AD CAMPAIGN RECOMMENDER-MODEL BUILDING

FROM –DEEPAK SINGH PANWAR

1. READING DATA

1. Reading Data

a. Reading data from S3 in EC2

In [3]:
```
train_mobile_brand = pd.read_csv("s3://adcampaignrecommenderdeepaksinghpanwar/train_mobile_brand.csv")
train_mobile_brand.head()
```

Out[3]:

	device_id	gender	age	group_train	phone_brand	device_model
0	-7548291590301750000	M	33	M32+	Huawei	è£è€€3C
1	6943568600617760000	M	37	M32+	Xiaomi	xnote
2	5441349705980020000	M	40	M32+	OPPO	R7s
3	-5393876656119450000	M	33	M32+	Xiaomi	MI 4
4	4543988487649880000	M	53	M32+	samsung	Galaxy S4

In [4]:
```
df_events_org = pd.read_csv("s3://adcampaignrecommenderdeepaksinghpanwar/train_event_data.csv",dtype={'device_id': np.str, 'latit
df_events_org.head(n=100)
```

Out[4]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude
0	-7548291590301750000	M	33	M32+	2369465.0	2016-05-03 15:55:35	33.98	116.79
1	-7548291590301750000	M	33	M32+	1080869.0	2016-05-03 06:07:16	33.98	116.79
2	-7548291590301750000	M	33	M32+	1079338.0	2016-05-04 03:28:02	33.98	116.79
3	-7548291590301750000	M	33	M32+	1078881.0	2016-05-04 02:53:08	33.98	116.79
4	-7548291590301750000	M	33	M32+	1068711.0	2016-05-03 15:59:35	33.98	116.79

In [5]:
```
app_events = pd.read_csv("s3://adcampaignrecommenderdeepaksinghpanwar/app_events.csv")
app_events.head()
```

Out[5]:

	event_id	app_id	is_installed	is_active
0	2	5927333115845830913	1	1
1	2	-5720078949152207372	1	0
2	2	-1633887856876571208	1	0
3	2	-653184325010919369	1	1
4	2	8693964245073640147	1	1

In [6]:
```
app_events_meta_data = pd.read_csv("s3://adcampaignrecommenderdeepaksinghpanwar/app_events_meta_data.csv")
app_events_meta_data.head()
```

Out[6]:

	app_id	label_id	category
0	7324880000000000000.0	251.0	Finance
1	-4494220000000000000.0	251.0	Finance
2	6058200000000000000.0	406.0	unknown
3	6058200000000000000.0	407.0	DS_P2P net loan
4	8694630000000000000.0	406.0	unknown

2. CLEANING DATA

2. Data Cleaning

a. Example - Geospatial Data (Lat and Long)

In [6]:
```
df_events = df_events_org[df_events_org['event_id'].notnull()]
df_events.head()
```

Out[6]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude
0	-7548291590301750000	M	33	M32+	2369465.0	2016-05-03 15:55:35	33.98	116.79
1	-7548291590301750000	M	33	M32+	1080869.0	2016-05-03 06:07:16	33.98	116.79
2	-7548291590301750000	M	33	M32+	1079338.0	2016-05-04 03:28:02	33.98	116.79
3	-7548291590301750000	M	33	M32+	1078881.0	2016-05-04 02:53:08	33.98	116.79
4	-7548291590301750000	M	33	M32+	1068711.0	2016-05-03 15:59:35	33.98	116.79

b. Cleaning of Other data requried

Check for missing values: Use the isnull() method to check if there are any missing values in the DataFrame. If there are, you can decide whether to drop the rows with missing values or fill in the missing values with an appropriate value.

In [7]:
```
# Check for missing values
print(app_events.isnull().sum())

# Drop rows with missing values
app_events.dropna(inplace=True)

# Fill missing values with a specific value
app_events.fillna(value=0, inplace=True)
```

```
event_id        0
app_id          0
is_installed    0
is_active       0
dtype: int64
```

Check for duplicates: Use the duplicated() method to check if there are any duplicate rows in the DataFrame. If there are, you can decide whether to drop the duplicate rows or keep only the first occurrence of each unique row.

In [8]:
```
# Check for duplicates
print(app_events.duplicated().sum())

# Drop duplicate rows
app_events.drop_duplicates(inplace=True)

# Keep only the first occurrence of each unique row
app_events.drop_duplicates(keep='first', inplace=True)
```

```
0
```

2. CLEANING DATA

Convert data types: Check the data types of each column in the DataFrame using the dtypes attribute. If any columns have the wrong data type, use the astype() method to convert them to the correct data type.

In [9]:

```
# Check data types
print(app_events.dtypes)

# Convert a column to a different data type
app_events['event_id'] = app_events['event_id'].astype(str)
app_events['is_active'] = app_events['is_active'].astype(bool)
```

```
event_id        int64
app_id          int64
is_installed    int64
is_active       int64
dtype: object
```

Remove outliers: Use the describe() method to get a summary of the numerical columns in the DataFrame. If there are any outliers, you can remove them using a suitable method such as Z-score or Interquartile Range (IQR)

In [10]:

```
# Get summary statistics
print(app_events.describe())
```

```
             app_id  is_installed
count  3.247307e+07    32473067.0
mean   1.182779e+18           1.0
std    5.360173e+18           0.0
min   -9.221157e+18           1.0
25%   -3.474568e+18           1.0
50%    1.387044e+18           1.0
75%    6.043001e+18           1.0
max    9.222488e+18           1.0
```

In [11]:

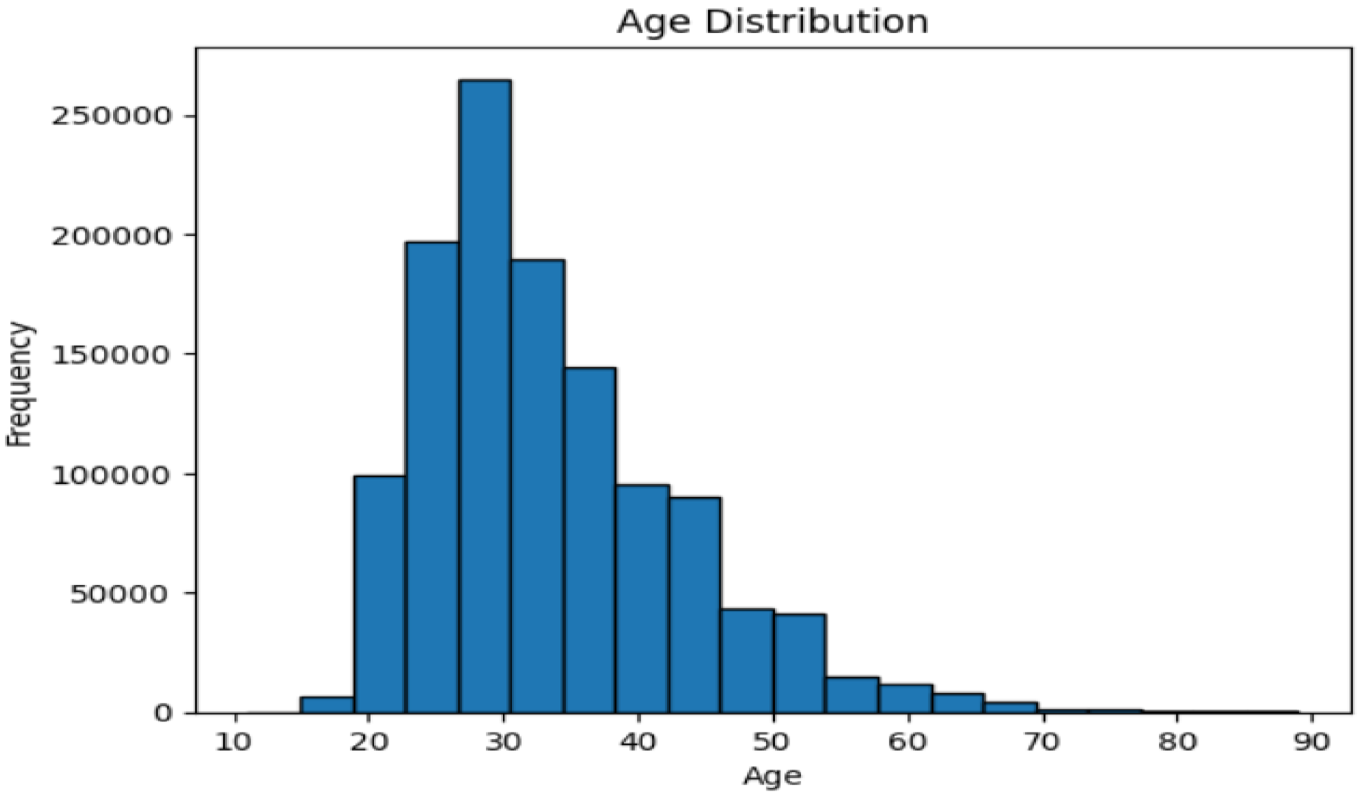
```
# Convert the date column to a datetime format
df_events['date'] = pd.to_datetime(df_events['datetimestamp'])
```

3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 01. Age and Gender Distribution:
- a. Plot appropriate graphs that represent the distribution of age and gender in the dataset[Univariate]

In [12]:

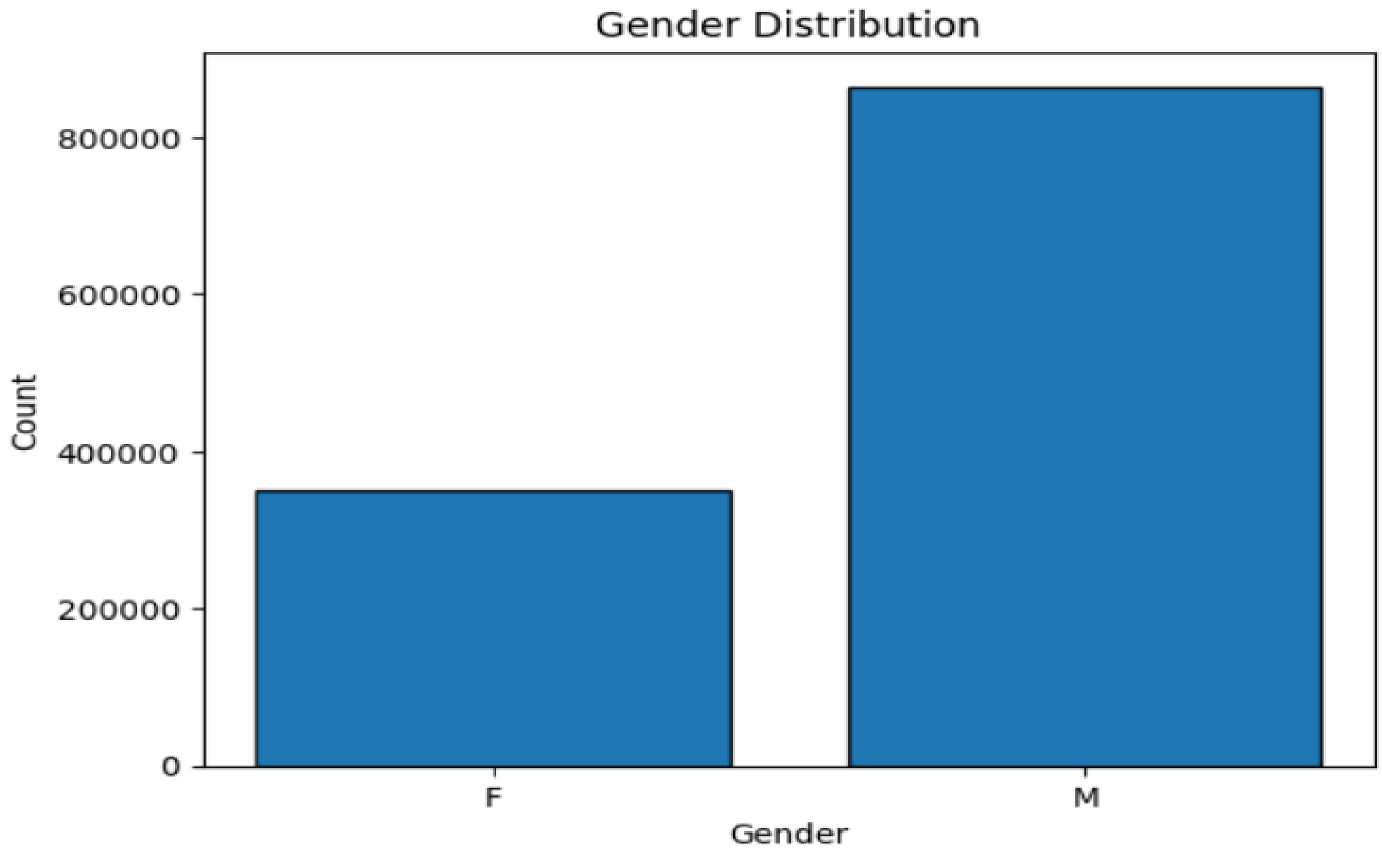
```
# Plot histogram of age
plt.hist(df_events.age, bins=20, edgecolor='black')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
```



3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

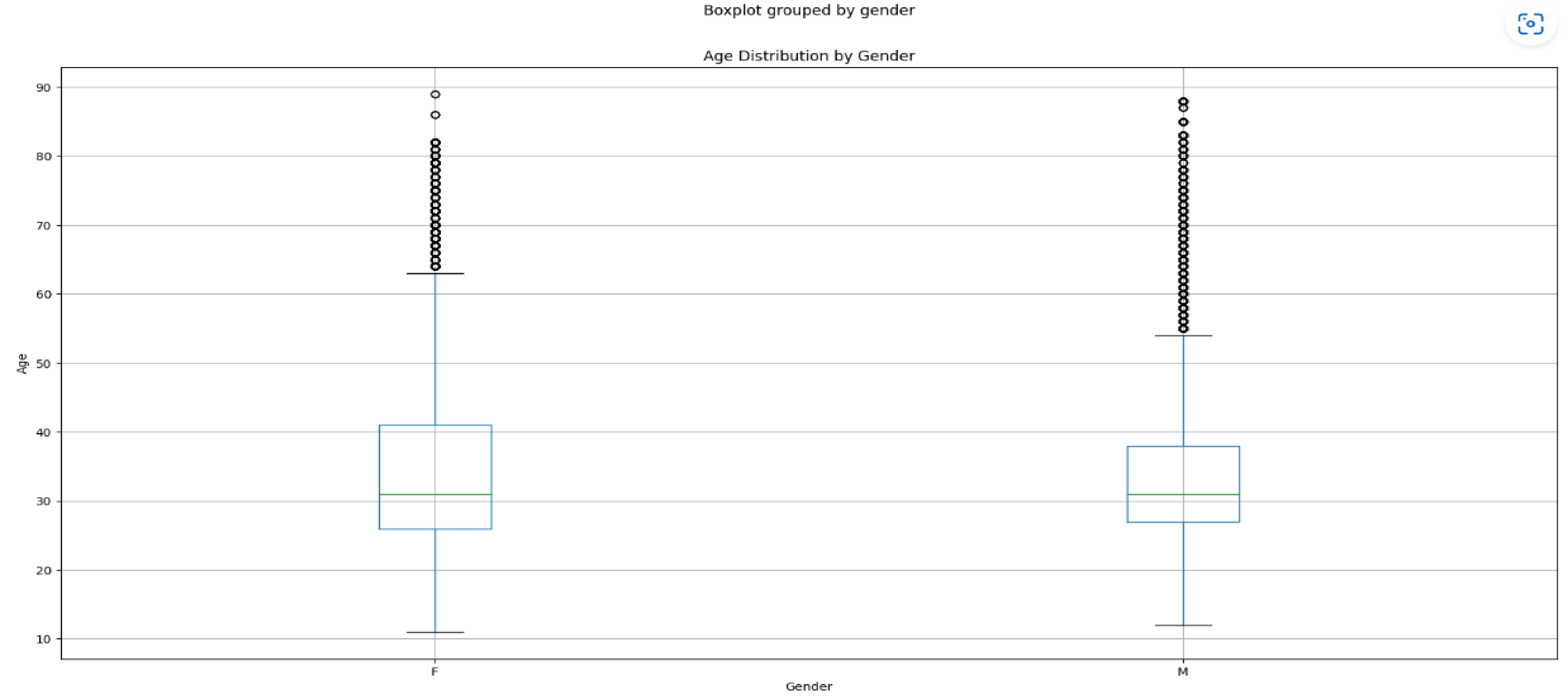
- Gender Distribution

In [13]:
```
# Plot bar chart of gender
gender_counts = np.unique(df_events.gender, return_counts=True)
plt.bar(gender_counts[0], gender_counts[1], edgecolor='black')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Distribution')
plt.show()
```



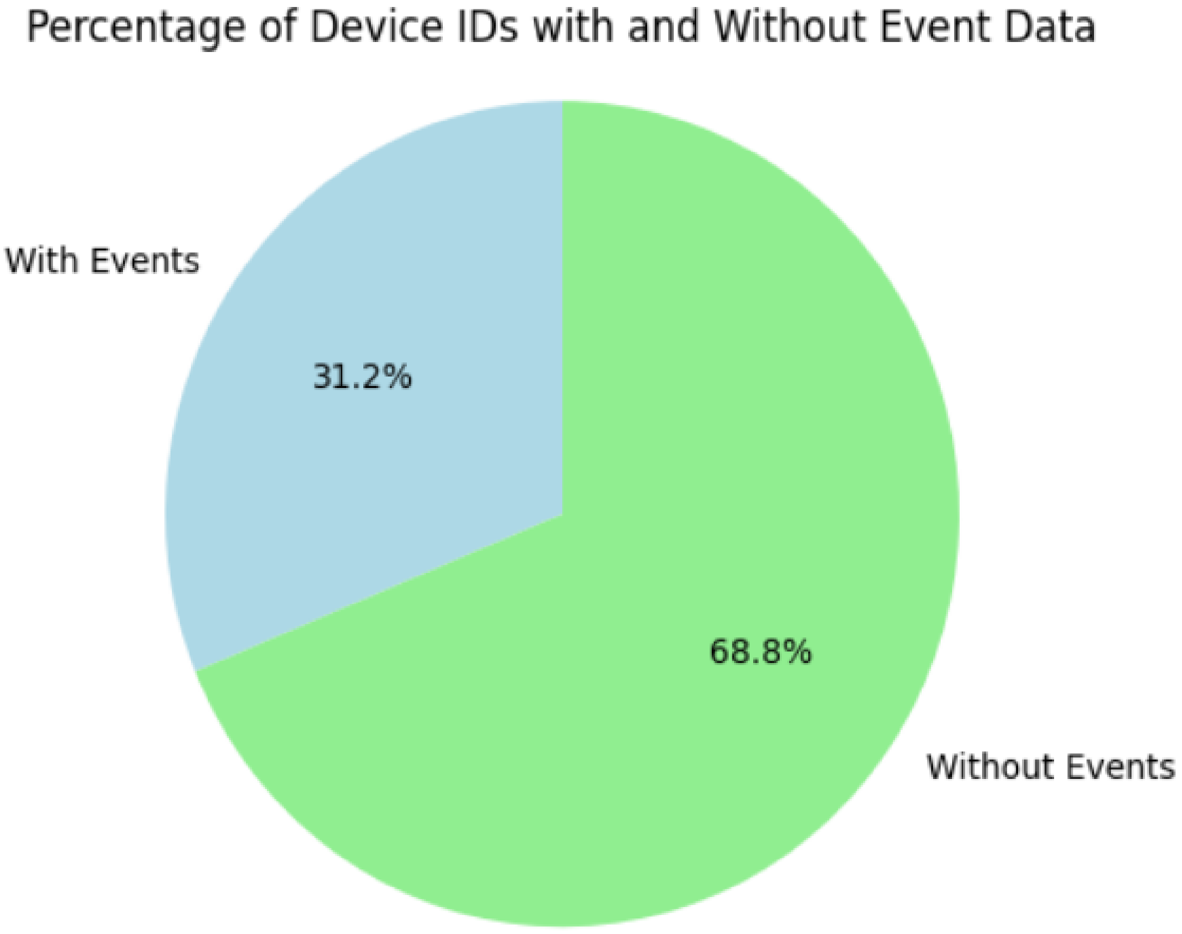
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- b. Boxplot analysis for gender and age[Bivariate]



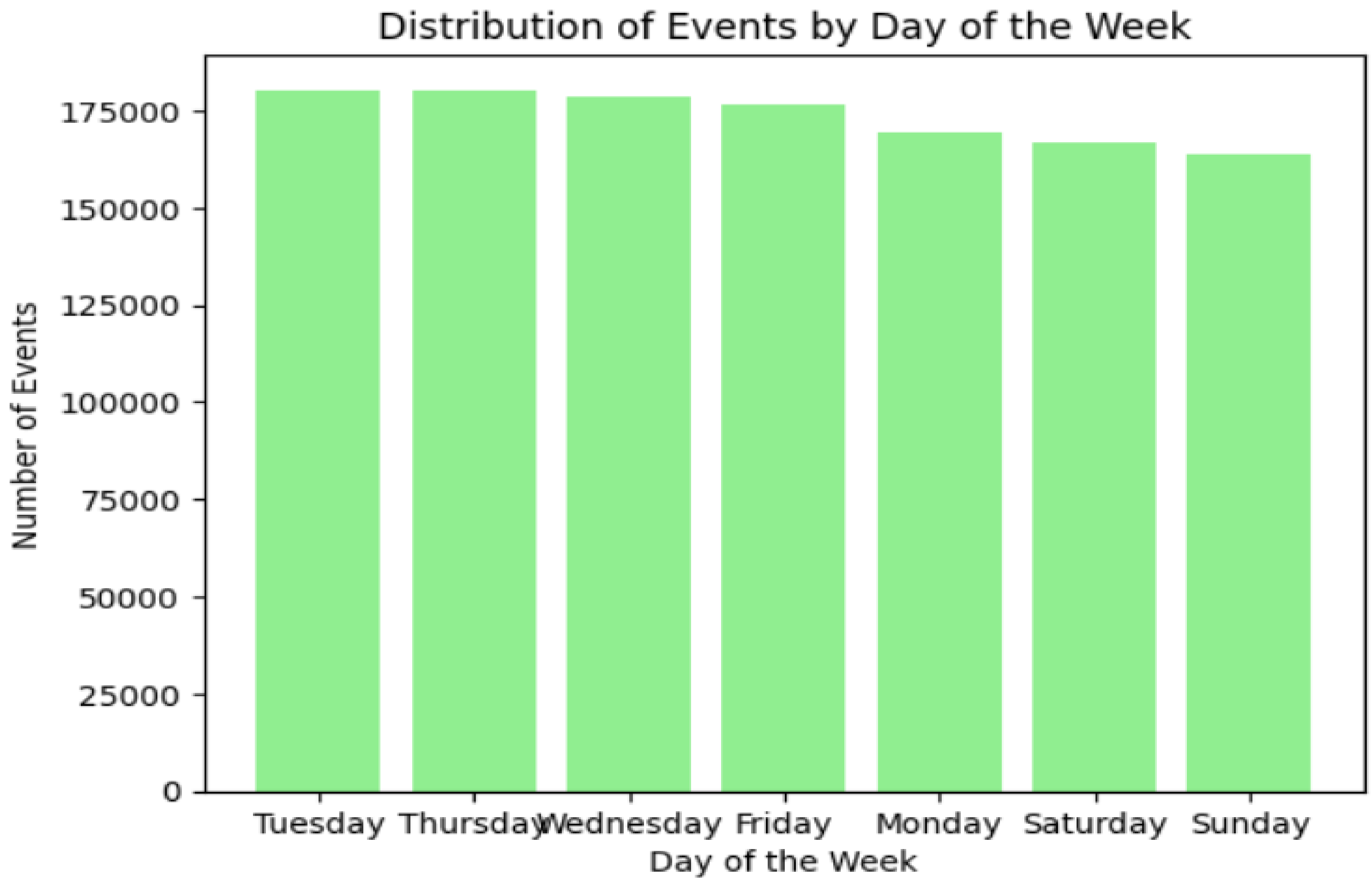
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 02. Trends in Event Data[with respect to devices,days of week,hour,gender and age groups]:
- a. Plot percentage of device_ids with and without event data



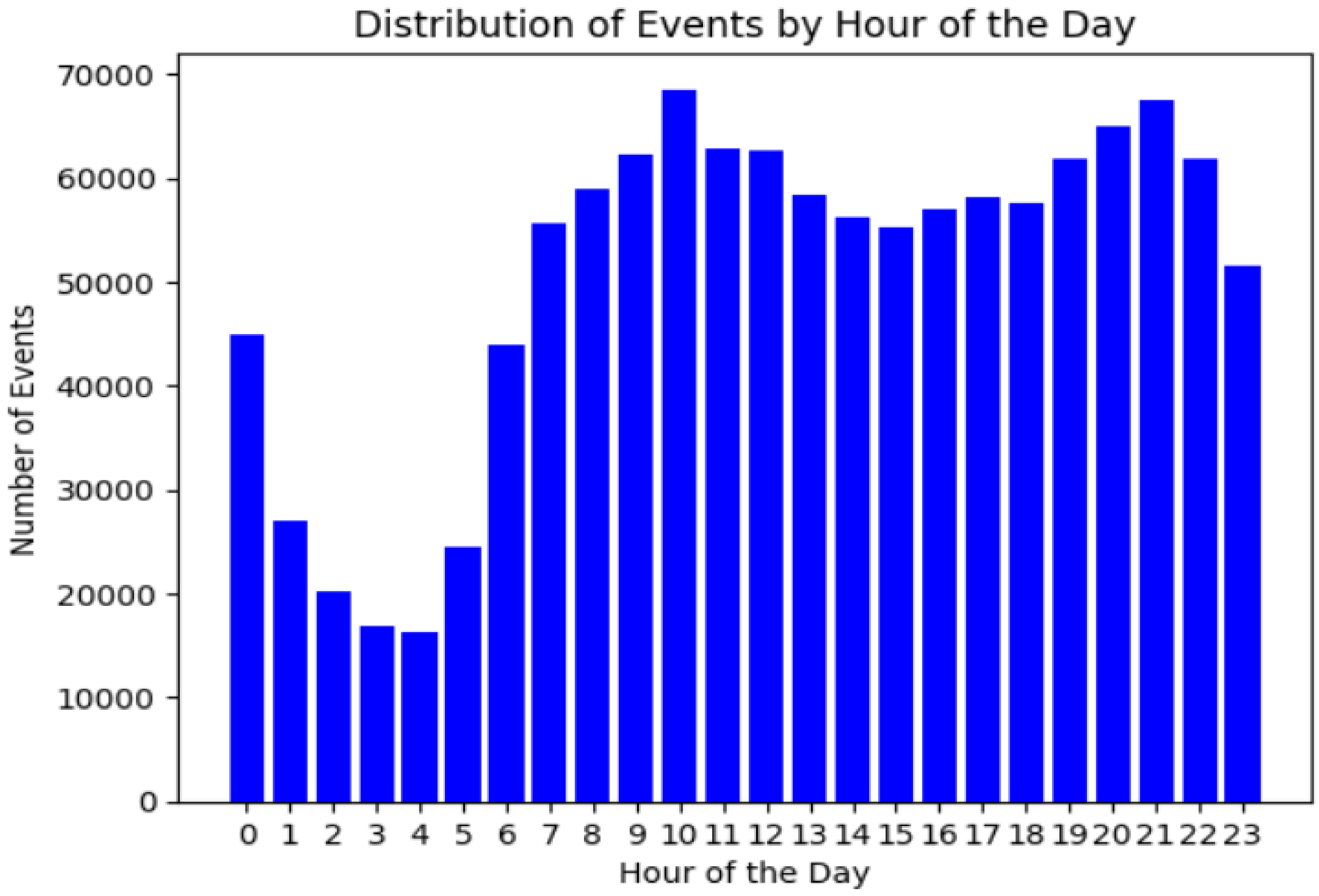
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 02. Trends in Event Data[with respect to devices,days of week,hour,gender and age groups]:
- b. Graph representing the distribution of events on different days of a week



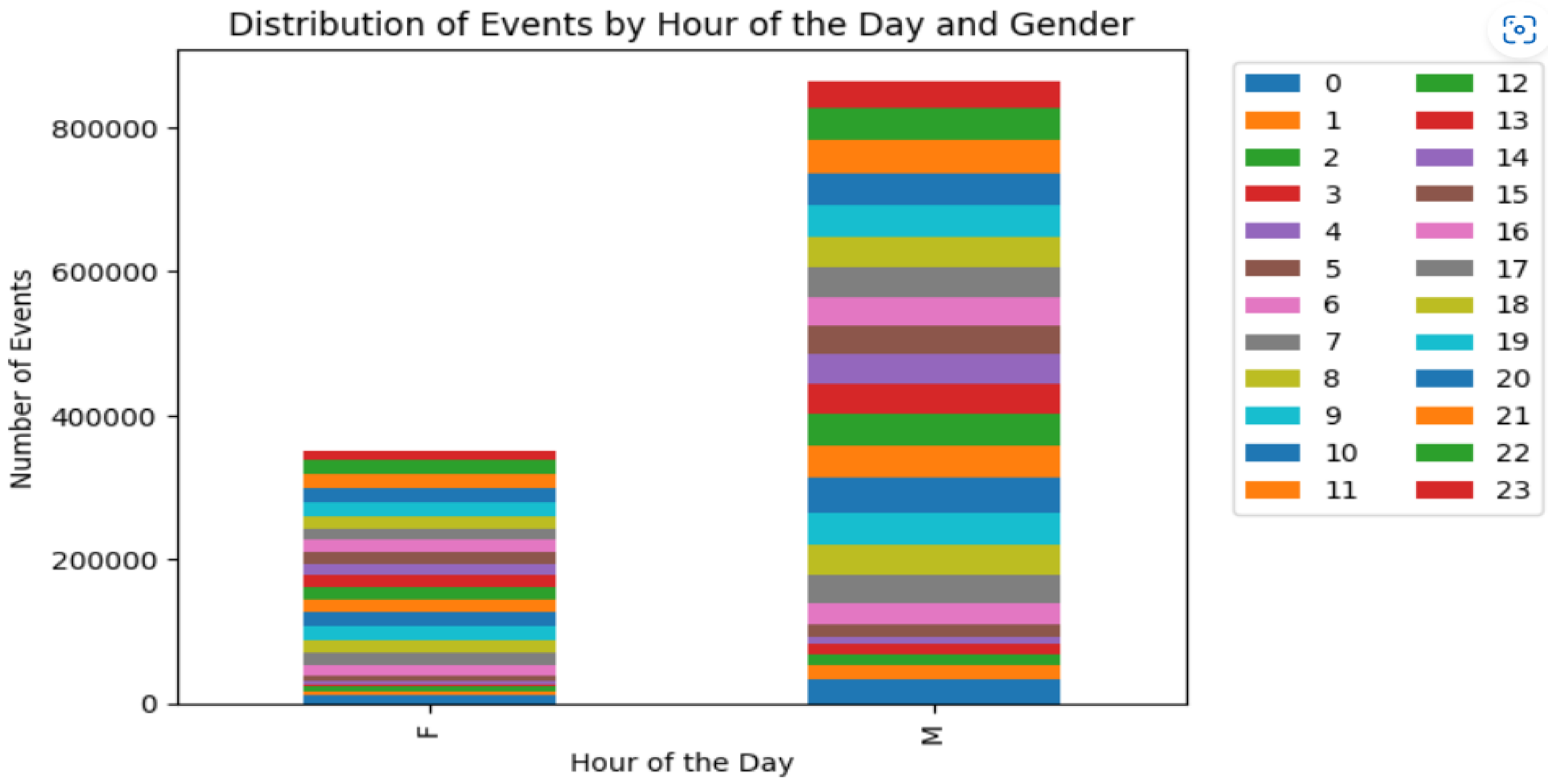
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 02. Trends in Event Data[with respect to devices,days of week,hour,gender and age groups]:
- c. Graph representing the distribution of events per hour[For one week data]



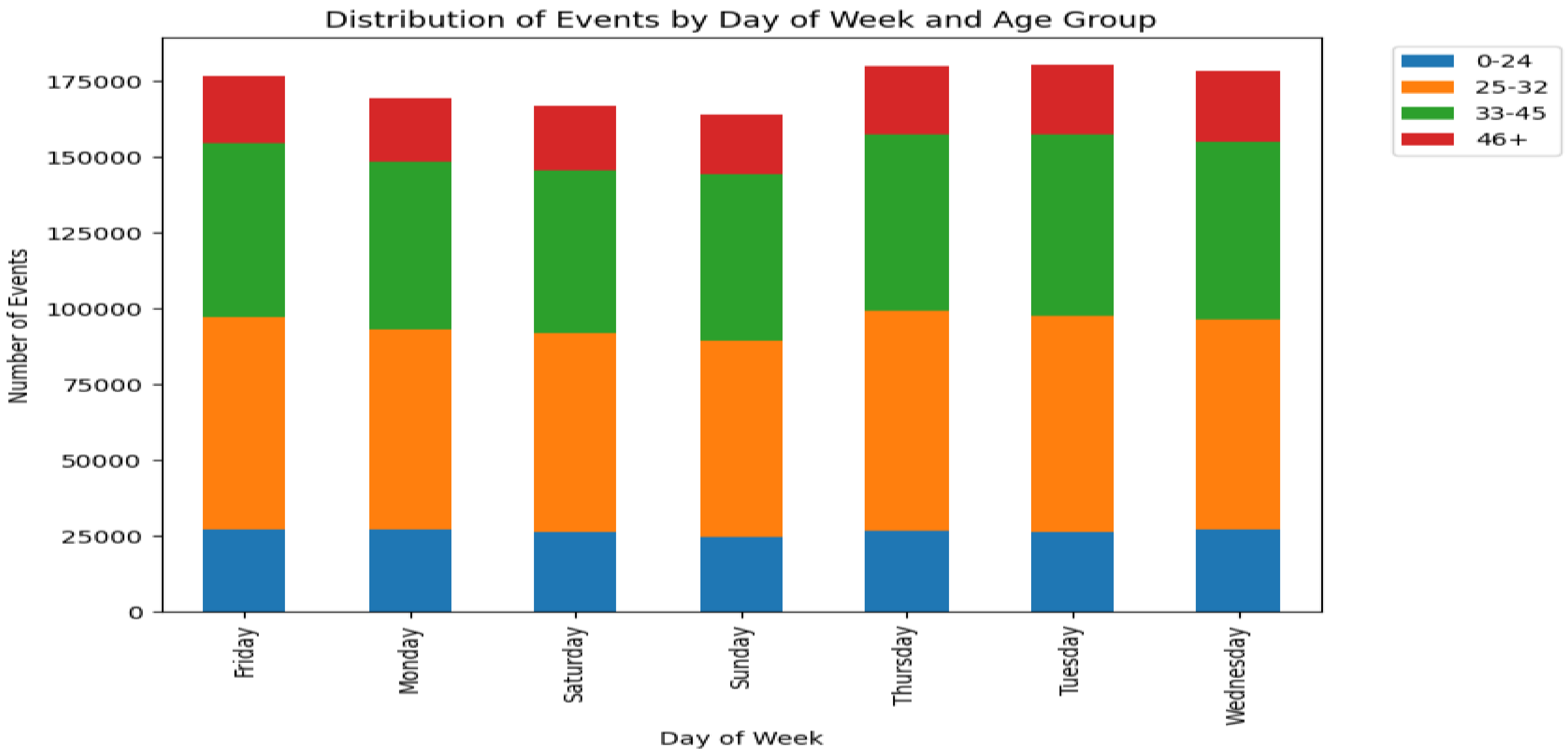
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 02. Trends in Event Data[with respect to devices,days of week,hour,gender and age groups]:
- d. Difference in distribution of events per hour for males and females[Show the difference using appropriate chart for one-week data]



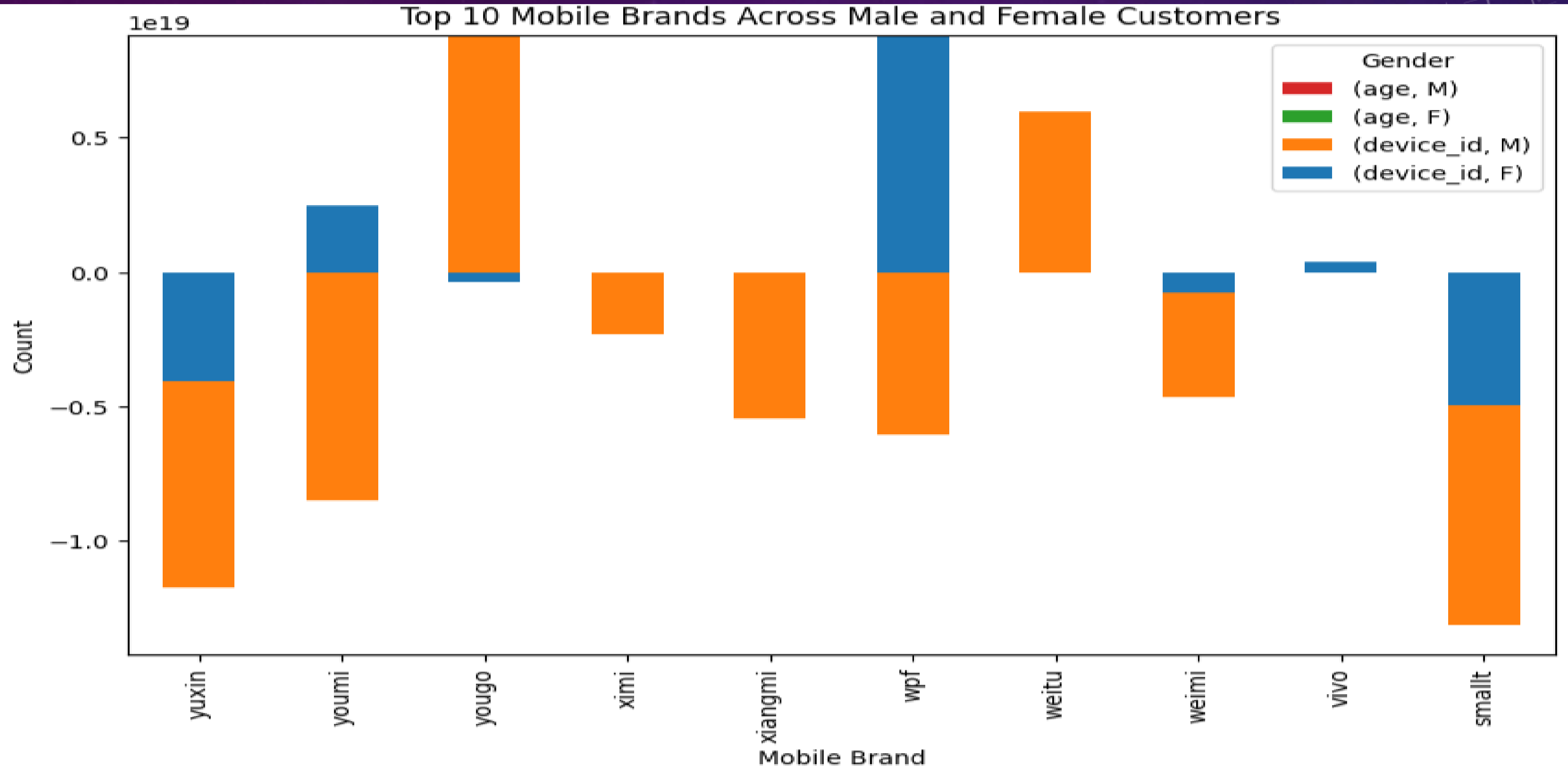
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 02. Trends in Event Data[with respect to devices,days of week,hour,gender and age groups]:
- e. Is there any difference in the distribution of events for different age groups over different days of a week? [Consider the age groups as 0-24,25-32,33-45,and 46+]



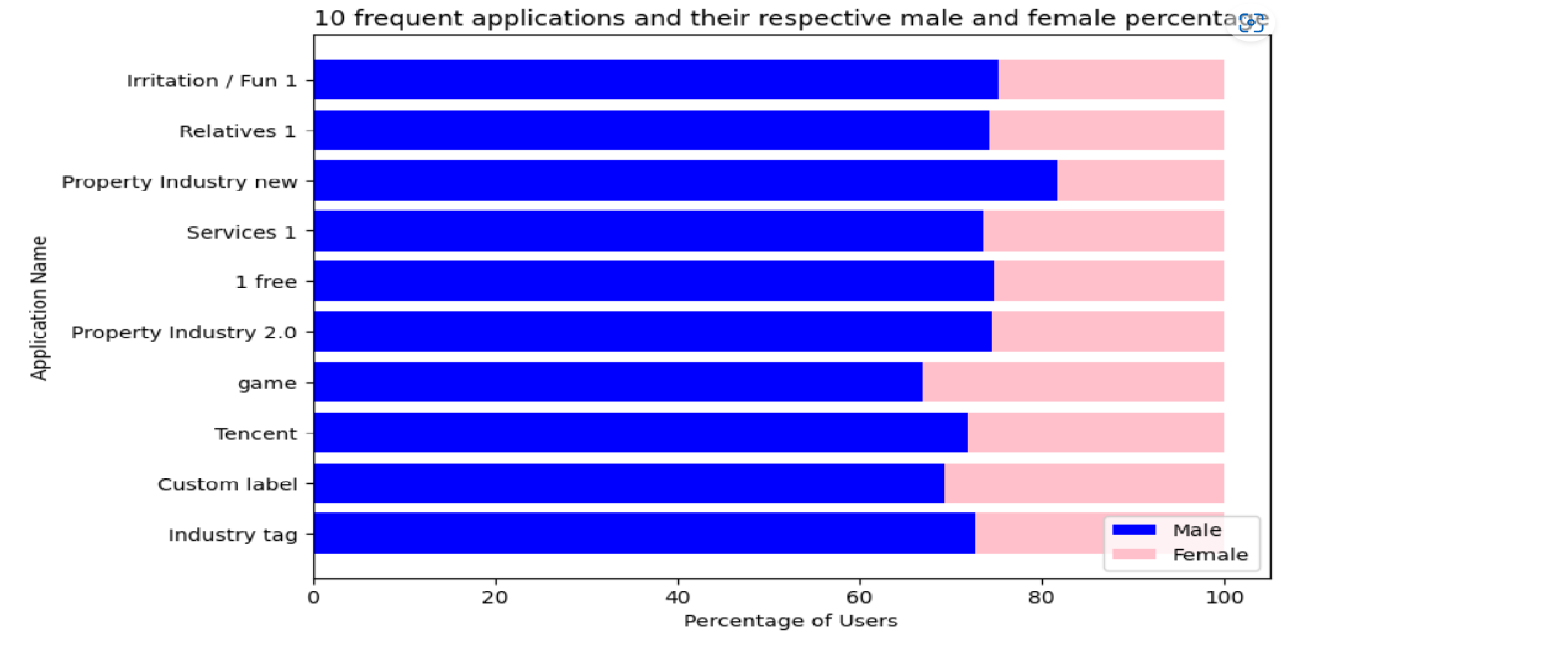
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 03. Phone Brand and Application Preferences:
- a. Stacked bar chart for top 10 mobile brand across male and female customers



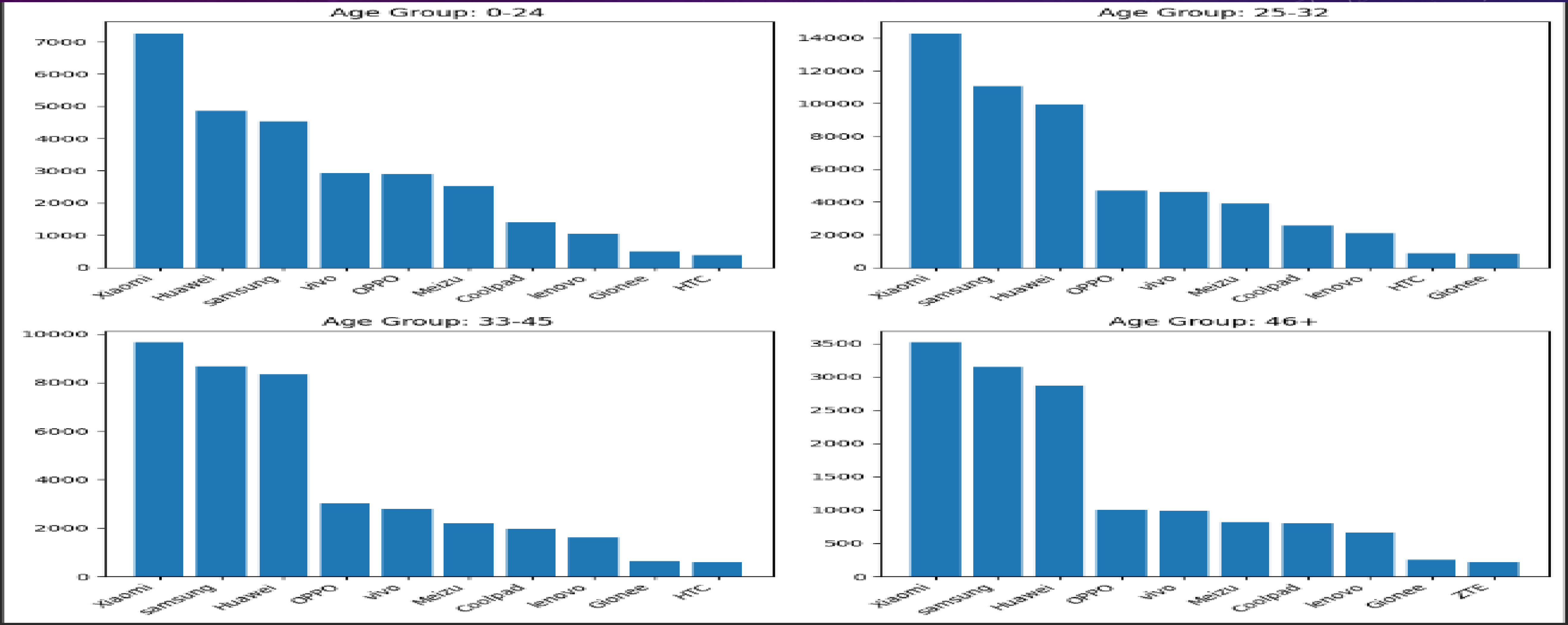
3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 03. Phone Brand and Application Preferences:
- b. Chart representing 10 frequent applications and their respective male and female percentage



3. BASIC EDA AND VISUALIZATION AND FEATURE ENGINEERING

- 03. Phone Brand and Application Preferences:
- c. Top 10 mobile phone brands by age groups [Consider the age groups as 0-24,25-32,33-45 and 46+]



FEATURE ENGINEERING

- Average number of events per device ID:

In [26]:

```python
# Calculate the average number of events per device ID
avg_events_per_id = df_events.groupby('device_id')['event_id'].count().mean()
avg_events_per_id
```

Out[26]: 52.14920634920635

- **Percentage of time the mobile phone was active by calculating the number of events for a device ID:**

Percentage of time the mobile phone was active by calculating the number of events for a device ID:

In [27]:

```python
# Convert the datetimestamp column to a pandas datetime series
df_events['datetimestamp'] = pd.to_datetime(df_events['datetimestamp'])

# Calculate the total number of events and duration for each device ID
device_events = df_events.groupby('device_id').agg({'event_id': 'count', 'datetimestamp': lambda x: x.max() - x.min()})

# Calculate the percentage of time the mobile phone was active
device_events['active_percentage'] = device_events['event_id'] / ((device_events['datetimestamp'].dt.total_seconds() / 60) *
device_events
```

Out[27]:

device_id	event_id	datetimestamp	active_percentage
-1001337759327040000	109	3 days 22:07:00	0.001340
-1002595372059170000	7	1 days 00:09:57	0.000335
-1002733576670970000	55	6 days 15:19:36	0.000400
-1002969456091700000	7	6 days 12:59:34	0.000052
-1005411102947240000	44	2 days 08:49:50	0.000896
...	...	...	...
997893466461332000	7	4 days 00:55:49	0.000084
998208026013018000	71	6 days 04:40:35	0.000553
998402647311351000	4	0 days 01:24:26	0.003290
999208698621622000	37	6 days 05:56:31	0.000286
99976251796408100	3	0 days 00:07:33	0.027594

23310 rows × 3 columns

FEATURE ENGINEERING

- Difference in latitudes and longitudes over a morning period versus the evening and the night periods:

Difference in latitudes and longitudes over a morning period versus the evening and the night periods:

In [28]:

```
# Calculate the morning, evening, and night periods
morning = ((df_events['datetimestamp'].dt.hour >= 6) & (df_events['datetimestamp'].dt.hour <= 12))
evening = ((df_events['datetimestamp'].dt.hour >= 13) & (df_events['datetimestamp'].dt.hour <= 18))
night = ((df_events['datetimestamp'].dt.hour >= 19) | (df_events['datetimestamp'].dt.hour <= 5))

# Calculate the difference in latitudes and longitudes for each device ID during each period
df_morning = df_events[morning].groupby('device_id').agg({'latitude': lambda x: x.max() - x.min(), 'longitude': lambda x: x.
df_evening = df_events[evening].groupby('device_id').agg({'latitude': lambda x: x.max() - x.min(), 'longitude': lambda x: x.
df_night = df_events[night].groupby('device_id').agg({'latitude': lambda x: x.max() - x.min(), 'longitude': lambda x: x.max(

```

In [29]:

```
df_morning
```

Out[29]:

device_id	latitude	longitude
-1001337759327040000	0.83	0.83
-1002595372059170000	0.00	0.00
-1002733576670970000	0.00	0.00
-1002969456091700000	0.00	0.00
-1006244095847670000	0.00	0.00
...	...	...
996163203858940000	0.00	0.00
997893466461332000	0.00	0.00
998208026013018000	32.02	120.85
999208698621622000	0.00	0.01
99976251796408100	0.00	0.00

17985 rows × 2 columns

In [30]:

```
df_evening
```

Out[30]:

device_id	latitude	longitude
-1001337759327040000	0.09	0.09
-1002595372059170000	0.00	0.00
-1002733576670970000	37.79	112.59
-1002969456091700000	0.00	0.00
-1005411102947240000	30.62	114.40
...	...	...
99479029389286800	0.00	0.00
994957985493748000	0.00	0.00
997893466461332000	0.00	0.00
998208026013018000	0.00	0.00
999208698621622000	0.08	0.09

17493 rows × 2 columns

In [31]:

```
df_night
```

Out[31]:

device_id	latitude	longitude
-1002595372059170000	0.00	0.00
-1002733576670970000	37.79	112.59
-1002969456091700000	0.00	0.00
-1005411102947240000	30.62	114.40
-1006244095847670000	0.00	0.00
...	...	...
994957985493748000	0.00	0.00
997893466461332000	0.00	0.00
998208026013018000	0.00	0.00
998402647311351000	0.00	0.01
999208698621622000	27.90	114.38

18513 rows × 2 columns

FEATURE ENGINEERING

- Median latitude and longitude over a period of different events per device ID:

In [32]:

```python
# Calculate the median latitude and longitude for each device ID
median_lat = df_events.groupby('device_id')['latitude'].median()
median_long = df_events.groupby('device_id')['longitude'].median()
```

In [33]:

```python
median_lat
```

Out[33]:

```
device_id
-1001337759327040000    30.20
-1002595372059170000     0.00
-1002733576670970000     0.00
-1002969456091700000     0.00
-1005411102947240000    30.62
                         ...
9978934664613320000      0.00
9982080260130180000      0.00
9984026473113510000     48.50
9992086986216220000     27.82
9997625179640810O        0.00
Name: latitude, Length: 23310, dtype: float64
```

In [34]:

```python
median_long
```

Out[34]:

```
device_id
-1001337759327040000    120.11
-1002595372059170000      0.00
-1002733576670970000      0.00
-1002969456091700000      0.00
-1005411102947240000    114.40
                          ...
9978934664613320000       0.00
9982080260130180000       0.00
9984026473113510000     128.72
9992086986216220000     114.38
999762517964081OO         0.00
Name: longitude, Length: 23310, dtype: float64
```

FEATURE ENGINEERING

- Grouping app categories:

Grouping app categories:

In [35]:

```python
# count the number of unique categories in the dataframe
num_categories = len(app_events_meta_data['category'].unique())
print(f'There are {num_categories} unique categories in the dataframe')

# group the data by category and count the number of apps in each category
grouped_data = app_events_meta_data.groupby('category').count()['app_id']
grouped_data
```

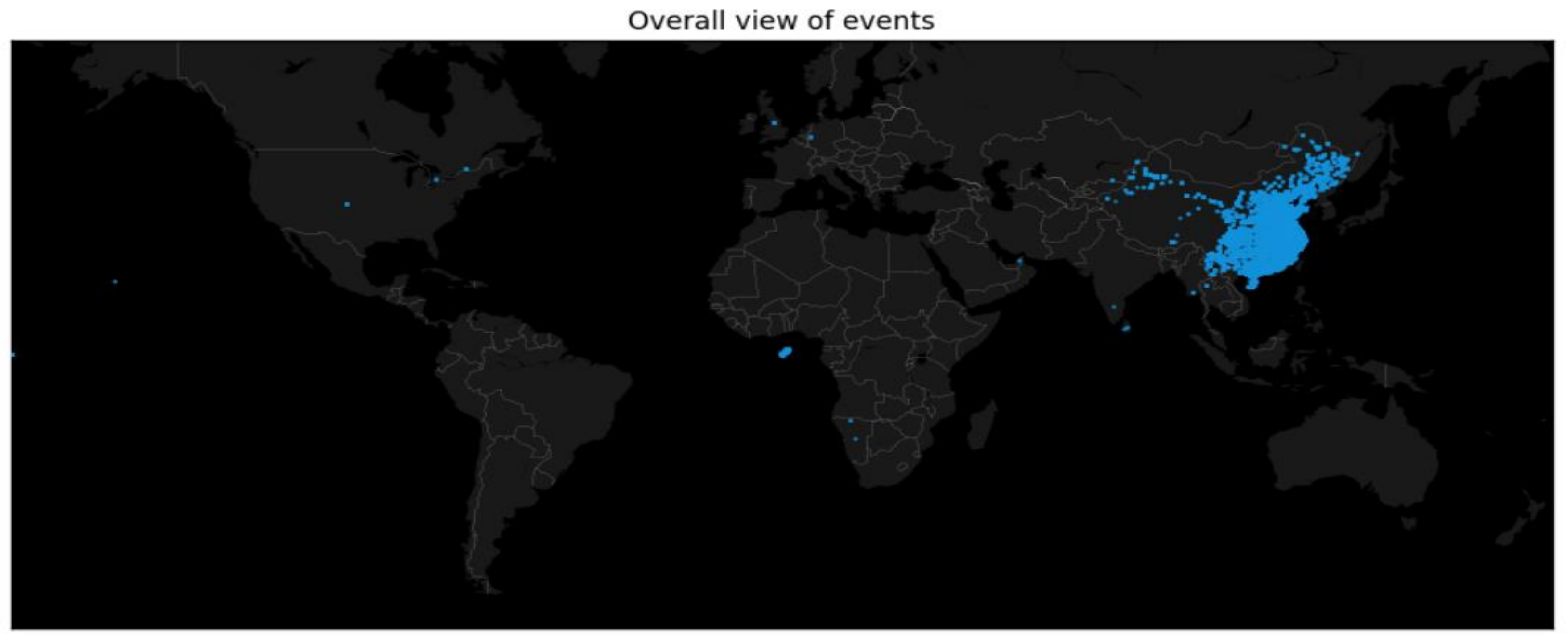
```
There are 556 unique categories in the dataframe
```

Out[35]:

```
category
1 free                 19083
1 reputation              90
1 vitality               276
3 kindom game            157
80s Japanese comic       150
                       ...
violence comic            58
vitality                 295
war chess                 13
weibo                     67
zombies game             116
Name: app_id, Length: 555, dtype: int64
```

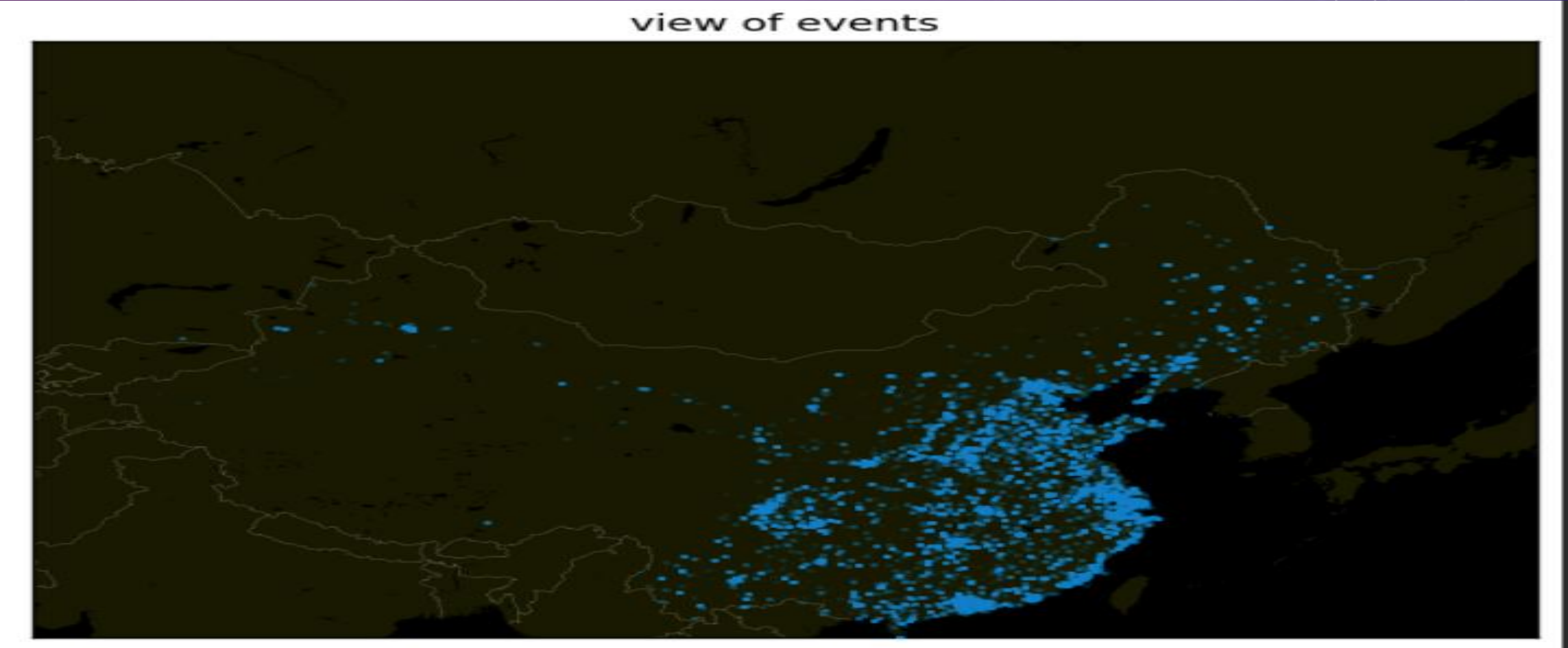
4. ADVANCED VISUALIZATION AND CLUSTERING

- a. Geospatial Visualization
- Overall view of events



4. ADVANCED VISUALIZATION AND CLUSTERING

- a. Geospatial Visualization
- View of events

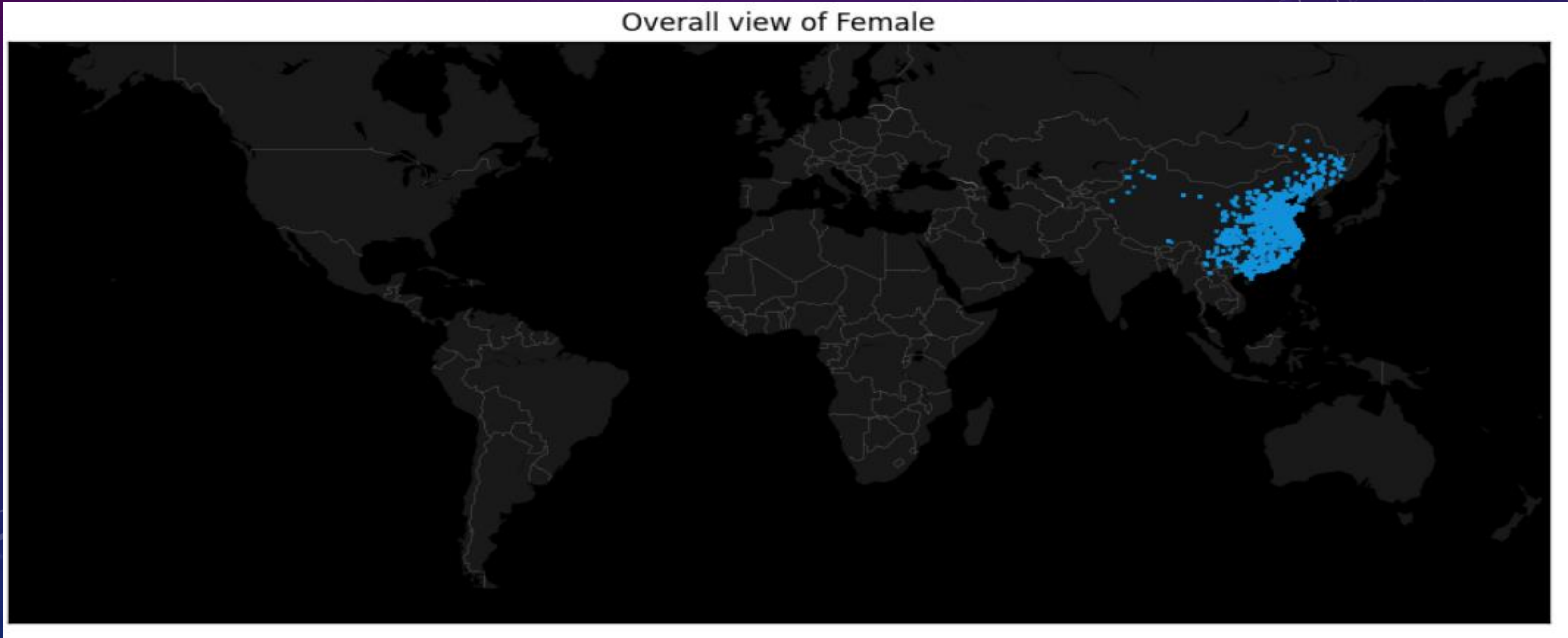


4. ADVANCED VISUALIZATION AND CLUSTERING

- a. Geospatial Visualization
- Overall view of male

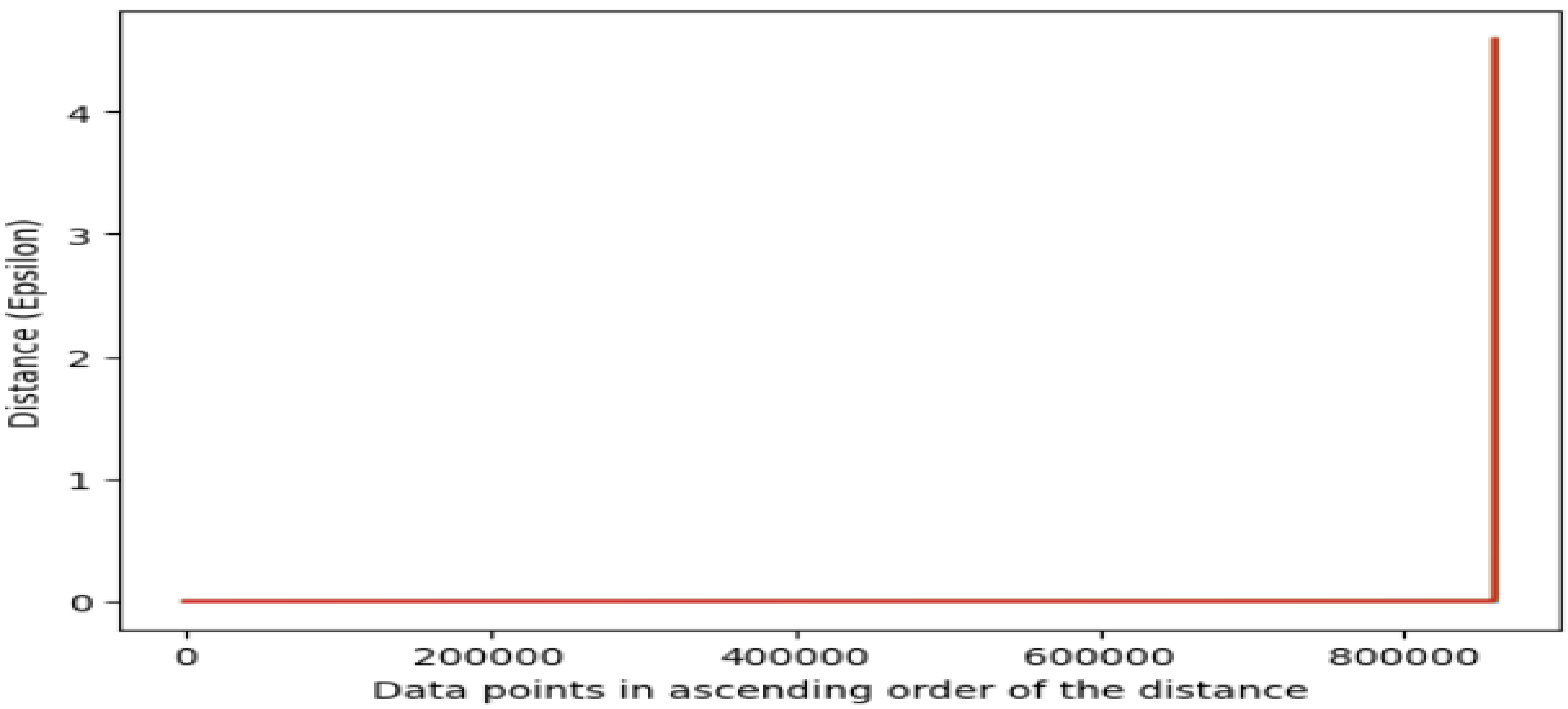
4. ADVANCED VISUALIZATION AND CLUSTERING

- a. Geospatial Visualization
- Overall view of female



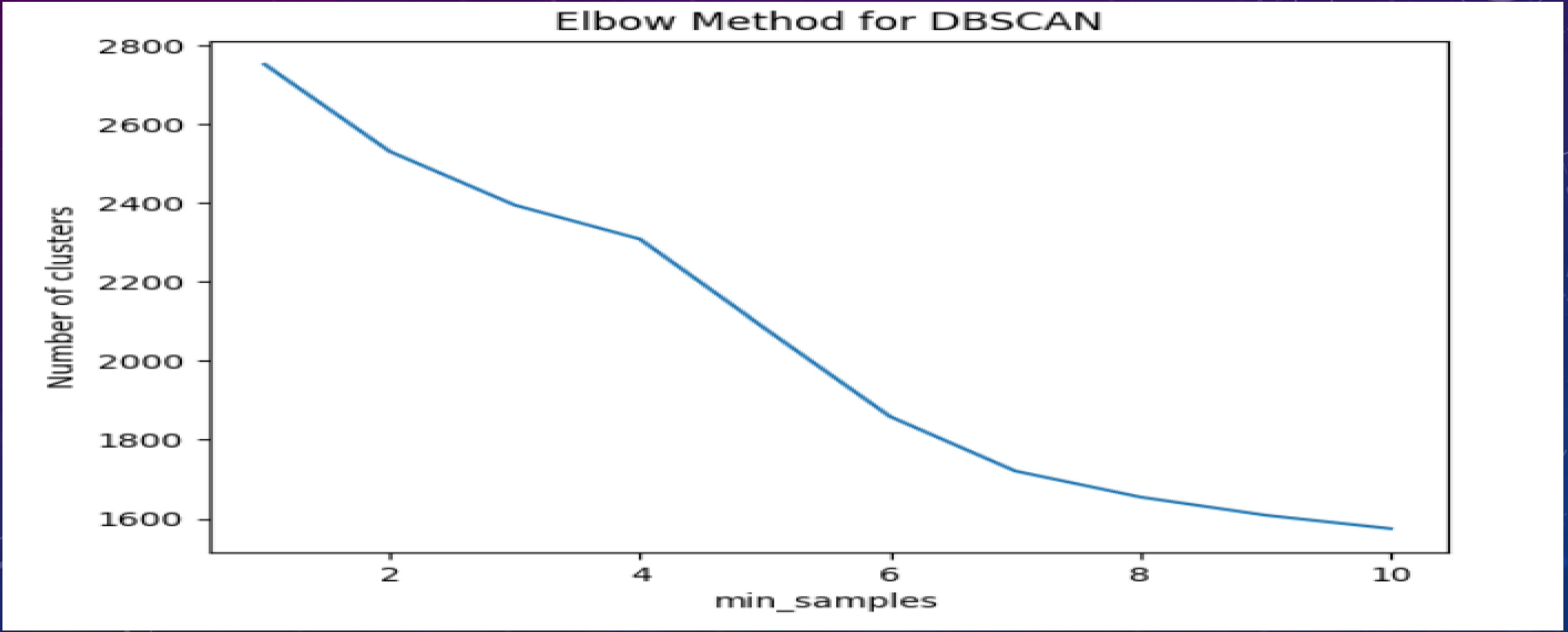
4. ADVANCED VISUALIZATION AND CLUSTERING

- b. DBSCAN Clustering as a preprocessing technique
- Data points in ascending order of the distance



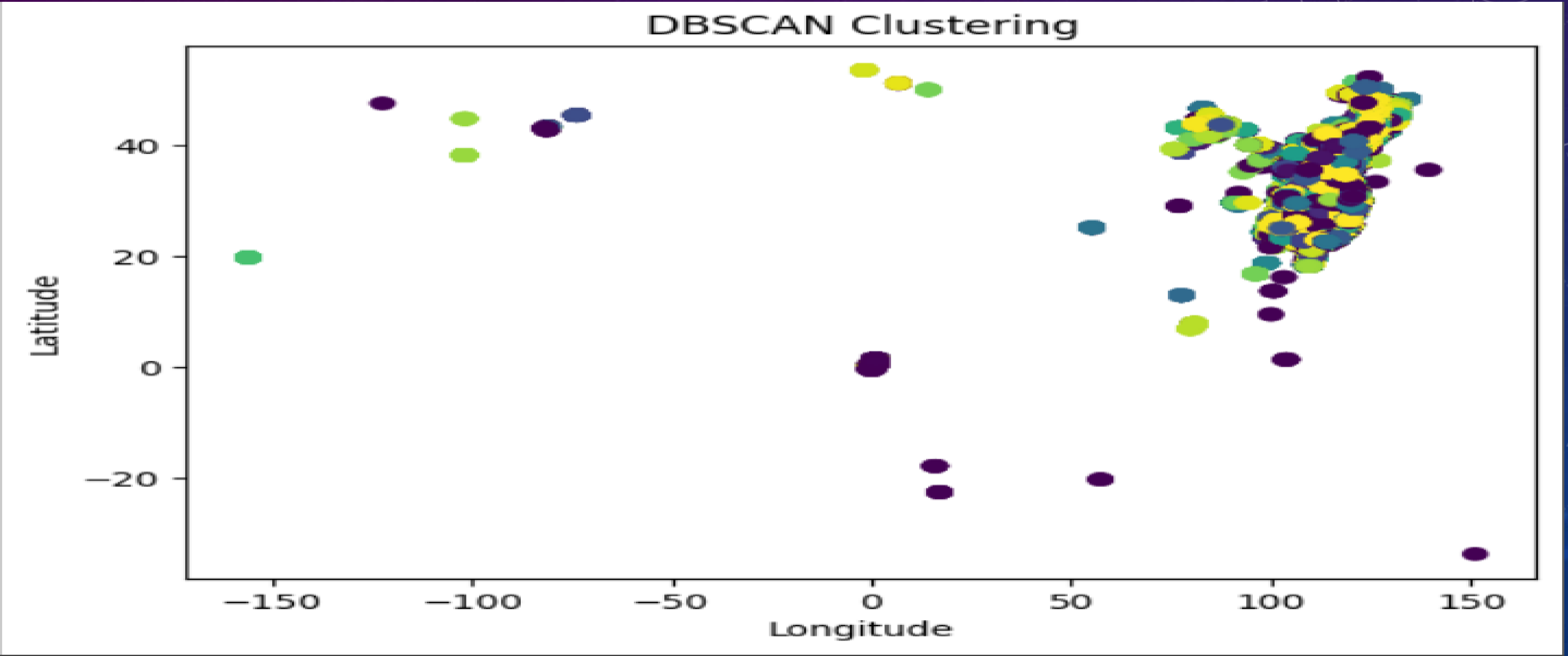
4. ADVANCED VISUALIZATION AND CLUSTERING

- b. DBSCAN Clustering as a preprocessing technique
- Elbow Method for DBSCAN



4. ADVANCED VISUALIZATION AND CLUSTERING

- b. DBSCAN Clustering as a preprocessing technique
- DBSCAN Clustering



4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario1 – gender prediction

- c. Final data preparation and train-test-split

In [53]:
```
# Convert the phone_brand and device_model columns to categorical features
df_events_sample = pd.concat([df_events_sample, pd.get_dummies(df_events_sample["day_of_week"])], axis=1)
```

In [54]:
```
df_events_sample = df_events_sample.drop(['datetimestamp','date','day_of_week'], axis=1)
```

In [55]:
```
df_events_sample
```

Out[55]:

	device_id	gender	age	group_train	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wedne
1064209	2399102809401320000	M	44	M32+	622238.0	29.71	116.00	6	1	0	0	0	0	0	
846019	6564014326179130000	F	39	F32+	1037689.0	26.24	117.61	12	0	0	0	0	0	1	
42305	-2107177132625990000	M	37	M32+	2764947.0	34.52	114.70	22	0	0	0	0	1	0	
22827	-6242501228649110000	M	20	M0-24	529189.0	27.85	111.21	21	0	0	0	1	0	0	
967026	489170860872393000	M	31	M25-32	3137397.0	30.30	120.11	8	0	0	1	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
372651	-4434742026584890000	M	26	M25-32	757243.0	30.55	114.27	0	0	0	0	0	0	1	
205307	1666354678009810000	F	26	F25-32	825768.0	32.06	110.73	20	0	1	0	0	0	0	
492222	5758161416832330000	M	26	M25-32	471289.0	36.06	111.49	20	0	0	0	0	1	0	
969805	6435531434380250000	M	25	M25-32	1632723.0	25.70	100.17	7	0	0	0	0	1	0	
3335	1779631023439400000	F	39	F32+	2918734.0	30.57	114.26	18	0	0	1	0	0	0	

100000 rows × 15 columns

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario1 – gender prediction

- c. Final data preparation and train-test-split

Convert the gender into 1 and 0

In [56]:

```
le = LabelEncoder()
df_events_sample['gender'] = le.fit_transform(df_events_sample['gender'])
```

In [57]:

```
# Separate the target variable from the features
X = df_events_sample.drop(['gender'], axis=1)
y = df_events_sample['gender']
X = X.drop(['group_train','age'],axis=1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [58]:

```
# generating the data into test_data_without_age_and_gender.csv so that we can use it for displaying the table column with f
X_test.to_csv('test_data_without_age_and_gender.csv',index=False)
```

In [59]:

```
X_train
```

Out[59]:

	device_id	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
1008613	-3926780516486000000	1420659.0	30.63	120.58	9	0	1	0	0	0	0	0
769519	-6970593614144750000	332676.0	24.48	118.17	10	0	0	0	1	0	0	0
1116296	589483191079996000	2030001.0	22.66	114.02	17	0	0	1	0	0	0	0
795844	-4968154927622700000	840263.0	39.96	116.38	9	0	0	0	0	1	0	0
787776	-3230567043058250000	1237796.0	41.84	123.44	2	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
608404	-1796008854790130000	861331.0	30.87	104.46	19	0	0	0	1	0	0	0
457731	6546537548974030000	2176814.0	39.14	117.20	7	0	0	0	0	0	1	0
905768	2204630292293180000	1044331.0	39.63	118.15	17	1	0	0	0	0	0	0
1154281	5375599021847300000	2666696.0	38.07	115.13	16	0	0	0	0	0	1	0
872065	3849458946444850000	488644.0	31.74	120.12	11	0	1	0	0	0	0	0

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario1 – gender prediction

- c. Final data preparation and train-test-split

```
In [59]:  1  X_train
```

Out[59]:

	device_id	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
1008613	-3926780516486000000	1420659.0	30.63	120.58	9	0	1	0	0	0	0	0
769519	-6970593614144750000	332676.0	24.48	118.17	10	0	0	0	1	0	0	0
1116296	5894831910799996000	2030001.0	22.66	114.02	17	0	0	1	0	0	0	0
795844	-4968154927622700000	840263.0	39.96	116.38	9	0	0	0	0	1	0	0
787776	-3230567043058250000	1237796.0	41.84	123.44	2	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
608404	-1796008854790130000	861331.0	30.87	104.46	19	0	0	0	1	0	0	0
457731	6546537548974030000	2176814.0	39.14	117.20	7	0	0	0	0	0	1	0
905768	2204630292293180000	1044331.0	39.63	118.15	17	1	0	0	0	0	0	0
1154281	5375599021847300000	2666696.0	38.07	115.13	16	0	0	0	0	0	1	0
872065	3849458946444850000	488644.0	31.74	120.12	11	0	1	0	0	0	0	0

80000 rows × 12 columns

```
In [60]:  1  X_test
```

Out[60]:

	device_id	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
712128	-8023825242156080000	1454712.0	23.09	109.63	15	0	1	0	0	0	0	0
1172263	-6298897121637350000	1547479.0	30.00	104.00	11	0	0	0	0	1	0	0
200307	-1443596674095190000	2296875.0	36.85	120.48	19	1	0	0	0	0	0	0
1062813	7458466907666100000	1753402.0	26.22	119.52	17	0	0	1	0	0	0	0
1107853	3074308677943390000	1357010.0	34.42	108.58	3	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
1123175	7825425485977200000	1535032.0	32.04	118.76	22	1	0	0	0	0	0	0
868673	3457634523376510000	1160316.0	36.78	116.92	20	1	0	0	0	0	0	0
71499	8221364290697850000	1565185.0	28.21	112.90	22	0	0	0	1	0	0	0
1070365	6851410235503600000	3227279.0	29.68	109.15	8	0	0	0	0	1	0	0
1189507	-4142301196234850000	2305079.0	39.19	117.47	4	0	0	0	0	1	0	0

20000 rows × 12 columns

MODEL BUILDING

Scenario1 – gender prediction

- a. Hyperparameters tuning for various Algorithms

```
hyperparameters = {
    'LogisticRegression': {
        'algorithm':LogisticRegression(),
        'parameters':{
            'penalty':['l2', 'elasticnet', 'none'],
            'C' : [0.01, 0.1, 1, 10, 100],
            'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
        }
    },
    'Random_Forest_Classifier': {
        'algorithm': RandomForestClassifier(),
        'parameters': {
            'n_estimators': [50,100, 150, 200,250,300],
            'max_depth': [10, 20, 30],
            'min_samples_split':[2, 4, 8],
            'max_features': [4, 6, 8, 10],
            'min_samples_leaf': [1, 2, 4,6, 8, 10],
            'max_features': ['auto', 'sqrt', 'log2', None]
         }
    },

     'XGBClassifier': {
        'algorithm': XGBClassifier(),
        'parameters' : {
            'min_child_weight': [1, 5, 10],
            'gamma': [0.5, 1, 1.5, 2, 5],
            'subsample': [0.6, 0.8, 1.0],
            #'colsample_by_tree': [0.6, 0.8, 1.0],
            'max_depth': [3, 4, 5],
            'n_estimators': range(60, 360, 40),
            'learning_rate': [0.1, 0.01, 0.05]
         }
    }
}
```

MODEL BUILDING

Scenario1 – gender prediction

- a. Hyperparameters tuning for various Algorithms

In [63]:

```python
# get the model hyperparameters from the dictionary and execute one by one for finding best hyper parameters
for hyperparam in hyperparameters.values():
    algorithm = hyperparam.get('algorithm')
    parameters = hyperparam.get('parameters')
    grid_search_cv_model = GridSearchCV(estimator=algorithm, param_grid=parameters, cv=3)
    grid_search_cv_model.fit(X_train, y_train)
    print('Algorithm: ' + str(algorithm))
    print('Optimized Parameters: ' + str(grid_search_cv_model.best_params_))
    print("====================================")
```

```
Algorithm: LogisticRegression()
Optimized Parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
====================================
Algorithm: RandomForestClassifier()
Optimized Parameters: {'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
200}
====================================
Algorithm: XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
Optimized Parameters: {'gamma': 1.5, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 340, 'subsamp
le': 0.6}
====================================
```

MODEL BUILDING

Scenario1 – gender prediction

- a. Stacking

In [70]:

```
# set the models
log_reg = LogisticRegression(random_state=45,C=0.01, penalty='l2', solver='newton-cg')
rf = RandomForestClassifier(random_state=45,max_depth=30, max_features='log2', min_samples_leaf=1, min_samples_split=2, n_es
xgb = XGBClassifier(random_state=45,gamma=1.5, learning_rate=0.1, max_depth=5, min_child_weight=1, n_estimators=340, subsamp
stacking_demo = StackingCVClassifier(random_state=45,classifiers=[log_reg, rf], meta_classifier=xgb, use_probas=True, cv=5)
```

In [71]:

```
# Do CV
for clf, label in zip([log_reg, rf, xgb],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, X_train, y_train, cv=5,scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

```

```
Accuracy: 0.73 (+/- 0.00) [lr]
Accuracy: 0.92 (+/- 0.00) [Random Forest]
Accuracy: 0.87 (+/- 0.00) [StackingClassifier]
```

- I will be using the Random Forest Classifier here since it has the maximum Accuracy of 0.92

MODEL BUILDING

Scenario1 – gender prediction

- b. Gender Prediction

b. Gender Prediction

Accuracy

In [72]:

```
scores = model_selection.cross_val_score(rf, X_train, y_train, cv=3,scoring='accuracy')
print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), 'Random Forest'))
```

```
Accuracy: 0.92 (+/- 0.00) [Random Forest]
```

- I have used the Random Forest Classifier here since it has the maximum Accuracy of 0.92

MODEL BUILDING

Scenario1 – gender prediction

- ROC curve and AUC

ROC curve and AUC

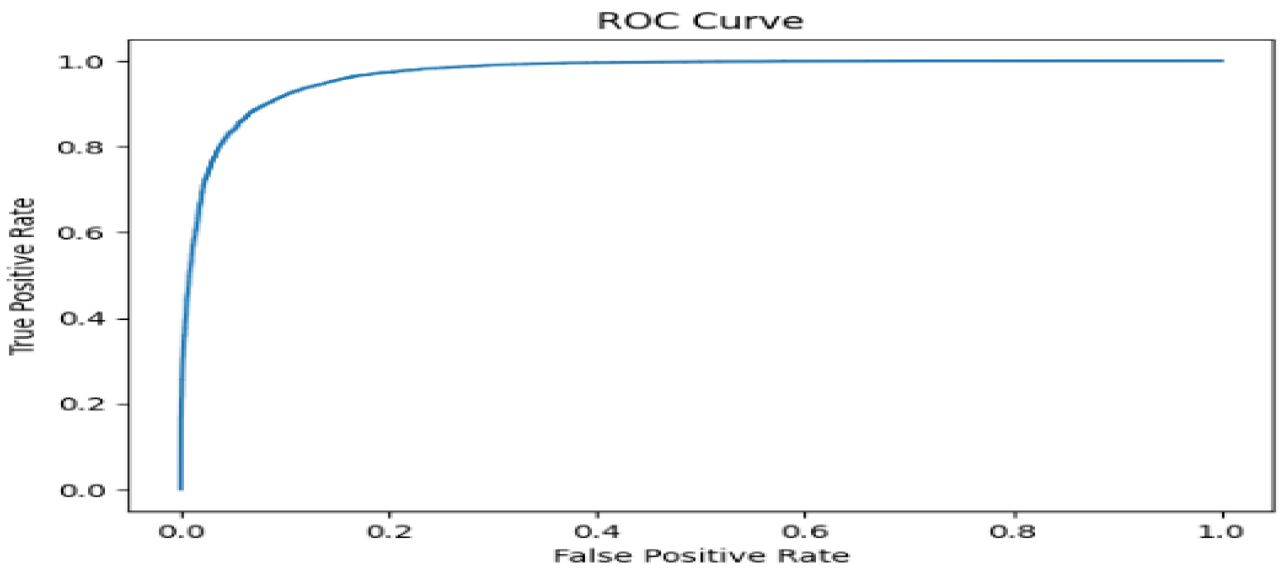
In [75]:

```
# predict probabilities for positive class
y_probs = rf.predict_proba(X_test)[:, 1]

# calculate false positive rate, true positive rate, and threshold values
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# plot ROC curve
plt.plot(fpr, tpr)
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

# calculate AUC score
auc = roc_auc_score(y_test, y_probs)
print('AUC Score:', auc)
```




```
AUC Score: 0.9699972061007937
```

MODEL BUILDING

Scenario1 – gender prediction

- Kolmogorov–Smirnov (KS) statistic

Kolmogorov–Smirnov (KS) statistic

In [76]:

```python
# obtain predicted probabilities for the test data
y_pred_prob = rf.predict_proba(X_test)

# separate the test data into positive and negative classes
pos_idx = np.where(y_test == 1)[0]
neg_idx = np.where(y_test == 0)[0]

# sort the predicted probabilities in descending order
pos_probs = y_pred_prob[pos_idx][:, 1]
neg_probs = y_pred_prob[neg_idx][:, 1]
pos_probs_sorted = np.sort(pos_probs)[::-1]
neg_probs_sorted = np.sort(neg_probs)[::-1]

min_len = min(len(pos_probs_sorted), len(neg_probs_sorted))
pos_cdf = np.cumsum(pos_probs_sorted[:min_len]) / np.sum(pos_probs_sorted[:min_len])
neg_cdf = np.cumsum(neg_probs_sorted[:min_len]) / np.sum(neg_probs_sorted[:min_len])

# compute the KS statistic as the maximum absolute difference between the CDFs
ks = np.max(np.abs(pos_cdf - neg_cdf))
ks
```

Out[76]: 0.1660929886061295З

MODEL BUILDING

Scenario1 – gender prediction

- The probability bands identified through the KS table for the top 3 and bottom 3 deciles on the train data set

The probability bands identified through the KS table for the top 3 and bottom 3 deciles on the train data set

In [77]:

```python
# get predicted probabilities for the positive class
y_proba_train = rf.predict_proba(X_train)[:, 1]

# calculate the false positive rate, true positive rate, and thresholds for the ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_proba_train)

# reset the index of y_train
y_train_reset = y_train.reset_index(drop=True)

# sort the predicted probabilities in descending order
idx = y_proba_train.argsort()[::-1]
y_proba_train_sorted = y_proba_train[idx]
y_train_sorted = y_train_reset[idx]

# calculate the number of samples and positive samples in each decile
n_samples = len(y_train_sorted)
n_pos_samples = y_train_sorted.sum()
n_neg_samples = n_samples - n_pos_samples
decile_size = n_samples // 10
pos_counts = np.cumsum(y_train_sorted)
neg_counts = np.arange(1, n_samples+1) - pos_counts

# calculate the CDFs for the positive and negative classes in each decile
pos_cdfs = pos_counts / n_pos_samples
neg_cdfs = neg_counts / n_neg_samples

# calculate the KS statistic and the threshold for the top 3 and bottom 3 deciles
ks_top = pos_cdfs[:decile_size].max() - neg_cdfs[:decile_size].max()
ks_bottom = pos_cdfs[-decile_size:].max() - neg_cdfs[-decile_size:].max()
thresh_top = y_proba_train_sorted[decile_size]
thresh_bottom = y_proba_train_sorted[-decile_size]

# calculate the probability bands for the top 3 and bottom 3 deciles
band_top = y_proba_train_sorted[:decile_size].min(), y_proba_train_sorted[:decile_size].max()
band_bottom = y_proba_train_sorted[-decile_size:].min(), y_proba_train_sorted[-decile_size:].max()

print(f"KS statistic for top 3 deciles: {ks_top:.3f}")
print(f"KS statistic for bottom 3 deciles: {ks_bottom:.3f}")
print(f"Probability band for top 3 deciles: {band_top}")
print(f"Probability band for bottom 3 deciles: {band_bottom}")
```

```
KS statistic for top 3 deciles: 0.137
KS statistic for bottom 3 deciles: 0.000
Probability band for top 3 deciles: (0.989388221942366, 1.0)
Probability band for bottom 3 deciles: (0.0, 0.08131578947368422)
```

MODEL BUILDING

Scenario1 – gender prediction

- Gender Predictions Result

Gender Predictions Result

In [78]:

```
gender_map = {1: "M", 0: "F"}
y_pred_gender = [gender_map[y] for y in y_pred]
print(y_pred_gender)
```

```
['M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'M', 'F',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'M', 'M', 'F',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M',
'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'F',
'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'F', 'F',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'F', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'M',
'F', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'F', 'F',
'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'M',
'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M',
'M', 'M', 'F', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'F', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'F', 'M', 'F', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'F',
```

MODEL BUILDING

Scenario1 – gender prediction

- Saving the Model for future use

Saving the Model for future use

In [79]:

```

# Save the model as a pickle file
with open('scenario1_gender_model.pkl', 'wb') as file:
    pickle.dump(rf, file)
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario1 – age prediction

- c. Final data preparation and train-test-split

In [52]:
```
# Convert the phone_brand and device_model columns to categorical features
df_events_sample = pd.concat([df_events_sample, pd.get_dummies(df_events_sample["day_of_week"])], axis=1)
```

In [53]:
```
df_events_sample = df_events_sample.drop(['datetimestamp','date','day_of_week'], axis=1)
```

In [54]:
```
df_events_sample.head()
```

Out[54]:

	device_id	gender	age	group_train	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednes
771407	-5821716319485760000	M	27	M25-32	2002264.0	47.61	126.10	16	0	1	0	0	0	0	
463847	8859850364207260000	M	36	M32+	2462345.0	39.13	117.16	20	1	0	0	0	0	0	
47758	-1076279644989310000	M	25	M25-32	2029553.0	23.32	116.35	17	0	0	0	0	1	0	
456384	295685201038338000	F	26	F25-32	1713875.0	34.31	108.95	11	0	0	0	0	0	0	
753031	-7500045804733750000	F	28	F25-32	2767699.0	30.76	108.44	0	0	0	0	0	1	0	

Convert the gender into 1 and 0

In [55]:
```
le = LabelEncoder()
df_events_sample['gender'] = le.fit_transform(df_events_sample['gender'])
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario1 – age prediction

- c. Final data preparation and train-test-split

Out[56]:

	device_id	gender	age	group_train	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednes
771407	-5821716319485760000	1	27	M25-32	2002264.0	47.61	126.10	16	0	1	0	0	0	0	
463847	8859850364207260000	1	36	M32+	2462345.0	39.13	117.16	20	1	0	0	0	0	0	
47758	-1076279644989310000	1	25	M25-32	2029553.0	23.32	116.35	17	0	0	0	0	1	0	
456384	2956852010383380000	0	26	F25-32	1713875.0	34.31	108.95	11	0	0	0	0	0	0	
753031	-7500045804733750000	0	28	F25-32	2767699.0	30.76	108.44	0	0	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
965659	1999051376221720000	0	49	F32+	2076810.0	34.60	113.52	9	0	0	0	0	0	1	
191584	8759274431511730000	1	22	M0-24	3149162.0	37.77	112.54	19	0	0	0	1	0	0	
206720	-7275422412456760000	0	24	F0-24	2941381.0	27.09	114.91	11	0	0	1	0	0	0	
981226	6918962482889750000	0	26	F25-32	780972.0	26.77	113.52	23	1	0	0	0	0	0	
55813	-8340098378141150000	1	28	M25-32	1258839.0	34.74	111.92	17	0	0	0	0	0	0	

100000 rows × 15 columns

In [57]:

```
# Separate the target variable from the features
X = df_events_sample.drop(['age'], axis=1)
y = df_events_sample['age']
X = X.drop(['group_train','gender'], axis=1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario1 – age prediction

- c. Final data preparation and train-test-split

Out[56]:

	device_id	gender	age	group_train	event_id	latitude	longitude	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednes
771407	-5821716319485760000	1	27	M25-32	2002264.0	47.61	126.10	16	0	1	0	0	0	0	
463847	8859850364207260000	1	36	M32+	2462345.0	39.13	117.16	20	1	0	0	0	0	0	
47758	-1076279644989310000	1	25	M25-32	2029553.0	23.32	116.35	17	0	0	0	0	1	0	
456384	2956852010383338000	0	26	F25-32	1713875.0	34.31	108.95	11	0	0	0	0	0	0	
753031	-7500045804733750000	0	28	F25-32	2767699.0	30.76	108.44	0	0	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
965659	1999051376221720000	0	49	F32+	2076810.0	34.60	113.52	9	0	0	0	0	0	1	
191584	8759274431511730000	1	22	M0-24	3149162.0	37.77	112.54	19	0	0	0	1	0	0	
206720	-7275422412456760000	0	24	F0-24	2941381.0	27.09	114.91	11	0	0	1	0	0	0	
981226	6918962482889750000	0	26	F25-32	780972.0	26.77	113.52	23	1	0	0	0	0	0	
55813	-8340098378141150000	1	28	M25-32	1258839.0	34.74	111.92	17	0	0	0	0	0	0	

100000 rows × 15 columns

In [57]:

```python
# Separate the target variable from the features
X = df_events_sample.drop(['age'], axis=1)
y = df_events_sample['age']
X = X.drop(['group_train','gender'], axis=1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

MODEL BUILDING

Scenario1 – age prediction

- Random Forest Regressor is a popular machine learning algorithm for solving regression problems. Here are some of the reasons why I have used it for age prediction:
- Handles non-linear data: Random Forest Regressor can handle non-linear and complex data distributions. It can capture the non-linear relationships between the input features and the target variable (age in this case) much better than linear models.
- Robust to noise and outliers: Random Forest Regressor is less sensitive to noise and outliers in the data. It can handle missing or corrupted values by imputing missing values or using surrogate splits.
- Feature Importance: Random Forest Regressor can be used to determine the relative importance of each feature in predicting the target variable. This can help in identifying the most important factors that contribute to the age prediction.
- Scalability: Random Forest Regressor is highly scalable and can handle large datasets with thousands of features.
- Ensemble Learning: Random Forest Regressor uses an ensemble of decision trees, which are trained on different subsets of the data. This results in a more robust and accurate prediction compared to a single decision tree.
- Overall, Random Forest Regressor is a powerful and flexible algorithm that can handle a wide range of data distributions and can be used for accurate age prediction.

MODEL BUILDING

Scenario1 – age prediction

- a. Hyperparameters tuning for various Algorithms

HyperParameters tuning for various Algorithms

In [63]:

```
hyperparameters = {
    'Random_Forest_Regression': {
        'algorithm': RandomForestRegressor(),
        'parameters': {
            'n_estimators': [50,100, 150, 200,250,300],
            'max_depth': [10, 20, 30],
            'min_samples_split':[2, 4, 8],
            'max_features': [4, 6, 8, 10],
            'min_samples_leaf': [1, 2, 4,6, 8, 10],
            'max_features': ['auto', 'sqrt', 'log2', None]
         }
    }
}
```

In [64]:

```
# get the model hyperparameters from the dictionary and execute one by one for finding best hyper parameters
for hyperparam in hyperparameters.values():
    algorithm = hyperparam.get('algorithm')
    parameters = hyperparam.get('parameters')
    grid_search_cv_model = GridSearchCV(estimator=algorithm, param_grid=parameters, cv=3)
    grid_search_cv_model.fit(X_train, y_train)
    print('Algorithm: ' + str(algorithm))
    print('Optimized Parameters: ' + str(grid_search_cv_model.best_params_))
    print("====================================")
```

MODEL BUILDING

Scenario1 – age prediction

- Age Prediction Using Regression
    - RMSE and R-Squared

RMSE

In [66]:

```
rf_regressor = RandomForestRegressor(max_depth=10, max_features=4, min_samples_leaf=10, n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)
y_pred = rf_regressor.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)
```

```
RMSE: 8.937342911776064
```

R-Squared

In [67]:

```
# calculate the R-squared score
r2 = r2_score(y_test, y_pred)

print("R-squared score:", r2)
```

```
R-squared score: 0.13920855556765255
```

MODEL BUILDING

Scenario1 – age prediction

- Age Prediction Using Regression

Percentage population distribution

Percentage population distribution

on the train set and test set, which lies between +/- 25% of the actual and predicted value. Let the actual age be A and the predicted age be P.) The error between the actual age and the predicted age is given by the following formula:

$(A-PA)\times 100$

In [68]:

```python
# assume y_test contains the actual age and y_pred contains the predicted age
error = (y_test - y_pred) * 100 / y_test
within_25pct = ((error >= -25) & (error <= 25)).sum() / len(error) * 100

print(f"Percentage of population within +/- 25% of actual age: {within_25pct:.2f}%")

```

```
Percentage of population within +/- 25% of actual age: 65.42%
```

MODEL BUILDING

Scenario1 – age prediction

- Age Prediction Using Regression

Age Prediction Results

Age Prediction Results

In [69]:
```
y_pred = np.round(y_pred).astype(int)
y_pred
```

Out[69]:
```
array([34, 32, 34, ..., 31, 33, 35])
```

MODEL BUILDING

Scenario1 – age prediction

- Age Prediction Using Regression

Saving the model for future use

Saving the Model for future use

In [70]:

```
# # Save the model as a pickle file
with open('scenario1_age_model.pkl', 'wb') as file:
    pickle.dump(rf_regressor, file)
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – gender prediction

- c. Final data preparation and train-test-split

c. Final data preprocessing & train-test split

In [52]:
```
train_test_split = pd.read_csv("s3://Ad_Campaign_Recommender_deepaksinghpanwar/train_test_split.csv")
train_test_split.head()
```

Out[52]:

	device_id	gender	age	group	train_test_flag
0	-7548291590301750000	M	33	M32+	train
1	6943568600617760000	M	37	M32+	train
2	5441349705980020000	M	40	M32+	train
3	-5393876656119450000	M	33	M32+	train
4	4543988487649880000	M	53	M32+	train

In [53]:
```
train_device_ids = train_test_split[train_test_split['train_test_flag'] == 'train']['device_id'].unique()
test_device_ids = train_test_split[train_test_split['train_test_flag'] == 'test']['device_id'].unique()
```

In [54]:
```
# Create dataframe with categorical values for the gender
le = LabelEncoder()
train_mobile_brand['gender'] = le.fit_transform(train_mobile_brand['gender'])
train_mobile_brand
```

Out[54]:

	device_id	gender	age	group_train	phone_brand	device_model
0	-7548291590301750000	1	33	M32+	Huawei	è□£è€€3C
1	6943568600617760000	1	37	M32+	Xiaomi	xnote
2	5441349705980020000	1	40	M32+	OPPO	R7s
3	-5393876656119450000	1	33	M32+	Xiaomi	MI 4
4	4543988487649880000	1	53	M32+	samsung	Galaxy S4
...	...	...	...	...	...	...
74835	-8270585312108800000	0	32	F25-32	OPPO	U707T
74836	9140950698473710000	1	41	M32+	Huawei	Mate 8

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – gender prediction

- c. Final data preparation and train-test-split

In [55]:
```
# Convert the phone_brand and device_model columns to categorical features
train_mobile_brand = pd.concat([train_mobile_brand, pd.get_dummies(train_mobile_brand["phone_brand"], prefix="brand")], axis
train_mobile_brand = pd.concat([train_mobile_brand, pd.get_dummies(train_mobile_brand["device_model"], prefix="model")], axi
# Drop the original phone_brand and device_model columns
train_mobile_brand = train_mobile_brand.drop(["phone_brand", "device_model"], axis=1)
```

In [56]:
```
train_mobile_brand
```

Out[56]:

	device_id	gender	age	group_train	brand_AUX	brand_Bacardi	brand_Bifer	brand_CUBE	brand_Changhong	brand_Cong	...	model_é»„é—å£«é□'æ
0	-7548291590301750000	1	33	M32+	0	0	0	0	0	0	...	
1	6943568600617760000	1	37	M32+	0	0	0	0	0	0	...	
2	5441349705980020000	1	40	M32+	0	0	0	0	0	0	...	
3	-5393876656119450000	1	33	M32+	0	0	0	0	0	0	...	
4	4543988487649880000	1	53	M32+	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
74835	-8270585312108800000	0	32	F25-32	0	0	0	0	0	0	...	
74836	9140950698473710000	1	41	M32+	0	0	0	0	0	0	...	
74837	-5051737733034250000	1	25	M25-32	0	0	0	0	0	0	...	
74838	-6901678500015010000	0	20	F0-24	0	0	0	0	0	0	...	
74839	6076451050607320000	1	21	M0-24	0	0	0	0	0	0	...	

74840 rows × 1539 columns

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – gender prediction

- c. Final data preparation and train-test-split

In [57]:
```
# Split the dataset into train and test based on the device ID mapping
X_train = train_mobile_brand[train_mobile_brand['device_id'].isin(train_device_ids)]
X_test = train_mobile_brand[train_mobile_brand['device_id'].isin(test_device_ids)]
```

In [58]:
```
df_target_label = train_mobile_brand['gender']
```

In [59]:
```
# Split the dataset into train and test based on the device ID mapping
y_train = df_target_label[train_mobile_brand['device_id'].isin(train_device_ids)]
y_test = df_target_label[train_mobile_brand['device_id'].isin(test_device_ids)]
```

In [60]:
```
len(y_train)
```

Out[60]: 58705

In [61]:
```
len(y_test)
```

Out[61]: 16135

In [62]:
```
X_train = X_train.drop(["gender", "age","group_train"], axis=1)
X_test = X_test.drop(["gender", "age","group_train"], axis=1)
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – gender prediction

- c. Final data preparation and train-test-split

In [63]:
```
X_train
```

Out[63]:

	device_id	brand_AUX	brand_Bacardi	brand_Bifer	brand_CUBE	brand_Changhong	brand_Cong	brand_Coolpad	brand_Ctyon	brand_Daq
0	-7548291590301750000	0	0	0	0	0	0	0	0	0
1	6943568600617760000	0	0	0	0	0	0	0	0	0
2	5441349705980020000	0	0	0	0	0	0	0	0	0
3	-5393876656119450000	0	0	0	0	0	0	0	0	0
4	4543988487649880000	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
64555	-1439729875487580000	0	0	0	0	0	0	0	0	0
64556	-6052522297875340000	0	0	0	0	0	0	0	0	0
64557	-3585655385248180000	0	0	0	0	0	0	0	0	0
64558	-5933719272299020000	0	0	0	0	0	0	1	0	0
64559	6656679857451020000	0	0	0	0	0	0	0	0	0

58705 rows × 1536 columns

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – gender prediction

- c. Final data preparation and train-test-split

In [64]:
```
X_test
```

Out[64]:

	device_id	brand_AUX	brand_Bacardi	brand_Bifer	brand_CUBE	brand_Changhong	brand_Cong	brand_Coolpad	brand_Ctyon	brand_Daq
17545	2948104315232910000	0	0	0	0	0	0	0	0	0
17546	8231243155939480000	0	0	0	0	0	0	0	0	0
17547	-3994292212856080000	0	0	0	0	0	0	0	0	0
17548	7217910398487470000	0	0	0	0	0	0	0	0	0
17549	8642523170587800000	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
74835	-8270585312108800000	0	0	0	0	0	0	0	0	0
74836	9140950698473710000	0	0	0	0	0	0	0	0	0
74837	-5051737733034250000	0	0	0	0	0	0	0	0	0
74838	-6901678500015010000	0	0	0	0	0	0	0	0	0
74839	6076451050607320000	0	0	0	0	0	0	0	0	0

16135 rows × 1536 columns

MODEL BUILDING

Scenario2 – gender prediction

- a. Hyperparameters tuning for various Algorithms

```
hyperparameters = {
    'LogisticRegression': {
        'algorithm':LogisticRegression(),
        'parameters':{
            'penalty':['l2', 'elasticnet', 'none'],
            'C' : [0.01, 0.1, 1, 10, 100],
            'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
        }
    },
    'Random_Forest_Classifier': {
        'algorithm': RandomForestClassifier(),
        'parameters': {
            'n_estimators': [50,100, 150, 200,250,300],
            'max_depth': [10, 20, 30],
            'min_samples_split':[2, 4, 8],
            'max_features': [4, 6, 8, 10],
            'min_samples_leaf': [1, 2, 4,6, 8, 10],
            'max_features': ['auto', 'sqrt', 'log2', None]
         }
    },

     'XGBClassifier': {
        'algorithm': XGBClassifier(),
        'parameters' : {
            'min_child_weight': [1, 5, 10],
            'gamma': [0.5, 1, 1.5, 2, 5],
            'subsample': [0.6, 0.8, 1.0],
            #'colsample_by_tree': [0.6, 0.8, 1.0],
            'max_depth': [3, 4, 5],
            'n_estimators': range(60, 360, 40),
            'learning_rate': [0.1, 0.01, 0.05]
         }
    }
}
```

MODEL BUILDING

Scenario2 – gender prediction

- a. Hyperparameters tuning for various Algorithms

In [63]:

```
# get the model hyperparameters from the dictionary and execute one by one for finding best hyper parameters
for hyperparam in hyperparameters.values():
    algorithm = hyperparam.get('algorithm')
    parameters = hyperparam.get('parameters')
    grid_search_cv_model = GridSearchCV(estimator=algorithm, param_grid=parameters, cv=3)
    grid_search_cv_model.fit(X_train, y_train)
    print('Algorithm: ' + str(algorithm))
    print('Optimized Parameters: ' + str(grid_search_cv_model.best_params_))
    print("====================================")
```

```
Algorithm: LogisticRegression()
Optimized Parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
====================================
Algorithm: RandomForestClassifier()
Optimized Parameters: {'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
200}
====================================
Algorithm: XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
Optimized Parameters: {'gamma': 1.5, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 340, 'subsamp
le': 0.6}
====================================
```

MODEL BUILDING

Scenario2 – gender prediction

- a. Stacking

In [69]:
```
# 1st set of models
lr_gender = LogisticRegression(random_state=45,C=0.01, penalty='l2', solver='newton-cg')
rfc_gender = RandomForestClassifier(random_state=45,max_depth=10, max_features='auto', min_samples_leaf=1, min_samples_split
xgb_gender = XGBClassifier(random_state=45,gamma=1.5, learning_rate=0.1, max_depth=3, min_child_weight=5, n_estimators=140,
stacking_gender = StackingCVClassifier(classifiers=[lr_gender, rfc_gender], meta_classifier=xgb_gender, use_probas=True, cv=
```

In [70]:
```
# Do CV
for clf, label in zip([lr_gender, rfc_gender, stacking_gender],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, X_train, y_train, cv=5,scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

```

```
Accuracy: 0.64 (+/- 0.00) [lr]
Accuracy: 0.64 (+/- 0.00) [Random Forest]
Accuracy: 0.65 (+/- 0.00) [StackingClassifier]
```

MODEL BUILDING

Scenario2 – gender prediction

- Accuracy

b. Gender Prediction

Accuracy

In [79]:

```
scores = model_selection.cross_val_score(stacking_gender, X_train, y_train, cv=5,scoring='accuracy')
print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), 'StackingClassifier'))
```

```
Accuracy: 0.64 (+/- 0.00) [StackingClassifier]
```

MODEL BUILDING

Scenario2 – gender prediction

- Confusion matrix (F1 Score, Precision, Recall)

Confusion matrix (F1 Score, Precision, Recall)

In [78]:

```
# obtain predicted labels on test data

# fit the model to the training data
stacking_gender.fit(X_train, y_train)

y_pred = stacking_gender.predict(X_test)

# compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# compute F1 score
f1 = f1_score(y_test, y_pred)

# compute precision score
precision = precision_score(y_test, y_pred)

# compute recall score
recall = recall_score(y_test, y_pred)

# print results
print("Confusion Matrix:\n", cm)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
 [[  362  5384]
 [  348 10041]]
F1 Score: 0.7779499496397304
Precision: 0.6509562398703403
Recall: 0.9665030320531331
```

MODEL BUILDING

Scenario2 – gender prediction

- Confusion matrix (F1 Score, Precision, Recall)

Confusion matrix (F1 Score, Precision, Recall)

In [78]:

```
# obtain predicted labels on test data

# fit the model to the training data
stacking_gender.fit(X_train, y_train)

y_pred = stacking_gender.predict(X_test)

# compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# compute F1 score
f1 = f1_score(y_test, y_pred)

# compute precision score
precision = precision_score(y_test, y_pred)

# compute recall score
recall = recall_score(y_test, y_pred)

# print results
print("Confusion Matrix:\n", cm)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
 [[  362  5384]
 [  348 10041]]
F1 Score: 0.7779499496397304
Precision: 0.6509562398703403
Recall: 0.9665030320531331
```

MODEL BUILDING

Scenario2 – gender prediction

- ROC curve and AUC

ROC curve and AUC

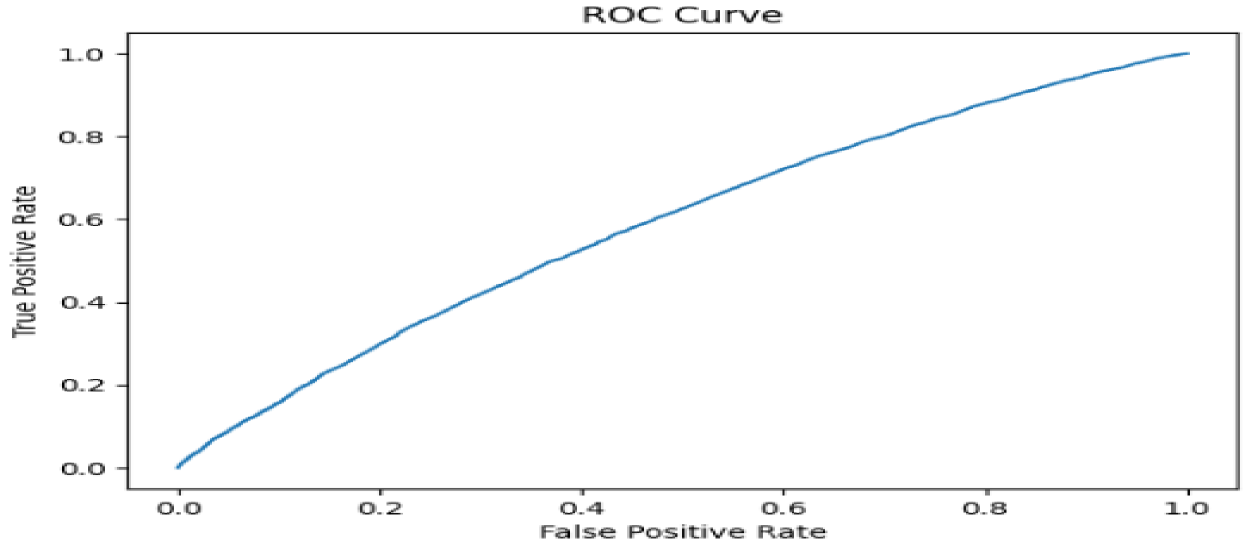
In [80]:

```
# predict probabilities for positive class
y_probs = stacking_gender.predict_proba(X_test)[:, 1]

# calculate false positive rate, true positive rate, and threshold values
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# plot ROC curve
plt.plot(fpr, tpr)
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

# calculate AUC score
auc = roc_auc_score(y_test, y_probs)
print('AUC Score:', auc)
```




```
AUC Score: 0.5893460368015556
```

MODEL BUILDING

Scenario2 – gender prediction

- Kolmogorov–Smirnov (KS) statistic

Kolmogorov–Smirnov (KS) statistic

In [81]:

```python
# obtain predicted probabilities for the test data
y_pred_prob = stacking_gender.predict_proba(X_test)

# separate the test data into positive and negative classes
pos_idx = np.where(y_test == 1)[0]
neg_idx = np.where(y_test == 0)[0]

# sort the predicted probabilities in descending order
pos_probs = y_pred_prob[pos_idx][:, 1]
neg_probs = y_pred_prob[neg_idx][:, 1]
pos_probs_sorted = np.sort(pos_probs)[::-1]
neg_probs_sorted = np.sort(neg_probs)[::-1]

min_len = min(len(pos_probs_sorted), len(neg_probs_sorted))
pos_cdf = np.cumsum(pos_probs_sorted[:min_len]) / np.sum(pos_probs_sorted[:min_len])
neg_cdf = np.cumsum(neg_probs_sorted[:min_len]) / np.sum(neg_probs_sorted[:min_len])

# compute the KS statistic as the maximum absolute difference between the CDFs
ks = np.max(np.abs(pos_cdf - neg_cdf))
ks
```

Out[81]: 0.032200575

MODEL BUILDING

Scenario2 – gender prediction

- The probability bands identified through the KS table for the top 3 and bottom 3 deciles on the train data set

The probability bands identified through the KS table for the top 3 and bottom 3 deciles on the train data set

In [82]:

```python

# get predicted probabilities for the positive class
y_proba_train = stacking_gender.predict_proba(X_train)[:, 1]

# calculate the false positive rate, true positive rate, and thresholds for the ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_proba_train)

# reset the index of y_train
y_train_reset = y_train.reset_index(drop=True)

# sort the predicted probabilities in descending order
idx = y_proba_train.argsort()[::-1]
y_proba_train_sorted = y_proba_train[idx]
y_train_sorted = y_train_reset[idx]

# calculate the number of samples and positive samples in each decile
n_samples = len(y_train_sorted)
n_pos_samples = y_train_sorted.sum()
n_neg_samples = n_samples - n_pos_samples
decile_size = n_samples // 10
pos_counts = np.cumsum(y_train_sorted)
neg_counts = np.arange(1, n_samples+1) - pos_counts

# calculate the CDFs for the positive and negative classes in each decile
pos_cdfs = pos_counts / n_pos_samples
neg_cdfs = neg_counts / n_neg_samples

# calculate the KS statistic and the threshold for the top 3 and bottom 3 deciles
ks_top = pos_cdfs[:decile_size].max() - neg_cdfs[:decile_size].max()
ks_bottom = pos_cdfs[-decile_size:].max() - neg_cdfs[-decile_size:].max()
thresh_top = y_proba_train_sorted[decile_size]
thresh_bottom = y_proba_train_sorted[-decile_size]

# calculate the probability bands for the top 3 and bottom 3 deciles
band_top = y_proba_train_sorted[:decile_size].min(), y_proba_train_sorted[:decile_size].max()
band_bottom = y_proba_train_sorted[-decile_size:].min(), y_proba_train_sorted[-decile_size:].max()

print(f"KS statistic for top 3 deciles: {ks_top:.3f}")
print(f"KS statistic for bottom 3 deciles: {ks_bottom:.3f}")
print(f"Probability band for top 3 deciles: {band_top}")
print(f"Probability band for bottom 3 deciles: {band_bottom}")

```

```
KS statistic for top 3 deciles: 0.090
KS statistic for bottom 3 deciles: 0.000
Probability band for top 3 deciles: (0.7265414, 0.8766709)
Probability band for bottom 3 deciles: (0.2090812, 0.54918253)
```

MODEL BUILDING

Scenario2 – gender prediction

- Gender Predictions Result

Gender Predictions Result

In [83]:

```python
gender_map = {1: "M", 0: "F"}
y_pred_gender = [gender_map[y] for y in y_pred]
print(y_pred_gender)
```

```
['F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'F', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
```

MODEL BUILDING

Scenario2 – gender prediction

- Saving the Model for future use

Saving the Model for future use

In [ ]:

```
# Save the model as a pickle file
with open('scenario2_gender_model.pkl', 'wb') as file:
    pickle.dump(stacking_gender, file)
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – age prediction

- c. Final data preparation and train-test-split

c. Final data preparation and train-test-split

In [52]:

```
train_test_split = pd.read_csv("s3://Ad_Campaign_Recommender_deepaksinghpanwar/train_test_split.csv")
train_test_split.head()
```

Out[52]:

	device_id	gender	age	group	train_test_flag
0	-7548291590301750000	M	33	M32+	train
1	6943568600617760000	M	37	M32+	train
2	5441349705980020000	M	40	M32+	train
3	-5393876656119450000	M	33	M32+	train
4	4543988487649880000	M	53	M32+	train

In [53]:

```
train_device_ids = train_test_split[train_test_split['train_test_flag'] == 'train']['device_id'].unique()
test_device_ids = train_test_split[train_test_split['train_test_flag'] == 'test']['device_id'].unique()
```

In [54]:

```
# Create dataframe with categorical values for gender
le = LabelEncoder()
train_mobile_brand['gender'] = le.fit_transform(train_mobile_brand['gender'])
train_mobile_brand
```

Out[54]:

	device_id	gender	age	group_train	phone_brand	device_model
0	-7548291590301750000	1	33	M32+	Huawei	è□£è€€3C
1	6943568600617760000	1	37	M32+	Xiaomi	xnote
2	5441349705980020000	1	40	M32+	OPPO	R7s
3	-5393876656119450000	1	33	M32+	Xiaomi	MI 4
4	4543988487649880000	1	53	M32+	samsung	Galaxy S4
...	...	...	...	...	...	...

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – age prediction

- c. Final data preparation and train-test-split

In [55]:
```
# Convert the phone_brand and device_model columns to categorical features
train_mobile_brand = pd.concat([train_mobile_brand, pd.get_dummies(train_mobile_brand["phone_brand"], prefix="brand")], axis
train_mobile_brand = pd.concat([train_mobile_brand, pd.get_dummies(train_mobile_brand["device_model"], prefix="model")], axi
# Drop the original phone_brand and device_model columns
train_mobile_brand = train_mobile_brand.drop(["phone_brand", "device_model"], axis=1)
```

In [56]:
```
train_mobile_brand
```

Out[56]:

	device_id	gender	age	group_train	brand_AUX	brand_Bacardi	brand_Bifer	brand_CUBE	brand_Changhong	brand_Cong	...	model_é»„é—å£«é□'æ"
0	-7548291590301750000	1	33	M32+	0	0	0	0	0	0	...	
1	6943568600617760000	1	37	M32+	0	0	0	0	0	0	...	
2	5441349705980020000	1	40	M32+	0	0	0	0	0	0	...	
3	-5393876656119450000	1	33	M32+	0	0	0	0	0	0	...	
4	4543988487649880000	1	53	M32+	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
74835	-8270585312108800000	0	32	F25-32	0	0	0	0	0	0	...	
74836	9140950698473710000	1	41	M32+	0	0	0	0	0	0	...	
74837	-5051737733034250000	1	25	M25-32	0	0	0	0	0	0	...	
74838	-6901678500015010000	0	20	F0-24	0	0	0	0	0	0	...	
74839	6076451050607320000	1	21	M0-24	0	0	0	0	0	0	...	

74840 rows × 1539 columns

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – age prediction

- c. Final data preparation and train-test-split

```
In [56]:
# Split the dataset into train and test based on the device ID mapping
X_train = train_mobile_brand[train_mobile_brand['device_id'].isin(train_device_ids)]
X_test = train_mobile_brand[train_mobile_brand['device_id'].isin(test_device_ids)]

In [57]:
df_target_label = train_mobile_brand['age']

In [58]:
# Split the dataset into train and test based on the device ID mapping
y_train = df_target_label[train_mobile_brand['device_id'].isin(train_device_ids)]
y_test = df_target_label[train_mobile_brand['device_id'].isin(test_device_ids)]

In [59]:
y_train

Out[59]:
0        33
1        37
2        40
3        33
4        53
         ..
64555    14
64556    17
64557    24
64558    21
64559    29
Name: age, Length: 58705, dtype: int64

In [60]:
y_test

Out[60]:
17545    65
17546    47
17547    31
17548    29
17549    31
         ..
74835    32
74836    41
74837    25
74838    20
74839    21
Name: age, Length: 16135, dtype: int64
```

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – age prediction

- c. Final data preparation and train-test-split

In [61]:
```
X_train = X_train.drop(["gender", "age","group_train"], axis=1)
X_test = X_test.drop(["gender", "age","group_train"], axis=1)
```

In [62]:
```
X_train
```

Out[62]:

	device_id	brand_AUX	brand_Bacardi	brand_Bifer	brand_CUBE	brand_Changhong	brand_Cong	brand_Coolpad	brand_Ctyon	brand_Daq
0	-7548291590301750000	0	0	0	0	0	0	0	0	0
1	6943568600617760000	0	0	0	0	0	0	0	0	0
2	5441349705980020000	0	0	0	0	0	0	0	0	0
3	-5393876656119450000	0	0	0	0	0	0	0	0	0
4	4543988487649880000	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
64555	-1439729875487580000	0	0	0	0	0	0	0	0	0
64556	-6052522297875340000	0	0	0	0	0	0	0	0	0
64557	-3585655385248180000	0	0	0	0	0	0	0	0	0
64558	-5933719272299020000	0	0	0	0	0	0	1	0	0
64559	6656679857451020000	0	0	0	0	0	0	0	0	0

58705 rows × 1536 columns

4. ADVANCED VISUALIZATION AND CLUSTERING

Scenario2 – age prediction

- c. Final data preparation and train-test-split

In [63]:
```
1 X_test
```

Out[63]:

	device_id	brand_AUX	brand_Bacardi	brand_Bifer	brand_CUBE	brand_Changhong	brand_Cong	brand_Coolpad	brand_Ctyon	brand_Daq
17545	2948104315232910000	0	0	0	0	0	0	0	0	0
17546	8231243155939480000	0	0	0	0	0	0	0	0	0
17547	-3994292212856080000	0	0	0	0	0	0	0	0	0
17548	7217910398487470000	0	0	0	0	0	0	0	0	0
17549	8642523170587800000	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
74835	-8270585312108800000	0	0	0	0	0	0	0	0	0
74836	9140950698473710000	0	0	0	0	0	0	0	0	0
74837	-5051737733034250000	0	0	0	0	0	0	0	0	0
74838	-6901678500015010000	0	0	0	0	0	0	0	0	0
74839	6076451050607320000	0	0	0	0	0	0	0	0	0

16135 rows × 1536 columns

MODEL BUILDING

Scenario2 – age prediction

- Random Forest Regressor is a popular machine learning algorithm for solving regression problems. Here are some of the reasons why I have used it for age prediction:
- Handles non-linear data: Random Forest Regressor can handle non-linear and complex data distributions. It can capture the non-linear relationships between the input features and the target variable (age in this case) much better than linear models.
- Robust to noise and outliers: Random Forest Regressor is less sensitive to noise and outliers in the data. It can handle missing or corrupted values by imputing missing values or using surrogate splits.
- Feature Importance: Random Forest Regressor can be used to determine the relative importance of each feature in predicting the target variable. This can help in identifying the most important factors that contribute to the age prediction.
- Scalability: Random Forest Regressor is highly scalable and can handle large datasets with thousands of features.
- Ensemble Learning: Random Forest Regressor uses an ensemble of decision trees, which are trained on different subsets of the data. This results in a more robust and accurate prediction compared to a single decision tree.
- Overall, Random Forest Regressor is a powerful and flexible algorithm that can handle a wide range of data distributions and can be used for accurate age prediction.

MODEL BUILDING

Scenario2 – age prediction

- a. Hyperparameters tuning for various Algorithms

HyperParameters tuning for various Algorithms

In [63]:
```
hyperparameters = {
    'Random_Forest_Regression': {
        'algorithm': RandomForestRegressor(),
        'parameters': {
            'n_estimators': [50,100, 150, 200,250,300],
            'max_depth': [10, 20, 30],
            'min_samples_split':[2, 4, 8],
            'max_features': [4, 6, 8, 10],
            'min_samples_leaf': [1, 2, 4,6, 8, 10],
            'max_features': ['auto', 'sqrt', 'log2', None]
        }
    }
}
```

In [64]:
```
# get the model hyperparameters from the dictionary and execute one by one for finding best hyper parameters
for hyperparam in hyperparameters.values():
    algorithm = hyperparam.get('algorithm')
    parameters = hyperparam.get('parameters')
    grid_search_cv_model = GridSearchCV(estimator=algorithm, param_grid=parameters, cv=3)
    grid_search_cv_model.fit(X_train, y_train)
    print('Algorithm: ' + str(algorithm))
    print('Optimized Parameters: ' + str(grid_search_cv_model.best_params_))
    print("====================================")
```

MODEL BUILDING

Scenario2 – age prediction

- Age Prediction Using Regression
    - RMSE and R-Squared

1. Age Prediction Using Regression

RMSE

In [67]:
```
rf_regressor = RandomForestRegressor(max_depth=10, max_features=4, min_samples_leaf=10, n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)
y_pred = rf_regressor.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)
```

```
RMSE: 9.842927821195607
```

R-Squared

In [68]:
```
# calculate the R-squared score
r2 = r2_score(y_test, y_pred)

print("R-squared score:", r2)
```

```
R-squared score: 0.0013832469815485693
```

MODEL BUILDING

Scenario2 – age prediction

- Age Prediction Using Regression

Percentage population distribution

Percentage population distribution

on the train set and test set, which lies between +/- 25% of the actual and predicted value. Let the actual age be A and the predicted age be P.) The error between the actual age and the predicted age is given by the following formula:

$(A-PA)\times100$

In [69]:

```
# assume y_test contains the actual age and y_pred contains the predicted age
error = (y_test - y_pred) * 100 / y_test
within_25pct = ((error >= -25) & (error <= 25)).sum() / len(error) * 100

print(f"Percentage of population within +/- 25% of actual age: {within_25pct:.2f}%")

```

```
Percentage of population within +/- 25% of actual age: 56.43%
```

MODEL BUILDING

Scenario2 – age prediction

- Age Prediction Using Regression

Age Prediction Results

Age Predicion Results

In [70]:
```
y_pred = np.round(y_pred).astype(int)
y_pred
```

Out[70]:
```
array([31, 31, 31, ..., 31, 31, 31])
```

MODEL BUILDING

Scenario2 – age prediction

- Age Prediction Using Regression

Saving the model for future use

Saving the Model for future use

In [71]:

```
# # Save the model as a pickle file
with open('scenario2_age_model.pkl', 'wb') as file:
    pickle.dump(rf_regressor, file)
```