DATA7201 Project – Ad campaign analysis

Abstract

This project explores the application of big data analytics to political advertising on Facebook, focusing specifically on the analysis of ad campaigns targeting Australian users from 2020 to 2024, I then apply a series of data preprocessing and text analytics techniques to extract meaningful insights, text analytics to be specific.

The primary aim is to conduct topic modelling using Latent Dirichlet Allocation (LDA) to identify themes and sentiments within the ads. My findings reveal that political messages are mostly neutral with spikes of positive and negative sentiments, potentially reflecting diverse campaign strategies. Moreover, recurring themes such as public health, environmental issues, and economic policies highlight how political entities engage with societal concerns.

This study demonstrates the practical application of distributed computing and machine learning in analyzing large datasets. Through this analysis, I provide certain understanding of how political campaigns utilize digital platforms for targeted messaging, offering insights into the intersection of technology, politics, and society.

Total word count: 1312 words

Table of Contents

Introduction

In this digital era, we have emerged in vast amounts of data, commonly called as the 'big data'. Social media, transactions, mobile applications, basically everything has its connection with the data. With that being said, managing and extracting value from such sources become an urgent mission for many organizations, showing how important it is to master the art of big data analytics.

Big data analytics involves examining complex data sets to unravel patterns, correlations, and insights. Given the enormous volume, rapid velocity, and diverse variety of big data, dealing with it can be a troublesome task. This is where distributed systems find their spotlight.

Distributed systems influence multiple nodes to manage and analyze data, allowing them to efficiently scale as data grows, provide redundancy to safeguard against potential system failures, and facilitate rapid data processing to support real-time decision-making. This is crucial across various examples: E-commerce platforms like Amazon analyze customer behavior and manage their stocks in real-time; healthcare providers utilize these systems to process patient data swiftly to improve health assistance; and financial services rely on them to detect and prevent fraud dynamically. Thus, the integration of distributed systems is essential for applying the full potential of big data analytics, ensuring that organizations can maintain efficiency and competitive advantage in this data-driven economy.

In this project, I will exploit data analytics to explore and demonstrate insights from a given dataset.

Data Analytics

Objective

The primary goal of this analysis is to perform topic modelling on the text content of political Facebook ads to understand the main themes and topics that are being communicated. This involves several preprocessing steps to clean and prepare the text data, followed by applying user-defined functions to clean the text.

Data Preprocessing

The dataset chosen for my project is **Facebook Ad Library API,** a collection of sponsored political posts on Facebook targeted at Australian users during 4 years (03/2020-02/2024).

It contains a total of 26 features:

Feature	Description	Example
ad_creation_time	The UTC date and time when the ad was created	2021-12-02
ad_creative_bodies	A list of text displayed in each unique ad card of a carousel or similar ad format	["Text for card 1", "Text for card 2"]
ad_creative_body	The primary text content of the ad	Stay informed with the latest news from our State…
ad_creative_link_caption	The caption associated with the hyperlink in the ad found below the image or video	nswliberal.org.au
ad_creative_link_captions	A list of captions associated with each unique ad card	["Caption for card 1", "Caption for card 2"]
ad_creative_link_description	A brief description associated with the hyperlink	Sign up for our e-newsletter for the latest news…
ad_creative_link_descriptions	List of descriptions for each unique link in ads that feature multiple links or carousel cards	["Description for link 1", "Description for link 2"]
ad_creative_link_title	The title of the hyperlink in the ad	Stay Informed
ad_creative_link_titles	A list of titles for each unique ad card's hyperlink in multi-card ads	["Title for card 1", "Title for card 2"]
ad_delivery_start_time	The start date and time when the ad began being delivered to audiences in UTC	2021-12-02
ad_delivery_stop_time	The end time for ad delivery. If blank, the ad runs indefinitely until manually stopped or budget is depleted	2023-04-20
ad_snapshot_url	URL to the archived version of the ad	http://....
bylines	Information about the sponsor of the ad	Paid for by NSW Campaign
currency	The currency used for transactions related to the ad campaign	AUD

Feature	Description	Example
delivery_by_region	Distribution data showing which regions (state, province) the ad was shown in, based on the location of the audience.	"Queensland", "New South Wales"
demographic_distribution	Breakdown of the audience that the ad reached, segmented by age, gender	{"18-24": {"male": 500, "female": 600}}
estimated_audience_size	Estimated range of the audience size that the ad targeted	{0, 999}
funding_entity	The organization or individual that funded the ad	NSW Campaign
id	A unique identifier for the ad within the Facebook Ad Library	214843134128684
impressions	A range indicating the number of times the ad was displayed to users	{0, 999}
languages	The languages used in the ad	EN
page_id	The identifier for the Facebook Page that ran the ad	44701328033
page_name	The name of the Facebook Page that ran the ad	NSW Liberal Party
publisher_platforms	The Meta platforms (e.g., Facebook, Instagram) where the ad was published.	"Facebook", "Instagram"
region_distribution	A breakdown of where the ad was shown by geographic region	[{1, Queensland}]
spend	The amount of money spent on the ad	{0, 99}

Table 1: Features

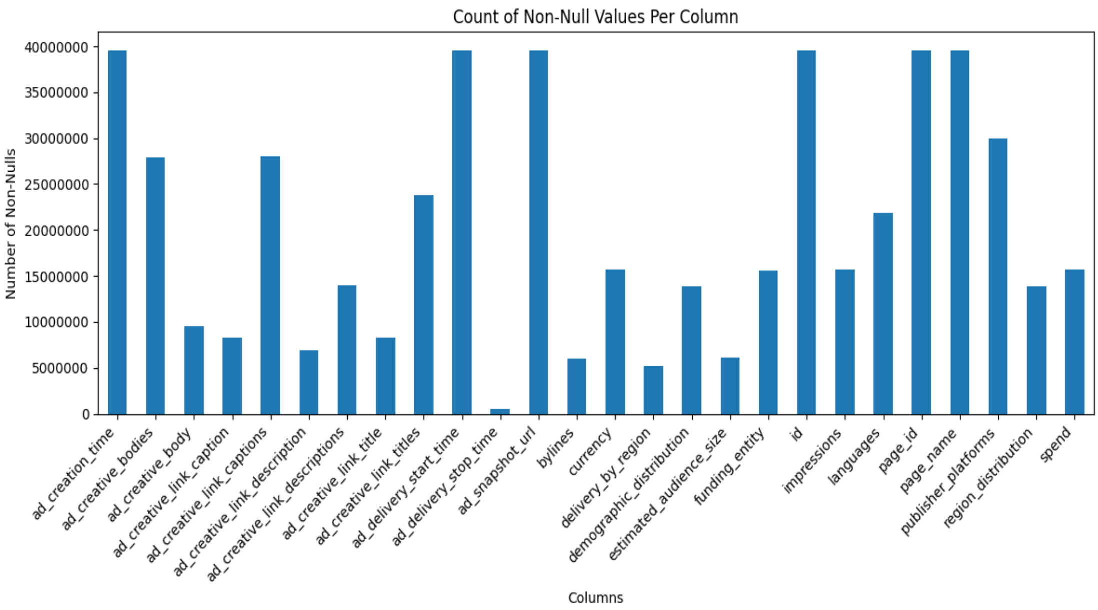

Figure 1: Feature Count

I establish a brief understanding of the features to see what elements are in the dataset and feel like assessing the topic analysis would be something interesting when it comes to political ad campaigns (see Table 1, Figure 1 for feature summary).

Since ‘ad_creative_body’ is the primary message of each campaign we filter out the rows that have NULL values in this column (see Figure 2) and filter out the features we needed for our later analysis (df is the loaded main dataset).

```
# Filter out rows with NULL values in the ad_creative_body column
filtered_df = filtered_df.filter(col("ad_creative_body").isNotNull())

# List of columns needed for analysis
columns_needed = ["ad_creation_time", "ad_creative_body", "ad_delivery_start_time", "spend", "impressions"]

# Select only the needed columns
filtered_df = df.select(columns_needed)
```

Figure 2: Filter code

A user-defined function ‘clean_text’ is defined and registered to clean text data by converting it to lowercase, removing punctuation, and stripping whitespace. I later applied it to our ‘ad_creative_body’ and added the cleaned version column called ‘clean_ad_body; then we tokenize the words and removed the stop words (see Figure 3).

```python
# Define a UDF to clean the text
def clean_text(text):
    import re
    # Convert to lower case
    text = text.lower()
    # Remove punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Strip whitespaces
    text = text.strip()
    return text

# Register the UDF
clean_text_udf = udf(clean_text, StringType())

# Apply the UDF to clean 'ad_creative_body' into 'clean_ad_body'
filtered_df = filtered_df.withColumn("cleaned_ad_body", clean_text_udf(col("ad_creative_body")))

# Tokenize the cleaned text
tokenizer = Tokenizer(inputCol="cleaned_ad_body", outputCol="words")
filtered_df = tokenizer.transform(filtered_df)

# Remove stopwords
remover = StopWordsRemover(inputCol="words", outputCol="filtered_text")
filtered_df = remover.transform(filtered_df)
```

Figure 3: clean text function

Here is a showcase of how the result would look like (Figure 4).

```
-RECORD 0-----------------------------------------------------------------------------------------------------------
-------------------------
 cleaned_ad_body | im ready to fight scott morrisons cashless pension card being rolled out in lilley
 words           | [im, ready, to, fight, scott, morrisons, cashless, pension, card, being, rolled, out, in, lilley]
 filtered_text   | [im, ready, fight, scott, morrisons, cashless, pension, card, rolled, lilley]
-RECORD 1
```

Figure 4: filtered_text showcase

Text analytics

Due to the heavy load of data in our 'filtered_df' (9568757 indexes), I sampled it into a smaller dataset to reduce size and speed up the analytics process and eventually saved it as a JSON file to process the analysis and visualization in the local computer (See Figure 5). Unfortunately, after down-scaling a couple of times, the memory still wasn't enough to process it, so I had to skip this part and do the analysis on the person space.

```python
# Sample a smaller subset
sampled_df = filtered_df.sample(fraction=0.0001, seed=66)

# Define the path where you want to save the JSON file
json_path =' projeca/sample_data.json'

# Save the DataFrame to JSON
sampled_df.write.json(json_path, mode='overwrite')

print(f"Sampled data saved to {json_path}")
```

Figure 5: Sample smaller dataset

Next is the important part. In this analysis, the text data is first tokenized. Then, a count vectorizer is used to convert the collection of words into numerical feature vectors, which represent the frequency of each word. To further refine these features, the term frequencies are scaled using the inverse document frequency (IDF). The resulting feature vectors are then used to train a Latent Dirichlet Allocation (LDA) model, which identifies the topics within the text. For quick results, the LDA model is trained with only one iteration. Finally, the topics are mapped back to their corresponding words into a vocabulary map, and the distribution of these topics over time is then visualized in the next part. (Code shown in Figure 6)

```python
# Count Vectorizer
cv = CountVectorizer(inputCol="filtered_text", outputCol="rawFeatures")
cv_model = cv.fit(sampled_df)
featurized_data = cv_model.transform(sampled_df)

# IDF
idf = IDF(inputCol="rawFeatures", outputCol="features")
idf_model = idf.fit(featurized_data)
rescaled_data = idf_model.transform(featurized_data)

# Persist with MEMORY_AND_DISK storage level
rescaled_data.persist(StorageLevel.MEMORY_AND_DISK)

# Number of topics
num_topics = 5

# Train an LDA model with a iteration for faster results
lda = LDA(k=num_topics, maxIter=1)
lda_model = lda.fit(rescaled_data)

# Show the topics found by the LDA model
topics = lda_model.describeTopics(maxTermsPerTopic=10)
topics.show(truncate=False)

# Map topics to words using the vocabulary
vocab = cv_model.vocabulary
mapped_topics = topics.rdd.map(lambda row: (row['topic'], [vocab[idx] for idx in row['termIndices']])).collect()
for topic, words in mapped_topics:
    print(f"Topic {topic}: {words}")
```

Figure 6: Text analytics

The image below is an example of what the topics look like.

```
Topic 0: ['vaccine', 'get', 'passport', 'vaxxed', 'climate', 'choice', 'australians', 'afghan', 'fair', 'need']
Topic 1: ['energy', 'torres', 'strait', 'campaign', 'help', 'sign', 'tas', 'emissions', 'options', 'training']
Topic 2: ['westgate', 'girls', 'support', 'day', 'pasadena', 'campaign', 'work', 'community', 'businesses', 'park']
Topic 3: ['greenwashing', 'portfolios', 'ethical', 'beyond', 'sustainable', 'go', 'selected', 'bph', 'solve', 'ability']
Topic 4: ['lending', 'oceans', 'pushing', 'laws', 'devastating', 'global', 'protect', 'change', 'oil', 'deforestation']
```

Figure 7: Topic examples

Visualization

Sentiment score distribution

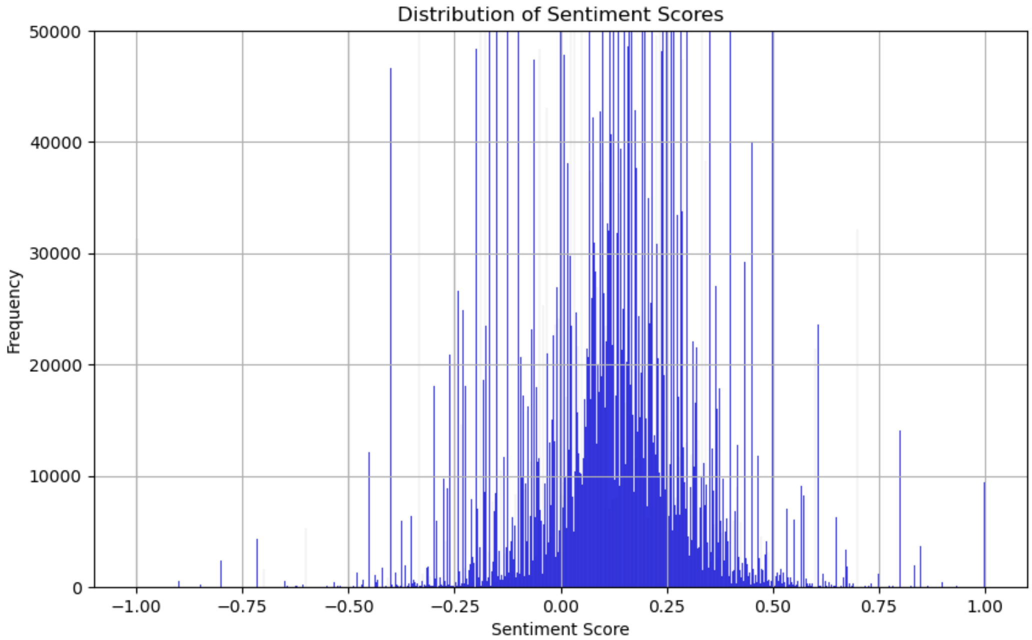

Figure 8: Distribution of sentiment score among topics

The first visualization I implemented was the sentiment score distribution (Figure 8). I used **TextBlob**, a pre-trained Sentiment Analysis Model and assigned a simple scoring mechanism for the topics, as well as **Matplotlib** for plotting the distribution of sentiment scores. This can provide several insights into the overall emotional tone of the ad campaigns. Here's what we can infer from the plot:

1. **Sentiment Score Range**:
    - The sentiment scores range from -1 to 1, where -1 indicates very negative sentiment, 0 is neutral, and 1 is very positive.

2. **Distribution Shape**:
   - The distribution appears to be centered around 0, indicating a neutral sentiment for most of the ads.
   - There are spikes at various points, suggesting the presence of both positive and negative sentiment ads, but still, most ads are staying around the neutral point.
3. **Frequency**:
   - The y-axis shows the number of ads.
   - The highest frequency is near the neutral score (0), which implies that most ads have a neutral sentiment.

Word cloud

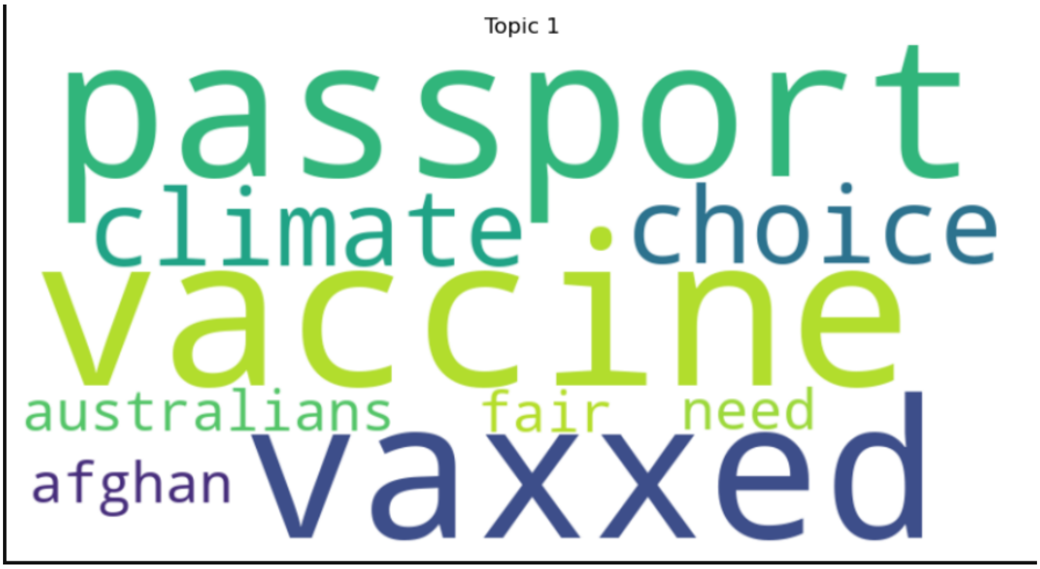

Figure 9: Word Cloud(topic 1)

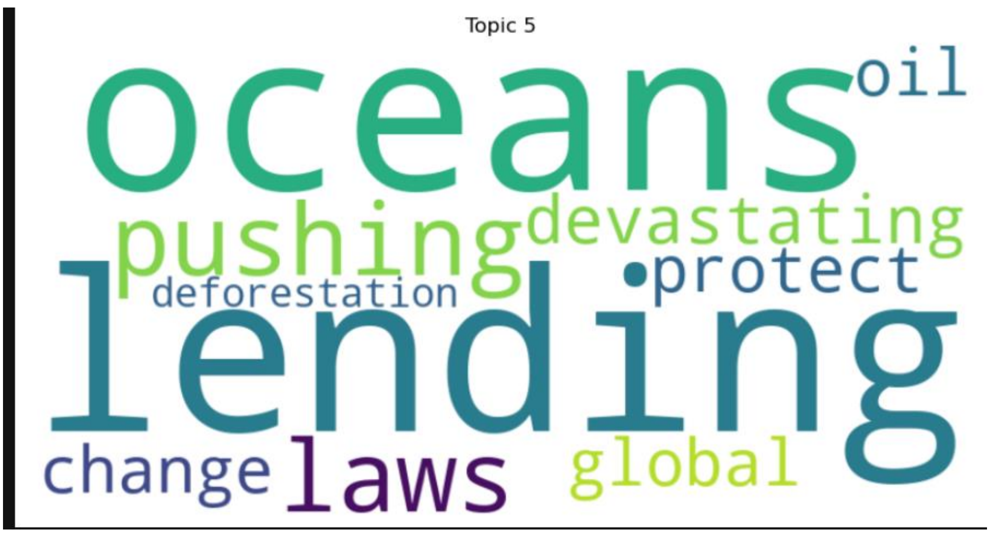

Figure 10: Word Cloud (Topic 5)

The last analysis I did was word cloud visualisation (Figures 9 & 10 Shown) by importing **WordCloud** for generating word clouds for each topic and **Matplotlib** for visualization in the python library. For demonstrating purpose, I used topic 1 and topic 5 for example. We can easily observe that:

- **Health and Environmental Intersection**: The word clouds suggest a significant overlap in discussions around public health and environmental issues.
- **Advocacy and Policy**: Both topics highlight the importance of advocacy and policy creation, maybe for health measures or enacting laws to protect the environment.
- **Global and Local Concerns**: The presence of words like "global," "australians," and "afghan" suggests that the campaigns address both global and local issues.
- **Economic and Industrial Impact**: The emphasis on words like "lending" and "oil" indicates that economic activities and their impact on society.

Conclusion

To conclude my project, the analysis of political Facebook ads reveals some insights worth mentioning into the themes and sentiments expressed through these campaigns. The primary findings show that most of the sentiments are neutral, following a normal distribution pattern. The topic modelling (LDA) reveals recurring themes related to public health, environmental issues, and economic concerns, showcasing how political entities target and engage with specific societal issues. The visualizations also display the emphasis on policy advocacy and the intersection of global and local concerns within the ads. It demonstrates how effective advanced data analytics techniques are, such as distributed computing and machine learning, in extracting notable insights from large datasets.

A key lesson for me from this project is the importance of feature extraction. Without a proper plan beforehand, computational resources could be used in waste and lose a lot of progress. In other words, deciphering patterns within big data before starting is a skill data scientists must acquire.

Appendix

Note: Codes were used in Jupyter Notebook in Python language

import sys print(sys.executable) print(sys.version)

Check Python path for Pyshark

import sys print("\nPython paths:") for p in sys.path: print(p)

```
import os

# Specify the path to the Python 3.8 executable
python_path = "/conda/bin/python"

# Set the environment variables for both the driver and the workers
os.environ["PYSPARK_PYTHON"] = python_path
os.environ["PYSPARK_DRIVER_PYTHON"] = python_path



from pyspark.sql import SparkSession

# Set up spark session
spark = SparkSession.builder \
    .appName("FacebookAdsAnalysis") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://master.data7201.emr:8020") \
    .config("spark.hadoop.conf.dir", "/etc/hadoop/conf") \
    .config("spark.python.worker.reuse", "true") \
    .config("spark.sql.execution.arrow.pyspark.enabled", "true") \
    .config("spark.executorEnv.PYSPARK_PYTHON", python_path) \
```

```
        .config("spark.executorEnv.PYSPARK_DRIVER_PYTHON", python_path) \
        .config("spark.network.timeout", "800s")\
        .config("spark.executor.heartbeatInterval", "120s")\
        .config("spark.driver.memory", "4g") \
        .config("spark.executor.memory", "4g") \
        .config("spark.memory.fraction", "0.8") \
        .getOrCreate()

    # Read data
    df =
spark.read.json("hdfs://master.data7201.emr:8020/data/ProjectDatasetFacebookAU/
")
```

See structure

```
df.printSchema()
```

See how many features(columns)

```
print(len(df.columns))
```

Inspect first few rows

```
df.show(5, truncate= False, vertical= True)
```

Show the first Non-NULL values in each feature for example

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

def get_first_non_null_value(df, column): # Filter out rows where the column is not NULL and select the first value non_null_df = df.filter(col(column).isNotNull()).select(column).limit(1) if non_null_df.count() > 0: return non_null_df.first()[column] else: return None

List of columns to check

columns_to_check = df.columns

Dictionary to store the first non-null value for each column

first_non_null_values = {}

for column in columns_to_check: first_value = get_first_non_null_value(df, column) first_non_null_values[column] = first_value

Show the results

for column, value in first_non_null_values.items(): print(f"The first non-null value for '{column}' is: {value}")

Summary of statistics

df.summary().show(truncate= False, vertical= True)

from pyspark.sql.functions import col, count

Counting non-nulls in each column

non_null_counts = df.select([count(col(c)).alias(c) for c in df.columns])

non_null_counts.show(truncate=False, vertical=True)

Feature count (Horizontal view)

import pandas as pd import matplotlib.pyplot as plt

Ensure Spark DataFrame is manageable in size before conversion

sample_df = non_null_counts.limit(100).toPandas()

Transpose the DataFrame for better axis alignment

transposed_df = sample_df.T transposed_df.columns = ['Non-Null Counts'] # Naming the column

Plotting

plt.figure(figsize=(14, 8)) # Adjust figure size to fit all feature names ax = transposed_df.plot(kind='barh', legend=None, figsize=(12, 6)) # Use horizontal bars ax.set_title('Count of Non-Null Values Per Feature') ax.set_ylabel('Feature Name') ax.set_xlabel('Number of Non-Null Values') plt.yticks(rotation=0) # No rotation plt.xticks() # Ensure x-ticks are appropriate

Disable scientific notation for x-axis (better visual)

ax.get_xaxis().get_major_formatter().set_scientific(False)

plt.tight_layout() # Adjust layout to make room for label text plt.savefig('feature_count (horizontal).png') # Save the figure for future use plt.show()

Feature count (Vertical view)

import pandas as pd import matplotlib.pyplot as plt

Ensure Spark DataFrame is manageable in size before conversion

sample_df = non_null_counts.limit(100).toPandas()

Plotting

plt.figure(figsize=(14, 8)) # Adjust figure size to make room for x-axis labels ax = sample_df.T.plot(kind='bar', legend=None, figsize=(12, 6)) # Transpose for better readability ax.set_title('Count of Non-Null Values Per Feature') ax.set_xlabel('Feature name') ax.set_ylabel('Number of feaures') plt.xticks(rotation=45, ha='right') # Rotate labels for better readability and adjust alignment plt.yticks() # Ensure y-ticks are appropriate

Disable scientific notation for y-axis

ax.get_yaxis().get_major_formatter().set_scientific(False)

```
plt.tight_layout() # Adjust layout to make room for label text

plt.savefig('feature count (vertical).png') # Savefile

plt.show()

    # Reduce columns to reduce workload
    from pyspark.sql.functions import col

    # Filter out rows with NULL values in the ad_creative_body column
    filtered_df = df.filter(col("ad_creative_body").isNotNull())

    # List of columns needed for analysis
    columns_needed = ["ad_creation_time", "ad_creative_body",
"ad_delivery_start_time", "spend", "impressions"]

    # Select only the needed columns
    filtered_df = df.select(columns_needed)



    filtered_df.cache()  # Cache the filter dataset if it is used multiple times

    # Unpersist the main dataset as it's no longer needed
    df.unpersist()

    from pyspark.sql.functions import udf, col, size
    from pyspark.sql.types import StringType
```

```
from pyspark.ml.feature import Tokenizer, StopWordsRemover, CountVectorizer, IDF
from pyspark.ml.clustering import LDA

# Define a UDF to clean the text
def clean_text(text):
    if text is None:
        return ""
    import re
    # Convert to lower case
    text = text.lower()
    # Remove punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Strip whitespaces
    text = text.strip()
    return text

# Register the UDF
clean_text_udf = udf(clean_text, StringType())

# Apply the UDF to clean 'ad_creative_body' into 'cleaned_ad_body'
filtered_df = filtered_df.withColumn("cleaned_ad_body", clean_text_udf(col("ad_creative_body")))

# Tokenize the cleaned text
tokenizer = Tokenizer(inputCol="cleaned_ad_body", outputCol="words")
filtered_df = tokenizer.transform(filtered_df)

# Remove stopwords
```

```
    remover = StopWordsRemover(inputCol="words", outputCol="filtered_text")
    filtered_df = remover.transform(filtered_df)

    # Show the cleaned part to verify the preprocessing
    filtered_df.select("cleaned_ad_body", "words", "filtered_text").show(5,
truncate=False, vertical=True)

    # Drop the 'cleaned_ad_body' and 'words' columns after showing them
    filtered_df = filtered_df.drop("cleaned_ad_body", "words")

    # Ensure there are no empty filtered_text rows (due to all words being stopwords)
    filtered_df = filtered_df.filter(size(col("filtered_text")) > 0)



   # Sample a smaller subset
    sampled_df = filtered_df.sample(fraction=0.0001, seed=66)

  # Define the path where you want to save the JSON file
   json_path =' projeca/sample_data.json'

# Save the DataFrame to JSON
sampled_df.write.json(json_path, mode='overwrite')

print(f"Sampled data saved to {json_path}")

    # Topic analysis
    from pyspark.ml.feature import Tokenizer, StopWordsRemover, CountVectorizer,
IDF
    from pyspark.ml.clustering import LDA
```

```
from pyspark import StorageLevel

# Count Vectorizer
cv = CountVectorizer(inputCol="filtered_text", outputCol="rawFeatures")
cv_model = cv.fit(sampled_df)
featurized_data = cv_model.transform(sampled_df)

# IDF
idf = IDF(inputCol="rawFeatures", outputCol="features")
idf_model = idf.fit(featurized_data)
rescaled_data = idf_model.transform(featurized_data)

# Persist with MEMORY_AND_DISK storage level
rescaled_data.persist(StorageLevel.MEMORY_AND_DISK)

# Number of topics
num_topics = 5

# Train an LDA model with a iteration for faster results
lda = LDA(k=num_topics, maxIter=1)
lda_model = lda.fit(rescaled_data)

# Show the topics found by the LDA model
topics = lda_model.describeTopics(maxTermsPerTopic=10)
topics.show(truncate=False)

# Map topics to words using the vocabulary
vocab = cv_model.vocabulary
```

```
    mapped_topics = topics.rdd.map(lambda row: (row['topic'], [vocab[idx] for idx in
row['termIndices']])).collect()
    for topic, words in mapped_topics:
        print(f"Topic {topic}: {words}")

    # Unpersist DataFrame to free up memory
    rescaled_data.unpersist()

    # Volume overtime
    import matplotlib.pyplot as plt
    import pandas as pd
    from pyspark.sql.functions import explode, array, lit, to_timestamp

    # Add a timestamp column based on 'ad_creation_time' for aggregation
    filtered_df = filtered_df.withColumn("timestamp",
to_timestamp("ad_creation_time", "yyyy-MM-dd"))

    # Extract topic distribution into an explodable format
    topic_distribution = lda_model.transform(rescaled_data).select("timestamp",
"topicDistribution")

    # Create a column for each topic with its respective distribution
    for i in range(num_topics):
        topic_distribution = topic_distribution.withColumn(f"topic_{i}",
topic_distribution["topicDistribution"].getItem(i))

    # Drop the original distribution column
    topic_distribution = topic_distribution.drop("topicDistribution")

    # Aggregate the topic data by timestamp
```

```
    time_series_data = topic_distribution.groupBy("timestamp").sum(*(f"topic_{i}" for i
in range(num_topics)))

    # Convert to Pandas DataFrame for plotting
    time_series_pd = time_series_data.toPandas()

    # Ensure timestamp column is a datetime type
    time_series_pd['timestamp'] = pd.to_datetime(time_series_pd['timestamp'])

    # Set timestamp as the index
    time_series_pd.set_index('timestamp', inplace=True)

    # Plotting
    plt.figure(figsize=(12, 6))
    for i in range(num_topics):
        plt.plot(time_series_pd.index, time_series_pd[f'sum(topic_{i})'], label=f'Topic {i}')

    plt.xlabel('Time')
    plt.ylabel('Topic Volume')
    plt.title('Topic Volume Over Time')
    plt.legend()
    plt.grid(True)

    # Save the plot as a PNG file
    plt.savefig('topic_volume_over_time.png')
    plt.show()

    # Word Cloud Visualization
    import matplotlib.pyplot as plt
```

```
from wordcloud import WordCloud

# Map topics to words using the vocabulary
vocab = cv_model.vocabulary
mapped_topics = topics.rdd.map(lambda row: (row['topic'], [vocab[idx] for idx in
row['termIndices']])).collect()

# Generate word clouds for each topic
for i, topic in enumerate(mapped_topics):
    plt.figure(figsize=(10, 5))
    wc = WordCloud(width=800, height=400, background_color='white').generate("
".join(topic[1]))
    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'Topic {i + 1}')
    plt.axis("off")
    plt.savefig(f'WordCloud_Topic_{i + 1}.png')  # Save each word cloud with a
unique filename
    plt.show()

#Demographic Distribution

import matplotlib.pyplot as plt
import pandas as pd

# Analyze demographic distribution
demographic_distribution =
filtered_df.groupBy("demographic_distribution").sum(*(f"topic_{i}" for i in
range(num_topics)))

# Convert to Pandas DataFrame for plotting
```

```
    demographic_distribution_pd = demographic_distribution.toPandas()

    # Ensure the demographic_distribution column is treated as a string
    demographic_distribution_pd['demographic_distribution'] =
demographic_distribution_pd['demographic_distribution'].astype(str)

    # Plotting
    plt.figure(figsize=(12, 6))
    width = 0.35  # Bar width
    ind = range(len(demographic_distribution_pd['demographic_distribution']))  # the x
locations for the groups

    # Plot bars for each topic
    for i in range(num_topics):
        plt.bar(ind, demographic_distribution_pd[f'sum(topic_{i})'], width, label=f'Topic
{i}', alpha=0.7)

    plt.xlabel('Demographic Distribution')
    plt.ylabel('Topic Volume')
    plt.title('Topic Volume by Demographic Distribution')
    plt.xticks(ind, demographic_distribution_pd['demographic_distribution'],
rotation='vertical')
    plt.legend()
    plt.grid(True)

    # Save the plot as a PNG file
    plt.savefig('topic_volume_by_demographic_distribution.png')
    plt.show()

    # Analyze spending and impressions
```

```
spending_impact = filtered_df.groupBy("spend").agg(
    spark_sum("impressions").alias("total_impressions"),
    *(spark_sum(f"topic_{i}").alias(f"sum_topic_{i}") for i in range(num_topics))
)

# Convert to Pandas DataFrame for plotting
spending_impact_pd = spending_impact.toPandas()

# Ensure the spend column is treated appropriately, converting to string if necessary
spending_impact_pd['spend'] = spending_impact_pd['spend'].astype(str)

# Plotting
plt.figure(figsize=(12, 6))

# Plot topic volumes with different markers for clarity
markers = ['o', 'v', '^', '<', '>', 's', 'p', '*', 'h', 'H', 'D', 'd', 'P', 'X']

for i in range(num_topics):
    plt.plot(spending_impact_pd['spend'], spending_impact_pd[f'sum_topic_{i}'], marker=markers[i % len(markers)], label=f'Topic {i}')

# Plot total impressions
plt.plot(spending_impact_pd['spend'], spending_impact_pd['total_impressions'], label='Total Impressions', linestyle='--', marker='x')

plt.xlabel('Spend')
plt.ylabel('Volume')
plt.title('Topic Volume and Impressions by Spend')
plt.legend()
```

```
plt.grid(True)

# Save the plot as a PNG file
plt.savefig('topic_volume_and_impressions_by_spend.png')
plt.show()
```