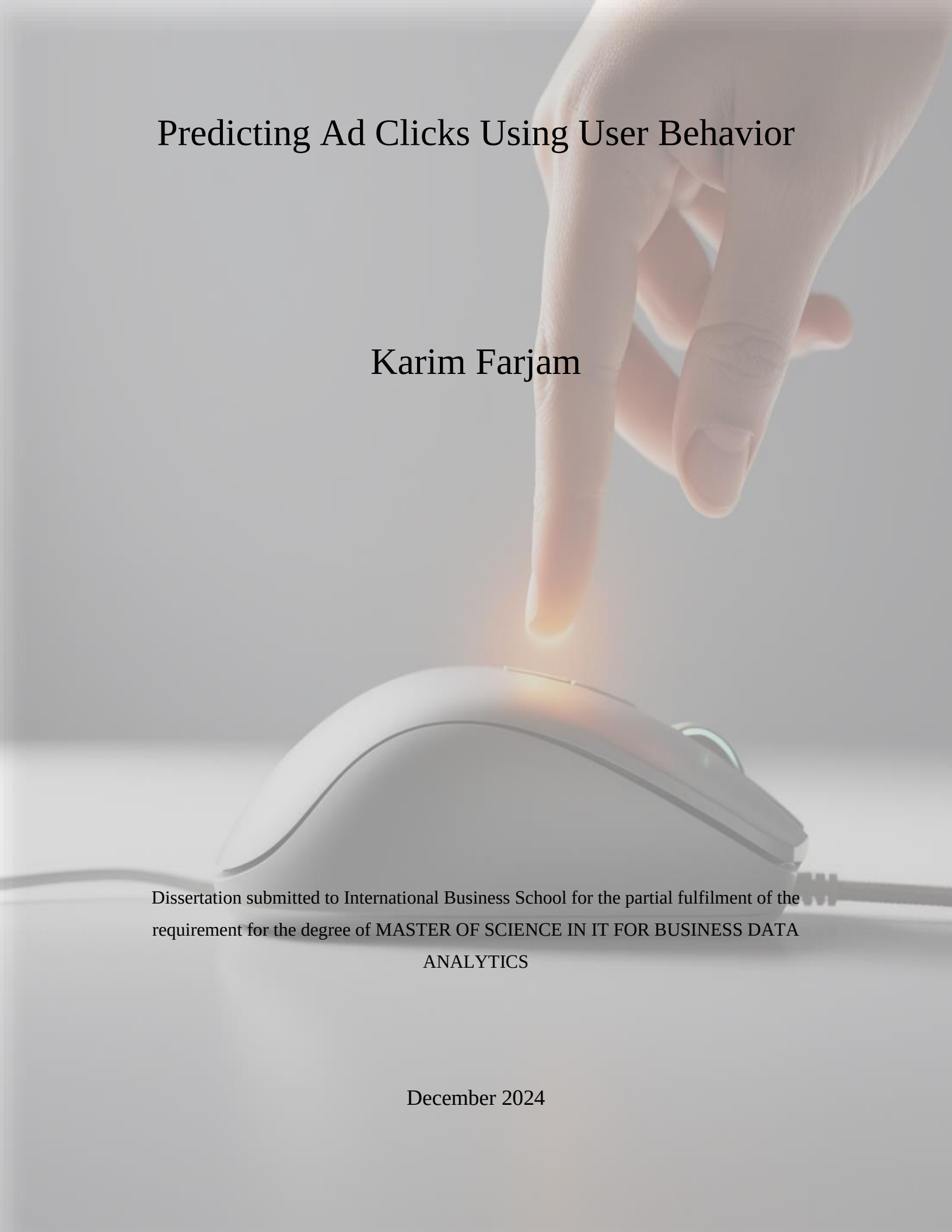


Predicting Ad Clicks Using User Behavior

Karim Farjam

Dissertation submitted to International Business School for the partial fulfilment of the requirement for the degree of MASTER OF SCIENCE IN IT FOR BUSINESS DATA ANALYTICS

December 2024

DECLARATION

This dissertation is a product of my own work and is the result of nothing done in collaboration.

I consent to International Business School's free use including/excluding online reproduction, including/excluding electronically, and including/excluding adaptation for teaching and education activities of any whole or part item of this dissertation.

Karim Farjam

Word length: 11,519 words

Contents

Introduction:

One of the most important aspects of business is advertising. For many years, it has played a critical role in shaping business success. Although methods of advertising have evolved over time, its importance remains unchanged.

In the past, companies relied on traditional media such as billboards, newspapers, magazines, and radio to promote their products and increase revenue. However, with the advent of technology and the internet, advertising strategies have undergone significant shifts. Nowadays, most people spend a considerable amount of time online, engaging in activities ranging from studying to paying bills. To align with this shift—and contribute to environmental conservation by reducing paper usage—digital advertising has become the preferred choice.

In our goal is to predict whether users will click on advertisements. To achieve this, we aim to build a predictive model that accurately determines whether an advertisement was clicked (ad_clicked). We are provided with a folder containing CSV files that track user activity on the webpage.

Given the limitations of this project, we must apply two filters: first, include only users who spent at least 5 seconds on the site. Second, use only the first 5 seconds of cursor movement data to predict whether a user clicked on an ad (ad_clicked). This approach ensures that users did not land on the site accidentally. For the final dataset intended for modeling, certain features such as age, income, gender, attention, and education will need to be dropped. These features will be retained for exploratory data analysis (EDA) but will be excluded afterward to focus on the most relevant variables for prediction.

This approach can reveal two possibilities, the advertisement may have been highly attractive, prompting the user to click within 5 seconds, or the user may have clicked on the ad accidentally. Both scenarios provide valuable insights to help us achieve our objective of understanding and optimizing user engagement with advertisements.

For the models Using classifier algorithms is better for predicting ad clicks because target variable (ad_clicked) is a binary classification problem (values typically represented as 0 for "no click" and 1 for "clicked").

Read Dataset:

First we have to load the datasets:

```
# Use gdown to download the file from Google Drive
!gdown "https://drive.google.com/uc?id=1woOqHNNHLqAZc4QE6CmoedDrFndT1LDA"
-O adv.zip

# Unzip the downloaded file
!unzip adv.zip
```

```
# create new folder
!mkdir -p /content/click_ad

# Path to the new uploaded zip file and extract destination
import zipfile
import os


uploaded_zip_path = '/content/adv.zip'
extract_dir = '/content/click_ad/'

# Extract the zip file
with zipfile.ZipFile(uploaded_zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

# List the contents of the extracted folder to see the structure
extracted_files = os.listdir(extract_dir)
extracted_files
```

```
['participants.tsv', 'groundtruth.tsv', 'logs']
```

gdown is a command-line tool used to download files from Google Drive using their unique file IDs. Since the downloaded file is a ZIP archive, it needs to be unzipped before its contents can be accessed and used. For better control, I saved all the data in separate folder and we can see we have all files and folder we need(participants.tsv, groundruth.tsv and logs).

```
import pandas as pd
import os

# Load the groundtruth and participants data
groundtruth_path = os.path.join(extract_dir, 'groundtruth.tsv')
participants_path = os.path.join(extract_dir, 'participants.tsv')

groundtruth_df = pd.read_csv(groundtruth_path, sep='\t')
participants_df = pd.read_csv(participants_path, sep='\t')
```

```
groundtruth_df.head()
```

	user_id	ad_clicked	attention	log_id
0	5npsk114ba8hfbj4jr3lt8jhf5	0	4	20181002033126
1	5o9js8slc8rg2a8mo5p3r93qm0	1	5	20181001211223
2	pi17qjfqmnhpsiahbumcsdq0r6	0	4	20181001170952
3	3rptg9g7l83imkbdsu2miignv7	0	1	20181001140754
4	049onniafv6fe4e6q42k6nq1n2	0	1	20181001132434

Groundtruth Dataset Description:

Variable	Description
user_id	(string) Participant's ID
ad_clicked	(int) Whether the participant clicked on the ad (1) or not (0).
attention	(int) Self-reported attention score, in 1-5 Likert-type scale (1 denotes no attention)
log_id	(string) Mouse tracking log ID

```
participants_df.head()
```

	user_id	country	education	age	income	gender	ad_position	ad_type	ad_category	serp_id	query	log_id
0	5npsk114ba8hfbj4jr3lt8jhf5	PHL	3	3	1	male	top-left	dd	Computers & Electronics	tablets	tablets	20181002033126
1	5o9js8slc8rg2a8mo5p3r93qm0	VEN	3	1	1	male	top-right	dd	Shop - Luxury Goods	casio-watches	casio watches	20181001211223
2	pi17qjfqmnhpsiahbumcsdq0r6	VEN	2	3	1	male	top-left	native	Shop - Luxury Goods	chivas-regal	chivas regal	20181001170952
3	3rptg9g7l83imkbdsu2miignv7	VEN	3	2	1	male	top-right	dd	Shop - Luxury Goods	chivas-regal	chivas regal	20181001140754
4	049onniafv6fe4e6q42k6nq1n2	VEN	3	5	1	male	top-left	native	Autos & Vehicles	audi-r8-used	audi r8 used	20181001132434

Participants Dataset Description:

Variable	Description
user_id	(string) Participant's ID
country	(string) Participant's country
education	(int) Level of education, according to 6 bins: 1: High school 2: College 3: Bachelor's 4: Graduate 5: Master's 6: Doctorate
Age	(int) Participant's age, according to 9 bins: 1: 18-23 2: 24-29 3: 30-35 4: 36-41 5: 42-47 6: 48-53 7: 54-59 8: 60-65 9: +66
Income	(int) Level of income, according to 8 bins: 1: 25K 2: 25-34K 3: 35-49K 4: 50-74K 5: 75-99K 6: 100-149K

	7: 150-249K 8: +250K
Gender	(string) Participant's gender.
Ad_position	(string) Ad stimulus position.
Ad_type	(string) Ad stimulus type.
Ad_category	(string) Ad stimulus category.
Serp_id	(string) Ad SERP identifier.
Query	(string) Ad stimulus query.

```
unique_participants = participants_df['user_id'].nunique()
unique_groundtruth = groundtruth_df['user_id'].nunique()

print(f"Number of unique participants in participants.tsv: {unique_participants}")
print(f"Number of unique groundtruth in groundtruth.tsv: {unique_groundtruth}")
```

```
Number of unique participants in participants.tsv: 2909
Number of unique groundtruth in groundtruth.tsv: 2909
```

To make sure that user_id values are unique and that the set of user_id's is the same in the two datafiles.

Logs folder:

```
log_files = os.listdir(os.path.join(extract_dir, 'logs'))
log_files[:5]
```

```
['20170425152248.csv',
 '20170222190241.csv',
 '20170131134842.csv',
 '20170211165250.csv',
 '20181001172227.csv']
```

for better understanding is better to checking the files in the logs folder to examine one sample file and Display the first few log files for an overview. As we can see here its seems that the first part of log files is common with the log_id in participants_df and groundtruth_df, so we can use it in the next steps.

```
len(log_files)
```

```
2909
```

we have 2909 csv file in our log folder exactly same with unique_participants, unique_groundtruth.

Now we want to load one of the log files to know about the information of users in csv file.

Logs dataset Description:

```
sample_log_path = os.path.join(extract_dir, 'logs', log_files[0]) # Remove skiprows=1 from here
sample_log_df = pd.read_csv(sample_log_path, delimiter=' ', header=None,
                            names=["cursor", "timestamp", "xpos", "ypos", "event", "xpath", "attrs", "extras"], skiprows=1)

# Display the first few rows of the sample log file
sample_log_df.head()
```

	cursor	timestamp	xpos	ypos	event	xpath	attrs	extras
0	0	1493126819126	0	0	load	/	{}	{}
1	0	1493126819895	374	150	mousedown	//*[@id='rso']/div[1]/div/div[1]/div/h3/a	{}	{"topRight":900,"topLeft":446,"bottomRight":93...
2	0	1493126819964	374	150	mouseup	//*[@id='rso']/div[1]/div/div[1]/div/h3/a	{}	{"topRight":900,"topLeft":446,"bottomRight":93...
3	0	1493126819965	374	150	click	//*[@id='rso']/div[1]/div/div[1]/div/h3/a	{}	{"topRight":900,"topLeft":446,"bottomRight":93...
4	0	1493126819965	0	0	beforeunload	/	{}	{}

Variable	Description
Cursor	(int) This column is always 0 because all participants used a regular computer mouse.
Timestamp	(int) This column is always 0 because all participants used a regular computer mouse.
Xpos	(float) X position of the mouse cursor
Ypos	(float) Y position of the mouse cursor
Event	(string) Browser's event name; e.g. load, mousemove, click, etc.
Xpath	string) Target element that relates to the event, in XPath notation.
Attrs	(string) Optional. Element attributes, if any.
Extras	(string) Optional. A JSON string with Euclidean distances to different reference points of the ad's bounding box

Now we want to filter our CSV files to include only users who spent at least 5 seconds on the webpage.

```
log_durations = []  # List to store log_id and time spent

# Loop over each log file to calculate the duration
for log_file in log_files:
    # Load each log file
    log_df = pd.read_csv(os.path.join(extract_dir, 'logs', log_file), delimiter=' ', header=None,
                         names=["cursor", "timestamp", "xpos", "ypos", "event", "xpath", "attrs", "extras"])

    # Skip files with header issues by checking the first row's timestamp column
    if log_df["timestamp"].iloc[1:].apply(lambda x: str(x).isdigit()).all():
        # Calculate the time duration (in seconds) as the difference between the max and min timestamp
        min_time = log_df["timestamp"].iloc[1:].astype(int).min()
        max_time = log_df["timestamp"].iloc[1:].astype(int).max()
        duration = (max_time - min_time) / 1000  # Convert milliseconds to seconds

        # Append the result if duration is at least 5 seconds
        if duration >= 5:
            log_durations.append((log_file.split('.')[0], duration))

# Convert to DataFrame
log_duration_df = pd.DataFrame(log_durations, columns=["log_id", "duration"])

# Display users who spent at least 5 seconds on the page
log_duration_df.head()
```

(Anon., n.d.)

	log_id	duration
0	20170211165250	6.680
1	20181001172227	13.380
2	20170425124359	13.728
3	20180827201856	9.547
4	20170312025352	31.810

To ensure that all CSV files have a duration time of more than 5 seconds, we check the length of log_duration_df to see if there are any CSV files with a duration less than 5 seconds.

```
len(log_duration_df[log_duration_df['duration'] < 5])
```

```
0
```

As we can see the length of log_duration_df is 0 it mean that all of filtered users spend at least five seconds in webpage.

```
print('number of csv file before filter 5 seconds activity',len(log_files))
print('number of csv file after filter 5 seconds activity',len(log_duration_df))
print('number of csv file less than 5 secondes avtivity:' ,len(log_files) - len(log_duration_df))
```

```
number of csv file before filter 5 seconds activity 2909
number of csv file after filter 5 seconds activity 2501
number of csv file less than 5 secondes avtivity: 408
```

408 users spend less than five seconds in webpage and will be remove in our dataset.

```
log_duration_df.dtypes
```

	0
log_id	object
duration	float64

dtype: object

```
# Convert 'log_id' in log_duration_df to integer for alignment with groundtruth and participants data
log_duration_df['log_id'] = log_duration_df['log_id'].astype(int)

# Re-filter groundtruth and participants data using the updated integer log_id
final_groundtruth_df = groundtruth_df[groundtruth_df['log_id'].isin(log_duration_df['log_id'])]
final_participants_df = participants_df[participants_df['log_id'].isin(log_duration_df['log_id'])]
```

First we check the dtypes of log_duration_df because we need to use log_id for two others datasets and we need just that users spend more five seconds in webpage, because of this we filter groundtruth_df and participants_df contains only this users.

```
final_groundtruth_df.shape, final_participants_df.shape
```

```
((2501, 4), (2501, 12))
```

we can see the shape reduce from 2909 to 2501 like log_duration_df.

We filtered data to users who spent at least 5 seconds on the webpage. now we want to Filtering the logs data to retain only the first 5 seconds for each user based on the filtered log IDs.

```python
# Initialize a list to hold the filtered cursor data
filtered_cursor_data = []

# Loop through each qualifying log_id to filter its corresponding file
for log_id in log_duration_df['log_id']:
    log_file_path = os.path.join(extract_dir, 'logs', f"{log_id}.csv")

    # Check if the log file exists (some log IDs may not have a corresponding file)
    if os.path.exists(log_file_path):
        # Load the log file
        log_df = pd.read_csv(log_file_path, delimiter=' ', header=None,
                             names=["cursor", "timestamp", "xpos", "ypos", "event", "xpath", "attrs", "extras"])

        # Convert timestamp column to integer and filter the first 5 seconds of data
        log_df = log_df.iloc[1:]  # Remove the header row if it's incorrectly included
        log_df["timestamp"] = log_df["timestamp"].astype(int)

        # Calculate the starting timestamp and filter rows within the first 5 seconds
        start_time = log_df["timestamp"].min()
        log_df_5sec = log_df[log_df["timestamp"] <= start_time + 5000]

        # Add the filtered data along with the log_id for identification
        log_df_5sec["log_id"] = log_id  # Add a log_id column
        filtered_cursor_data.append(log_df_5sec)

# Concatenate all filtered cursor data into a single DataFrame
filtered_cursor_data_df = pd.concat(filtered_cursor_data, ignore_index=True)
```

```python
filtered_cursor_data_df.head(20)
```

	cursor	timestamp	xpos	ypos	event	xpath	attrs	extras	log_id
0	0	1538456374085	0	0	load	/	{}	{}	20181002070022
1	0	1538456378799	0	0	focus	/	{}	{}	20181002070022
2	0	1485988833930	0	0	load	/	{}	{}	20170201233856
3	0	1485988833941	0	0	select	//*[@id='lst-ib']	{}	{}	20170201233856
4	0	1485988834404	722	354	mousemove	//*[@id='rso']/div[1]/div[1]/div/h3	{}	{"topRight":249,"topLeft":644,"bottomRight":39...	20170201233856
5	0	1485988834554	1076	172	mousemove	//*[@id='rcnt']	{}	{"topRight":338,"topLeft":950,"bottomRight":36...	20170201233856
6	0	1485988834854	1179	138	mousemove	//*[@id='rcnt']	{}	{"topRight":436,"topLeft":1051,"bottomRight":4...	20170201233856
7	0	1485988835195	1365	67	mouseover	/html	{}	{"topRight":622,"topLeft":1238,"bottomRight":6...	20170201233856
8	0	1485988835196	1365	67	mousedown	/html	{}	{"topRight":622,"topLeft":1238,"bottomRight":6...	20170201233856
9	0	1485988835359	0	0	scroll	/	{}	{}	20170201233856
10	0	1485988835535	0	0	scroll	/	{}	{}	20170201233856
11	0	1485988835693	0	0	scroll	/	{}	{}	20170201233856
12	0	1485988835833	1364	415	mouseup	/html	{}	{"topRight":693,"topLeft":1274,"bottomRight":6...	20170201233856
13	0	1485988836136	1328	416	mouseover	//*[@id='cnt']	{}	{"topRight":661,"topLeft":1239,"bottomRight":5...	20170201233856
14	0	1485988836481	725	464	mousemove	//*[@id='rso']/div[1]/div[2]/div/h3	{}	{"topRight":359,"topLeft":696,"bottomRight":14...	20170201233856
15	0	1485988836635	558	446	mousemove	//*[@id='rso']/div[1]	{}	{"topRight":388,"topLeft":548,"bottomRight":22...	20170201233856
16	0	1485988836823	540	451	mousemove	//*[@id='rso']/div[1]/div[2]/div/h3/a	{}	{"topRight":401,"topLeft":537,"bottomRight":24...	20170201233856
17	0	1485988837177	0	0	blur	/	{}	{}	20170201233856
18	0	1486784235772	0	0	load	/	{}	{}	20170211043537
19	0	1486784238795	0	0	focus	/	{}	{}	20170211043537

Display the first 20 rows of the filtered cursor data. To ensure we are using only the first 5 seconds of each user's data, we check the timestamps. For example, the user with log_id 20170201233856 has the first timestamp as 1485988837177 and the last timestamp as 1485988833930. Also we can see some users did not have any activity in first five seconds.

```
print('difference of first and last timestamp for user with log id 20170201233856 is:', (1485988837177 - 1485988833930)/1000,'s')
```

```
difference of first and last timestamp for user with log id 20170201233856 is: 3.247 s
```

Now we need to check the data types in filtered_cursor_data_df, as we plan to use these features for visualization and, ultimately, for feature engineering to create new features.

```
filtered_cursor_data_df.dtypes
```

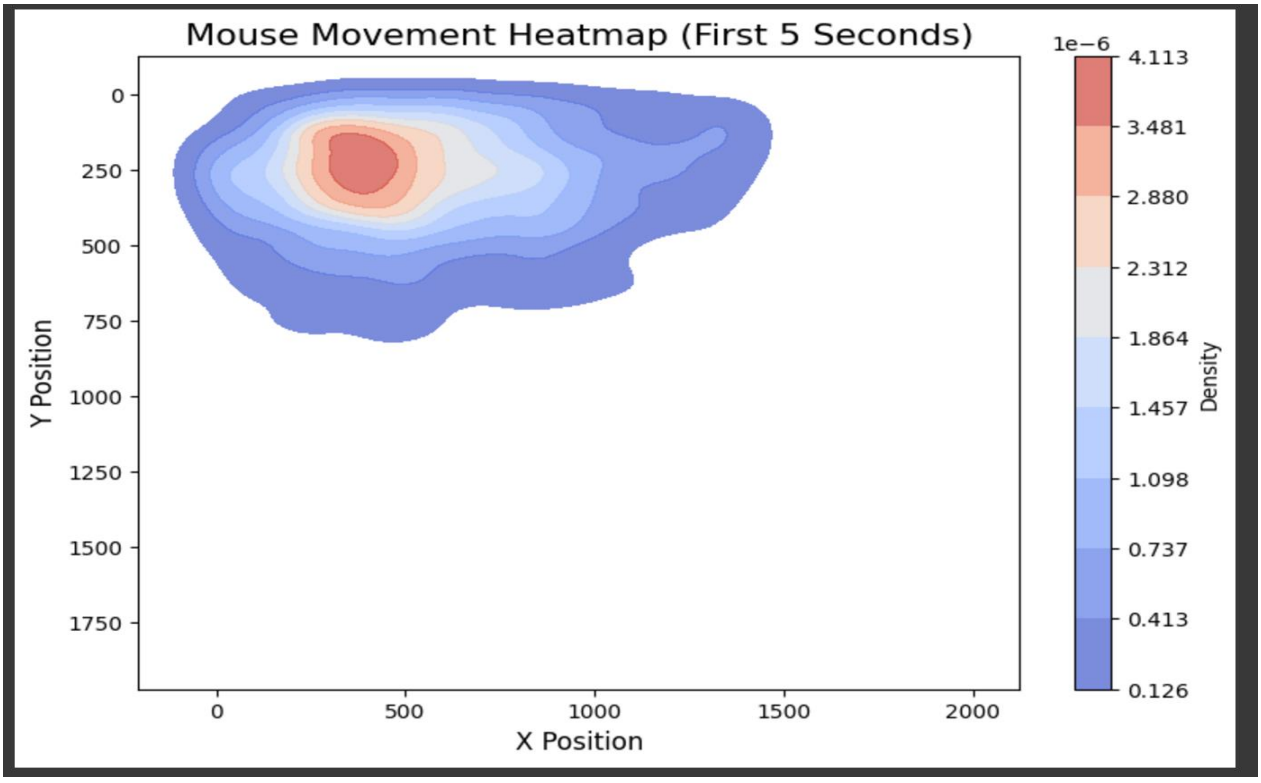
	0
cursor	object
timestamp	int64
xpos	object
ypos	object
event	object
xpath	object
attrs	object
extras	object
log_id	int64

dtype: object

To better understand user activity during these 5 seconds, I decided to visualize some features that may provide valuable insights and assist in the next steps of the analysis.

```
# Convert xpos and ypos columns to numeric types in filtered cursor data
filtered_cursor_data_df['xpos'] = pd.to_numeric(filtered_cursor_data_df['xpos'], errors='coerce')
filtered_cursor_data_df['ypos'] = pd.to_numeric(filtered_cursor_data_df['ypos'], errors='coerce')
```

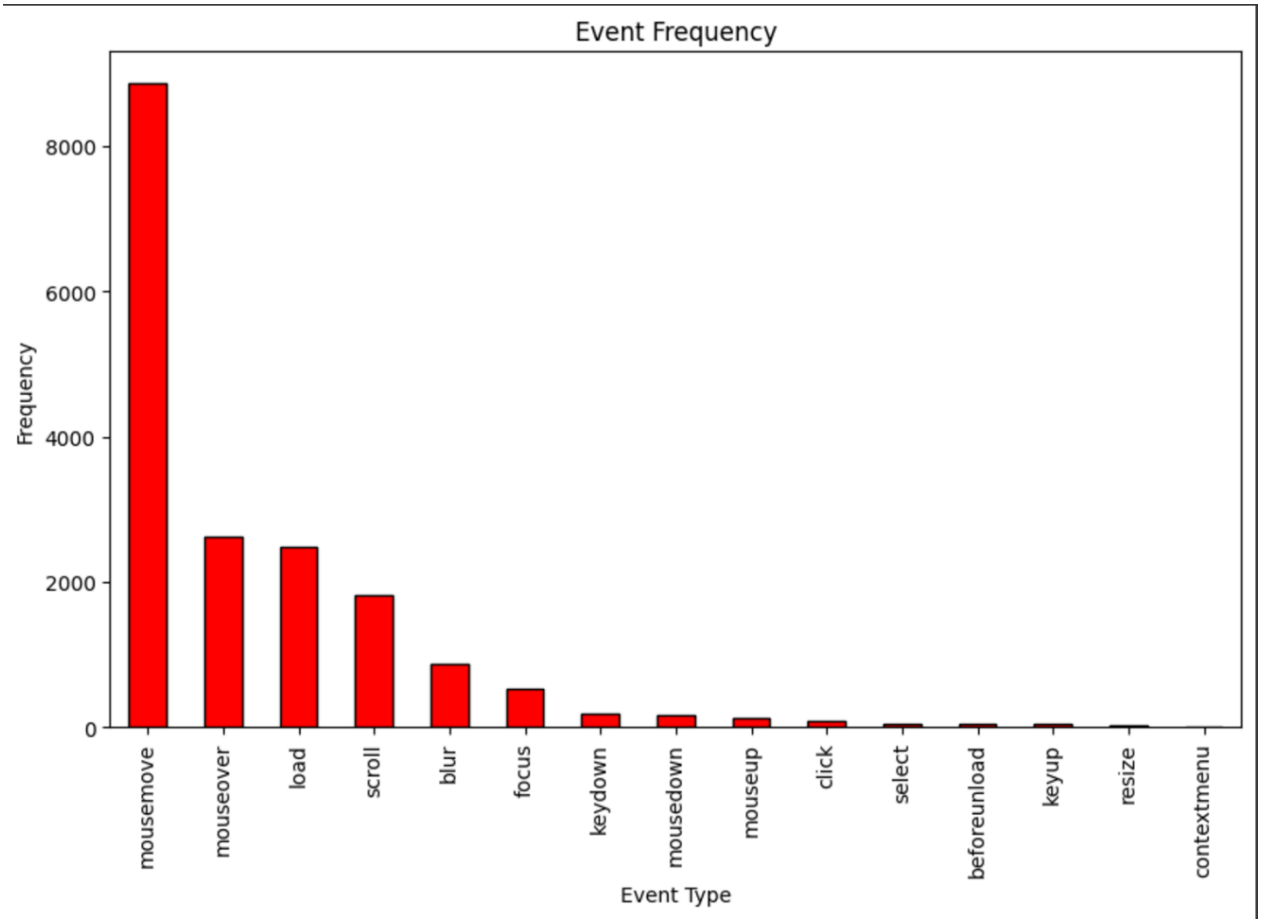
```
mousemove_data = filtered_cursor_data_df[filtered_cursor_data_df['event'] == 'mousemove']

plt.figure(figsize=(9, 6))
heatmap = sns.kdeplot(
    x=mousemove_data['xpos'],
    y=mousemove_data['ypos'],
    fill=True,
    cmap="coolwarm",
    alpha=0.7
)
plt.title('Mouse Movement Heatmap (First 5 Seconds)', fontsize=16)
plt.xlabel('X Position', fontsize=12)
plt.ylabel('Y Position', fontsize=12)
plt.gca().invert_yaxis()  # Invert Y-axis for web screen coordinate system
cbar = heatmap.figure.colorbar(heatmap.collections[0], ax=heatmap.axes, label="Density")
plt.show()
```



(Anon., n.d.)

This heatmap visualizes mouse movement density during the first 5 seconds of user interaction on a webpage, highlighting areas of concentrated activity with warmer colors (red) and less frequent movements with cooler colors (blue). The red region in the center suggests that users focused their attention on this area, possibly due to the presence of visually engaging elements such as ads, images, or buttons. The inverted Y-axis aligns with the webpage's screen coordinate system, ensuring the heatmap accurately reflects the layout. This visualization helps identify effective areas of interaction, indicating where content placement is working well, while also revealing underutilized sections of the page that could be optimized to improve user engagement. We can see the top of the webpage can be more important compare to the bottom.

```
# Visualization 4: Event-specific Features
plt.figure(figsize=(10, 6))
event_counts = filtered_cursor_data_df['event'].value_counts()
event_counts.plot(kind='bar', color='red', edgecolor='black')
plt.xlabel('Event Type')
plt.ylabel('Frequency')
plt.title('Event Frequency')
plt.show()
```



We can observe that the frequency of mouse movements is significantly higher compared to other events. Additionally, the frequency of clicks is very low. These clicks indicate whether users clicked on any part of the webpage or not. Based on this observation, we can infer that we are likely dealing with an imbalanced dataset.

Considering the features of filtered_cursor_data_df, we can enhance the dataset by adding new features that could be useful for modeling. Features like xpath, attrs, and cursor do not provide meaningful information, as attrs and cursor have the same value for all rows. The most important features in this dataset are xpos, ypos, and event. Using these features, we can create additional derived features to capture more insights and improve the predictive model.

```
# Define the function to extract cursor movement metrics for the first 5 seconds in a log file
def extract_cursor_features(log_df):
    # Calculate the starting timestamp and filter rows within the first 5 seconds
    start_time = log_df["timestamp"].min()
    filtered_log_df = log_df[log_df["timestamp"] <= start_time + 5000]

     # Calculate total distance traveled
    filtered_log_df = filtered_log_df.sort_values(by="timestamp")
    filtered_log_df["xpos_shifted"] = filtered_log_df["xpos"].shift()
    filtered_log_df["ypos_shifted"] = filtered_log_df["ypos"].shift()

    # Calculate the Euclidean distance between consecutive points
    filtered_log_df["distance"] = np.sqrt(
        (filtered_log_df["xpos"] - filtered_log_df["xpos_shifted"]) ** 2 +
        (filtered_log_df["ypos"] - filtered_log_df["ypos_shifted"]) ** 2
    )
    total_distance_traveled = filtered_log_df["distance"].sum(skipna=True)

    # Add names of event features to the dataset
    event_counts = filtered_log_df['event'].value_counts().to_dict()

    # add total unique event
    total_unique_event = len(event_counts)

    # Return the features as a dictionary
    return {

        'total_unique_event': total_unique_event,
        'total_distance': total_distance_traveled,
    }

sample_features = extract_cursor_features(filtered_cursor_data_df)
```

```python
# Filter the log files to use only those in log_duration_df (logs with at least 5 seconds of activity)
filtered_log_ids = log_duration_df['log_id'].astype(str).tolist()
filtered_log_files = [f"{log_id}.csv" for log_id in filtered_log_ids if f"{log_id}.csv" in log_files]
logs_path = os.path.join(extract_dir, 'logs')

# Initialize list to hold the features from each filtered log
cursor_features = []

# Loop through each filtered log file to extract cursor movement features
for log_filename in filtered_log_files:
    log_id = log_filename.split(".")[0]  # Extract log_id from the filename
    log_df = pd.read_csv(os.path.join(logs_path, log_filename), delim_whitespace=True, header=None,
                         names=["cursor", "timestamp", "xpos", "ypos", "event", "xpath", "attrs", "extras"])

    # Convert xpos, ypos, and timestamp to numeric in case they are not already
    log_df['xpos'] = pd.to_numeric(log_df['xpos'], errors='coerce')
    log_df['ypos'] = pd.to_numeric(log_df['ypos'], errors='coerce')
    log_df['timestamp'] = pd.to_numeric(log_df['timestamp'], errors='coerce')

    # Extract features using the function defined above
    features = extract_cursor_features(log_df)
    features['log_id'] = log_id  # Add log_id for merging with participant and groundtruth data

    # Append the extracted features to the list
    cursor_features.append(features)

# Convert the list of features into a DataFrame
cursor_features_df = pd.DataFrame(cursor_features)
```

(Anon., n.d.)

This code processes cursor movement logs files to extract meaningful features for analysis or modeling. It first filters the logs files to include only those corresponding to log_ids with at least 5 seconds of activity, as specified in log_duration_df. These filtered files are read into pandas DataFrames with predefined columns, such as xpos, ypos, timestamp, and event. Non-numeric columns like xpos, ypos, and timestamp are converted to numeric types, handling any invalid values as NaN. Using a custom function, extract_cursor_features, cursor movement features are computed from each log file, and the log_id is added for identification. The extracted features are stored in a list, which is later converted into a unified DataFrame (cursor_features_df). This dataset provides structured, relevant features derived from cursor movements, enabling further analysis, machine learning, or user behavior insights.

```python
cursor_features_df.head()
```

	movement_distance	total_unique_event	log_id
0	0.000000	2	20181002070022
1	0.000000	8	20170201233856
2	1123.929268	6	20170211043537
3	0.000000	1	20170214183039
4	0.000000	4	20181002105243

```
cursor_features_df.shape
```

```
(2501, 3)
```

We can see the shape of cursor_features_df is the same with previous steps and it means that we have information of first five seconds of users that spent al least five seconds in webpage.

To create the final dataset, we aim to combine groundtruth_df, participants_df, and cursor_features_df so that each row represents comprehensive information about each user. To achieve this, we can merge these datasets together based on the log_id, which is a common key in all three datasets. This merging process ensures that user activity data from cursor_features_df is aligned with user details from participants_df and the corresponding ground truth data from groundtruth_df. The resulting dataset will be a unified table containing all relevant features, making it ready for further analysis or modeling.

```
# Ensure log_id columns are of type string for merging
cursor_features_df['log_id'] = cursor_features_df['log_id'].astype(str)
groundtruth_df['log_id'] = groundtruth_df['log_id'].astype(str)

# Retry merging after type conversion
merged_data = cursor_features_df.merge(groundtruth_df, on='log_id', how='inner')
final_data = merged_data.merge(participants_df, on='user_id', how='inner')

# Display the first few rows of the final combined dataset
final_data.head()
```

	movement_distance	total_unique_event	log_id_x	user_id	ad_clicked	attention	country	education	age	income	gender	ad_position
0	0.000000	2	20181002070022	bnh7toi7cl5vjgurds827as4u0	0	5	TWN	3	3	6	male	top-left
1	0.000000	8	20170201233856	73gjnkrfthrcu64mhk2upi3as7	0	1	USA	1	na	4	female	top-left
2	1123.929268	6	20170211043537	svfikhps7bcun80uoqplejel97	0	2	USA	1	4	2	female	top-left
3	0.000000	1	20170214183039	8iuc69tqes72lvqc2cck7jusm6	0	5	USA	5	2	4	female	top-right
4	0.000000	4	20181002105243	bl4td3oo6qav1uc9efi1m6d0p2	0	1	VEN	1	6	na	male	top-right

ad_type	ad_category	serp_id	query	log_id_y
dd	Computers & Electronics	samsung-tablet	samsung tablet	20181002070022
dd	Shop - Apparel	nike-windbreaker_2	nike windbreaker	20170201233856
dd	Shop - Apparel	mvmt-watches_2	mvmt watches	20170211043537
dd	Shop - Luxury Goods	black-label	black label	20170214183039
dd	Shop - Apparel	nike-shoes	nike shoes	20181002105243

By examining the log_id, we can infer that it contains information about the year, month, day, hour, minute, and second of each user's activity on the webpage. To enrich our dataset, we

extracted these components from the log_id and added them as separate features to the dataset. This allows us to analyze user activity patterns more effectively, such as identifying peak usage times or understanding behavior trends based on the time and date of their interactions.

```
# add year, month and day, houre,minutes and seconde features base on log_id_x
final_data['year'] = final_data['log_id_x'].str[:4]
final_data['month'] = final_data['log_id_x'].str[4:6]
final_data['day'] = final_data['log_id_x'].str[6:8]
final_data['hour'] = final_data['log_id_x'].str[8:10]
```

```
# convert to dytpyes
final_data['year'] = final_data['year'].astype(int)
final_data['month'] = final_data['month'].astype(int)
final_data['day'] = final_data['day'].astype(int)
final_data['hour'] = final_data['hour'].astype(int)
```

```
final_data.head()
```

	movement_distance	total_unique_event	log_id_x	user_id	ad_clicked	attention	country	education	age	income	gender	ad_position	ad_type
0	0.000000	2	20181002070022	bnh7toi7cl5vjgurds827as4u0	0	5	TWN	3	3	6	male	top-left	dd
1	0.000000	8	20170201233856	73gjnkrfthrcu64mhk2upi3as7	0	1	USA	1	na	4	female	top-left	dd
2	1123.929268	6	20170211043537	svfikhps7bcun80uoqplejel97	0	2	USA	1	4	2	female	top-left	dd
3	0.000000	1	20170214183039	8iuc69tqes72lvqc2cck7jusm6	0	5	USA	5	2	4	female	top-right	dd
4	0.000000	4	20181002105243	bl4td3oo6qav1uc9efi1m6d0p2	0	1	VEN	1	6	na	male	top-right	dd

ad_category	serp_id	query	log_id_y	year	month	day	hour
Computers & Electronics	samsung-tablet	samsung tablet	20181002070022	2018	10	2	7
Shop - Apparel	nike-windbreaker_2	nike windbreaker	20170201233856	2017	2	1	23
Shop - Apparel	mvmt-watches_2	mvmt watches	20170211043537	2017	2	11	4
Shop - Luxury Goods	black-label	black label	20170214183039	2017	2	14	18
Shop - Apparel	nike-shoes	nike shoes	20181002105243	2018	10	2	10

Dataset Overview:

Dataset Basic Information

```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2501 entries, 0 to 2500
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   movement_distance   2501 non-null   float64
 1   total_unique_event  2501 non-null   int64
 2   log_id_x            2501 non-null   object
 3   user_id             2501 non-null   object
 4   ad_clicked          2501 non-null   int64
 5   attention           2501 non-null   int64
 6   country             2501 non-null   object
 7   education           2501 non-null   object
 8   age                 2501 non-null   object
 9   income              2501 non-null   object
 10  gender              2501 non-null   object
 11  ad_position         2501 non-null   object
 12  ad_type             2501 non-null   object
 13  ad_category         2501 non-null   object
 14  serp_id             2501 non-null   object
 15  query               2501 non-null   object
 16  log_id_y            2501 non-null   int64
 17  year                2501 non-null   int64
 18  month               2501 non-null   int64
 19  day                 2501 non-null   int64
 20  hour                2501 non-null   int64
dtypes: float64(1), int64(8), object(12)
memory usage: 410.4+ KB
```

It appears that some columns, such as education, age, income, and attention, need to have their data types converted to integers. Before making this change, we should first examine their unique values to ensure that the conversion is appropriate and won't result in errors or data loss.

```
# value of age and income
for column in ['education', 'age', 'income', 'attention']:
    print(f"Unique values in {column}: {final_data[column].unique()}")
```

```
Unique values in education: ['4' 'na' '2' '1' '3' '5' '6']
Unique values in age: ['3' '4' '2' '5' '6' '1' '7' 'na' '8' '9']
Unique values in income: ['3' '1' '2' '4' 'na' '5' '6' '8' '7']
Unique values in attention: [4 5 2 1 3]
```

```
# Convert relevant columns to numeric data types (education, age, income, attention)
numeric_columns = ['education', 'age', 'income', 'attention']
final_data[numeric_columns] = final_data[numeric_columns].apply(pd.to_numeric, errors='coerce')
```

For the final dataset intended for modeling, we will exclude features such as education, age, income, attention, and gender. While these features are not necessary for the modeling process,

they can be valuable for exploratory data analysis (EDA) to understand patterns and relationships within the data. Once the EDA is complete, we will drop these features before proceeding with modeling to focus only on the most relevant variables.

```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2501 entries, 0 to 2500
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   movement_distance   2501 non-null   float64
 1   total_unique_event  2501 non-null   int64  
 2   log_id_x            2501 non-null   object 
 3   user_id             2501 non-null   object 
 4   ad_clicked          2501 non-null   int64  
 5   attention           2501 non-null   int64  
 6   country             2501 non-null   object 
 7   education           2460 non-null   float64
 8   age                 2483 non-null   float64
 9   income              2344 non-null   float64
 10  gender              2501 non-null   object 
 11  ad_position         2501 non-null   object 
 12  ad_type             2501 non-null   object 
 13  ad_category         2501 non-null   object 
 14  serp_id             2501 non-null   object 
 15  query               2501 non-null   object 
 16  log_id_y            2501 non-null   int64  
 17  year                2501 non-null   int64  
 18  month               2501 non-null   int64  
 19  day                 2501 non-null   int64  
 20  hour                2501 non-null   int64  
dtypes: float64(4), int64(8), object(9)
memory usage: 410.4+ KB
```

Inferences:

- **Number of Entries**: The dataset consists of 2501 entries, ranging from index 0 to 2500.
- **Columns:** There are 21 columns
- **Data Types:**
    - Number of object columns: 9
    - Number of numeric columns: 12
- **Missing Values**: We observed missing values in the columns education, age, and income.

Summary Statistics for Numerical Variables:

```
final_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
total_unique_event	2501.0	2.980008e+00	1.578480e+00	1.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00	1.000000e+01
total_distance	2501.0	1.241449e+03	1.525951e+03	0.000000e+00	0.000000e+00	9.165099e+02	1.759715e+03	1.363388e+04
ad_clicked	2501.0	2.335066e-01	4.231464e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
attention	2501.0	3.598960e+00	1.320418e+00	1.000000e+00	2.000000e+00	4.000000e+00	5.000000e+00	5.000000e+00
education	2460.0	2.850000e+00	1.388627e+00	1.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00	6.000000e+00
age	2483.0	3.426903e+00	1.896547e+00	1.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00	9.000000e+00
income	2344.0	2.664249e+00	1.750893e+00	1.000000e+00	1.000000e+00	2.000000e+00	4.000000e+00	8.000000e+00
log_id_y	2501.0	2.017498e+13	6.569162e+09	2.016121e+13	2.017021e+13	2.018012e+13	2.018100e+13	2.018100e+13
year	2501.0	2.017435e+03	6.483275e-01	2.016000e+03	2.017000e+03	2.018000e+03	2.018000e+03	2.018000e+03
month	2501.0	6.161136e+00	3.983871e+00	1.000000e+00	2.000000e+00	7.000000e+00	1.000000e+01	1.200000e+01
day	2501.0	1.174610e+01	1.028186e+01	1.000000e+00	1.000000e+00	1.000000e+01	2.100000e+01	3.100000e+01
hour	2501.0	1.307397e+01	6.778416e+00	0.000000e+00	8.000000e+00	1.400000e+01	1.800000e+01	2.300000e+01

Inferences:

Missing Data:

- The education, age, and income columns have counts of 2460, 2483, and 2344, respectively, indicating missing data in these columns.

Total Unique Events:

- The relatively small range between the maximum and the 75th percentile indicates fewer extreme outliers.
- The mean (2.98) and median (3) are close, suggesting minimal skewness in the distribution.

Total Distance:

- The large gap between the maximum value and the 75th percentile suggests significant outliers in the data.
- The mean (1,241.45) is much lower than the 75th percentile, indicating a right-skewed distribution, where most values are small, with a few large outliers.

Attention:

- Most values are concentrated at higher levels of attention, with minimal outliers due to the constrained range (1 to 5).

- The slight difference between the mean (3.6) and the median (4) suggests a possible slight negative skew.

Education:

- The distribution appears relatively uniform within the range, but missing data needs to be addressed.

Age:

- The distribution of ages seems right-skewed with higher concentrations at lower values.

Income:

- The lower median and mean (2.66) suggest the majority of individuals fall into the lower income brackets.

Other Temporal Variables:

- Columns such as hour, month, and day have values distributed within expected ranges, with no evident outliers or skewness based on the summary.

```
final_data['log_id_y'].nunique()
```

```
2501
```

we can drop log_id_y from dataset, we divided it to year, month, hour, minutes and it used for merging and contain unique value for users.

```
final_data.drop('log_id_y', axis=1, inplace=True)
```

Summary Statistics for Categorical Variables:

```
final_data.describe(include='object').T
```

	count	unique	top	freq
log_id_x	2501	2501	20181002070022	1
user_id	2501	2501	bnh7toi7cl5vjgurds827as4u0	1
country	2501	68	USA	1423
gender	2501	3	male	1458
ad_position	2501	2	top-left	1687
ad_type	2501	2	dd	1670
ad_category	2501	14	Computers & Electronics	693
serp_id	2501	63	samsung-tablet	89
query	2501	55	laptop bag	123

- log_id_x:
  There are 2501 unique log_id_x values, indicating that this column uniquely identifies each record in the dataset. Every log_id_x is distinct, making it unsuitable for aggregation or group-level analysis. Including the most frequent value (20181002070022) is not meaningful because all values are unique.
- user_id:
  Similar to log_id_x, there are 2501 unique user_ids, meaning each row corresponds to a unique user session. This emphasizes that the dataset is user-specific. Mentioning a most frequent user_id is irrelevant since every user ID appears only once.
- country:
  There are 68 unique countries, with "USA" being the most frequent (1423 occurrences, ~57%). This indicates that a significant portion of the user base is concentrated in the USA, which might influence ad strategies or the need for localization for other countries
- gender:
  There are 3 unique genders, with "male" being the most frequent (1458 occurrences, ~58%). This male-dominant user demographic could suggest targeting specific products or services tailored toward men.
- ad_position:
  There are 2 unique ad positions, with "top-left" being the most frequent (1687 occurrences, ~67%). This suggests that "top-left" is the preferred or most commonly allocated ad position, likely due to higher visibility and click-through potential.
- ad_type:

Two ad types are recorded, with "dd" being the most frequent (1670 occurrences, ~67%). This likely represents a default or standard ad format, potentially dominating the dataset.

- ad_category:
  There are 14 unique ad categories, with "Computers & Electronics" being the most frequent (693 occurrences, ~28%). This indicates a focus on tech-related advertisements, which could align with user interests or the website's core audience.
- serp_id:
  There are 63 unique serp_ids (search result page identifiers), with "samsung-tablet" being the most frequent (89 occurrences). This suggests notable user interest in Samsung tablets, which might guide recommendations or ad placement decisions.
- query:
  There are 55 unique queries, with "laptop bag" being the most frequent (123 occurrences, ~5%). This highlights user interest in specific tech accessories, suggesting an opportunity to refine product recommendations or optimize ad content for such queries.

we can see features serp_id and quary have many same values and both are subsets of ad_category. Base on these information we can make a decision to remove some features like 'log_id_x', 'user_id','serp_id','query' but we will do in next step to see the unique values.

```python
def grab_col_names(df):
    # Separate columns into categorical and numerical
    categorical_cols = df.select_dtypes(include=['object', 'category']).columns
    numerical_cols = df.select_dtypes(include=['number']).columns

    print(f'cat_col: {len(categorical_cols)}')
    print(f'num_col: {len(numerical_cols)}')

    return categorical_cols, numerical_cols
```

```python
def identify_check(dataframe):
    print('-' * 80)
    print(f'Categorical : {categorical_cols}')
    print('-' * 80)
    print(f'Numerical : {numerical_cols}')


categorical_cols, numerical_cols = grab_col_names(final_data)
identify_check(final_data)
```

```
cat_col: 9
num_col: 11
--------------------------------------------------------------------------------
Categorical : Index(['log_id_x', 'user_id', 'country', 'gender', 'ad_position', 'ad_type',
       'ad_category', 'serp_id', 'query'],
      dtype='object')
--------------------------------------------------------------------------------
Numerical : Index(['movement_distance', 'total_unique_event', 'ad_clicked', 'attention',
       'education', 'age', 'income', 'year', 'month', 'day', 'hour'],
      dtype='object')
```

```python
# find unqie value and number of uniqe values in categorical culomns
for col in categorical_cols:
  print(col,':',final_data[col].nunique())
  print(col,':',final_data[col].unique())
  print('-'*50)
```

```
log_id_x : 2501
log_id_x : ['20181002070022' '20170201233856' '20170211043537' ... '20170303012811'
 '20161231015823' '20170207022424']
--------------------------------------------------
user_id : 2501
user_id : ['bnh7toi7cl5vjgurds827as4u0' '73gjnkrfthrcu64mhk2upi3as7'
 'svfikhps7bcun80uoqplejel97' ... 'vovjem42oh2velc5vms8psjse6'
 'h26fts4n0rcd1t7ru56o3bn172' 'abdokccrssh5e19qa1l7tsmit0']
--------------------------------------------------
country : 68
country : ['TWN' 'USA' 'VEN' 'BGD' 'NLD' 'SAU' 'GBR' 'GRC' 'SRB' 'SWE' 'DEU' 'UKR'
 'BRA' 'ROU' 'EGY' 'ESP' 'ITA' 'RUS' 'CAN' 'IDN' 'MYS' 'MEX' 'IND' 'TUR'
 'POL' 'na' 'VNM' 'PHL' 'DZA' 'COL' 'SVN' 'LTU' 'MKD' 'BOL' 'LKA' 'PER'
 'FRA' 'AUT' 'MDA' 'AUS' 'MAR' 'BIH' 'BEL' 'ARG' 'DOM' 'NGA' 'NPL' 'CHL'
 'HRV' 'BGR' 'QAT' 'PRT' 'ISR' 'SGP' 'PAK' 'ZAF' 'TUN' 'DNK' 'KEN' 'IRL'
 'HKG' 'CHE' 'KOR' 'ECU' 'KWT' 'CYP' 'CZE' 'HUN']
--------------------------------------------------
gender : 3
gender : ['male' 'female' 'na']
--------------------------------------------------
ad_position : 2
ad_position : ['top-left' 'top-right']
--------------------------------------------------
ad_type : 2
ad_type : ['dd' 'native']
--------------------------------------------------
ad_category : 14
ad_category : ['Computers & Electronics' 'Shop - Apparel' 'Shop - Luxury Goods'
 'Shop - Sporting Goods' 'Shop - Event Ticket Sales'
 'Shop - Photo & Video Services' 'Travel' 'Shop - Gifts & Special Event'
--------------------------------------------------
serp_id : 63
serp_id : ['samsung-tablet' 'nike-windbreaker_2' 'mvmt-watches_2' 'black-label'
 'nike-shoes' 'buy-dji-mavic-pro_2' 'adidas-ultra-boost' 'nfl-ticket'
 'tablets' 'laptop-bag_2' 'alta-fitbit' 'dell-tablet' 'iphone-7-plus'
 'ipad-air_2' 'macbook' 'buy-dji-mavic-pro' 'flowers_2' 'nike-windbreaker'
 'ipad-pro' 'louis-vuitton' 'casio-watches' 'lenovo-tablet'
 'shop-adidas-nmd_2' 'disney-lego-minifigures' 'jewelry'
 'garmin-vivosmart' 'cubs-tickets' 'shop-victorias-secret_2'
 'breitling-watches' 'mvmt-watches' 'galaxy-s7' 'rolex' 'nike-shoes_2'
 'chivas-regal' 'stubhub' 'flights' 'nerf-rival-shop' 'iphone-6se'
 'xbox-one' 'buy-paragon-game' 'ford-focus' 'delta-airlines'
 'norwegian-air' 'cartier-watch' 'fitbit' 'shop-adidas-nmd' 'ps4'
 'buy-mountain-bike' 'famous-grouse' 'nerf-rival-shop_2' 'jack-daniels'
 'mercedes-e-class-2016' 'hotels' 'american-airlines' 'hp-tablet'
 'laptop-bag' 'pizza' 'adele-tickets' 'apartments-for-rent'
 'shop-victorias-secret' 'british-airways' 'airbnb' 'audi-r8-used']
--------------------------------------------------
query : 55
query : ['samsung tablet' 'nike windbreaker' 'mvmt watches' 'black label'
 'nike shoes' 'buy dji mavic pro' 'adidas ultra boost' 'nfl ticket'
 'tablets' 'laptop bag' 'alta fitbit' 'dell tablet' 'iphone 7 plus'
 'iPad Air 2' 'macbook' 'flowers' 'ipad pro' 'louis vuitton'
 'casio watches' 'lenovo tablet' 'shop adidas nmd'
 'disney lego minifigures' 'jewelry' 'garmin vivosmart' 'cubs tickets'
 "shop victoria's secret" 'breitling watches' 'galaxy s7' 'rolex'
 'chivas regal' 'stubhub' 'flights' 'nerf rival shop' 'iphone 6se'
 'xbox one' 'buy paragon game' 'ford focus' 'delta airlines'
 'norwegian air' 'cartier watch' 'fitbit' 'ps4' 'buy mountain bike'
 'famous grouse' 'jack daniels' 'mercedes e class 2016' 'hotels'
 'american airlines' 'hp tablet' 'pizza' 'adele tickets'
 'apartments for rent' 'british airways' 'airbnb' 'audi r8 used']
--------------------------------------------------
```

```
final_data['gender'].value_counts()
```

gender	count
male	1458
female	1030
na	13

dtype: int64

As the number of ‘na’ is too low for the handeling 'na' we can replace na Proportionally Based on Existing Distribution. Randomly assign "male" or "female" to na values based on the existing gender distribution. The reason is that it maintains the original distribution, reducing the risk of introducing bias.

```
proportions = final_data['gender'].value_counts(normalize=True)
# Ensure probabilities sum to 1 by normalizing again
probabilities = [proportions['male'], proportions['female']]
probabilities = [p / sum(probabilities) for p in probabilities]  # Normalize

final_data.loc[final_data['gender'] == 'na', 'gender'] = np.random.choice(
    ['male', 'female'],
    size=final_data['gender'].value_counts()['na'],
    p=probabilities # Use normalized probabilities
)
```

```
final_data['gender'].value_counts()
```

gender	count
male	1465
female	1036

After gender we are going to check country

```
# percantage of values in country columns
final_data['country'].value_counts(normalize=True)*100
```

country	proportion
USA	56.897241
VEN	13.914434
GBR	7.916833
CAN	2.918832
EGY	1.519392
...	...
CZE	0.039984
KEN	0.039984
CYP	0.039984
SVN	0.039984
ZAF	0.039984

68 rows × 1 columns

dtype: float64

more than 50% of values in country feature is USA than we can convert values to USA and non-USA.

```
final_data['country'] = final_data['country'].apply(lambda x: 'non-USA' if x != 'USA' else 'USA')
```

```
final_data['country'].value_counts()
```

country	count
USA	1423
non-USA	1078

dtype: int64

Now we can dorp 'log_id_x', 'user_id','serp_id','query' from dataset and one more time check our dataset.

```
# drop log_id_x and user_id
final_data.drop(['log_id_x', 'user_id','serp_id','query'], axis=1, inplace=True)
```

```
final_data.head()
```

	movement_distance	total_unique_event	ad_clicked	attention	country	education	age	income	gender	ad_position	ad_type	ad_category	year	month	day	hour
0	0.000000	2	0	5	non-USA	3.0	3.0	6.0	male	top-left	dd	Computers & Electronics	2018	10	2	7
1	0.000000	8	0	1	USA	1.0	NaN	4.0	female	top-left	dd	Shop - Apparel	2017	2	1	23
2	1123.929268	6	0	2	USA	1.0	4.0	2.0	female	top-left	dd	Shop - Apparel	2017	2	11	4
3	0.000000	1	0	5	USA	5.0	2.0	4.0	female	top-right	dd	Shop - Luxury Goods	2017	2	14	18
4	0.000000	4	0	1	non-USA	1.0	6.0	NaN	male	top-right	dd	Shop - Apparel	2018	10	2	10

Data Cleaning:

Data cleaning is the process of preparing and organizing raw data to make it suitable for further analysis. This involves identifying and correcting errors or incomplete entries, eliminating irrelevant information, filling in missing values, and transforming variables as needed. It is a crucial step in any machine learning project, as it enhances model accuracy by minimizing noise and inconsistencies in the data. Neglecting proper data cleaning can result in overly complex models that are hard to interpret and prone to overfitting. Additionally, even a small amount of noisy or unclean data can significantly impact a model's performance, leading to reduced accuracy and reliability.

```
# Percentage of missing values sorted in descending order.
final_data.isnull().mean().sort_values(ascending=False)*100
```

	0
income	6.277489
education	1.639344
age	0.719712
movement_distance	0.000000
total_unique_event	0.000000
ad_clicked	0.000000
attention	0.000000
country	0.000000
gender	0.000000
ad_position	0.000000
ad_type	0.000000
ad_category	0.000000
year	0.000000
month	0.000000
day	0.000000
hour	0.000000

Only income, gender, and age have missing values, but the percentage is low. To handle the missing values, it is better to first examine the mean and median for these features.

```
# describe for age, income and education
final_data[['age', 'income', 'education']].describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	2483.0	3.426903	1.896547	1.0	2.0	3.0	4.0	9.0
income	2344.0	2.664249	1.750893	1.0	1.0	2.0	4.0	8.0
education	2460.0	2.850000	1.388627	1.0	2.0	3.0	4.0	6.0

since we plan to drop these features from the dataset after performing EDA, we can fill the missing values with the median for now.

```
# fillna null values in education, age and income with medain
final_data['education'].fillna(final_data['education'].median(), inplace=True)
final_data['age'].fillna(final_data['age'].median(), inplace=True)
final_data['income'].fillna(final_data['income'].median(), inplace=True)
```

EDA:

Our Exploratory Data Analysis (EDA) includes two sections:

1. **Univariate Analysis**: We will examine each feature individually to understand its distribution and range.
2. **Bivariate Analysis**: In this step, we will analyze the relationship between each feature and the target variable to understand the significance and impact of each feature on the target outcome.

With these two steps, we aim to gain insights into the individual characteristics of the data and also how each feature relates to our main goal**: predicting the target variable.**

Univariate Analysis:

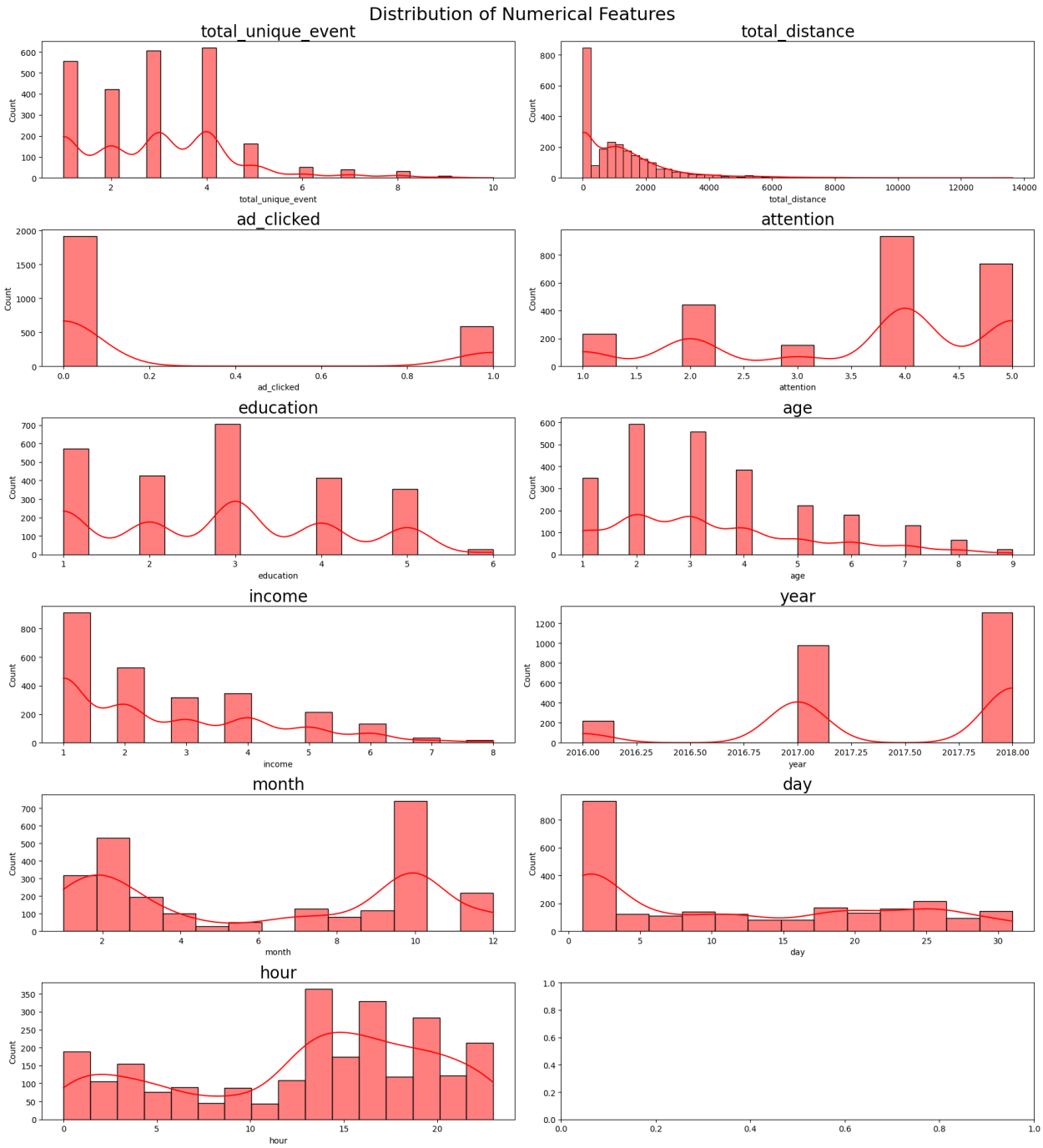
We undertake univariate analysis on the dataset's features, based on their datatype:

- For **numerical data**: We employ histograms to gain insight into the distribution of each feature. This allows us to understand the central tendency, spread, and shape of the dataset's distribution

- For **categorical data**: Bar plots are utilized to visualize the frequency of each category. This provides a clear representation of the prominence of each category within the respective feature. By employing these visualization techniques, we're better positioned to understand the individual characteristics of each feature in the dataset.

Numerical Variables Univariate Analysis:

```
# Set up the subplot for a 3x2 layout
numerical_features = [col for col in final_data.columns if final_data[col].dtype != 'O']

#Calculate the number of rows and columns needed for the subplots
num_rows = int(np.ceil(len(numerical_features) / 2))  # Calculate rows dynamically
num_cols = 2

fig, ax = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(18, 20))  # Adjust grid

# Loop to plot histograms for each numerical feature
for i, feature in enumerate(numerical_features):
    row = i // num_cols  # Update row calculation
    col_idx = i % num_cols

    # Plot histogram with KDE
    sns.histplot(final_data[feature], kde=True, ax=ax[row, col_idx], color='red')

    ax[row, col_idx].set_title(f'{feature}', fontsize=20)

# Hide any unused subplots (if any)
# No need for this part as we calculated rows and cols dynamically

plt.suptitle('Distribution of Numerical Features', fontsize=22)
plt.tight_layout()
plt.subplots_adjust(top=0.95)
plt.show()
```



Inferences:

- Education:
  - The distribution of education levels shows relatively balanced counts across levels 1 to 4, with level 3 (Bachelor's) having the highest count.
  - Very few participants have education levels 5 (Master's) and 6 (Doctorate), indicating fewer highly educated participants.

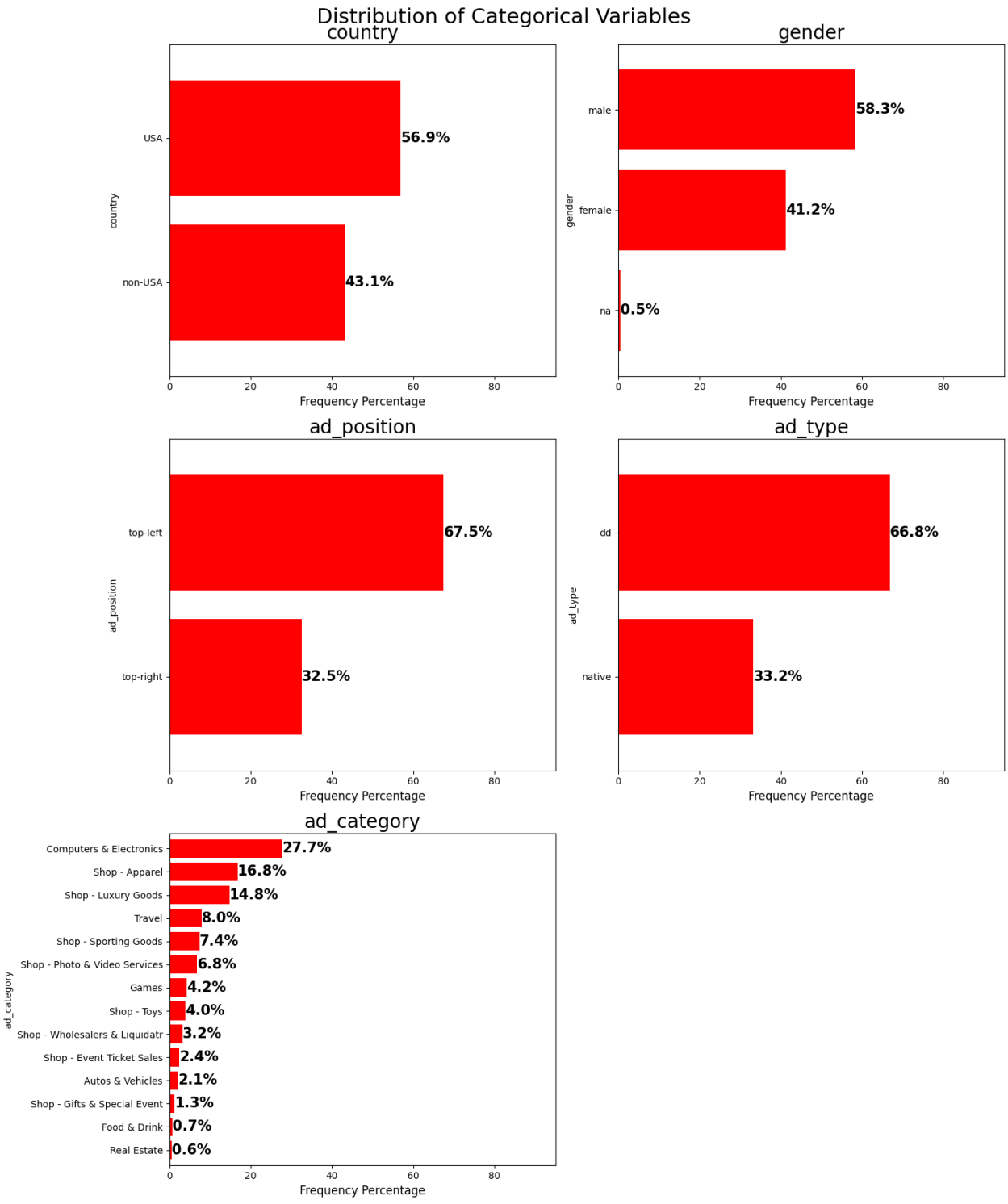
- Age:
  - The distribution of age seems right-skewed, with most people clustered around the lower age levels (1–4). Higher age levels are much less frequent.
- Income:
  - The distribution of income is skewed towards lower income levels, with bin 1 (25K or lower) having the highest count.
  - There is a steep drop-off after income bin 4 (50-74K), indicating that most participants have lower to middle-range income levels.
- Ad Clicked:
  - The majority of participants did not click on the ad (denoted by ad_clicked = 0), with very few clicks observed (ad_clicked = 1).
  - This suggests that ad click-through rates are quite low in this dataset.
- Attention:
  - Most participants self-reported higher levels of attention, with the majority giving scores of 4 and 5.
  - There are very few participants with attention scores of 1 or 3, indicating that participants generally felt engaged or attentive during the task.
- Total distance:
  - This is also heavily right-skewed, with most users registering very low movement distances.
  - The majority of the Total distance are below 2000, but the long tail stretches to over 12000, indicating outliers.
  - While most users move minimally, a few users interact across large distances on the page. These outliers might reflect a different user behavior (e.g., scrolling extensively or navigating complex interfaces).
- Total Unique Events
  - This distribution appears slightly less skewed but still leans toward low values.
  - Most users interact with fewer than 5 unique events, and the counts drop off steadily for values beyond 6 or 7.
  - Users tend to repeat the same types of events rather than interacting with a wide variety. Understanding what these events are could help optimize the user interface or content.
- Month
  - User activity is unevenly distributed across months.
  - There are peaks around February (Month 2) and October (Month 10), while activity is significantly lower during other months, especially in the middle of the year (around months 6–8).

    - The spikes in February and October might correspond to specific campaigns, seasonal trends, or major events driving user activity.
    - The low activity during mid-year could reflect a seasonal dip, which may warrant investigation to uncover why.
- Day
    - The distribution is skewed toward the start of the month, with a noticeable spike on Day 1.
    - Activity flattens and spreads more evenly throughout the rest of the month.
    - The spike on Day 1 might be related to monthly resets, promotional campaigns, or billing cycles that motivate early-month user activity.
    - This suggests that Day 1 is critical for user engagement and should be considered when planning campaigns or tracking monthly activity patterns.
- Hour
    - User activity is fairly evenly distributed across the 24-hour cycle, but there are noticeable peaks in the afternoon (around 14:00–18:00) and a slight dip during early morning hours (e.g., 2:00–6:00).
    - The peaks in the afternoon could align with user availability, such as breaks during work or school hours, while the dip during early morning aligns with typical sleep hours.
    - This suggests that ad placements might perform best in the afternoon and early evening when user engagement is higher.

Categorical Variables Univariate Analysis

```
# Filter out categorical features for the univariate analysis
categorical_features = ["country", "gender", "ad_position", "ad_type", "ad_category"]
df_categorical = final_data[categorical_features]
```

```
# Set up the subplot for a 4x2 layout
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(15, 18))

# Loop to plot bar charts for each categorical feature in the 4x2 layout
for i, col in enumerate(categorical_features):
    row = i // 2
    col_idx = i % 2

    # Calculate frequency percentages
    value_counts = final_data[col].value_counts(normalize=True).mul(100).sort_values()

    # Plot bar chart
    value_counts.plot(kind='barh', ax=ax[row, col_idx], width=0.8, color='red')

    # Add frequency percentages to the bars
    for index, value in enumerate(value_counts):
        ax[row, col_idx].text(value, index, str(round(value, 1)) + '%', fontsize=15, weight='bold', va='center')

    ax[row, col_idx].set_xlim([0, 95])
    ax[row, col_idx].set_xlabel('Frequency Percentage', fontsize=12)
    ax[row, col_idx].set_title(f'{col}', fontsize=20)

ax[2,1].axis('off')
plt.suptitle('Distribution of Categorical Variables', fontsize=22)
plt.tight_layout()
plt.subplots_adjust(top=0.95)
plt.show()
```


Distribution of Categorical Variables
country
USA
non-USA
56.9%
43.1%
country
Frequency Percentage
gender
male
female
na
58.3%
41.2%
0.5%
gender
Frequency Percentage
ad_position
top-left
top-right
67.5%
32.5%
ad_position
Frequency Percentage
ad_type
dd
native
66.8%
33.2%
ad_type
Frequency Percentage
ad_category
Computers & Electronics
Shop - Apparel
Shop - Luxury Goods
Travel
Shop - Sporting Goods
Shop - Photo & Video Services
Games
Shop - Toys
Shop - Wholesalers & Liquidatr
Shop - Event Ticket Sales
Autos & Vehicles
Shop - Gifts & Special Event
Food & Drink
Real Estate
27.7%
16.8%
14.8%
8.0%
7.4%
6.8%
4.2%
4.0%
3.2%
2.4%
2.1%
1.3%
0.7%
0.6%
ad_category
Frequency Percentage

Inferences:

- Country:
  - 56.9% of the participants are from the USA, while 43.1% are from non-USA countries.
  - This suggests that the dataset is more representative of a USA-based audience, but it still has a significant portion of non-USA participants, which may offer some diversity in regional preferences or behavior.
- Gender:
  - The majority of participants are male (58.6%), while 41.4% are female.
  - The imbalance may affect the generalizability of any insights or models based on this data, particularly if there are gender-specific trends in the income or other factors being studied.
- Ad Position:
  - 67.5% of the ads are placed in the top-left position, while only 32.5% are in the top-right.
  - There's a clear preference or dominance of the top-left ad position, which might suggest that this position is more frequently used or perhaps more effective in drawing user attention.
- Ad Type:
  - The majority of ads (66.8%) are dd type, while the remaining 33.2% are native ads.
  - The dominance of the dd ad type suggests this format is more popular or widely used, which might correlate with its effectiveness or ease of placement.
- Ad Category:
  - The most frequent ad categories are:
    - Computers & Electronics (27.7%),
    - Shop - Apparel (16.8%),
    - Shop - Luxury Goods (14.8%).
  - These three categories make up nearly 60% of all ads, indicating that technology, apparel, and luxury goods are the most commonly advertised.
  - Less common categories include Real Estate (0.6%), Food & Drink (0.7%), and Shop - Gifts & Special Events (1.3%), suggesting that these areas might either have a smaller market or lower ad spend in this dataset.

Bivariate Analysis

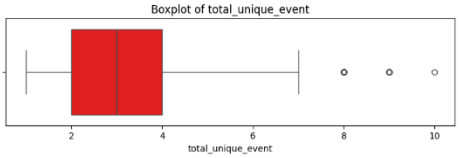
For bivariate analysis of the dataset's features in relation to the target variable:

- For numeric data: I will utilize bar plots to display the average value of each feature across the target classes and KDE plots to examine the distribution of each feature within the target classes. This helps in identifying how the features differ between the two target outcomes.
- For categorical data: I will use 100% stacked bar plots to illustrate the proportions of each category across the target classes, providing a clear view of how various categories within a feature are associated with the target.

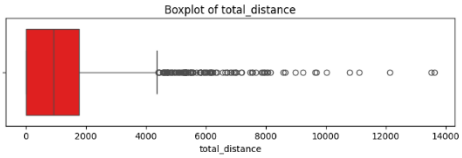
Numerical Features vs Target

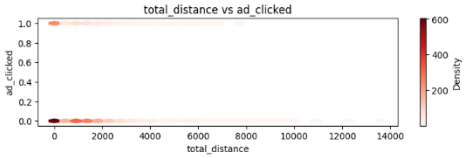
I am going to visualize each numerical feature against the target using two types of charts, Boxplot and Hexbin Plot

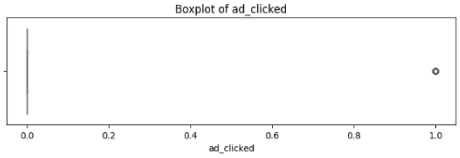
```
fig, ax = plt.subplots(len(numerical_features), 2, figsize=(20, 35))
plt.subplots_adjust(hspace=0.5)

for i in range(len(numerical_features)):

    sns.boxplot(x=numerical_features[i], data=final_data, ax=ax[i, 0], color='red')
    ax[i, 0].set_title(f"Boxplot of {numerical_features[i]}")

    hb = ax[i, 1].hexbin(
        final_data[numerical_features[i]],
        final_data['ad_clicked'],
        gridsize=30,
        cmap="Reds",
        mincnt=1
    )
    cb = fig.colorbar(hb, ax=ax[i, 1])
    cb.set_label("Density")
    ax[i, 1].set_title(f"{numerical_features[i]} vs ad_clicked")
    ax[i, 1].set_xlabel(numerical_features[i])
    ax[i, 1].set_ylabel('ad_clicked')


plt.show()
```

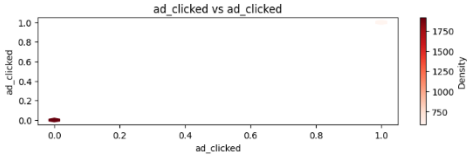

Boxplot of total_unique_event
total_unique_event

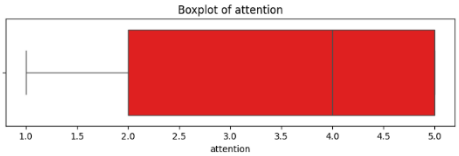

total_unique_event vs ad_clicked
ad_clicked
total_unique_event
Density

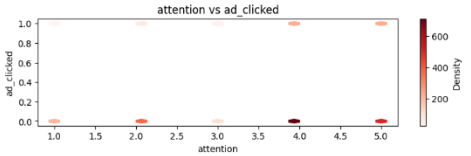

Boxplot of total_distance
total_distance

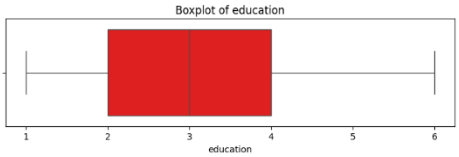

total_distance vs ad_clicked
ad_clicked
total_distance
Density

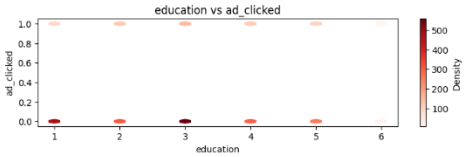

Boxplot of ad_clicked
ad_clicked

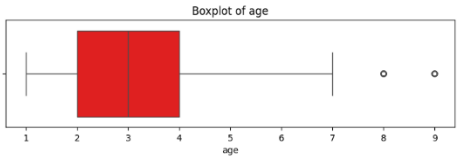

ad_clicked vs ad_clicked
ad_clicked
ad_clicked
Density


Boxplot of attention
attention

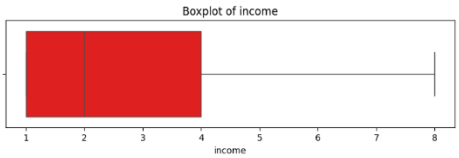

attention vs ad_clicked
ad_clicked
attention
Density


Boxplot of education
education

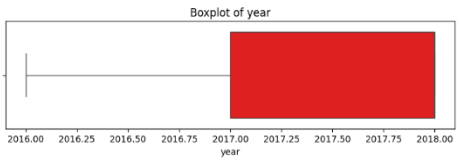

education vs ad_clicked
ad_clicked
education
Density


Boxplot of age
age

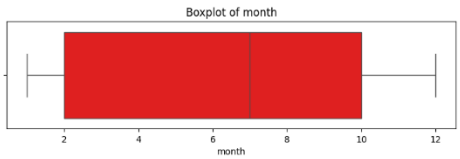

age vs ad_clicked
ad_clicked
age
Density


Boxplot of income
income

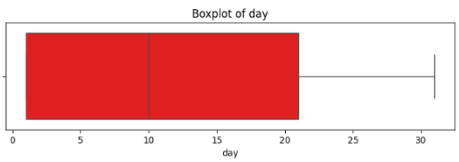

income vs ad_clicked
ad_clicked
income
Density

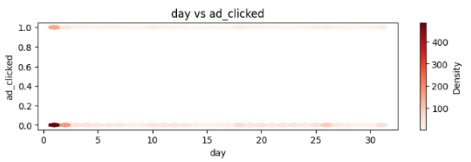

Boxplot of year
year

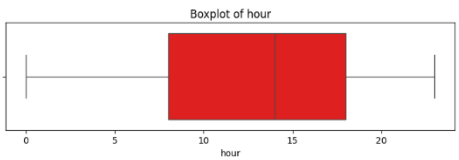

year vs ad_clicked
ad_clicked
year
Density

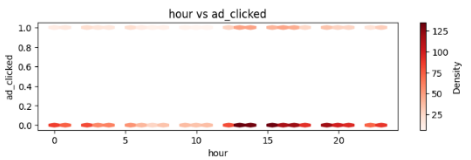

Boxplot of month
month


month vs ad_clicked
ad_clicked
month
Density


Boxplot of day
day


day vs ad_clicked
ad_clicked
day
Density


Boxplot of hour
hour

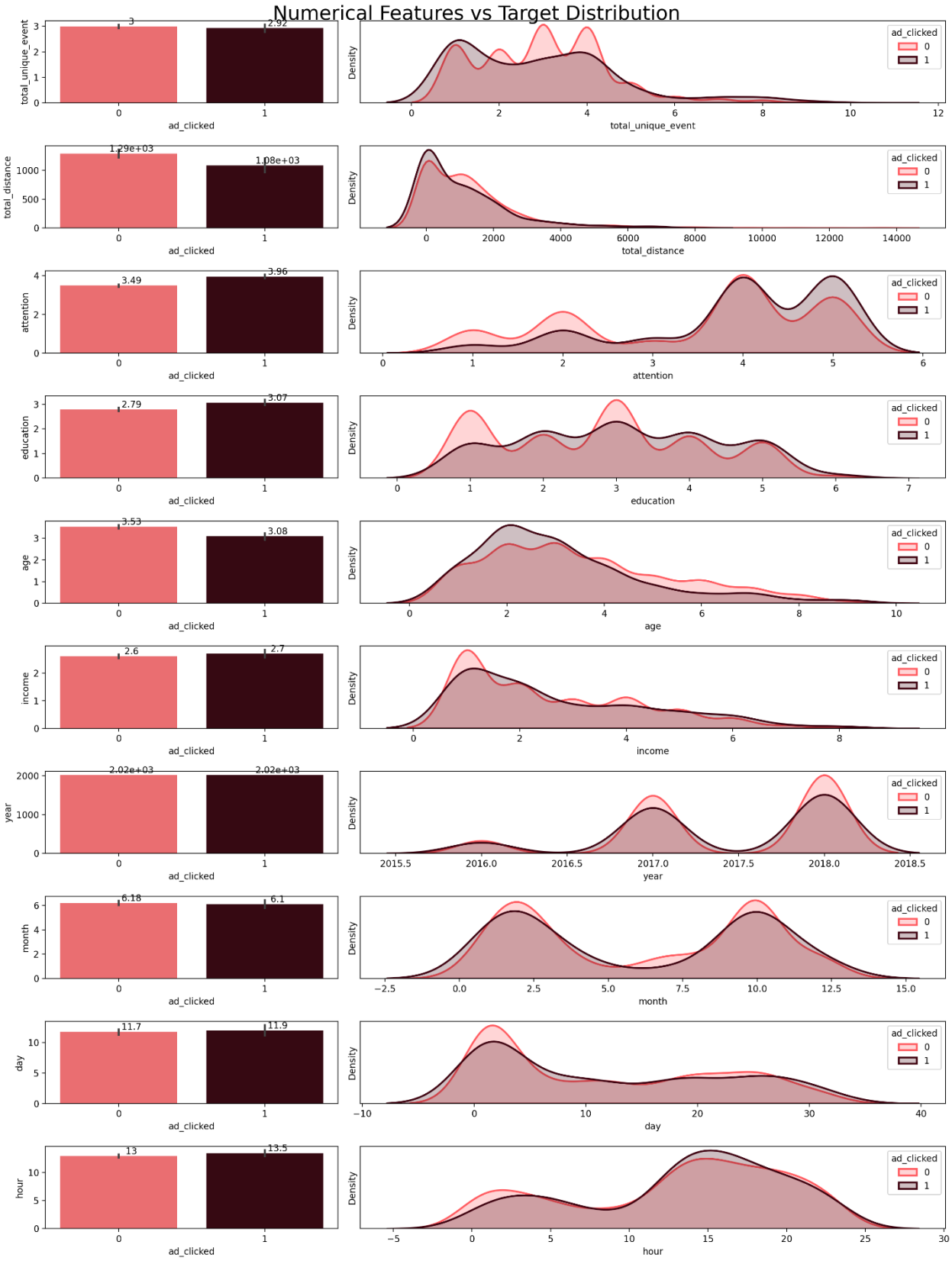

hour vs ad_clicked
ad_clicked
hour
Density

Inferences:

- Movement Distance:
    - The distribution of total distance is right-skewed, with significant outliers beyond 4,000 units. The majority of users travel shorter distances, concentrated below 2,000 units. The boxplot shows that most users fall within a compact range, with a long tail of outliers extending up to nearly 14,000 units.
- Total Unique Events:
    - Users mostly engage in 2–4 unique events, with outliers up to 10. Similar to total events, ad_clicked = 1 users are more evenly distributed across unique events but still concentrate at lower values.
- Attention:
    - Attention scores are fairly evenly distributed across the 1–5 range. Higher attention scores (4–5) appear to correlate slightly with ad clicks (ad_clicked = 1).
- Education:
    - Most users have education levels between 2–4, with sparse outliers above 6.No strong relationship observed between education level and ad clicks.
- Age:
    - Age skews toward younger users, with the majority between 2–5 (likely representing age groups or brackets). No clear relationship between age and ad clicks; density is distributed across all age ranges.
- income:
    - Most users fall within the income range of 2–4, with occasional outliers up to 8. No strong connection observed between income and ad clicks; density is concentrated around lower-income groups for both 0 and 1.
- Year:
    - User data is concentrated in 2017 and 2018, with minimal representation in 2016. Most clicks and non-clicks occur in 2017 and 2018, consistent with higher user engagement during these periods.
- Month:
    - Engagement is distributed fairly evenly across months, with slight peaks in February and October. No clear correlation between specific months and ad clicks, though density for both 0 and 1 is spread across all months.
- Day:
    - Engagement is fairly evenly distributed across days of the month. Ad clicks (1) are slightly more frequent toward the beginning of the month, though density remains low overall.
- Hour:
    - Engagement is consistent across hours of the day, with no strong peaks. Ad clicks and non-clicks are evenly distributed across the day, with slightly higher density in the afternoon.

I will visualize each continuous feature against the target using bar plots to show mean values and KDE plots to illustrate the distribution for each target category.

```
# Define a list containing the names of important numerical features in the dataset
Num_Features = ['total_unique_event', 'total_distance', 'attention', 'education', 'age', 'income', 'year', 'month', 'day', 'hour']

# Set consistent colors for both bar and KDE plots
bar_color_0 = '#ff595e'
bar_color_1 = '#3d010e'

sns.set_palette([bar_color_0, bar_color_1])  # Set the palette globally

Target = 'ad_clicked'
fig, ax = plt.subplots(10, 2, figsize=(15, 20), dpi=200, gridspec_kw={'width_ratios': [1, 2]})

for i, col in enumerate(Num_Features):
    # Barplot with consistent colors
    graph = sns.barplot(data=final_data, x=Target,y=col,ax=ax[i, 0],
        palette=[bar_color_0, bar_color_1]
    )
    # KDE Plot with the same colors
    sns.kdeplot(data=final_data[final_data[Target] == 0],x=col,fill=True,linewidth=2,ax=ax[i, 1],label='0',color=bar_color_0)
    sns.kdeplot( data=final_data[final_data[Target] == 1],x=col,fill=True,linewidth=2,ax=ax[i, 1],label='1', color=bar_color_1)

    ax[i, 1].set_yticks([])
    ax[i, 1].legend(title='ad_clicked', loc='upper right')

    # Add bar sizes to the barplot
    for cont in graph.containers:
        graph.bar_label(cont, fmt='         %.3g')

# Set overall title and layout
plt.suptitle('Numerical Features vs Target Distribution', fontsize=22)
plt.tight_layout()
plt.show()
```

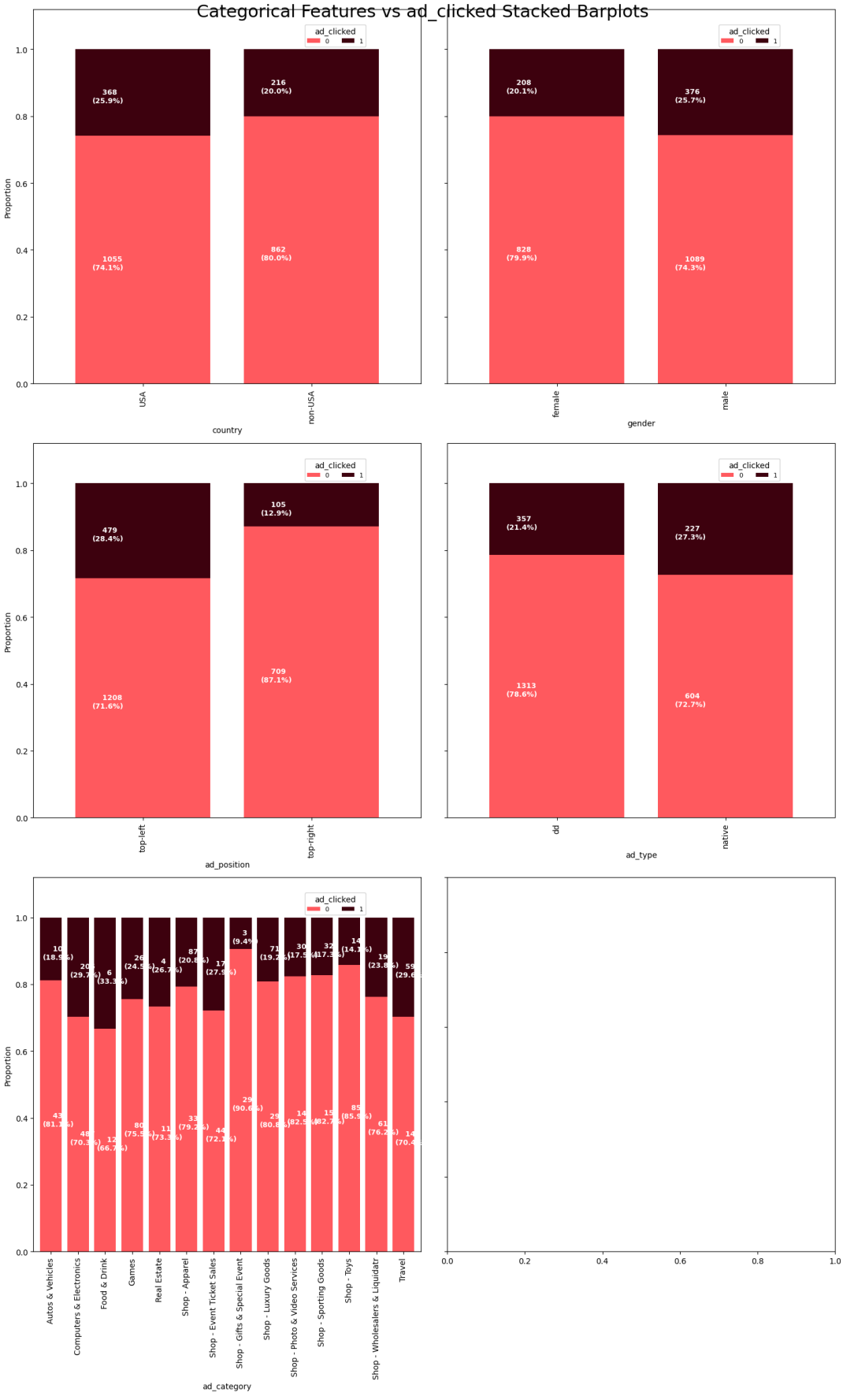

Numerical Features vs Target Distribution
total_unique_event
total_distance
attention
education
age
income
year
month
day
hour
ad_clicked
Density
0
1

- Total Unique Events:
    - The KDE plot reveals overlapping distributions for both groups, but users who didn't click on ads (ad_clicked = 0) have slightly more activity concentrated in higher total unique event values (e.g., around 4).
- Total Distance
    - The distributions are quite similar between the two groups, with some overlap in the KDE plot. The mean total distance is slightly higher for users who clicked on ads. However, this feature does not show a strong differentiation between the groups.
- Attention
    - The KDE plot highlights that users who clicked on ads tend to have a slightly higher attention score, with peaks closer to the higher end of the scale. This indicates that attention may play a moderate role in ad engagement.
- Education
    - The distributions for both groups are very similar, with overlapping KDE plots. The mean education level is slightly higher for users who clicked on ads, but this feature shows limited distinguishing power.
- Age
    - Users who clicked on ads are slightly younger on average, as indicated by the KDE plot. The peak for ad_clicked = 1 skews slightly towards the lower age range compared to the other group.
- Income
    - The KDE plot shows substantial overlap, but the mean income for users who clicked on ads is slightly higher. This feature has minimal distinguishing power.
- Year
    - The distributions for both groups are almost identical, indicating no significant relationship between the year and ad clicks.
- Month
    - The KDE plot shows overlapping distributions, with users who clicked on ads slightly concentrated around mid-year months. This feature has limited differentiation.
- Day
    - Both groups exhibit nearly identical distributions, indicating that the day of the month has no meaningful impact on ad clicks.
- Hour
    - The KDE plot reveals noticeable differences. Users who clicked on ads tend to interact more during specific hours (around the later hours), suggesting that the time of day could have a moderate impact on ad clicks.

Based on the visual differences in distributions and mean values, most features do not have a significant impact on the target variable. However, Attention (attention) and Hour (hour) appear to have the greatest impact compared to the other features.

Categorical Features vs Target:

```
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(15,25))
for i,col in enumerate(categorical_features):

    # Create a cross tabulation showing the proportion of purchased and non-purchased loans for each category of the feature
    cross_tab = pd.crosstab(index=final_data[col], columns=final_data['ad_clicked'])

    # Using the normalize=True argument gives us the index-wise proportion of the data
    cross_tab_prop = pd.crosstab(index=final_data[col], columns=final_data['ad_clicked'], normalize='index')

    # Define colormap
    cmp = ListedColormap(['#ff595e','#3d010e'])

    # Plot the cross tabulation as a stacked bar chart
    x, y = i//2, i%2
    cross_tab_prop.plot(kind='bar', ax=ax[x,y], stacked=True, width=0.8, colormap=cmp,
                        legend=False, ylabel='Proportion', sharey=True)

    # Add the proportions and counts of the individual bars to our plot
    for idx, val in enumerate([*cross_tab.index.values]):
        for (proportion, count, y_location) in zip(cross_tab_prop.loc[val],cross_tab.loc[val],cross_tab_prop.loc[val].cumsum()):
            ax[x,y].text(x=idx-0.3, y=(y_location-proportion)+(proportion/2)-0.03,
                         s = f'    {count}\n({np.round(proportion * 100, 1)}%)',
                         color = "white", fontsize=9, fontweight="bold")

    # Add legend
    ax[x,y].legend(title='ad_clicked', loc=(0.7,0.9), fontsize=8, ncol=2)
    # Set y limit
    ax[x,y].set_ylim([0,1.12])
    # Rotate xticks
    ax[x,y].set_xticklabels(ax[x,y].get_xticklabels(), rotation=90)

plt.suptitle('Categorical Features vs ad_clicked Stacked Barplots', fontsize=22)
plt.tight_layout()
plt.show()
```


Categorical Features vs ad_clicked Stacked Barplots
ad_clicked
0
1
Proportion
368
(25.9%)
1055
(74.1%)
216
(20.0%)
862
(80.0%)
USA
non-USA
country
208
(20.1%)
828
(79.9%)
376
(25.7%)
1089
(74.3%)
female
male
gender
479
(28.4%)
1208
(71.6%)
105
(12.9%)
709
(87.1%)
top-left
top-right
ad_position
357
(21.4%)
1313
(78.6%)
227
(27.3%)
604
(72.7%)
dd
native
ad_type
Autos & Vehicles
Computers & Electronics
Food & Drink
Games
Real Estate
Shop - Apparel
Shop - Event Ticket Sales
Shop - Gifts & Special Event
Shop - Luxury Goods
Shop - Photo & Video Services
Shop - Sporting Goods
Shop - Toys
Shop - Wholesalers & Liquidatr
Travel
ad_category

Inferences:

- Country:
    - Users from the USA clicked on ads at a rate of 25.9%. The majority (74.1%) did not click on ads. Users outside the USA clicked on ads slightly less frequently (~20%). A higher proportion of users (80%) from non-USA regions did not click on ads. Users from the USA are slightly more likely to engage with ads compared to non-USA users, which might indicate cultural or regional differences in user behavior.
- Gender:
    - Males clicked on ads at a rate of 25.7%, slightly higher than females. Most males (74.5%) did not click on ads. Females clicked on ads less frequently (~20.1%), with ~79.9% not clicking. Male users show a marginally higher likelihood of clicking ads compared to females, but the overall difference is not substantial.
- Ad Position:
    - Ads in the "top-left" position had a click-through rate (CTR) of 28.4%. Most users (71.6%) ignored ads in this position. Ads in the "top-right" position had a lower CTR of 12.9%. A significant majority (87.1%) did not click ads in this position. Ads placed in the "top-left" position perform significantly better than those in the "top-right," likely due to higher visibility or user attention.
- Ad Type:
    - This ad type had a CTR of ~21.4%, with ~78.6% of users ignoring it. Native ads performed better, with a CTR of ~27.3% and ~72.7% of users ignoring them. Native ads are more effective than dynamic display ads, possibly because they blend better with the content or feel less intrusive to users.
- ad_category:
    - Across all features, the proportion of users clicking on ads (ad_clicked = 1) is relatively low, indicating low overall engagement with ads. Geography: Users from the USA are more likely to click ads. Ad Placement: Ads in the "top-left" position perform much better than "top-right." Ad Type: Native ads outperform dynamic display ads. Category: Certain categories like "Travel" and "Computers & Electronics" drive higher engagement.

In summary, based on the visual representation:

- Moderate Impact on Target: **Ad Position**, **ad_category**
- Lower Impact on Target: Gender, Ad Type, Country

It is good sometimes we look at the dataset:

```
final_data.head(1)
```

	movement_distance	total_unique_event	ad_clicked	attention	country	education	age	income	gender	ad_position	ad_type	ad_category	year	month	day	hour
0	0.0	2	0	5	non-USA	3.0	3.0	6.0	male	top-left	dd	Computers & Electronics	2018	10	2	7

Data Preprocessing

Outliers:

Outliers are extreme data points that deviate significantly from the majority of values in a dataset, either being much higher or much lower than the typical range.

Here we can see the number of outliers in our dataset

```
# drop ad_clicked from numerical_features
numerical_features.remove('ad_clicked')

Q1 = final_data[numerical_features].quantile(0.25)
Q3 = final_data[numerical_features].quantile(0.75)
IQR = Q3 - Q1
outliers_count_specified = ((final_data[numerical_features] < (Q1 - 1.5 * IQR)) | (final_data[numerical_features] > (Q3 + 1.5 * IQR))).sum()

outliers_count_specified
```

	0
total_unique_event	43
total_distance	108
attention	0
education	0
age	87
income	0
year	0
month	0
day	0
hour	0

dtype: int64

For modeling, our focus will be on tree-based models, as these algorithms are generally robust to outliers. Since tree-based methods make splits based on feature values, outliers are typically isolated in leaf nodes, resulting in minimal impact on the overall decision-making process. Therefore, we can retain outliers in the dataset.

Duplicate Values:

Removing duplicate values involves eliminating duplicate records from the dataset before using it in a machine learning algorithm. This ensures that only unique samples are included, improving the quality of training and evaluation for the machine learning model.

```
final_data.duplicated().sum()
```

```
4
```

```
final_data.drop_duplicates(inplace=True)
```

```
final_data.duplicated().sum()
```

```
0
```

Due to project limitations, we have decided not to include 'age,' 'income,' 'education,' 'gender,' and 'attention' in our dataset for modeling; therefore, these features will be dropped.

```
# drop age income
filtered_data.drop(['age', 'income','education','gender','attention'], axis=1, inplace=True)
```

Categorical Variables Encoding:

After analyzing the dataset, we can categorize the features into three groups:

- **No Encoding Needed**: These are the features that do not require any form of encoding because they are already in a numerical format that can be fed into a model.
- **One-Hot Encoding**: This is required for nominal variables, which are categorical variables without any intrinsic order. One-hot encoding converts each unique value of the feature into a separate column with a 1 or 0, indicating the presence of that value.
- **Label Encoding**: This is used for ordinal variables, which are categorical variables with a meaningful order. Label encoding assigns a unique integer to each category in the feature, maintaining the order of the values. (Shahules, 2019)

By categorizing the features into these groups, we can apply the appropriate encoding method to each feature, preparing the dataset for modeling.

```
filtered_data.head(1)
```

	movement_distance	total_unique_event	ad_clicked	country	ad_position	ad_type	ad_category	year	month	day	hour
0	0.0	2	0	non-USA	top-left	dd	Computers & Electronics	2018	10	2	7

No Encoding Needed:

The following features do not require any encoding as they are either numerical, binary, or ordinal variables that have already been encoded as numbers:

- total_distance: This is a numerical variable.
- ad_clicked: This is Target and a numerical variable.
- total_unique_event: This is a numerical variable.
- year: This is a numerical variable.
- month: This is a numerical variable.
- day: This is a numerical variable.
- hour: This is a numerical variable.

One-Hot Encoding:

The following features are nominal variables and should be one-hot encoded:

- country: This is a nominal variable.
- ad_position: This is a nominal variable.
- ad_type: This is a nominal variable.
- ad_category: This is a nominal variable.

```
# One-hot encode the columns "country", "gender", "ad_position", "ad_type", "ad_category"
df = pd.get_dummies(final_data, columns=['country', 'ad_position', 'ad_type', 'ad_category'], drop_first=True)
df = df.astype(int)
```

```
df.head()
```

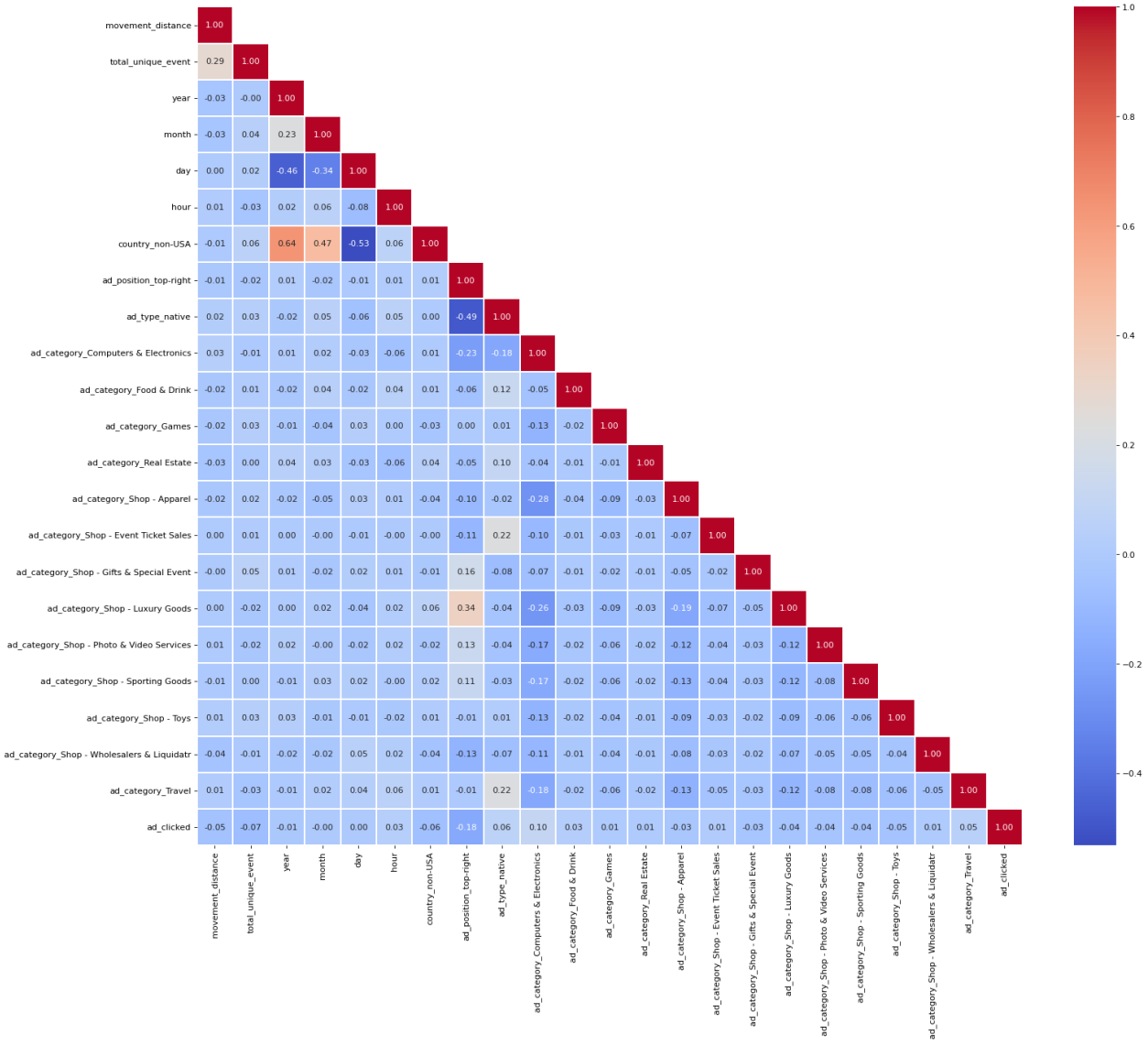
	total_unique_event	total_distance	ad_clicked	year	month	day	hour	country_non-USA	ad_position_top-right	ad_type_native	ad_category_Computers & Electronics	ad_category_Food & Drink	ad_c
	3	312	1	2018	10	1	13	1	0	1	0	0	
	4	833	1	2018	10	1	17	1	0	1	0	0	
	2	621	0	2018	9	29	13	1	0	0	1	0	
	3	786	0	2017	3	16	10	1	0	1	0	1	
	6	2870	1	2017	2	10	22	0	1	0	0	0	

```
df.shape
```

```
(2497, 23)
```

Correlation Analysis

```
# Calculation of the Spearman correlation
target = 'ad_clicked'
df_ordered = pd.concat([df.drop(target,axis=1), df[target]],axis=1)
corr = df_ordered.corr(method='spearman')

# Create a mask so that we see the correlation values only once
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask,1)] = True

# Plot the heatmap correlation
plt.figure(figsize=(22,18), dpi=80)
sns.heatmap(corr, mask=mask, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.2)
plt.show()
```



Inferences:

The heatmap reveals mostly weak to moderate correlations with the target variable (ad_clicked). However, certain features, such as ad position, demonstrate a slightly higher impact on ad clicks compared to others. Additionally, it is crucial for modeling to evaluate whether the correlations between features and the target variable are appropriate, as this influences the predictive power and effectiveness of the model.

Train Test Split:

First, it is essential to define the features (X) and the target labels (y) in the dataset.

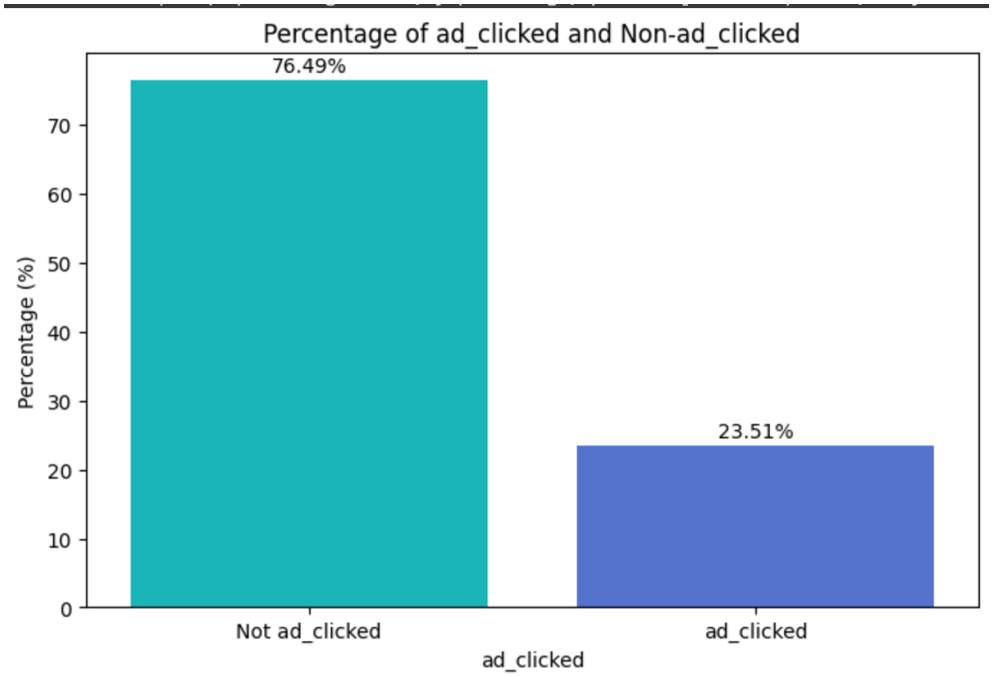
```
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']
```

In supervised machine learning, performing a train-test split is essential to evaluate a model's performance after training. The dataset is typically divided into two parts: the training set, used to train the model, and the test set, used to assess how well the model performs on unseen data. This approach helps evaluate the model's ability to generalize by measuring its performance on new, unseen examples. Train-test splitting is also useful for identifying issues such as bias and variance, ensuring that the model generalizes effectively to real-world scenarios.

Imbalanced dataset:

An imbalanced dataset refers to a dataset where the classes are not evenly distributed. In such cases, the number of samples in each target class is unequal, with some classes having significantly more samples than others.

```
# Calculating the percentage of each class
percentage = df['ad_clicked'].value_counts(normalize=True) * 100

# Plotting the percentage of each class
plt.figure(figsize=(8, 5))
ax = sns.barplot(x=percentage.index, y=percentage, palette=['darkturquoise', 'royalblue'])
plt.title('Percentage of Cancellations and Non-Cancellations')
plt.xlabel('Is Canceled')
plt.ylabel('Percentage (%)')
plt.xticks(ticks=[0, 1], labels=['Not ad_clicked', 'ad_clicked'])
plt.yticks(ticks=range(0,80,10))

# Displaying the percentage on the bars
for i, p in enumerate(percentage):
    ax.text(i, p + 0.5, f'{p:.2f}%', ha='center', va='bottom')

plt.show()
```



We can see that the dataset is imbalanced, with approximately 76% of the data belonging to the "not-click" class (0) and about 24% to the "click" class (1)

Techniques to Handle Imbalanced Dataset:

1. The Approach to Train-Test Split:

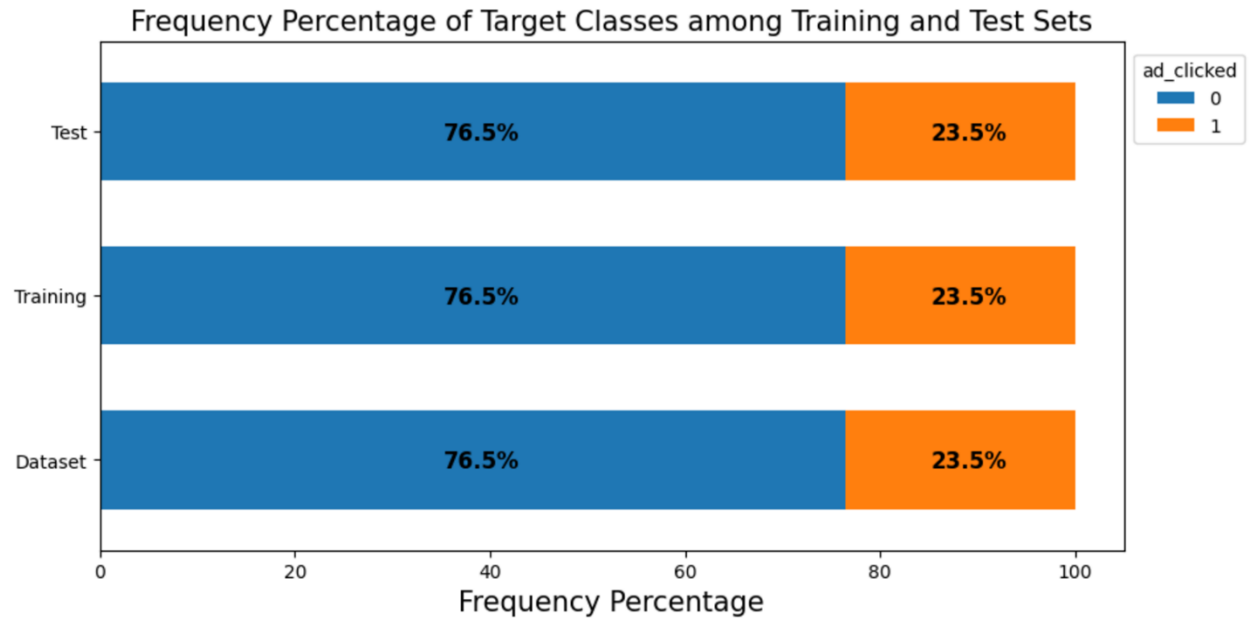
When dealing with imbalanced datasets, it is crucial to use stratification during the train-test split. Stratification ensures that the class proportions in the training and test sets remain consistent with the original dataset. This approach is important because it provides a more accurate evaluation of the model, preventing bias caused by one class being over-represented in either the training or test set. Additionally, stratified sampling preserves any existing trends or correlations between classes in the dataset, maintaining the integrity of the data after splitting.

```
# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)
```

We utilized stratification to ensure that the distribution of the target variable (ad_clicked) remains consistent across both the training and test sets.

Here we can see the distribution of each class in both the training and test sets.

```python
# Define a dataframe containing frequency percentages
df_perc = pd.concat([y.value_counts(normalize=True).mul(100).round(1),
                     y_train.value_counts(normalize=True).mul(100).round(1),
                     y_test.value_counts(normalize=True).mul(100).round(1)], axis=1)
df_perc.columns=['Dataset','Training','Test']
df_perc = df_perc.T

# Plot frequency percentages barplot
df_perc.plot(kind='barh', stacked=True, figsize=(10,5), width=0.6)

# Add the percentages to our plot
for idx, val in enumerate([*df_perc.index.values]):
    for (percentage, y_location) in zip(df_perc.loc[val], df_perc.loc[val].cumsum()):
        plt.text(x=(y_location - percentage) + (percentage / 2)-3,
                 y=idx - 0.05,
                 s=f'{percentage}%',
                 color="black",
                 fontsize=12,
                 fontweight="bold")

plt.legend(title='ad_clicked', loc=(1.01,0.8))
plt.xlabel('Frequency Percentage', fontsize=15)
plt.title('Frequency Percentage of Target Classes among Training and Test Sets', fontsize=15)
plt.show()
```



As observed, the samples are divided randomly while ensuring that the class proportions remain consistent across both the training and test sets.

2. The approach for model evaluation:

When evaluating models for imbalanced datasets, it is essential to use metrics such as recall, precision, F1 score, and AUC (area under the curve), which are better suited for handling class imbalance compared to traditional accuracy metrics that do not account for uneven class distributions.

In this project, the primary objective is to classify users who are more likely to click on an ad. The choice of evaluation metrics will play a crucial role in assessing the model's ability to accurately identify these users.

Recall measures the proportion of actual positive cases correctly identified by the model. A high recall indicates a low number of false negatives, which is particularly important in this project as it ensures the model captures most instances of user ad clicks without missing many.

Precision measures the proportion of positive predictions made by the model that are actually correct. A high precision score indicates a low number of false positives, which is desirable in this project as it ensures the model accurately identifies users who clicked on ads without incorrectly labeling many non-ad click users.

The **F1-score** represents the balance between recall and precision, calculated as the harmonic mean of the two metrics. A high F1-score indicates that the model achieves a good trade-off, combining both high recall and high precision effectively.

For this project, both recall and precision for class '1' (users who clicked on an ad) are important metrics. Therefore, the **F1-score for class '1' should be considered the most important metric**. A high F1-score indicates a balance between correctly identifying as many ad-clicking users as possible (high recall) while minimizing the number of false positives (high precision).

3. Approaches for model building:

Oversampling: Oversampling is a method that involves increasing the representation of the minority class by duplicating its examples, ensuring a more balanced dataset. This helps reduce the risk of the model overfitting to the majority class.

Disadvantages: Overfitting, which can occur as a result of duplicating observations from the minority class.

Undersampling: Undersampling is a method that involves reducing the number of majority class examples in the dataset to achieve a more balanced class distribution. This approach helps ensure the model doesn't underfit the minority class.

There are various techniques for undersampling a dataset, with one of the most common being random undersampling. This technique involves randomly selecting and removing a portion of the majority class examples from the dataset.

Disadvantages: Loss of information resulting from removing a significant portion of the majority class in the training set.

SMOTE: SMOTE (Synthetic Minority Oversampling Technique) is an oversampling method that generates synthetic samples for the minority class. This technique helps address the overfitting issue often associated with random oversampling by creating new instances through interpolation between closely located positive instances in the feature space. (SWASTIK, n.d.)

Disadvantages: Since SMOTE generates synthetic data points randomly, the newly created samples may lack real-world value, potentially reducing overall model accuracy. Additionally, in cases where natural class boundaries exist or classes overlap, SMOTE can inadvertently introduce outliers due to the artificial creation of data points.

Tree-Based Algorithms: Tree-based algorithms such as Random Forest, Extra Trees Classifier, and XGBoost are inherently capable of handling imbalanced datasets. They operate by making decisions through majority voting among multiple randomized decision trees, eliminating the need to balance classes before training.

Because we have imbalance dataset we **decided to use tree base algorithms** to handle it.

Models:

Using classifier algorithms is better for predicting ad clicks because target variable (ad_clicked) is a binary classification problem (values typically represented as 0 for "no click" and 1 for "clicked"). Regression models predict continuous values, which are not suitable for problems where the target is like ad_clicked.

Decision Tree Model Building:

Decision Tree model building is a supervised learning technique widely used for both classification and regression tasks. The process involves recursively dividing the dataset into subsets based on the feature that provides the highest information gain or minimizes Gini impurity. Each node in the tree signifies a decision point based on a specific feature, with branches representing possible outcomes, leading to further splits or a leaf node that makes the final prediction. Decision Trees are highly interpretable, capable of handling both numerical and categorical data, and require minimal preprocessing. However, they are susceptible to overfitting, especially with deeper trees, which can be addressed through techniques like pruning or setting a maximum depth limit. (Quinlan, 1985)

This model like all models has some advantages and disadvantages:

Advantages:

- **Easy to understand and interpret:** Decision Trees are straightforward to visualize and comprehend.
- **Handles both numerical and categorical data:** They are versatile and can work with both numerical and categorical inputs.
- **Feature selection**: Decision Trees inherently perform feature selection, as important features tend to appear near the root of the tree.
- **Non-parametric:** They are non-parametric models, meaning they do not assume any specific distribution of the data.

Disadvantages:

- **Overfitting**: Decision Trees are susceptible to overfitting, particularly when the tree is deep or the training dataset is small, leading to poor generalization to unseen data.
- **Instability:** Even small changes in the dataset can lead to significant changes in the tree structure, making them less stable.
- **Bias towards features with many categories**: Decision Trees may favor features with many categories, which can disproportionately influence the tree's construction.

- **Limited ability to approximate complex functions**: They struggle with approximating complex functions due to their reliance on axis-parallel splits at each node.

Model Definition:

First we have to define the base Decision Tree model

```
dt_base = DecisionTreeClassifier(random_state=0)
```

Decision Tree Hyperparameter Tuning:

In this website ad-click prediction problem, the target variable is ad_clicked, which indicates whether a user clicked on an ad (1) or not (0). Both false positives (a user is predicted to have clicked on an ad, but they did not) and false negatives (a user is predicted to have not clicked, but they actually did) can be significant depending on the context.

However, false negatives may be more critical because failing to identify users likely to click on ads could lead to missed revenue opportunities or ineffective ad targeting. Therefore, it is essential to minimize false negatives, which means maximizing recall for the ad_clicked class (1) would be a sensible approach.

Precision is also important because it minimizes false positives, ensuring that the model avoids predicting ad clicks where they do not occur. This is crucial for optimizing ad spend and ensuring resources are not wasted targeting users unlikely to engage.

As a result, the F1-score, which balances both precision and recall, would be an appropriate metric for this project. Specifically, the F1-score for the ad_clicked class (1) should be prioritized as the key metric for evaluating model performance.

New we can create a tune_clf_hyperparameters, he goal is to find the combination of hyperparameters that maximizes the specified scoring metric (typically f1, useful for imbalanced datasets like ad-click prediction). The function returns the best-fit model (best_estimator_) and the optimal hyperparameters for further use. This approach ensures the selected classifier is fine-tuned for performance, particularly on the target task.

```python
def tune_clf_hyperparameters(clf, param_grid, X_train, y_train, scoring='f1', n_splits=5):
    '''
    This function optimizes the hyperparameters for a classifier by searching over a specified hyperparameter grid.
    It uses GridSearchCV and cross-validation (StratifiedKFold) to evaluate different combinations of hyperparameters.
    The combination with the highest F1-score for class 1 (ad click) is selected as the default scoring metric.
    The function returns the classifier with the optimal hyperparameters.
    '''

    # Create the cross-validation object using StratifiedKFold to ensure the class distribution is the same across all the folds
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=0)

    # Create the GridSearchCV object
    clf_grid = GridSearchCV(clf, param_grid, cv=cv, scoring=scoring, n_jobs=-1)

    # Fit the GridSearchCV object to the training data
    clf_grid.fit(X_train, y_train)

    # Get the best hyperparameters
    best_hyperparameters = clf_grid.best_params_

    # Return best_estimator_ attribute which gives us the best model that has been fitted to the training data
    return clf_grid.best_estimator_, best_hyperparameters
```

Decision Tree classifiers are susceptible to overfitting, which happens when the tree becomes overly complex and closely aligns with the training data, including capturing noise and irrelevant patterns.

How can prevent overfitting in decision tree classifiers:

- **Pruning**: Eliminating branches of the tree that have minimal impact on classification accuracy.
- **Using Ensemble Methods**: Combining multiple decision trees, such as in Random Forest or Gradient Boosting, to create a more robust and reliable model.
- **Limiting Tree Size**: Defining constraints like the minimum number of samples required to split a node or setting a maximum depth for the tree to reduce complexity.

hyperparameters for Decision Tree Classifiers:

- Criterion: Specifies the metric used to evaluate the quality of a split. Commonly used criteria include "Gini Impurity" and "Information Gain."
- Maximum Depth: Defines the maximum allowable depth of the tree. Deeper trees are more complex and prone to overfitting, so setting a maximum depth helps control model complexity and reduce overfitting.
- Minimum Samples per Split: Sets the minimum number of samples required to split an internal node. Nodes with fewer samples than this threshold cannot be split further, which helps to limit tree size and prevent overfitting.
- Minimum Samples per Leaf: Specifies the minimum number of samples required for a leaf node. Leaf nodes with fewer samples than this threshold are pruned, simplifying the model.

- Maximum Features: Determines the maximum number of features considered when splitting a node. Limiting the number of features reduces model complexity and helps prevent overfitting.
- Class Weight: Specifies weights for each class, often used to handle class imbalance in the dataset. (scikit-learn., n.d.)

First, I used only max_depth as a hyperparameter to tune the model. After obtaining the results, I proceeded to refine and improve the model further.

```
# Define the hyperparameter grid
param_grid_dt = {
    'max_depth': [5, 6, 7, 8],
}

# Call the function for hyperparameter tuning
best_dt, best_dt_hyperparams = tune_clf_hyperparameters(dt_base, param_grid_dt, X_train, y_train)

print('DT Optimal Hyperparameters: \n', best_dt_hyperparams)
```

```
DT Optimal Hyperparameters:
 {'max_depth': 7}
```

To simplify and standardize the evaluation of different models, we will create a set of functions to compute essential performance metrics. This structured approach will ensure consistency in model assessments and make it easier to compare their performance effectively.

```
def model_evaluation(clf, X_train, X_test, y_train, y_test, model_name):
    '''
    This function provides a complete report of the model's performance
including classification reports,
    confusion matrix and ROC curve.
    '''
    sns.set(font_scale=1.2)

    # Generate classification report for training set
    y_pred_train = clf.predict(X_train)
    print("\n\t  Classification report for training set")
    print("-"*55)
    print(classification_report(y_train, y_pred_train))

    # Generate classification report for test set
```

```
    y_pred_test = clf.predict(X_test)
    print("\n\t   Classification report for test set")
    print("-"*55)
    print(classification_report(y_test, y_pred_test))
    # Create figure and subplots
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5), dpi=100,
gridspec_kw={'width_ratios': [2, 2, 1]})

    # Define a colormap
    royalblue_r = LinearSegmentedColormap.from_list('royalblue_r',
['royalblue', 'white'])


    # Plot confusion matrix for test set
    ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test,
colorbar=False, cmap=royalblue_r, ax=ax1)
    ax1.set_title('Confusion Matrix for Test Data')
    ax1.grid(False)

    # Plot ROC curve for test data and display AUC score
    RocCurveDisplay.from_estimator(clf, X_test, y_test, ax=ax2)
    ax2.set_xlabel('False Positive Rate')
    ax2.set_ylabel('True Positive Rate')
    ax2.set_title('ROC Curve for Test Data (Positive label: 1)')

    # Report results for the class specified by positive label
    result = metrics_calculator(clf, X_test, y_test, model_name)
    table = ax3.table(cellText=result.values, colLabels=result.columns,
rowLabels=result.index, loc='center')
    table.scale(0.6, 2)
    table.set_fontsize(12)
    ax3.axis('tight')
    ax3.axis('off')
    # Modify color
    for key, cell in table.get_celld().items():
        if key[0] == 0:
            cell.set_color('royalblue')
    plt.tight_layout()
plt.show()
```

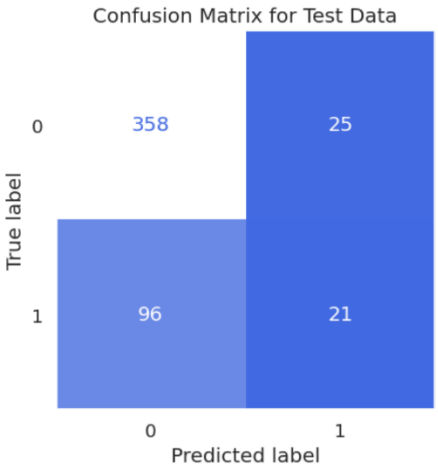
The purpose of this function is to standardize and streamline the evaluation of multiple models, making it easy to compare their performance on test data, especially for imbalanced classification problems like ad-click prediction.

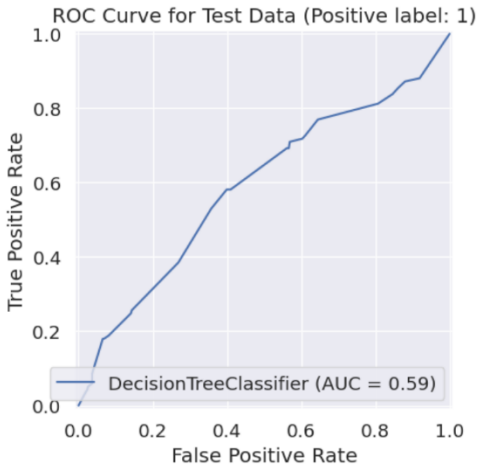
```
def metrics_calculator(clf, X_test, y_test, model_name):
    '''
    This function calculates all desired performance metrics for a given model on test data.
    The metrics are calculated specifically for class 1.
    '''
    y_pred = clf.predict(X_test)
    result = pd.DataFrame(data=[accuracy_score(y_test, y_pred),
                                precision_score(y_test, y_pred, pos_label=1),
                                recall_score(y_test, y_pred, pos_label=1),
                                f1_score(y_test, y_pred, pos_label=1),
                                roc_auc_score(y_test, clf.predict_proba(X_test)[:,1])],
                          index=['Accuracy','Precision (Class 1)','Recall (Class 1)','F1-score (Class 1)','AUC (Class 1)'],
                          columns = [model_name])

    result = (result * 100).round(2).astype(str) + '%'
    return result
```

Now we can call the functions to evaluate our Decision Tree (DT) classifier.

```
model_evaluation(best_dt, X_train, X_test, y_train, y_test, 'Decision Tree')
```

```
         Classification report for training set
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.81      0.96      0.88      1531
           1       0.68      0.28      0.40       466

    accuracy                           0.80      1997
   macro avg       0.75      0.62      0.64      1997
weighted avg       0.78      0.80      0.77      1997


          Classification report for test set
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.93      0.86       383
           1       0.46      0.18      0.26       117

    accuracy                           0.76       500
   macro avg       0.62      0.56      0.56       500
weighted avg       0.71      0.76      0.72       500
```





	Decision Tree
Accuracy	75.8%
Precision (Class 1)	45.65%
Recall (Class 1)	17.95%
F1-score (Class 1)	25.77%
AUC (Class 1)	58.85%

Inferences:

Our **Decision Tree model's** performance on the test data is as follows:

- **Accuracy**: 75.8%, indicating that the model correctly predicts whether a user will click on an advertisement approximately 76% of the time.
- **Precision (Class 1 - Click)**: 45.65%, meaning that of all the predictions where the model identified a user clicking on an advertisement, only about 46% were correct.
- **Recall (Class 1)**: 17.95%, showing that the model successfully identified only about 18% of the actual clicks on advertisements.
- **F1-Score (Class 1)**: 25.77%, which is the harmonic mean of Precision and Recall, reflecting a poor balance between these two metrics for the minority class (Class 1).
- **AUC (Area Under the ROC Curve) for Class 1**: 58.85%, indicating moderate discriminative ability between users who click and do not click on advertisements.

The recall for Class 1 (17.95%) is low, meaning that the model fails to identify the majority of actual clicks. This is likely due to class imbalance or insufficient focus on the minority class during training. Precision is relatively better for Class 1, but this comes at the cost of a very low recall, resulting in a poor F1-score. The training metrics show higher recall and F1-scores for Class 1 compared to the test set, indicating potential overfitting to the training data

for improve the model's performance and mitigated overfitting base on previous result we can have some change in param grid:

Inclusion of criterion: By specifying both Gini impurity and Entropy as possible splitting criteria Gini impurity generally leads to simpler splits and is computationally faster. This allows the model to select the best splitting criterion for your dataset, potentially improving split quality and model performance.

Increasing the max_depth range to include deeper trees ([5, 7, 9, 12]) enables the model to capture more complex relationships in the data. Shallower trees prevent overfitting but may underfit. Deeper trees allow the model to capture finer patterns.

Addition of **min_samples_split:** Including min_samples_split ([2, 5, 10]) ensures that nodes are only split when there are sufficient samples. Smaller values (e.g., 2) allow finer splits, capturing more details in the data. Larger values (e.g., 10) prevent overfitting by avoiding unnecessary splits. This parameter controls tree growth and improves both generalization and computational efficiency.

Addition of **min_samples_leaf**: Including min_samples_leaf ([1, 2, 5]) ensures that leaf nodes have a minimum number of samples. Smaller values capture more details, while larger values

prevent overfitting by enforcing simpler trees. This parameter is crucial for controlling the complexity of the tree and improving robustness.

Handling Class Imbalance with class_weight: Adding class_weight with values like {0: 1, 1: w} for different weights ([1, 2, 3]) helps address class imbalance. By increasing the weight for the minority class (ad_clicked = 1), the model gives more importance to predicting clicks, improving recall for the minority class. This can lead to a better balance between precision and recall for the minority class.

Inclusion of **max_features**: The max_features parameter controls how many features are considered at each split. 'None' allows the model to consider all features, 'sqrt' and 'log2' restrict the features considered, reducing the likelihood of overfitting.

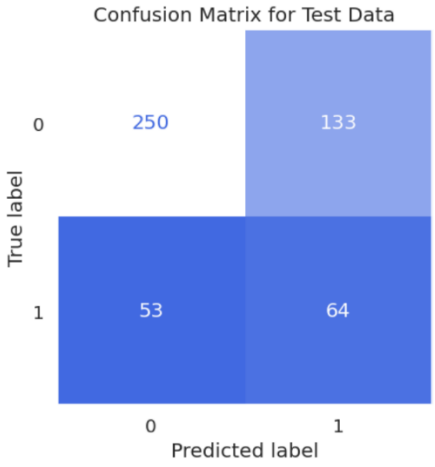
Now we run codes again with these change to see we will have improvement or not

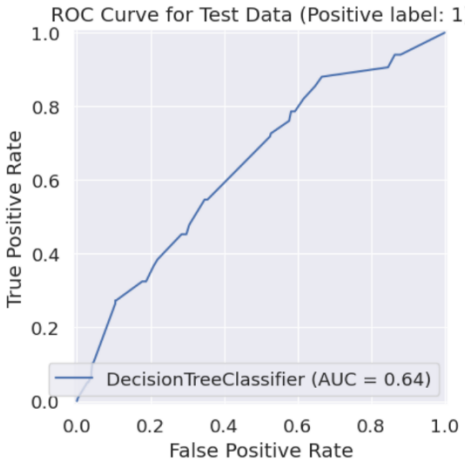
```
# Split the data
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

dt_base = DecisionTreeClassifier(random_state=0)

param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [ 5, 7, 9, 12],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'class_weight': [{0: 1, 1: w} for w in [1, 2, 3]],
    'max_features': [None,'sqrt', 'log2'],
}

best_dt, best_dt_hyperparams = tune_clf_hyperparameters(dt_base, param_grid_dt, X_train, y_train)

print('DT Optimal Hyperparameters: \n', best_dt_hyperparams)
```

```
DT Optimal Hyperparameters:
 {'class_weight': {0: 1, 1: 3}, 'criterion': 'gini', 'max_depth': 7, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

```
model_evaluation(best_dt, X_train, X_test, y_train, y_test, 'Decision Tree')
```

```
         Classification report for training set
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.88      0.69      0.77      1531
           1       0.41      0.69      0.51       466

    accuracy                           0.69      1997
   macro avg       0.64      0.69      0.64      1997
weighted avg       0.77      0.69      0.71      1997


          Classification report for test set
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.65      0.73       383
           1       0.32      0.55      0.41       117

    accuracy                           0.63       500
   macro avg       0.57      0.60      0.57       500
weighted avg       0.71      0.63      0.65       500
```





	Decision Tree
Accuracy	62.8%
Precision (Class 1)	32.49%
Recall (Class 1)	54.7%
F1-score (Class 1)	40.76%
AUC (Class 1)	64.03%

Inferences:

Our **Decision Tree model's** performance on the test data is as follows:

- **Accuracy**: 62.8%, indicating that the model correctly predicts whether a user will click on an advertisement about 63% of the time.
- **Precision (Class 1 - Click)**: 32.49%, meaning that approximately 32% of the predictions where the model identified a user clicking on an advertisement were correct.
- **Recall (Class 1)**: 54.7%, indicating that the model successfully identified about 55% of the actual clicks on advertisements.
- **F1-Score (Class 1)**: 40.76%, which is the harmonic mean of Precision and Recall, reflecting a moderate balance between these two metrics for the minority class.
- **AUC (Area Under the ROC Curve) for Class 1**: 64.03%, showing moderate discriminative ability between users who click and those who do not click on advertisements.

The recall for Class 1 (54.7%) suggests that the model captures a reasonable proportion of actual clicks, which is crucial for tasks prioritizing recall. Precision for Class 1 remains relatively low (32.49%). The F1-score (40.76%) reflects a moderate trade-off between precision and recall, which could be optimized further depending on business priorities. The performance gap between the training and test sets is moderate, indicating that the model generalizes reasonably well without significant overfitting.

In addressing the imbalance in the dataset, we observed that SMOTE is a commonly used technique for handling imbalanced datasets. Although I applied it during this project, it did not lead to a significant improvement in the results. This is likely because the features in our dataset have very low correlation with the target variable. Additionally, techniques like SMOTE are applied only to the training data, meaning they do not directly impact the test results. As a result,

the overall performance metrics showed little change. Considering these factors, I decided not to use SMOTE in this instance, as it did not provide a meaningful advantage for our specific dataset.

Now we can save the model so that it can be compared with other models in the next steps.

```
# Save the final performance of DT
DT_result = metrics_calculator(best_dt, X_test, y_test, 'DT')
DT_result
```

	DT
Accuracy	67.6%
Precision (Class 1)	35.29%
Recall (Class 1)	46.15%
F1-score (Class 1)	40.0%
AUC (Class 1)	62.05%

Random Forest Model Building:

Random Forest is an ensemble learning method designed for both classification and regression tasks. It constructs multiple Decision Trees during the training process and combines their outputs—using averaging for regression or majority voting for classification—to enhance prediction accuracy and robustness. By randomly selecting subsets of features and samples for each tree, Random Forest mitigates overfitting and improves generalization. Additionally, it provides feature importance scores, making it a valuable tool for feature selection. Its versatility lies in its ability to handle large datasets, accommodate missing values, and work effectively with both numerical and categorical data. However, the computational cost can be significant due to the large number of trees involved. (Breiman, 2001)

Advantages:

- **High Accuracy**: Random Forest often provides excellent accuracy by combining the predictions of multiple decision trees, leveraging the ensemble learning technique.
- **Reduces Overfitting**: By averaging or voting across multiple trees, Random Forest reduces the likelihood of overfitting compared to a single decision tree.
- **Handles Both Classification and Regression Tasks**: It is versatile and can be used for a wide range of tasks, including classification and regression problems.
- **Robustness to Noise and Outliers**: Since it relies on multiple decision trees, it is less affected by noise and outliers in the dataset.

Disadvantages:

- **Complexity and Slow Training**: Training multiple decision trees can be computationally expensive and time-consuming, especially for large datasets or when the number of trees is high.

- **Memory Usage**: Random Forest can be memory-intensive since it needs to store a large number of decision trees.

- **Interpretability**: While Random Forest provides feature importance, the overall model is less interpretable compared to single decision trees. It functions more like a "black box."

Random Forest Hyperparameter Tuning:

Optimizing the hyperparameters of a Random Forest classifier can significantly enhance its performance for a specific problem. These hyperparameters govern the model's complexity and behavior, and their configuration greatly influences its accuracy and ability to generalize. For instance, setting the maximum tree depth too high can lead to overfitting, while setting it too low may cause underfitting. Similarly, parameters such as the minimum number of samples required to split a node or the number of features considered at each split also play a critical role. Hyperparameter tuning allows us to identify the optimal combination of values that maximize the model's performance for the task at hand.

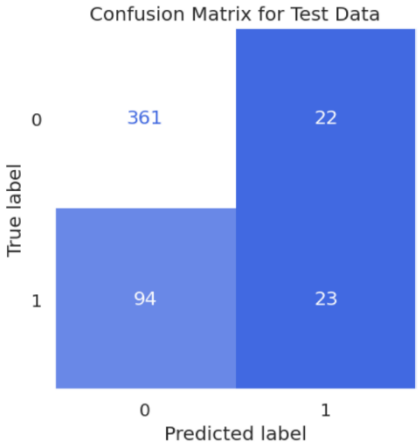
Some hyperparameters for Random Forest Classifiers include:

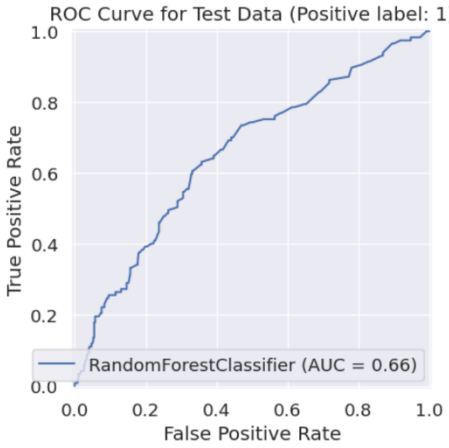
- **n_estimators**: The total number of trees in the Random Forest. Increasing this value can improve performance but also increases computational cost.
- **criterion**: The metric used to evaluate the quality of a split. Common options include "Gini impurity" and "information gain" (entropy).
- **max_depth**: The maximum allowable depth of each tree. This parameter helps control the model’s complexity, preventing overfitting when set appropriately.
- **min_samples_split**: The minimum number of samples required to split an internal node. Higher values enforce more conservative splits and reduce overfitting.
- **min_samples_leaf**: The minimum number of samples that must be present at a leaf node. Larger values can smooth predictions by preventing overly specific splits.
- **class_weight**: Specifies the relative importance (weights) assigned to each class, which is useful for handling imbalanced datasets.
- **max_features**: The maximum number of features to consider when determining the best split at a node. It can be set as a specific number, a float (representing a percentage), or options like 'sqrt' or 'log2' to adjust the trade-off between model complexity and computational efficiency. (scikit-learn, n.d.)

First we just use n_estimators for param grid and in next step try to improve the result

```python
# Split the data
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

rf_base = RandomForestClassifier(random_state=0)

# Define the hyperparameter grid to search
param_grid_rf = {
    'n_estimators': [50, 100, 150],
}

best_rf, best_rf_hyperparams = tune_clf_hyperparameters(rf_base, param_grid_rf, X_train, y_train)

print('Random Forest Optimal Hyperparameters: \n', best_rf_hyperparams)
model_evaluation(best_rf, X_train, X_test, y_train, y_test, 'Random Forest')
```

```
          Classification report for training set
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1531
           1       0.97      0.96      0.97       466

    accuracy                           0.98      1997
   macro avg       0.98      0.98      0.98      1997
weighted avg       0.98      0.98      0.98      1997


          Classification report for test set
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.94      0.86       383
           1       0.51      0.20      0.28       117

    accuracy                           0.77       500
   macro avg       0.65      0.57      0.57       500
weighted avg       0.73      0.77      0.73       500
```





	Random Forest
Accuracy	76.8%
Precision (Class 1)	51.11%
Recall (Class 1)	19.66%
F1-score (Class 1)	28.4%
AUC (Class 1)	65.58%

Inferences:

The Random Forest model's performance on the test data is as follows:

- **Accuracy**: 76.8%, indicating that the model correctly predicts whether a user will click on an advertisement nearly 77% of the time.
- **Precision (Class 1 - Click)**: 51.11%, meaning that of all the predictions where the model identified a user clicking on an advertisement, only 51.11% were correct.
- **Recall (Class 1)**: 19.66%, meaning that the model correctly identified approximately 19.66% of the actual clicks on advertisements.
- **F1-score (Class 1)**: 28.4%, which is the harmonic mean of Precision and Recall, reflecting a balance between these two metrics for class 1.
- **AUC (Area Under the ROC Curve) for Class 1**: 65.58%, indicating moderate discriminative ability between users who click and do not click on advertisements.

The **confusion matrix** reveals significant challenges, with many **False Positives** (non-clicks predicted as clicks) and **False Negatives** (actual clicks missed by the model). The **Recall for Class 1** is particularly low (19.66%), indicating that the model struggles to identify the majority of actual clicks on advertisements. Despite the model's good overall accuracy, the imbalance in class performance indicates it needs better tuning for the minority class (Class 1 - Click).

Now we try to have better accuracy and f1-score with some changes in param grid:

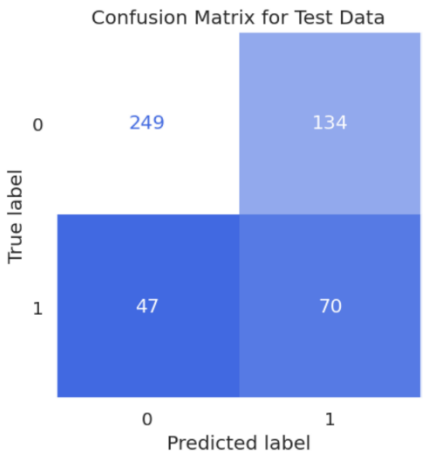
By introducing parameters such as max_depth, min_samples_split, and min_samples_leaf, the updated model reduced the risk of overfitting by forcing the trees to be simpler and better generalized to unseen data. Incorporating class_weight allowed the model to focus more on the minority class (ad_clicked = 1), leading to better recall and F1-score for the minority class. The inclusion of max_features helped each tree use a subset of features, increasing the diversity among the ensemble trees and reducing correlation between them, which often results in better generalization.

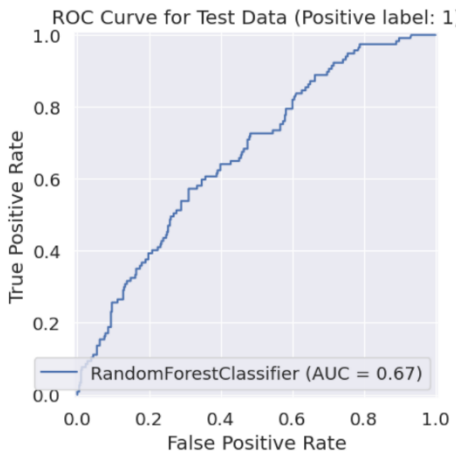
Now run the code again with some changes

```
# Split the data
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

# Define the base Random Forest model
rf_base = RandomForestClassifier(random_state=0)

# Define the hyperparameter grid to search
param_grid_rf = {
    'n_estimators': [50, 100, 150],
    'criterion': ['entropy', 'gini'],
    'max_depth': [10, 12, 14],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [5, 10],
    'max_features': ['sqrt', 'log2'],
    'class_weight': [{0: 1, 1: w} for w in [2, 3, 4]],
}

# Call the function for hyperparameter tuning
best_rf, best_rf_hyperparams = tune_clf_hyperparameters(rf_base, param_grid_rf, X_train, y_train)

# Print the optimal hyperparameters for the Random Forest model
print('Random Forest Optimal Hyperparameters: \n', best_rf_hyperparams)

# Evaluate the best model on the training and test set
model_evaluation(best_rf, X_train, X_test, y_train, y_test, 'Random Forest')
```

```
Random Forest Optimal Hyperparameters:
 {'class_weight': {0: 1, 1: 4}, 'criterion': 'gini', 'max_depth': 14, 'max_features': 'sqrt', 'min_samples_leaf': 10, 'min_samples_split': 5, 'n_estimators': 50}
```

```
         Classification report for training set
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.93      0.69      0.79      1531
           1       0.44      0.82      0.57       466

    accuracy                           0.72      1997
   macro avg       0.68      0.75      0.68      1997
weighted avg       0.81      0.72      0.74      1997


          Classification report for test set
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.65      0.73       383
           1       0.34      0.60      0.44       117

    accuracy                           0.64       500
   macro avg       0.59      0.62      0.58       500
weighted avg       0.72      0.64      0.66       500
```





	Random Forest
Accuracy	63.8%
Precision (Class 1)	34.31%
Recall (Class 1)	59.83%
F1-score (Class 1)	43.61%
AUC (Class 1)	66.77%

Inferences:

Our Random Forest model's performance on the test data is as follows:

- **Accuracy:** The model achieves an accuracy of **63.8%**, indicating that it correctly predicts whether a user will click on an advertisement approximately 64% of the time.

- **Precision (Class 1 - Click):** Precision for users who clicked on advertisements is **34.31%**, meaning that out of all the predictions where the model identified a user as having clicked, only 34% were correct.

- **Recall (Class 1 - Click):** Recall for users who clicked is **59.83%**, which shows that the model successfully identified nearly 60% of actual clicks on advertisements.

- **F1-score (Class 1 - Click):** The F1-score for class 1 is **43.61%**, representing a balance between precision and recall. This score indicates that while the recall is reasonable, the precision can be improved further.

- **AUC (Area Under the ROC Curve):** The AUC for class 1 is **66.77%**, suggesting moderate discriminative ability between users who click and do not click on advertisements.

The precision for class 1 (users who clicked) is relatively low, indicating a higher number of false positives. While the recall is better, capturing nearly 60% of actual clicks, improving precision could lead to a better F1-score and more reliable predictions. There is no significant evidence of overfitting since the gap between training and test performance is moderate.

```
# Save the final performance of random Forest
rf_result = metrics_calculator(best_rf, X_test, y_test, 'Random Forest')
rf_result
```

	Random Forest
Accuracy	63.8%
Precision (Class 1)	34.31%
Recall (Class 1)	59.83%
F1-score (Class 1)	43.61%
AUC (Class 1)	66.77%

XGBoost Model Building:

XGBoost, short for eXtreme Gradient Boosting, is a highly optimized implementation of the gradient boosting algorithm for tree-based machine learning models, designed for speed and memory efficiency. It stands out from other gradient boosting implementations with unique features such as handling missing values, enabling parallel processing for faster training and prediction, tree pruning to reduce overfitting, and regularization techniques to enhance model generalization and prevent overfitting.

XGBoost Hyperparameter Tuning:

XGBoost has several hyperparameters that require careful tuning to enhance the model's performance. Below are some of the key hyperparameters:

- **n_estimators**: This defines the number of boosting rounds or trees to be built. It is generally set to a high value, but XGBoost includes an early stopping feature to terminate the process if no further improvements are observed.
- **learning_rate**: This parameter controls the step size shrinkage to prevent overfitting. Its value ranges from 0 to 1.
- **max_depth**: This specifies the maximum depth of a tree and determines the complexity of the model. It can range from 1 to infinity.
- **subsample**: This represents the fraction of observations that are randomly sampled for each tree. The value ranges from 0 to 1.
- **colsample_bytree**: This parameter specifies the fraction of features randomly sampled for each tree, with values ranging from 0 to 1.

We will tune these hyperparameters using the tune_clf_hyperparameters function, which performs a grid search across the specified parameter grid and returns the best-performing model.

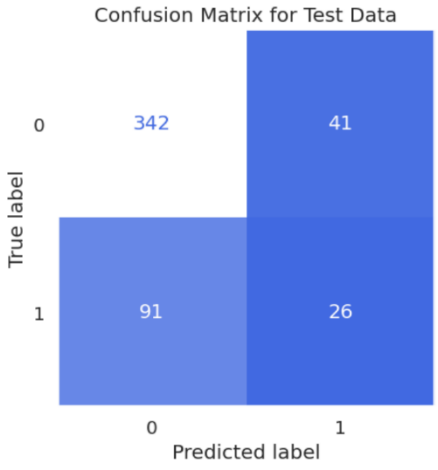
Additionally, XGBoost provides a built-in function, xgb.cv, that can be used to determine the optimal number of boosting rounds (n_estimators). This function trains the model multiple times on different data subsets and identifies the number of boosting rounds that minimizes the error. However, since we are already tuning n_estimators as part of the grid search, using xgb.cv is unnecessary in this scenario. First I use this model with just n_estimators.

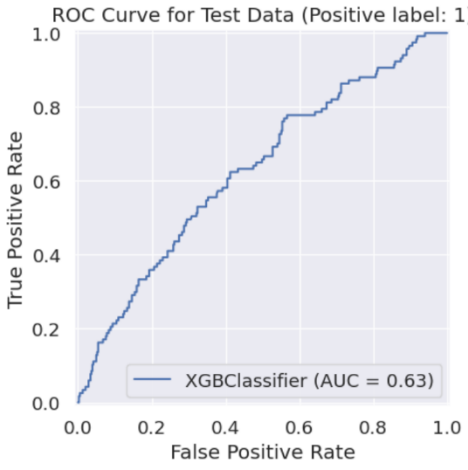
```python
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

# Define the model
xgb_base = xgb.XGBClassifier(random_state=0)

# Define the parameter grid
param_grid_xgb = {
    'n_estimators': [100 ,150, 200],
}

# Call the function for hyperparameter tuning
best_xgb, best_xgb_hyperparams = tune_clf_hyperparameters(xgb_base, param_grid_xgb, X_train, y_train)

print('XGBoost Optimal Hyperparameters: \n', best_xgb_hyperparams)
```

```
XGBoost Optimal Hyperparameters:
 {'n_estimators': 200}
```

```python
model_evaluation(best_xgb, X_train, X_test, y_train, y_test, 'XGBoost')
```

```
        Classification report for training set
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1531
           1       0.97      0.94      0.95       466

    accuracy                           0.98      1997
   macro avg       0.98      0.96      0.97      1997
weighted avg       0.98      0.98      0.98      1997


         Classification report for test set
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       383
           1       0.39      0.22      0.28       117

    accuracy                           0.74       500
   macro avg       0.59      0.56      0.56       500
weighted avg       0.70      0.74      0.71       500
```





	XGBoost
Accuracy	73.6%
Precision (Class 1)	38.81%
Recall (Class 1)	22.22%
F1-score (Class 1)	28.26%
AUC (Class 1)	62.88%

Inferences:

Our **XGBoost model's** performance on the test data is as follows:

- **Accuracy**: 73.6%, indicating that the model correctly predicts whether a user will click on an advertisement about 74% of the time.
- **Precision for Class 1 (Clicks)**: 38.81%, meaning that of all the cases where the model predicted a click, only about 39% were correct.
- **Recall for Class 1**: 22.22%, meaning that the model successfully identified only 22% of the actual clicks on advertisements.
- **F1-Score for Class 1**: 28.26%, which represents the harmonic mean of precision and recall, reflecting a significant trade-off between these metrics for the minority class.
- **AUC (Area Under the ROC Curve) for Class 1**: 62.88%, indicating moderate discriminative ability between users who click and those who do not.

The training metrics are significantly higher than the test metrics, suggesting overfitting to the training data. For instance, the F1-score for Class 1 in the training set is 95%, while in the test set, it is only 28.26%. The model has a relatively higher precision for Class 1 but struggles with recall, missing many actual clicks. The model's performance indicates that it struggles to generalize to unseen data, particularly for the minority class (Class 1).

We focus on param grid we try to increase our result.

The learning rate controls the step size at which the model learns from the data. By including learning_rate in the hyperparameter grid ([0.01, 0.05]), the model can balance between convergence speed and model performance.

The maximum depth of trees ([4, 5, 6]) determines how complex the decision trees in XGBoost can be:

- Shallower trees (e.g., 4) reduce the risk of overfitting by limiting the model’s capacity to memorize noise.
- Deeper trees (e.g., 6) allow the model to capture more complex patterns.

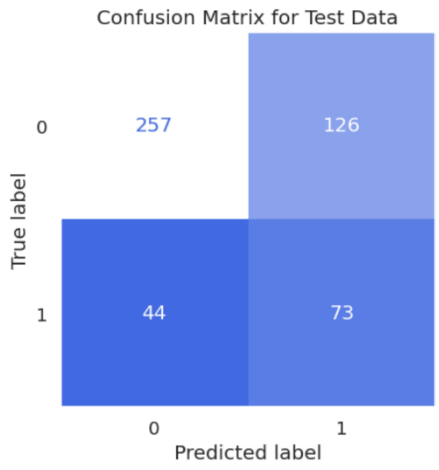
The scale_pos_weight parameter addresses class imbalance by weighting the positive class more heavily during training. By including values [2, 3], the model could compensate for the imbalance in ad_clicked predictions, improving the ability to detect clicks (minority class).

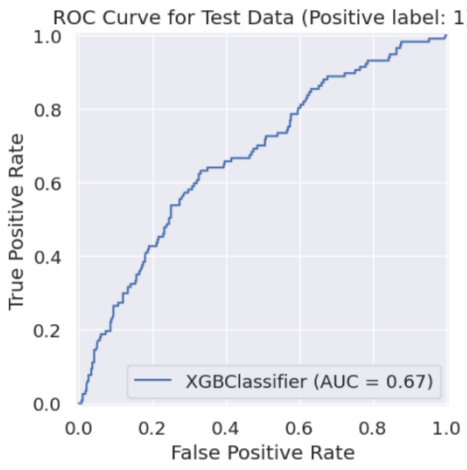
```python
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

# Define the model
xgb_base = xgb.XGBClassifier(random_state=0)

# Define the parameter grid
param_grid_xgb = {
    'n_estimators': [100,150,200],
    'learning_rate': [0.01, 0.05],
    'max_depth': [4, 5, 6],
    'scale_pos_weight': [2, 3],
}

# Call the function for hyperparameter tuning
best_xgb, best_xgb_hyperparams = tune_clf_hyperparameters(xgb_base, param_grid_xgb, X_train, y_train)

print('XGBoost Optimal Hyperparameters: \n', best_xgb_hyperparams)
```

```
XGBoost Optimal Hyperparameters:
 {'learning_rate': 0.05, 'max_depth': 4, 'n_estimators': 100, 'scale_pos_weight': 3}
```

```python
model_evaluation(best_xgb, X_train, X_test, y_train, y_test, 'XGBoost')
```

```
          Classification report for training set
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.72      0.80      1531
           1       0.45      0.75      0.56       466

    accuracy                           0.73      1997
   macro avg       0.68      0.74      0.68      1997
weighted avg       0.80      0.73      0.75      1997


          Classification report for test set
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.85      0.67      0.75       383
           1       0.37      0.62      0.46       117

    accuracy                           0.66       500
   macro avg       0.61      0.65      0.61       500
weighted avg       0.74      0.66      0.68       500
```





	XGBoost
Accuracy	66.0%
Precision (Class 1)	36.68%
Recall (Class 1)	62.39%
F1-score (Class 1)	46.2%
AUC (Class 1)	67.25%

Inferences:

Our **XGBoost model's** performance on the test data is as follows:

- **Accuracy**: 66.0%, indicating that the model correctly predicts whether a user will click on an advertisement about 66% of the time.
- **Precision for Class 1 (Clicks)**: 36.68%, meaning that about 37% of the predictions for users clicking on advertisements were correct.
- **Recall for Class 1**: 62.39%, showing that the model successfully identified approximately 62% of the actual clicks on advertisements.
- **F1-Score for Class 1**: 46.2%, which is the harmonic mean of precision and recall, reflecting a moderate balance between these two metrics for the minority class.
- **AUC (Area Under the ROC Curve) for Class 1**: 67.25%, indicating a moderate discriminative ability between users who click and those who do not.

Improvements in Recall: The recall for Class 1 (62.39%) suggests that the model captures a higher proportion of actual clicks compared to previous models, though precision remains relatively low.

Overfitting Check: The training performance metrics (e.g., F1-score of 56% for Class 1) and the test performance metrics (e.g., F1-score of 46.2% for Class 1) are reasonably close, indicating minimal overfitting.

Trade-off Between Precision and Recall: The model prioritizes recall over precision for Class 1, which may lead to higher false positives (non-clicks predicted as clicks). This can be useful when capturing as many true clicks as possible is more important than precision.

```
# Save the final performance of LR
XGBoost_result = metrics_calculator(best_xgb, X_test, y_test, 'XGBoost')
XGBoost_result
```

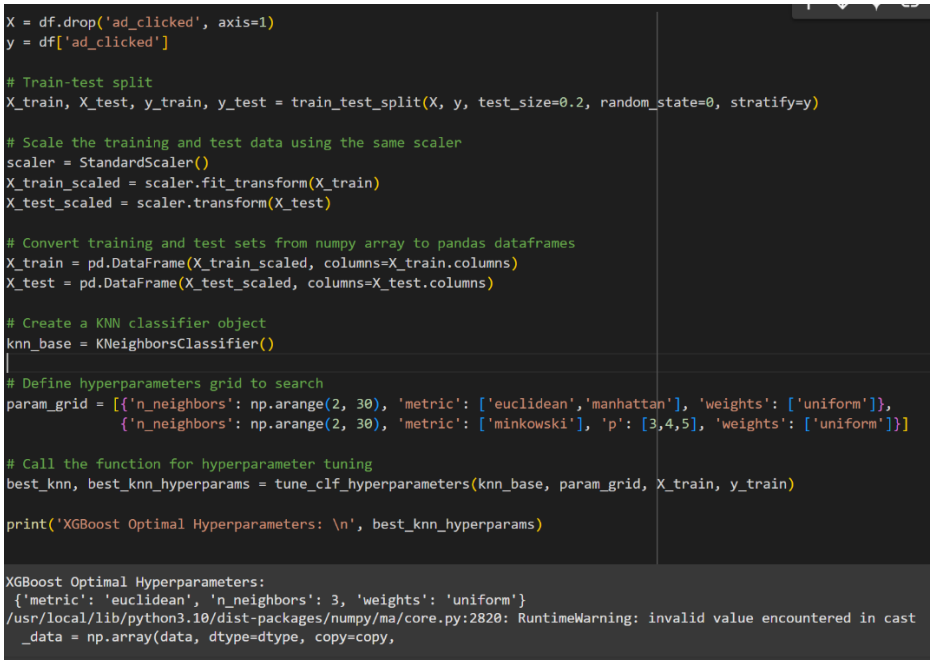
	XGBoost
Accuracy	66.0%
Precision (Class 1)	36.68%
Recall (Class 1)	62.39%
F1-score (Class 1)	46.2%
AUC (Class 1)	67.25%

KNN Model Building

In the previous section, we established that tree-based models are better suited for handling imbalanced datasets. However, to gain deeper insights and ensure a thorough understanding, I decided to experiment with another type of model to compare their performance and observe the differences. K-Nearest Neighbors (KNN) is a supervised learning algorithm commonly used for both classification and regression tasks in machine learning. It predicts the class or value of a test sample by identifying the K closest data points in the training set and using their majority class (in classification) or average value (in regression) for the prediction.

For this model we have to use standard scaling because it uses a distance metric.

For the hyperparameters, I want to use n_neighbors, which specifies the number of nearest neighbors to consider when predicting the class of a new sample. The weights parameter determines how distances between neighbors influence the prediction. With uniform, all neighbors contribute equally, while with distance, closer neighbors have a greater impact. The metric parameter specifies the distance metric used to calculate the nearest neighbors, with common options being Euclidean (straight-line distance), Manhattan (distance along axes at right angles), and Minkowski (a generalized metric encompassing both Euclidean and Manhattan). The p parameter in the Minkowski metric adjusts the distance calculation, where p=1 corresponds to Manhattan distance, p=2 to Euclidean distance, and other values of p offer flexibility by weighting distances differently.

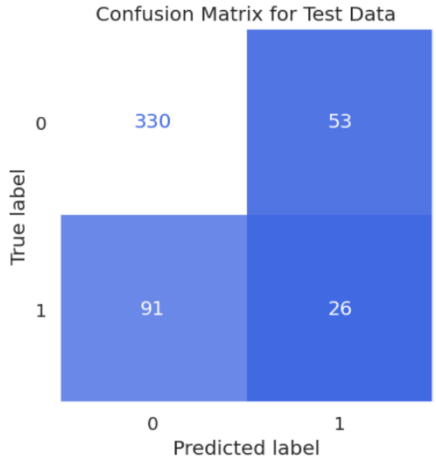


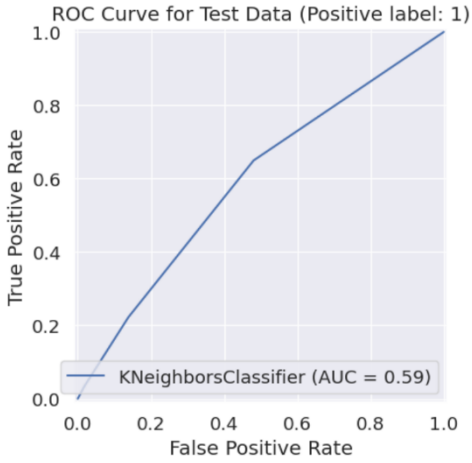
```
X = df.drop('ad_clicked', axis=1)
y = df['ad_clicked']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)

# Scale the training and test data using the same scaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert training and test sets from numpy array to pandas dataframes
X_train = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Create a KNN classifier object
knn_base = KNeighborsClassifier()

# Define hyperparameters grid to search
param_grid = [{'n_neighbors': np.arange(2, 30), 'metric': ['euclidean','manhattan'], 'weights': ['uniform']},
              {'n_neighbors': np.arange(2, 30), 'metric': ['minkowski'], 'p': [3,4,5], 'weights': ['uniform']}]

# Call the function for hyperparameter tuning
best_knn, best_knn_hyperparams = tune_clf_hyperparameters(knn_base, param_grid, X_train, y_train)

print('XGBoost Optimal Hyperparameters: \n', best_knn_hyperparams)
```

```
XGBoost Optimal Hyperparameters:
 {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

```
model_evaluation(best_knn, X_train, X_test, y_train, y_test, 'knn')
```

```
         Classification report for training set
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.87      0.94      0.90      1531
           1       0.71      0.52      0.60       466

    accuracy                           0.84      1997
   macro avg       0.79      0.73      0.75      1997
weighted avg       0.83      0.84      0.83      1997


          Classification report for test set
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       383
           1       0.33      0.22      0.27       117

    accuracy                           0.71       500
   macro avg       0.56      0.54      0.54       500
weighted avg       0.68      0.71      0.69       500
```





	knn
Accuracy	71.2%
Precision (Class 1)	32.91%
Recall (Class 1)	22.22%
F1-score (Class 1)	26.53%
AUC (Class 1)	59.33%

As we can see here, even though we used uniform to handle the imbalanced dataset, the F1-score is significantly lower compared to tree-based algorithms. Therefore, this model is not suitable for achieving our goal. But we save the model for compare to other models.

```
# Save the final performance of knn
knn_result = metrics_calculator(best_knn, X_test, y_test, 'knn')
knn_result
```

	knn
Accuracy	71.2%
Precision (Class 1)	32.91%
Recall (Class 1)	22.22%
F1-score (Class 1)	26.53%
AUC (Class 1)	59.33%

As previously discussed, the most critical metric for this project is the F1-score for class '1' (ad clicks). A high F1-score reflects a balance between capturing as many users likely to click on ads (high recall) and minimizing false positives (high precision).

This code provides a clearer and more comprehensive understanding of our models' performance.

```
# Concatenate previous classifiers performance results into a single dataframe
results = pd.concat([LR_result,DT_result,xgb_result,rf_result,ada_result], axis=1).T

# Reset the index to ensure unique index values
results.reset_index(inplace=True) # Added this line to reset the index

#Check for and drop any duplicate columns
results = results.loc[:,~results.columns.duplicated()] #Added this line to remove duplicate columns if any

# Rename the index column to 'Model' or any other desired name
results.index.name = 'Model' # Added this line to rename the index column


# Sort the dataframe in descending order based on F1-score (class 1) values
results.sort_values(by='F1-score (Class 1)', ascending=False, inplace=True)

# Color the F1-score column
results.style.applymap(lambda x: 'background-color: royalblue', subset='F1-score (Class 1)')
```

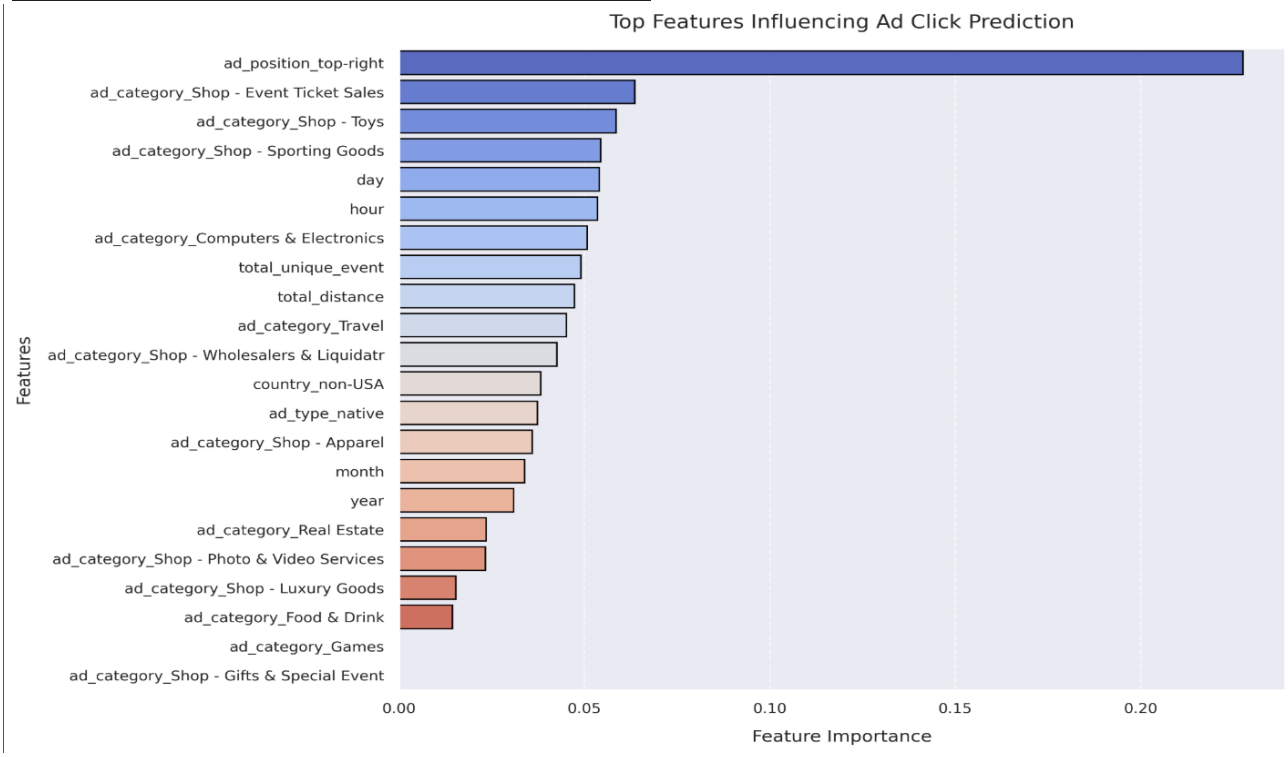
Model	index	Accuracy	Precision (Class 1)	Recall (Class 1)	F1-score (Class 1)	AUC (Class 1)
1	XGBoost	66.0%	36.68%	62.39%	46.2%	67.25%
2	Random Forest	63.8%	34.31%	59.83%	43.61%	66.77%
0	Decision Tree	62.8%	32.49%	54.7%	40.76%	64.03%
3	knn	71.2%	32.91%	22.22%	26.53%	59.33%

XGBoost is the best-performing model for this task, as it balances precision, recall, and F1-score for class '1', which is crucial for predicting ad clicks. KNN is unsuitable for this problem due to its inability to handle class imbalance and poor performance on key metrics.

XGBoost Feature Importance

XGBoost, as a tree-based model, inherently computes feature importances, providing a quantitative measure of how each feature contributes to predicting the target variable. A higher importance score indicates that the feature is more relevant and influential in determining the output. This capability is valuable for both feature selection and model optimization, enabling the identification of critical features and the potential elimination of less impactful ones to enhance model efficiency and performance.

```python
# Compute feature importances
feature_importances = best_xgb.feature_importances_

# Sort the feature importances in descending order and get the indices
sorted_indices = np.argsort(feature_importances)[::-1]

# Create a DataFrame for better control over feature names and values
importance_df = pd.DataFrame({
    'Feature': X.columns[sorted_indices],
    'Importance': feature_importances[sorted_indices]
})

# Plot feature importances with enhancements
plt.figure(figsize=(12, 8), dpi=300)
sns.barplot(x='Importance',y='Feature', data=importance_df, palette='coolwarm',
    edgecolor='black'
)

# Add labels and title
plt.xlabel('Feature Importance', fontsize=12, labelpad=10)
plt.ylabel('Features', fontsize=12, labelpad=10)
plt.title('Top Features Influencing Ad Click Prediction', fontsize=14, pad=15)

# Improve tick appearance
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

# Add grid for clarity
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Display the plot
plt.tight_layout()
plt.show()
```



The XGBoost model has identified that **ad position, ad categories** such as event ticket sales, toys, and sporting goods, along with features like the day, hour, and user interactions (e.g., total unique events and total distance), are the most influential factors in predicting whether a user will click on an advertisement. This highlights that **ad placement and ad category** webpage play a significant role in determining ad click behavior. Understanding these insights can help optimize ad targeting strategies to improve user engagement.

Conclusions and recommendations:

Base on the result of EDA and Feature Importance:

ad position is the most important feature for click on advertisement and it can have two reason:

1. For the top-right position, the scroll bar is typically located on the right side of most websites. This naturally guides the user's mouse movement and visual focus toward the right side of the webpage. As a result, the top-right position can be an effective placement for advertisements, leveraging this unconscious behavior to capture attention.
2. The top-left position of a website often contains essential elements like the menu or navigation bar. This means users are likely to move their mouse toward this area, making it a prime location for placing advertisements to capture attention effectively.

Perhaps due to well-placed and relevant advertisements on the website, users may unintentionally click on them. Once redirected, they might decide to explore the site further, potentially leading to a purchase or spending more time browsing.

The second most important feature is the ad category. It appears that the most popular categories users prefer to click on are those related to entertainment, such as travel, event tickets, and items appealing to children, like toys and electronics for younger user. This suggests that our website attracts a significant number of younger visitors or individuals interested in entertainment and leisure activities.

Most users click on advertisements between 3 PM and 6 PM, making this time frame crucial for considering advertisement placement on the website. Additionally, the first days of each month are particularly significant for ad engagement. The website also experienced the highest number of advertisement clicks during January, February, and October.

References

Anon., n.d. [Online]
Available at: https://chatgpt.com/
[Accessed 11 12 2024].

Anon., n.d. *chatgot.* [Online]
Available at: https://chatgpt.com/
[Accessed 11 12 2024].

Breiman, L. ". F., 2001. *stat.berkeley..* [Online]
Available at: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf
[Accessed 11 12 2024].

Quinlan, J., 1985. *hunch.* [Online]
Available at: https://hunch.net/~coms-4771/quinlan.pdf
[Accessed 11 12 2024].

scikit-learn., n.d. *scikit-learn..* [Online]
Available at: https://scikit-learn.org/1.5/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[Accessed 11 12 2024].

scikit-learn, n.d. *scikit-learn.* [Online]
Available at: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[Accessed 11 12 2024].

Shahules, 2019. *kaggle..* [Online]
Available at: https://www.kaggle.com/code/shahules/an-overview-of-encoding-techniques
[Accessed 11 12 2024].

SWASTIK, n.d. *analyticsvidhya.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/
[Accessed 11 12 2024].