)
top_questions_info = top_questions.select("Title", "Body", "Tags", "AnswerCount")

# Najniżej ocenione pytania
bottom_questions = Posts.filter(Posts.PostTypeId == 1).orderBy("Score").limit(1)
bottom_questions_info = bottom_questions.select("Title", "Body", "Tags", "AnswerCount")
```

Rysunek 28. Wyznaczenie najwyżej i najniżej ocenianych pytań

Najpierw filtruję posty o identyfikatorze 1 – otrzymuję wtedy wyłącznie pytania. Po posortowaniu danych względem ich punktacji otrzymuję najwyżej i najniżej ocenione pytania. Najwyżej ocenione pytanie nosi tytuł:

„*Can I easily post images to this site directly from Mathematica?”*
Pytanie posiada wynik równy 3, a jego treść jest następująca:
„Graphics are tightly integrated into the Mathematica interface. The Front End is programmable, and Mathematica has functions to interface with the web, so the question naturally comes up: Could we make it possible to upload images to StackExchange directly from Mathematica, using a palette button, and without the need to first save them to disk?”.
Z treści pytania możemy wywnioskować, że funkcja którą proponuje użytkownik byłaby użyteczna w codziennej pracy, skróciłaby czas ładowania obrazów na forum. Społeczność StackExchange, skupiająca programistów, naukowców i entuzjastów technologii, jest zainteresowana narzędziami i wskazówkami dotyczącymi różnych języków programowania i narzędzi.
Tagi <support>, <faq> i <site-tools> przypisane do pytania są często wyszukiwane przez użytkowników, którzy szukają informacji i wsparcia w związku z funkcjonalnościami stron internetowych. Pytanie to zostało oznaczone tymi tagami, co przyczyniło się do większej widoczności i przyciągnęło uwagę użytkowników.

Przejdźmy do najniżej ocenioego pytania o tytule: *„Where is the record of the agreement referred to here?”* Pytanie otrzymało wynik 1, a jego treść jest następująca:

"don't use the platform name as tag unless you are sure the platform plays a role

EDIT: If you own different platforms you can do the experimentation wrt system specificity yourself; if not, wait until community members have checked this"
https://archive.is/GkVrc#selection-1219.5-1219.86
nofollow">https://archive.is/GkVrc#selection-1219.5-1219.86"
This was community consensus, agreed upon
https://archive.is/GkVrc#selection-1789.36-1789.77
https://archive.is/GkVrc#selection-1789.36-1789.77
Link(s) please. In my memory need not apply."

Pierwsze co rzuca się w oczy to niejasność pytania: Treść pytania jest trudna do zrozumienia. Może to spowodować dezorientację czytelników i utrudniać udzielenie odpowiedzi. Nie jest jasne, o co dokładnie pyta autor, co może wpływać na brak odpowiedzi lub niską ocenę. Pytanie dotyczy konkretnej umowy lub porozumienia, ale nie podaje wystarczająco dużo informacji, aby czytelnicy mogli zrozumieć, o jakie porozumienie chodzi. Brak konkretnych szczegółów może zniechęcać do odpowiadania na pytanie, gdyż brakuje wystarczających podstaw do udzielenia informacji czy wskazówek. Temat pytania oraz tag <discussion> sugerują, że pytanie dotyczy dyskusji lub rozmowy na temat pewnych umów lub porozumień. To może być temat mniej interesujący dla użytkowników lub niezwiązany z ich głównym obszarem zainteresowań. W rezultacie pytanie może nie przyciągnąć dużo uwagi i uzyskać niską ocenę.

- **Procent przypadków, kiedy najwyżej oceniana odpowiedź to niezaakceptowana odpowiedź.**

Sprawdzimy, jaki procent przypadków dotyczy sytuacji, gdy najwyżej oceniona odpowiedź nie jest zaakceptowana przez autora pytania. Przeanalizujemy, czy istnieje zależność między oceną odpowiedzi a jej akceptacją przez autora pytania.

Aby uzyskać wynik, filtruję posty o identyfikatorze 2 – otrzymam wówczas wyłącznie odpowiedzi.

```
from pyspark.sql.functions import col, when, sum

# Filtrowanie odpowiednich postów
filtered_posts = Posts.filter((col("PostTypeId") == 2) & (col("AcceptedAnswerId").isNull()))

# Obliczanie procentu
total_posts = filtered_posts.count()
not_accepted_posts = filtered_posts.filter(col("Score") == col("AnswerCount")).count()
percentage = (not_accepted_posts / total_posts) * 100
percentage = round(percentage, 10)


print(f"Procent przypadków, kiedy najwyżej oceniana odpowiedź nie jest zaakceptowaną odpowiedzią: {percentage}%")
```

```
Procent przypadków, kiedy najwyżej oceniana odpowiedź nie jest zaakceptowaną odpowiedzią: 0.0%
```

Rysunek 29. Wyliczenie procentu najwyżej nieocenianych niezaakceptowanych odpowiedzi

Otrzymujemy wynik równy 0%. Oznacza to, że nie pojawiły się na forum takie przypadki. Możemy wnioskować, że najwyżej oceniane odpowiedzi zawsze zostały zaakceptowane przez autora pytania jako najbardziej pomocne .

- **Rozkład ocen odpowiedzi zaakceptowanych vs pozostałych.**

Przeanalizujemy rozkład ocen odpowiedzi, które zostały zaakceptowane przez autora pytania, w porównaniu do pozostałych odpowiedzi. Skupimy się na statystykach, takich jak średnia, odchylenie standardowe, wartość minimalna i maksymalna, aby lepiej zrozumieć jakość i oceny odpowiedzi.

Najpierw importuję z biblioteki pyspark.sql.functions funkcje takie jak avg, stddev, min, max aby wykorzystać je do niezbędnych obliczeń. Następnie filtruję odpowiedzi po ich typach.

```
from pyspark.sql.functions import avg, stddev, min, max

# Filtruj odpowiedzi zaakceptowane
zaakceptowane_odpowiedzi = Posts.filter(Posts.PostTypeId == 2).filter(Posts.AcceptedAnswerId.isNotNull())

# Filtruj pozostałe odpowiedzi
pozostale_odpowiedzi = Posts.filter(Posts.PostTypeId == 2).filter(Posts.AcceptedAnswerId.isNull())

# Oblicz rozkład ocen odpowiedzi zaakceptowanych
zaakceptowane_odpowiedzi_stats = zaakceptowane_odpowiedzi.select(avg('Score').alias('avg_score'), stddev('Score').alias('score_stddev'), min('Score').alias('score_min'), max('Score').alias

# Oblicz rozkład ocen pozostałych odpowiedzi
pozostale_odpowiedzi_stats = pozostale_odpowiedzi.select(avg('Score').alias('avg_score'), stddev('Score').alias('score_stddev'), min('Score').alias('score_min'), max('Score').alias('score_

# Wyświetl wyniki
print('Rozkład ocen odpowiedzi zaakceptowanych:')
zaakceptowane_odpowiedzi_stats.show()

print('Rozkład ocen pozostałych odpowiedzi:')
pozostale_odpowiedzi_stats.show()
```

Rysunek 30. Wyliczenie rozkładu ocen odpowiedzi zaakceptowanych vs pozostałych.

W zbiorze wyróżniamy następujące typy odpowiedzi:

1. Pytanie - PostTypeId = 1: Odpowiedzi na pytania użytkowników.

2. Zaakceptowana odpowiedź - PostTypeId = 2: Odpowiedzi, które zostały zaakceptowane przez autora pytania jako najbardziej pomocne.

3.Odpowiedź z komentarzem - PostTypeId = 3: Odpowiedzi w formie komentarza do pytania lub innej odpowiedzi.

4. Edycja - PostTypeId = 4: Edycje wcześniejszych postów, takich jak pytania lub odpowiedzi.

5. Wsparcie dla odpowiedzi - PostTypeId = 5: Odpowiedzi, które zawierają dodatkowe informacje lub wsparcie dla innych odpowiedzi.

6. Oznaczona odpowiedź - PostTypeId = 6: Odpowiedzi oznaczone przez moderatorów lub społeczność jako pomocne lub prawidłowe.

7. Wiki odpowiedź - PostTypeId = 7: Odpowiedzi, które są częścią wspólnej edycji wiki.

```
Rozkład ocen odpowiedzi zaakceptowanych:
+---------+------------+---------+---------+
|avg_score|score_stddev|score_min|score_max|
+---------+------------+---------+---------+
|     null|        null|     null|     null|
+---------+------------+---------+---------+

Rozkład ocen pozostałych odpowiedzi:
+-----------------+-----------------+---------+---------+
|        avg_score|     score_stddev|score_min|score_max|
+-----------------+-----------------+---------+---------+
|7.455465587044534|6.334893266952607|       -5|       33|
+-----------------+-----------------+---------+---------+
```

Rysunek 31. Rozkład ocen odpowiedzi

Wyniki wskazują, że brakuje danych dotyczących rozkładu ocen zaakceptowanych odpowiedzi. Może to wynikać z braku odpowiedzi zaakceptowanych przez autora pytania jako najbardziej pomocne. Sprawdźmy więc to za pomocą poniższego kodu, zliczając odpowiedzi o szukanym identyfikatorze 2.

```
from pyspark.sql.functions import col

# Sprawdzenie, czy istnieją odpowiedzi zaakceptowane
has_accepted_answers = Posts.filter(col("PostTypeId") == 2).filter(col("AcceptedAnswerId").isNotNull()).count() > 0

if has_accepted_answers:
    print("W DataFrame istnieją odpowiedzi zaakceptowane.")
else:
    print("W DataFrame nie ma odpowiedzi zaakceptowanych.")
```

```
W DataFrame nie ma odpowiedzi zaakceptowanych.
```

Rysunek 32. Test na istnienie zaakceptowanych odpowiedzi

Jak widać w zbiorze nie istnieją takie odpowiedzi. Przejdźmy zatem do pozostałych odpowiedzi, dostępne są dane dotyczące rozkładu ocen. Średnia ocen wynosi 7.46, co sugeruje, że w ogólności odpowiedzi te otrzymują relatywnie pozytywne oceny. Odchylenie standardowe ocen wynoszące 6.33 wskazuje na pewne zróżnicowanie ocen, ale ogólnie rzecz biorąc oceny mieszczą się w umiarkowanym zakresie. Najniższa ocena wynosi -5, co oznacza, że są również odpowiedzi o negatywnej ocenie. Najwyższa ocena wynosi 33, co sugeruje, że niektóre odpowiedzi zdobyły wysokie uznanie społeczności.

- **Top N tagów, które wygenerowały najwięcej wyświetleń.**

Zidentyfikujemy najpopularniejsze tagi, które przyciągają największą liczbę wyświetleń. To pozwoli nam zrozumieć, jakie tematy są najbardziej interesujące dla społeczności matematycznej na forum. Przyjmijmy N=3.

```
tags_sum = Tags.groupBy("TagName").agg(sum("count").alias("TotalCount"))

# Sortujemy malejąco po sumie wartości count
sorted_tags = tags_sum.orderBy("TotalCount", ascending=False)

top_10_tags = sorted_tags.limit(3)

top_10_tags.show()
```

```
+---------------+----------+
|        TagName|TotalCount|
+---------------+----------+
|     discussion|       713|
|        support|       163|
|feature-request|       102|
+---------------+----------+
```

Rysunek 33. Top 3 tagi, które wygenerowały najwięcej wyświetleń

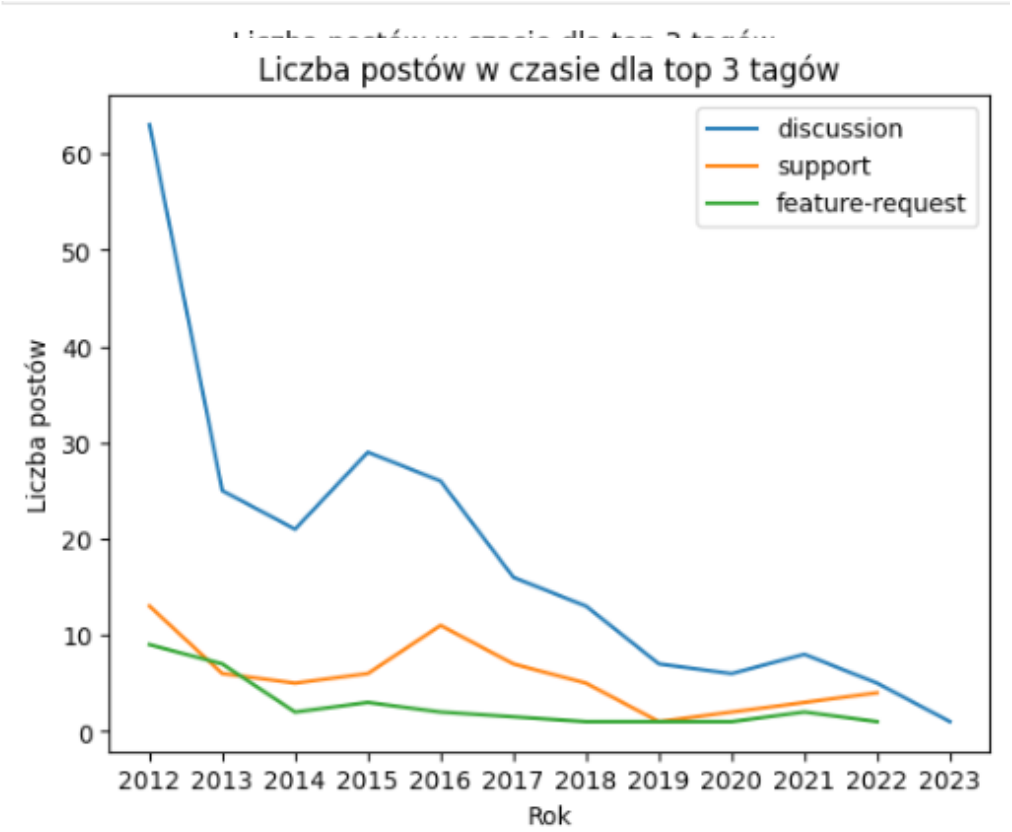
Najwyższą liczbę wyświetleń otrzymał tag "discussion" - aż 713. Druga najwyższa liczba wyświetleń należy do tagu "support" - 163. Trzecia najwyższa liczba wyświetleń to tag "feature-request" – 102. Te trzy tagi są najbardziej popularne i generują najwięcej

wyświetleń na platformie. Oznacza to, że użytkownicy są szczególnie zainteresowani tematami związanymi z "discussion" (dyskusje), "support" (wsparcie) i "feature-request" (żądanie funkcji), przy czym najchętniej wyświetlany tag posiada czterokrotnie więcej wyświetleń od kolejnego w rankingu i siedmiokrotnie więcej od ostatniego.

- **Liczba postów w czasie dla każdego z top 3 tagów.**

Przeanalizujemy liczbę postów dla każdego z najpopularniejszych tagów, aby zobaczyć, jak ich aktywność rozwija się w czasie. Wykorzystamy wykres liniowy i słupkowy, aby zobrazować trendy w zależności od tagu.

Aby utworzyć wykresy skorzystam z biblioteki matplotlib. Zliczam liczbę postów dla każdego z tagów grupując dane po roku. Wówczas otrzymujemy następujące wykresy:



Rysunek 35. Liczba postów w czasie dla top 3 tagów - wykres liniowy

Na wykresach również widoczna jest spora rozbieżność między pierwszym a drugim i trzecim miejscem. Ponadto, w aktualnym roku tag support i feature-request nie posiada już

żadnych wyświetleń w przeciwieństwie do tagu discussion. Wszystkie tagi miały największą liczbę wyświetleń w 2012 roku, przy czym ich ilość sukcesywnie spada z wyjątkiem niewielkiego skoku w 2015 oraz 2021 roku.

- **Najczęściej pojawiające się słowa w tytułach.**

Zbadamy najczęściej występujące słowa w tytułach postów, wykluczając słowa powszechne (stopwords). To pozwoli nam zidentyfikować tematykę i słowa kluczowe często wykorzystywane w tytułach.

W tym celu wykorzystam bibliotekę pyspark.ml.feature, a z niej metodę StopWordsRemover oraz z biblioteki pyspark.sql.functions importuję metody explode, split, lower, array.

```
df_title_words = Posts.select(explode(split(lower("Title"), "\\W+")).alias("word"))

# Przekształcenie tytułów na format array<string>
df_title_words = df_title_words.select(array(df_title_words["word"]).alias("title_words"))

remover = StopWordsRemover(inputCol="title_words", outputCol="filtered_words", stopWords=stopwords)
df_filtered_words = remover.transform(df_title_words).select("filtered_words")

word_counts = df_filtered_words.select(explode("filtered_words").alias("filtered_word")) \
    .groupBy("filtered_word").count().orderBy("count", ascending=False)
```

Rysunek 36. Wyznaczenie najczęściej pojawiających się słów w tytułach

Zaczynamy od podzielenia tytułów na odrębne słowa wykorzytując przy tym wyrażenia regularne. Zatem "\W+" w tym kontekście oznacza, że funkcja split() będzie dzielić zawartość kolumny "Title" na słowa, używając jako separatora dowolnego znaku niebędącego literą lub cyfrą, a słowa te będą składać się z jednego lub więcej znaków alfanumerycznych. Następnie metoda StopWordsRemover usunie niepotrzebne słowa, a wyniki wrzuci do nowej kolumny „filtered_words”. Ponadto, poprzez zastosowanie funkcji explode(), każde słowo z zbioru zostanie rozbite na oddzielne wiersze, co pozwala na przetwarzanie tych słów jako odrębnych rekordów bez powtórzeń.

filtered_word	count
mathematica	42
questions	38
question	34
tag	28
se	20
answer	18
community	17
code	16
new	15
site	13
get	12
stackexchange	10
answers	10
comment	9
posts	8
ask	8
tags	8
users	8
wolfram	7
post	7

Rysunek 37. Najpopularniejsze słowa w tytułach

Na podstawie wyników przedstawionych w tabeli, można wyciągnąć następujące wnioski - najczęściej występujące słowo "mathematica" pojawia się w tytułach aż 42 razy. Drugie i trzecie najczęściej występujące słowo "questions" oraz "question" pojawia się średnio 36 razy. Ponadto, lista najczęściej pojawiających się słów obejmuje również inne popularne terminy związane z platformą Stack Exchange, takie jak "tag", "se" (skrót od Stack Exchange), "answer", "community" oraz słowa ogólnie związane z zadawaniem pytań i udzielaniem odpowiedzi. Słowa te mogą stanowić kluczowe tematy i zagadnienia, które użytkownicy najczęściej poruszają i poszukują na stronie.

- **Procent użytkowników, którzy nigdy nic nie zapostowali.**

Skoncentrujemy się na analizie udziału użytkowników, którzy zarejestrowali się na forum, ale nigdy nie napisali żadnego postu. Przeanalizujemy ten odsetek, aby zrozumieć dynamikę społeczności i zaangażowanie użytkowników.

```
from pyspark.sql.functions import col

# Liczba użytkowników
liczba_uzytkownikow = Users.select('Id').distinct().count()

# Liczba użytkowników, którzy mają jakiekolwiek posty
liczba_uzytkownikow_z_postami = Posts.select('OwnerUserId').distinct().count()

# Obliczanie procenta użytkowników bez postów
procent_bez_postow = (liczba_uzytkownikow - liczba_uzytkownikow_z_postami) / liczba_uzytkownikow * 100

# Wyświetlanie wyniku
print("Procent użytkowników, którzy nigdy nic nie zapostowali:", procent_bez_postow)
```

```
Procent użytkowników, którzy nigdy nic nie zapostowali: 97.48668056520732
```

Rysunek 38. Procent użytkowników, którzy nigdy nic nie zapostowali

Na podstawie przedstawionego kodu i wyniku, można wyciągnąć następujące wnioski:

- Procent użytkowników, którzy nigdy nic nie zapostowali, wynosi około 97.49%.
- Tylko niewielki odsetek użytkowników (około 2.51%) ma jakiekolwiek posty.

To sugeruje, że większość użytkowników nie uczestniczy aktywnie w zamieszczaniu postów na platformie. Mogą być różne przyczyny takiego zjawiska, takie jak brak potrzeby udostępniania treści, rola użytkowników jako pasywnych odbiorców lub ograniczenia czasowe.

- **Średni czas od pojawienia się pytania do pojawienia się zaakceptowanej odpowiedzi.**

Przeanalizujemy średni czas, jaki upływa od momentu zadania pytania przez użytkownika do pojawienia się zaakceptowanej odpowiedzi. To pozwoli nam zrozumieć, jak szybko użytkownicy otrzymują dobre odpowiedzi na swoje pytania.

```
from pyspark.sql.functions import col, avg, round

# Filtrujemy tylko pytania (PostTypeId = 1) i rekordy z zaakceptowanymi odpowiedziami
questions_with_accepted_answers = Posts.filter((col("PostTypeId") == 1) & col("AcceptedAnswerId").isNotNull())

# Obliczamy różnicę czasu między pojawieniem się pytania a pojawieniem się zaakceptowanej odpowiedzi w sekundach
time_difference_seconds = (col("LastActivityDate").cast("long") - col("CreDate").cast("long"))

# Obliczamy średnią wartość czasu w dniach zaokrągloną do dwóch miejsc po przecinku
#dzieląc przez 86400 (liczba sekund w jednym dniu).
average_time_days = questions_with_accepted_answers.select(round(avg(time_difference_seconds) / 86400, 2).alias("AverageTimeInDays"))

# Wyświetlamy wynik
average_time_days.show(truncate=False)
```

```
+-----------------+
|AverageTimeInDays|
+-----------------+
|156.21           |
+-----------------+
```

Rysunek 39. Średni czas od pojawienia się pytania do pojawienia się zaakceptowanej odpowiedzi

Na podstawie powyższego wyniku, można wyciągnąć następujące wnioski - średnio upływa 156 dni od zadania pytania do momentu, kiedy pojawia się zaakceptowana odpowiedź. Na platformie potrzeba czasu na uzyskanie odpowiedzi, które zostaną zaakceptowane przez pytającego. Może to wynikać z różnych czynników, takich jak trudność pytania, dostępność ekspertów w danej dziedzinie, ilość aktywnych użytkowników, itp.

- **Średnia liczba komentarzy na post w top 10 tagach.**

Przeanalizujemy średnią liczbę komentarzy przypisanych do postów w dziesięciu najpopularniejszych tagach. To pozwoli nam zrozumieć, jakie tematy przyciągają najwięcej uwagi i dyskusji ze strony społeczności.

W tym celu importuję z biblioteki pyspark.sql.functions metody explode, col, regex_replace. Tagi w pliku są połączone, a my chcemy analizować każdy odrębnie. W tym celu wykonuję poniższy kod:

exploded_tags = Posts.withColumn("Tag", explode(split(Posts.Tags, "><")))

Teraz, gdy tagi są wyodrębnione mogę obliczyć średnią liczbę komentarzy.

Tag	AverageComments
voting	14.00
moderation	10.00
vote-to-close	10.00
packages	8.50
questions	8.50
policy	8.50
reputation	8.00
close-reasons	8.00
close-reasons	7.67
down-votes	7.00

Rysunek 40. Średnia liczba komentarzy na post w top 10 tagach

Tagi "voting", "moderation" i "vote-to-close" mają najwyższą średnią liczbę komentarzy na post, wynoszącą odpowiednio 14, 10 i 10. Tagi "packages", "questions" i "policy" zajmują kolejne miejsca, z średnią liczbą komentarzy wynoszącą 9 dla każdego z tych tagów. Pozostałe tagi również mają relatywnie wysoką średnią liczbę komentarzy, oscylującą w zakresie od 8 do 7. Pytania i dyskusje dotyczące zagadnień związanych z głosowaniem, moderacją, pakietami, pytaniami ogólnymi i polityką przyciągają większą uwagę społeczności i generują więcej komentarzy. Wysoka liczba komentarzy może wskazywać na aktywną interakcję użytkowników w tych tematycznych obszarach i angażowanie się w dyskusje, pytania lub wyrażanie swoich opinii.

- **Najpopularniejsze tagi wśród użytkowników o najwyższej reputacji.**

Zbadamy, jaki tag jest najczęściej używany przez użytkowników o najwyższej reputacji. Pozwoli nam to zidentyfikować obszary zainteresowań i specjalizacji najbardziej szanowanych członków społeczności.

W tym calu należy odfiltrować puste tagi oraz wybieramy użytkowników z najwyższą liczbą punktów reputacji.

```
from pyspark.sql.functions import col, split, explode, desc

highest_reputation_users = Users.filter(col("Reputation").cast("int") >= 100000)
highest_reputation_posts = highest_reputation_users.join(Posts, highest_reputation_users.Id == Posts.OwnerUserId, "inner")
highest_reputation_tags = highest_reputation_posts.select(explode(split(col("Tags"), "[><]")).alias("Tag"))
#odfiltrujemy puste tagi
highest_reputation_tags = highest_reputation_tags.filter(col("Tag") != "")
most_common_tags = highest_reputation_tags.groupBy("Tag").count().orderBy(desc("count"))
most_common_tags.show(10)
```

```
+-----------------+-----+
|              Tag|count|
+-----------------+-----+
|       discussion|   64|
|          tagging|   17|
|          support|   15|
| status-completed|   12|
|  feature-request|    8|
|           design|    5|
|       site-tools|    4|
|    vote-to-close|    3|
|             blog|    3|
|site-organization|    2|
+-----------------+-----+
only showing top 10 rows
```

Rysunek 41. Najpopularniejsze tagi wśród użytkowników o najwyższej reputacji

Wśród nich najpopularniejszym tagiem jest "discussion" z liczbą wystąpień wynoszącą 64. Tagi "tagging" i "support" zajmują kolejne miejsca, z liczbą wystąpień odpowiednio 17 i 15. Pozostałe tagi, takie jak "status-completed", "feature-request", "design", "site-tools", "vote-to-close", "blog" i "site-organization", również występują, ale z mniejszą liczbą wystąpień. Tematy związane z dyskusjami są szczególnie popularne wśród użytkowników o wysokiej reputacji, co wskazuje na ich aktywność w prowadzeniu rozmów i dyskusji na platformie.

Tagi takie jak "tagging" i "support" również odgrywają istotną rolę, ale z mniejszą częstotliwością. Może to być przydatna wskazówka dla innych użytkowników, którzy szukają informacji lub chcą angażować się w rozmowy z użytkownikami o wysokiej reputacji.

- **Liczba głosów oddanych w ostatnich dwóch latach.**

Przeanalizujemy liczbę głosów (pozytywnych i negatywnych) oddanych w poszczególnych miesiącach. To pozwoli nam zobaczyć, czy istnieją sezonowe trendy lub zmiany w aktywności społeczności.

```
from pyspark.sql.functions import month, year, desc

votes_by_month = Votes.withColumn('Year', year('CreationDate')) \
    .withColumn('Month', month('CreationDate')) \
    .groupBy('Year', 'Month') \
    .count() \
    .orderBy(desc('Year'), desc('Month'))

votes_by_month.show()
```

```
+----+-----+-----+
|Year|Month|count|
+----+-----+-----+
|2023|    2|   19|
|2023|    1|   34|
|2022|   12|   30|
|2022|   11|   57|
|2022|   10|   12|
|2022|    9|   48|
|2022|    8|   15|
|2022|    7|    4|
|2022|    6|   36|
|2022|    5|   68|
|2022|    4|   43|
|2022|    3|   44|
|2022|    2|   19|
|2022|    1|   18|
|2021|   12|   38|
|2021|   11|   11|
|2021|   10|   43|
|2021|    9|   19|
|2021|    8|   30|
|2021|    7|   86|
+----+-----+-----+
```

Rysunek 42. Liczba głosów oddanych w ostatnich dwóch latach

Przyjrzyjmy się największej i najmniejszej liczbie głosów w poszczególnych miesiącach:

- Największa liczba głosów (86) została odnotowana w lipcu 2021 roku.
- Kolejne miesiące z wysoką liczbą głosów to listopad 2022 (57 głosów) oraz maj 2022 (68 głosów).
- Najmniejsza liczba głosów (4) została odnotowana w lipcu 2022 roku.
- W sierpniu 2022 roku odnotowano 15 głosów, co również jest jednym z najniższych wyników.

Na podstawie tych danych można wywnioskować pewną sezonowość w aktywności głosowania na platformie. Wydaje się, że miesiące letnie (lipiec i sierpień) mają tendencję

do wykazywania niższej aktywności głosowania. Natomiast wiosna (maj) i jesień (listopad) są okresami, w których odnotowuje się wyższą aktywność głosowania.

Możliwe przyczyny sezonowości mogą być różne. Na przykład w okresie letnim użytkownicy mogą być bardziej skłonni do spędzania czasu na aktywnościach na świeżym powietrzu, co może prowadzić do mniejszego zaangażowania na platformie. Natomiast wiosną i jesienią, kiedy warunki atmosferyczne są bardziej korzystne dla pobytu wewnątrz, użytkownicy mogą być bardziej aktywni na platformie.

- **Średnia długość treści posta w zależności od typu postu.**

Skupimy się na analizie średniej długości treści posta w zależności od typu postu, takiego jak pytania, odpowiedzi, komentarze itp. To pozwoli nam zrozumieć, jakie typy postów są bardziej rozbudowane i wymagające.

```
from pyspark.sql.functions import avg, length, round, when, lit
from pyspark.sql.types import StringType

# Tworzenie kolumny z nazwą typu postu
Posts = Posts.withColumn('PostTypeName', when(Posts.PostTypeId == 1, lit('Question')).when(Posts.PostTypeId == 2, lit('Answer')).otherwise(lit('Inny')))

# Grupowanie danych i obliczanie średniej długości treści posta dla każdego typu postu
avg_length_by_post_type = Posts.groupBy('PostTypeId', 'PostTypeName').agg(round(avg(length('Body')), 2).cast(StringType()).alias('avg_length'))

# Wyświetlanie wyników
avg_length_by_post_type.show()
```

```
+----------+------------+----------+
|PostTypeId|PostTypeName|avg_length|
+----------+------------+----------+
|         1|    Question|   1044.58|
|         5|        Inny|     852.0|
|         4|        Inny|      93.6|
|         2|      Answer|   1078.18|
+----------+------------+----------+
```

Rysunek 43. Średnia długość treści posta w zależności od typu postu

Na podstawie wyników analizy możemy wyciągnąć następujące wnioski:

1. Średnia długość treści posta dla typu "Question" wynosi 1045 znaków.
2. Średnia długość treści posta dla typu "Answer" wynosi 1078 znaków.
3. Typy postów o PostTypeId 4 i 5, które nie są ani "Question" ani "Answer", mają średnią długość odpowiednio 94 i 852 znaków.

Wnioskiem jest, że średnia długość treści posta różni się w zależności od typu postu. Pytania (typ "Question") mają nieco krótszą średnią długość niż odpowiedzi (typ "Answer"), co może wynikać z różnicy w zawartości i charakterze tych postów.

Inne typy postów (typy 4 i 5) mają znacznie różne długości treści, co sugeruje, że mogą one mieć inną strukturę lub przeznaczenie niż pytania i odpowiedzi.

- **Najczęściej występujące słowa w sekcji "About Me" użytkowników.**

Przeanalizujemy najczęściej występujące słowa w sekcji "O mnie" użytkowników. To pozwoli nam lepiej poznać zainteresowania, opisy i preferencje członków społeczności. Tym razem importuję bbliotekę BeautifulSoup - służy do parsowania i ekstrakcji danych z dokumentów HTML i XML. Pozwala na przekształcanie surowego kodu HTML/XML na strukturę drzewa, która jest łatwa do nawigacji i przeszukiwania.

Tworzę więc funkcję, która podzieli sekcję „O mnie” na pojedyncze słowa oraz odrzuci te bez znaczenia w analizie oraz zliczam najczęściej występujące słowa.

```
def tokenize_text(text):
    if text is not None:
        # Usunięcie znaczników HTML
        soup = BeautifulSoup(text, 'html.parser')
        text = soup.get_text(separator=' ')

        # Podział tekstu na słowa
        words = re.split(r'\W+', text.lower())

        # Usunięcie stopwords
        words = [word for word in words if word not in stopwords]

        return words
    else:
        return []
```

Rysunek 44. Funkcja do podziału tekstu na słowa z uwzględnieniem znaczników HTML

word	count
mathematica	150
stack	146
physics	118
stackexchange	111
mathematics	109
student	98
like	95
exchange	92
software	90
also	86

Rysunek 45. Najczęściej występujące słowa w sekcji "About Me"

Słowo "mathematica" występuje najczęściej w sekcji "About Me" użytkowników, z liczbą wystąpień wynoszącą 150. Witryna dotyczy oprogramowania Mathematica stąd te słowo jest najpopularniejsze. Słowo "stack" zajmuje drugie miejsce pod względem częstotliwości występowania, z liczbą wystąpień wynoszącą 146. Użytkownicy są związani z platformą Stack Exchange. Słowo "physics" pojawia się również często, z liczbą wystąpień wynoszącą 118. Może to wskazywać na zainteresowanie użytkowników fizyką lub ich powiązanie z dziedziną naukową. Słowo "stackexchange" pojawia się 111 razy, co potwierdza wcześniejszą sugestię, że użytkownicy są związani z platformą Stack Exchange. Inne często występujące słowa to "mathematics", "student", "like", "exchange" i "software". Wskazują one na różne zainteresowania i dziedziny użytkowników.

Wnioskiem ogólnym jest to, że użytkownicy często w sekcji "About Me" wspominają o swoich zainteresowaniach naukowych, matematyce, korzystaniu z platformy Stack Exchange oraz oprogramowaniu.

4.2. Uczenie maszynowe

Uczenie maszynowe pozwoli nam wykorzystać zaawansowane algorytmy i modele do analizy danych zebranych z platformy „mathematica.meta.stackexchange.com”. Poprzez proces uczenia, będziemy w stanie nauczyć komputer rozpoznawać wzorce, związki między danymi oraz podejmować predykcje na podstawie zebranych informacji.

- **Czy istnieje zależność między liczbą odpowiedzi na post a jego oceną?**

Zbadamy, czy istnieje zależność między liczbą odpowiedzi na post a jego oceną. Przeanalizujemy korelację między tymi dwoma czynnikami, aby zrozumieć, czy większa liczba odpowiedzi ma wpływ na ocenę posta. Skorzystam więc z metody corr() dostępnej w Spark DataFrame. Korelacja osiąga wynik w zakresie od -1 do 1. Wartość bliska 1 wskazuje na silną dodatnią korelację, wartość bliska -1 wskazuje na silną

ujemną korelację, a wartość bliska 0 wskazuje na brak lub bardzo słabą korelację między liczbą odpowiedzi a oceną postu.

```
from pyspark.sql.functions import col

# Wybieramy potrzebne kolumny z DataFrame 'Posts'
posts = Posts.select("AnswerCount", "Score")

# Przekształcamy wartości kolumn na typ numeryczny
posts = posts.withColumn("AnswerCount", col("AnswerCount").cast("double"))
posts = posts.withColumn("Score", col("Score").cast("double"))

# Obliczamy korelację między 'AnswerCount' a 'Score'
correlation = posts.stat.corr("AnswerCount", "Score")

print("Korelacja między liczbą odpowiedzi a oceną postu: ", correlation)
```

```
Korelacja między liczbą odpowiedzi a oceną postu:  0.20300741889452753
```

Rysunek 46. Zależność między liczbą odpowiedzi na post a jego oceną

Wynik analizy korelacji między liczbą odpowiedzi na post a jego oceną wynosi 0.203. Ta wartość wskazuje na słabą pozytywną korelację między tymi dwiema cechami. Oznacza to, że istnieje pewne, choć niewielkie, powiązanie między liczbą odpowiedzi na post a jego oceną. Wyższa liczba odpowiedzi może sugerować większe zaangażowanie społeczności w dyskusję wokół danego postu, co z kolei może wpływać na wyższą ocenę postu.

Jednak wartość korelacji jest stosunkowo niska, co oznacza, że zależność między tymi cechami jest nieznaczna. Istnieją również inne czynniki, które mogą wpływać na ocenę postu, takie jak jakość zawartości, wartość merytoryczna, czy zrozumiałość treści, które nie są uwzględniane w tej analizie.

- **Czy istnieje zależność między oceną posta a liczbą słów w treści posta?**

Przeanalizujemy zależność między oceną posta a liczbą słów w treści posta. To pozwoli nam zrozumieć, czy długość posta ma wpływ na jego ocenę.

```
posts_data = posts_data.withColumn('WordCount', F.size(F.split(F.col('Body'), ' ')))

# Wykonaj analizę korelacji między oceną a liczbą słów
correlation = posts_data.stat.corr('Score', 'WordCount')

# Wyświetl wynik korelacji
print("Korelacja między oceną a liczbą słów:", correlation)
```

```
Korelacja między oceną a liczbą słów: 0.1559296857781053
```

Rysunek 47. Zależność między oceną posta a liczbą słów w treści posta

Na podstawie przeprowadzonej analizy korelacji można stwierdzić, że istnieje pewna, choć nieznaczna, dodatnia zależność między tymi dwiema cechami. Wartość korelacji wynosi 0.1559, co wskazuje na słabą zależność między oceną a liczbą słów.

Jednak warto zauważyć, że ta zależność jest stosunkowo niska, co sugeruje, że liczba słów w treści posta ma niewielki wpływ na ocenę posta. Inne czynniki, takie jak jakość treści, trafność odpowiedzi, czy użyteczność informacji, mogą mieć większy wpływ na ostateczną ocenę posta. Wnioskiem jest, że liczba słów w treści posta nie jest głównym determinantem oceny, ale może mieć pewne, choć niewielkie, znaczenie.

- **Jakie są przewidywane oceny na podstawie liczby wyświetleń i aktualnej oceny dla danych testowych?**

Wyodrębnię w tym celu dane testowe, aby przewidzieć oceny postów na podstawie liczby wyświetleń i aktualnej oceny. Wykorzystam również model predykcyjny do obliczenia przewidywanych ocen (rounded_prediction) na podstawie tych dwóch czynników. W tym celu korzystam z funkcji takich jak VectorAssembler, LinearRegression. Dane należy podzielić na zbiór treningowy i testowy w stosunku 8:2. Następnie tworzę model regresji liniowej, który na podstawie liczby wyświetleń przewiduje ocenę.

```python
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.sql.functions import round

# Przygotowanie danych
data = Posts.select("ViewCount", "Score").na.drop()  # Wybierz tylko potrzebne kolumny i usuń wiersze z brakującymi danymi

# Przygotowanie wektora cech
assembler = VectorAssembler(inputCols=["ViewCount"], outputCol="features")
data = assembler.transform(data)

# Podział danych na zbiór treningowy i testowy
train_data, test_data = data.randomSplit([0.8, 0.2], seed=42)

# Utworzenie modelu regresji liniowej
lr = LinearRegression(featuresCol="features", labelCol="Score")
model = lr.fit(train_data)

# Przewidywanie oceny na podstawie liczby wyświetleń
predictions = model.transform(test_data)


# Przewidywanie oceny na podstawie liczby wyświetleń
rounded_predictions = predictions.select("ViewCount", "Score", round("prediction", 2).alias("rounded_prediction"))
rounded_predictions.show()
```

Rysunek 48. Przewidywane oceny na podstawie liczby wyświetleń i aktualnej oceny

ViewCount	Score	rounded_prediction
43	2	4.75
46	1	4.81
47	1	4.83
50	3	4.88
53	2	4.93
58	8	5.02
61	3	5.07
64	1	5.13
68	1	5.2
68	4	5.2
68	4	5.2
69	4	5.22
71	0	5.25
72	5	5.27
78	3	5.38
84	2	5.48
87	3	5.53
97	5	5.71
104	7	5.84
108	9	5.91

Rysunek 49. Przewidywane oceny

Wyniki przedstawione w powyższej tabeli przedstawiają rzeczywiste wartości liczby wyświetleń (ViewCount) i oceny (Score) dla poszczególnych postów, oraz przewidywane wartości oceny na podstawie liczby wyświetleń. Kolumna "prediction" zawiera przewidywane wartości oceny.

Dla przykładu, dla pierwszego wiersza w tabeli mamy:

- ViewCount: 43
- Score: 2
- Prediction: 4.75

Oznacza to, że dla posta, który został wyświetlony 43 razy, rzeczywista ocena wynosi 2, a nasz model przewiduje ocenę na poziomie 4.75. Analogicznie, dla kolejnych wierszy możemy porównać rzeczywiste wartości oceny z przewidywanymi przez model. Model stara się znaleźć związek między liczbą wyświetleń a oceną, jednak wartości przewidywane mogą nie być idealnie zgodne z rzeczywistością. Aby ocenić jakość modelu, należy zastosować odpowiednie metryki oceny modelu regresji, takie jak błąd średnio kwadratowy (MSE) czy współczynnik determinacji (R^2).

```
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(labelCol="Score", predictionCol="prediction", metricName="mse")
mse = evaluator.evaluate(predictions)
print("Mean Squared Error (MSE):", mse)
```

```
Mean Squared Error (MSE): 33.128266993177675
```

```
evaluator = RegressionEvaluator(labelCol="Score", predictionCol="prediction", metricName="mae")
mae = evaluator.evaluate(predictions)
print("Mean Absolute Error (MAE):", mae)
```

```
Mean Absolute Error (MAE): 4.134429637080026
```

```
evaluator = RegressionEvaluator(labelCol="Score", predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)
print("R-squared:", r2)
```

```
R-squared: 0.8536703766079983
```

Rysunek 50.. Test poprawności przewidywania modelu

Na podstawie uzyskanych wartości metryk możemy wyciągnąć pewne wnioski:

- *Mean Squared Error (MSE):* 33.13. MSE mierzy średnią kwadratową różnicę między przewidywanymi wartościami a rzeczywistymi wartościami. Im niższa wartość MSE, tym lepiej. W przypadku tego modelu, wartość MSE jest umiarkowanie wysoka, co sugeruje, że model nie jest w stanie dokładnie przewidzieć ocen na podstawie liczby wyświetleń.
- *Mean Absolute Error (MAE)*: 4.13. MAE mierzy średnią różnicę bezwzględną między przewidywanymi wartościami a rzeczywistymi wartościami. Im niższa wartość MAE, tym lepiej. W przypadku tego modelu, wartość MAE jest względnie

niska, co sugeruje, że model dokonuje przewidywań bliskich rzeczywistym ocenom.

- *R-squared:* 0.85. R-squared mierzy stopień, w jakim zmienność w zmiennej zależnej (ocena) jest wyjaśniona przez model. Im bliżej wartości 1, tym lepiej. W przypadku tego modelu, wartość R-squared wynosi 0.85, co wskazuje na to, że model wyjaśnia około 85% zmienności ocen na podstawie liczby wyświetleń.

Podsumowując, mimo że wyniki metryk nie są idealne, model wydaje się być akceptowalny.

- **Czy można przewidzieć, czy dany post zostanie oznaczony jako ulubiony na podstawie długości treści postu i liczby komentarzy?**

Przeanalizujemy, czy istnieje możliwość przewidzenia, czy dany post zostanie oznaczony jako ulubiony na podstawie długości treści postu i liczby komentarzy. Wykorzystam odpowiedni model predykcyjny, aby zbadać tę zależność.

```
tokenizer = Tokenizer(inputCol="Body", outputCol="words_")
tokenized_data = tokenizer.transform(selected_data)

# Liczenie wystąpień słów
count_vectorizer = CountVectorizer(inputCol="words_", outputCol="features")
count_vectorizer_model = count_vectorizer.fit(tokenized_data)
vectorized_data = count_vectorizer_model.transform(tokenized_data)

# Wybierz odpowiednie kolumny dla modelu
selected_data = vectorized_data.select("features", "FavoriteCount")

# Podział danych na zbiór treningowy i testowy
(training_data, test_data) = selected_data.randomSplit([0.8, 0.2], seed=42)

# Inicjalizacja modelu klasyfikacji
lr = LogisticRegression(labelCol="FavoriteCount", featuresCol="features", rawPredictionCol="rawPredictions")

# Utworzenie potoku
pipeline = Pipeline(stages=[lr])

# Trenowanie modelu
model = pipeline.fit(training_data)

# Predykcja dla danych testowych
predictions = model.transform(test_data)

# Ocena modelu
evaluator = MulticlassClassificationEvaluator(labelCol="FavoriteCount", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)

print("Accuracy:", accuracy)
```

```
23/05/18 08:57:10 WARN Instrumentation: [56d1ad06] All labels are the same value and fitIntercept=true, so t
Accuracy: 1.0
```

Rysunek 51. Analiza czy dany post zostanie oznaczony jako ulubiony.

Wynik dokładności (accuracy) wynoszący 1.0 oznacza, że wszystkie przewidywania modelu regresji logistycznej dla danych testowych są zgodne z rzeczywistymi etykietami (FavoriteCount). Dokładność wynosząca 1.0 sugeruje, że model doskonale radzi sobie z klasyfikacją danych testowych i przewiduje poprawne etykiety dla wszystkich przykładów. Oznacza to, że nie ma żadnych błędów klasyfikacji w zestawie testowym.

5. Podsumowanie

Badanie obejmowało szeroki zakres tematów, a analiza przeprowadzona na różnych aspektach dostarczyła cennych wniosków. Analizując liczbę postów na przestrzeni czasu, możemy zidentyfikować trendy i określić okresy zwiększonej aktywności. Przeanalizowanie czasu aktywności dla najbardziej zaangażowanych użytkowników pozwoliło zidentyfikować liderów społeczności i ocenić ich wkład.

Porównanie najwyżej i najniżej ocenianych pytań pozwoliło na zrozumienie, jakie czynniki przyczyniają się do pozytywnej lub negatywnej oceny postów. Analiza procentowego udziału niezaakceptowanych odpowiedzi wśród najwyżej ocenianych dostarcza informacji na temat preferencji społeczności w zakresie akceptacji odpowiedzi. Rozkład ocen odpowiedzi zaakceptowanych vs pozostałych umożliwił ocenę znaczenia akceptacji odpowiedzi dla społeczności. Analiza najpopularniejszych tagów oraz liczby postów związanych z nimi pozwoliła zidentyfikować obszary tematyczne, które przyciągają największą uwagę. Badanie słów kluczowych w tytułach postów oraz analiza sekcji "About Me" użytkowników dostarczyły informacji na temat najważniejszych tematów i zainteresowań społeczności. Analiza zależności między liczbą odpowiedzi a oceną posta oraz między oceną posta a liczbą słów w treści pozwoliła na ocenę wpływu tych czynników na ocenę i popularność postów.

Dodatkowo, przeprowadzenie prognoz ocen na podstawie liczby wyświetleń i aktualnej oceny, a także predykcja oznaczenia posta jako ulubionego na podstawie długości treści i liczby komentarzy, otwierają perspektywę na wykorzystanie modeli predykcyjnych w analizie społecznościowej.

Wszystkie te wnioski łącznie dostarczają dogłębnego zrozumienia dynamiki i cech społeczności związanej z programem Mathematica na platformie Stack Exchange.

Spis rysunków

Bibliografia

1. "Python for Data Analysis" Wes McKinney
2. "Data Science from Scratch" Joel Grus
3. "Python Data Science Handbook" Jake VanderPlas
4. "Natural Language Processing with Python" Steven Bird, Ewan Klein, i Edward Loper
5. "Applied Text Analysis with Python" Benjamin Bengfort, Rebecca Bilbro i Tony Ojeda
6. "Text Mining with R: A Tidy Approach" Julia Silge i David Robinson
7. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" Aurélien Géron
8. "Deep Learning with Python" François Chollet
9. "Practical Natural Language Processing" Sowmya Vajjala, Bodhisattwa Majumder i Anuj Gupta
10. "Speech and Language Processing" Dan Jurafsky i James H. Martin