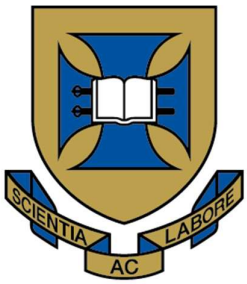


THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Political Analysis of Facebook Ads: A Big Data Approach

DATA7201

Project

Anmol Arora

47945209

anmol.arora@uqconnect.edu.au

School of Electrical Engineering and Computer Science

University of Queensland

Submitted for the degree of Master of Data Science

20th May 2024

Structured Abstract

Title:

Political Analysis of Facebook Ads: A Big Data Approach

Objectives:

To examine Facebooks political advertising trends during the 2022 Australian Federal Election.

Methods:

- **Data Pre-Processing:** Used PySpark and ‘pandas’ to load data, handle null values and remove duplicates.
- **Data Analysis:** Using PySpark and libraries such as ‘pandas’ and ‘matplotlib’ looked at various interesting questions using graphs and visualizations such as: URL analysis in ads, usage of hashtags in ads etc. during the 2022 Federal Election.

Analysis:

Conducted analysis on these topics:

1. Ad Volume
2. Demographic Targeting
3. Ad Durations
4. Ad Expenditures
5. Hashtag Usage
6. Popular URLs
7. Ad Impressions

Conclusion:

Focus on ads is a big factor that can not be ignored during big political events such as the Australian Federal Election.

Table of Contents

1 Introduction

1.1 Background

The technique of looking through huge and varied data sets to find hidden patterns, unidentified relationships, market trends, consumer preferences, and other important insights is known as big data analytics. To handle and analyze big data successfully, one must use complex tools and techniques due to its immense volume, velocity, and diversity. Organisations can improve strategic planning, streamline operations, and make data-driven choices with the help of big data analytics.

1.2 Motivation

Due to traditional data processing systems' inability to handle large datasets, distributed system solutions are required in big data analytics. Distributed systems, like Spark and Hadoop, split up the data processing work among several nodes, enabling effective computing and parallel processing. Distributed data processing is made possible, for instance, by Hadoop's MapReduce programming style and scalable and dependable HDFS (Hadoop Distributed File System). Large-scale data processing and machine learning applications can benefit from Apache Spark's in-memory processing capabilities, which speed up data analysis operations and facilitate iterative techniques.

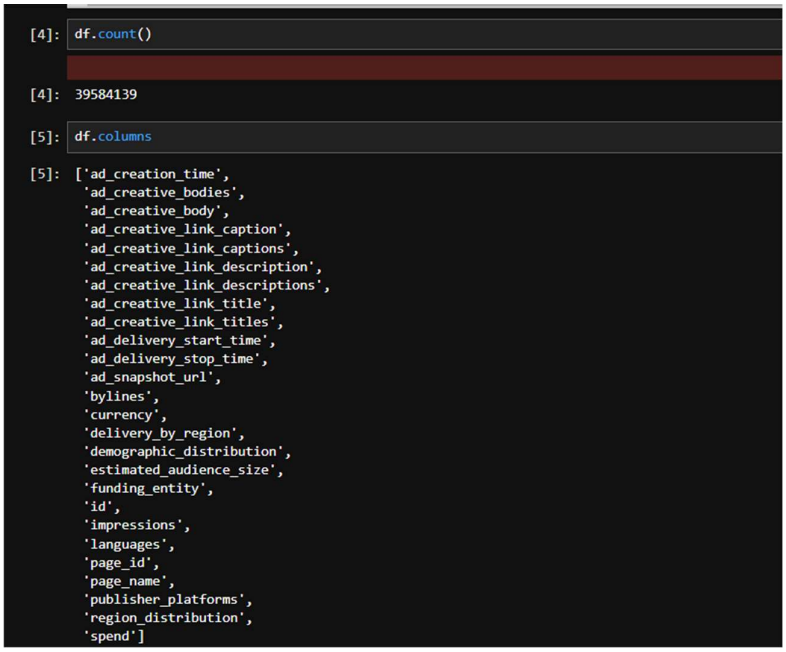
2 Dataset Analytics

2.1 Main Focus of the Analysis

In this project, I mainly focused on identifying political patterns or trends in the dataset of Facebook ads provided to us and the questions that I answered, mainly revolved around the 2022 Australian Federal Election. Gaining an understanding of several topics was the goal for me, such as the duration of ads over time, most famous hashtags present in ads, ad volume while taking into consideration the main issues in Australia etc.

2.2 Description of the Dataset Used

The Facebook Ad Library API was used to gather the dataset, which is given in JSON format. Every 12 hours or more often, a snapshot of the then current ad campaigns was captured and stored in a JSON file. As a result, for campaigns that lasted longer than 12 hours, the dataset contained duplicate items, which had to be fixed during the pre-processing step. The dataset contains advertisements aimed at Australian consumers from March 2020 to February 2024 and it has 26 columns and 39,584,139 rows.



2.3 Data Pre-Processing

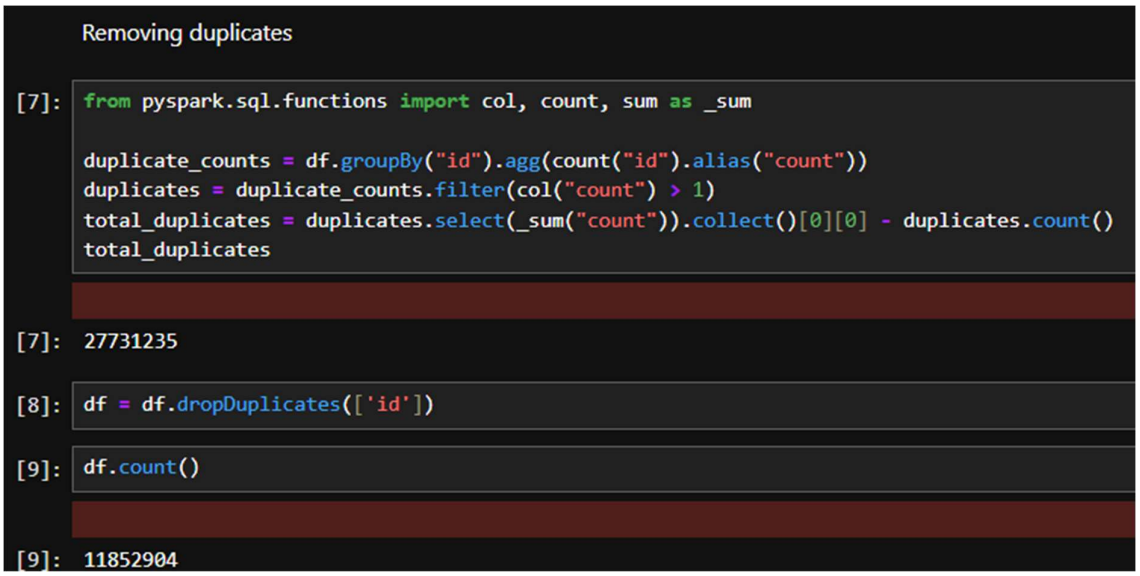
2.3.1 Loading the Dataset

Firstly, I loaded the dataset from the HDFS using PySpark. PySpark, which is the Python API for Apache Spark, helps in managing large JSON files and do pre-processing on big data.

```
# Load the dataset from HDFS
file_path = "/data/ProjectDatasetFacebookAU/*"
df = spark.read.json(file_path)
```

2.3.2 Removing Duplicates

To guarantee that each advertising campaign was represented just once, duplicate entries were found and eliminated using the unique identifier ‘id’. For an accurate analysis and to prevent biased results and support the integrity of the findings, this step is essential.

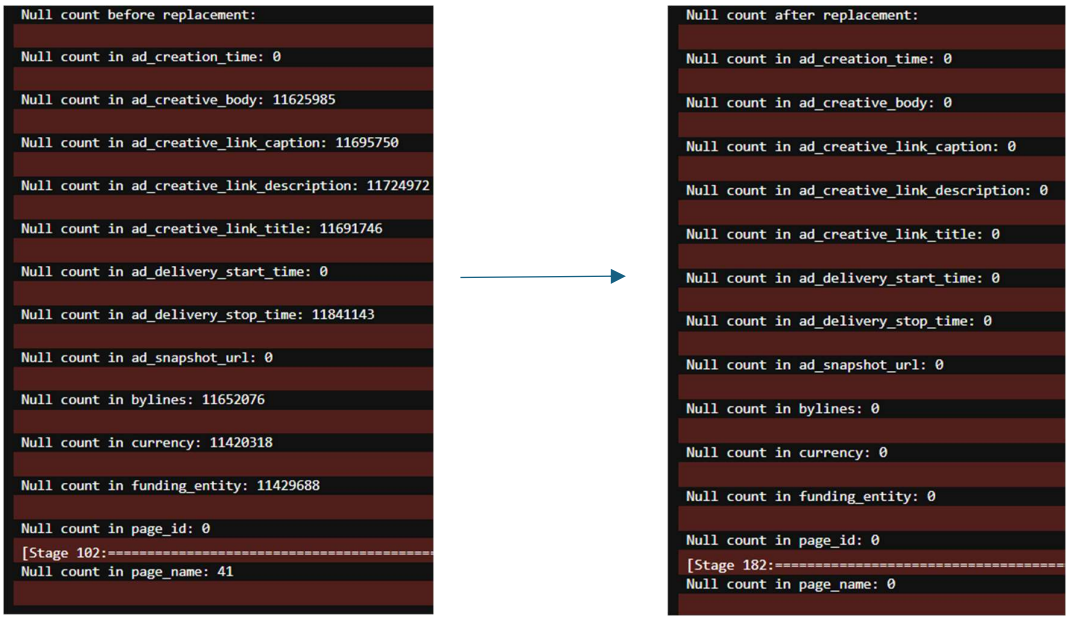


After, removing duplicates, I was only left with 11,852,904 rows, which means that originally almost 2/3rd (27,731,235 rows) of my data was duplicates.

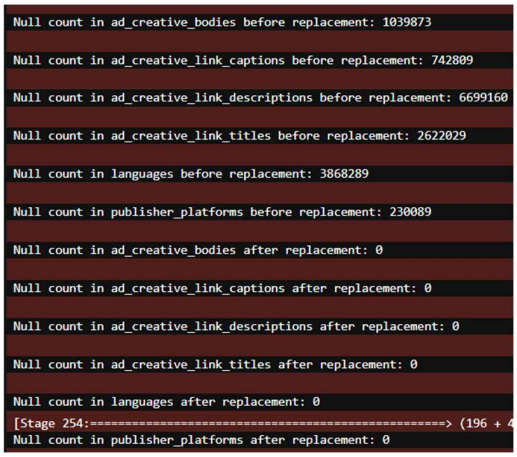
2.3.3 Handling Null Values

I handled Null values in 3 steps:

- Handling Null values in ‘string’ columns: To maintain consistency, null values in string columns were substituted with empty strings. By using this method, data processing and analysis problems are avoided, and non-null data was included in text fields such as ‘ad_creative_body’.



- Handling Null values in ‘array’ columns: Array columns containing null values were substituted with empty arrays. This guarantees error-free execution of list-based operations, preserving data integrity.



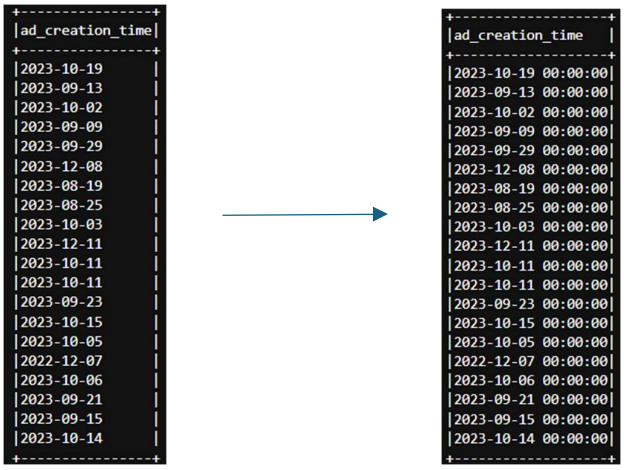
- Handling Null values in ‘struct’ columns: Null values in nested fields for struct columns were substituted with default values. By doing this, precise nested enquiries and analysis can still be carried out.

```
Null count in estimated_audience_size before replacement: 11651782

Null count in impressions before replacement: 11420318

Null count in spend before replacement: 11420318

Null count in demographic_distribution before replacement: 11486308

Null count in region_distribution before replacement: 11491738

Null count in estimated_audience_size after replacement: 0

Null count in impressions after replacement: 0

Null count in spend after replacement: 0

Null count in demographic_distribution after replacement: 0
[Stage 312:=================================================>(901
Null count in region_distribution after replacement: 0
```

2.3.4 Data Transformation

I transformed 3 columns: 'ad_creation_time', 'ad_delivery_start_time' and 'ad_delivery_stop_time', to timestamp format to ensure correct time-based analysis.

Apart from this, some columns were transformed into Double data type for higher accuracy because making sure that all numerical columns are in Double format helps to standardize the dataset.



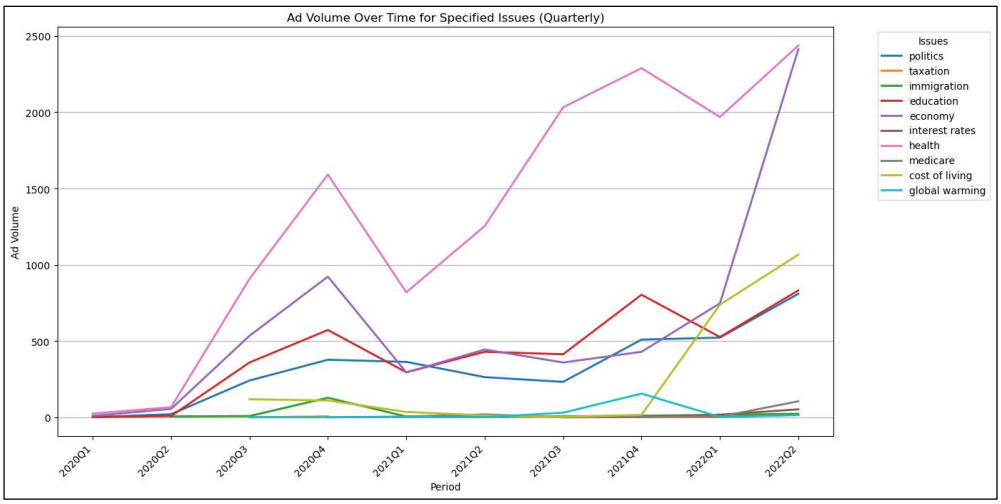
ad_creation_time
2023-10-19
2023-09-13
2023-10-02
2023-09-09
2023-09-29
2023-12-08
2023-08-19
2023-08-25
2023-10-03
2023-12-11
2023-10-11
2023-10-11
2023-09-23
2023-10-15
2023-10-05
2022-12-07
2023-10-06
2023-09-21
2023-09-15
2023-10-14

ad_creation_time
2023-10-19 00:00:00
2023-09-13 00:00:00
2023-10-02 00:00:00
2023-09-09 00:00:00
2023-09-29 00:00:00
2023-12-08 00:00:00
2023-08-19 00:00:00
2023-08-25 00:00:00
2023-10-03 00:00:00
2023-12-11 00:00:00
2023-10-11 00:00:00
2023-10-11 00:00:00
2023-09-23 00:00:00
2023-10-15 00:00:00
2023-10-05 00:00:00
2022-12-07 00:00:00
2023-10-06 00:00:00
2023-09-21 00:00:00
2023-09-15 00:00:00
2023-10-14 00:00:00

2.4 Analytics Methods & Results

- Analyzing Ad Volume for Issues in Australia until the 2022 election results.

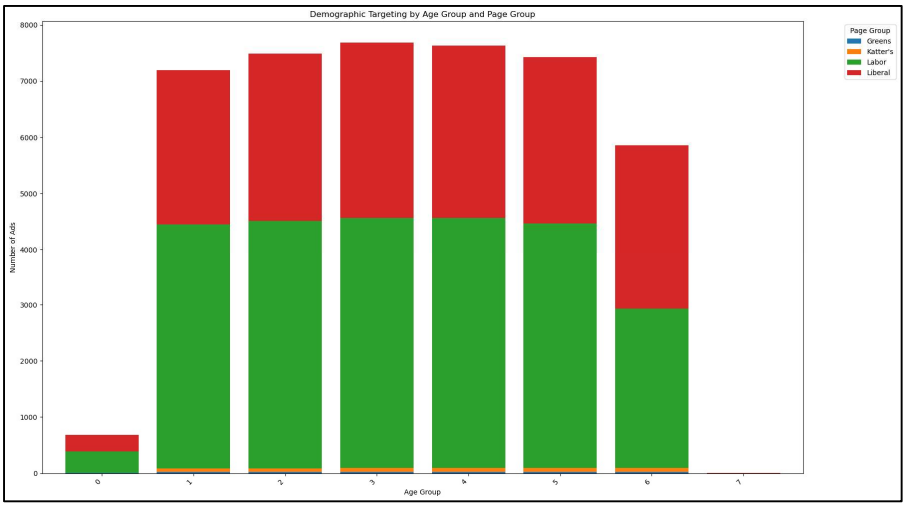
I analyzed ad volume over time for key issues, such as politics, economy, health, education etc., in Australia leading up to the 2022 election results.

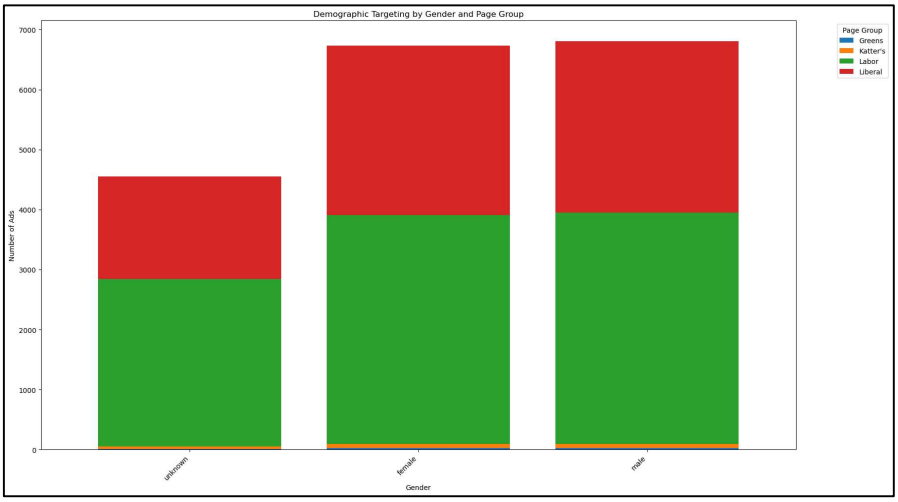


Leading up to the Voting Day for the election, 'Health' and 'Economy' were the most talked advertised issues spoken for/against on Facebook.

- Focusing on Political Party Accounts and their Target Demographics.

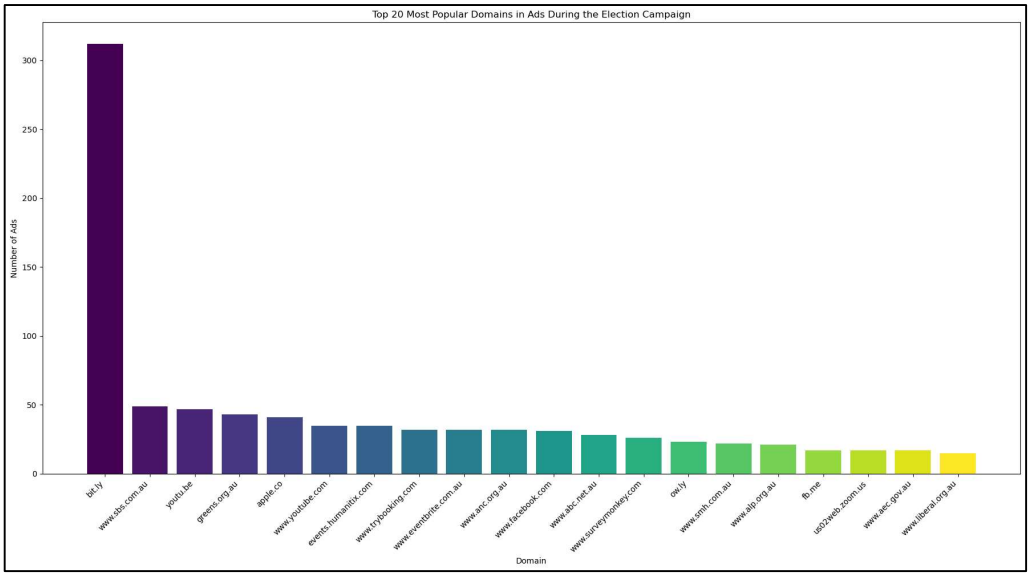
I focused on the 4 major parties' (Labor, Liberal, Greens and Katter's political parties) and their leaders' pages on Facebook and looked at what demographics they targeted based on Gender and Age groups.





I found out that the Australian Labor Party and its leader (current Prime Minister) Anthony Albanese, through their Facebook ads targeted people of almost all age groups & genders equally, while the Liberal Party of Australia and its leader, Scott Morrison came in close second in each demographic.

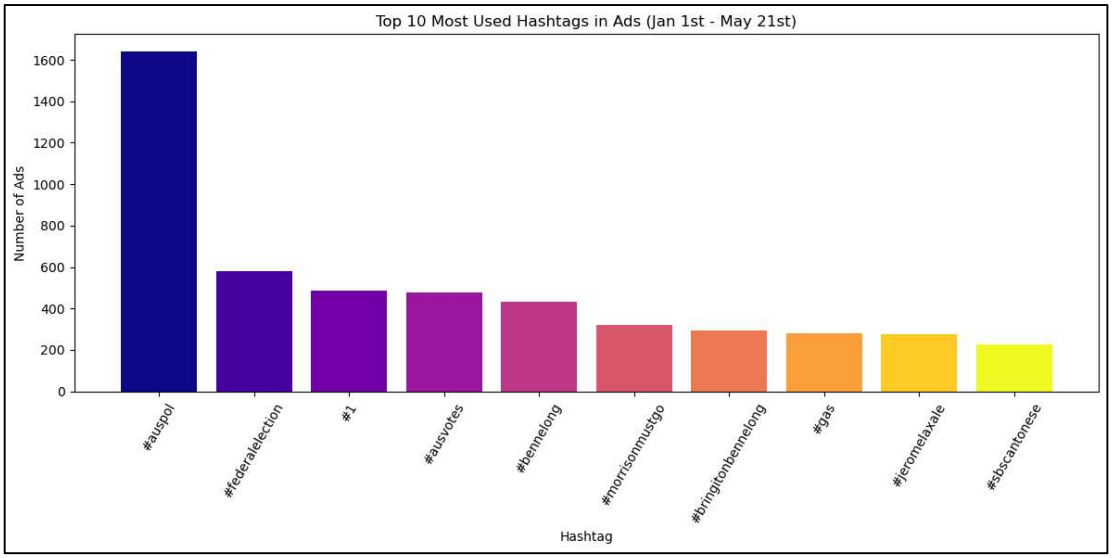
- Looking at URLs included in Ads to find the most Popular Internet Domains during the 2022 Election Campaign.

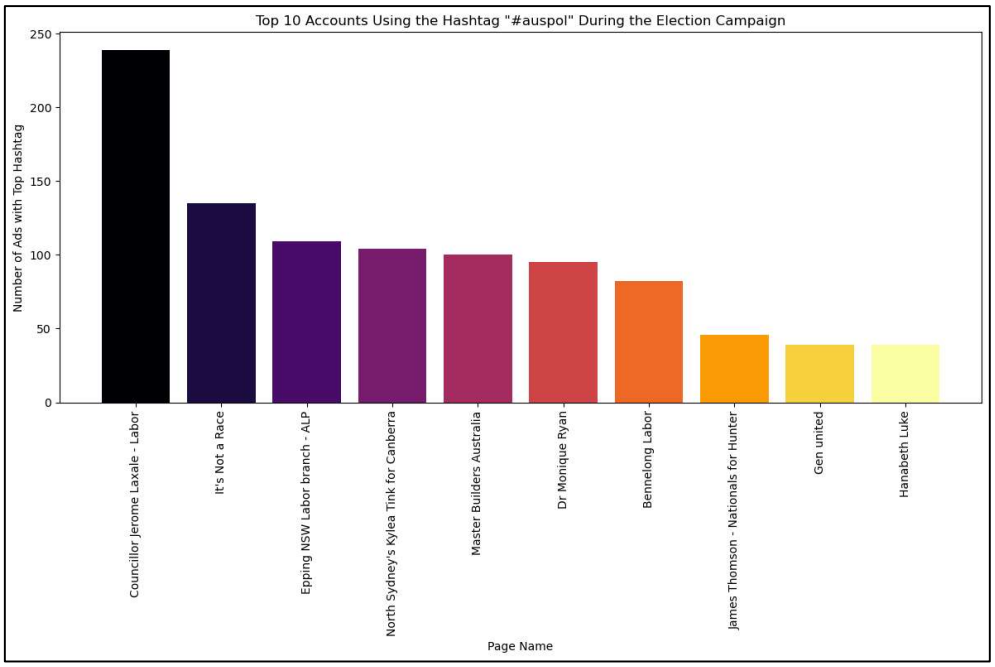


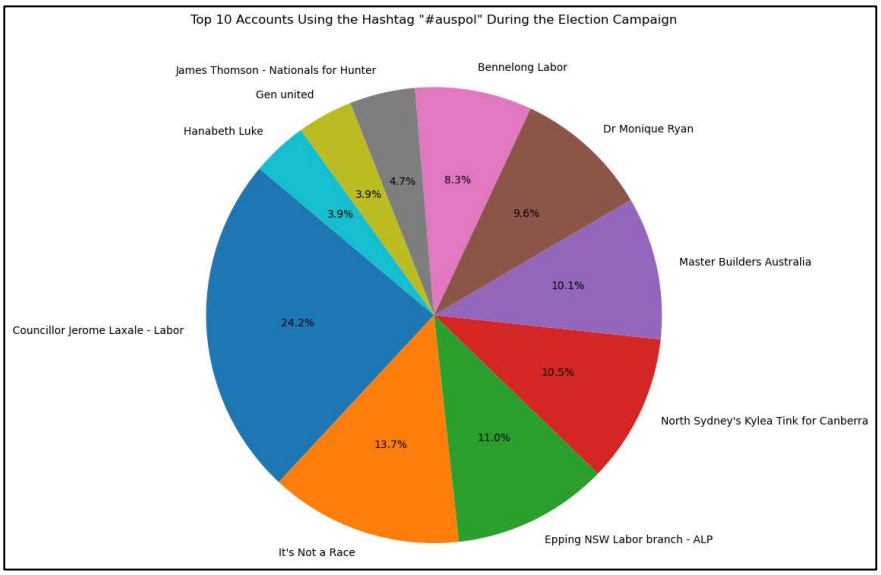
From the bar chart, we can see that the domain name appearing in most ads, by a large margin was ‘bit.ly’, which means that most links’ URLs were shortened using ‘bit.ly’ (a URL shortener).

- Looking at a Specific Hashtag & What Facebook Page used it the most.

I first plotted a bar-chart and found out that the most used hashtag in ads on Facebook in the same ~5 month-period (from 1st Jan 2022 to 21st May 2022) was '#auspol', followed by '#federalelection'.

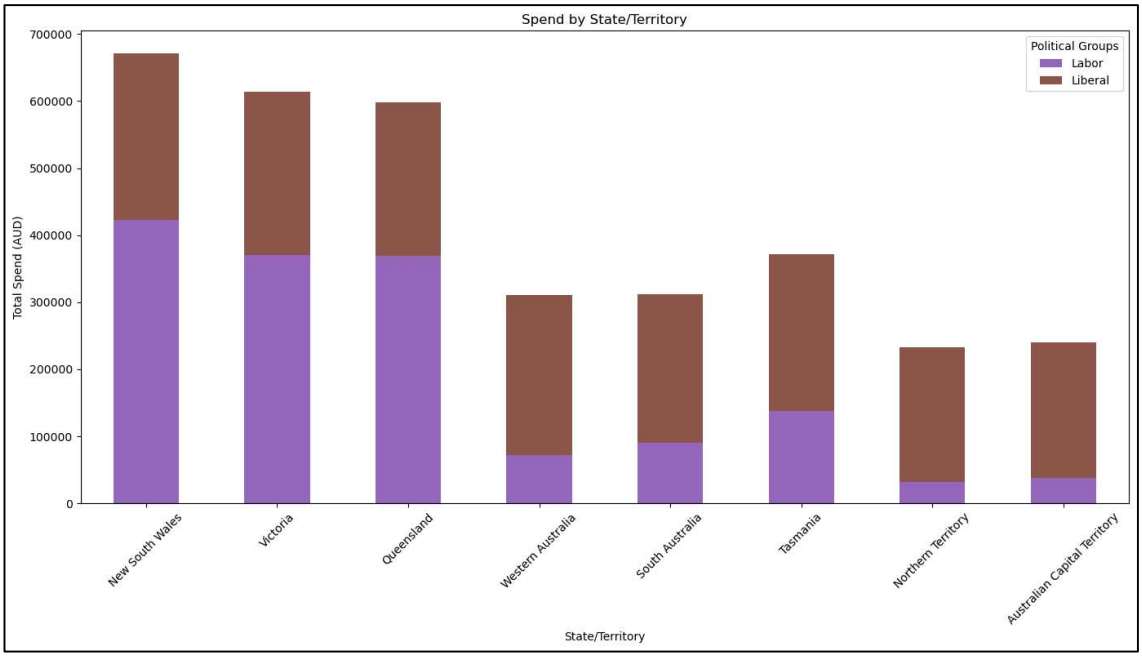






From the bar graph and pie chart we can deduce that the account named: 'Councillor Jarome Laxale – Labor', used the hashtag "#auspol" the most.

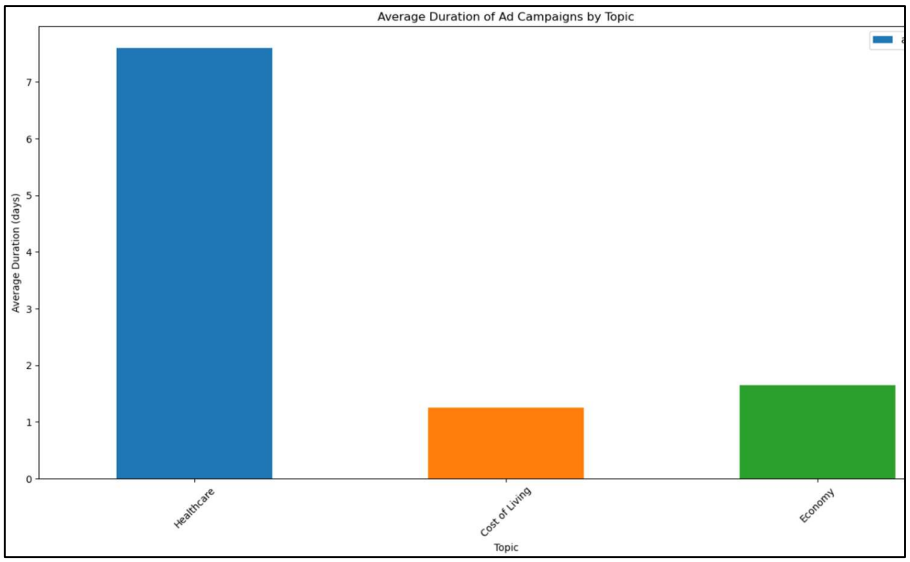
- Looking at the Spend by State/Territory during the 2022 Federal Election by the Two Major Parties' and Their Respective Leaders' Pages



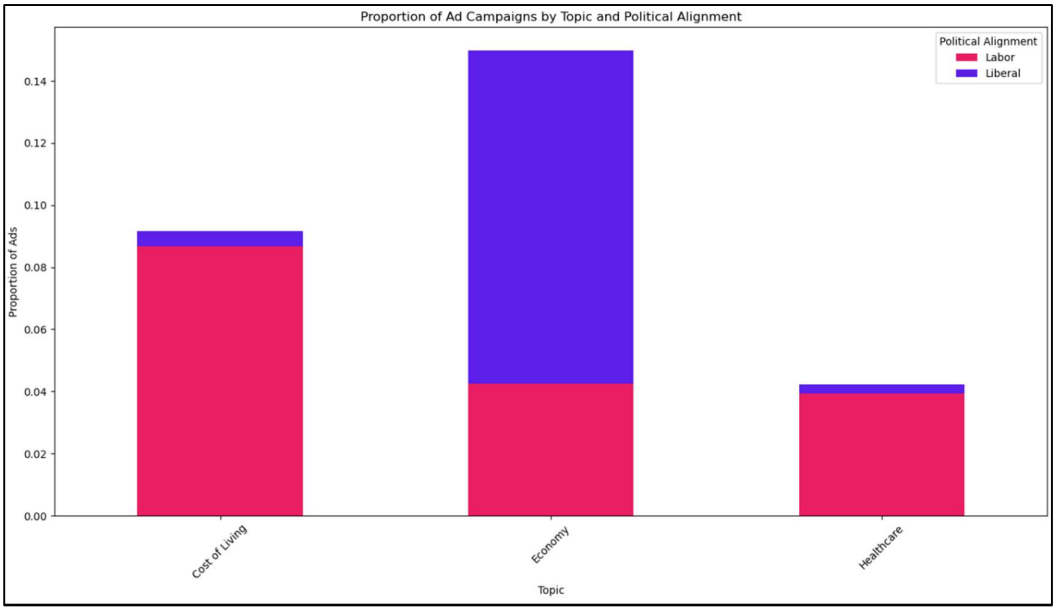
	State	Labor	Liberal
0	New South Wales	423000.0	248200.0
1	Victoria	370100.0	244000.0
2	Queensland	369400.0	229100.0
3	Western Australia	71600.0	239500.0
4	South Australia	90400.0	221700.0
5	Tasmania	138300.0	233300.0
6	Northern Territory	31400.0	201500.0
7	Australian Capital Territory	37700.0	202000.0

We can see from the bar graph that the Labor Party spent more money than the Liberal Party in the 3 main states of New South Wales ($ 423,000), Victoria ($ 370,100), and Queensland ($ 369,400), while the latter dominated in the other 3 states of Western Australia ($ 239,500), South Australia ($ 221,700) & Tasmania ($ 233,300) and the 2 Territories of Northern Territory ($ 201,500) & Australian Capital Territory ($ 202,000).

- Analysis of Ad Campaign Durations and Political Alignment Composition for Topics on Economy, Healthcare, and Cost of Living

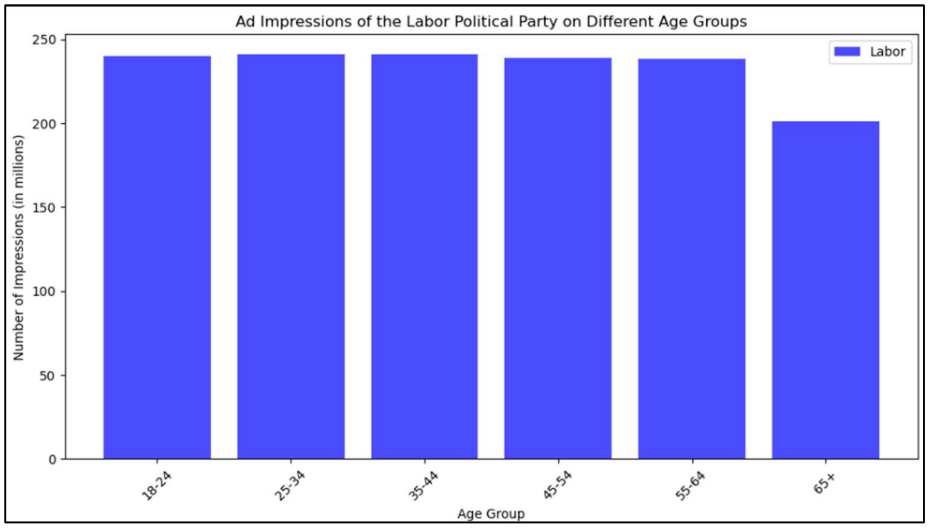


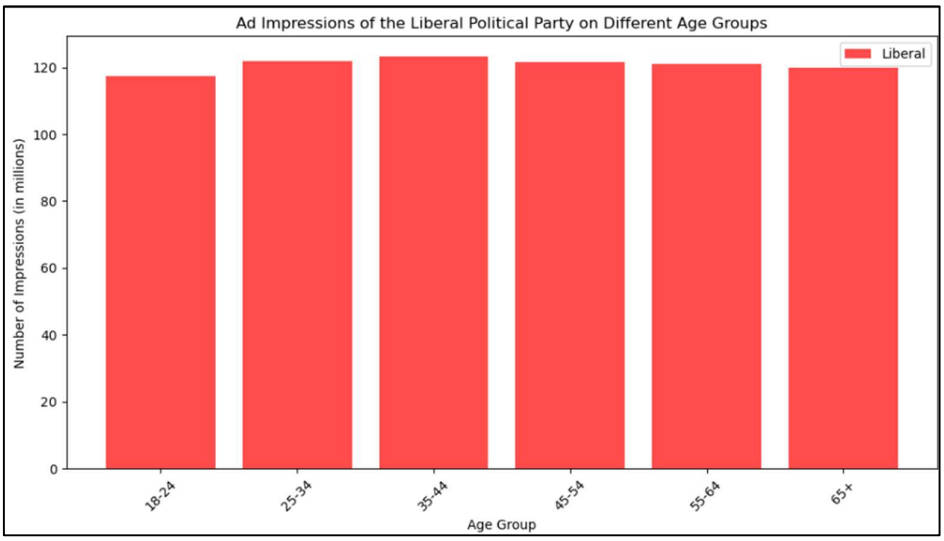
From the above bar chart, we observed that ads categorized as being related to ‘Healthcare’ stayed on for an average duration of 7.6 days, followed by ‘Economy ads’ at 1.653 days.



The bar chart demonstrates that, with a sizable number of ads from both parties, albeit mostly from Labor, ‘Economy’ was the most often mentioned issue. We can see that the Liberal Party had more ads on ‘Economy’ than the Labor Party, while the latter’s ads mostly were on the topics of ‘Healthcare’ & ‘Cost of Living’.

- Ad Impressions of Both Political Parties on Different Age Groups





All age categories had a constant 120 million impressions from the Liberal party's advertisements, while the ads for the Labor party received more impressions—roughly 250 million—in most age groups and about 200 million for people aged 65 and above.

2.5 Data Analytics Tools Used

1. PySpark: To handle big data efficiently, leveraging distributed computing.
2. Pandas: To manipulate the data and get it into a suitable format.
3. Matplotlib: To create plots representing analysis.
4. SQL: Queries were used to extract insights from the data.

3 Conclusion

3.1 Summary of Findings

- **Ad Volume**: Leading up to the 2022 election, the most talked-about concerns were the economy and health.
- **Demographic Targeting**: The Liberal Party had a consistent targeting approach, whereas the Labor Party focused on a wide variety of demographics.
- **Popular URLs**: Most popular domain in URLs found in ads was “bit.ly."
- **Hashtag Usage**: '#auspol' was the most popular hashtag, primarily used by pages connected to the Labor party.
- **Ad Spend**: While Liberals dominated in minor states and territories, Labor spent more in the big states (QLD, VIC, NSW).
- **Ad Durations**: The average length of healthcare advertisements was the longest among the 3 major talked-about issued around the 2022 election.
- **Ad Impressions**: Compared to Liberal ads, Labor ads had more impressions in most age categories.

3.2 Lessons Learned

Accurate analysis depends on using big data technologies and frameworks such as PySpark and performing efficient data preparation and visualizations.

3.3 Main Message

Important information about political advertising tactics may be found in big data analytics. These observations can direct future policy development and campaign strategy.

Word Count: 1501

Appendix

```
[1]: import os
     from pyspark.sql import SparkSession

     spark = SparkSession.builder \
         .appName("FacebookAdAnalysis") \
         .getOrCreate()

     # Load the dataset from HDFS
     file_path = "/data/ProjectDatasetFacebookAU/*"
     df = spark.read.json(file_path)
```

```
24/05/24 19:03:12 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
```

```
[2]: df.printSchema()
```

```
root
 |-- ad_creation_time: string (nullable = true)
 |-- ad_creative_bodies: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_body: string (nullable = true)
 |-- ad_creative_link_caption: string (nullable = true)
 |-- ad_creative_link_captions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_description: string (nullable = true)
 |-- ad_creative_link_descriptions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_title: string (nullable = true)
 |-- ad_creative_link_titles: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_delivery_start_time: string (nullable = true)
 |-- ad_delivery_stop_time: string (nullable = true)
```

```
 |-- ad_delivery_start_time: string (nullable = true)
 |-- ad_delivery_stop_time: string (nullable = true)
 |-- ad_snapshot_url: string (nullable = true)
 |-- bylines: string (nullable = true)
 |-- currency: string (nullable = true)
 |-- delivery_by_region: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- percentage: string (nullable = true)
 |    |    |-- region: string (nullable = true)
 |-- demographic_distribution: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- age: string (nullable = true)
 |    |    |-- gender: string (nullable = true)
 |    |    |-- percentage: string (nullable = true)
 |-- estimated_audience_size: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- funding_entity: string (nullable = true)
 |-- id: string (nullable = true)
 |-- impressions: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- languages: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- page_id: string (nullable = true)
 |-- page_name: string (nullable = true)
 |-- publisher_platforms: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- region_distribution: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- percentage: string (nullable = true)
 |    |    |-- region: string (nullable = true)
 |-- spend: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
```

```
[3]: df.describe().toPandas()
```

```
24/05/24 19:03:43 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by sett
ing 'spark.sql.debug.maxToStringFields'.
```

[3]:

	summary	ad_creation_time	ad_creative_body	ad_creative_link_caption	ad_creative_link_description	ad_creative_link_title	ad_delivery_start_time	ad_delivery_stop_
0	count	39584139	9568757	8249081	6875053	8327851	39584139	50
1	mean	None	None	None	None	2020.2068965517242	None	
2	stddev	None	None	None	None	0.4122508203948578	None	
3	min	2019-12-26		"Prospective Issues, 2021" with Ross Cameron	\n		2019-12-26	202 29T00:50:59+
4	max	2024-03-21	🥝 Have you had ENOUGH of the hs, lack of clim...	普通话性别平等问答 Mandarin Q & A on Gender Equality	🌾 Gentle on your hair & our planet 🌱	🌾 Use code RETURN15 to get 15% Off now 🌐	2024-03-21	2024-0

```
[4]: df.count()
```

```
[4]: 39584139
```

```
[5]: df.columns
```

```
[5]: ['ad_creation_time',
      'ad_creative_bodies',
      'ad_creative_body',
      'ad_creative_link_caption',
      'ad_creative_link_captions',
      'ad_creative_link_description',
      'ad_creative_link_descriptions',
      'ad_creative_link_title',
      'ad_creative_link_titles',
      'ad_delivery_start_time',
      'ad_delivery_stop_time',
      'ad_snapshot_url',
      'bylines',
      'currency',
      'delivery_by_region',
      'demographic_distribution',
      'estimated_audience_size',
      'funding_entity',
      'id',
      'impressions',
      'languages',
      'page_id',
      'page_name',
      'publisher_platforms',
      'region_distribution',
      'spend']
```

```
[6]: len(df.columns)
```

```
[6]: 26
```

Removing duplicates

```
[7]: from pyspark.sql.functions import col, count, sum as _sum

     duplicate_counts = df.groupBy("id").agg(count("id").alias("count"))
     duplicates = duplicate_counts.filter(col("count") > 1)
     total_duplicates = duplicates.select(_sum("count")).collect()[0][0] - duplicates.count()
     total_duplicates
```

```
[7]: 27731235
```

```
[8]: df = df.dropDuplicates(['id'])
```

```
[9]: df.count()
```

```
[9]: 11852904
```

```
[11]: df.select("ad_creation_time").show(20, truncate=False)
```

```
[Stage 27:=====================================================>(907 + 1) / 908]
+----------------+
|ad_creation_time|
+----------------+
|2023-10-19      |
|2023-09-13      |
|2023-10-02      |
|2023-09-09      |
|2023-09-29      |
|2023-12-08      |
|2023-08-19      |
|2023-08-25      |
|2023-10-03      |
|2023-12-11      |
|2023-10-11      |
|2023-10-11      |
|2023-09-23      |
|2023-10-15      |
|2023-10-05      |
|2022-12-07      |
|2023-10-06      |
|2023-09-21      |
|2023-09-15      |
|2023-10-14      |
+----------------+
only showing top 20 rows
```

Handling Null Values in String Columns

[12]:
```
from pyspark.sql.functions import col, count, when, to_timestamp

# string columns
string_columns = [
    'ad_creation_time', 'ad_creative_body', 'ad_creative_link_caption',
    'ad_creative_link_description', 'ad_creative_link_title',
    'ad_delivery_start_time', 'ad_delivery_stop_time', 'ad_snapshot_url',
    'bylines', 'currency', 'funding_entity', 'page_id', 'page_name'
]

print("Null count before replacement:")
for column in string_columns:
    null_count = df.select(count(when(col(column).isNull(), column)).alias("Null_Count")).collect()[0]["Null_Count"]
    print(f"Null count in {column}: {null_count}")
```

```
Null count before replacement:

Null count in ad_creation_time: 0

Null count in ad_creative_body: 11625985

Null count in ad_creative_link_caption: 11695750

Null count in ad_creative_link_description: 11724972

Null count in ad_creative_link_title: 11691746

Null count in ad_delivery_start_time: 0

Null count in ad_delivery_stop_time: 11841143

Null count in ad_snapshot_url: 0

Null count in bylines: 11652076

Null count in currency: 11420318

Null count in funding_entity: 11429688

Null count in page_id: 0
[Stage 102:=====================================================>(901 + 7) / 908]
Null count in page_name: 41
```

[13]:
```
string_columns_with_nulls = [
    'ad_creative_body', 'ad_creative_link_caption',
    'ad_creative_link_description', 'ad_creative_link_title',
    'ad_delivery_stop_time', 'bylines', 'currency',
    'funding_entity', 'page_name'
]
```

```
for column in string_columns_with_nulls:
    df = df.withColumn(column, when(col(column).isNull(), "").otherwise(col(column)))

print("Null count after replacement:")
for column in string_columns:
    null_count = df.select(count(when(col(column).isNull(), column)).alias("Null_Count")).collect()[0]["Null_Count"]
    print(f"Null count in {column}: {null_count}")
```

```
Null count after replacement:

Null count in ad_creation_time: 0

Null count in ad_creative_body: 0

Null count in ad_creative_link_caption: 0

Null count in ad_creative_link_description: 0

Null count in ad_creative_link_title: 0

Null count in ad_delivery_start_time: 0

Null count in ad_delivery_stop_time: 0

Null count in ad_snapshot_url: 0

Null count in bylines: 0

Null count in currency: 0
```

```
Null count in currency: 0

Null count in funding_entity: 0

Null count in page_id: 0
[Stage 182:==========================================>       (172 + 28) / 200]
Null count in page_name: 0
```

Handling Null Values in Array Columns

[14]:
```python
from pyspark.sql.functions import col, count, when, array

# array columns
array_columns = [
    'ad_creative_bodies', 'ad_creative_link_captions',
    'ad_creative_link_descriptions', 'ad_creative_link_titles',
    'languages', 'publisher_platforms'
]

for column in array_columns:
    null_count = df.select(count(when(col(column).isNull(), column)).alias("Null_Count")).collect()[0]["Null_Count"]
    print(f"Null count in {column} before replacement: {null_count}")

for column in array_columns:
    df = df.withColumn(column, when(col(column).isNull(), array()).otherwise(col(column)))
```

```python
for column in array_columns:
    null_count = df.select(count(when(col(column).isNull(), column)).alias("Null_Count")).collect()[0]["Null_Count"]
    print(f"Null count in {column} after replacement: {null_count}")
```

```
Null count in ad_creative_bodies before replacement: 1039873

Null count in ad_creative_link_captions before replacement: 742809

Null count in ad_creative_link_descriptions before replacement: 6699160

Null count in ad_creative_link_titles before replacement: 2622029

Null count in languages before replacement: 3868289

Null count in publisher_platforms before replacement: 230089

Null count in ad_creative_bodies after replacement: 0

Null count in ad_creative_link_captions after replacement: 0

Null count in ad_creative_link_descriptions after replacement: 0

Null count in ad_creative_link_titles after replacement: 0

Null count in languages after replacement: 0
[Stage 254:=====================================================> (196 + 4) / 200]
Null count in publisher_platforms after replacement: 0
```

Handling Null Values in Struct Columns

[15]:
```python
from pyspark.sql.functions import col, count, when, struct, lit, array

# struct and array<struct> columns
struct_columns = [
    'estimated_audience_size', 'impressions', 'spend',
    'demographic_distribution', 'region_distribution'
]

for column in struct_columns:
    null_count = df.select(count(when(col(column).isNull(), column)).alias("Null_Count")).collect()[0]["Null_Count"]
    print(f"Null count in {column} before replacement: {null_count}")

default_estimated_audience_size = struct(lit("0").alias("lower_bound"), lit("0").alias("upper_bound"))
default_impressions = struct(lit("0").alias("lower_bound"), lit("0").alias("upper_bound"))
default_spend = struct(lit("0").alias("lower_bound"), lit("0").alias("upper_bound"))

df = df.withColumn("estimated_audience_size", when(col("estimated_audience_size").isNull(), default_estimated_audience_size).otherwise(col("estimated_a
df = df.withColumn("impressions", when(col("impressions").isNull(), default_impressions).otherwise(col("impressions")))
df = df.withColumn("spend", when(col("spend").isNull(), default_spend).otherwise(col("spend")))

default_demographic = struct(lit("").alias("age"), lit("").alias("gender"), lit("0.0").alias("percentage"))
default_region = struct(lit("0.0").alias("percentage"), lit("").alias("region"))

df = df.withColumn("demographic_distribution", when(col("demographic_distribution").isNull(), array(default_demographic)).otherwise(col("demographic_di
df = df.withColumn("region_distribution", when(col("region_distribution").isNull(), array(default_region)).otherwise(col("region_distribution")))

for column in struct_columns:
    null_count = df.select(count(when(col(column).isNull(), column)).alias("Null_Count")).collect()[0]["Null_Count"]
    print(f"Null count in {column} after replacement: {null_count}")
```

```
Null count in estimated_audience_size before replacement: 11651782

Null count in impressions before replacement: 11420318

Null count in spend before replacement: 11420318

Null count in demographic_distribution before replacement: 11486308

Null count in region_distribution before replacement: 11491738

Null count in estimated_audience_size after replacement: 0

Null count in impressions after replacement: 0

Null count in spend after replacement: 0

Null count in demographic_distribution after replacement: 0
[Stage 312:=====================================================>(901 + 7) / 908]
Null count in region_distribution after replacement: 0
```

[16]:
```
df.printSchema()
```

```
root
 |-- ad_creation_time: string (nullable = true)
 |-- ad_creative_bodies: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_body: string (nullable = true)
 |-- ad_creative_link_caption: string (nullable = true)
 |-- ad_creative_link_captions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_description: string (nullable = true)
 |-- ad_creative_link_descriptions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_title: string (nullable = true)
 |-- ad_creative_link_titles: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_delivery_start_time: string (nullable = true)
 |-- ad_delivery_stop_time: string (nullable = true)
 |-- ad_snapshot_url: string (nullable = true)
 |-- bylines: string (nullable = true)
 |-- currency: string (nullable = true)
 |-- delivery_by_region: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- percentage: string (nullable = true)
 |    |    |-- region: string (nullable = true)
 |-- demographic_distribution: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- age: string (nullable = true)
 |    |    |-- gender: string (nullable = true)
 |    |    |-- percentage: string (nullable = true)
 |-- estimated_audience_size: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- funding_entity: string (nullable = true)
 |-- id: string (nullable = true)
 |-- impressions: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- languages: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- page_id: string (nullable = true)
 |-- page_name: string (nullable = true)
 |-- publisher_platforms: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- region_distribution: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- percentage: string (nullable = true)
 |    |    |-- region: string (nullable = true)
 |-- spend: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
```

Converting some columns to timestamps

[17]:
```
from pyspark.sql.functions import to_timestamp

df = df.withColumn("ad_creation_time", to_timestamp(col("ad_creation_time"), "yyyy-MM-dd"))
df = df.withColumn("ad_delivery_start_time", to_timestamp(col("ad_delivery_start_time"), "yyyy-MM-dd"))
df = df.withColumn("ad_delivery_stop_time", to_timestamp(col("ad_delivery_stop_time"), "yyyy-MM-dd"))
```

[18]:
```
df.select("ad_creation_time").show(20, truncate=False)
```

```
[Stage 318:=====================================================>(902 + 6) / 908]
+-------------------+
|ad_creation_time   |
+-------------------+
|2023-10-19 00:00:00|
|2023-09-13 00:00:00|
|2023-10-02 00:00:00|
|2023-09-09 00:00:00|
|2023-09-29 00:00:00|
|2023-12-08 00:00:00|
|2023-08-19 00:00:00|
|2023-08-25 00:00:00|
|2023-10-03 00:00:00|
|2023-12-11 00:00:00|
|2023-10-11 00:00:00|
|2023-10-11 00:00:00|
|2023-09-23 00:00:00|
|2023-10-15 00:00:00|
|2023-10-05 00:00:00|
|2022-12-07 00:00:00|
|2023-10-06 00:00:00|
|2023-09-21 00:00:00|
|2023-09-15 00:00:00|
|2023-10-14 00:00:00|
+-------------------+
only showing top 20 rows
```

[19]:
```
df.select("ad_delivery_start_time").show(20, truncate=False)
```

```
[Stage 321:=====================================================>(907 + 1) / 908]
+----------------------+
|ad_delivery_start_time|
+----------------------+
|2023-10-19 00:00:00   |
|2023-09-13 00:00:00   |
|2023-10-02 00:00:00   |
|2023-09-10 00:00:00   |
|2023-09-29 00:00:00   |
|2023-12-08 00:00:00   |
|2023-08-20 00:00:00   |
|2023-08-26 00:00:00   |
|2023-10-03 00:00:00   |
|2023-12-12 00:00:00   |
|2023-10-11 00:00:00   |
|2023-10-11 00:00:00   |
|2023-09-23 00:00:00   |
|2023-10-16 00:00:00   |
|2023-10-06 00:00:00   |
|2022-12-07 00:00:00   |
|2023-10-07 00:00:00   |
|2023-09-21 00:00:00   |
|2023-09-15 00:00:00   |
|2023-10-14 00:00:00   |
+----------------------+
only showing top 20 rows
```

[20]:
```
df.select("ad_creation_time").show(20, truncate=False)
```

```
[Stage 324:=====================================================>(902 + 6) / 908]
+-------------------+
|ad_creation_time   |
+-------------------+
|2023-10-19 00:00:00|
|2023-09-13 00:00:00|
|2023-10-02 00:00:00|
|2023-09-09 00:00:00|
|2023-09-29 00:00:00|
|2023-12-08 00:00:00|
|2023-08-19 00:00:00|
|2023-08-25 00:00:00|
|2023-10-03 00:00:00|
|2023-12-11 00:00:00|
|2023-10-11 00:00:00|
|2023-10-11 00:00:00|
|2023-09-23 00:00:00|
|2023-10-15 00:00:00|
|2023-10-05 00:00:00|
|2022-12-07 00:00:00|
|2023-10-06 00:00:00|
|2023-09-21 00:00:00|
```

```
|2023-10-06 00:00:00|
|2023-09-21 00:00:00|
|2023-09-15 00:00:00|
|2023-10-14 00:00:00|
+-------------------+
only showing top 20 rows
```

Convert `spend` and `impressions` Fields to Double

[21]:
```python
from pyspark.sql.functions import col

df = df.withColumn("spend_lower_bound", col("spend.lower_bound").cast("double"))
df = df.withColumn("spend_upper_bound", col("spend.upper_bound").cast("double"))

df = df.withColumn("impressions_lower_bound", col("impressions.lower_bound").cast("double"))
df = df.withColumn("impressions_upper_bound", col("impressions.upper_bound").cast("double"))
```

[22]:
```python
df.printSchema()
```

```
root
 |-- ad_creation_time: timestamp (nullable = true)
 |-- ad_creative_bodies: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_body: string (nullable = true)
 |-- ad_creative_link_caption: string (nullable = true)
 |-- ad_creative_link_captions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_description: string (nullable = true)
 |-- ad_creative_link_descriptions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_title: string (nullable = true)
 |-- ad_creative_link_titles: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_delivery_start_time: timestamp (nullable = true)
 |-- ad_delivery_stop_time: timestamp (nullable = true)
 |-- ad_snapshot_url: string (nullable = true)
 |-- bylines: string (nullable = true)
 |-- currency: string (nullable = true)
 |-- delivery_by_region: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- percentage: string (nullable = true)
 |    |    |-- region: string (nullable = true)
 |-- demographic_distribution: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- age: string (nullable = true)
 |    |    |-- gender: string (nullable = true)
 |    |    |-- percentage: string (nullable = true)
 |-- estimated_audience_size: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- funding_entity: string (nullable = true)
 |-- id: string (nullable = true)
 |-- impressions: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- languages: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- page_id: string (nullable = true)
 |-- page_name: string (nullable = true)
 |-- publisher_platforms: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- region_distribution: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- percentage: string (nullable = true)
 |    |    |-- region: string (nullable = true)
 |-- spend: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- spend_lower_bound: double (nullable = true)
 |-- spend_upper_bound: double (nullable = true)
 |-- impressions_lower_bound: double (nullable = true)
 |-- impressions_upper_bound: double (nullable = true)
```

Convert `percentage` Fields in `demographic_distribution` and `region_distribution` to Double

[23]:
```python
from pyspark.sql.functions import col, explode, collect_list, struct

df = df.withColumn("demographic", explode(col("demographic_distribution")))
df = df.withColumn("demographic_percentage", col("demographic.percentage").cast("double"))
df = df.groupBy("id", "ad_creation_time", "ad_creative_bodies", "ad_creative_body", "ad_creative_link_caption",
                "ad_creative_link_captions", "ad_creative_link_description", "ad_creative_link_descriptions",
                "ad_creative_link_title", "ad_creative_link_titles", "ad_delivery_start_time", "ad_delivery_stop_time",
                "ad_snapshot_url", "bylines", "currency", "estimated_audience_size", "funding_entity", "impressions",
                "languages", "page_id", "page_name", "publisher_platforms", "region_distribution", "spend",
                "spend_lower_bound", "spend_upper_bound", "impressions_lower_bound", "impressions_upper_bound"
               ).agg(collect_list(struct("demographic.age", "demographic.gender", "demographic_percentage")).alias("demographic_distribution"))
```

[24]:
```
df = df.withColumn("region", explode(col("region_distribution")))
df = df.withColumn("region_percentage", col("region.percentage").cast("double"))
df = df.groupBy("id", "ad_creation_time", "ad_creative_bodies", "ad_creative_body", "ad_creative_link_caption",
                "ad_creative_link_captions", "ad_creative_link_description", "ad_creative_link_descriptions",
                "ad_creative_link_title", "ad_creative_link_titles", "ad_delivery_start_time", "ad_delivery_stop_time",
                "ad_snapshot_url", "bylines", "currency", "estimated_audience_size", "funding_entity", "impressions",
                "languages", "page_id", "page_name", "publisher_platforms", "demographic_distribution", "spend",
                "spend_lower_bound", "spend_upper_bound", "impressions_lower_bound", "impressions_upper_bound"
               ).agg(collect_list(struct("region_percentage", "region.region")).alias("region_distribution"))
```

[25]:
```
df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- ad_creation_time: timestamp (nullable = true)
 |-- ad_creative_bodies: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_body: string (nullable = true)
 |-- ad_creative_link_caption: string (nullable = true)
 |-- ad_creative_link_captions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_description: string (nullable = true)
 |-- ad_creative_link_descriptions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_creative_link_title: string (nullable = true)
 |-- ad_creative_link_titles: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- ad_delivery_start_time: timestamp (nullable = true)
 |-- ad_delivery_stop_time: timestamp (nullable = true)
 |-- ad_snapshot_url: string (nullable = true)
 |-- bylines: string (nullable = true)
 |-- currency: string (nullable = true)
 |-- estimated_audience_size: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- funding_entity: string (nullable = true)
 |-- impressions: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- languages: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- page_id: string (nullable = true)
 |-- page_name: string (nullable = true)
 |-- publisher_platforms: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- demographic_distribution: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- age: string (nullable = true)
 |    |    |-- gender: string (nullable = true)
 |    |    |-- demographic_percentage: double (nullable = true)
 |-- spend: struct (nullable = true)
 |    |-- lower_bound: string (nullable = true)
 |    |-- upper_bound: string (nullable = true)
 |-- spend_lower_bound: double (nullable = true)
 |-- spend_upper_bound: double (nullable = true)
 |-- impressions_lower_bound: double (nullable = true)
 |-- impressions_upper_bound: double (nullable = true)
 |-- region_distribution: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- region_percentage: double (nullable = true)
 |    |    |-- region: string (nullable = true)
```

Analyzing Ad Volume for Issues in Australia until the 2022 election results

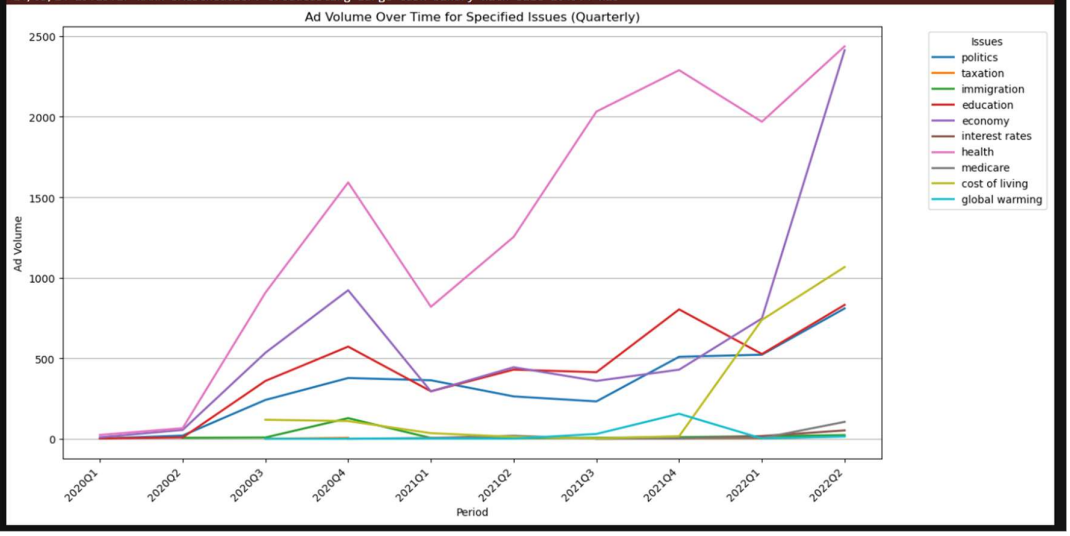
[26]:
```
from pyspark.sql.functions import col, count, quarter, year, concat_ws
import pandas as pd

df1 = df.select(*df.columns)

# Define the topic keywords for issues
topic_keywords = ["politics", "taxation", "immigration", "education", "economy", "interest rates", "health", "medicare", "cost of living", "global warm

def get_ad_volume_by_issue(issue):
    issue_ads = df1.filter(col("ad_creative_body").contains(issue))
    issue_ads_quarterly = issue_ads.groupBy(year(col("ad_creation_time")).alias("year"), quarter(col("ad_creation_time")).alias("quarter")) \
                                   .agg(count("*").alias(issue))
    return issue_ads_quarterly

ad_volumes_list = [get_ad_volume_by_issue(issue) for issue in topic_keywords]
ad_volumes_df = ad_volumes_list[0]
for ad_volume in ad_volumes_list[1:]:
    ad_volumes_df = ad_volumes_df.join(ad_volume, ["year", "quarter"], "outer")
```

[27]:
```
spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")
```

[28]:
```
import matplotlib.pyplot as plt
import pandas as pd

ad_volumes_pd = ad_volumes_df.withColumn("Period", concat_ws("Q", col("year"), col("quarter"))).orderBy("year", "quarter").toPandas()

plt.figure(figsize=(14, 7))
for issue in topic_keywords:
    plt.plot(ad_volumes_pd["Period"], ad_volumes_pd[issue], label=issue, linewidth = 2)
plt.xlabel('Period')
plt.ylabel('Ad Volume')
plt.title('Ad Volume Over Time for Specified Issues (Quarterly)')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Issues', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

```
24/05/24 19:13:16 WARN DAGScheduler: Broadcasting large task binary with size 1066.4 KiB
24/05/24 19:13:16 WARN DAGScheduler: Broadcasting large task binary with size 1066.7 KiB
24/05/24 19:13:17 WARN DAGScheduler: Broadcasting large task binary with size 1049.4 KiB
```

```
24/05/24 19:13:17 WARN DAGScheduler: Broadcasting large task binary with size 1049.4 KiB
```



Focusing on Political Party Accounts and their Target Demographics

[29]:
```
from pyspark.sql.functions import col, count

df2 = df.select(*df.columns)

# keywords related to the parties and their leaders
party_keywords = [
    "Labor Party", "Anthony Albanese",
    "Liberal Party", "Scott Morrison",
    "Australian Greens", "Adam Bandt",
    "Katter's Australian Party", "Bob Katter"
]

# filter ads based on the keywords
filtered_ads = df2.filter(
    col("ad_creative_body").rlike("|".join(party_keywords))
)

page_ad_counts = filtered_ads.groupBy("page_name").agg(count("*").alias("ad_count"))
top_20_pages = page_ad_counts.orderBy(col("ad_count").desc()).limit(20)
top_20_pages_pd = top_20_pages.toPandas()
print(top_20_pages_pd)
```

```
[Stage 485:=================================================>(197 + 3) / 200]
```

```
[Stage 485:=====================================================>(197 + 3) / 200]
```

	page_name	ad_count
0	Australian Labor Party	1736
1	Queensland Labor	464
2	Liberal Party of Australia	437
3	Crikey	316
4	Greenpeace Australia Pacific	253
5	Victorian Labor	252
6	Bill Shorten	193
7	Climate 200	189
8	Senator Anthony Chisholm	172
9	Stand Up For WA	163
10	Amnesty International Australia	157
11	Anthony Albanese	149
12	Mark Butler MP	147
13	GetUp!	146
14	Julian Hill MP	121
15	NSW Liberal Party	113
16	Armenian National Committee of Australia	106
17	Australian Education Union Tasmania	93
18	Councillor Jerome Laxale - Labor	91
19	Richard Marles MP	80

[30]:
```
from pyspark.sql.functions import col

scott_morrison_ads = df2.filter(col("page_name").rlike("(?i)Scott Morrison"))

scott_morrison_ad_count = scott_morrison_ads.count()

scott_morrison_page_counts = scott_morrison_ads.groupBy("page_name").count().orderBy(col("count").desc())
scott_morrison_page_counts.show(truncate=False)
```

```
[Stage 497:================================================>   (185 + 15) / 200]
+----------------------+-----+
|page_name             |count|
+----------------------+-----+
|Scott Morrison (ScoMo)|196  |
+----------------------+-----+
```

[31]:
```
from pyspark.sql.functions import col

greens_ads = df2.filter(col("page_name").rlike("(?i)Australian Greens"))

greens_ad_count = greens_ads.count()

greens_page_counts = greens_ads.groupBy("page_name").count().orderBy(col("count").desc())
greens_page_counts.show(truncate=False)
```

```
+----------------------+-----+
|page_name             |count|
+----------------------+-----+
|The Australian Greens|43   |
+----------------------+-----+
```

[32]:
```
from pyspark.sql.functions import col

adam_ads = df2.filter(col("page_name").rlike("(?i)Adam Brandt"))

adam_ad_count = adam_ads.count()

adam_page_counts = adam_ads.groupBy("page_name").count().orderBy(col("count").desc())
adam_page_counts.show(truncate=False)
```

```
[Stage 521:================================================>   (186 + 14) / 200]
+---------+-----+
|page_name|count|
+---------+-----+
+---------+-----+
```

[33]:
```
from pyspark.sql.functions import col

katters_ads = df2.filter(col("page_name").rlike("(?i)Katter's Australian Party"))

katters_ad_count = katters_ads.count()
```

```
katters_page_counts = katters_ads.groupBy("page_name").count().orderBy(col("count").desc())
katters_page_counts.show(truncate=False)
```

```
[Stage 530:=====================================================>  (192 + 8) / 200]
+-------------------------+-----+
|page_name                |count|
+-------------------------+-----+
|Katter's Australian Party|81   |
+-------------------------+-----+
```

[34]:
```
from pyspark.sql.functions import col

katter_ads = df2.filter(col("page_name").rlike("(?i)Bob Katter"))

katter_ad_count = katter_ads.count()

katter_page_counts = katter_ads.groupBy("page_name").count().orderBy(col("count").desc())
katter_page_counts.show(truncate=False)
```

```
[Stage 542:=====================================================> (193 + 7) / 200]
+----------+-----+
|page_name |count|
+----------+-----+
|Bob Katter|5    |
+----------+-----+
```

Looking at the 4 Major Parties in the 2022 Election

[35]:
```
from pyspark.sql.functions import explode, col, when, count, lit
import pandas as pd
import matplotlib.pyplot as plt

# selecting pages that are relevant
facebook_pages = ["Australian Labor Party", "Liberal Party of Australia", "Katter's Australian Party", "The Australian Greens"]
party_ads = df2.filter(col("page_name").isin(facebook_pages))

# combine and rename the aggregates
party_ads = party_ads.withColumn("page_group",
    when(col("page_name") == "Australian Labor Party", "Labor")
    .when(col("page_name") == "Liberal Party of Australia", "Liberal")
    .when(col("page_name") == "Katter's Australian Party", "Katter's")
    .when(col("page_name") == "The Australian Greens", "Greens")
)

# explode the demographic data
demographics_page_agg = party_ads.select("page_group", explode(col("demographic_distribution")).alias("demographic"))

# handle missing or unnamed gender categories after exploding
demographics_page_agg = demographics_page_agg.withColumn("gender",
    when(col("demographic.gender") == "", "Unknown")
    .when(col("demographic.gender").isNull(), "Unknown")
    .when(col("demographic.gender") == "male", "male")
    .when(col("demographic.gender") == "female", "female")
    .otherwise("Unknown")
)
```

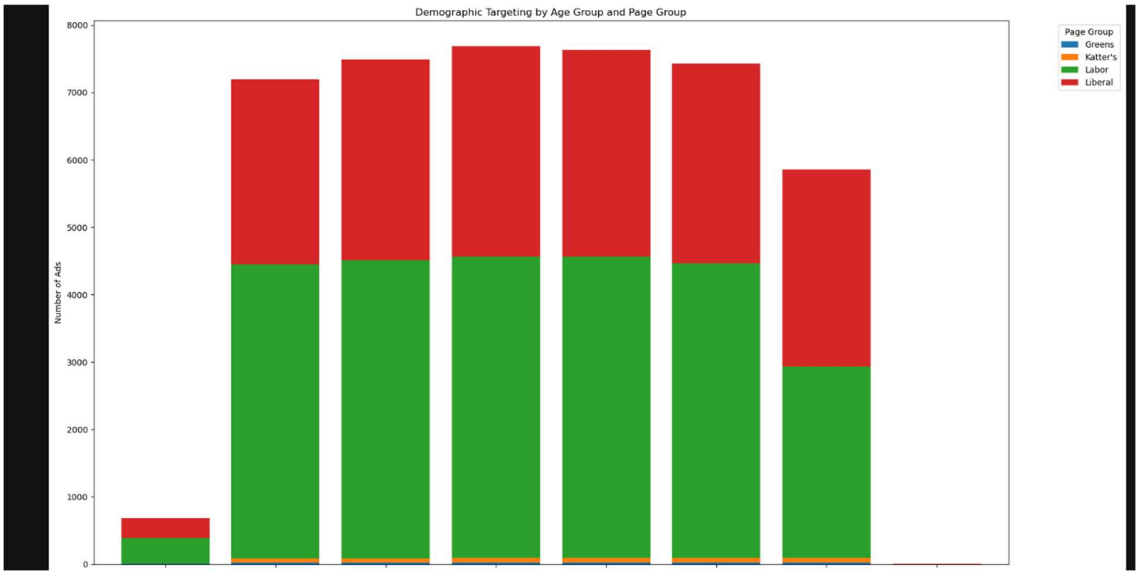
```
# select necessary columns for aggregation
demographics_page_agg = demographics_page_agg.select(
    col("page_group"),
    col("demographic.age").alias("age"),
    col("gender"),
    col("demographic.demographic_percentage").alias("percentage")
)

# aggregate by count of ads
ad_count_page_agg = demographics_page_agg.groupBy("page_group", "age", "gender").agg(count("*").alias("ad_count"))

ad_count_page_pd = ad_count_page_agg.toPandas()

print(ad_count_page_pd.groupby('gender')['ad_count'].sum())
print(ad_count_page_pd.groupby(['page_group', 'gender'])['ad_count'].sum())
print(ad_count_page_pd.groupby('age')['ad_count'].sum())

age_order = ["13-17", "18-24", "25-34", "35-44", "45-54", "55-64", "65+", "Unknown"]
ad_count_page_pd["age"] = pd.Categorical(ad_count_page_pd["age"], categories=age_order, ordered=True)

pivot_age_page = ad_count_page_pd.pivot_table(index=["age"], columns="page_group", values="ad_count", fill_value=0).reset_index()

pivot_age_page = pivot_age_page[['age', 'Greens', "Katter's", 'Labor', 'Liberal']]

plt.figure(figsize=(18, 10))
pivot_age_page.plot(kind='bar', stacked=True, ax=plt.gca(), width=0.8)
plt.xlabel('Age Group')
plt.ylabel('Number of Ads')
plt.title('Demographic Targeting by Age Group and Page Group')
plt.legend(title='Page Group', bbox_to_anchor=(1.05, 1), loc='upper left')
```

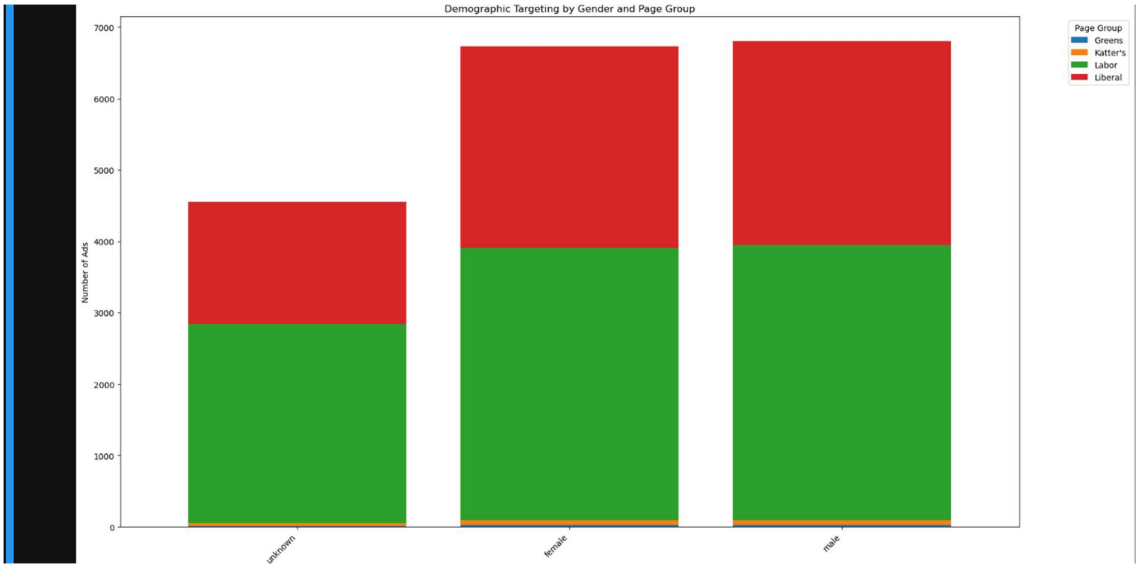
```
plt.title('Demographic Targeting by Age Group and Page Group')
plt.legend(title='Page Group', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Filter out any empty or unnamed gender categories
ad_count_page_pd = ad_count_page_pd[ad_count_page_pd['gender'].isin(['male', 'female', 'Unknown'])]

pivot_gender_page = ad_count_page_pd.pivot_table(index=["gender"], columns="page_group", values="ad_count", fill_value=0).reset_index()

pivot_gender_page = pivot_gender_page[['gender', 'Greens', "Katter's", 'Labor', 'Liberal']]

plt.figure(figsize=(18, 10))
pivot_gender_page.plot(kind='bar', stacked=True, ax=plt.gca(), width=0.8)
plt.xlabel('Gender')
plt.ylabel('Number of Ads')
plt.title('Demographic Targeting by Gender and Page Group')
plt.legend(title='Page Group', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=45, ha='right', ticks=range(3), labels=['unknown', 'female', 'male'])
plt.tight_layout()
plt.show()
```

```
[Stage 548:==============================================>  (192 + 8) / 200]
```

```
gender
Unknown    38086
female     47120
male       47660
Name: ad_count, dtype: int64
page_group  gender
Greens      Unknown       79
            female       172
            male         162
Katter's    Unknown      348
            female       457
            male         463
Labor       Unknown    22311
            female     26738
            male       26986
Liberal     Unknown    15348
            female     19753
            male       20049
```

```
Name: ad_count, dtype: int64
age
             975
13-17       2037
18-24      21577
25-34      22470
35-44      23050
45-54      22895
55-64      22276
65+        17585
Unknown        1
Name: ad_count, dtype: int64
```





Looking at URLs included in Ads to find the most Popular Internet Domains during the 2022 Election Campaign

[36]:
```
from pyspark.sql.functions import col, length

non_empty_string_counts = df.select(
    (count(col("ad_creative_body").cast("string")) - count(when(length(col("ad_creative_body")) == 0, True))).alias("non_empty_ad_creative_body"),
    (count(col("ad_creative_link_caption").cast("string")) - count(when(length(col("ad_creative_link_caption")) == 0, True))).alias("non_empty_ad_creat
    (count(col("ad_creative_link_description").cast("string")) - count(when(length(col("ad_creative_link_description")) == 0, True))).alias("non_empty_
    (count(col("ad_creative_link_title").cast("string")) - count(when(length(col("ad_creative_link_title")) == 0, True))).alias("non_empty_ad_creative_
).collect()[0]

print(f"Non-empty rows in ad_creative_body: {non_empty_string_counts['non_empty_ad_creative_body']}")
print(f"Non-empty rows in ad_creative_link_caption: {non_empty_string_counts['non_empty_ad_creative_link_caption']}")
print(f"Non-empty rows in ad_creative_link_description: {non_empty_string_counts['non_empty_ad_creative_link_description']}")
print(f"Non-empty rows in ad_creative_link_title: {non_empty_string_counts['non_empty_ad_creative_link_title']}")
```

```
[Stage 554:=================================================>(199 + 1) / 200]
Non-empty rows in ad_creative_body: 226469
Non-empty rows in ad_creative_link_caption: 156650
Non-empty rows in ad_creative_link_description: 127229
Non-empty rows in ad_creative_link_title: 160673
```

[37]:
```
from pyspark.sql.functions import col, size, count, when

non_empty_array_counts = df.select(
    (count(col("ad_creative_bodies").cast("string")) - count(when(size(col("ad_creative_bodies")) == 0, True))).alias("non_empty_ad_creative_bodies"),
    (count(col("ad_creative_link_captions").cast("string")) - count(when(size(col("ad_creative_link_captions")) == 0, True))).alias("non_empty_ad_creat
    (count(col("ad_creative_link_descriptions").cast("string")) - count(when(size(col("ad_creative_link_descriptions")) == 0, True))).alias("non_empty_
    (count(col("ad_creative_link_titles").cast("string")) - count(when(size(col("ad_creative_link_titles")) == 0, True))).alias("non_empty_ad_creative_
).collect()[0]

print(f"Non-empty rows in ad_creative_bodies: {non_empty_array_counts['non_empty_ad_creative_bodies']}")
print(f"Non-empty rows in ad_creative_link_captions: {non_empty_array_counts['non_empty_ad_creative_link_captions']}")
print(f"Non-empty rows in ad_creative_link_descriptions: {non_empty_array_counts['non_empty_ad_creative_link_descriptions']}")
print(f"Non-empty rows in ad_creative_link_titles: {non_empty_array_counts['non_empty_ad_creative_link_titles']}")
```

```
[Stage 560:=================================================> (195 + 5) / 200]
Non-empty rows in ad_creative_bodies: 10813058
Non-empty rows in ad_creative_link_captions: 11110793
Non-empty rows in ad_creative_link_descriptions: 5154236
Non-empty rows in ad_creative_link_titles: 9231222
```

[38]:
```
from pyspark.sql.functions import col, array_contains
from pyspark.sql.functions import sum as spark_sum

df_check = df.select(*df.columns)
```

```
# check if string column values are contained in their corresponding array columns
comparison_results = df_check.select(
    (array_contains(col("ad_creative_bodies"), col("ad_creative_body"))).alias("body_match"),
    (array_contains(col("ad_creative_link_captions"), col("ad_creative_link_caption"))).alias("caption_match"),
    (array_contains(col("ad_creative_link_descriptions"), col("ad_creative_link_description"))).alias("description_match"),
    (array_contains(col("ad_creative_link_titles"), col("ad_creative_link_title"))).alias("title_match")
)

# count matches for each pair of columns
match_counts = comparison_results.select(
    spark_sum(col("body_match").cast("int")).alias("body_match_count"),
    spark_sum(col("caption_match").cast("int")).alias("caption_match_count"),
    spark_sum(col("description_match").cast("int")).alias("description_match_count"),
    spark_sum(col("title_match").cast("int")).alias("title_match_count")
)

match_counts.show()
```

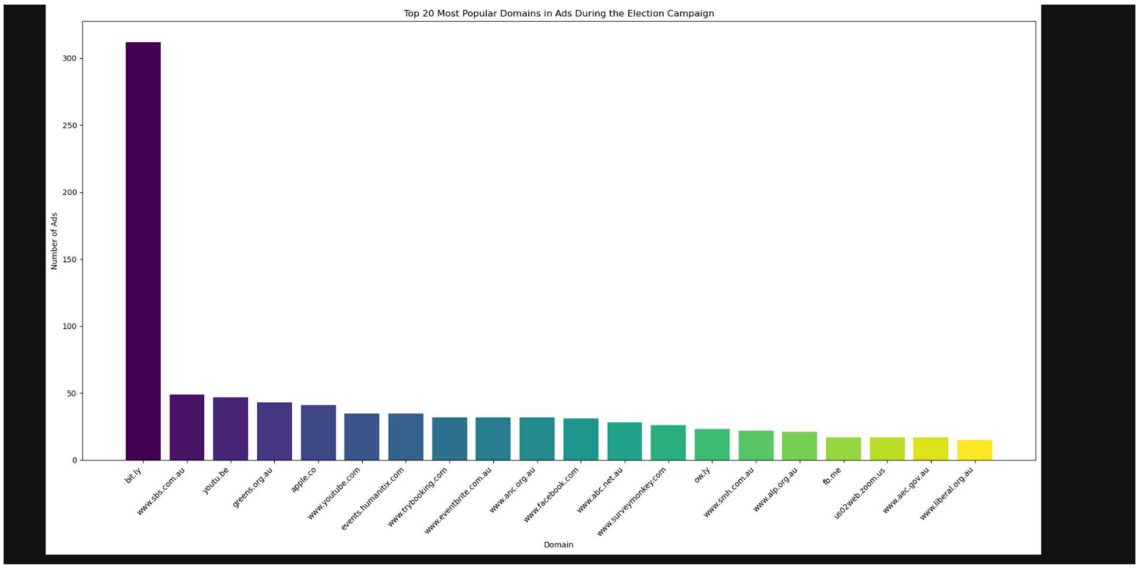
```
[Stage 566:================================================>   (188 + 12) / 200]
+---------------+------------------+----------------------+----------------+
|body_match_count|caption_match_count|description_match_count|title_match_count|
+---------------+------------------+----------------------+----------------+
|              0|                 0|                     0|               0|
+---------------+------------------+----------------------+----------------+
```

Here, as we can see the string columns for the URLs in ads have substantially lower non-empty rows when compared to their counterpart array columns. Upon checking, I found that not a single value in the string columns matches with the value in their respective array columns. Hence, I will check for URLs in both columns and if I find the same URL in the same row in both the columns, I will only count it once to maintain consistency in the analysis.

[45]:
```
from pyspark.sql.functions import col, explode, regexp_extract, lower, count, to_date, lit, array_union
import matplotlib.cm as cm

df3 = df.select(*df.columns)

# filter ads for the election campaign period (January 2022 to May 2022)
election_campaign_ads = df3.filter((col("ad_creation_time") >= to_date(lit("2022-01-01"))) &
                                   (col("ad_creation_time") <= to_date(lit("2022-05-21"))))

# URL pattern
url_pattern = r'(https?://[^\s]+)'

# extract URLs from array columns and string columns
combined_array_column = array_union(array_union(col("ad_creative_link_descriptions"), col("ad_creative_link_captions")),
                                    array_union(col("ad_creative_link_titles"), col("ad_creative_bodies")))

urls_from_arrays = election_campaign_ads.select(
    explode(combined_array_column).alias("description")
).select(
    regexp_extract(col("description"), url_pattern, 0).alias("url")
)
```

```
urls_from_strings = election_campaign_ads.select(
    regexp_extract(col("ad_creative_link_description"), url_pattern, 0).alias("url")
).union(
    election_campaign_ads.select(regexp_extract(col("ad_creative_link_caption"), url_pattern, 0).alias("url"))
).union(
    election_campaign_ads.select(regexp_extract(col("ad_creative_link_title"), url_pattern, 0).alias("url"))
).union(
    election_campaign_ads.select(regexp_extract(col("ad_creative_body"), url_pattern, 0).alias("url"))
)

all_urls_df = urls_from_arrays.union(urls_from_strings).distinct()

all_urls_df = all_urls_df.filter(col("url") != "")

all_urls_df = all_urls_df.withColumn("domain", lower(regexp_extract(col("url"), r'https?://([^/]+)', 1)))

domain_counts = all_urls_df.groupBy("domain").agg(count("*").alias("count"))

domain_counts_pd = domain_counts.toPandas()

domain_counts_pd = domain_counts_pd.sort_values(by="count", ascending=False)

plt.figure(figsize=(18, 10))
top_20_domains = domain_counts_pd.head(20)
colormap = cm.get_cmap('viridis', len(top_20_domains))
color = [colormap(i) for i in range(len(top_20_domains))]
plt.bar(top_20_domains["domain"], top_20_domains["count"], color=color)
plt.xlabel('Domain')
plt.ylabel('Number of Ads')
plt.title('Top 20 Most Popular Domains in Ads During the Election Campaign')
plt.xticks(rotation=45, ha='right')
```

```
plt.tight_layout()
plt.show()
```



Here, I can see that the most popular domain in URL links in ads during the election campaign of 2022 was `bit.ly`. But, `bit.ly` is just a URL shortener just like `tinyurl`, so the contents of the URL links cannot be implied by the shortened URLs.

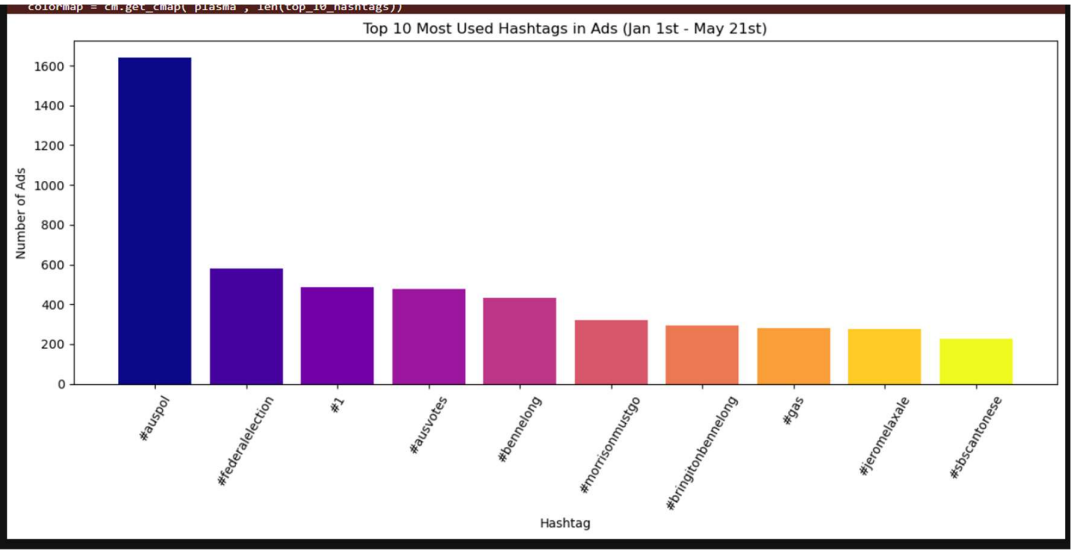
Looking at a Specific Hashtag & What Facebook Page used it the most

[48]:
```
from pyspark.sql.functions import col, to_date, lit, explode
import re
from pyspark.sql.functions import udf
from pyspark.sql.types import ArrayType, StringType
import matplotlib.pyplot as plt

df4 = df.select(*df.columns)

# define a UDF to extract hashtags from text
def extract_hashtags(text):
    return re.findall(r"#\w+", text.lower())

extract_hashtags_udf = udf(extract_hashtags, ArrayType(StringType()))

# filter ads for the election campaign period (January 1st to May 21st)
campaign_ads = df4.filter((col("ad_creation_time") >= to_date(lit("2022-01-01"))) &
                          (col("ad_creation_time") <= to_date(lit("2022-05-21"))))

# apply the UDF to extract hashtags from relevant columns
campaign_ads = campaign_ads.withColumn("hashtags",
                                       extract_hashtags_udf(col("ad_creative_body")))

hashtags_df = campaign_ads.select(explode(col("hashtags")).alias("hashtag"))
```

```
hashtag_counts = hashtags_df.groupBy("hashtag").count().orderBy("count", ascending=False)

hashtag_counts_pd = hashtag_counts.toPandas()

top_10_hashtags = hashtag_counts_pd.head(10)

colormap = cm.get_cmap('plasma', len(top_10_hashtags))
color = [colormap(i) for i in range(len(top_10_hashtags))]
plt.figure(figsize=(12, 6))
plt.bar(top_10_hashtags["hashtag"], top_10_hashtags["count"], color=color)
plt.xlabel('Hashtag')
plt.ylabel('Number of Ads')
plt.title('Top 10 Most Used Hashtags in Ads (Jan 1st - May 21st)')
plt.xticks(rotation=60)
plt.tight_layout()
plt.show()
```

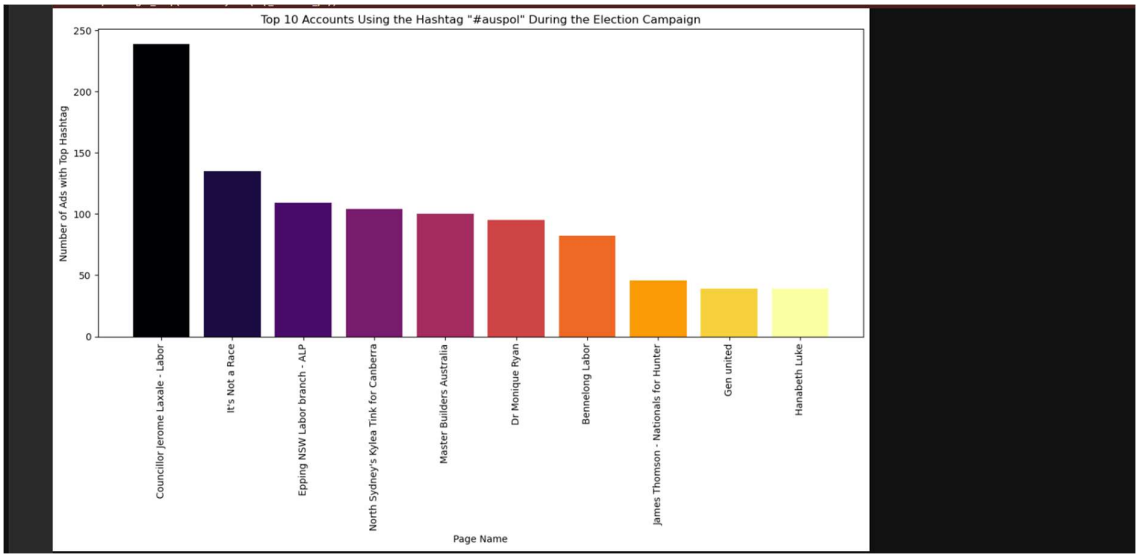
```
/tmp/ipykernel_419291/1359782910.py:31: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two mi
nor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = cm.get_cmap('plasma', len(top_10_hashtags))
```

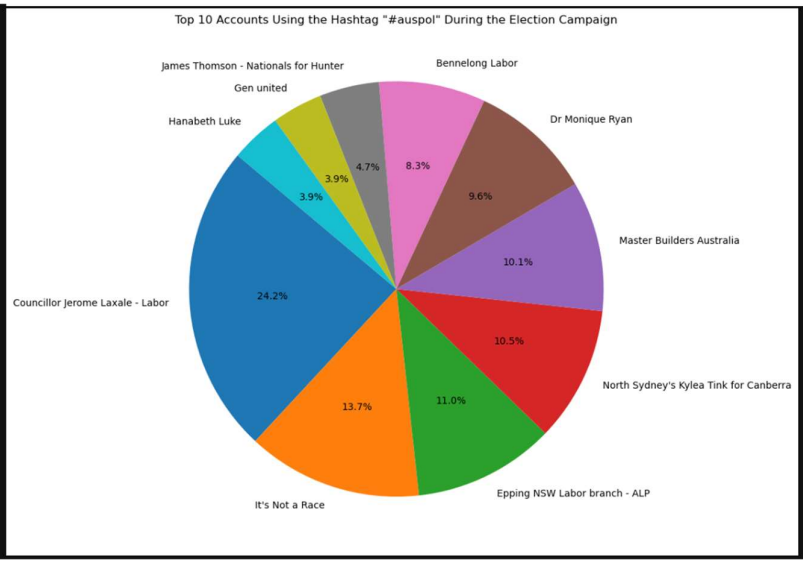
```
colormap = cm.get_cmap('plasma', len(top_10_hashtags))
```



[49]:
```
top_hashtag = hashtag_counts_pd.head(1)["hashtag"].values[0]

ads_with_top_hashtag = campaign_ads.filter(array_contains(col("hashtags"), top_hashtag))

top_accounts = ads_with_top_hashtag.groupBy("page_name").agg(count("id").alias("count")).orderBy("count", ascending=False)

top_accounts_pd = top_accounts.limit(10).toPandas()

print(top_accounts_pd)

colormap = cm.get_cmap('inferno', len(top_accounts_pd))
color = [colormap(i) for i in range(len(top_accounts_pd))]
plt.figure(figsize=(12, 8))
plt.bar(top_accounts_pd["page_name"], top_accounts_pd["count"], color=color)
plt.xlabel('Page Name')
plt.ylabel('Number of Ads with Top Hashtag')
plt.title(f'Top 10 Accounts Using the Hashtag "{top_hashtag}" During the Election Campaign')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))
plt.pie(top_accounts_pd["count"], labels=top_accounts_pd["page_name"], autopct='%1.1f%%', startangle=140)
plt.title(f'Top 10 Accounts Using the Hashtag "{top_hashtag}" During the Election Campaign')
plt.tight_layout()
plt.show()
```

```
                                page_name  count
0        Councillor Jerome Laxale - Labor    239
1                          It's Not a Race    135
2           Epping NSW Labor branch - ALP    109
3  North Sydney's Kylea Tink for Canberra    104
4                Master Builders Australia    100
5                          Dr Monique Ryan     95
6                          Bennelong Labor     82
7    James Thomson - Nationals for Hunter      46
8                             Gen united     39
9                          Hanabeth Luke      39
/tmp/ipykernel_419291/2370076124.py:11: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two mi
nor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  colormap = cm.get_cmap('inferno', len(top_accounts_pd))
```

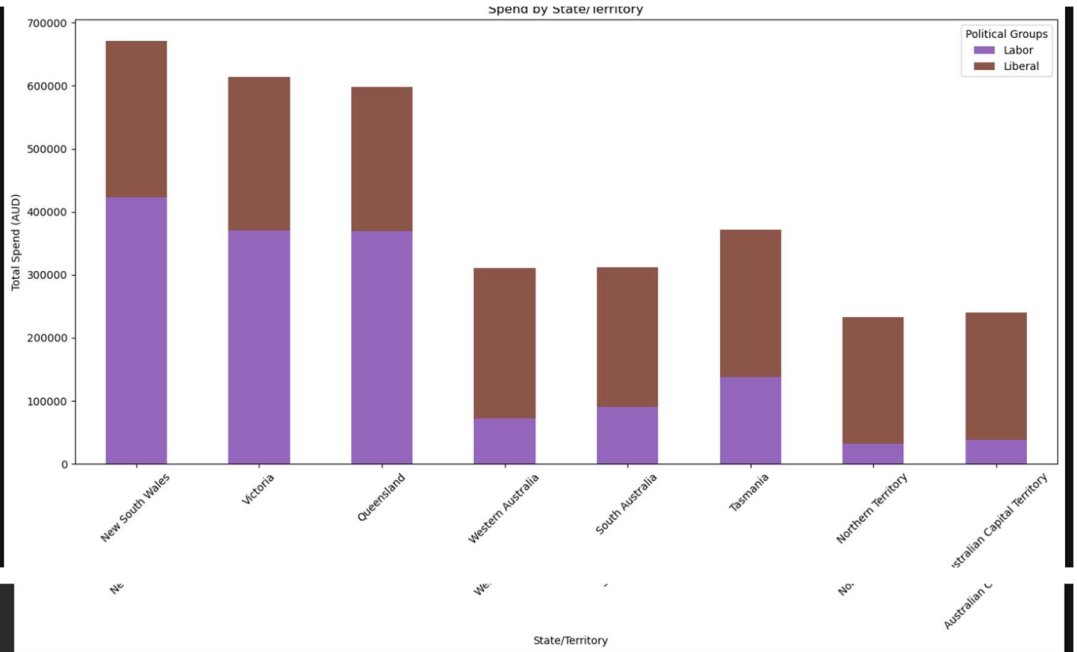




Looking at the Spend by State/Territory during the 2022 Federal Election by the Two Major Parties' and Their Respective Leaders' Pages ¶

[50]:
```
from pyspark.sql.functions import col, when, explode, sum as spark_sum, lit, to_date
import matplotlib.pyplot as plt
import pandas as pd

# define the page groups
page_groups = {
    "Labor": ["Anthony Albanese", "Australian Labor Party"],
    "Liberal": ["Scott Morrison (ScoMo)", "Liberal Party of Australia"]
}

# filter ads for the election campaign period (January 2022 to May 2022)
df5 = df.filter((col("ad_creation_time") >= to_date(lit("2022-01-01"))) &
                (col("ad_creation_time") <= to_date(lit("2022-05-21"))))

# initialize the page_group column
df5 = df5.withColumn("page_group", lit(None).cast("string"))

# create a new column for page groups
for group, pages in page_groups.items():
    df5 = df5.withColumn("page_group",
                         when(col("page_name").isin(pages), group).otherwise(col("page_group"))
                         )

# explode the region distribution data
regions_exp = df5.select("page_group", "spend_lower_bound", explode(col("region_distribution")).alias("region"))
```

```
# select necessary columns for aggregation
regions_exp = regions_exp.select(
    col("page_group"),
    col("region.region").alias("state"),
    col("spend_lower_bound").alias("spend")
)

state_spend = regions_exp.groupBy("page_group", "state").agg(spark_sum("spend").alias("total_spend"))

state_spend_pd = state_spend.toPandas()

state_order = ["New South Wales", "Victoria", "Queensland", "Western Australia", "South Australia", "Tasmania", "Northern Territory", "Australian Capit
state_spend_pd['state'] = pd.Categorical(state_spend_pd['state'], categories=state_order, ordered=True)
state_group_spend = state_spend_pd.groupby(['state', 'page_group'], observed=True).sum(numeric_only=True).unstack().fillna(0)

state_group_spend.columns = state_group_spend.columns.droplevel()
state_group_spend.plot(kind='bar', stacked=True, figsize=(14, 8), color=['#9467bd', '#8c564b'])
plt.xlabel('State/Territory')
plt.ylabel('Total Spend (AUD)')
plt.title('Spend by State/Territory')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Political Groups', labels=['Labor', 'Liberal'])
plt.show()
```



[51]:
```
state_group_spend.reset_index(inplace=True)
state_group_spend.columns.name = None  # Remove the name of the columns index
state_group_spend.columns = ['State', 'Labor', 'Liberal']  # Rename the columns
state_group_spend
```

[51]:

	State	Labor	Liberal
0	New South Wales	423000.0	248200.0
1	Victoria	370100.0	244000.0
2	Queensland	369400.0	229100.0
3	Western Australia	71600.0	239500.0
4	South Australia	90400.0	221700.0
5	Tasmania	138300.0	233300.0
6	Northern Territory	31400.0	201500.0
7	Australian Capital Territory	37700.0	202000.0

[52]:
```
print(state_group_spend)
```

```
                         State     Labor   Liberal
0              New South Wales  423000.0  248200.0
1                     Victoria  370100.0  244000.0
2                   Queensland  369400.0  229100.0
3            Western Australia   71600.0  239500.0
4              South Australia   90400.0  221700.0
5                     Tasmania  138300.0  233300.0
6            Northern Territory   31400.0  201500.0
7  Australian Capital Territory   37700.0  202000.0
```

Analysis of Ad Campaign Durations and Political Alignment Composition for Topics on Economy, Healthcare, and Cost of Living

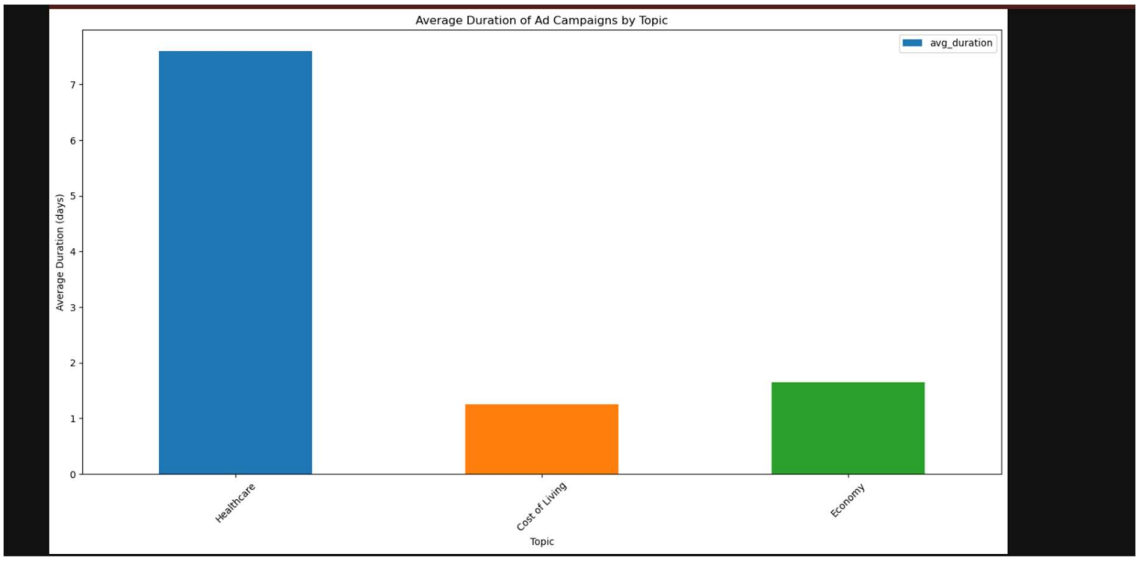
Here, we count each ad only once for the duration, as at the start of our analysis we had removed duplicates, the calculation of the durations relies on unique ads' timestamps.

[56]:
```
from pyspark.sql.functions import col, when, lit, to_date, datediff, expr
import matplotlib.pyplot as plt
import pandas as pd

page_groups = {
    "Labor": ["Anthony Albanese", "Australian Labor Party"],
    "Liberal": ["Scott Morrison (ScoMo)", "Liberal Party of Australia"]
}
```

```
# define topics with comprehensive keywords
topics = {
    "Healthcare": ["health", "hospital", "medicine", "doctor", "nurse", "healthcare", "medical"],
    "Economy": ["economy", "tax", "budget", "finance", "economic", "job", "employment", "business"],
    "Cost of Living": ["cost of living", "living cost", "rent", "mortgage", "utility", "expenses", "food prices", "housing"]
}

df6 = df.filter((col("ad_creation_time") >= to_date(lit("2022-01-01"))) &
                (col("ad_creation_time") <= to_date(lit("2022-05-21"))))

df6 = df6.withColumn("topic", lit(None).cast("string"))

for topic, keywords in topics.items():
    condition = expr(" or ".join([f"ad_creative_body like '%{keyword}%' or ad_creative_link_title like '%{keyword}%'" for keyword in keywords]))
    df6 = df6.withColumn("topic", when(condition, topic).otherwise(col("topic")))

df6 = df6.withColumn("duration", datediff(col("ad_delivery_stop_time"), col("ad_delivery_start_time")))

duration_stats = df6.groupBy("topic").agg(
    expr("avg(duration)").alias("avg_duration")
)

duration_stats_pd = duration_stats.toPandas()

duration_stats_pd = duration_stats_pd[duration_stats_pd['topic'].notna()]

print("Ads by Topic:")
print(duration_stats_pd)
```

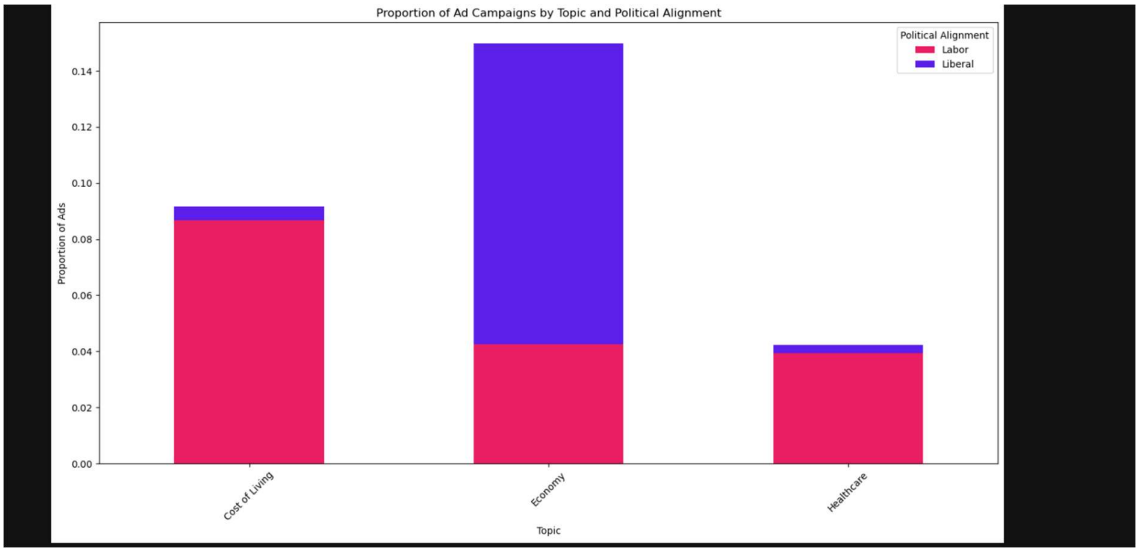
```
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']
duration_stats_pd.plot(kind='bar', x='topic', y='avg_duration', figsize=(14, 8), color=colors)
plt.xlabel('Topic')
plt.ylabel('Average Duration (days)')
plt.title('Average Duration of Ad Campaigns by Topic')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
[Stage 796:================================================>  (192 + 8) / 200]
Ads by Topic:
          topic  avg_duration
0    Healthcare      7.600000
2  Cost of Living      1.250000
3       Economy      1.653061
```



[54]:
```
from pyspark.sql.functions import col, when, lit, to_date, datediff, expr, lower
import matplotlib.pyplot as plt
import pandas as pd

page_groups = {
    "Labor": ["Anthony Albanese", "Australian Labor Party"],
    "Liberal": ["Scott Morrison (ScoMo)", "Liberal Party of Australia"]
}

topics = {
    "Healthcare": ["health", "hospital", "medicine", "doctor", "nurse", "healthcare", "medical"],
    "Economy": ["economy", "tax", "budget", "finance", "economic", "job", "employment", "business"],
    "Cost of Living": ["cost of living", "living cost", "rent", "mortgage", "utility", "expenses", "food prices", "housing"]
}

df6 = df.filter((col("ad_creation_time") >= lit("2022-01-01")) &
                (col("ad_creation_time") <= lit("2022-05-21")))

df6 = df6.withColumn("topic", lit(None).cast("string"))
df6 = df6.withColumn("alignment", lit(None).cast("string"))

for topic, keywords in topics.items():
    condition = expr(" or ".join([f"lower(ad_creative_body) like '%{keyword}%' or lower(ad_creative_link_title) like '%{keyword}%'" for keyword in keyw
    df6 = df6.withColumn("topic", when(condition, topic).otherwise(col("topic")))

for alignment, pages in page_groups.items():
    condition = expr(" or ".join([f"page_name like '%{page}%'" for page in pages]))
    df6 = df6.withColumn("alignment", when(condition, alignment).otherwise(col("alignment")))

df6 = df6.withColumn("duration", when(col("ad_delivery_stop_time").isNotNull() & col("ad_delivery_start_time").isNotNull(),
                                      datediff(col("ad_delivery_stop_time"), col("ad_delivery_start_time"))).otherwise(lit(None)))
```

```
duration_stats = df6.groupBy("topic", "alignment").count()

duration_stats_pd = duration_stats.toPandas()

duration_stats_pd = duration_stats_pd[duration_stats_pd['topic'].notna()]

duration_stats_pivot = duration_stats_pd.pivot(index='topic', columns='alignment', values='count').fillna(0)
duration_stats_pivot['Total'] = duration_stats_pivot.sum(axis=1)
duration_stats_pivot['Labor'] = duration_stats_pivot['Labor'] / duration_stats_pivot['Total']
duration_stats_pivot['Liberal'] = duration_stats_pivot['Liberal'] / duration_stats_pivot['Total']

ax = duration_stats_pivot[['Labor', 'Liberal']].plot(kind='bar', stacked=True, figsize=(14, 8), color=['#E91E63', '#5C1EE9'])
plt.xlabel('Topic')
plt.ylabel('Proportion of Ads')
plt.title('Proportion of Ad Campaigns by Topic and Political Alignment')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Political Alignment')
plt.show()
```

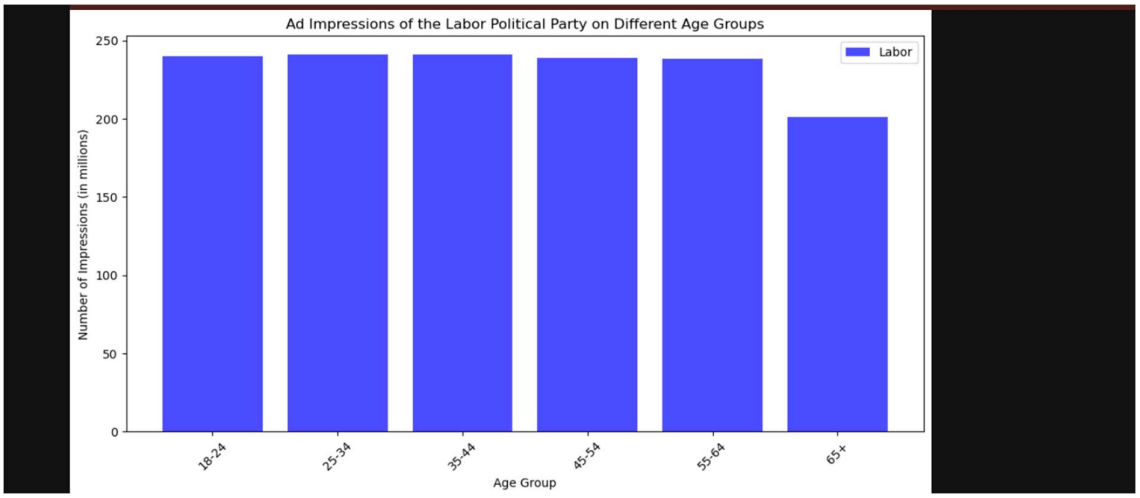


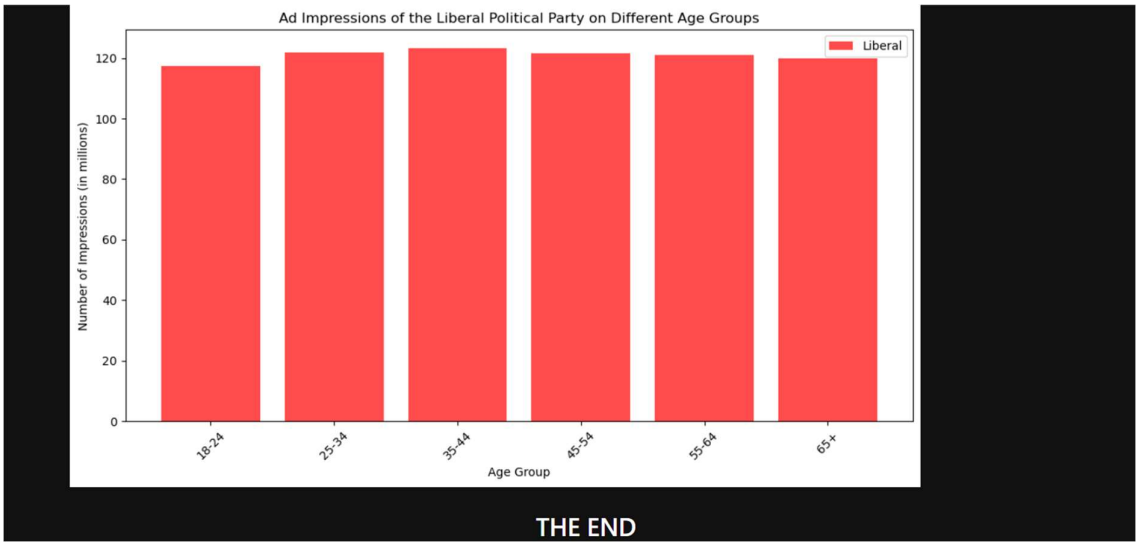
Ad Impressions of Both Political Parties on Different Age Groups

[58]:

```
def process_ads(political_group):
    condition = expr(" or ".join([f"page_name like '%{page}%'" for page in page_groups[political_group]]))
    ads = df.filter(condition)

    # explode the demographic_distribution array column
    ads = ads.withColumn("demographic", explode("demographic_distribution")) \
             .withColumn("age", col("demographic.age")) \
             .withColumn("impressions", col("impressions.lower_bound").cast("double")) \
             .select("age", "impressions")

    # filter out unwanted age groups and aggregate impressions by age group
    ads = ads.filter(col("age").isin(["18-24", "25-34", "35-44", "45-54", "55-64", "65+"]))
    impressions_stats = ads.groupBy("age").agg(_sum("impressions").alias("total_impressions")).toPandas()

    # convert impressions to millions
    impressions_stats["total_impressions"] = impressions_stats["total_impressions"] / 1e6

    age_order = ["18-24", "25-34", "35-44", "45-54", "55-64", "65+"]
    impressions_stats["age"] = pd.Categorical(impressions_stats["age"], categories=age_order, ordered=True)
    impressions_stats = impressions_stats.sort_values("age")

    return impressions_stats

labor_impressions_stats = process_ads('Labor')
liberal_impressions_stats = process_ads('Liberal')
```

```
plt.figure(figsize=(12, 6))
plt.bar(labor_impressions_stats['age'], labor_impressions_stats['total_impressions'], color='blue', alpha=0.7, label='Labor')
plt.title('Ad Impressions of the Labor Political Party on Different Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Number of Impressions (in millions)')
plt.xticks(rotation=45)
plt.legend()
plt.show()

plt.figure(figsize=(12, 6))
plt.bar(liberal_impressions_stats['age'], liberal_impressions_stats['total_impressions'], color='red', alpha=0.7, label='Liberal')
plt.title('Ad Impressions of the Liberal Political Party on Different Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Number of Impressions (in millions)')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```


Ad Impressions of the Labor Political Party on Different Age Groups
Labor
Number of Impressions (in millions)
0
50
100
150
200
250
18-24
25-34
35-44
45-54
55-64
65+
Age Group


Ad Impressions of the Liberal Political Party on Different Age Groups
Liberal
Number of Impressions (in millions)
0
20
40
60
80
100
120
18-24
25-34
35-44
45-54
55-64
65+
Age Group

THE END