Ad Campaign Recommender
(Task1 – Model Building)

CAPSTONE 1 – BY **SURBHI SINHA**

MS IN DATA SCIENCE

UNIVERSITY OF ARIZONA

Problem Statement

- The objective of this capstone project is to predict the demographics of a user (age and gender) using information such as app download, phone brand, and app usage behavior.
- Use-case: We must target users with "personalized" ad campaigns. This should improve their experience and open avenues for revenue generation through cross-selling and upselling.
- We have 2 Scenarios for which we need to do this prediction:
  - Scenario 1: We have all the data present (i.e., you have the latitudinal and longitudinal data, application id data, event data, and device data).
  - Scenario 2: For some device ids, you may only have the mobile phone, brand, and device data available.

Data 1 – Train Mobile Brand

This data contains the gender, age, phone brand and device model for every device ID

[3]:
```
# This is first data set which has data of all device id with the corresponding gender, age, phone brand & device model
train_mobile_brand_df = pd.read_csv("data/train_mobile_brand.csv")
```

[4]:
```
train_mobile_brand_df.head()
```

[4]:

	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

Data 2 - Train Event Data

This Data contains multiple events, timestamp and location data in form of latitude, longitude and gender & age for every device Id

[6]:
```
# This is second data set which has all the device id and the event_id and the time & location of event
train_event_data_df = pd.read_csv("data/train_event_data.csv")
```

[7]:
```
train_event_data_df.head()
```

[7]:

	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

Data 3 - App Events

This data contains the app_id for every event_id as well as the states if the app is installed or not and app is active or not

[9]:
```
# This is third data set which has the app_id for the event id and the app status if it is installed or active
app_event_df = pd.read_csv("data/app_events.csv")
```

[10]:
```
app_event_df.head()
```

[10]:

	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

Data 4 - App Event Meta Data

This data contains the metadata for every app including the label Id & category

[12]:
```
# This is 4th data set which contains the app_id and the corresponding label_id and app category
dtype = {"0": int, "1": int, "2": str}

# We are skipping two rows. Row1 contains the title and not actual data, and row 420939 has an EOF character which fails the parsing of file
app_event_meta_data_df = pd.read_csv("data/app_events_meta_data.csv", skiprows=[1, 420939], dtype=dtype)
app_event_meta_data_df.head()
```

[12]:

	app_id	label_id	category
0	7324884708820027918	251	Finance
1	-4494216993218550286	251	Finance
2	6058196446775239644	406	unknown
3	6058196446775239644	407	DS_P2P net loan
4	8694625920731541625	406	unknown

Data Cleaning – Remove garbage characters in device model

In the dataframe `train_mobile_brand_df` above we have seen that multiple device model containing garbage string such as `è £è€€3C`. We would want to filter these unwanted characters to ASCII Characters only

[14]:
```
#Function to Clean non-ASCII Characters
def cleanString(str):
    if str.isascii():
        return str

    else:
        return "".join(list(map(lambda x:x if x.isascii() else "", str)))

train_mobile_brand_df["device_model"] = train_mobile_brand_df["device_model"].apply(cleanString)
train_mobile_brand_df.head()
```

[14]:

	device_id	gender	age	group_train	phone_brand	device_model
0	-7548291590301750000	M	33	M32+	Huawei	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

Data Cleaning – Remove Duplicates after changing string cases

As next step we will try to remove any duplicated entries in this data frame by first converting all `phone_brand` & `device_model` to string upper case

[15]:
```
# Convert phone_brand & device_model to upper case
train_mobile_brand_df.phone_brand = train_mobile_brand_df.phone_brand.str.upper()
train_mobile_brand_df.device_model = train_mobile_brand_df.device_model.str.upper()

# See duplicated values in this dataframe
print("Duplicate values before cleaning:")
train_mobile_brand_df.duplicated().value_counts(dropna=False)
```

```
Duplicate values before cleaning:
```

[15]:
```
False    74646
True       194
Name: count, dtype: int64
```

[16]:
```
# Drop all duplicates
train_mobile_brand_df.drop_duplicates(subset=None, keep="first", inplace=True)

# See duplicated values in this dataframe after cleaning
print("Duplicate values after cleaning:")
train_mobile_brand_df.duplicated().value_counts(dropna=False)
```

```
Duplicate values after cleaning:
```

[16]:
```
False    74646
Name: count, dtype: int64
```

Data Cleaning – Change timestamp to date object and create more columns

in `train_event_data_df` we will first convert datetimeStamp pbject to pandas datetime

```
train_event_data_df["datetimestamp"]=pd.to_datetime(train_event_data_df["datetimestamp"])
train_event_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1266933 entries, 0 to 1266932
Data columns (total 8 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   device_id      1266933 non-null  int64
 1   gender         1266933 non-null  object
 2   age            1266933 non-null  int64
 3   group_train    1266933 non-null  object
 4   event_id       1215598 non-null  float64
 5   datetimestamp  1215598 non-null  datetime64[ns]
 6   latitude       1215598 non-null  float64
 7   longitude      1215598 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(2), object(2)
memory usage: 77.3+ MB
```

We will now create additional columns such as hour, day of week etc for EDA below

```
train_event_data_df['day_of_week'] = train_event_data_df['datetimestamp'].dt.day_name()
train_event_data_df['hour'] = train_event_data_df['datetimestamp'].dt.hour
```

Data Preparation - Age Binning

We will also add another column representing age group in 4 bins of 0-24, 25-32, 33-45. 46+. This will be used in EDA below

[19]:
```
train_event_data_df['age_group'] = pd.cut(train_event_data_df['age'], bins=[0, 24, 32, 45, float('inf')], labels=['0-24', '25-32', '33-45', '46+'])
```

[20]:
```
train_event_data_df.head()
```

[20]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude	day_of_week	hour	age_group
0	-7548291590301750000	M	33	M32+	2369465.0	2016-05-03 15:55:35	33.98	116.79	Tuesday	15.0	33-45
1	-7548291590301750000	M	33	M32+	1080869.0	2016-05-03 06:07:16	33.98	116.79	Tuesday	6.0	33-45
2	-7548291590301750000	M	33	M32+	1079338.0	2016-05-04 03:28:02	33.98	116.79	Wednesday	3.0	33-45
3	-7548291590301750000	M	33	M32+	1078881.0	2016-05-04 02:53:08	33.98	116.79	Wednesday	2.0	33-45
4	-7548291590301750000	M	33	M32+	1068711.0	2016-05-03 15:59:35	33.98	116.79	Tuesday	15.0	33-45

Data Preparation – Splitting of Data for Scenario 1 & Scenario 2

- Splitting the train_event_data into scenario 1 and scenario 2
- Scenario 1 - A user enables the location on their device while using an app. In a way, the location or event id generated against any activity will have location data.
- Scenario 2 - The user activity is unavailable, thereby leaving you with only device information.

[21]:
```
train_events_with_location = train_event_data_df.dropna(subset=['latitude', 'longitude'])
train_events_without_location = train_event_data_df[train_event_data_df['latitude'].isnull() | train_event_data_df['longitude'].isnull()]

train_events_with_location["coordinates"] = train_events_with_location[["latitude", "longitude"]].apply(lambda x: (x["latitude"], x["longitude"]), axis = 1)
```

[23]:
```
train_events_with_location.head()
```

[23]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude	day_of_week	hour	age_group	coordinates
0	-7548291590301750000	M	33	M32+	2369465.0	2016-05-03 15:55:35	33.98	116.79	Tuesday	15.0	33-45	(33.98, 116.79)
1	-7548291590301750000	M	33	M32+	1080869.0	2016-05-03 06:07:16	33.98	116.79	Tuesday	6.0	33-45	(33.98, 116.79)
2	-7548291590301750000	M	33	M32+	1079338.0	2016-05-04 03:28:02	33.98	116.79	Wednesday	3.0	33-45	(33.98, 116.79)
3	-7548291590301750000	M	33	M32+	1078881.0	2016-05-04 02:53:08	33.98	116.79	Wednesday	2.0	33-45	(33.98, 116.79)
4	-7548291590301750000	M	33	M32+	1068711.0	2016-05-03 15:59:35	33.98	116.79	Tuesday	15.0	33-45	(33.98, 116.79)

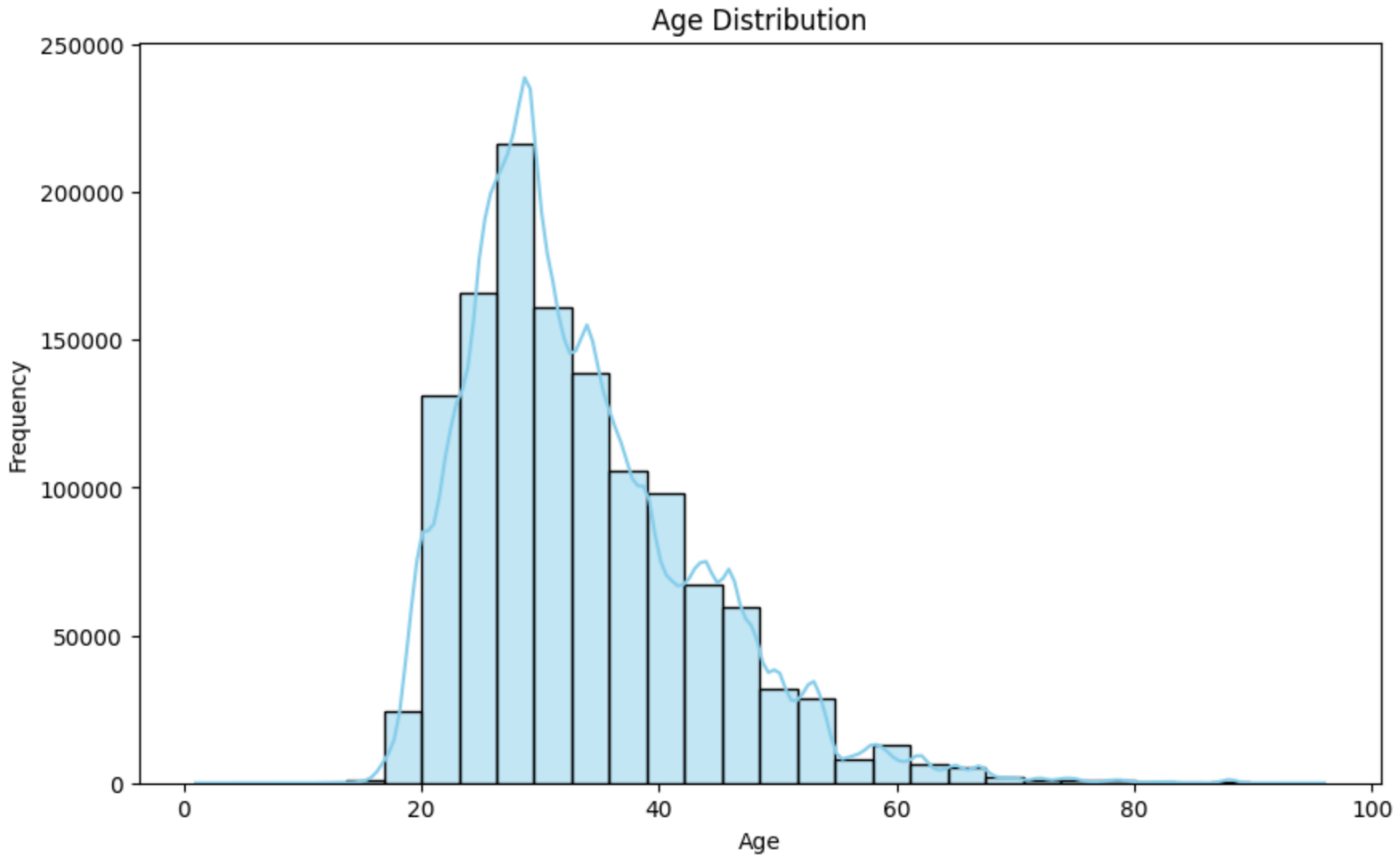
[25]:
```
train_events_without_location.head()
```

[25]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude	day_of_week	hour	age_group
1215595	398514470209561000	M	68	M32+	NaN	NaT	NaN	NaN	NaN	NaN	46+
1215596	-3073918292047050000	M	27	M25-32	NaN	NaT	NaN	NaN	NaN	NaN	25-32
1215597	5805880616488060000	M	39	M32+	NaN	NaT	NaN	NaN	NaN	NaN	33-45
1215598	-2403560729305410000	M	25	M25-32	NaN	NaT	NaN	NaN	NaN	NaN	25-32
1215599	-1889893391998300000	M	22	M0-24	NaN	NaT	NaN	NaN	NaN	NaN	0-24

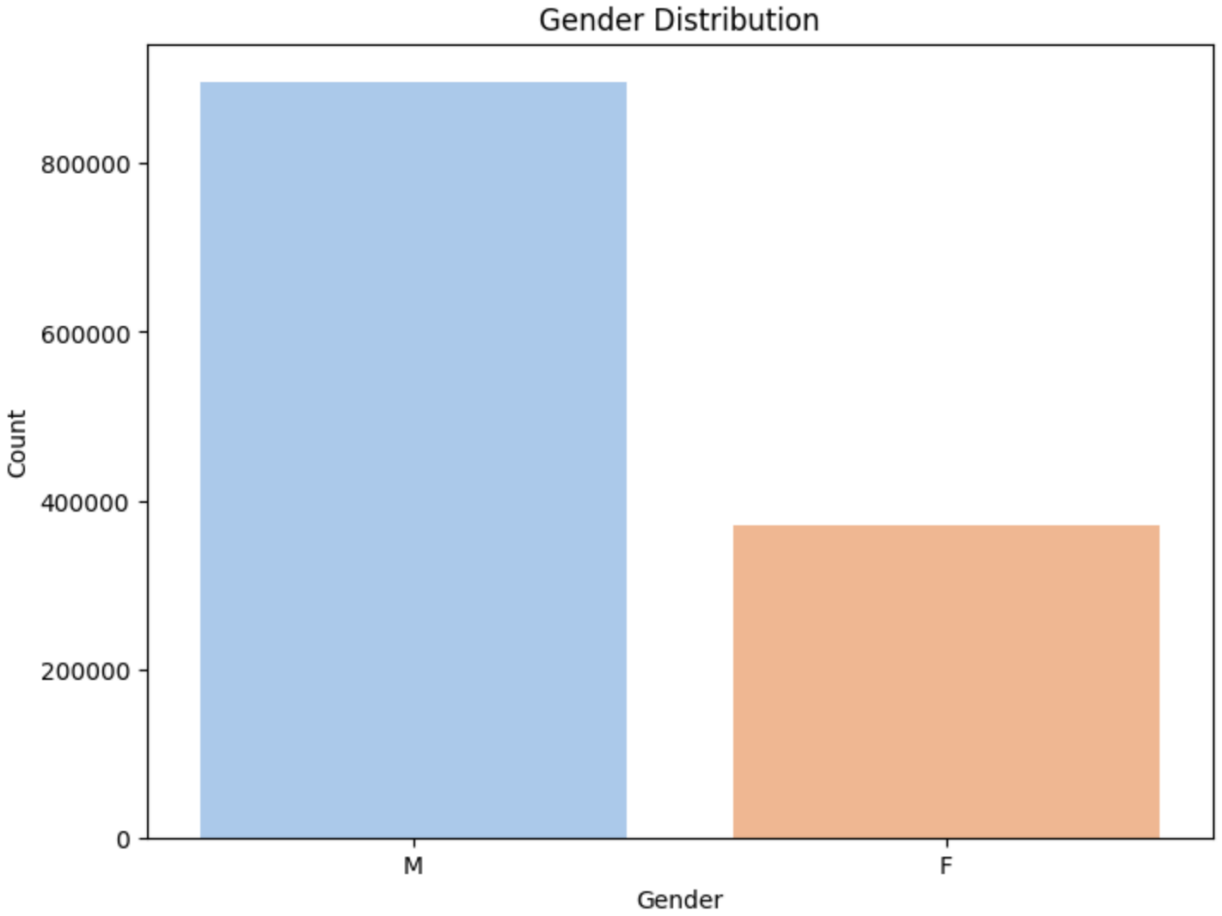
EDA – Age Distribution

Univariate Analysis of Age Distribution)



EDA - Gender distribution

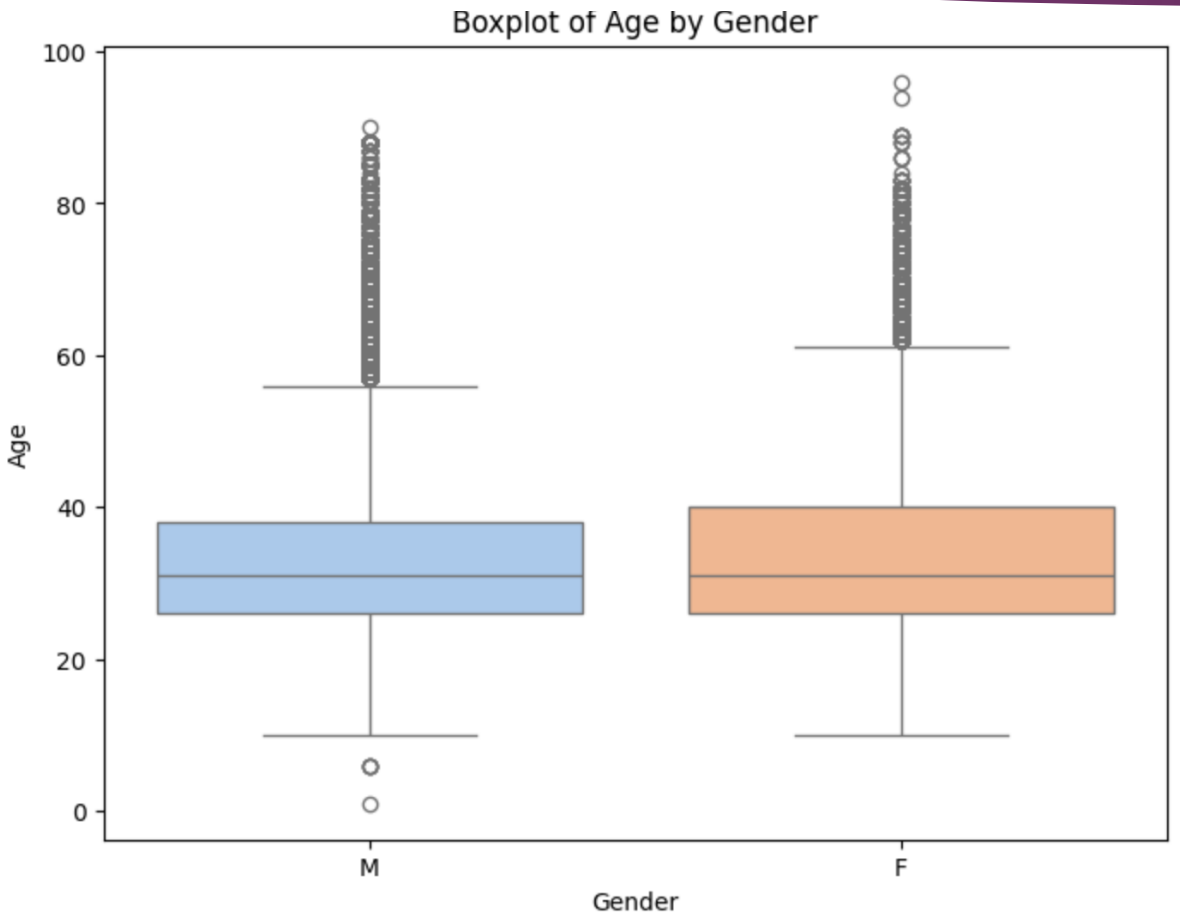
Univariate Analysis of Gender Distribution



From this plot we see that Male Population consists of roughly 75% of the over all events in the data set. So the data set is skewed for Males

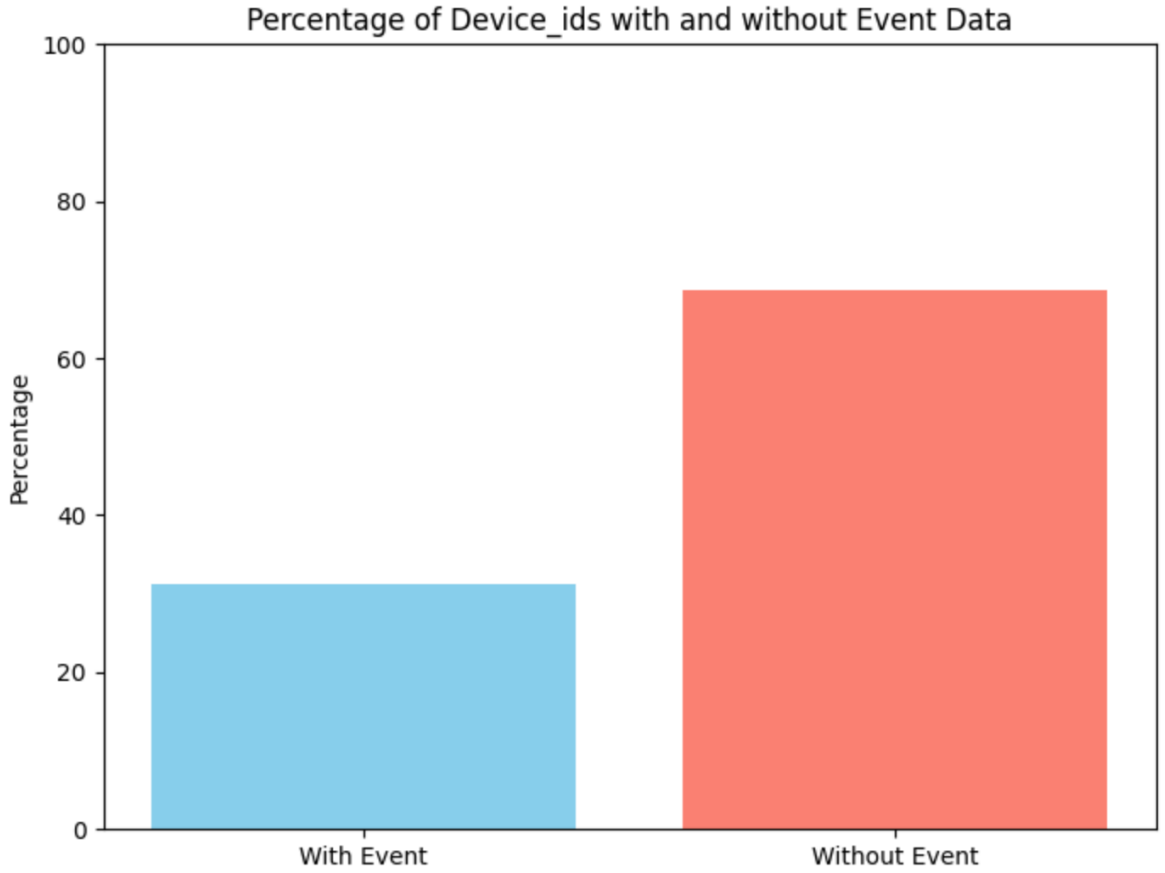
EDA – Boxplot of Age by gender

BiVariate Analysis



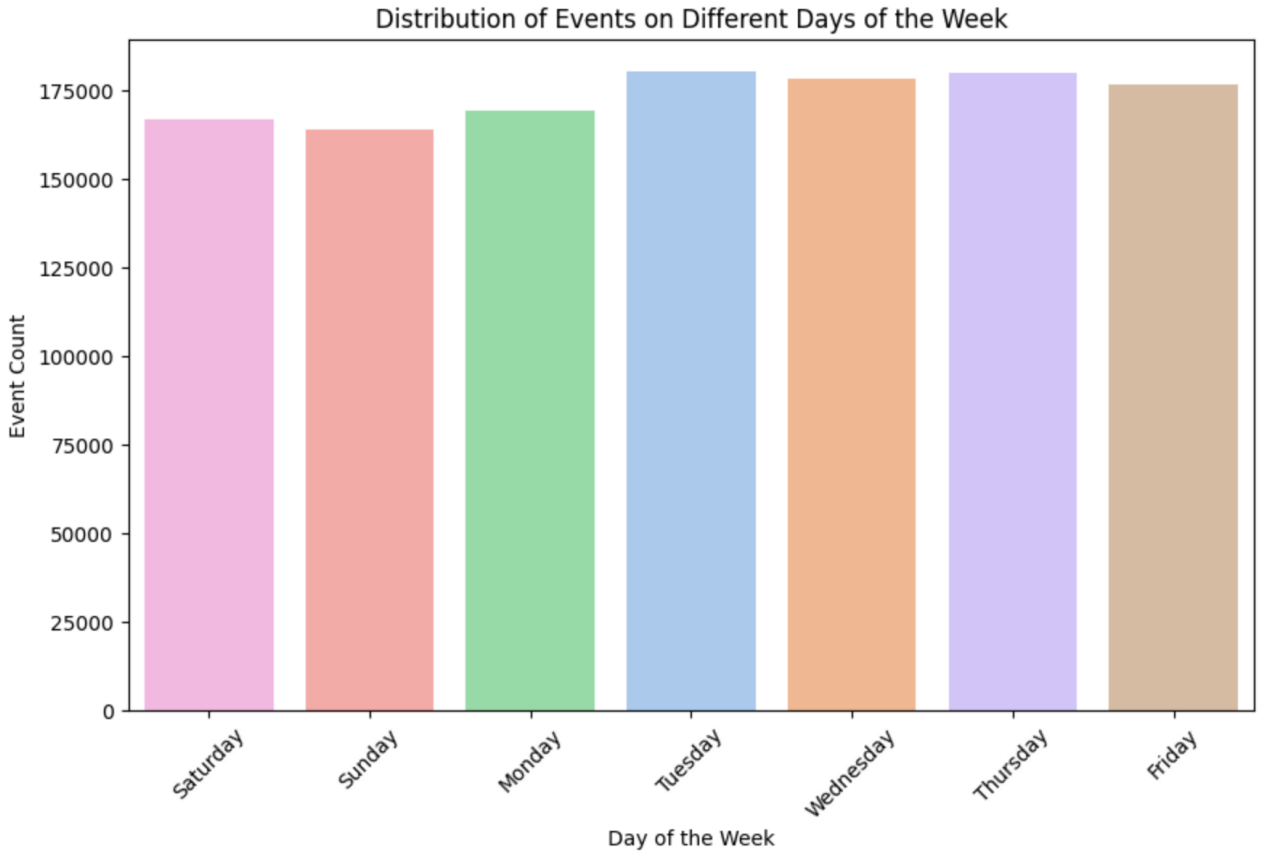
From this boxplot we see that the age distribution for Males and Females is approximately the same with same mean and similar min and max values.

EDA (Trends) - Device Ids with or without event data



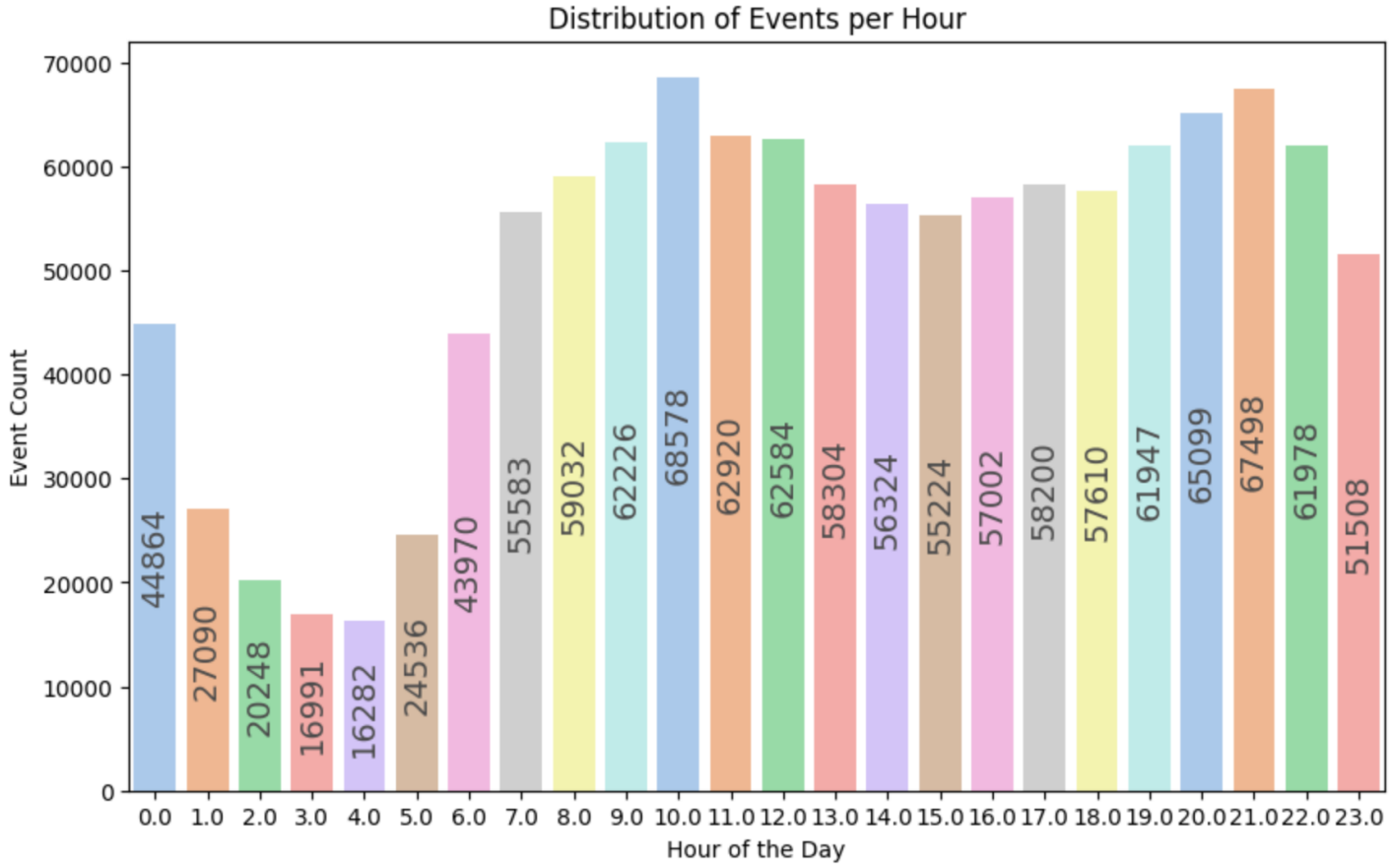
- Total % of devices with event data = **31.23%**
- Total % of devices without event data = **68.77%**
- From this graph we see that only **31%** of devices have event data getting generated along with it. while **69%** devices have no events getting generated from them.

EDA (Trends) - Distribution of Events on different days if week



We see that the usage on different days of the week mostly remains the same without much significant change. However the mobile usage is lesser on weekends as compared to week days.

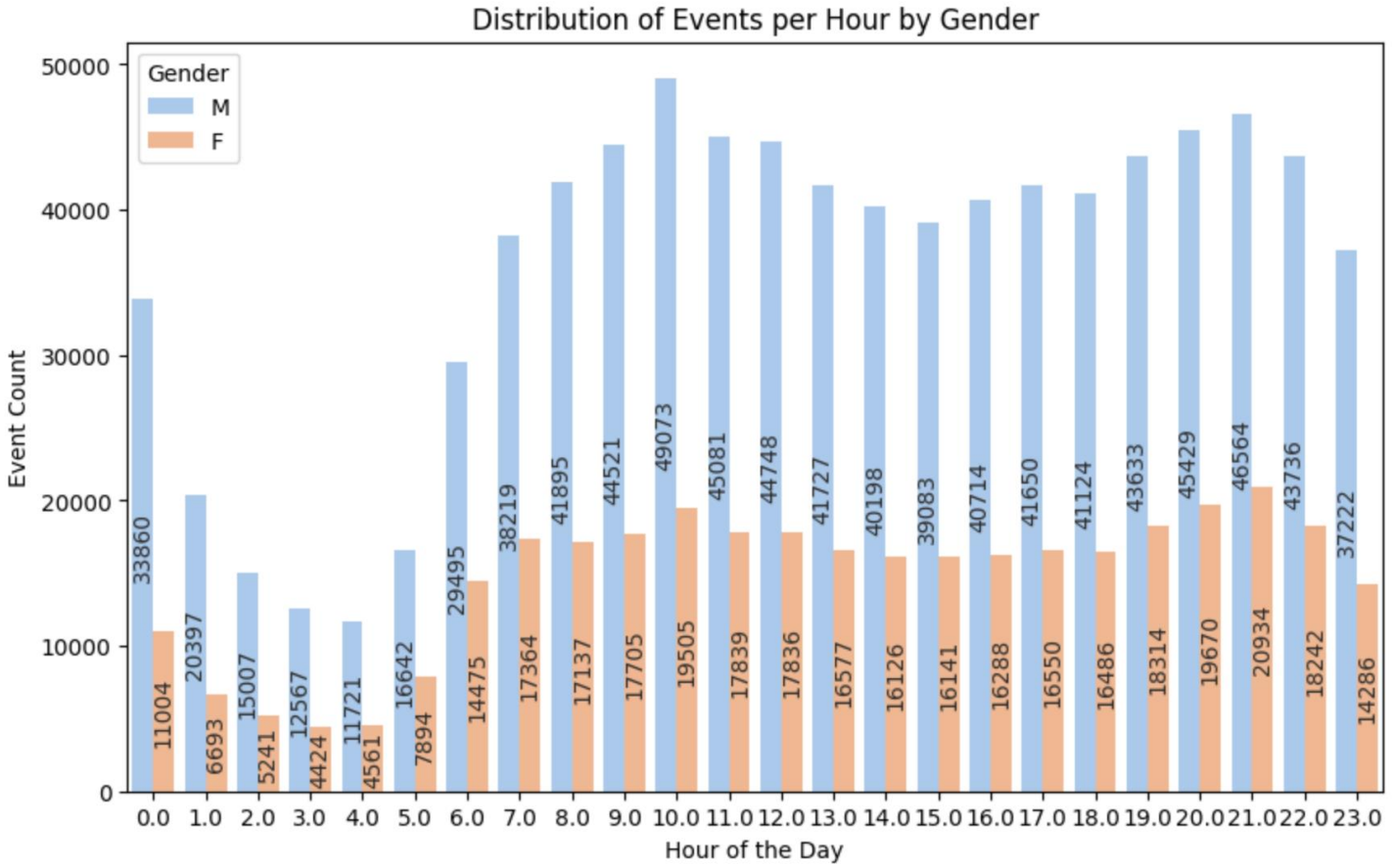
EDA (Trends) – Distribution of Events per hour (for 1 week data)



In this we see that the no of events is highest in Mornings and Evenings which could signify people use mobile phones while travelling to office.

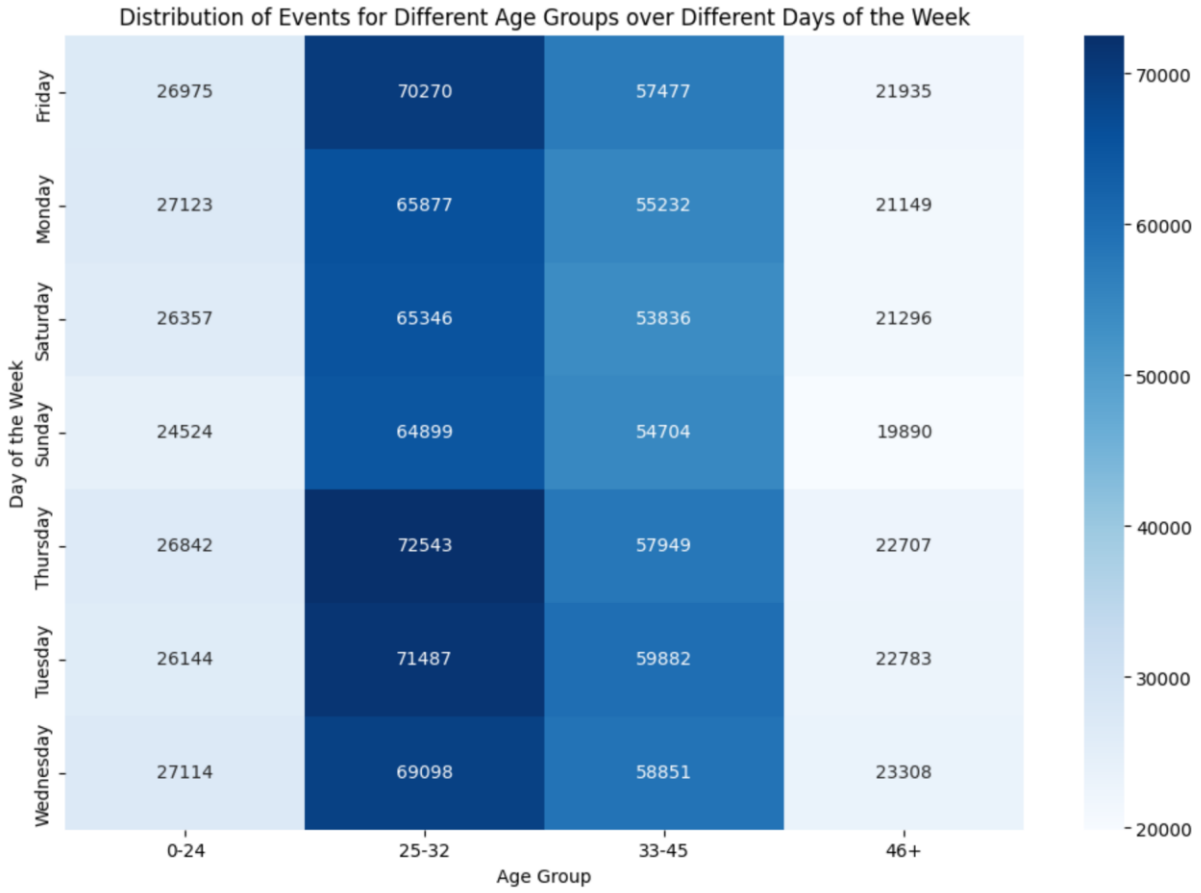
While the events drops significantly during night time when people are asleep.

EDA (Trends) – Distribution of events per hour for males & females



In this we see the trend doesn't change significantly for males vs females however for males the peak at morning and evening is more significant.

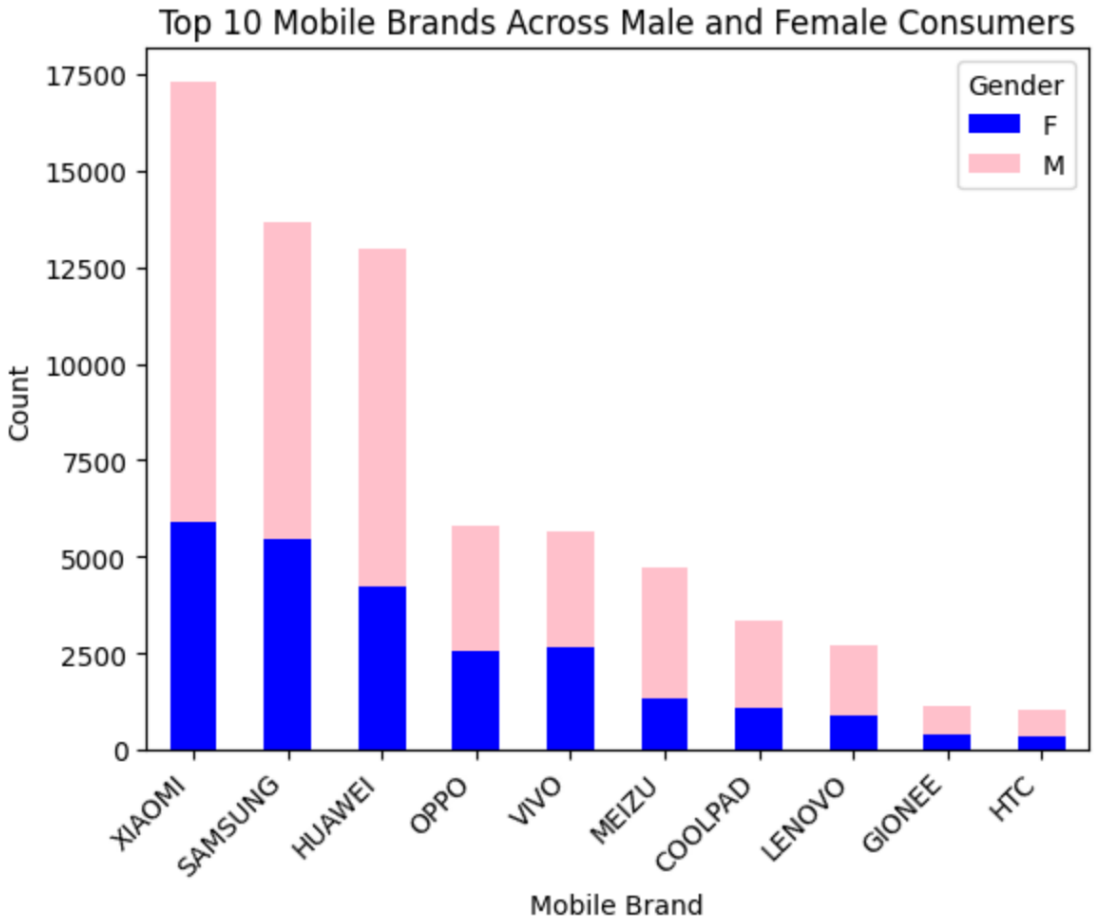
EDA (Trends) – Distribution of events for different age group over different days of week



App usage is highest for the 25-32 age group, followed by the 33-45 age group.

Younger and older age groups show lower usage and exhibit less variance in their data distribution throughout the week.

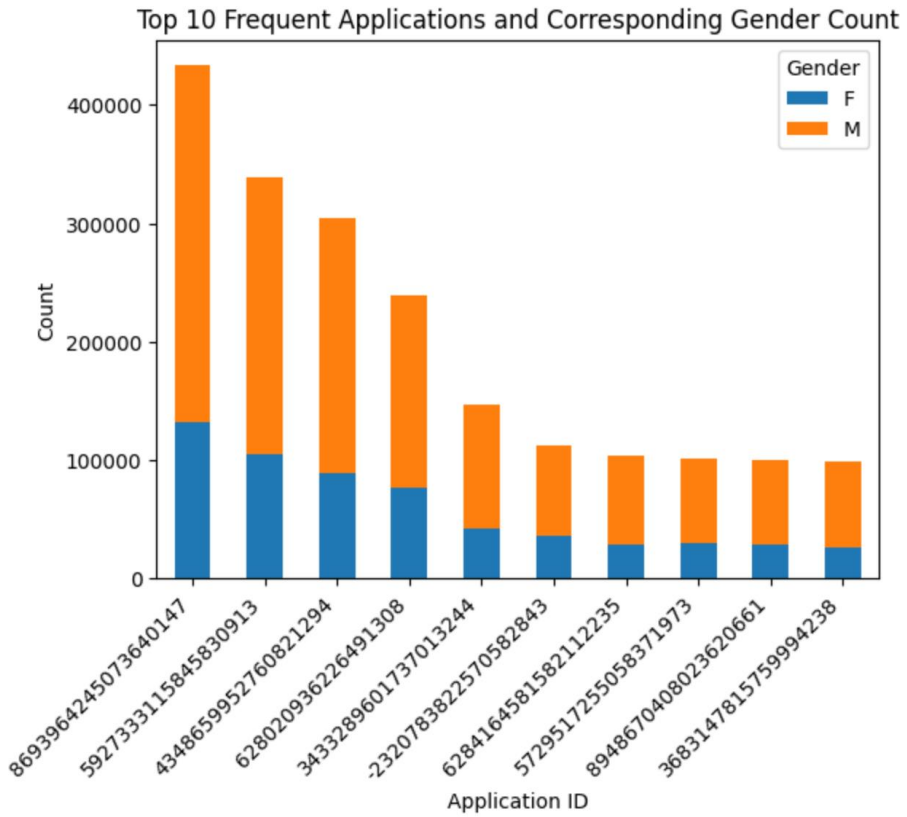
EDA (Mobile brands): Top 10 Mobile brand for males & females



From this we see that the top most used brand for males and females is **XIAOMI** followed by **Samsung** & **Huawei.**

After the top 3, and next 5, the brand uses decreases exponentially.

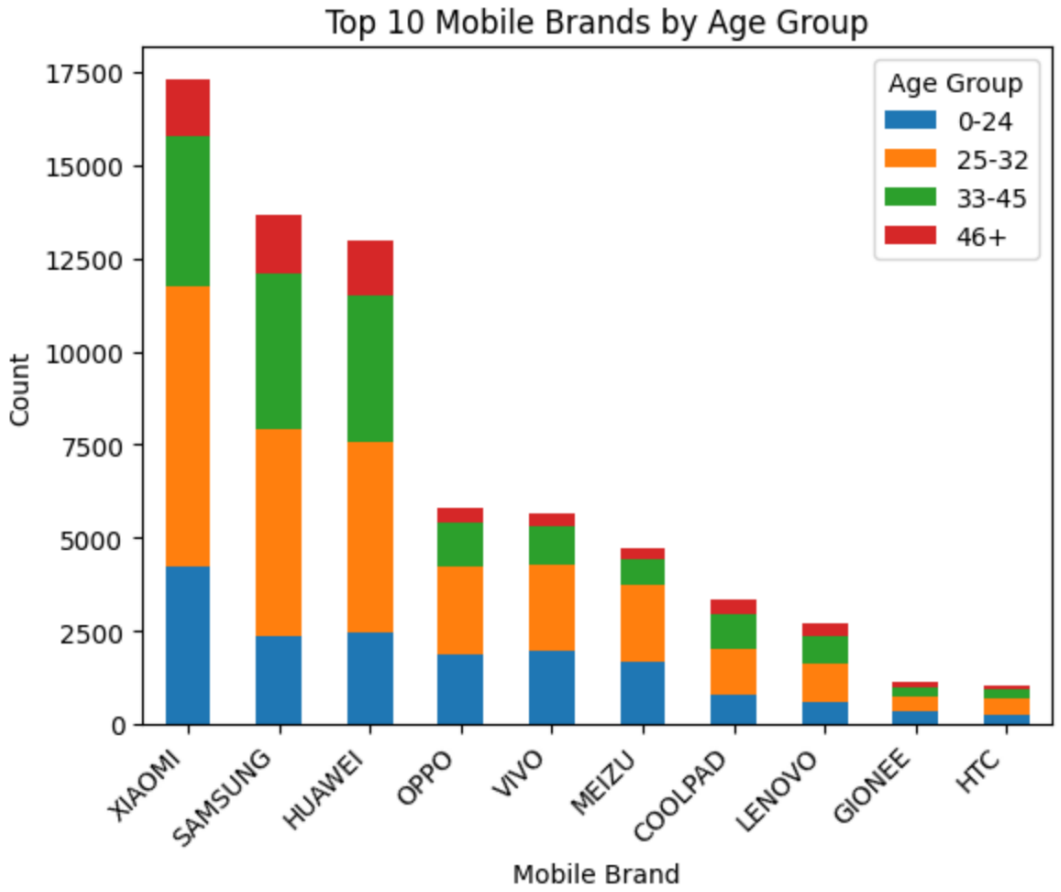
EDA (Apps): Top 10 frequent application for males & females



Most frequent app is with ID *8693964245073640147*.

After the first 4-5 Apps the frequency of next set of apps decreases exponentially in this case as well.

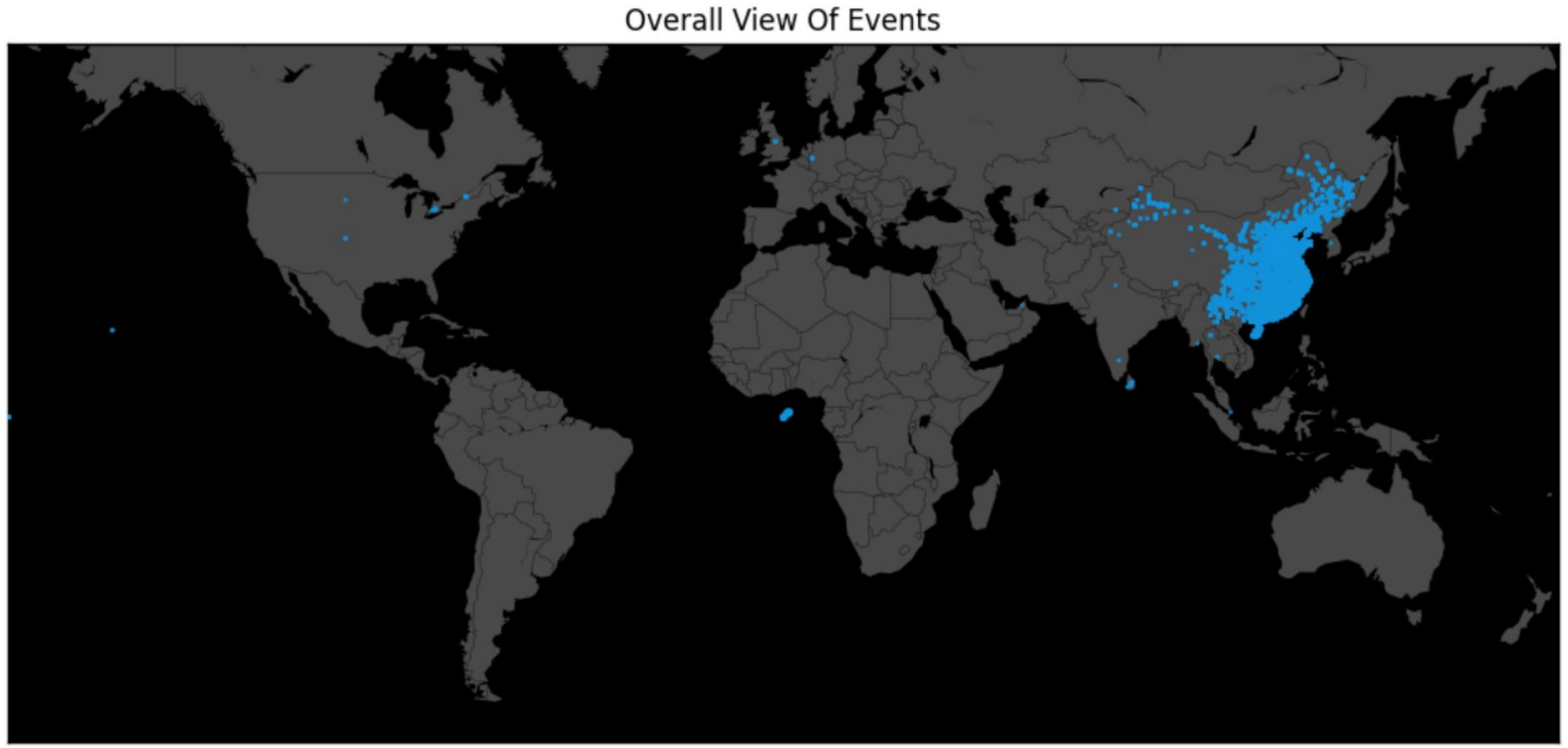
EDA (Mobile brands): Top 10 mobile brand by age group



The mobile brand distribution remains almost the same for age groups 24+ where XIAOMI is the highest and after top 3, uses decrease exponentially.

However for age group 0-24, The usage of first 6 brands is almost the same and doesn't follows the same trend as other age groups.

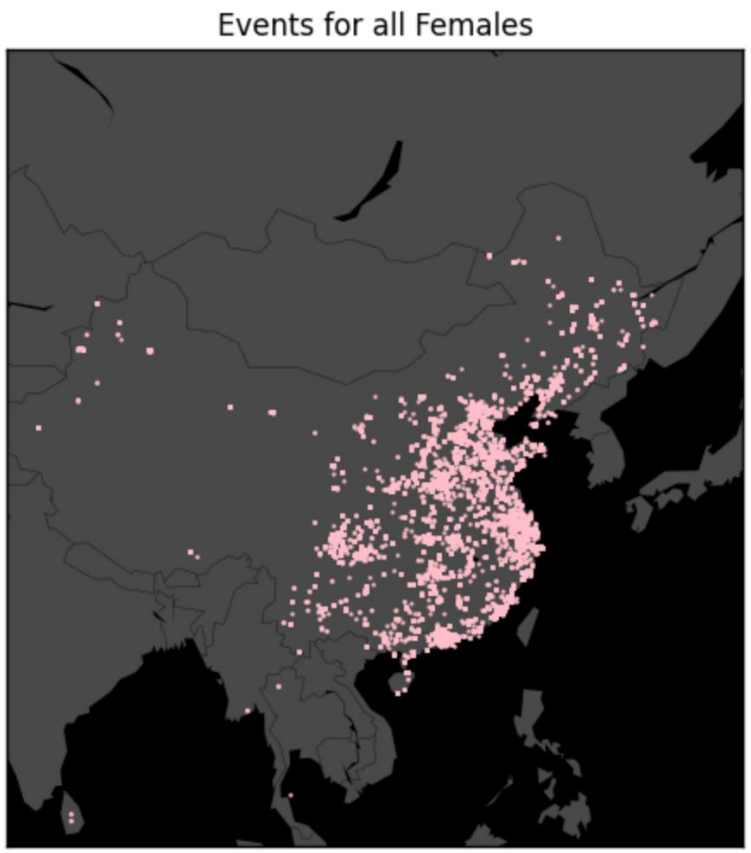
Geospatial Visualization – Overall View of Events

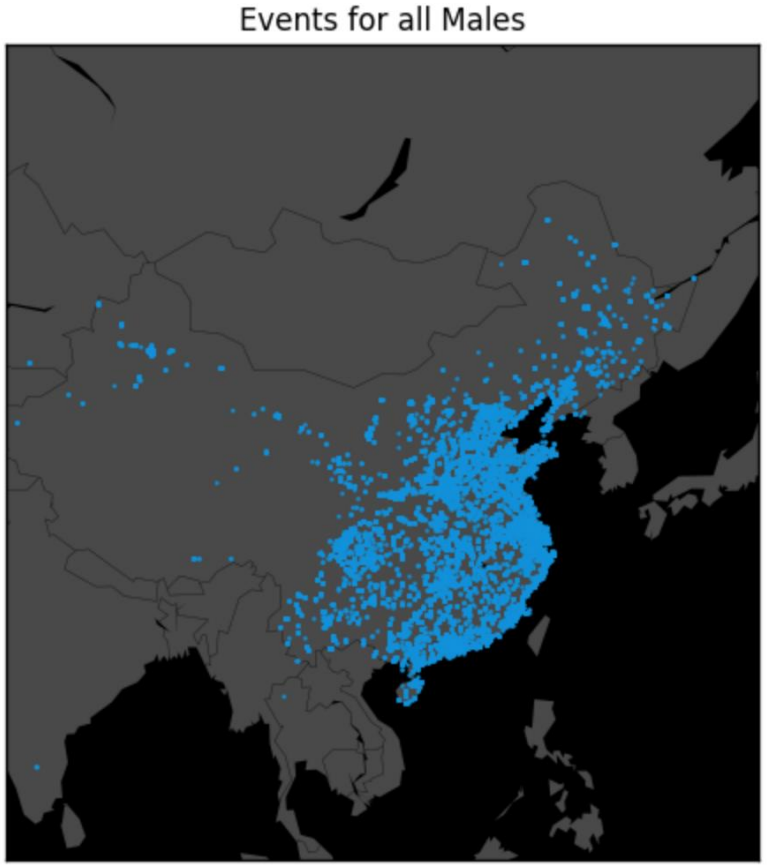


We see that most users are concentrated in the east Asia region between tropic of cancer & Capricorn.

We have a big no. of users at coordinate (0,0) which is in middle of sea, which shows us the need of cleaning this data.

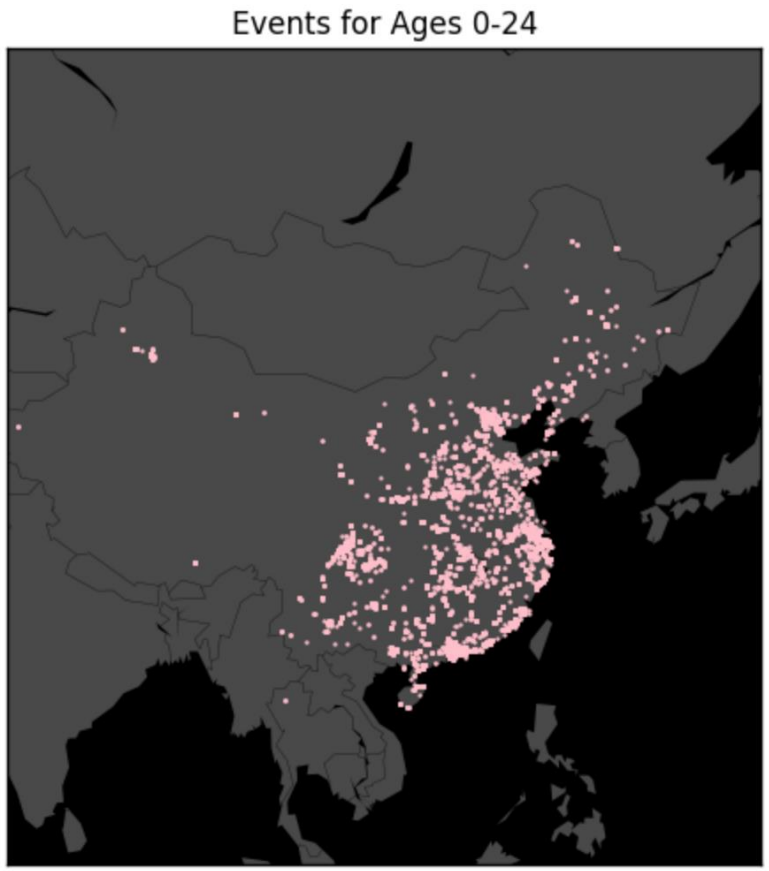
Geospatial Visualization – Events for different Genders

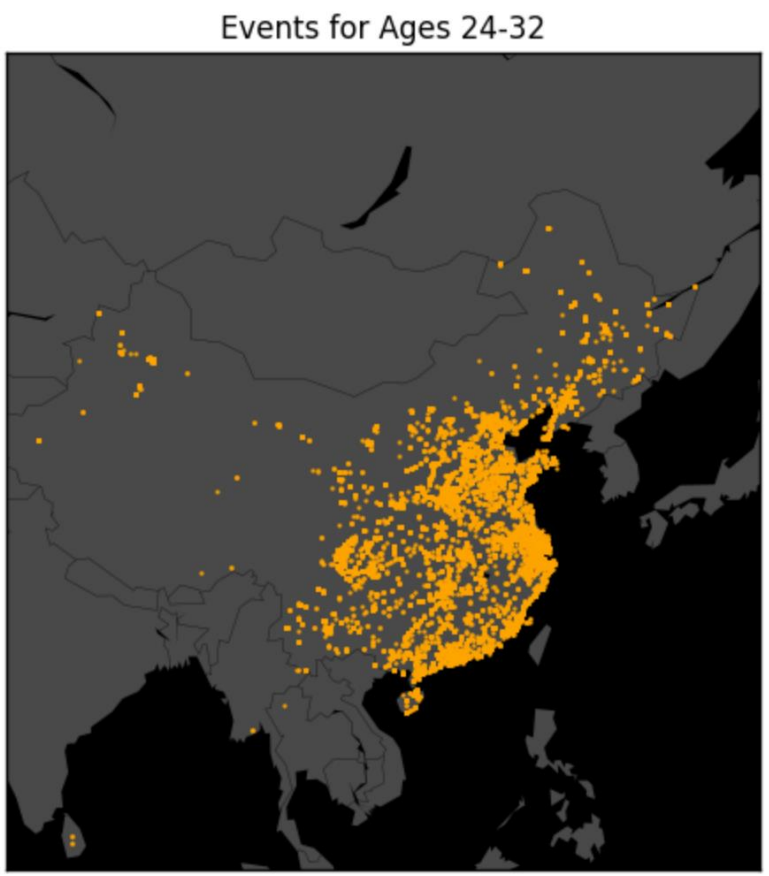


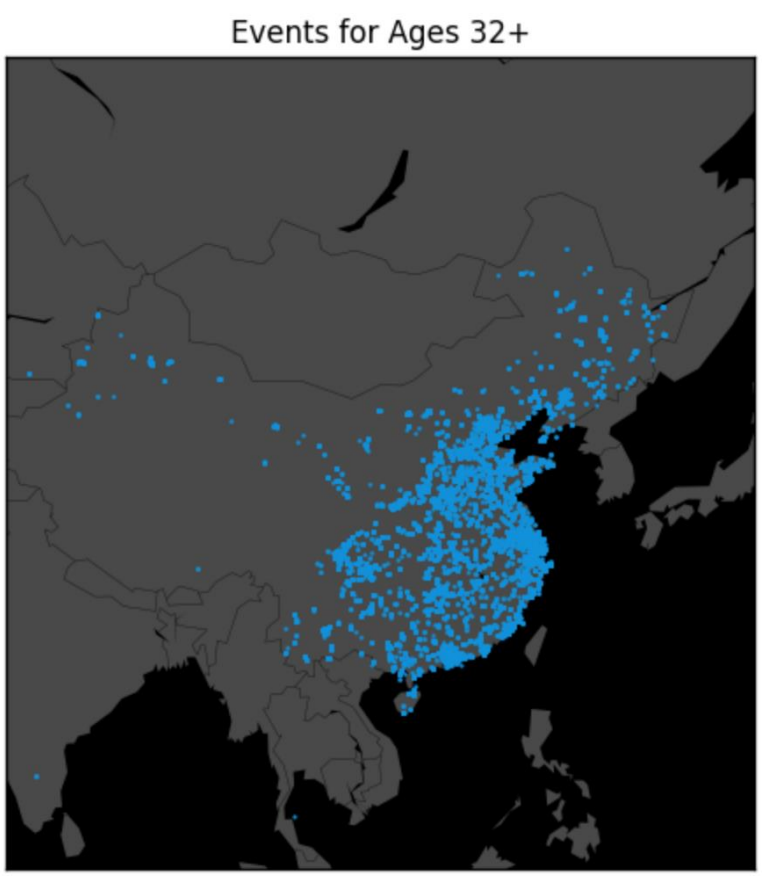


we see that
the **density** of events
from male population is
higher than females,
but geospatial
distribution looks similar.

Geospatial Visualization – Events for different Age groups







Feature Engineering – DBSCAN based clustering

[50]:
```
KMsPerRadian = 6371.0088

# epsilon = np.ones(coords.shape[0]) * 0.1 / kms_per_radian
eps_value = 100/ KMsPerRadian
coords = train_events_unique_location.loc[:, ["latitude", "longitude"]].values
dbscan = DBSCAN(eps= eps_value, algorithm = 'ball_tree', metric = 'haversine').fit(np.radians(coords))

cluster_labels = dbscan.labels_
num_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
print(num_clusters)
```

```
24
```

[51]:
```
train_events_unique_location["cluster"] = cluster_labels

train_events_unique_location["coordinates"]=train_events_unique_location[["latitude", "longitude"]].apply(lambda x:(x["latitude"],x["longitude"]), axis = 1)

train_events_with_location = pd.merge(train_events_with_location, train_events_unique_location,how="inner",on="coordinates")
train_events_with_location.head()
```

[51]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude_x	longitude_x	day_of_week	hour	age_group	coordinates	latitude_y	longitude_y	cluster
0	-7548291590301750000	M	33	M32+	2369465.0	2016-05-03 15:55:35	33.98	116.79	Tuesday	15.0	33-45	(33.98, 116.79)	33.98	116.79	0
1	-7548291590301750000	M	33	M32+	1080869.0	2016-05-03 06:07:16	33.98	116.79	Tuesday	6.0	33-45	(33.98, 116.79)	33.98	116.79	0
2	-7548291590301750000	M	33	M32+	1079338.0	2016-05-04 03:28:02	33.98	116.79	Wednesday	3.0	33-45	(33.98, 116.79)	33.98	116.79	0
3	-7548291590301750000	M	33	M32+	1078881.0	2016-05-04 02:53:08	33.98	116.79	Wednesday	2.0	33-45	(33.98, 116.79)	33.98	116.79	0
4	-7548291590301750000	M	33	M32+	1068711.0	2016-05-03 15:59:35	33.98	116.79	Tuesday	15.0	33-45	(33.98, 116.79)	33.98	116.79	0

We have done DBSCAN based clustering creating **24** clusters. Each location is now assigned a cluster ID from 0-23 using will be used in our model training as part of Data set preparation.

Feature Engineering – Events are different time of day

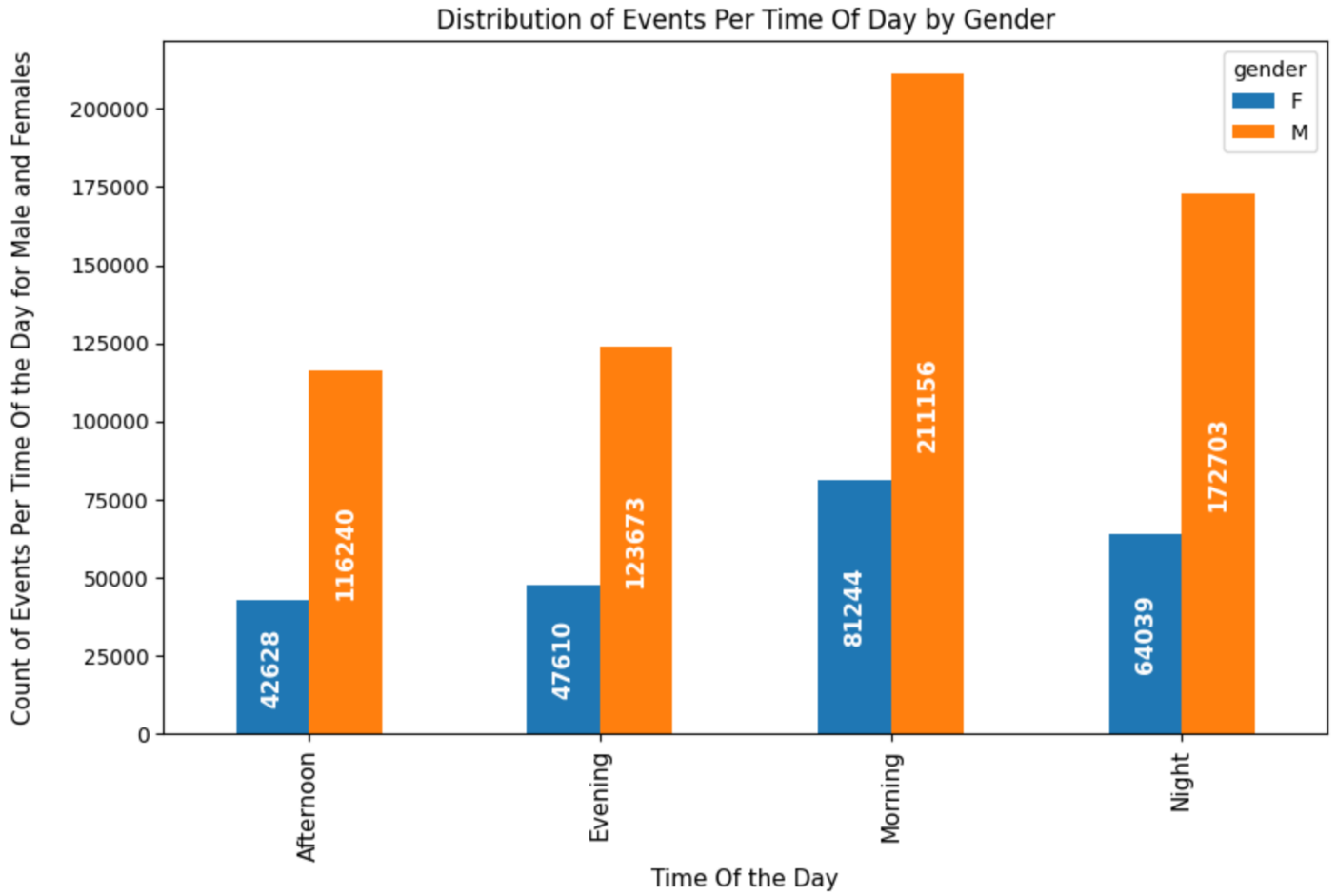
[53]:
```
# Define the bins for timeframe
bins = [0, 5, 12, 16, 20, 24]

# Define the labels for timeframe
labels = ['Night', 'Morning', 'Afternoon', 'Evening', "Night"]

train_events_with_location["time_of_day"]=pd.cut(train_events_with_location.hour,bins=bins,labels=labels,ordered=False,include_lowest=True)
train_events_with_location.head()
```

[53]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude_x	longitude_x	day_of_week	hour	age_group	coordinates	latitude_y	longitude_y	cluster	time_of_day
)	-7548291590301750000	M	33	M32+	2369465.0	2016-05-03 15:55:35	33.98	116.79	Tuesday	15.0	33-45	(33.98, 116.79)	33.98	116.79	0	Afternoon
I	-7548291590301750000	M	33	M32+	1080869.0	2016-05-03 06:07:16	33.98	116.79	Tuesday	6.0	33-45	(33.98, 116.79)	33.98	116.79	0	Morning
:	-7548291590301750000	M	33	M32+	1079338.0	2016-05-04 03:28:02	33.98	116.79	Wednesday	3.0	33-45	(33.98, 116.79)	33.98	116.79	0	Night
ł	-7548291590301750000	M	33	M32+	1078881.0	2016-05-04 02:53:08	33.98	116.79	Wednesday	2.0	33-45	(33.98, 116.79)	33.98	116.79	0	Night
ı	-7548291590301750000	M	33	M32+	1068711.0	2016-05-03 15:59:35	33.98	116.79	Tuesday	15.0	33-45	(33.98, 116.79)	33.98	116.79	0	Afternoon

We have created 4 time buckets and have assigned each event to one of those. These buckets are: **20-05 : Night**, **5-12: Morning**, **12-16: Afternoon**, **16-20: Evening**

EDA – Events at different time of day



Most events are generated during the morning hours, followed by the night hours.

Feature Engineering – Determine if the user is frequent of infrequent traveler

[55]:
```
# Getting records with location
train_events_with_location=train_events_with_location[train_events_with_location["coordinates"]!=(0,0)]

# Merging Morning and Afternoon together, while Evening and Night Together for simplistic calculation
train_events_with_location["time_of_day"]=train_events_with_location["time_of_day"].apply(lambda x: "Evening" if x == "Night" else "Morning" if x == "Afternoon" else x)

# Getting Mode of "Coordinates" to get the maximum time a device was in at Morning or Evening
device_location_at_time_of_day = train_events_with_location.groupby(['device_id', 'time_of_day'])["coordinates"].agg(lambda x:x.mode().iat[0]).reset_index().pivot(index="de

# Filling missing location as 0
device_location_at_time_of_day.fillna(0,inplace=True)

# Set if device user is frequent traveller throught a day
device_location_at_time_of_day["traveller_type"]=device_location_at_time_of_day.apply(lambda x: "Infrequent" if x["Evening"]==x["Morning"] else "Infrequent" if x["Evening"]

device_location_at_time_of_day
```

[55]:

time_of_day	device_id	Evening	Morning	traveller_type
0	-9222956879900150000	(23.19, 113.24)	(23.19, 113.24)	Infrequent
1	-9221026417907250000	(30.87, 114.36)	(30.87, 114.36)	Infrequent
2	-9220061629197650000	(46.6, 124.91)	(46.6, 124.91)	Infrequent
3	-9218769147970100000	(28.6, 112.33)	0	Infrequent
4	-9215352913819630000	0	(22.66, 114.02)	Infrequent
...	...	...	...	...
11975	9215085115859650000	(41.81, 123.46)	0	Infrequent
11976	9216925254504440000	(31.32, 120.56)	(31.32, 120.56)	Infrequent
11977	9219164468944550000	(30.27, 120.17)	(30.24, 120.16)	Frequent
11978	9219842210460030000	(26.52, 101.72)	0	Infrequent
11979	9220914901466450000	0	(32.67, 118.99)	Infrequent

11980 rows × 4 columns

Depending on where the user is mostly during the mornings and evenings we will try to determine whether the user is a **frequent** or **infrequent** traveler. This is a new feature which we will use in our feature vector.

Feature Engineering – Assigning high level categories to all apps

We have created a mapping for categories -> high level category and we will use it to create a new feature. For some of the categories we will keep high-level category as UNKNOWN.

```
[62]: device_event_with_app_and_category_df["highlevelcategory"].fillna("UNKNOWN",inplace=True)
      device_event_with_app_and_category_df.head()
```

[62]:

device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude	day_of_week	hour	age_group	coordinates	app_id	is_installed	is_active	highlevelcategory
)301750000	M	33	M32+	1141870.0	2016-05-01 10:08:16	33.98	116.79	Sunday	10.0	33-45	(33.98, 116.79)	3433289601737013244	1	1	BUSINESS
)301750000	M	33	M32+	1141870.0	2016-05-01 10:08:16	33.98	116.79	Sunday	10.0	33-45	(33.98, 116.79)	-5472633337921616096	1	1	NATURE
)301750000	M	33	M32+	1141870.0	2016-05-01 10:08:16	33.98	116.79	Sunday	10.0	33-45	(33.98, 116.79)	9112463114311278255	1	0	NA
)301750000	M	33	M32+	1141870.0	2016-05-01 10:08:16	33.98	116.79	Sunday	10.0	33-45	(33.98, 116.79)	8693964245073640147	1	1	FINANCE
)301750000	M	33	M32+	1141870.0	2016-05-01 10:08:16	33.98	116.79	Sunday	10.0	33-45	(33.98, 116.79)	5099453940784075687	1	1	EDUCATION

```
[63]: device_event_with_app_and_category_df.highl
```

[63]:

```
highlevelcategory
FINANCE          3910238
SHOPPING         2219132
UNKNOWN          2173833
BUSINESS         1925730
NA                810573
EDUCATION         555972
SOCIAL            272143
GAME               87265
ENTERTAINMENT      85114
NATURE             69916
OTHERS             41489
RELIGION           31134
TRAVEL             16104
HEALTH              8347
GENERAL             7057
FASHION             6620
PHOTOGRAPHY         5859
SERVICES            4253
TECHNOLOGY          4003
HEALLTH             2052
ART                  225
ASTROLOGY            161
SPORTS                34
Name: count, dtype: int64
```

Feature Engineering – Event Count per Device

[67]:
```
device_event_count_df=train_events_with_location.groupby("device_id").count()["event_id"].reset_index()
device_event_count_df=device_event_count_df.rename(columns={"event_id":"event_count"})
device_event_count_df.head()
```

[67]:

	device_id	event_count
0	-9222956879900150000	52
1	-9221026417907250000	132
2	-9220061629197650000	39
3	-9218769147970100000	17
4	-9215352913819630000	18

Data Preparation – Merge all Data Frames

Merge all necessary Data Frames

[69]:
```
device_data_df = train_events_with_location[["device_id", "gender", "age_group"]].drop_duplicates()
device_events_merged_df = pd.merge(device_data_df, device_location_at_time_of_day, how="left", on="device_id")
device_events_merged_df = pd.merge(device_events_merged_df, top_category_for_device_df, how="left", on="device_id")
device_events_merged_df = pd.merge(device_events_merged_df, device_cluster_df, how="left", on="device_id")
device_events_merged_df = pd.merge(device_events_merged_df, device_event_count_df, how="left", on="device_id")
device_events_merged_df = pd.merge(device_events_merged_df, mobile_brand_df, how="left", on="device_id")
print("Shape of merged df = ", device_events_merged_df.shape)
device_events_merged_df.head()
```

```
Shape of merged df =  (11980, 11)
```

[69]:

	device_id	gender	age_group	Evening	Morning	traveller_type	highlevelcategory	cluster	event_count	phone_brand	device_model
0	-7548291590301750000	M	33-45	(33.98, 116.79)	(33.98, 116.79)	Infrequent	BUSINESS	0	292	HUAWEI	3C
1	4543988487649880000	M	46+	(30.88, 104.26)	(30.92, 104.25)	Frequent	FINANCE	0	39	SAMSUNG	GALAXY S4
2	-1819925713085810000	F	0-24	(36.32, 114.59)	(36.33, 114.59)	Frequent	BUSINESS	0	38	OPPO	N1 MINI
3	3585775875204580000	F	46+	0	(35.3, 111.66)	Infrequent	FINANCE	0	14	XIAOMI	MI 4
4	7442042493953950000	M	25-32	(39.43, 116.98)	0	Infrequent	FINANCE	0	2	HUAWEI	HU1 PLUS

Data Preparation – Label Encoding

LabelEncoder

[73]:
```
#Defining Target Columns
target_columns = ['gender', 'age_group']

# Create an instance of the LabelEncoder
encoder = LabelEncoder()

# Apply label encoding on the selected columns
device_events_merged_encoded_df = device_events_merged_df.copy()
label_mappings = {}

for column in target_columns:
    encoded_labels = encoder.fit_transform(device_events_merged_df[column])
    device_events_merged_encoded_df[column] = encoded_labels
    label_mappings[column] = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))

# Print the mappings for each column
for column, mappings in label_mappings.items():
    print(f"Label Mappings for column '{column}':")
    for label, encoded_label in mappings.items():
        print(f"{label} : {encoded_label}")
    print()
```

```
Label Mappings for column 'gender':
F : 0
M : 1

Label Mappings for column 'age_group':
0-24 : 0
25-32 : 1
33-45 : 2
46+ : 3
```

We have done Label Encoding for rows gender & age group. The mapping are shown in the screenshot on the left,

Data Preparation – Standard Scaler

[75]:
```
num_columns=["event_count"]

# Apply StandardScaler
scaler = StandardScaler()
device_events_merged_encoded_df[num_columns] = scaler.fit_transform(device_events_merged_encoded_df[num_columns].values)
```

[76]:
```
device_events_merged_encoded_df.head()
```

[76]:

	device_id	gender	age_group	traveller_type	highlevelcategory	cluster	event_count	phone_brand	device_model
0	-7548291590301750000	1	2	Infrequent	BUSINESS	0	1.398279	HUAWEI	3C
1	4543988487649880000	1	3	Frequent	FINANCE	0	-0.207751	SAMSUNG	GALAXY S4
2	-1819925713085810000	0	0	Frequent	BUSINESS	0	-0.214099	OPPO	N1 MINI
3	3585775875204580000	0	3	Infrequent	FINANCE	0	-0.366450	XIAOMI	MI 4
4	7442042493953950000	1	1	Infrequent	FINANCE	0	-0.442625	HUAWEI	HU1 PLUS

Data Preparation – Target Encoding

For converting other values such as traveller_type, highlevelcateogry, cluster etc, we will use TargetEncoder

[77]:
```
target_columns = ['gender']

cat_columns = ["traveller_type", "highlevelcategory", "cluster", "phone_brand", "device_model"]
encoder = category_encoders.TargetEncoder(cols=cat_columns)

encoder.fit(device_events_merged_encoded_df, device_events_merged_encoded_df["gender"])

device_events_merged_encoded_df = encoder.transform(device_events_merged_encoded_df)
```

[78]:
```
device_events_merged_encoded_df.head()
```

[78]:

	device_id	gender	age_group	traveller_type	highlevelcategory	cluster	event_count	phone_brand	device_model
0	-7548291590301750000	1	2	0.696701	0.589212	0.707334	1.398279	0.734277	0.713415
1	4543988487649880000	1	3	0.743972	0.719878	0.707334	-0.207751	0.693680	0.663796
2	-1819925713085810000	0	0	0.743972	0.589212	0.707334	-0.214099	0.597403	0.479366
3	3585775875204580000	0	3	0.696701	0.719878	0.707334	-0.366450	0.712083	0.714556
4	7442042493953950000	1	1	0.696701	0.719878	0.707334	-0.442625	0.734277	0.745318

We will used target encoding to encode other columns. The target will be gender for gender prediction and age_group for age group prediction.

We will do these encodings for both Scenarios 1 & 2.

Data preparation – Train test info merge and data export for model training

[80]:
```
device_events_train_test_merged_df=pd.merge(device_events_merged_encoded_df,train_test_df[["device_id","train_test_flag"]],how="inner",on="device_id")
device_events_train_test_merged_df.head()
```

[80]:

	device_id	gender	age_group	traveller_type	highlevelcategory	cluster	event_count	phone_brand	device_model	train_test_flag
0	-7548291590301750000	1	2	0.696701	0.589212	0.707334	1.398279	0.734277	0.713415	train
1	4543988487649880000	1	3	0.743972	0.719878	0.707334	-0.207751	0.693680	0.663796	train
2	-1819925713085810000	0	0	0.743972	0.589212	0.707334	-0.214099	0.597403	0.479366	train
3	3585775875204580000	0	3	0.696701	0.719878	0.707334	-0.366450	0.712083	0.714556	train
4	7442042493953950000	1	1	0.696701	0.719878	0.707334	-0.442625	0.734277	0.745318	train

[81]:
```
# Define the directory path
directory = 'model_input'

# Create the directory if it doesn't exist
if not os.path.exists(directory):
    os.makedirs(directory)

# Define the file path
file_path = os.path.join(directory, 'device_events_train_test_merged.csv')

# Write the prepared database to a csv file which will be used in next notebook
device_events_train_test_merged_df.to_csv(file_path, index=False)
```

Would you like to re

Data Preparation – Train test split

[235]:
```
train_df=device_events_train_test_merged_df[device_events_train_test_merged_df["train_test_flag"]=="train"]
test_df=device_events_train_test_merged_df[device_events_train_test_merged_df["train_test_flag"]=="test"]
```

[236]:
```
train_df
```

[236]:

	device_id	gender	age_group	traveller_type	highlevelcategory	cluster	event_count	phone_brand	device_model	train_test_flag
0	-7548291590301750000	1	2	0.696701	0.589212	0.707334	1.398279	0.734277	0.713415	train
1	4543988487649880000	1	3	0.743972	0.719878	0.707334	-0.207751	0.693680	0.663796	train
2	-1819925713085810000	0	0	0.743972	0.589212	0.707334	-0.214099	0.597403	0.479366	train
3	3585775875204580000	0	3	0.696701	0.719878	0.707334	-0.366450	0.712083	0.714556	train
4	7442042493953950000	1	1	0.696701	0.719878	0.707334	-0.442625	0.734277	0.745318	train
...	...	...	...	...	...	...	...	...	...	...
8926	-7218221365876010000	1	1	0.696701	0.702925	0.707334	-0.442625	0.734277	0.736434	train
8927	-3559980869201690000	0	1	0.696701	0.719878	0.707334	-0.448973	0.734277	0.761814	train
8928	2168571045625280000	1	1	0.696701	0.719878	0.707334	-0.322014	0.734277	0.668979	train
8929	2481005796267730000	0	3	0.696701	0.719878	0.707334	0.795225	0.693680	0.726679	train
8930	6759696271364060000	1	1	0.696701	0.719878	0.707334	-0.366450	0.734277	0.736434	train

8931 rows × 10 columns

We will split data intro train and test.

Train has 8931 rows

Test has 3049 rows

Model Building Methodologies

We have used 4 different models with multiple variants for both Scenario 1 & Scenario 2. The models are:

- Logistic Regression – with different penalties – L2 (Ridge Regression) & L1(Lasso Regression)
- Random Forest with Hyperparameter tuning
- XGBoost with Hyperparameter tuning & Cross Validation
- Model Stacking with 3 modes:
  1. Classifier – Logistic Regression & Random Forest, Meta Classifier – XGBoost
  2. Classifier – XGBoost & Random Forest, Meta Classifier – Logistic Regression
  3. Classifier – Logistic Regression & XGBoost, Meta Classifier – Random Forest

Model Evaluation Metrics Used

We have used the following model evaluation metrices to evaluate our model:

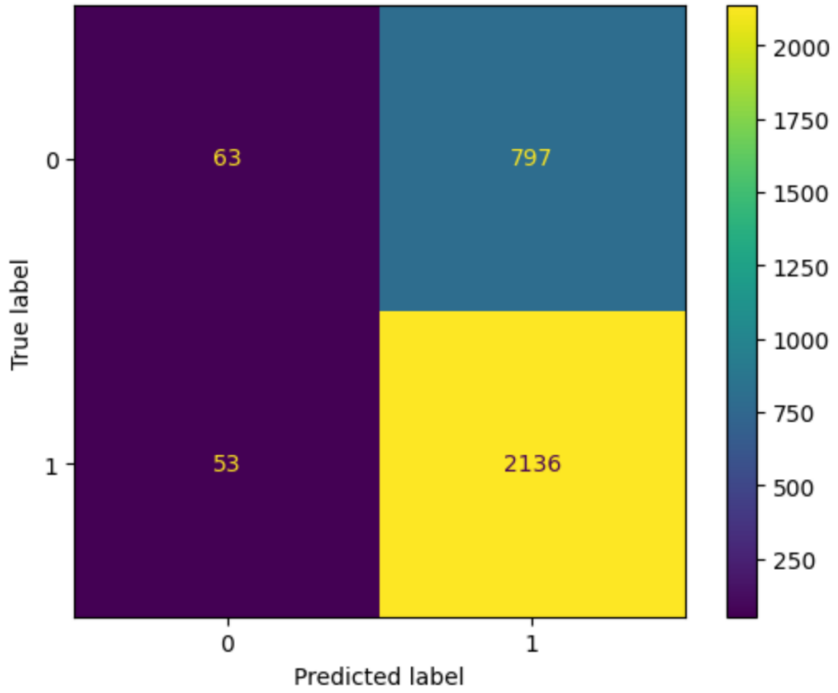
- **Model Statistics-** This Consists of following
  1. Model Accuracy (Training & test accuracy)
  2. Precision & Recall
  3. F1 Score
- **Confusion Matrix -** Confusion matrix is a table (2x2 for binary classification) which which displays the performance of a classification model on a set of test data for which the true values are known. The confusion matrix provides a comprehensive view of the performance of a classification model and allows the calculation of various performance metrics such as accuracy, precision, recall, F1 score, and others. These metrics can be calculated using the values from the confusion matrix.

Model Evaluation Metrics Used (contd..)

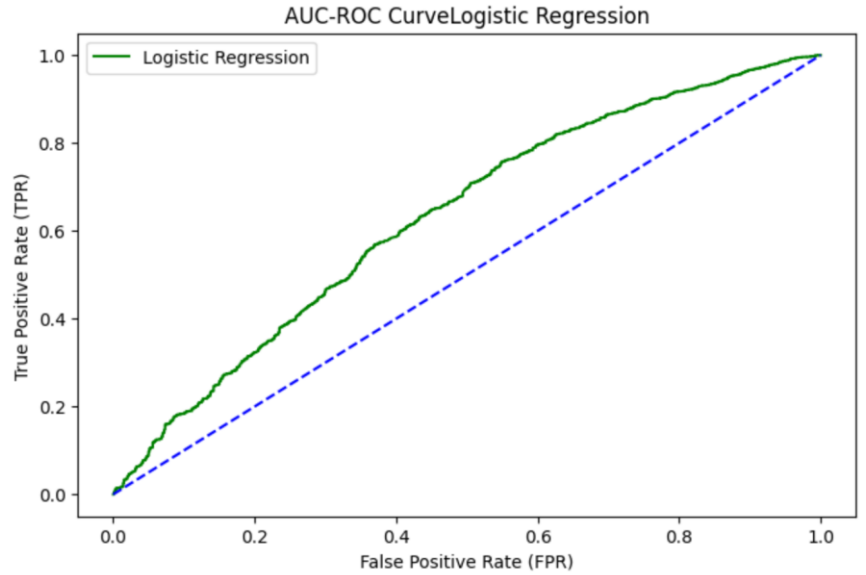
- **Lift Gain -** The lift_gain method calculates various performance metrics, including Lift, Gain, and the Kolmogorov-Smirnov (KS) statistic, based on a model's predicted probabilities and true labels. It divides the predictions into deciles, computes cumulative metrics and the KS statistic for each decile, and extracts probability bands for the top and bottom deciles. This provides insights into the model's performance across different probability ranges and facilitates evaluation and understanding of its behaviour.
- **ROC Curve -** The plot_roc_curve function computes and plots the Receiver Operating Characteristic (ROC) curve for a given model's predictions on test data. It displays the trade-off between the true positive rate (TPR) and false positive rate (FPR) at various probability thresholds, along with the Area Under the ROC Curve (ROC AUC) score. Additionally, it includes a dotted blue line representing the ROC curve for random predictions, aiding in the comparison of model performance.

Scenario 1 (Gender Prediction) – Logistic Regression with Ridge Regression

```
----------Model Statistics----------
Gender Model Accuracy: 0.7212200721548049
Gender Model training Accuracy: 0.7096629716717053
Gender Model Precision: 0.7282645755199455
Gender Model Recall: 0.9757880310644129
Gender Model F1 Score: 0.8340491995314331
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```




```
----------Decile Analysis----------
Top Probability: 0.6316585304687039
Bottom Probability: 0.7608180731494567
KS Statistic
Decile 0: 0.0017816354499771703
Decile 1: 0.00264961169483785
Decile 2: 0.004431247144814965
Decile 3: 0.0007309273640931924
Decile 4: 0.00296939241662858
Decile 5: 0.0038373686614892044
Decile 6: 0.0019643672910003573
Decile 7: 0.0032891731384193656
Decile 8: 0.005070808588396591
Decile 9: 0.002740977615349527
----------ROC Curve----------
ROC Area Under Curve SCORE ->  0.6305903194620035
```



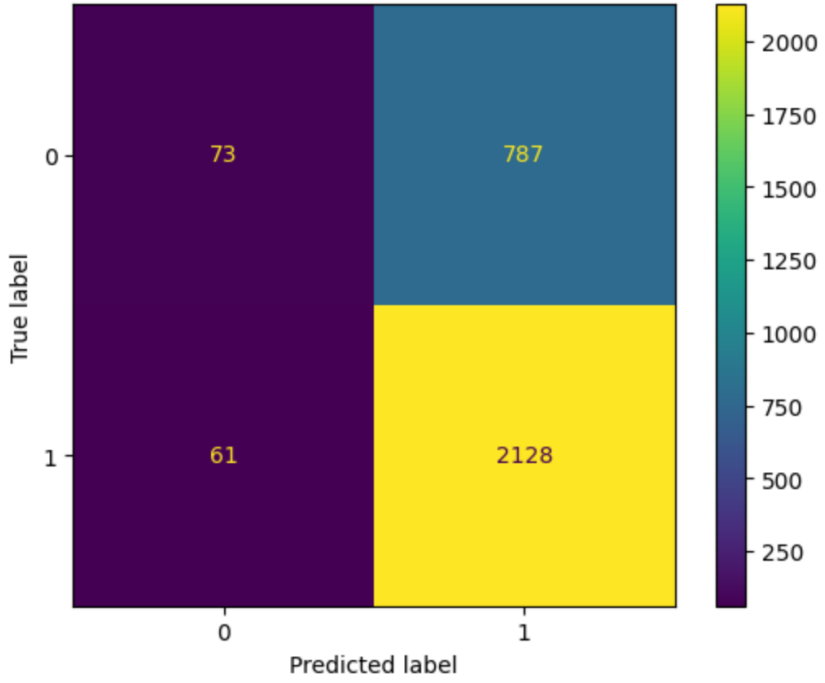
Final accuracy achieved is:

0.7212200721548049

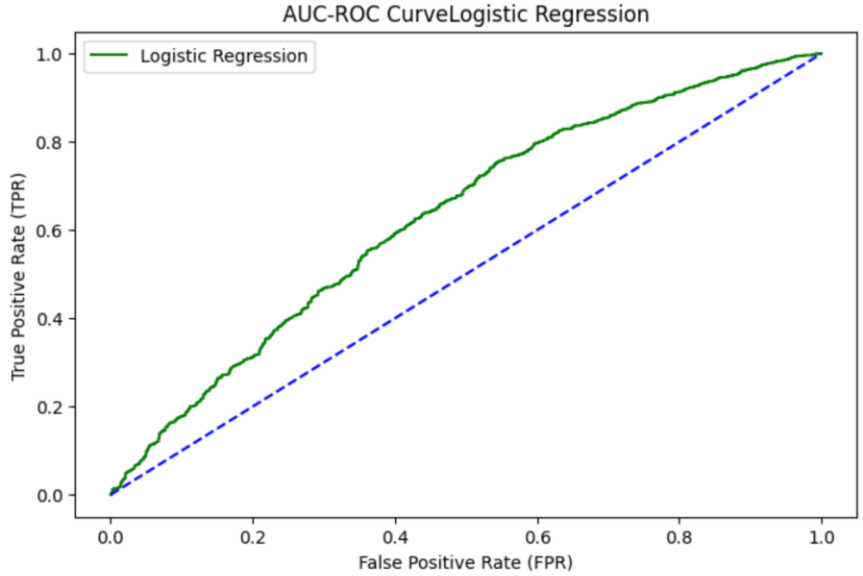
Also this is a high recall model

Scenario 1 (Gender Prediction) – Logistic Regression with Lasso Regularization

```
----------Model Statistics----------
Gender Model Accuracy: 0.7218760249262053
Gender Model training Accuracy: 0.7078714589631621
Gender Model Precision: 0.7300171526586621
Gender Model Recall: 0.972133394243947
Gender Model F1 Score: 0.8338557993730408
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```




```
----------Decile Analysis----------
Top Probability: 0.6268988718843206
Bottom Probability: 0.7653402833881494
KS Statistic
Decile 0: 0.0017816354499771703
Decile 1: 0.00264961169483785
Decile 2: 0.004431247144814965
Decile 3: 0.0007309273640931924
Decile 4: 0.00296939241662858
Decile 5: 0.0038373686614892044
Decile 6: 0.0019643672910003573
Decile 7: 0.0032891731384193656
Decile 8: 0.005070808588396591
Decile 9: 0.002740977615349527
----------ROC Curve----------
ROC Area Under Curve SCORE ->  0.6300415927417213
```



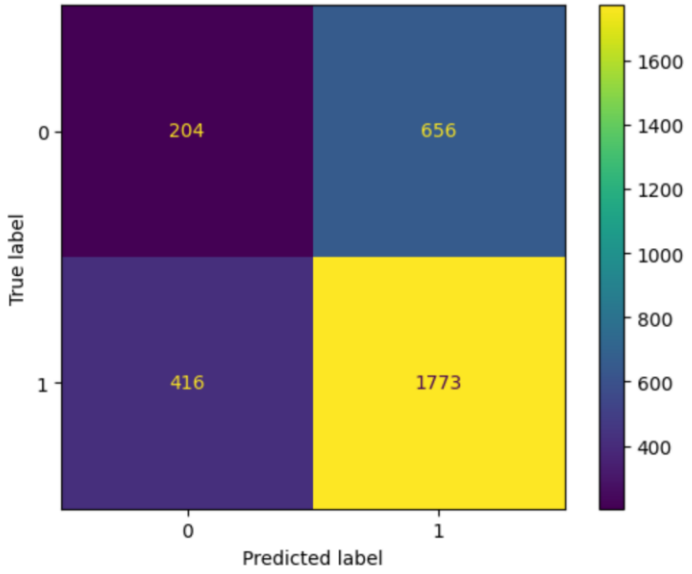
Final accuracy achieved is:

0.7218760249262053

This is better than ridge so we will chose this model for Logistic Regression

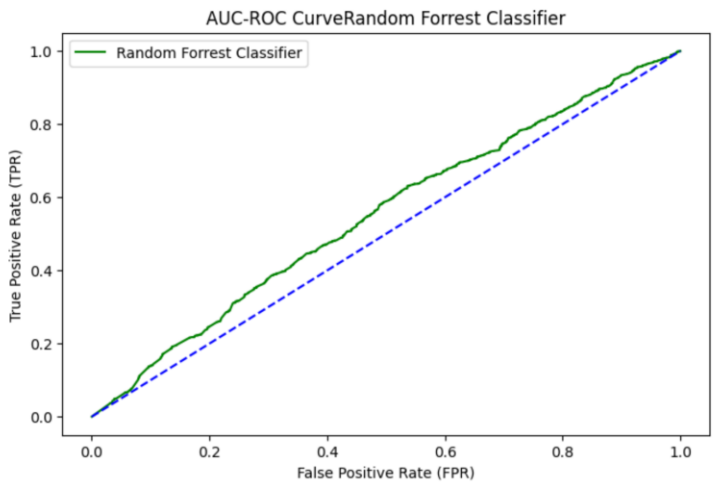
Scenario 1 (Gender Prediction) – Random Forest without any tuning

```
----------Model Statistics----------
Gender Model Accuracy: 0.6484093145293539
Gender Model training Accuracy: 0.9283394916582689
Gender Model Precision: 0.7299300123507616
Gender Model Recall: 0.8099588853357698
Gender Model F1 Score: 0.7678648765699437
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```



Predicted label

```
----------Decile Analysis----------
Top Probability: 0.49625
Bottom Probability: 0.89
KS Statistic
Decile 0: 0.0017816354499771703
Decile 1: 0.00264961169483785
Decile 2: 0.004431247144814965
Decile 3: 0.0007309273640931924
Decile 4: 0.00296939241662858
Decile 5: 0.0038373686614892044
Decile 6: 0.0019643672910003573
Decile 7: 0.0032891731384193656
Decile 8: 0.005070808588396591
Decile 9: 0.002740977615349527
----------ROC Curve----------
ROC Area Under Curve SCORE ->  0.5511667746767666
```



Final accuracy achieved is:

0.6484093145293539

From ROC curve we see that this is not a good performing model. So we will do hyper parameter tuning

Scenario 1 (Gender Prediction) – Hyper parameters used for tuning

```
param_grids = [
    {
        "n_estimators": [120, 130],
        "min_samples_split": [2, 3],
        "min_samples_leaf": [3, 5],
        "max_leaf_nodes": [100, 120],
        "max_depth": [50],
        "oob_score": [True],
        "bootstrap": [True],
    },
    {
        "n_estimators": [100, 120],
        "min_samples_split": [2, 3],
        "min_samples_leaf": [3, 5],
        "max_leaf_nodes": [100, 120],
        "max_depth": [50],
        "oob_score": [True],
        "bootstrap": [True],
    },
    {
        "n_estimators": [80, 100],
        "min_samples_split": [2, 3],
        "min_samples_leaf": [3, 5],
        "max_leaf_nodes": [100, 120],
        "max_depth": [50],
        "oob_score": [True],
        "bootstrap": [True],
    },
    {
        "n_estimators": [50, 80],
        "min_samples_split": [2, 5],
        "min_samples_leaf": [3, 5],
        "max_leaf_nodes": [100, 120],
        "max_depth": [50],
        "oob_score": [True],
        "bootstrap": [True],
    },
    {
        "n_estimators": [80],
        "min_samples_split": [2, 5],
        "min_samples_leaf": [5],
        "max_leaf_nodes": [80, 100],
        "max_depth": [50],
        "oob_score": [True],
        "bootstrap": [True],
    },
]
```

```
best_accuracy = 0
best_params_gender = {}
for i, param_grid in enumerate(param_grids):
    print("--------------------  Param Set ", i, "--------------------" )
    print(param_grid)
    y_prediction, accuracy, training_accuracy, cv, best_score, best_params_i = cross_validation(x_train, y_gender_train, x_test, y_gender_test, rf_gender_1, param_grid)
    print("Gender Model Accuracy:", accuracy)
    print("Gender Model training Accuracy:", training_accuracy)
    print("Best CV Score:", best_score)
    print("Best Paramater:", best_params_i)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_params_gender = best_params_i

print("\n ------------- \n")
print("After evaluating the 5 set of hyper parameters, the best accuracy is : ", best_accuracy)
print("set of params is : ", best_params_gender)
```

```
--------------------  Param Set  0 --------------------
{'n_estimators': [120, 130], 'min_samples_split': [2, 3], 'min_samples_leaf': [3, 5], 'max_leaf_nodes': [100, 120], 'max_depth': [50], 'oob_score': [True], 'bootstrap': [T
rue]}
Gender Model Accuracy: 0.725811741554608
Gender Model training Accuracy: 0.7332885455156197
Best CV Score: 0.7078714589631621
Best Paramater: {'bootstrap': True, 'max_depth': 50, 'max_leaf_nodes': 120, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 130, 'oob_score': True}
--------------------  Param Set  1 --------------------
{'n_estimators': [100, 120], 'min_samples_split': [2, 3], 'min_samples_leaf': [3, 5], 'max_leaf_nodes': [100, 120], 'max_depth': [50], 'oob_score': [True], 'bootstrap': [T
rue]}
Gender Model Accuracy: 0.7238438832404067
Gender Model training Accuracy: 0.7307132459970888
Best CV Score: 0.7074235807860262
Best Paramater: {'bootstrap': True, 'max_depth': 50, 'max_leaf_nodes': 100, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 100, 'oob_score': True}
--------------------  Param Set  2 --------------------
{'n_estimators': [80, 100], 'min_samples_split': [2, 3], 'min_samples_leaf': [3, 5], 'max_leaf_nodes': [100, 120], 'max_depth': [50], 'oob_score': [True], 'bootstrap': [Tr
ue]}
Gender Model Accuracy: 0.7235159068547065
Gender Model training Accuracy: 0.7308252155413727
Best CV Score: 0.7075355503303102
Best Paramater: {'bootstrap': True, 'max_depth': 50, 'max_leaf_nodes': 100, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 80, 'oob_score': True}
--------------------  Param Set  3 --------------------
```

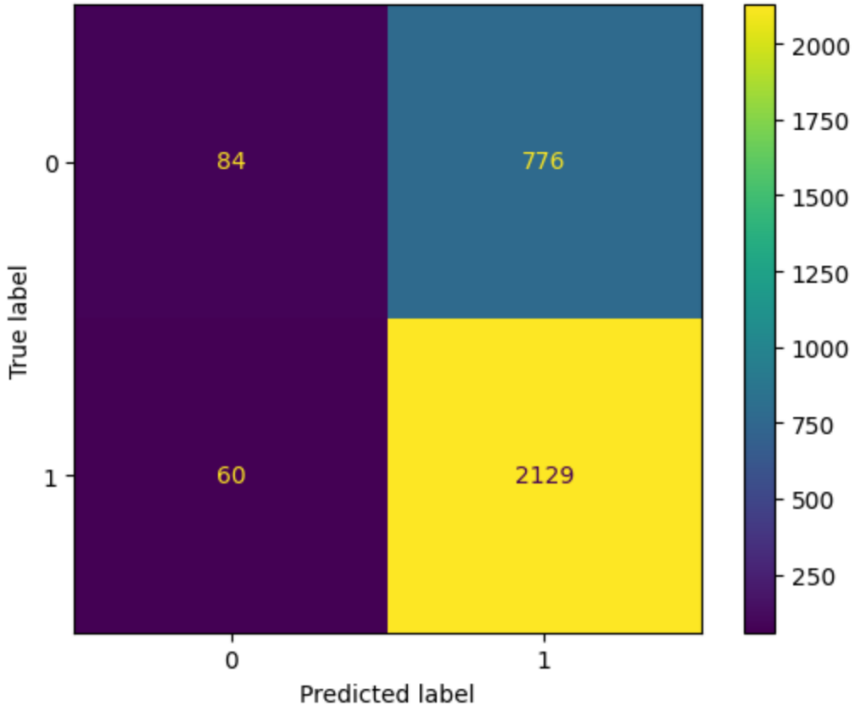
After evaluating the 5 set of hyper parameters, the best accuracy is :
0.715811741554608

set of params is :

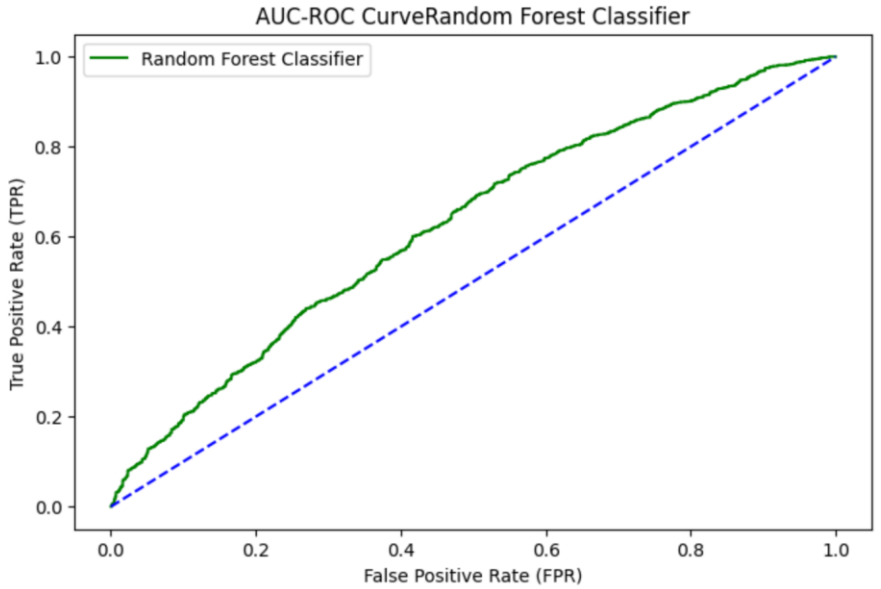
```
{'bootstrap': True,
'max_depth': 50,
'max_leaf_nodes': 120,
'min_samples_leaf': 5,
'min_samples_split': 2,
'n_estimators': 130,
'oob_score': True}
```

Scenario 1 (Gender Prediction) – Random Forest with Hyperparameter tuning

```
----------Model Statistics----------
Gender Model Accuracy: 0.725811741554608
Gender Model training Accuracy: 0.7332885455156197
Gender Model Precision: 0.7328743545611015
Gender Model Recall: 0.9725902238465053
Gender Model F1 Score: 0.8358853553199843
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```



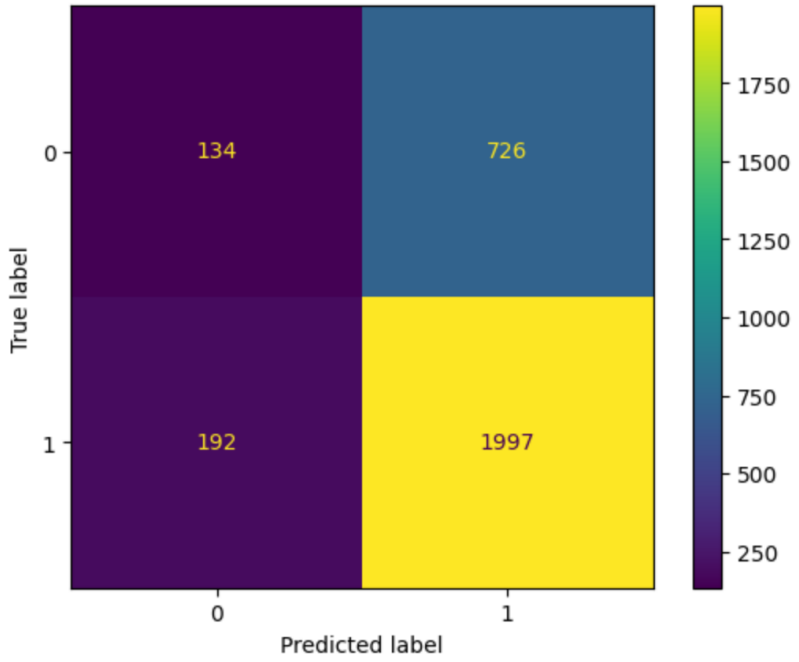

```
----------Decile Analysis----------
Top Probability: 0.6122615890874715
Bottom Probability: 0.7768039304671316
KS Statistic
Decile 0: 0.0017816354499771703
Decile 1: 0.00264961169483785
Decile 2: 0.004431247144814965
Decile 3: 0.0007309273640931924
Decile 4: 0.00296939241662858
Decile 5: 0.0038373686614892044
Decile 6: 0.0019643672910003573
Decile 7: 0.0032891731384193656
Decile 8: 0.005070808588396591
Decile 9: 0.002740977615349527
----------ROC Curve----------
ROC Area Under Curve SCORE ->  0.6253059695942715
```



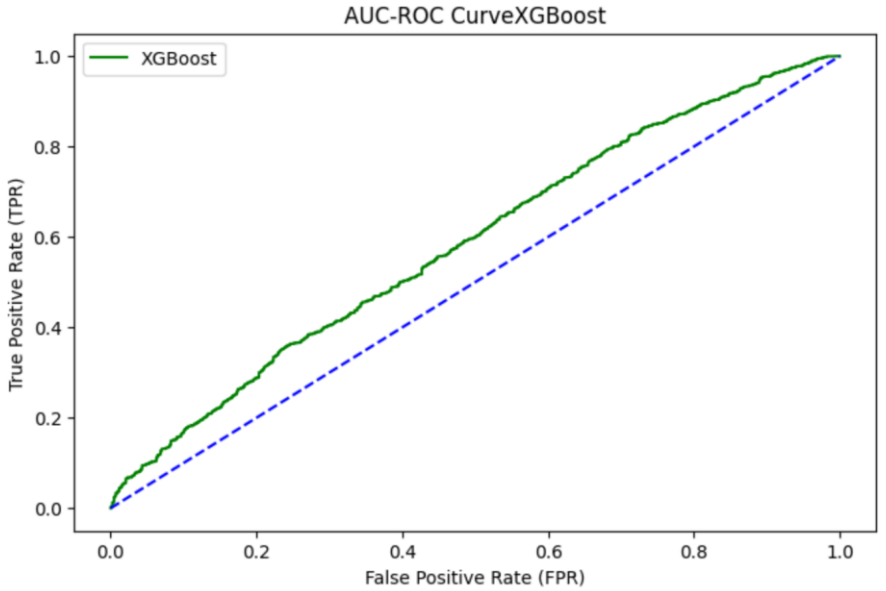
With the best parameters for this Random Forest model after hyper parameter tuning we have achieved a model with accuracy of **0.7158117415546 08**

Scenario 1 (Gender Prediction) – XGBoost Classifier Base Model

```
----------Model Statistics----------
Gender Model Accuracy: 0.6989176779271893
Gender Model training Accuracy: 0.7835628708991155
Gender Model Precision: 0.7333822989349982
Gender Model Recall: 0.9122887163088168
Gender Model F1 Score: 0.8131107491856677
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```




```
----------Decile Analysis----------
Top Probability: 0.592071
Bottom Probability: 0.80653703
KS Statistic
Decile 0: 0.0017816354499771703
Decile 1: 0.00264961169483785
Decile 2: 0.004431247144814965
Decile 3: 0.0007309273640931924
Decile 4: 0.00296939241662858
Decile 5: 0.0038373686614892044
Decile 6: 0.0019643672910003573
Decile 7: 0.0032891731384193656
Decile 8: 0.005070808588396591
Decile 9: 0.002740977615349527
----------ROC Curve----------
ROC Area Under Curve SCORE ->  0.5845493322851041
```



Model Accuracy of base model comes out to be

0.6989176779271893

Scenario 1 (Gender Prediction) – XGBoost Hyperparameter tuning

```
param_grids = [
    {
        "n_estimators": [10, 20],
        "max_depth": [3, 6],
        "learning_rate": [0.1, 0.3],
        "min_child_weight": [2, 3],
        "gamma": [0.1, 0.2],
    },
    {
        "n_estimators": [20, 50],
        "max_depth": [3, 6],
        "learning_rate": [0.1, 0.3],
        "min_child_weight": [2, 3],
        "gamma": [0.1, 0.2],
    },
    {
        "n_estimators": [20, 50],
        "max_depth": [3, 6],
        "learning_rate": [0.3, 0.4],
        "min_child_weight": [2, 3],
        "gamma": [0.2, 0.3],
    },
    {
        "n_estimators": [50, 80],
        "max_depth": [3, 6],
        "learning_rate": [0.3, 0.4],
        "min_child_weight": [2, 3],
        "gamma": [0.2, 0.3],
    },
    {
        "n_estimators": [50],
        "max_depth": [2, 3],
        "learning_rate": [0.3, 0.4],
        "min_child_weight": [3, 4],
        "gamma": [0.2, 0.3],
    },
]
```

```
best_accuracy = 0
best_params_gender = {}
for i, param_grid in enumerate(param_grids):
    print("--------------------  Param Set ", i, "--------------------" )
    print(param_grid)
    y_prediction, accuracy, training_accuracy, cv, best_score, best_params_i = cross_validation(x_train, y_gender_train, x_test,
    print("Gender Model Accuracy:", accuracy)
    print("Gender Model training Accuracy:", training_accuracy)
    print("Best CV Score:", best_score)
    print("Best Paramater:", best_params_i)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_params_gender = best_params_i

print("\n ------------ \n")
print("After evaluating the 5 set of hyper parameters, the best accuracy is : ", best_accuracy)
print("set of params is : ", best_params_gender)
```

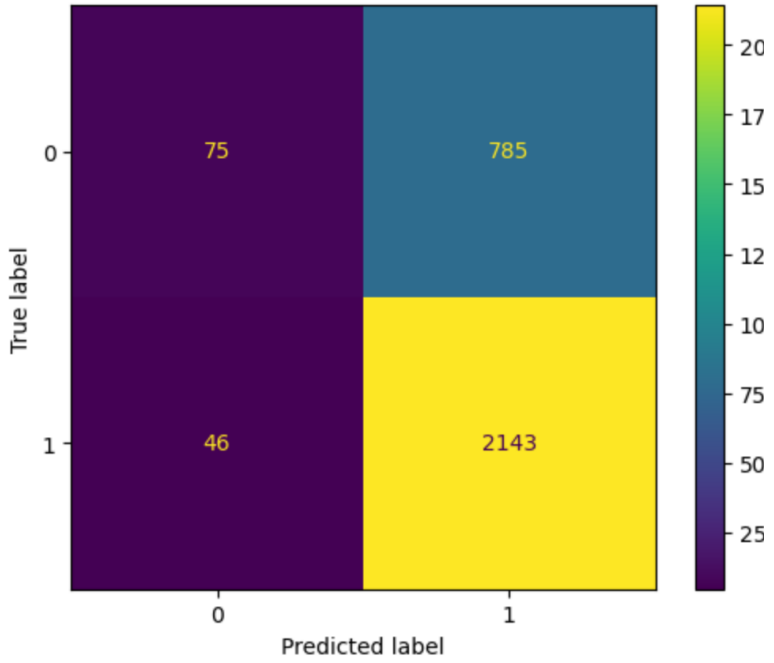
```
--------------------  Param Set  0 --------------------
{'n_estimators': [10, 20], 'max_depth': [3, 6], 'learning_rate': [0.1, 0.3], 'min_child_weight': [2, 3], 'gamma': [0.1, 0.2]}
Gender Model Accuracy: 0.7218760249262053
Gender Model training Accuracy: 0.7144776620759153
Best CV Score: 0.7095510021274215
Best Paramater: {'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 3, 'n_estimators': 10}
--------------------  Param Set  1 --------------------
{'n_estimators': [20, 50], 'max_depth': [3, 6], 'learning_rate': [0.1, 0.3], 'min_child_weight': [2, 3], 'gamma': [0.1, 0.2]}
Gender Model Accuracy: 0.7274516234831092
Gender Model training Accuracy: 0.7180606874930019
Best CV Score: 0.7084313066845818
Best Paramater: {'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 50}
--------------------  Param Set  2 --------------------
{'n_estimators': [20, 50], 'max_depth': [3, 6], 'learning_rate': [0.3, 0.4], 'min_child_weight': [2, 3], 'gamma': [0.2, 0.3]}
Gender Model Accuracy: 0.7261397179403083
Gender Model training Accuracy: 0.722987347441496
Best CV Score: 0.7076475198745942
Best Paramater: {'gamma': 0.3, 'learning_rate': 0.3, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 50}
--------------------  Param Set  3 --------------------
{'n_estimators': [50, 80], 'max_depth': [3, 6], 'learning_rate': [0.3, 0.4], 'min_child_weight': [2, 3], 'gamma': [0.2, 0.3]}
Gender Model Accuracy: 0.7261397179403083
Gender Model training Accuracy: 0.722987347441496
Best CV Score: 0.7076475198745942
Best Paramater: {'gamma': 0.3, 'learning_rate': 0.3, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 50}
--------------------  Param Set  4 --------------------
{'n_estimators': [50], 'max_depth': [2, 3], 'learning_rate': [0.3, 0.4], 'min_child_weight': [3, 4], 'gamma': [0.2, 0.3]}
Gender Model Accuracy: 0.7241718596261O7
```

After Hyper parameter tuning, the best accuracy achieved is **0.7274516234831092**

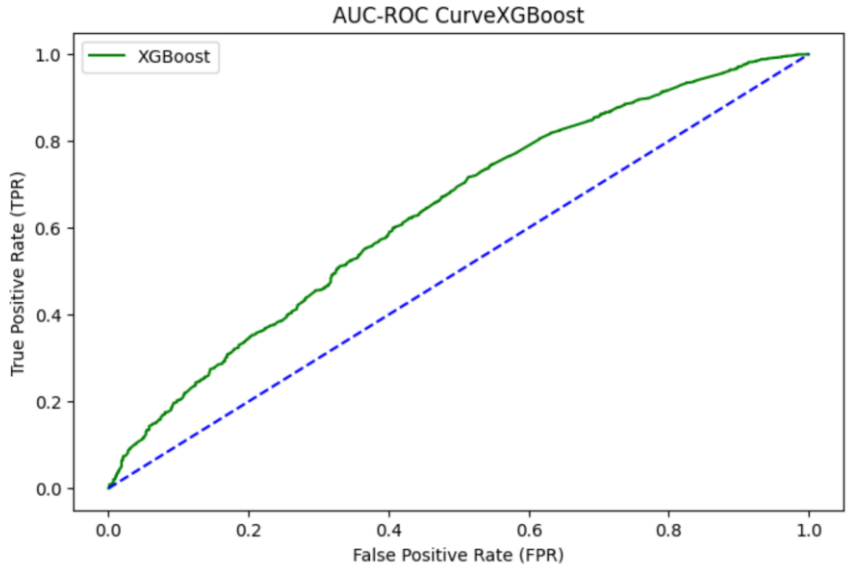
set of params is : `{'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 50}`

Scenario 1 (Gender Prediction) – XGBoost Final Model

```
----------Model Statistics----------
Gender Model Accuracy: 0.7274516234831092
Gender Model training Accuracy: 0.7180606874930019
Gender Model Precision: 0.7318989071038251
Gender Model Recall: 0.9789858382823207
Gender Model F1 Score: 0.8376001563416064
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```




```
----------Decile Analysis----------
Top Probability: 0.6333017
Bottom Probability: 0.776175
KS Statistic
Decile 0: 0.0017816354499771703
Decile 1: 0.00264961169483785
Decile 2: 0.004431247144814965
Decile 3: 0.0007309273640931924
Decile 4: 0.00296939241662858
Decile 5: 0.0038373686614892044
Decile 6: 0.0019643672910003573
Decile 7: 0.0032891731384193656
Decile 8: 0.005070808588396591
Decile 9: 0.002740977615349527
----------ROC Curve----------
ROC Area Under Curve SCORE ->  0.6350263473817289
```

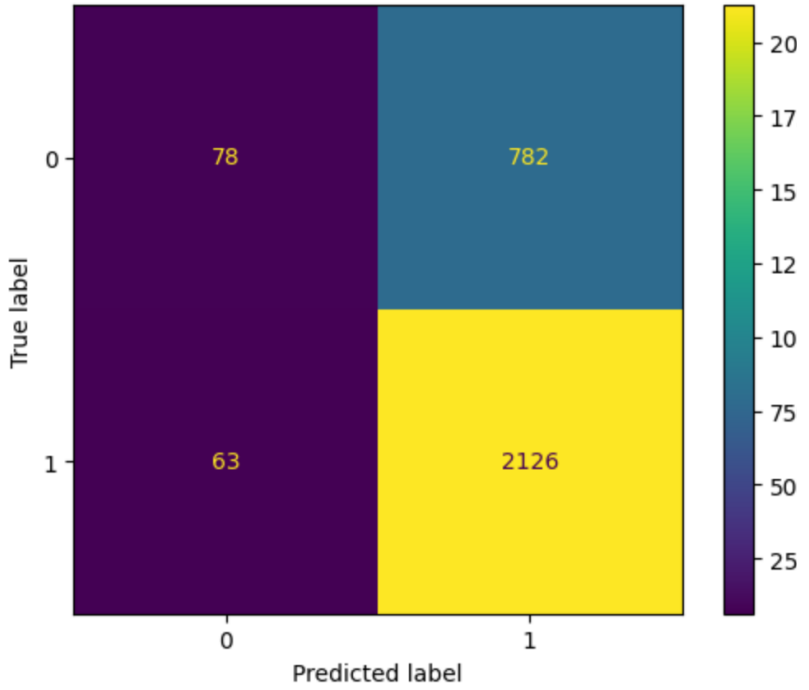


Accuracy achieved from the final XGBoost model is

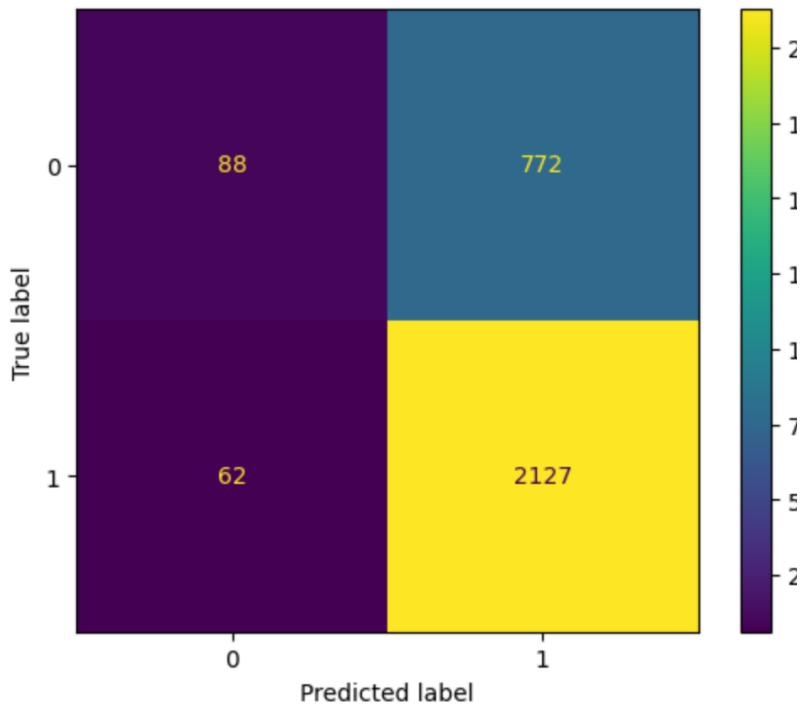
0.7180606874930019

Scenario 1 (Gender Prediction) – Model Stacking 3 scenarios

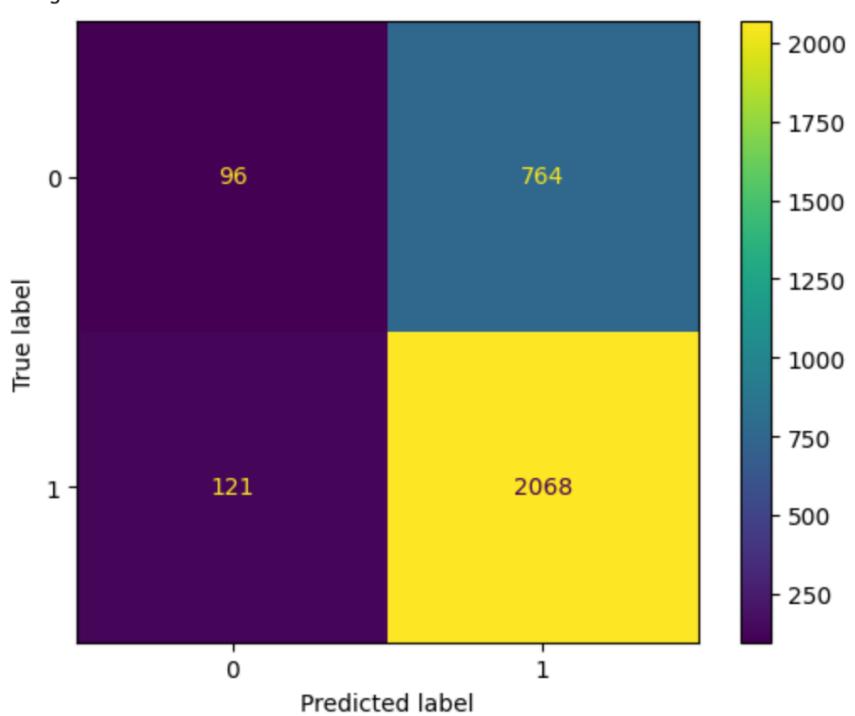
```
---------------Model Statistics---------------
Gender Model Accuracy: 0.722859954083306
Gender Model training Accuracy: 0.7263464337700145
Gender Model Precision: 0.7310866574965612
Gender Model Recall: 0.9712197350388305
Gender Model F1 Score: 0.8342162056111438
---------------Confusion Matrix---------------
<Figure size 800x500 with 0 Axes>
```




```
---------------Model Statistics---------------
Gender Model Accuracy: 0.7264676943260086
Gender Model training Accuracy: 0.7199641697458291
Gender Model Precision: 0.7337012763021732
Gender Model Recall: 0.9716765646413887
Gender Model F1 Score: 0.8360849056603774
---------------Confusion Matrix---------------
<Figure size 800x500 with 0 Axes>
```




```
---------------Model Statistics---------------
Gender Model Accuracy: 0.7097408986552968
Gender Model training Accuracy: 0.7085432762288657
Gender Model Precision: 0.730225988700565
Gender Model Recall: 0.9447236180904522
Gender Model F1 Score: 0.8237402907787293
---------------Confusion Matrix---------------
<Figure size 800x500 with 0 Axes>
```



Scenario 1 (Gender Prediction): Final Model for Stacking Classifier

From the three models we see that the first (Classifier – Logistic Regression & Random Forest, Meta Classifier – XGBoost) gives us the highest accuracy so we chose this as our final model for stacking classifier.

```
# Since Model 1 gives best accuracy, considering it is best Stacking Model
stacking_gender = stacking_gender_1
```

StackingCVClassifier

```
StackingCVClassifier(classifiers=[LogisticRegression(penalty='l1',
                                                     solver='liblinear'),
                                  RandomForestClassifier(max_depth=50,
                                                         max_leaf_nodes=120,
                                                         min_samples_leaf=5,
                                                         n_estimators=130,
                                                         oob_score=True,
                                                         random_state=100)],
                    cv=3,
                    meta_classifier=XGBClassifier(base_score=None,
                                                  booster=None, callbacks=None,
```

meta_classifier: XGBClassifier

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=2, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=50, n_jobs=None,
              num_parallel_tree=None, random_state=100, ...)
```

Scenario 1 (Gender Prediction): Final Model Selection

For the final model from the logistic regression, random forest, XG Boost & Stacking model the accuracy obtained are:

1. logistic regression - `0.7284355526402099`
2. random forest - `0.7277795998688095`
3. XG Boost - `0.727123647097409`
4. Stacking - `0.7267956707117088`

From the above 4, we see that logistic regression has the highest accuracy, so we will use logistic regression for the gender prediction

[282]:
```
final_model_gender = lr_gender
final_model_gender
```

[282]:
```
LogisticRegression(penalty='l1', solver='liblinear')
```

We will now export this model which we will use in our Flask application for Gender Prediction

[283]:
```
# Define the directory path
directory = 'models'

# Create the directory if it doesn't exist
if not os.path.exists(directory):
    os.makedirs(directory)

pickle.dump(final_model_gender,open("models/model_gender.pkl","wb"))
```

Scenario 1: Age Group Prediction

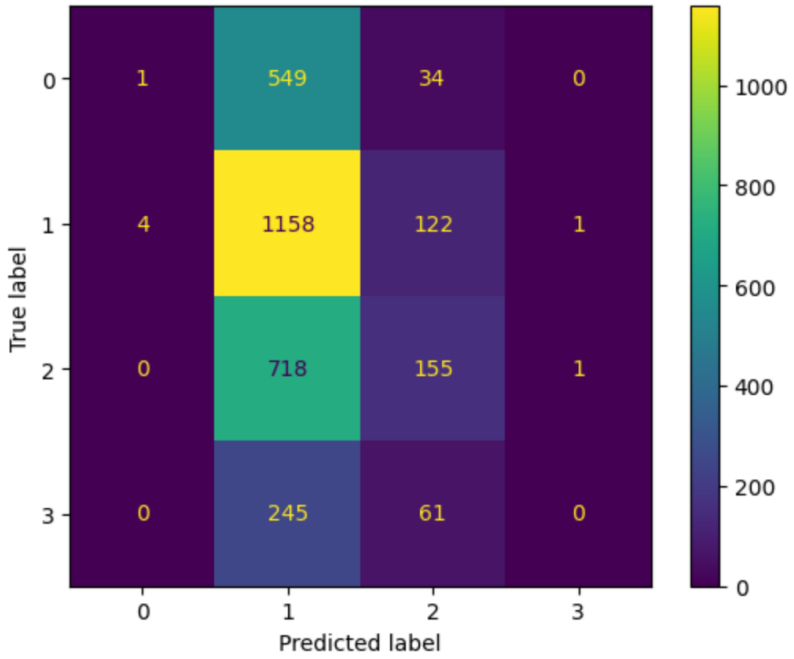
For age group prediction, we have already created 4 bins during our data preparation step. The bins are labelled **as 0-24 : 0, 25-32 : 1, 33-45 : 2, 46+ : 3**

Since post this bucketing, we only have 4 buckets to classify our data into. So we have modelled this problem as a **Classification problem** instead of a regression problem.

And we have used the same set of Models used in gender prediction for age group prediction as well

Scenario 1: Age Group Prediction – XGBoost with hyperparameter tuning

```
----------Model Statistics----------
Age Group Model Accuracy: 0.43096097081010165
Age Group Model training Accuracy: 0.4428395476430411
Age Group Model Precision: 0.34053173939266634
Age Group Model Recall: 0.43096097081010165
Age Group Model F1 Score: 0.3187639421739321
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```




```
----------Multiclass Log Loss----------
Multiclass Log Loss: 1.2401
```

Among all the 4 different type of models, XG Boost gives us the best accuracy of 0.4309609708101016 5, and we will use this as our final model

Scenario 1 (Age group prediction) – Final Model Selection

For the final model from the logistic regression, random forest, XG Boost & Stacking model the accuracy obtained are:

1. logistic regression - `0.43555264020990486`
2. random forest - `0.4312889471958019`
3. XG Boost - `0.44743029895868325`
4. Stacking - `0.432928829124303`

From the above 4, we see that XG Boost has the highest accuracy, so we will use XG Boost for the gender prediction

[320]:
```
final_model_age_group = xgb_age_group
final_model_age_group
```

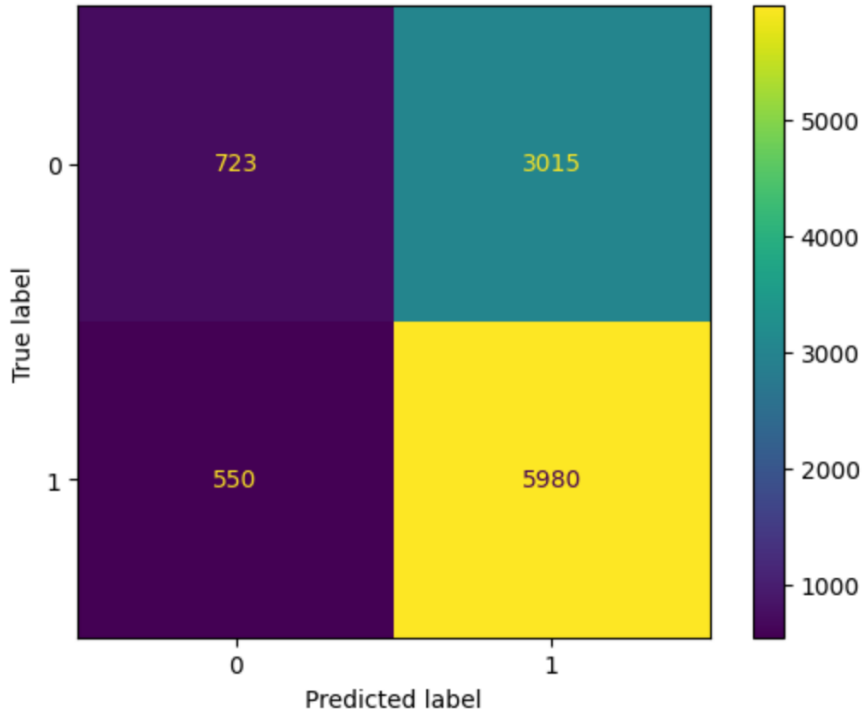
[320]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.1, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.3, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaves=None,
              min_child_weight=2, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=20, n_jobs=None,
              num_parallel_tree=None, objective='multi:softprob', ...)
```

We wil now export this model for age group prediction in our Flask applicatio

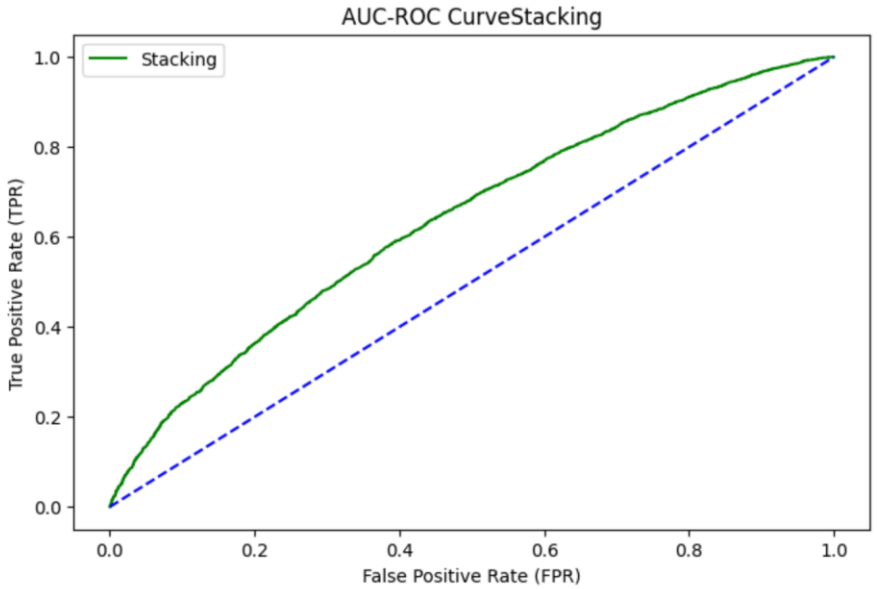
[321]:
```
pickle.dump(final_model_age_group, open("models/model_age_group.pkl","wb"))
```

Scenario 2 (Gender Prediction): Final Model – Stacking Classifier

```
----------------Model Statistics----------------
Gender Model Accuracy: 0.6528048305414881
Gender Model training Accuracy: 0.6577787517958458
Gender Model Precision: 0.6648137854363535
Gender Model Recall: 0.9157733537519143
Gender Model F1 Score: 0.7703703703703704
----------------Confusion Matrix----------------
<Figure size 800x500 with 0 Axes>
```




```
----------------Decile Analysis----------------
Top Probability: 0.12788807473802208
Bottom Probability: 0.8792568792312498
KS Statistic
Decile 0: 0.0032159264931087284
Decile 1: 0.003062787136294015
Decile 2: 0.003215926493108756
Decile 3: 0.00245022970903519
Decile 4: 0.0033690658499234694
Decile 5: 0.000612557427258853
Decile 6: 0.0004594180704440287
Decile 7: 0.0013782542113323082
Decile 8: 0.001990811638591161
Decile 9: 0.0007656967840734552
----------------ROC Curve----------------
ROC Area Under Curve SCORE ->  0.6369463037206555
```



Accuracy achieved after stacking is **0.6528048305414881**

Scenario 2 (Gender Prediction): Final Model – Stacking Classifier

[357]:
```
# Since Stacking model 3 has the highest accuracy we will cho
stacking_gender = stacking_gender_3
```

Gender Prediction - Final Model Selection

- LR - 0.6450136345929101
- RF - 0.6528048305414881
- XGB - 0.6521231008959876
- Stacking - 0.654460459680561

so we will chose the stacking gender model

[358]:
```
final_model_gender = stacking_gender
```

StackingCVClassifier

```
StackingCVClassifier(classifiers=[LogisticRegression(penalty='l1',
                                                     solver='liblinear'),
                                  XGBClassifier(base_score=None, booster=None,
                                                callbacks=None,
                                                colsample_bylevel=None,
                                                colsample_bynode=None,
                                                colsample_bytree=None,
                                                device=None,
                                                early_stopping_rounds=None,
                                                enable_categorical=False,
                                                eval_metric=None,
```

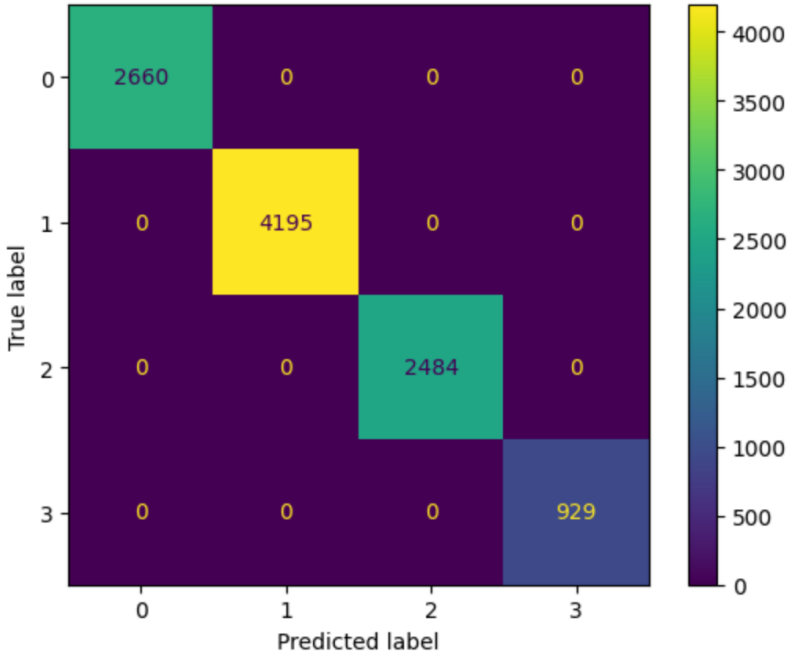
meta_classifier: RandomForestClassifier

```
RandomForestClassifier(max_depth=50, max_leaf_nodes=100, min_samples_leaf=3,
                       n_estimators=130, oob_score=True, random_state=100)
```

RandomForestClassifier

Scenario 2 (Age Group Prediction) – Best Model Stacking CV Classifier

```
----------Model Statistics----------
Age Group Model Accuracy: 1.0
Age Group Model training Accuracy: 1.0
Age Group Model Precision: 1.0
Age Group Model Recall: 1.0
Age Group Model F1 Score: 1.0
----------Confusion Matrix----------
<Figure size 800x500 with 0 Axes>
```




```
----------Multiclass Log Loss----------
Multiclass Log Loss: 0.0007
```

StackingCVClassifier

```
StackingCVClassifier(classifiers=[XGBClassifier(base_score=None, booster=None,
                                                callbacks=None,
                                                colsample_bylevel=None,
                                                colsample_bynode=None,
                                                colsample_bytree=None,
                                                device=None,
                                                early_stopping_rounds=None,
                                                enable_categorical=False,
                                                eval_metric=None,
                                                feature_types=None, gamma=0.1,
                                                grow_policy=None,
```

meta_classifier: LogisticRegression

```
LogisticRegression()
```

LogisticRegression

THANK YOU