Scalable Analytics Final Project

User Profiling: Comparative Analysis of Elite and Non-Elite Users on Yelp

Table of Content

Project Overview

The main objective of this project is to perform a comparative analysis between elite and non-elite users based on their interactions and reviews on the yelp platform. This study aims to uncover patterns in behavior, preferences, and sentiment, providing insights that can help tailor services and engagement strategies more effectively.

Team Members and Responsibilities (TEAM 1)

- Reihaneh Moghisi: Data Preprocessing, Sentiment Analysis
- Shakiba Mashini: Exploratory Data Analysis (EDA)
- Hussain Kaide Johar Manasi: Exploratory Data Analysis (EDA)
- Muhammad Harris: Topic Modeling
- Jinendra Shrimal: Text Generation
- Ciaran Hartley: Text Generation

Data Preparation

Elite Flag Addition

Firstly, we enhanced the user dataset by adding a column to flag elite and non-elite users. This categorization is crucial for all subsequent analyses, ensuring we can accurately segment data and insights.

pythonCopy code (refer to code file 1. Users Data Cleaning and Problem Statement)

```
from pyspark.sql.functions import explode, split, col
# Split and explode 'elite' years into separate rows
df2 = df.withColumn("elite_years", split(col("elite"), ",\s*")).select("user_id",
explode("elite_years").alias("elite_year"))
# Count occurrences by 'elite_year'
elite_users_by_year = df2.groupBy("elite_year").count().orderBy("elite_year")
elite_users_by_year.show()
```

Data Cleaning ‘Review Dataset’

We conducted thorough cleaning of the reviews data, which involved normalizing text, removing punctuation and stop words, thus preparing the data for advanced text analysis.

pythonCopy code (refer to code file 3.Reviews Data Cleaning and Sentiment Analysis)

```
from pyspark.sql.functions import lower, regexp_replace, split
from pyspark.ml.feature import StopWordsRemover
# Function to clean and prepare text data
def clean_text(df, column_name, clean_column_name):
    df = df.withColumn(clean_column_name, lower(col(column_name)))
    df = df.withColumn(clean_column_name, regexp_replace(col(clean_column_name), r'([a-z])-([a-z])', r'\1 \2'))
    df = df.withColumn(clean_column_name, regexp_replace(col(clean_column_name), r'(\w)\.(\w)', r'\1\2'))
    df = df.withColumn(clean_column_name, regexp_replace(col(clean_column_name), r'(\w)\'(\w)', r'\1\2'))
    return df
df = clean_text(df, "text", "clean_text_column")
df = remove_punctuation_in_dataframe_column(df, "clean_text_column")
df = df.withColumn("clean_text_column", split(col("clean_text_column"), "\s+"))
df = remove_stop_words(df, "clean_text_column")
```

Analysis Overview

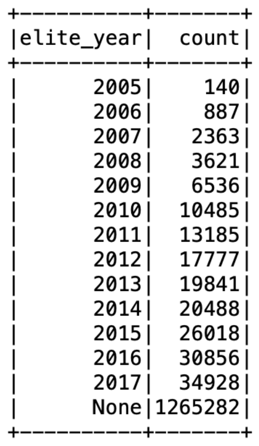
Exploratory Data Analysis (EDA)

We focused on uncovering top words and generating word clouds to visualize differences in language use between elite and non-elite users. This visual representation helps quickly grasp the prevalent themes in each group's reviews.

The analysis revealed a consistent increase in the number of elite users each year, indicating an effectively growing engagement and possibly improved user retention strategies by the platform.

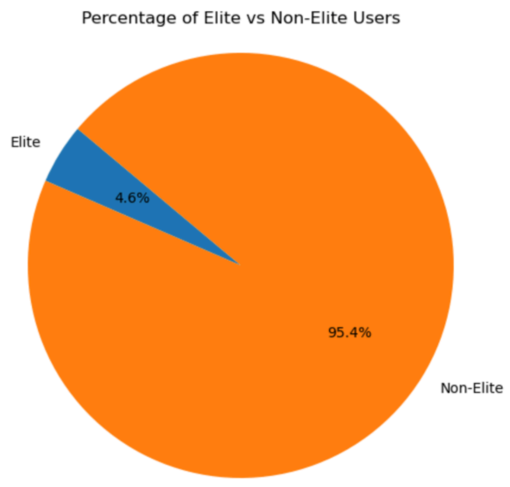
Peak Year

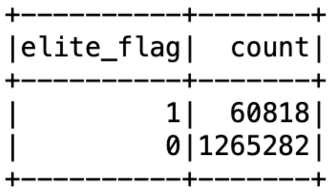
- **2017**: Marked as the year with the highest number of elite users. This spike in the elite user count could be attributed to successful marketing strategies or platform enhancements implemented in prior years.



```
+----------+-------+
|elite_year|  count|
+----------+-------+
|      2005|    140|
|      2006|    887|
|      2007|   2363|
|      2008|   3621|
|      2009|   6536|
|      2010|  10485|
|      2011|  13185|
|      2012|  17777|
|      2013|  19841|
|      2014|  20488|
|      2015|  26018|
|      2016|  30856|
|      2017|  34928|
|      None|1265282|
+----------+-------+
```

In general we see over 95% of users are non.Elite.





```
+----------+-------+
|elite_flag|  count|
+----------+-------+
|         1|  60818|
|         0|1265282|
+----------+-------+
```

We also observe significant differences between number of reviews, average star and number of useful reviews between elite and non elite users.

Elite users

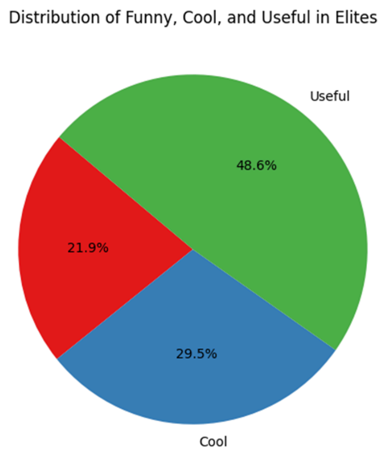
```
+-------------------+---------------------+------------------------+
|avg_avg_stars_elite|avg_review_count_elite|avg_useful_reviews_elite|
+-------------------+---------------------+------------------------+
|  3.847253937978893|   226.87628662567002|       476.1755894636456|
+-------------------+---------------------+------------------------+
```

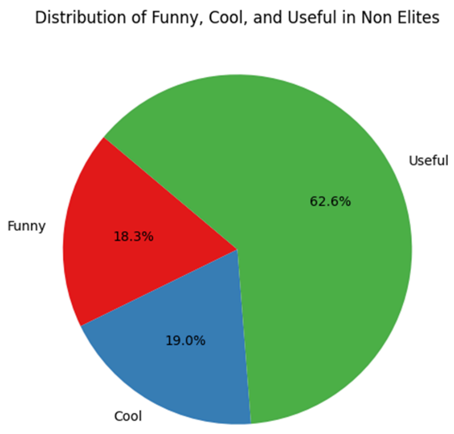
Non. Elite users:

```
+-----------------------+-------------------------+---------------------------+
|avg_avg_stars_non_elite|avg_review_count_non_elite|avg_useful_reviews_non_elite|
+-----------------------+-------------------------+---------------------------+
|      3.704283740697875|       13.323133499093483|         7.8670596752344535|
+-----------------------+-------------------------+---------------------------+
```

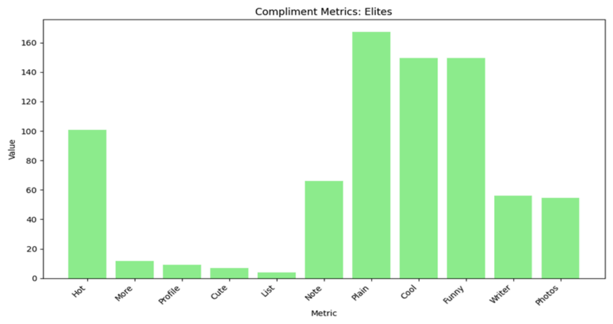
Key Insights

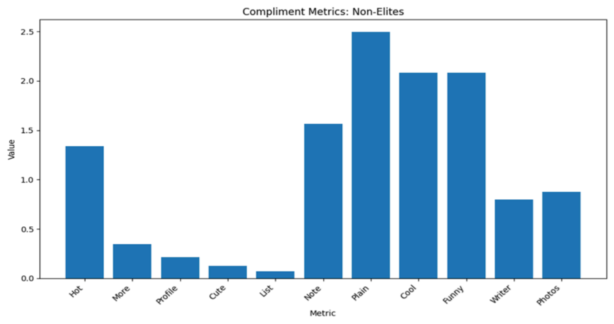
- Elite user reviews are generally cooler and funnier than non elite user reviews



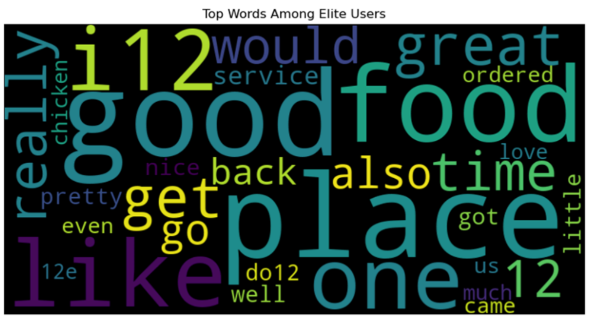


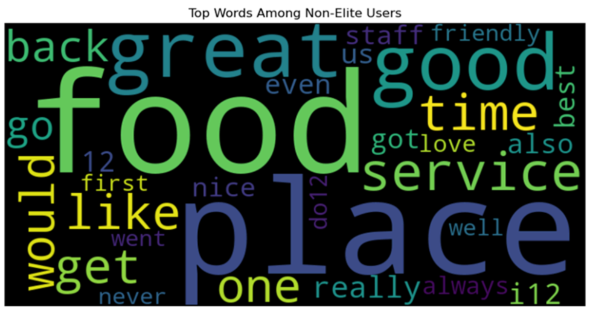
- Elite users receive 100x more compliments than non elite users, but the distribution of compliments is pretty much the same





- Word Clouds show that although both elite and non elite users use the same vocabulary, non elite users may prefer extra courtesies like good service, and friendly atmosphere in addition to just good food.





Sentiment Analysis

As part of our comprehensive analysis of Elite and Non-Elite users, we calculated and compared the average sentiment scores, specifically polarity and subjectivity, for each group. These metrics provide deeper insights into the emotional and subjective content of user reviews, which can be essential for understanding user satisfaction and engagement levels.

Sentiment Analysis Metrics

In our ongoing analysis comparing Elite and Non-Elite users on our platform, we have computed and analyzed the average sentiment scores—polarity and subjectivity—for each group. These metrics are crucial for discerning the emotional tone and the level of personal bias in user reviews, which can significantly influence user engagement strategies.

Polarity

- Definition: Polarity indicates the emotional orientation of written text. It ranges from -1 to +1, where -1 represents extremely negative sentiment, +1 indicates extremely positive sentiment, and a score around 0 suggests a neutral stance.
- Purpose: Understanding polarity helps in assessing the overall sentiment (positive, negative, or neutral) prevalent in user communications.

Subjectivity

- Definition: Subjectivity measures the extent to which information is expressed in a personal manner, using opinions rather than facts. Scores range from 0 (completely objective) to 1 (highly subjective).
- Purpose: This metric is useful to determine how much of the communication is based on personal opinions and feelings as opposed to factual content.

Methodology

We utilized the TextBlob library for this analysis because of its straightforward API for extracting polarity and subjectivity from textual content. The process involved:

1. Data Preparation: Converting arrays of words into a single coherent string per review.
2. Sentiment Calculation: Applying TextBlob to calculate polarity and subjectivity for each review.
3. Average Computation: Averaging these scores across all reviews for each user group (Elite and Non-Elite).

pythonCopy code (refer to code file 3.Reviews Data Cleaning and Sentiment Analysis)

```
from textblob import TextBlob
def get_polarity(text):
    return TextBlob(text).sentiment.polarity
def get_subjectivity(text):
    return TextBlob(text).sentiment.subjectivity
# Calculate average polarity and subjectivity
avg_sentiment = df.groupBy("elite_flag").agg(
    avg("polarity").alias("avg_polarity"),
    avg("subjectivity").alias("avg_subjectivity")
)
avg_sentiment.show()
```

Results and Interpretation

The sentiment analysis yielded insightful differences between the two user groups:

- Polarity: Non-Elite users had an average polarity of 0.24 compared to 0.22 for Elite users, indicating that Non-Elite users tend to express marginally more positive sentiments in their reviews.
- Subjectivity: The average subjectivity score was also slightly higher for Non-Elite users (0.55) compared to Elite users (0.54), suggesting that Non-Elite users' reviews are more opinion-based than those of Elite users.

Elite

```
+-------------------+------------------+
|       avg_polarity|  avg_subjectivity|
+-------------------+------------------+
|0.22608805795202244|0.5457307004258998|
+-------------------+------------------+
```

Non.Elite

```
+------------------+------------------+
|      avg_polarity|  avg_subjectivity|
+------------------+------------------+
|0.2456547673315709|0.5568185331539073|
+------------------+------------------+
```

Possible Justifications

- Engagement Levels: Non-Elite users might demonstrate slightly higher positivity (polarity) and personal input (subjectivity) as they might feel a stronger need to express distinct personal experiences and opinions to establish their presence or influence on the platform.
- Experience and Expectations: Elite users often have prolonged interaction with the platform, which could lead to more tempered expectations and expressions—reflecting in a slightly lower subjectivity and polarity score. Their reviews might be more balanced and measured as they are familiar with what to expect.
- Incentive and Motivation: Non-Elite users could be using their reviews as a platform to voice their distinct opinions strongly, perhaps in an effort to be recognized or move towards Elite status.

Conclusion

This analysis of polarity and subjectivity highlights subtle yet potentially significant differences in how Elite and Non-Elite users interact with and perceive the platform. The slightly higher subjectivity and polarity scores for Non-Elite users could indicate a different set of motivations and engagement patterns, which can be essential for developing targeted marketing and customer interaction strategies. By tailoring our approaches to these nuanced user behaviors, we can better align our services with user expectations and enhance overall satisfaction.

Topic Modeling

We implemented LDA topic modeling to identify prevalent topics within the corpus of reviews. This technique helps in understanding what subjects or themes are commonly discussed by elite versus non-elite users, providing insight into their interests and concerns.

Introduction:

Latent Dirichlet Allocation (LDA) is a powerful statistical model used for analyzing text data and discovering underlying topics within a collection of documents. It operates on the principle of probabilistic topic modeling, where each document is represented as a mixture of topics, and each topic is characterized by a distribution of words. This report explores the significance of LDA in text analysis and describes the pipeline involved in applying LDA to text data.

Why LDA?

LDA is utilized for several reasons:

1. Summarizing Text Data: LDA facilitates the summarization of large volumes of text data by extracting the most relevant topics and themes. It condenses the information into a concise representation, enabling users to gain insights and understanding quickly.
2. Enhancing Content Categorization: LDA improves content categorization by automatically assigning documents to relevant topics. It aids in organizing and classifying text data, making it more manageable and accessible for further analysis.

LDA Pipeline:

1. Text Tokenization:

The first step in the LDA pipeline is text tokenization. This process involves splitting the text into individual words or tokens using a regular expression pattern (\W, which matches any non-word character). The output is stored in the 'words' column.

2. Stop Words Removal:

After tokenization, common 'stop words' such as 'the', 'is', and 'at' are removed from the tokenized words. This step helps eliminate noise and irrelevant words from the text data, improving the quality of analysis. The result is stored in the 'filtered_words' column.

3. Feature Vectorization:

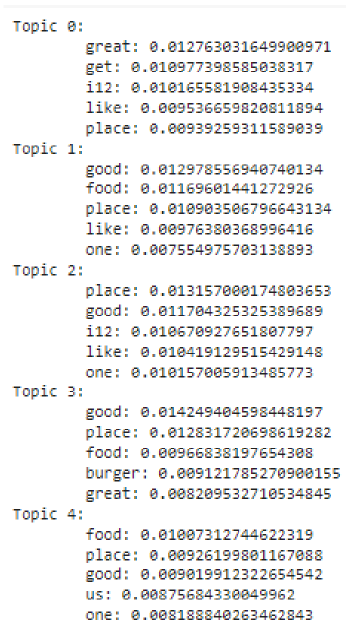
Feature vectorization converts the text data into a numerical representation by counting the occurrences of each word. Only the top 2000 words that appear in at least 10 documents are considered for vectorization. This process generates a feature vector of word counts, which serves as input for the LDA model.

4. LDA:

The final step in the pipeline involves applying the LDA model to the feature vectors. LDA utilizes the 'raw_features', which represent the numerical representation of text after tokenization and stop word removal. It efficiently discovers thematic structures within the text data using an 'online' optimizer, making it suitable for large datasets.

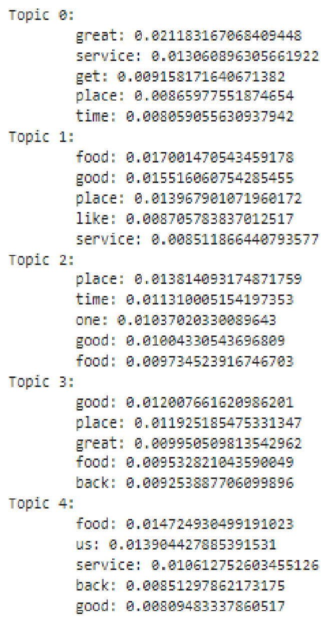
Results - Elite vs Non-Elite Users: (refer to code file 4 Topic Modeling using LDA)

Elite



```
Topic 0:
        great: 0.012763031649900971
        get: 0.010977398585038317
        i12: 0.010165581908435334
        like: 0.009536659820811894
        place: 0.00939259311589039
Topic 1:
        good: 0.012978556940740134
        food: 0.01169601441272926
        place: 0.010903506796643134
        like: 0.00976380368996416
        one: 0.007554975703138893
Topic 2:
        place: 0.013157000174803653
        good: 0.011704325325389689
        i12: 0.010670927651807797
        like: 0.010419129515429148
        one: 0.010157005913485773
Topic 3:
        good: 0.014249404598448197
        place: 0.012831720698619282
        food: 0.00966838197654308
        burger: 0.009121785270900155
        great: 0.008209532710534845
Topic 4:
        food: 0.01007312744622319
        place: 0.00926199801167088
        good: 0.009019912322654542
        us: 0.00875684330049962
        one: 0.008188840263462843
```

Non-Elite



```
Topic 0:
        great: 0.021183167068409448
        service: 0.013060896305661922
        get: 0.009158171640671382
        place: 0.008659775518746544
        time: 0.008059055630937942
Topic 1:
        food: 0.017001470543459178
        good: 0.015516060754285455
        place: 0.013967901071960172
        like: 0.008705783837012517
        service: 0.008511866440793577
Topic 2:
        place: 0.013814093174871759
        time: 0.011310005154197353
        one: 0.01037020330089643
        good: 0.01004330543696809
        food: 0.009734523916746703
Topic 3:
        good: 0.012007661620986201
        place: 0.011925185475331347
        great: 0.009950509813542962
        food: 0.009532821043590049
        back: 0.009253887706099896
Topic 4:
        food: 0.014724930499191023
        us: 0.013904427885391531
        service: 0.010612752603455126
        back: 0.00851297862173175
        good: 0.00809483337860517
```

Conclusion:

Latent Dirichlet Allocation (LDA) plays a crucial role in text analysis by uncovering hidden topics and themes within a collection of documents. By following a structured pipeline involving text tokenization, stop words removal, feature vectorization, and LDA modeling, users can extract valuable insights and understanding from text data. LDA facilitates content summarization, categorization, and organization, making it an indispensable tool in natural language processing and text mining applications.

Text Generation

We used Markov chains to generate text based on the review data, creating realistic and coherent sentences that mimic the writing styles of elite and non-elite users. This can be particularly useful for simulations and training customer service AI.

Introduction:

Markov chains are stochastic models used to describe sequences of events where the probability of each event depends only on the state attained in the previous event. In this report, we discuss the construction of a Markov chain dictionary from input text data and its application in predicting and generating text.

Building the Markov Chain:

The first step in utilizing a Markov chain for text generation is to construct a dictionary where each word in the corpus serves as a key, and the corresponding value is a list of words that frequently follow the key word in the text. This process involves analyzing the input text data to extract word sequences and count the occurrences of words following each other.

pythonCopy code (refer to code file 5. Text Generation using Markov Chains)

```
def markov_chain(text):
    '''The input is a string of text and the output will be a dictionary with each word as
      a key and each value as the list of words that come after the key in the text.'''
    # Tokenize the text by word, though including punctuation
    words = text.split(' ')
    # Initialize a default dictionary to hold all of the words and next words
    m_dict = defaultdict(list)
    # Create a zipped list of all of the word pairs and put them in word: list of next words format
    for current_word, next_word in zip(words[:-1], words[1:]):
        m_dict[current_word].append(next_word)
    # Convert the default dict back into a dictionary
    m_dict = dict(m_dict)
    return m_dict
```

Predicting Next Word:

Once the Markov chain dictionary is constructed, it can be consulted to predict the next word in a sequence given a starting word. This prediction is based on the probabilities of different words occurring after the current word in the text, as inferred from the Markov chain dictionary.

Text Generation Process:

Text generation using a Markov chain involves iteratively selecting a starting word randomly from the corpus and predicting subsequent words based on the Markov chain dictionary. This process continues until the desired length of text is generated or a stopping condition is met.

Sentence Generation Function:

A function is developed to facilitate sentence generation using the Markov chain. This function takes inputs such as the Markov chain dictionary and the desired number of words in the generated sentence. It randomly selects a starting word and predicts subsequent words based on the Markov chain dictionary until the desired number of words is reached or the end of a sentence is encountered.

pythonCopy code (refer to code file 5. Text Generation using Markov Chains)

```
def generate_sentence(corpus_dict, num_words):
    sentence = []
    word = random.choice(list(corpus_dict.keys()))  # Start with a random word from the corpus
    sentence.append(word)
    while len(sentence) < num_words:
        next_words = corpus_dict.get(word, ["."])
        next_word = random.choice(next_words)  # Get next word based on current word
        if next_word == ".":
            break
        sentence.append(next_word)
        word = next_word
        # If we reach the end of the corpus for a word, break the loop
        if word not in corpus_dict:
            break
    return " ".join(sentence)
```

Conversion to PySpark User Defined Function (UDF):

To scale up the text generation process for large datasets, the sentence generation function is converted into a PySpark UDF. This allows for seamless integration with Spark DataFrames, enabling efficient processing of text data in distributed computing environments.

pythonCopy code *(refer to code file 5. Text Generation using Markov Chains)*

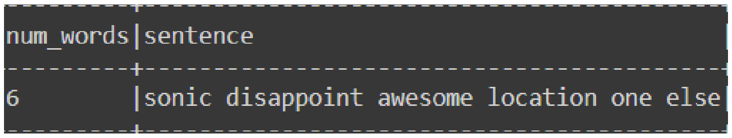
```
generate_sentence_udf_elite = udf(lambda x: generate_sentence(elite_markov_chain, x), StringType())
generate_sentence_udf_nonelite = udf(lambda x: generate_sentence(non_elite_markov_chain, x), StringType())
# Create a DataFrame with a single column containing the desired number of words
num_words = 10 # Edit this number at will
num_words_df = spark.range(1).withColumn("num_words", lit(num_words))
# Generate sentences based on the number of words
# Elite
sentence_df_elite = num_words_df.withColumn("sentence", generate_sentence_udf_elite("num_words"))
sentence_df_elite.show(truncate=False)
# Non-Elite
sentence_df_nonelite = num_words_df.withColumn("sentence", generate_sentence_udf_nonelite("num_words"))
sentence_df_nonelite.show(truncate=False)
```

Results and Analysis:

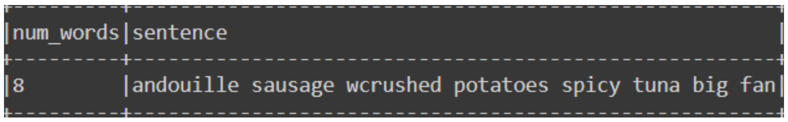
Generated sentences from elite user data tend to display a higher sense of focus compared to non-elite user data. This observation suggests that Markov chain-based text generation can capture stylistic differences in user-generated content, potentially providing insights into user behavior and preferences.

Example Outputs: *(refer to code file 5. Text Generation using Markov Chains)*

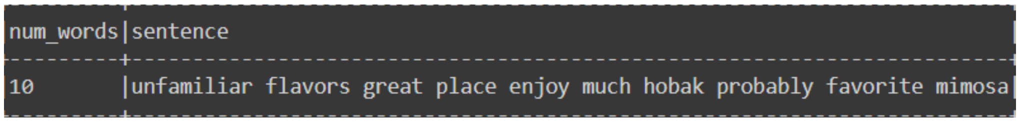
Elite Users:



```
---------+-----------------------------------------
num_words|sentence
---------+-----------------------------------------
6        |sonic disappoint awesome location one else
---------+-----------------------------------------
```

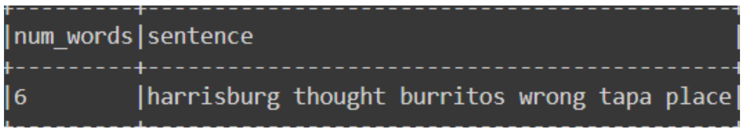


```
+---------+--------------------------------------------------
|num_words|sentence                                         |
+---------+--------------------------------------------------
|8        |andouille sausage wcrushed potatoes spicy tuna big fan|
+---------+--------------------------------------------------
```

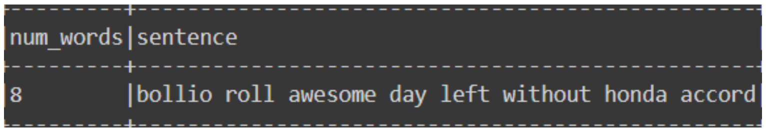


```
num_words|sentence
---------+--------------------------------------------------------------------------
10       |unfamiliar flavors great place enjoy much hobak probably favorite mimosa|
---------+--------------------------------------------------------------------------
```

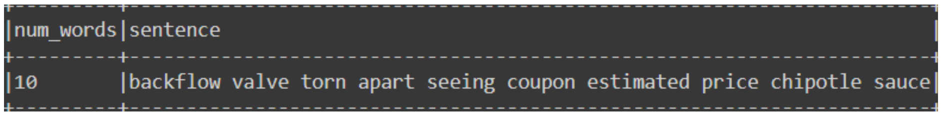
Non-Elite Users:



```
+----------+------------------------------------------+
|num_words|sentence                                   |
+----------+------------------------------------------+
|6        |harrisburg thought burritos wrong tapa place|
```



```
+---------+--------------------------------------------+
|num_words|sentence                                      |
+---------+--------------------------------------------+
|8        |bollio roll awesome day left without honda accord|
+---------+--------------------------------------------+
```



```
+---------+------------------------------------------------------------------+
|num_words|sentence                                                            |
+---------+------------------------------------------------------------------+
|10       |backflow valve torn apart seeing coupon estimated price chipotle sauce|
+---------+------------------------------------------------------------------+
```

Conclusion:

Markov chains offer a versatile approach to text generation, capable of capturing probabilistic dependencies in input text data and producing coherent and contextually relevant output. By leveraging Markov chains and PySpark UDFs, text generation tasks can be efficiently scaled for large datasets, enabling analysis of user-generated content at scale.

Conclusion and Future Work

This project has comprehensively investigated the interactions and review patterns of elite and non-elite users on our platform. By employing a multifaceted approach that incorporated data preparation, sentiment analysis, topic modeling, and text generation, we uncovered valuable insights into their behaviors and preferences.

Key Findings:

- Elite users are a growing segment, suggesting successful platform strategies for user engagement and retention.
- Non-elite users tend to express slightly more positive sentiment and subjective opinions in their reviews, which could indicate a stronger desire to establish their presence or influence.
- Elite user reviews might be more balanced and measured due to longer interaction and tempered expectations.
- LDA topic modeling revealed distinct thematic interests and concerns between the two user groups.
- Text generation using Markov chains demonstrated the potential to capture stylistic differences in user-generated content.

Implications:

These findings highlight the importance of tailoring our marketing and customer interaction strategies to cater to the nuanced differences in user behavior. By understanding the motivations and communication styles of both elite and non-elite users, we can create a more personalized and engaging platform experience.

Future Work:

- Further exploration of user demographics and their correlation with review sentiment and topics could provide deeper insights.
- Advanced sentiment analysis techniques could delve into the emotional nuances of user reviews.
- Integrating topic modeling with sentiment analysis could reveal the sentiment associated with specific topics discussed by each user group.

Appendix

Submission Summary	
File Name	Description
1. Users Data Cleaning and Problem Statement	Analysis for problem statement formation and code for data cleaning of users data.
2. EDA	Detailed Exploratory Analysis to highlight the user patterns, distributions, and metrics to compare the elite and non-elite users
3.Reviews Data Cleaning and Sentiment Analysis	Data Cleaning of Reviews Data, Text Manipulation, and Sentiment Analysis to further analyze user characteristics
4. Topic Modeling using LDA	Text Preprocessing andDriving different themes of the reviews among Elite and Non- Elite Users using Latent Dirichlet allocation.
5. Text Generation using Markov Chains	Utilized Markov chains to generate texts reflecting the voices of both elite and non-elite users' reviews.