Report

CGS-616

Rubal Gajbhiye - 241280002
Kuldeep Meena - 220556
J Kartik Sai - 241280001

A. Exploring the biases in search results

Methodology

1. **Search Engine Selection**: Search engines like Google, Bing, and DuckDuckGo were used to query searches and results were accumulated. These platforms were chosen for their wide user base and diverse algorithms.
2. **Search Phrases**: A variety of topics were selected to highlight potential biases, including "privacy breach," "hate speech," "fake news,". Other searches could be used appropriately
3. **Automated Data Collection**: Selenium WebDriver was used to automate searches using a simple python driver code.
    a. Temporary directories ensured each session was isolated to avoid conflicts.
    b. BeautifulSoup was used to scrape titles and links from the search results.
4. **Data Storage and Analysis**:
    a. All search results were saved in a CSV file for detailed analysis.
    b. WordClouds were generated to visualize frequently used terms in the results.
    c. Frequency analysis was performed to detect recurring patterns and biases.

Observed Results

Google Results:

- **Frequent Terms**: "social media," "platforms," "regulation," and "freedom of speech."
- **Nature of Results**: News articles from authoritative sources, such as government websites and major news outlets.
- **Example Links**:
    1. Regulating Hate Speech on Social Media
    2. Freedom of Speech vs. Hate Speech

Bing Results:

- **Frequent Terms**: "online hate," "moderation," and "harmful content."
- **Nature of Results**: A mix of news, blog posts, and opinion pieces.
- **Example Links**:
    1. The Role of Platforms in Combating Online Hate
    2. Moderation Challenges on Social Media

DuckDuckGo Results:

- **Frequent Terms**: "free speech," "community guidelines," and "privacy."
- **Nature of Results**: Primarily opinion articles and blog posts, with fewer authoritative sources.
- **Example Links**:
    1. Community Guidelines and Free Speech
    2. Balancing Privacy and Regulation

Observed Biases

1. **Geographical Bias**: The results varied depending on the location, with the U.S.-centric regulations dominating the content.
2. **Source Bias**: Google prioritized authoritative sources, while DuckDuckGo leaned more towards opinion-based content.
3. **Phrasing Bias**: Different search phrases (e.g., "hate speech regulation" vs. "free speech issues") produced noticeably different results.

Possible Causes for Biases in Search Results

1. **Algorithmic Prioritization**: Search engines rank results based on relevance algorithms, often favoring authoritative or popular sources.
2. **Geolocation**: Results are tailored to my location to improve relevance, inadvertently introducing bias.
3. **User Behavior**: Search history and click patterns influence personalized results.
4. **Content Availability**: Variations in the volume and quality of content on certain topics can skew search outcomes.

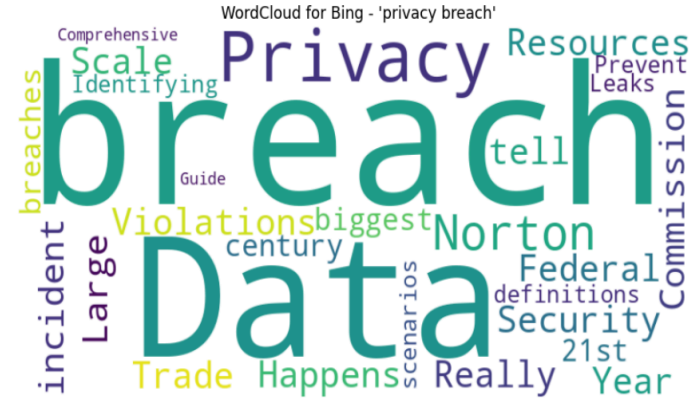
Recommendations for Reducing Biases

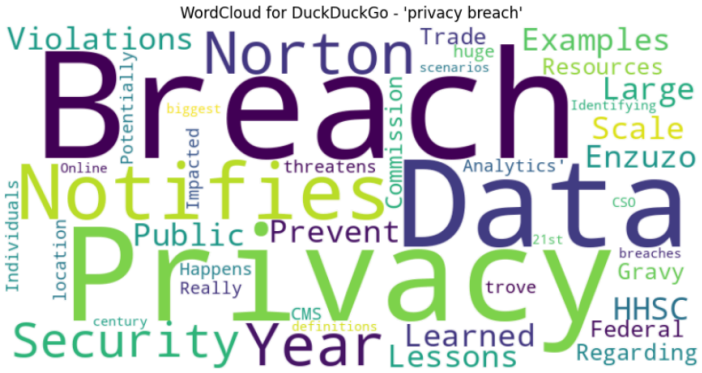
1. **Transparency in Algorithms**: Search engines should publicly disclose how they rank and prioritize content.
2. **Diversity in Sources**: Incorporate a wider range of perspectives, including underrepresented voices.
3. **Geolocation Options**: Allow users to easily view results from different regions.
4. **Neutral Query Processing**: Implement neutral language processing to reduce phrasing biases.
5. **User Education**: Educate users about potential biases and encourage critical evaluation of search results.

Limitations and Challenges

1. **Dynamic Search Results**: Search results evolve over time, making it challenging to replicate findings.
2. **Geolocation Constraints**: Accessing region-specific results was limited to physical location. While some engines allow geological location changes, the same could not be said for all engines.
3. **Data Collection Complexity**: Automating searches required careful handling of WebDriver configurations to avoid errors.

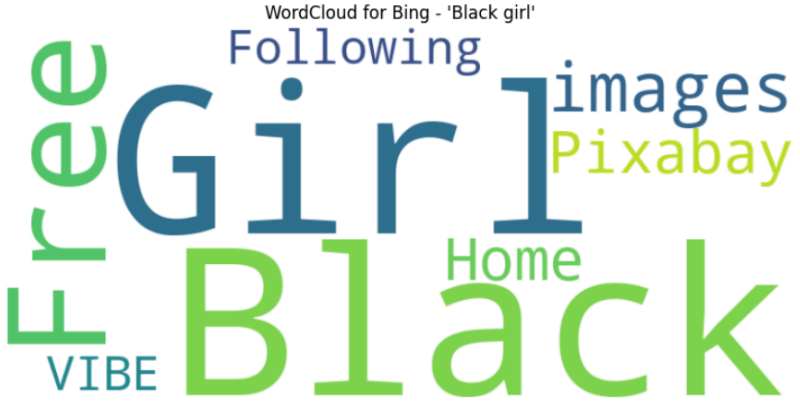
Search Result Word clouds



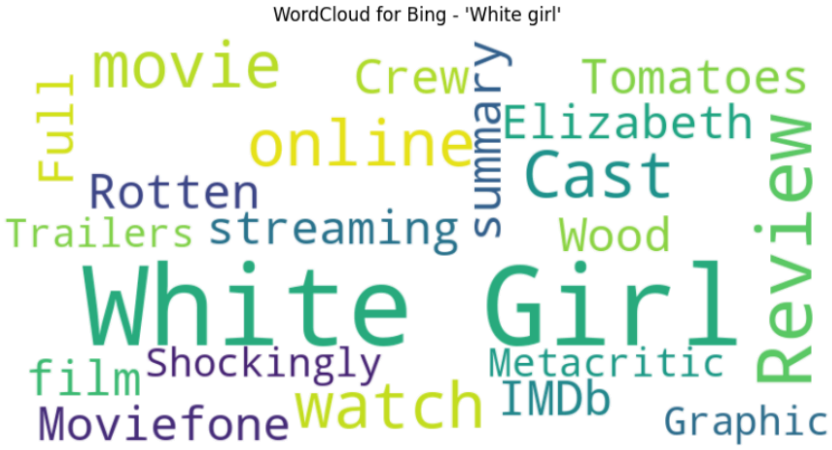


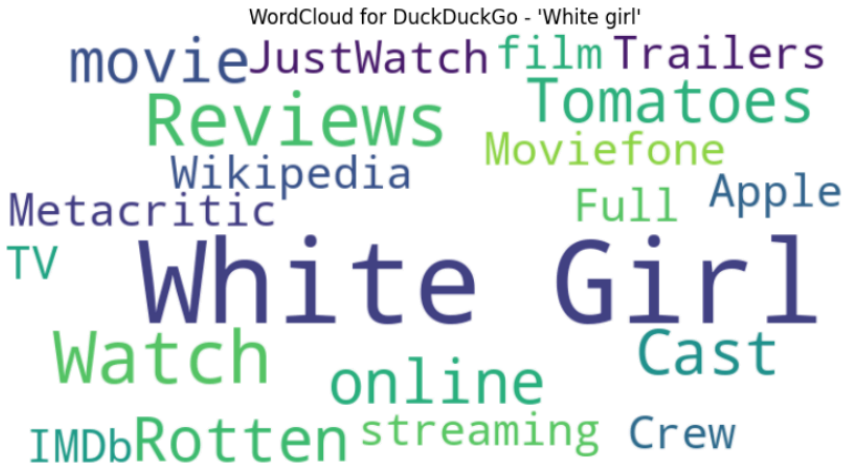
Topic : Privacy Breach

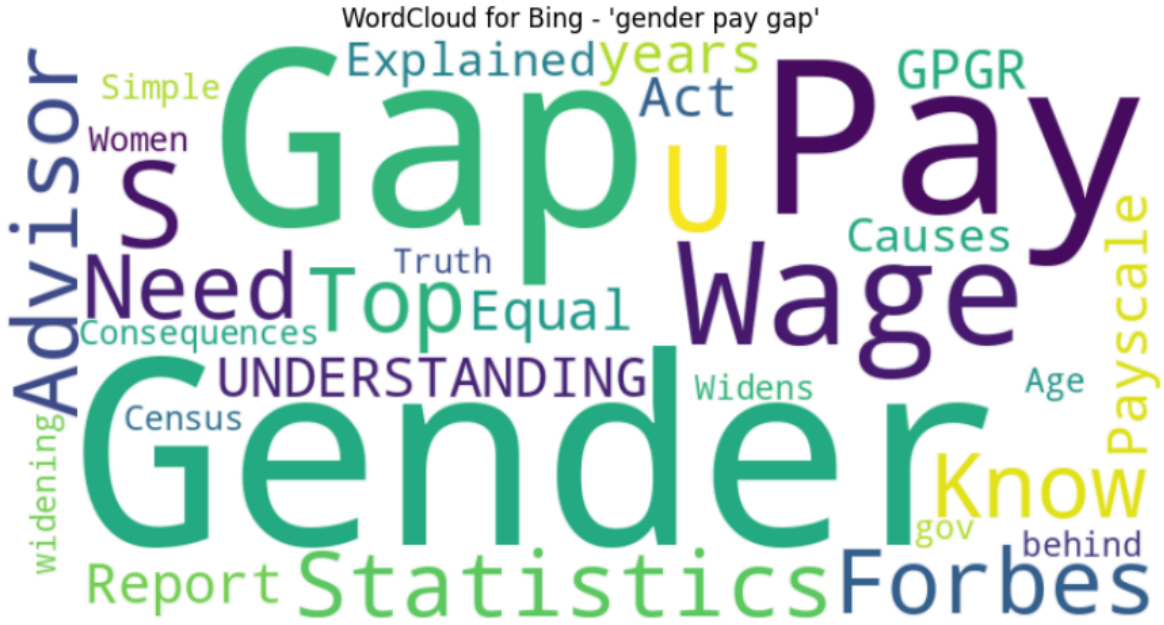
Topic: Hate Speech

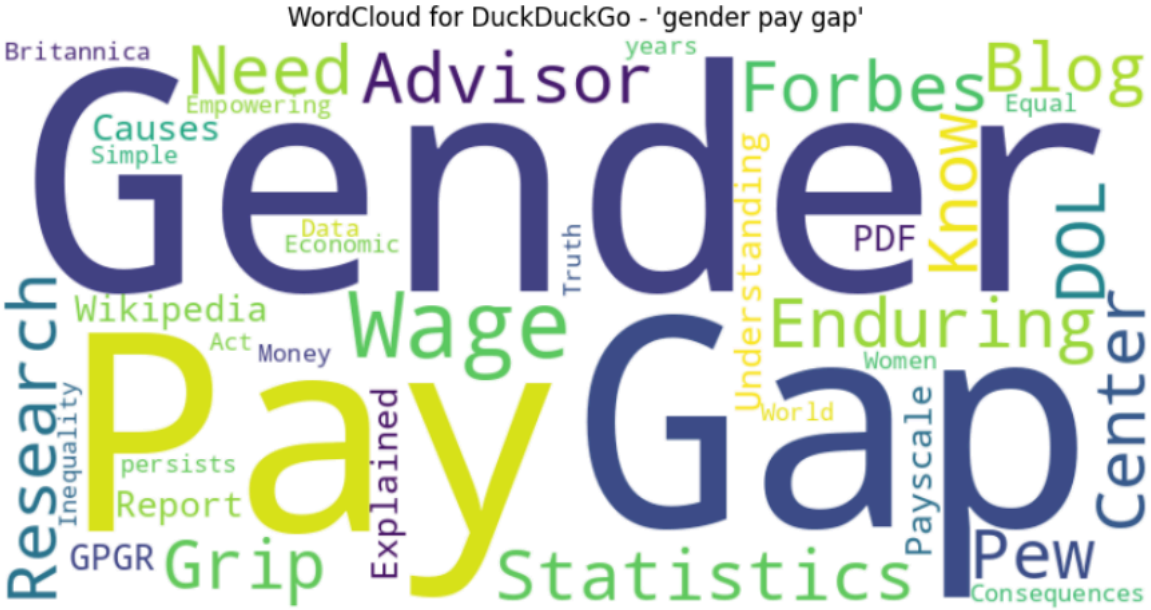


Topic: Black Girl

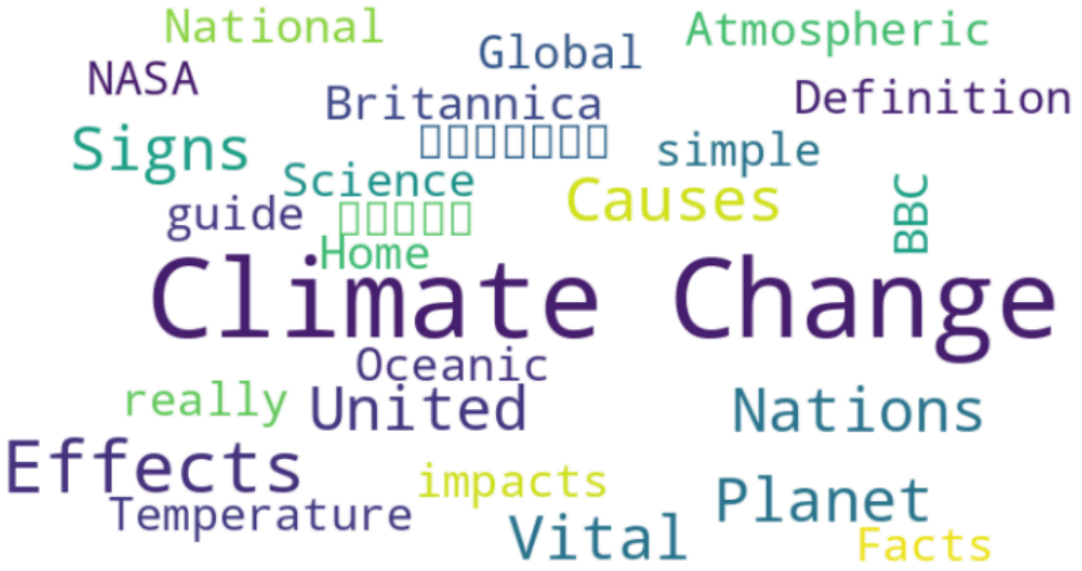




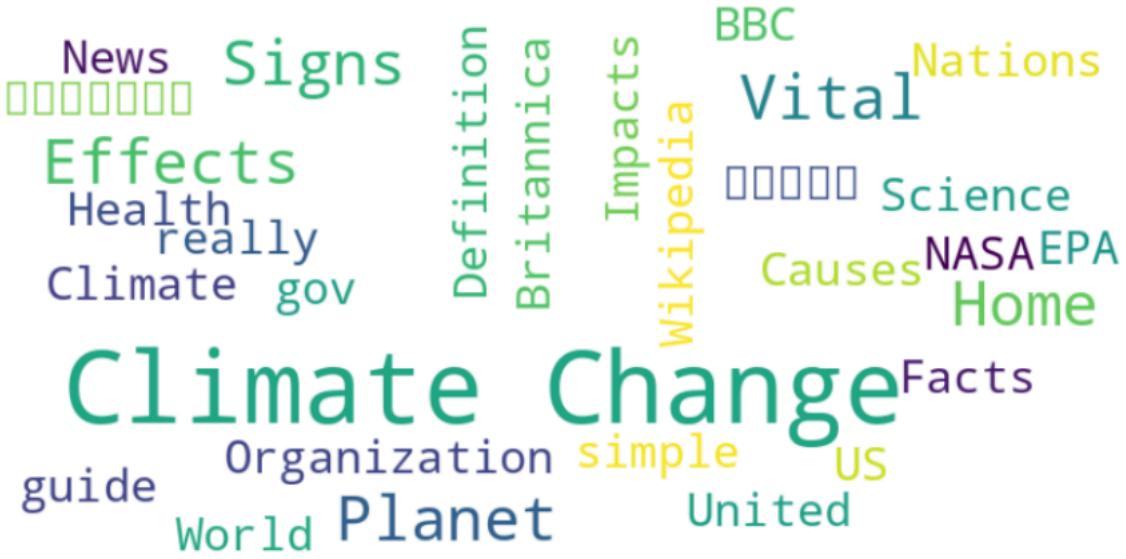

WordCloud for Bing - 'gender pay gap'
Gender
Gap
Pay
Wage
Statistics
Forbes
Need
Top
Advisor
Explained
years
Act
GPGR
Simple
Women
Causes
Truth
Equal
Consequences
UNDERSTANDING
Widens
Age
Census
Payscale
widening
Know
gov
behind
Report

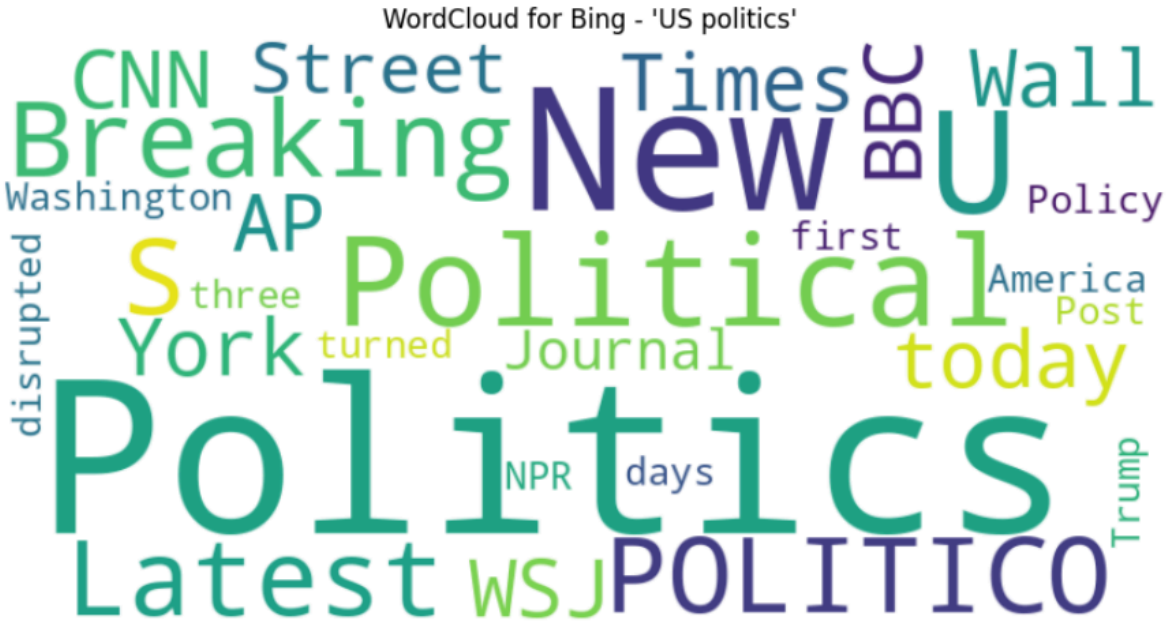

WordCloud for DuckDuckGo - 'gender pay gap'
Gender
Pay
Gap
Wage
Britannica
Need
Advisor
Forbes
Blog
years
Empowering
Causes
Simple
Equal
Know
Data
Economic
PDF
Truth
Understanding
DOL
Wikipedia
Enduring
Act
Money
Women
Research
Inequality
persists
Report
GPGR
Grip
Explained
Statistics
World
Payscale
Pew
Center
Consequences

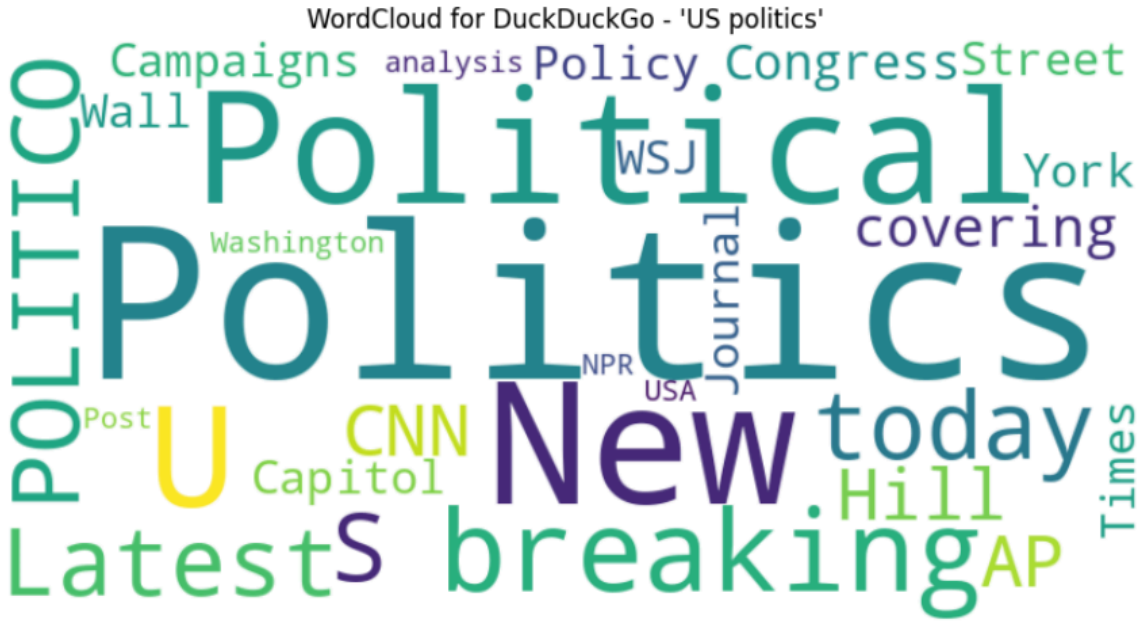
WordCloud for Bing - 'climate change'

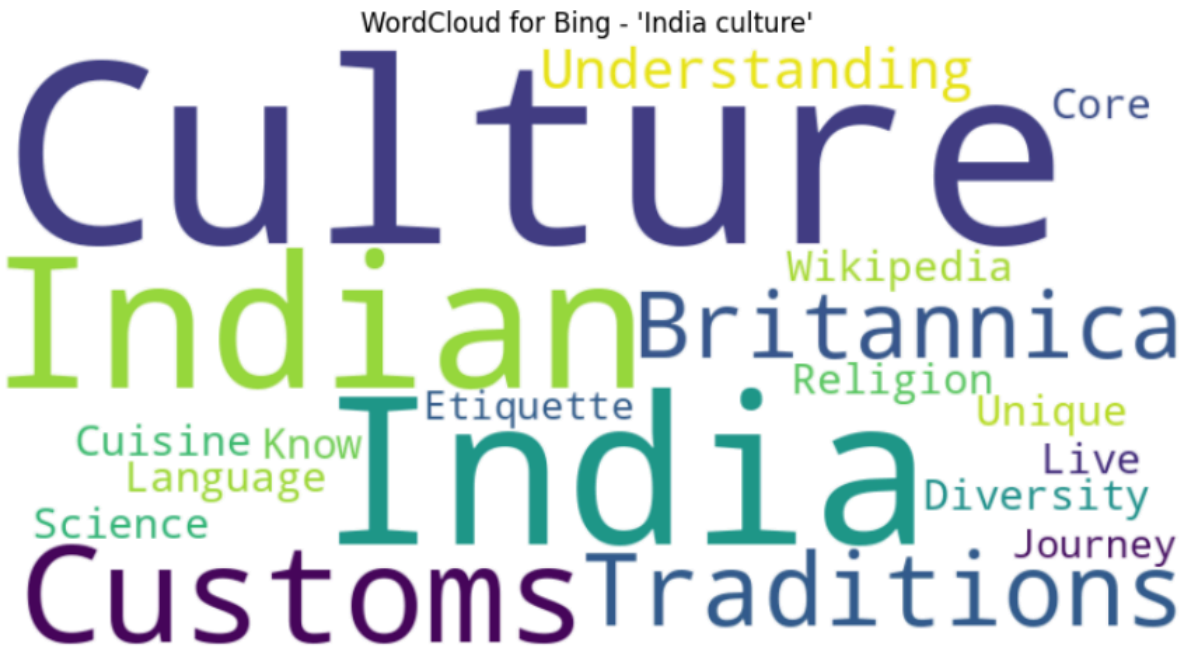


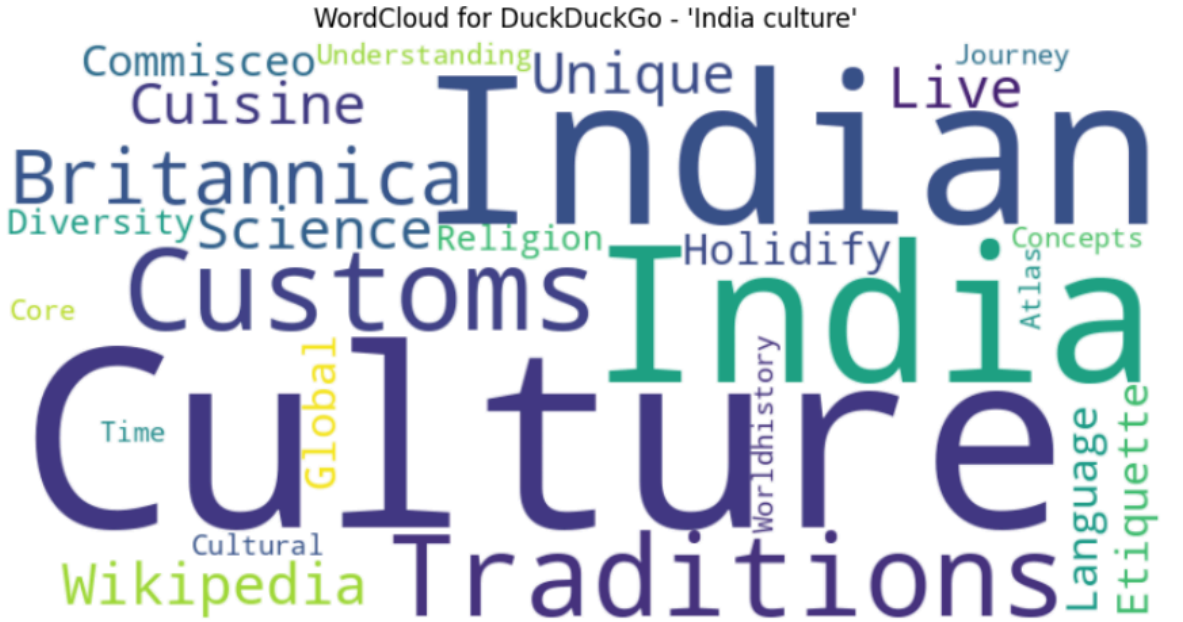
WordCloud for DuckDuckGo - 'climate change'




WordCloud for Bing - 'US politics'
CNN
Street
Times
Wall
Breaking
New
BBC
U
Washington
AP
Policy
first
S
Political
America
three
Post
York
turned
Journal
today
disrupted
Politics
NPR
days
Trump
Latest
WSJ
POLITICO

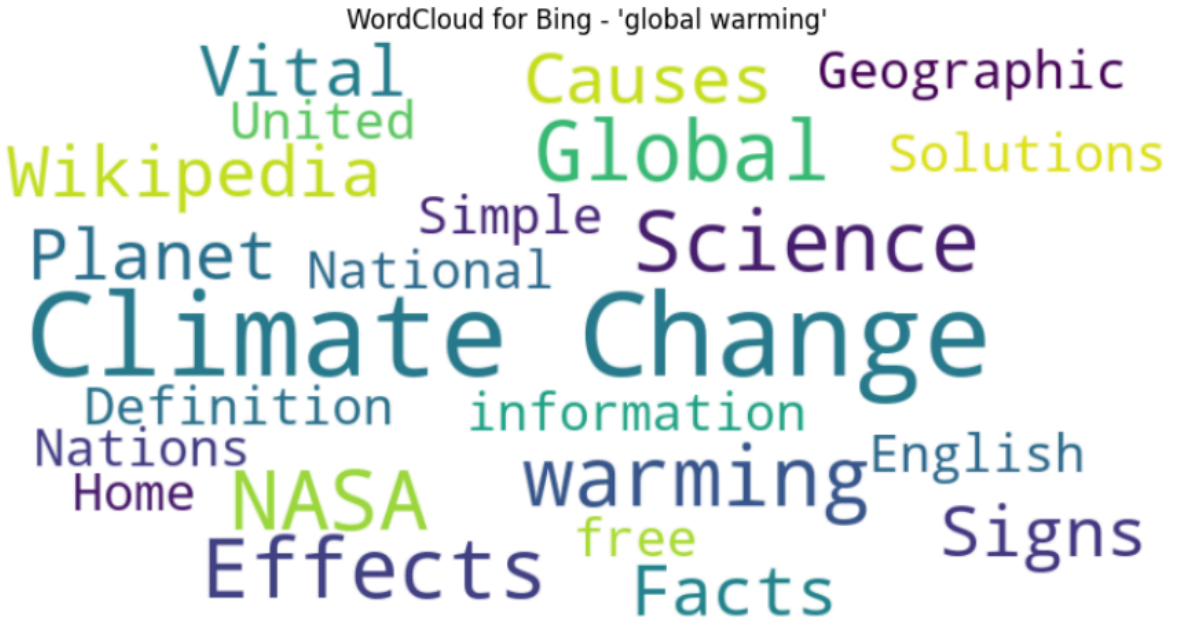

WordCloud for DuckDuckGo - 'US politics'
Campaigns
analysis
Policy
Congress
Street
POLITICO
Wall
Political
WSJ
York
covering
Politics
Washington
Journal
NPR
USA
Post
U
CNN
New
today
Capitol
Hill
Times
Latest
S
breaking
AP

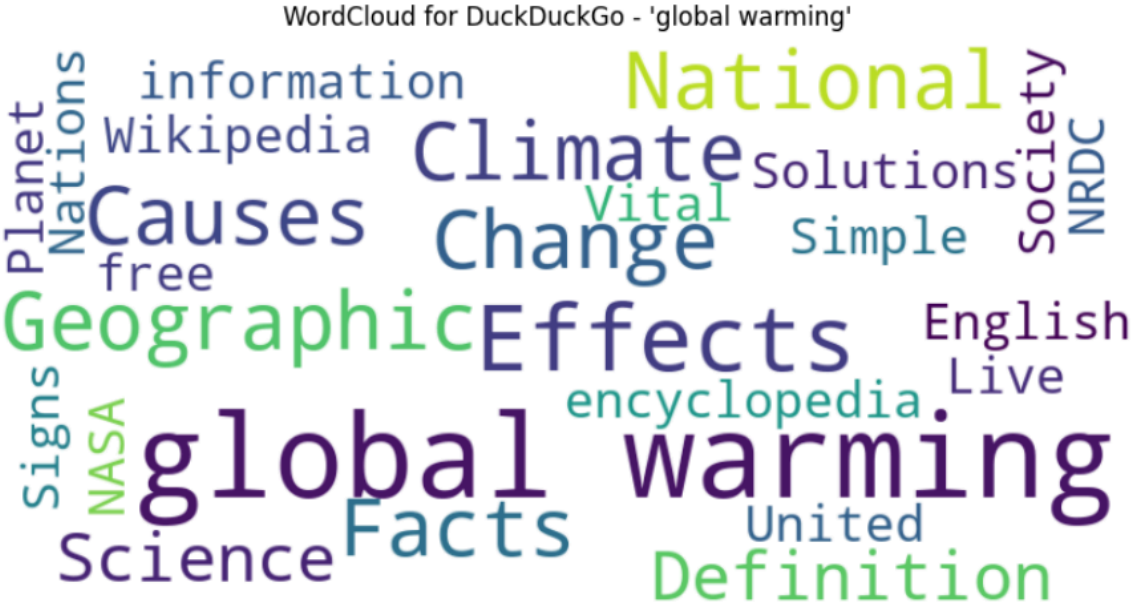

WordCloud for Bing - 'India culture'
Culture
Understanding
Core
Indian
Wikipedia
Britannica
Religion
Etiquette
Unique
India
Cuisine
Know
Language
Live
Diversity
Science
Journey
Customs
Traditions

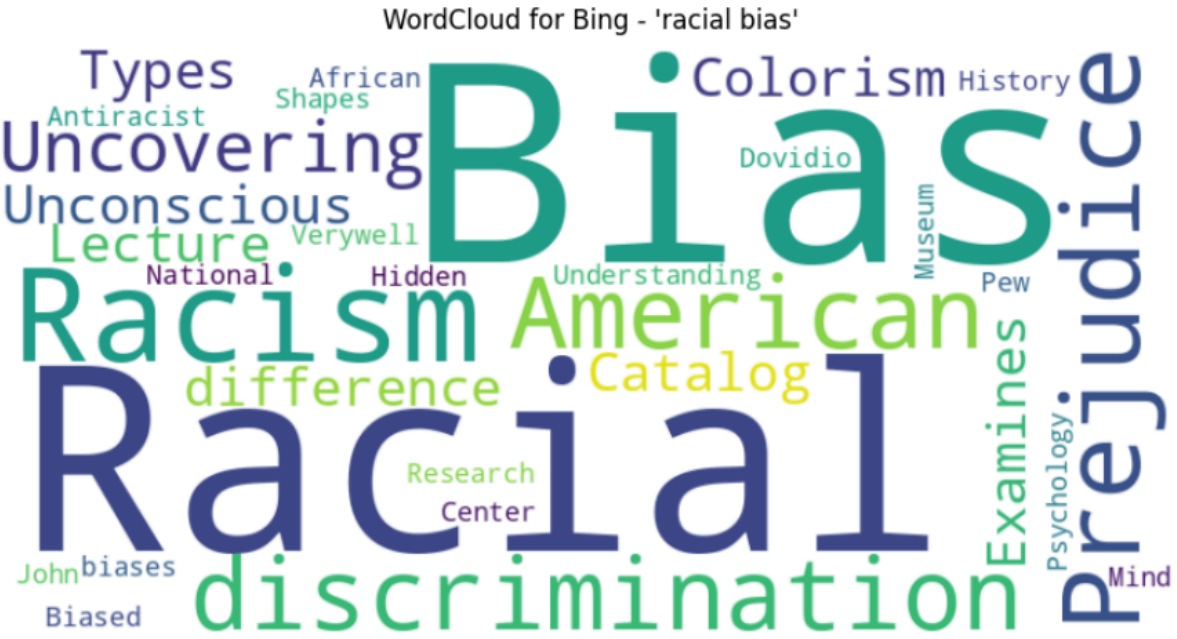

WordCloud for DuckDuckGo - 'India culture'
Commisceo
Understanding
Unique
Journey
Live
Cuisine
Indian
Britannica
Diversity
Science
Religion
Holidify
Concepts
Customs
India
Core
Atlas
Culture
Global
Worldhistory
Time
Language
Etiquette
Cultural
Wikipedia
Traditions

WordCloud for Bing - 'technology trends'

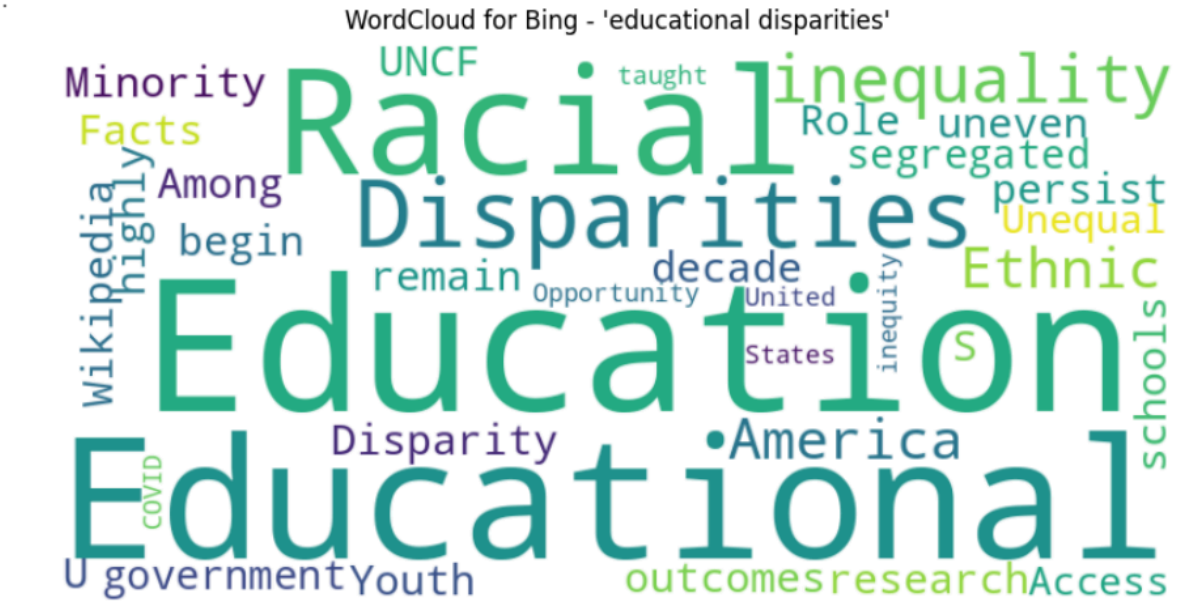
WordCloud for DuckDuckGo - 'technology trends'

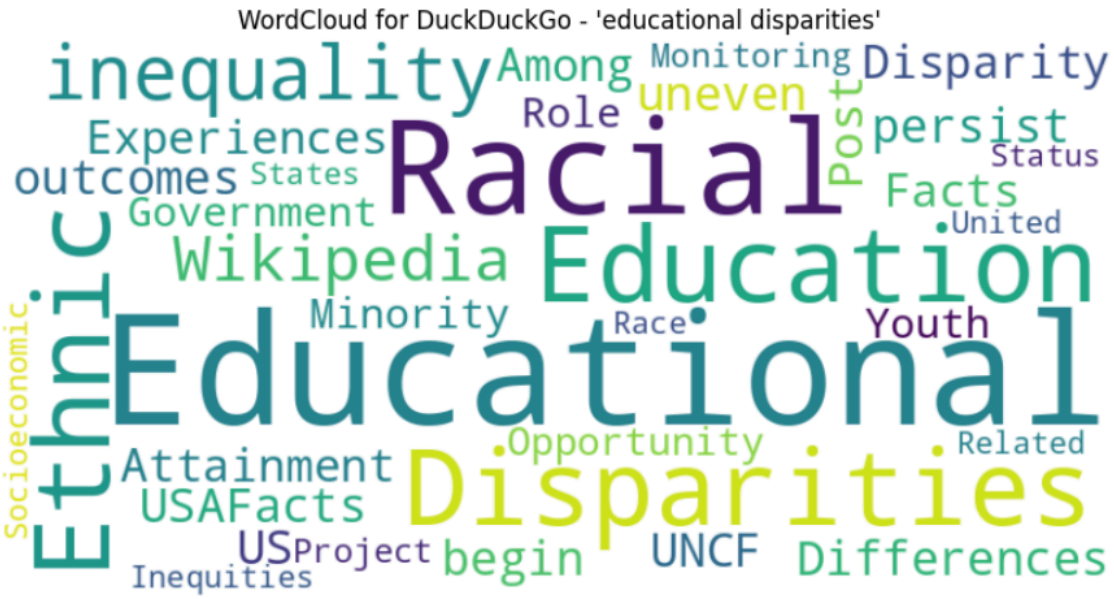

WordCloud for Bing - 'global warming'
Vital
Causes
Geographic
United
Global
Solutions
Wikipedia
Simple
Science
Planet
National
Climate Change
Definition
information
Nations
English
Home
NASA
warming
Signs
free
Effects
Facts


WordCloud for DuckDuckGo - 'global warming'
information
National
Planet
Nations
Wikipedia
Climate
Society
NRDC
Solutions
Causes
Vital
Change
Simple
free
Geographic
Effects
English
Live
encyclopedia
Signs
NASA
global warming
Facts
United
Science
Definition


WordCloud for Bing - 'racial bias'
Types
African
Shapes
Colorism
History
Antiracist
Uncovering
Bias
Dovidio
Unconscious
Museum
Lecture
Verywell
National
Hidden
Understanding
Pew
Racism
American
Prejudice
difference
Catalog
Racial
Examines
Psychology
Research
Center
John
biases
discrimination
Mind
Biased


WordCloud for DuckDuckGo - 'racial bias'
Examines
Psychological
discrimination
Racism
difference
Explained
Uncovering
Diversity
Unconscious
Types
Racial
Intro
Definition
Vocab
Verywell
Catalog
biases
approach
APA
systemic
Dovidio
Studies
Lecture
ThoughtCo
Psychology
bias
individuals
American
Mind
PhD
Prejudice
John
Ethnic
within
Stereotypes
Understanding
Association
Colorism


WordCloud for Bing - 'educational disparities'
Minority
UNCF
taught
Racial
inequality
Facts
Role
uneven
segregated
Among
Disparities
persist
Unequal
begin
Wikipedia
highly
remain
decade
Ethnic
Opportunity
United
inequity
Education
States
S
schools
Disparity
America
Educational
COVID
U
government
Youth
outcomes
research
Access


WordCloud for DuckDuckGo - 'educational disparities'
inequality
Among
Monitoring
Disparity
uneven
Role
Post
Experiences
Racial
persist
Status
outcomes
States
Facts
Government
United
Wikipedia
Education
Minority
Race
Youth
Ethnic
Educational
Socioeconomic
Opportunity
Related
Attainment
Disparities
USAFacts
US
Project
begin
UNCF
Differences
Inequities

B. Behavioral analysis using Drift Diffusion Models

Making the DDM model for the entire dataset based on the , Assumption that from Dataset only Product_Related_Duration and Informational_Duration influence the decision of the user to buy or to not to buy the product.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.stats import ks_2samp

# Step 1: Load the data
df = pd.read_csv("online_shoppers_intention.csv")

# Step 2: Filter data
df = df[df["Revenue"] == True]
df["RT"] = (df["Informational_Duration"] + df["ProductRelated_Duration"] + 1.0) / 1000.0#Converting to seconds
df = df[df["RT"] > 0]  # Remove invalid RTs

# Step 3: Outlier Removal
lower_bound = df["RT"].quantile(0.01)
upper_bound = df["RT"].quantile(0.99)
df = df[(df["RT"] >= lower_bound) & (df["RT"] <= upper_bound)]
```

Computing the Mean and Standard Deviation of the RT Column and then simulating the model try to fit the parameters and then trying to find Mean and Standard Deviation, to see if it is near to Actual Mean And Standard Deviation.

With the assumptions that the starting position is 0 and time_step = 0.01 (Smaller time steps provide finer resolution for the evidence accumulation process).

```python
# Step 4: Actual Mean And SD
mean_rt = df["RT"].mean()
std_rt = df["RT"].std()
print(f"Mean Response Time: {mean_rt:.2f} seconds")
print(f"Standard Deviation: {std_rt:.2f} seconds")
```

Simulation Function and Loss Function, Here a Assumption adding non_decision time with an Assumption that there might be some time which will be taken by the user to perceive the product and also the time spent after executing the decision of buying the product.

Simulation:

```python
# Step 5: DDM simulation
def simulate_ddm(trials, drift, noise, boundary, t_nd, time_step=0.001, max_time=2):
    reaction_times = []
    for _ in range(trials):
        x = 0  # Starting point
        t = 0
        while abs(x) < boundary and t < max_time:
            x += drift * time_step + noise * np.sqrt(time_step) * np.random.normal()
            t += time_step
        if abs(x) >= boundary and t < max_time:
            reaction_times.append(t + t_nd)  # Add non-decision time
    return np.array(reaction_times)
```

Standard Loss function Declaration:

```python
# Step 6: loss function Declaration
def loss_function(params, actual_rts):
    drift, noise, boundary, t_nd = params
    simulated_rts = simulate_ddm(
        trials=1000, drift=drift, noise=noise, boundary=boundary, t_nd=t_nd
    )
    sim_mean = simulated_rts.mean()
    sim_std = simulated_rts.std()
    # Loss is the squared error between mean and standard deviation
    mean_error = (sim_mean - actual_rts.mean()) ** 2
    std_error = (sim_std - actual_rts.std()) ** 2
    return mean_error + std_error
```

Initial Guesses and parameter bound assumptions:

```python
# Step 7: Intial Guesses and Parameter bound assumptions
initial_params = [0.4, 1.2, 2.5, 0.3]  # Initial guesses: drift, noise, boundary, t_nd
bounds = [(-1, 1), (0.1, 2), (0.5, 3), (0, 1)]  # Parameter bounds
result = minimize(loss_function, initial_params, args=(df["RT"]), bounds=bounds)
```

Driver code:

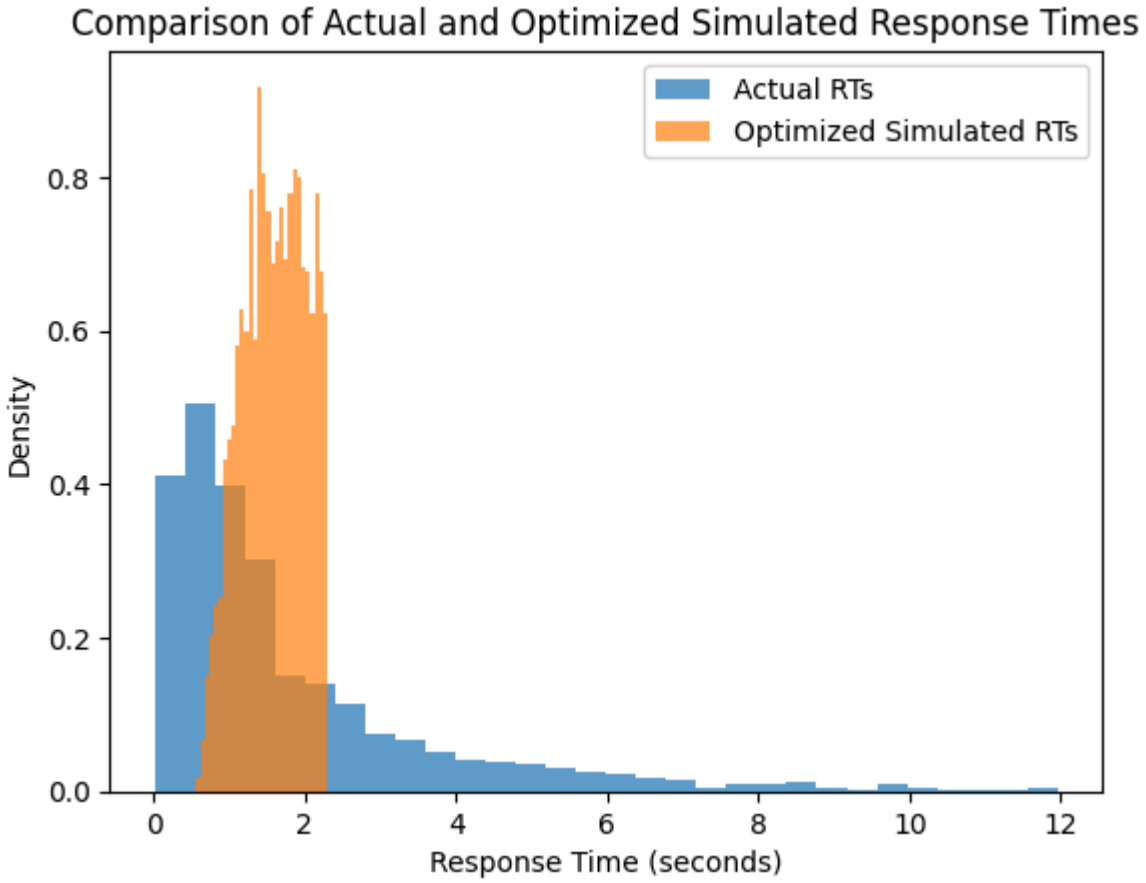
```python
#optimized parameters
fitted_drift, fitted_noise, fitted_boundary, fitted_t_nd = result.x
print(f"Fitted Drift: {fitted_drift:.3f}")
print(f"Fitted Noise: {fitted_noise:.3f}")
print(f"Fitted Boundary: {fitted_boundary:.3f}")
print(f"Fitted Non-Decision Time: {fitted_t_nd:.3f}")

# Step 8: Simulate DDM with optimized parameters
optimized_rts = simulate_ddm(
    trials=10000,  # Increased trials for smoother distribution
    drift=fitted_drift,
    noise=fitted_noise,
    boundary=fitted_boundary,
    t_nd=fitted_t_nd
)
```

```python
# Step 9: Compute mean and standard deviation of optimized RTs
sim_mean_rt = optimized_rts.mean()
sim_std_rt = optimized_rts.std()
print(f"Optimized Simulated Mean RT: {sim_mean_rt:.2f} seconds")
print(f"Optimized Simulated Standard Deviation: {sim_std_rt:.2f} seconds")

# Step 10: Validate the fit using KS test
ks_stat, p_value = ks_2samp(df["RT"], optimized_rts)
print(f"KS Statistic: {ks_stat:.3f}")

# Step 11: Visualization
plt.hist(df["RT"], bins=30, alpha=0.7, label="Actual RTs", density=True)
plt.hist(optimized_rts, bins=30, alpha=0.7, label="Optimized Simulated RTs", density=True)
plt.title("Comparison of Actual and Optimized Simulated Response Times")
plt.xlabel("Response Time (seconds)")
plt.ylabel("Density")
plt.legend()
plt.show()
```

Results And Statistics:

```
Mean Response Time: 1.82 seconds
Standard Deviation: 1.92 seconds
Fitted Drift: 0.400
Fitted Noise: 1.200
Fitted Boundary: 2.500
Fitted Non-Decision Time: 0.300
Optimized Simulated Mean RT: 1.59 seconds
Optimized Simulated Standard Deviation: 0.41 seconds
KS Statistic: 0.370
```

Visualization :



Interpretations of why failed :

1. **Response Time Distribution :**
   DDM assumes the RT distribution to be Normal or Lognormal but here the , The RT distribution of the given data set is left skewed which may due to Product_related_information and informationa_duration, these components are not directly representing the data.

2. **Non_decison_time** :
   The Informatin_Duration and Product_Related_Duration components are highly variable, making it difficult to separate them into decision-making time and non-decision time.

3. **Homogeneity of Different drift rates :**
   The dataset might include multiple product types or user behaviors, each with its own drift rate. Without modeling these differences explicitly, the DDM fit will fail.

Forming two groups based on familiar product purchase and unfamiliar product purchase

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.stats import ks_2samp

# Step 1: Load the data
df = pd.read_csv("online_shoppers_intention.csv")

# Step 2: Filter data
df = df[df["Revenue"] == True]
df["RT"] = (df["Informational_Duration"] + df["ProductRelated_Duration"] + 1.0) / 1000.0  # Convert to seconds
df = df[df["RT"] > 0]  # Remove invalid RTs
```

Group Forming on the assumptions that , Familiar products tend to have low Product_Related _Duration as they are familiar and the unfamiliar aren't.

```python
# Step 3: Median
median_duration = df["ProductRelated_Duration"].median()

# Group 1: Familiar products
familiar_products = df[df["ProductRelated_Duration"] <= median_duration]

# Group 2: Unfamiliar products
unfamiliar_products = df[df["ProductRelated_Duration"] > median_duration]
```

```python
# Step 4: Define DDM simulation function with non-decision time
def simulate_ddm(trials, drift, noise, boundary, t_nd, time_step=0.001, max_time=2):
    reaction_times = []
    for _ in range(trials):
        x = 0  # Starting point
        t = 0
        while abs(x) < boundary and t < max_time:
            x += drift * time_step + noise * np.sqrt(time_step) * np.random.normal()
            t += time_step
        if abs(x) >= boundary and t < max_time:
            reaction_times.append(t + t_nd)  # Add non-decision time
    return np.array(reaction_times)
```

```python
# Step 5: Define loss function
def loss_function(params, actual_rts):
    drift, noise, boundary, t_nd = params
    simulated_rts = simulate_ddm(
        trials=1000, drift=drift, noise=noise, boundary=boundary, t_nd=t_nd
    )
    sim_mean = simulated_rts.mean()
    sim_std = simulated_rts.std()
    # Compute loss as the squared error between mean and standard deviation
    mean_error = (sim_mean - actual_rts.mean()) ** 2
    std_error = (sim_std - actual_rts.std()) ** 2
    return mean_error + std_error
```

```python
# Step 6: Fit DDM
def fit_ddm(group_name, group_data):
    print(f"\nFitting DDM for {group_name} group...")

    # Optimize parameters for the group
    initial_params = [0.4, 1.2, 2.5, 0.3]  # Initial guesses
    bounds = [(-1, 1), (0.1, 2), (0.5, 3), (0, 1)]  # Parameter bounds
    result = minimize(loss_function, initial_params, args=(group_data["RT"]), bounds=bounds)

    # Extract optimized parameters
    fitted_drift, fitted_noise, fitted_boundary, fitted_t_nd = result.x
    print(f"Fitted Drift: {fitted_drift:.3f}")
    print(f"Fitted Noise: {fitted_noise:.3f}")
    print(f"Fitted Boundary: {fitted_boundary:.3f}")
    print(f"Fitted Non-Decision Time: {fitted_t_nd:.3f}")

    # Simulate RTs for the group
    simulated_rts = simulate_ddm(
        trials=10000, drift=fitted_drift, noise=fitted_noise, boundary=fitted_boundary, t_nd=fitted_t_nd
    )
```

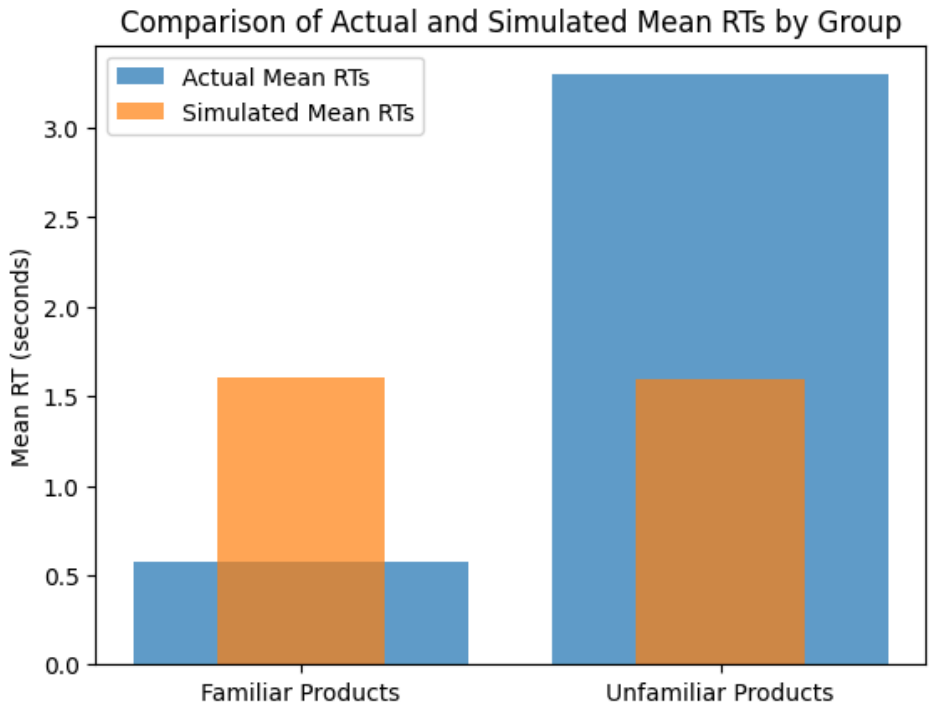
```python
    actual_mean = group_data["RT"].mean()
    actual_std = group_data["RT"].std()
    sim_mean = simulated_rts.mean()
    sim_std = simulated_rts.std()
    ks_stat, p_value = ks_2samp(group_data["RT"], simulated_rts)

    print(f"Actual Mean RT: {actual_mean:.2f} seconds, Actual Std RT: {actual_std:.2f} seconds")
    print(f"Simulated Mean RT: {sim_mean:.2f} seconds, Simulated Std RT: {sim_std:.2f} seconds")
    print(f"KS Statistic: {ks_stat:.3f}, P-value: {p_value:.3f}")

    return {
        "Group": group_name,
        "Fitted Drift": fitted_drift,
        "Fitted Noise": fitted_noise,
        "Fitted Boundary": fitted_boundary,
        "Fitted Non-Decision Time": fitted_t_nd,
        "Actual Mean RT": actual_mean,
        "Actual Std RT": actual_std,
        "Simulated Mean RT": sim_mean,
        "Simulated Std RT": sim_std,
        "KS Statistic": ks_stat,
        "P-Value": p_value
    }
```

```python
# Step 7: Fit models for familiar and unfamiliar products
familiar_results = fit_ddm("Familiar Products", familiar_products)
unfamiliar_results = fit_ddm("Unfamiliar Products", unfamiliar_products)

# Step 8: Visualization
groups = ["Familiar Products", "Unfamiliar Products"]
mean_rts = [familiar_results["Actual Mean RT"], unfamiliar_results["Actual Mean RT"]]
sim_mean_rts = [familiar_results["Simulated Mean RT"], unfamiliar_results["Simulated Mean RT"]]

plt.bar(groups, mean_rts, alpha=0.7, label="Actual Mean RTs")
plt.bar(groups, sim_mean_rts, alpha=0.7, label="Simulated Mean RTs", width=0.4)
plt.title("Comparison of Actual and Simulated Mean RTs by Group")
plt.ylabel("Mean RT (seconds)")
plt.legend()
plt.show()
```

Results:

```
Fitting DDM for Familiar Products group...
Fitted Drift: 0.400
Fitted Noise: 1.200
Fitted Boundary: 2.500
Fitted Non-Decision Time: 0.300
Actual Mean RT: 0.57 seconds, Actual Std RT: 0.32 seconds
Simulated Mean RT: 1.60 seconds, Simulated Std RT: 0.41 seconds
KS Statistic: 0.834

Fitting DDM for Unfamiliar Products group...
Fitted Drift: 0.400
Fitted Noise: 1.200
Fitted Boundary: 2.500
Fitted Non-Decision Time: 0.300
Actual Mean RT: 3.30 seconds, Actual Std RT: 2.72 seconds
Simulated Mean RT: 1.59 seconds, Simulated Std RT: 0.41 seconds
KS Statistic: 0.510
```

Visualization:



Interpretations from the data:

Smaller Actual mean of familiar product say that familiar products have quicker purchase time , but after fitting the DDM model it is not showing the same result.

1. Drift Rate for both the familiar and unfamiliar products were the same so we can say from the fitted model that evidence accumulation is the same.
2. The non-decision time for both the groups is the same .
3. Familiar products have very low actual mean as compared to unfamiliar products which says that they are purchased quicker , also the low actual standard deviation tell us that there is very less variability.
4. While the simulated mean for both the groups is significantly different , which tells us that the model was not a proper fit at all.
5. The KS statistic value for familiar products is much higher which says that the model is having trouble replicating the original data.

Why the interpretations when wrong:

1. **Noise Accumulation assumption :**
   Shopping decisions might involve sudden, non-linear shifts in preference (e.g., after seeing a discount or realizing a product feature), which deviate from gradual evidence accumulation.

2. **Non-Decision Time :**
   Non-Decision time could vary significantly in e-commerce , Complex interfaces or unfamiliar products might increase Non-decision time.

3. **The problem statement tell us to model only for Revenue = True:**
   The data is filtered for Revenue = True , meaning we only consider "Buy" decisions. There's no explicit modeling of the alternative ("Not Buy"), which removes one side of the binary decision-making process.