Task-3 (Model Building)

1. Please mention the approximate amount of contribution made by each member of the group towards the Capstone Project. For example, Student A = 60% and Student B = 40%. There may be a viva towards the end to decide the final marks.

   Piyush - 60 %, Raghul 40 %

2. **Top five rows of the data set at the beginning of the analysis**

   We have exported 4 data files from Hadoop using Hive command. Those are :

   - non_event_train – Contains the Device information like age, gender and brand
   - event_train – Contains all the event related information like latitude longitude and timestamp
   - app_events – Contains the application details like app_id, is_active
   - app_label – Contains the application id and their corresponding label categories

   Top 5 rows of each of the above data frame at the beginning of analysis is given below :

   i) non_event_train

```
non_event_train.head()
```

]:

	Device_id	Gender	Age	Group_Train	Phone_Brand	Device_model
0	-1819925713085810000	F	23	F0-24	OPPO	N1 Mini
1	3670076507269740000	M	33	M32+	Meizu	menote1 2
2	5333872006968810000	M	34	M32+	Xiaomi	xnote
3	4216041491117040000	M	60	M32+	Ishi	ihv1
4	-3441149835823130000	M	30	M25-32	Huawei	è□£è€€ç•…çŽ©5X

   ii) event_train

```
event_train.head()
```

	Event_id	Device_id	TimeStamp	Longitude	Latitude
0	\N	\N	\N	\N	\N
1	\N	\N	\N	\N	\N
2	\N	\N	\N	\N	\N
3	2774404	-1001337759327040000	2016-05-07 09:14:24	119.61	29.7
4	3065018	-1001337759327040000	2016-05-04 10:26:14	120.29	30.42

iii) app_event

```
app_events.head()
```

	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

iv) app_label

```
app_label.head()
```

	app_id	category
0	7324884708820027918	Finance
1	-4494216993218550286	Finance
2	-6493194103110420302	IMF
3	2705437723590691734	IMF
4	-1402740782309296538	IMF

3. List of data cleaning techniques applied such as missing value treatment, etc.

i) The duplicate records for Device_id containing more than one record is handled as part of Hive table dump. In case of duplicate records, the first record was kept when the device_id had two phone brand categories.

ii) There were many records 51335 records in *event_train* dataframe containing '\N' values for all the columns. These rows were dropped

iii) Converting all the columns to their appropriate data type.
Ex : Converting Latitude column to Float data type

iv) There were around 31% data in *event_train* dataframe containing invalid values for latitude and longitude, i.e. the values were between -1 and 1.
These values cannot be imputed with Median or Mean as it will make the data more biased.
The approach we had taken is using Forward fill for imputation by making all these values as Null first and then using ffill() method of pandas

Step 1 – Identify the Device Id's which are having at least one valid entry for latitude and longitude
Step 2 - Fill the values of latitude and longitude for the Device Id's in Step 1 with null
Step 3 – Sort the dataframe by device_id and timestamp

Step 4 – Fill the null values using ffill() method
Step 5 – For Device ID's not having a single valid latitude, impute it with the median value of latitude and longitude
Since ffill() method was used there can be few records with null values (Step -1) since the first record for the device_id can be null due to which ffill() method left it as it is.
There are around 18,658 rows, which is around 1.5 % data.
These rows were dropped.

v) Removing duplicates from app_label dataframe. There were 2166 duplicate records with same Application_id and Category. These were removed

vi) Removing is_installed column from app_events dataframe. The is_installed column was having only 1 – value (1) across the dataframe. Hence this column was dropped.

vii) Created a new column called 'Hour' and 'Weekday' in event_train dataframe which was derived from the column TimeStamp.

4. Feature engineering techniques that were used along with proper reasoning to support why the technique was used

i) Grouping of Categories in app_label dataframe :
There were around 471 distinct label categories. Many of the categories were clubbed together under a single category.

```
## the number of distinct categories will be reduced by 72, if merged. Hence merging for Game categories
app_label['category']= app_label["category"].apply(lambda x: "Game" if x.lower().find('game') >=0 else x)
```

```
## Merging Property Industry
app_label['category']= app_label["category"].apply(lambda x: "Property Industry" if x.lower().find('property') >=0 else x)
```

```
app_label['category']= app_label["category"].apply(lambda x: "Property Industry" if x.lower().find('property') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Services" if x.lower().find('services') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Free" if x.lower().find('free') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Personal Effectiveness" if x.find('Personal Effectiveness') >=0
app_label['category']= app_label["category"].apply(lambda x: "Chess" if x.lower().find('chess') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Game" if x.lower().find('puzzle') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Relatives" if x.lower().find('relatives') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Finance" if x.lower().find('finance') >=0 else x)
app_label['category']= app_label["category"].apply(lambda x: "Shooting" if x.lower().find('shooting') >=0 else x)
```

After grouping we got 252 distinct categories and the Top 20 categories were selected and the remaining categories were categorized as 'Other'
This was finally merged with the non_event_train data frame by picking up the most frequently used application category for a device
Top 20 Categories were :

```
app_label['category'].value_counts().head(20)
```

```
Game                        70584
Property Industry           57895
Industry tag                56902
Custom                      55416
Tencent                     49320
Free                        21919
Services                    14873
Fun                         10777
Relatives                    9841
Personal Effectiveness       8497
Cards RPG                    7375
Chess                        5290
ARPG                         5288
The elimination of class     5124
And the Church               4619
round                        3608
Finance                      3020
Shooting                     2938
Bank                         2660
Low Liquidity                2546
Name: category, dtype: int64
```

After categorizing the label category, we performed a check on duplicate entries and removed the duplicate entries.

ii) New feature called 'Time_active(s)'. This column will tell how much a device has been active.
Below mentioned steps were followed for creating this column:

- Identify min() and max() timestamp and by grouping for weekday and hour for each device
- Idenitfy the active time by subtracting the max and min time
- Sum up the active time by grouping with Device_id, this will give the active time for each device
- Merge it with the non_event_train dataframe and fill it with value 0 for non_event device Id's

iii) New features called ‘Latitude_diff’ and ‘Longitude_diff’. These columns will store the average change in Latitude and Longitude for a Device ID.
Below mentioned steps were followed for deriving these columns:

- Calculate the max and minimum latitude for each hour,each day for each device id
- Identify the difference in latitude and longitude for each hour for each device id
- Calculate the average change in latitude and longitude for each device id

iv) Grouping of Phone_Brand and Device_model in non_event_train dataframe.
There are 1439 distinct device_model and 98 Phone_Brand.
The top 20 Phone_brand and top 50 Device_model were kept and the remaining was categorized as ‘Other’
This categorizing was done after splitting the non_event_train dataframe into two. One for Scenario 1 – Device Id’s containing event information and one for Scenario 2 – Records containing only Device ID information.

v) Average No. of applications:
For an event, the total number of applications is calculated and stored.
One device can have multiple events, so from the above average of applications is calculated and stored in non_event_train dataframe

vi) Average No. of Active applications:
For an event, the total number of applications which are active is calculated and stored.
One device can have multiple events, so from the above average of applications which are active is calculated and stored in non_event_train dataframe

vii) No, of events:
For a device we calculated the number of events that got triggered and stored it in the non_event_train datafram

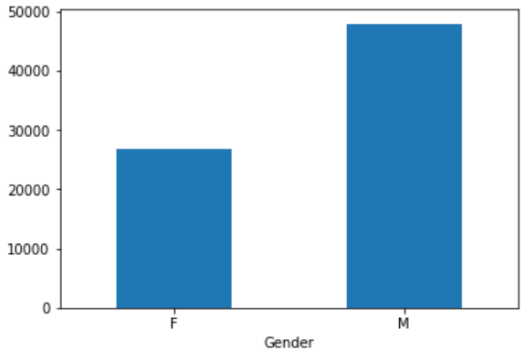
viii) Most Active hour:
From the event_train dataframe, the most active hour is derived and merged with non_event_train dataframe as Most_acitve_hour

ix) Most Active week:
From the event_train dataframe, the most active week is derived and merged with non_event_train dataframe as Most_acitve_week

5. Outputs to the various EDA and Visualisation codes along with the corresponding results and the insights gathered from each EDA and visualization

i) Plot appropriate graphs representing the distribution of age and gender in the data set [univariate]

```
#1Plot appropriate graphs representing the distribution of age and gender in the data set [univariate].
non_event_train.groupby('Gender')['Device_id'].count().plot.bar(x='Gender', y='Gender Count', rot=0)
plt.show()
```

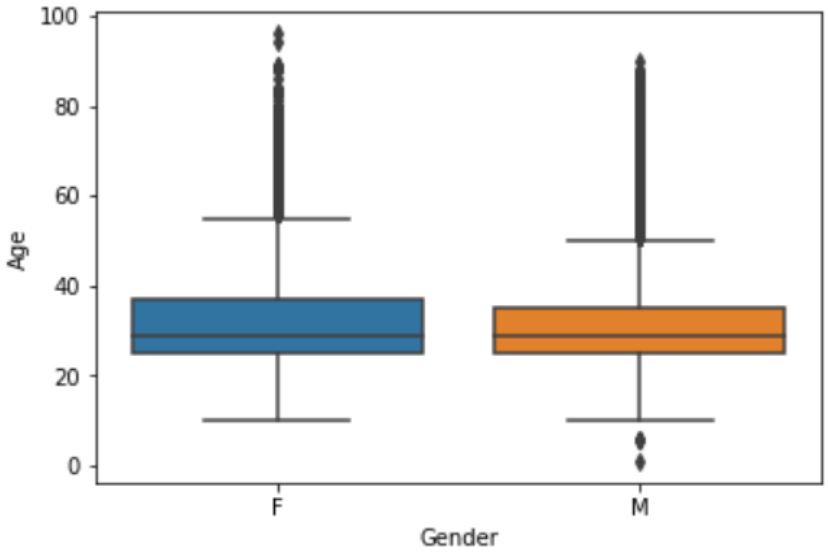


Conclusion: Male category is more in this dataset

ii) Boxplot analysis for gender and age [bivariate].

```
#2 Boxplot analysis for gender and age [bivariate]
sns.boxplot(x= non_event_train['Gender'], y = non_event_train['Age'])
```

```
<AxesSubplot:xlabel='Gender', ylabel='Age'>
```

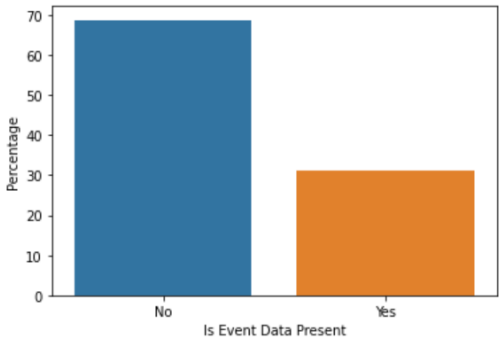


Conclusion: The data is more populated with the age group 20-40 for both Male and Female categories

iii) Plot the percentage of the device_ids with and without event data.

```
## 3. Plot the percentage of the device_ids with and without event data
graph= sns.barplot(data= tmp, x= 'Event_present', y='Percentage')
graph.set_xlabel('Is Event Data Present')
graph.set_ylabel('Percentage')
```

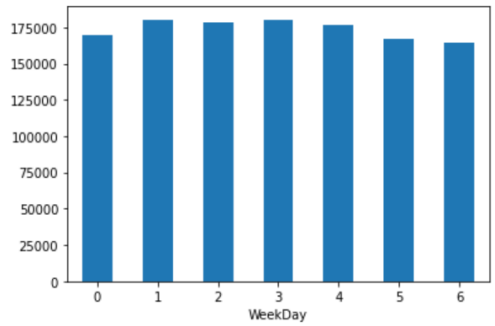
```
: Text(0, 0.5, 'Percentage')
```



Conclusion: The data without the Event information is predominantly more in this case study.

iv) Plot a graph representing the distribution of events over different days of a week.

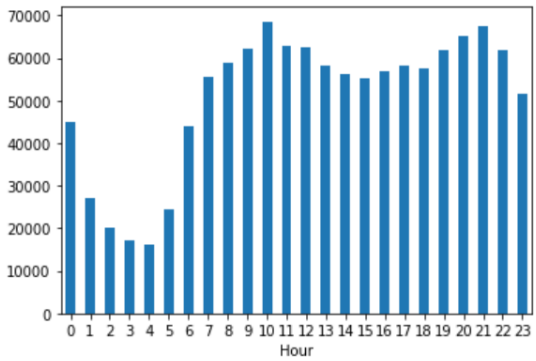
```
#4 Plot a graph representing the distribution of events over different days of a week
event_train.groupby('WeekDay')['Device_id'].count().plot.bar(x='WeekDay', y='WeekDay Count', rot=0)
plt.show()
```



Conclusion: The distribution of event is not different over various days of the week. Its almost same.

v) Plot a graph representing the distribution of events per hour [for one-week data]

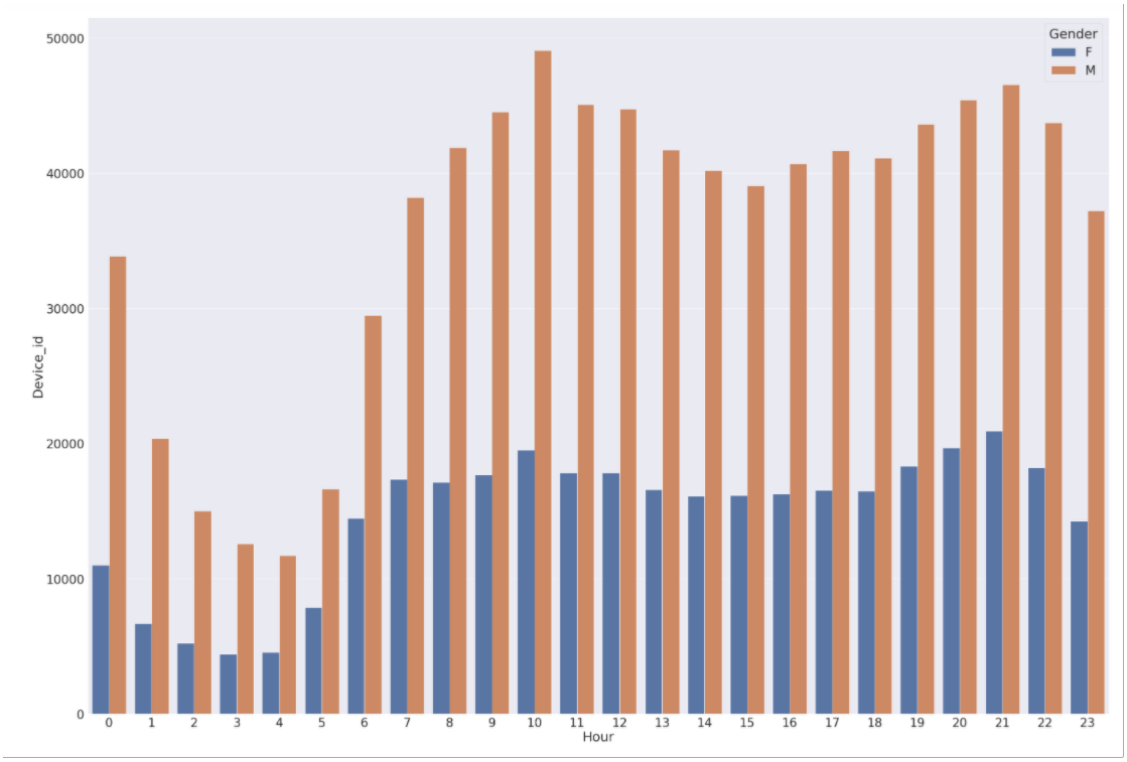
```
#5 Plot a graph representing the distribution of events per hour [for one-week data]
event_train.groupby('Hour')['Device_id'].count().plot.bar(x='Hour', y='Hour Count', rot=0)
plt.show()
```



Conclusion: Morning from 9am – 11am and evening from 8pm to 10pm there are more events triggered which suggest more people use their mobile at this time. And also very less event during midnight which makes sense since most people would be sleeping

vi) The difference in the distribution of events per hour for Male and Female consumers. [Show the difference using an appropriate chart for one-week data.]

```
#6 he difference in the distribution of events per hour for Male and Female consumers.
## [Show the difference using an appropriate chart for one-week data.]
plt.figure(figsize = (50,35))
sns.set(font_scale = 3)
temp_df= event_device_merged.groupby(['Hour', 'Gender'])['Device_id'].count().reset_index()
sns.barplot(data= temp_df, x= 'Hour', y='Device_id', hue= 'Gender')
```

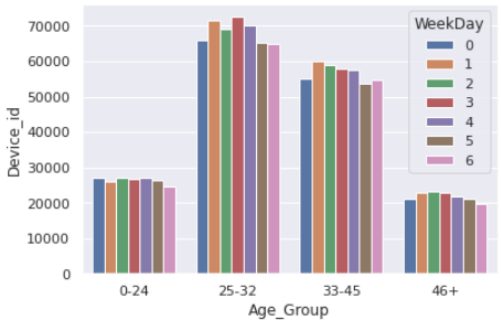


Conclusion: The count is more for every hour for male category which suggests male tend to use the phone more irrespective of the time in this case study

vii) Is there any difference in the distribution of Events for different Age Groups over different days of the week? [Consider the following age groups: 0–24, 25–32, 33–45, and 46+].

```
#7 Is there any difference in the distribution of Events for different Age Groups over different days of the week?
## [Consider the following age groups: 0-24, 25-32, 33-45, and 46+]
sns.set(font_scale = 1)
temp_df= event_device_merged.groupby(['Age_Group', 'WeekDay'])['Device_id'].count().reset_index()
sns.barplot(data= temp_df, x= 'Age_Group', y='Device_id', hue= 'WeekDay')
```

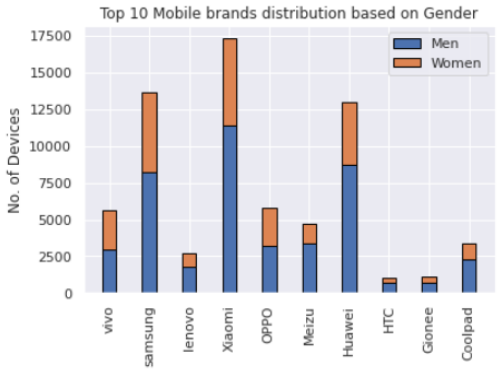
```
<AxesSubplot:xlabel='Age_Group', ylabel='Device_id'>
```



Conclusion: For different age over different days of the week its almost same. There isn't any difference in event counts over different days of the week.

viii) Stacked bar chart for the top 10 mobile brands across male and female consumers.

```
#8 Stacked bar chart for the top 10 mobile brands across male and female consumers
idx = np.arange(10)
width=0.35
p1= plt.bar(idx, temp_male['Device_id'].tolist(),width, edgecolor='black')
p2= plt.bar(idx, temp_female['Device_id'].tolist(), width, edgecolor='black',bottom=temp_male['Device_id'].tolist())
names = temp_male['Phone_Brand'].tolist()

plt.ylabel('No. of Devices')
plt.title('Top 10 Mobile brand''s distribution based on Gender')
plt.xticks(idx, names, rotation='vertical')
plt.legend((p1[0], p2[0]), ('Men', 'Women'))
plt.show()
```

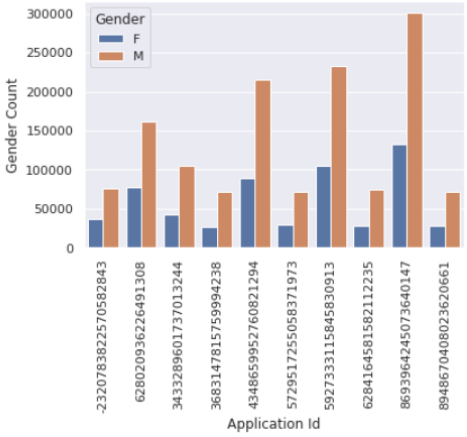


Conclusion: For every Mobile brand in the top 10 the Male category is more which is expected since the data in this case study has more records for Male category.

ix) Prepare a chart representing the ten frequently used applications and their respective male and female percentage

```
#9 Prepare a chart representing the ten frequently used applications and their respective male and female percentage.
temp_df= tmp_event_app_merge.groupby(['app_id', 'Gender'])['Device_id'].count().reset_index()
graph= sns.barplot(data= temp_df, x= 'app_id', y='Device_id', hue= 'Gender')
graph.set_xticklabels(graph.get_xticklabels(), rotation=90)
graph.set_xlabel('Application Id')
graph.set_ylabel('Gender Count')
```

```
]: Text(0, 0.5, 'Gender Count')
```

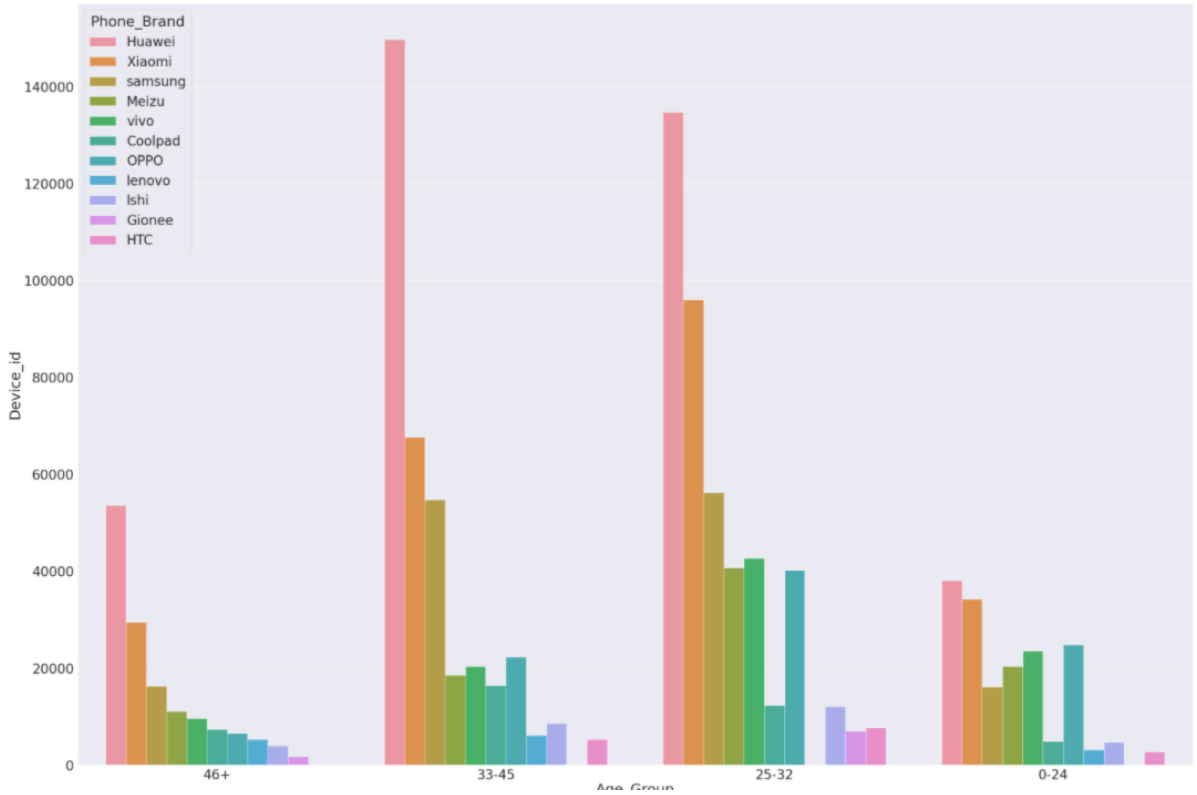


Conclusion: For every application in the top 10 the percentage of Male is more.

x) List the top 10 mobile phone brands bought by customers by age groups. [Consider the following age groups: 0–24, 25–32, 33–45, and 46+]

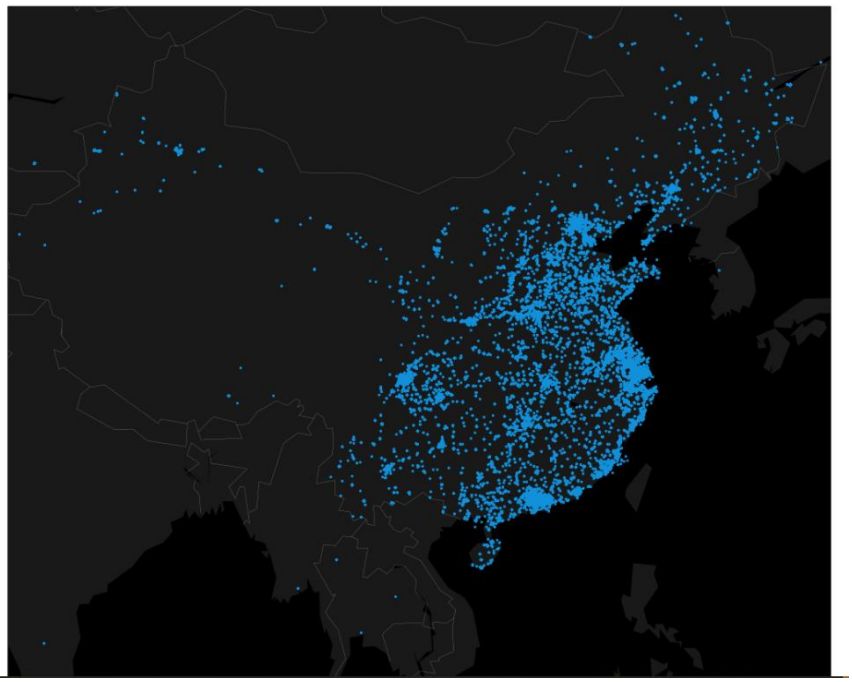
```
#10 List the top 10 mobile phone brands bought by customers by age groups.
##[Consider the following age groups: 0-24, 25-32, 33-45, and 46+]
plt.figure(figsize = (50,35))
temp_df= tmp.groupby('Age_Group').head(10)
sns.set(font_scale = 3)
sns.barplot(data= temp_df, x= 'Age_Group', y='Device_id', hue= 'Phone_Brand')
```

```
]: <AxesSubplot:xlabel='Age_Group', ylabel='Device_id'>
```



6. Geospatial visualizations along with the insights gathered from this visualization

i) Plot the visualization plot for a sample of 1 lakh data points



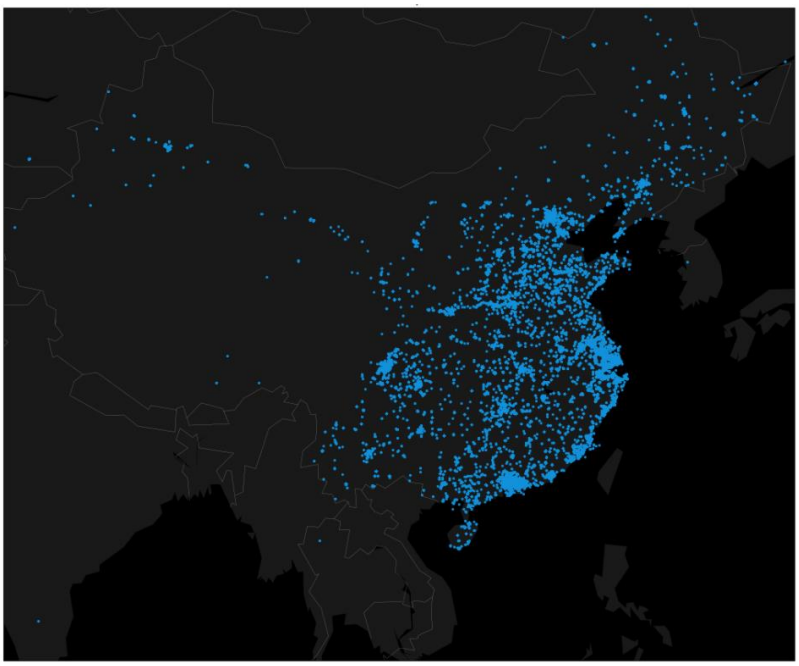
Conclusion: The majority the data is from China region

ii) Compare the event visualisation plots based on the users' gender information. [This can be done on the sample of 1 lakh data points.]

For Male:

```
## The above median value for latitude and longitude belongs to CHina region
## therefeore using the values of china map in basemap for visualization
fig = plt.figure(figsize=(18, 18))

# Mercator of World
m = Basemap(projection='merc',
            llcrnrlat=10,  #Latitude of lower left hand corner of the desired map domain
            urcrnrlat=51, #Latitude of upper right hand corner of the desired map domain
            llcrnrlon=75, #Longitude of lower left hand corner of the desired map domain
            urcrnrlon=135, #Longitude of upper right hand corner of the desired map domain
            lat_ts=0, #Latitude of true scale
            resolution='c') #resolution of boundary dataset being used - c for crude


m.fillcontinents(color='#191919',lake_color='#000000') # dark grey land, black lakes
m.drawmapboundary(fill_color='#000000')                # black background
m.drawcountries(linewidth=0.15, color="w")              # thin white line for country borders

# Plot the data
mxy = m(temp_df[temp_df['Gender']=='M']["Longitude"].tolist(), temp_df[temp_df['Gender']=='M']["Latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#1292db", zorder=2)    # zorder for the points

plt.title("Map")
plt.show()
```

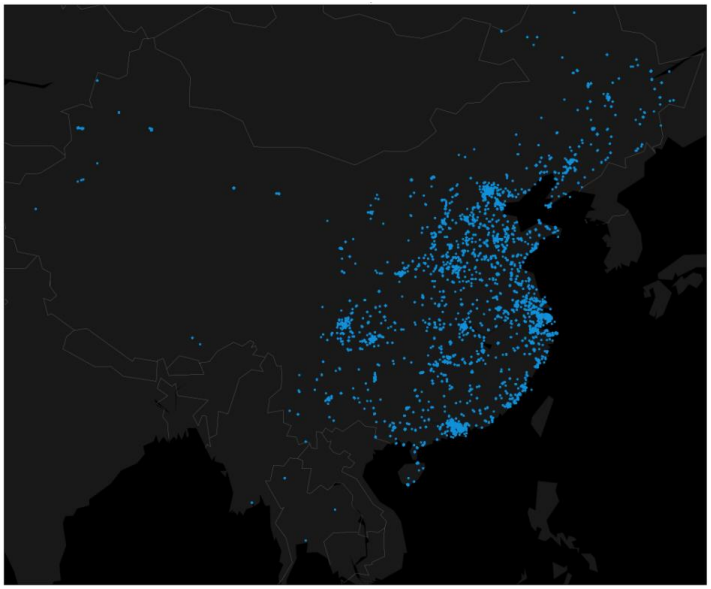
Map :



For Female :

```
## The above median value for latitude and longitude belongs to CHina region
## therefeore using the values of china map in basemap for visualization
fig = plt.figure(figsize=(18, 18))

# Mercator of World
m = Basemap(projection='merc',
            llcrnrlat=10,  #latitude of lower left hand corner of the desired map domain
            urcrnrlat=51, #Latitude of upper right hand corner of the desired map domain
            llcrnrlon=75, #Longitude of lower left hand corner of the desired map domain
            urcrnrlon=135, #Longitude of upper right hand corner of the desired map domain
            lat_ts=0, #Latitude of true scale
            resolution='c') #resolution of boundary dataset being used - c for crude


m.fillcontinents(color='#191919',lake_color='#000000') # dark grey land, black lakes
m.drawmapboundary(fill_color='#000000')                # black background
m.drawcountries(linewidth=0.15, color="w")              # thin white line for country borders

# Plot the data
mxy = m(temp_df[temp_df['Gender']=='F']["Longitude"].tolist(), temp_df[temp_df['Gender']=='F']["Latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#1292db", zorder=2)    # zorder for the points

plt.title("India Map")
plt.show()
```

Map:



Conclusion: The distribution of data for Male and Female isn't much different. They are all scattered across Eastern China.

iii) Compare the event visualisation plots based on the following age groups:

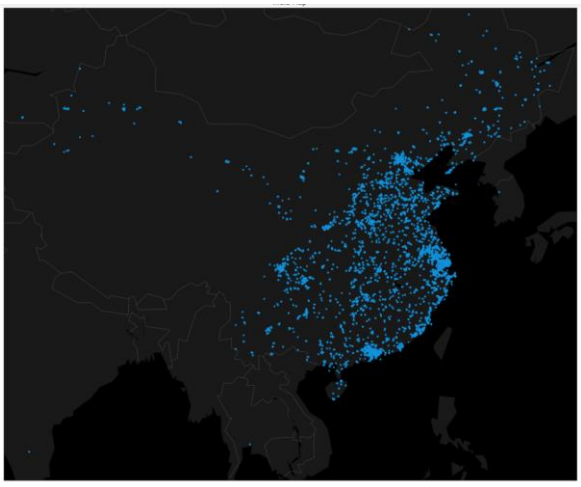
a.) 0-24

```
## The above median value for latitude and longitude belongs to CHina region
## therefeore using the values of china map in basemap for visualization
fig = plt.figure(figsize=(18, 18))

# Mercator of World
m = Basemap(projection='merc',
            llcrnrlat=10,  #Latitude of lower left hand corner of the desired map domain
            urcrnrlat=51, #Latitude of upper right hand corner of the desired map domain
            llcrnrlon=75, #Longitude of lower left hand corner of the desired map domain
            urcrnrlon=135, #Longitude of upper right hand corner of the desired map domain
            lat_ts=0, #Latitude of true scale
            resolution='c') #resolution of boundary dataset being used - c for crude


m.fillcontinents(color='#191919',lake_color='#000000') # dark grey land, black lakes
m.drawmapboundary(fill_color='#000000')                # black background
m.drawcountries(linewidth=0.15, color="w")              # thin white line for country borders

# Plot the data
mxy = m(temp_df[temp_df['Age_Group']=='0-24']["Longitude"].tolist(), temp_df[temp_df['Age_Group']=='0-24']["Latitude"].tolist
m.scatter(mxy[0], mxy[1], s=5, c="#1292db", zorder=2)    # zorder for the points

plt.title("India Map")
plt.show()
```

Map (0-24 Age Group) :

Map (25-32 Age Group)

Map(32+ Age Group)



Conclusion: The spatial distribution of data across different age groups is similar. Although there are very few data point for age group 0-24

7. Results interpreting the clusters formed as part of DBSCAN Clustering and how the cluster information is being used

The data that was fed into DBScan function was having only those records which were having correct values for the Latitude and Longitude

```
kms_per_radian = 6371.0088
epsilon = 2 / kms_per_radian
db = DBSCAN(eps=epsilon, min_samples=1, algorithm='ball_tree', metric='haversine').fit(np.radians(lat_long_cords))
```

```
cluster_labels = db.labels_
num_clusters = len(set(cluster_labels))
print('Number of clusters: {}'.format(num_clusters))
```

```
Number of clusters: 9909
```

For those records which were not having valid Latitude and Longitude values, they were populated with value 9910.

In [101]:
```
event_train= event_train.merge(dbscan_lat_log, how= 'left', on= ['Event_id','Latitude','Longitude'])
```

In [102]:
```
event_train.head()
```

Out[102]:

	Event_id	Device_id	TimeStamp	Longitude	Latitude	Hour	WeekDay	Cluster_label
0	1180629	-9222956879900150000	2016-05-06 15:31:56	113.24	23.19	15	4	0.0
1	1650018	-9222956879900150000	2016-05-06 15:32:26	113.24	23.19	15	4	0.0
2	2807359	-9222956879900150000	2016-05-06 15:32:54	113.24	23.19	15	4	0.0
3	2085015	-9222956879900150000	2016-05-06 15:33:24	113.24	23.19	15	4	0.0
4	229087	-9222956879900150000	2016-05-06 15:33:50	113.24	23.19	15	4	0.0

In [103]:
```
event_train.isnull().sum()
```

Out[103]:
```
Event_id              0
Device_id             0
TimeStamp             0
Longitude             0
Latitude              0
Hour                  0
WeekDay               0
Cluster_label    222059
dtype: int64
```

In [104]:
```
event_train['Cluster_label'].max()
```

Out[104]:
```
9908.0
```

In [105]:
```
## for all the events with non valid latitude and longitude values, cluster_label will be assigned as 9909
event_train.fillna(9909, inplace= True)
```

This column will be used as a Feature in the Model Building.

8. **A brief summary of any additional subtask that was performed and may have improved the data cleaning and feature generation step**
   i) Selected only top 20 application labels
   ii) Selected only top 20 phone_brand
   iii) Selected only top 50 device_model
   iv) Identifying event count for each device
   v) Identifying application count and active application count for each device
   vi) Identifying the most active hour and most active week for a device
   vii) Identifying the correlation between feature variables and plotting using heatmap..

9. **All the data preparation steps that were used before applying the ML algorithm**
   i) Categorizing the label categories in descending order of the count so that it can be fed to ML model and don't need to use one-hot or label-encoding
   ii) Categorizing the Phone_brand and Device_model categories in descending order of the count so that it can be fed to ML model and don't need to use one-hot or label-encoding
   iii) Since we are treating Age as Classification problem, encoding the Age as follows :
      - 1 for 0 -24
      - 2 for 25-32
      - 3 for 32+
   iv) Encoding Gender Column with 1 for Male and 0 for Female
   v) Dropping unique and unnecessary columns like Device_id and Group_Train
   vi) Used HeatMap for mapping the Correlation of feature variables and removed the column 'latitude_diff' (Latitude Difference) which was having high correlation with 'longitude_diff'
   vii) The final dataframe was split into two :
      - Scenario_1 – This dataframe contains Device and Event information.
      - Scenario_2 – This dataframe contains only Device Information.
   viii) The dataframe for each scenario was then split into training and testing data by using the 'train_test_flag' column provided in the train_test_split file.
   ix) Using StandardScaler() to scale the non-categorical features before feeding into ML models

10. **Documentation of all the machine learning models that were built along with the respective parameters that were used (e.g., DBSCAN, XGBoost, Random Forest, GridSearchCV, etc.)**

    i) DBScan :

```
kms_per_radian = 6371.0088
epsilon = 2 / kms_per_radian
db = DBSCAN(eps=epsilon, min_samples=1, algorithm='ball_tree', metric='haversine').fit(np.radians(lat_long_cords))
```

```
cluster_labels = db.labels_
num_clusters = len(set(cluster_labels))
print('Number of clusters: {}'.format(num_clusters))
```

```
Number of clusters: 9909
```

ii) There were 12 Machine Learning models created. 6 for Scenario 1 with 3 each for Age and Gender prediction and 6 for Scenario 2 with 3 each for Age and Gender prediction.

Scenario – 1 :

```
## train and test data for age and gender prediction
x_train_data_age_scenario_1 =train_data_scenario_1[train_data_scenario_1.columns[train_data_scenario_1.columns!= 'Age_Group']]
x_test_data_age_scenario_1 =test_data_scenario_1[test_data_scenario_1.columns[test_data_scenario_1.columns!= 'Age_Group']]

y_train_data_age_scenario_1 =train_data_scenario_1['Age_Group']
y_test_data_age_scenario_1 =test_data_scenario_1['Age_Group']

x_train_data_gender_scenario_1 =train_data_scenario_1[train_data_scenario_1.columns[train_data_scenario_1.columns!= 'Gender']]
x_test_data_gender_scenario_1 =test_data_scenario_1[test_data_scenario_1.columns[test_data_scenario_1.columns!= 'Gender']]

y_train_data_gender_scenario_1 =train_data_scenario_1['Gender']
y_test_data_gender_scenario_1 =test_data_scenario_1['Gender']
```

a.) Age Prediction :

- Logistic Regression

```
logisticRegr = LogisticRegression(multi_class= 'multinomial', solver='lbfgs')
logisticRegr.fit(x_train_data_age_scenario_1, y_train_data_age_scenario_1)
predictions = logisticRegr.predict(x_test_data_age_scenario_1)
```

```
/home/ec2-user/.local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

- XGBoost – using GridSearchCV

```
params = {
        'min_child_weight': [1, 5],
        'gamma': [0.5,  1.5,  5],
        'subsample': [0.6,  1.0],
        'colsample_by_tree': [0.8],
        'max_depth': [ 4, 5],
        'n_estimators': [60, 180],
        'learning_rate': [0.1,  0.05]
 }
```

```
skf = StratifiedKFold(n_splits=10, shuffle = True)
grid = GridSearchCV(xgb_model, params,
                    cv = skf.split(x_train_data_age_scenario_1, y_train_data_age_scenario_1),
                    scoring='accuracy',
                    return_train_score= True)
```

```
best_pars = grid.best_params_
```

```
best_pars
```

```
]: {'colsample_by_tree': 0.8,
    'gamma': 5,
    'learning_rate': 0.05,
    'max_depth': 4,
    'min_child_weight': 1,
    'n_estimators': 180,
    'subsample': 0.6}
```

- Stacking

```
# 1st set of models
clf1 = LogisticRegression()
clf2 = RandomForestClassifier(random_state=1, max_depth = 5, n_estimators=100,
                              min_samples_split=15, min_samples_leaf=20, max_leaf_nodes=10)
xgb = XGBClassifier(eval_metric = "logloss")

stacking_1 = StackingCVClassifier(classifiers=[clf1, clf2], meta_classifier=xgb, use_probas=True, cv=5)
```

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_1],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_age_scenario_1, y_train_data_age_scenario_1, cv=3, scoring='ac
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.41 (+/- 0.00) [lr]
Accuracy: 0.45 (+/- 0.01) [Random Forest]
Accuracy: 0.42 (+/- 0.01) [StackingClassifier]
```

b.) Gender Prediction

- Logistic Regression

```
logisticRegr_gender = LogisticRegression()
logisticRegr_gender.fit(x_train_data_gender_scenario_1, y_train_data_gender_scenario_1)
predictions_gender = logisticRegr_gender.predict(x_test_data_gender_scenario_1)
```

```
score_gender = logisticRegr_gender.score(x_test_data_gender_scenario_1, y_test_data_gender_scenario_1)
print(score_gender)
```

```
0.6561427590940289
```

- XGBoost using GridSearchCV

```
##XG Boost for predicting Gender

xgb_model = xgb.XGBClassifier()
skf = StratifiedKFold(n_splits=10, shuffle = True)
grid_gender = GridSearchCV(xgb_model, params,
                  cv = skf.split(x_train_data_gender_scenario_1, y_train_data_gender_scenario_1),
                  scoring='accuracy',
                  return_train_score= True)
```

```
grid_gender.fit(x_train_data_gender_scenario_1, y_train_data_gender_scenario_1)
```

```
best_pars_gender = grid_gender.best_params_
print(best_pars_gender)
```

```
{'colsample_by_tree': 0.8, 'gamma': 5, 'learning_rate': 0.05, 'max_depth': 5, 'min_child_weight': 5, 'n_estimators': 180, 's
ubsample': 0.6}
```

```
best_model_gender = grid_gender.best_estimator_
```

- Stacking

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_1],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_gender_scenario_1, y_train_data_gender_scenario_1, cv=5, scori
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.65 (+/- 0.00) [lr]
Accuracy: 0.65 (+/- 0.00) [Random Forest]
Accuracy: 0.64 (+/- 0.01) [StackingClassifier]
```

Scenario -2 :

```
## train and test data for age and gender prediction
x_train_data_age_scenario_2 =train_data_scenario_2[train_data_scenario_2.columns[train_data_scenario_2.columns!= 'Age_Group']
x_test_data_age_scenario_2 =test_data_scenario_2[test_data_scenario_2.columns[test_data_scenario_2.columns!= 'Age_Group']]

y_train_data_age_scenario_2 =train_data_scenario_2['Age_Group']
y_test_data_age_scenario_2 =test_data_scenario_2['Age_Group']

x_train_data_gender_scenario_2 =train_data_scenario_2[train_data_scenario_2.columns[train_data_scenario_2.columns!= 'Gender']
x_test_data_gender_scenario_2 =test_data_scenario_2[test_data_scenario_2.columns[test_data_scenario_2.columns!= 'Gender']]

y_train_data_gender_scenario_2 =train_data_scenario_2['Gender']
y_test_data_gender_scenario_2 =test_data_scenario_2['Gender']
```

a.) Age Prediction :

- Logistic Regression

```
logisticRegr_age_2 = LogisticRegression(multi_class= 'multinomial', solver='lbfgs')
logisticRegr_age_2.fit(x_train_data_age_scenario_2, y_train_data_age_scenario_2)
predictions_age_2 = logisticRegr_age_2.predict(x_test_data_age_scenario_2)
```

```
score_age_2 = logisticRegr_age_2.score(x_test_data_age_scenario_2, y_test_data_age_scenario_2)
print(score_age_2)
```

```
0.40855083755356447
```

- XGBoost using GridSearchCV

```
xgb_model = xgb.XGBClassifier()
skf = StratifiedKFold(n_splits=10, shuffle = True)
grid_age_2 = GridSearchCV(xgb_model, params,
                          cv = skf.split(x_train_data_age_scenario_2, y_train_data_age_scenario_2),
                          scoring='accuracy',
                          return_train_score= True)
```

```
grid_age_2.fit(x_train_data_age_scenario_2, y_train_data_age_scenario_2)
```

```
best_pars_age_2 = grid_age_2.best_params_
print(best_pars_age_2)
```

```
{'colsample_by_tree': 0.8, 'gamma': 0.5, 'learning_rate': 0.1, 'max_depth': 4, 'min_child_weight': 5, 'n_estimators': 60, 's
ubsample': 0.6}
```

- Stacking

```
# 1st set of models
clf1 = LogisticRegression()
clf2 = RandomForestClassifier(random_state=1, max_depth = 5, n_estimators=100,
                              min_samples_split=15, min_samples_leaf=20, max_leaf_nodes=10)
xgb = XGBClassifier(eval_metric = "logloss")

stacking_2 = StackingCVClassifier(classifiers=[clf1, clf2], meta_classifier=xgb, use_probas=True, cv=5)
```

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_2],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_age_scenario_2, y_train_data_age_scenario_2, cv=3, scoring='ac
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.41 (+/- 0.00) [lr]
Accuracy: 0.41 (+/- 0.00) [Random Forest]
Accuracy: 0.40 (+/- 0.00) [StackingClassifier]
```

b.) Gender Prediction:

- Logistic Regression

```
logisticRegr_gender_2 = LogisticRegression()
logisticRegr_gender_2.fit(x_train_data_gender_scenario_2, y_train_data_gender_scenario_2)
predictions_gender_2 = logisticRegr_gender_2.predict(x_test_data_gender_scenario_2)
```

```
score_gender_2 = logisticRegr_gender_2.score(x_test_data_gender_scenario_2, y_test_data_gender_scenario_2)
print(score_gender_2)
```

```
0.635956369302688
```

- XGBoost using GridSearchCV

```
xgb_model = xgb.XGBClassifier()
skf = StratifiedKFold(n_splits=10, shuffle = True)
grid_gender_2 = GridSearchCV(xgb_model, params,
                    cv = skf.split(x_train_data_gender_scenario_2, y_train_data_gender_scenario_2),
                    scoring='accuracy',
                    return_train_score= True)
```

```
grid_gender_2.fit(x_train_data_gender_scenario_2, y_train_data_gender_scenario_2)
```

```
best_pars_gender_2 = grid_age_2.best_params_
print(best_pars_gender_2)
```

```
{'colsample_by_tree': 0.8, 'gamma': 0.5, 'learning_rate': 0.1, 'max_depth': 4, 'min_child_weight': 5, 'n_estimators': 60, 's
ubsample': 0.6}
```

- Stacking

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_2],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_gender_scenario_2, y_train_data_gender_scenario_2, cv=3, scori
    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: 0.62 (+/- 0.00) [StackingClassifier]
```

11. The reason for using regression or classification for age prediction

We will be treating Age prediction as a multi-classification problem, since it would be easy to identify and promotions or offers for a particular group which is easy by predicting the group they fall in. In case of any offers or promotions or discounts that would be given, it is easier to target the group of audience rather than singling out. For example, we can give offer for any streaming apps like Amazon prime or Netflix we can target the age group 0-24, and any Investment related app offers to age group 25-32 etc.

And also when I treated this as a Regression model, the rmse value and the accuracy was very poor.

12. The outcomes of the evaluation metrics (results for both Scenario 1 and Scenario 2 must be shown separately).

<u>Age Prediction:</u>

Below are the models that were built for age prediction for both Scenario 1 and Scenario 2.

Scenario-1:

a. Logistic Regresiion:

i) Accuracy

```
score = logisticRegr.score(x_test_data_age_scenario_1, y_test_data_age_scenario_1)
print(score)
```

```
0.41901166781056964
```

ii) Confusion Matrix

```
confusion_matrix(y_test_data_age_scenario_1, y_preds1)
```

```
array([[   2,  796,  404],
       [   3, 1599,  782],
       [   7, 1394,  841]])
```

iii) Classification Report

```
y_preds1= logisticRegr.predict(x_test_data_age_scenario_1)
print(classification_report(y_test_data_age_scenario_1, y_preds1))
```

```
              precision    recall  f1-score   support

           1       0.17      0.00      0.00      1202
           2       0.42      0.67      0.52      2384
           3       0.41      0.38      0.39      2242

    accuracy                           0.42      5828
   macro avg       0.33      0.35      0.31      5828
weighted avg       0.37      0.42      0.36      5828
```

iv) Multi class Log Loss Score

```
log_loss(y_test_data_age_scenario_1, ypred_prob)
```

```
1.0572053887572366
```

b. XGBoost

i) Accuracy

```
best_model.score(x_test_data_age_scenario_1, y_test_data_age_scenario_1)
```

```
0.4543582704186685
```

ii) Confusion Matrix

```
confusion_matrix(y_test_data_age_scenario_1, y_preds1)
```

```
array([[  10,  827,  365],
       [  10, 1385,  989],
       [   2,  987, 1253]])
```

iii) Classification Report

```
confusion_matrix(y_test_data_age_scenario_1, y_preds1)
# Print the confusion matrix
print(classification_report(y_test_data_age_scenario_1, y_preds1))
```

```
              precision    recall  f1-score   support

           1       0.45      0.01      0.02      1202
           2       0.43      0.58      0.50      2384
           3       0.48      0.56      0.52      2242

    accuracy                           0.45      5828
   macro avg       0.46      0.38      0.34      5828
weighted avg       0.46      0.45      0.41      5828
```

iv) Multi class Log Loss Score

```
##Log loss
ypred_prob = best_model.predict_proba(x_test_data_age_scenario_1)
log_loss(y_test_data_age_scenario_1, ypred_prob)
```

```
1.025850849204384
```

c. Stacking:

i) Accuracy

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_1],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_age_scenario_1, y_train_data_age_scenario_1, cv=3, scoring='ac
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.41 (+/- 0.00) [lr]
Accuracy: 0.45 (+/- 0.01) [Random Forest]
Accuracy: 0.42 (+/- 0.01) [StackingClassifier]
```

ii) Confusion Matrix

```
stack_age_1= stacking_1.fit(x_train_data_age_scenario_1, y_train_data_age_scenario_1)
y_preds_age1= stack_age_1.predict(x_test_data_age_scenario_1)
confusion_matrix(y_test_data_age_scenario_1, y_preds_age1)
```

```
]: array([[  74,  694,  434],
         [  88, 1289, 1007],
         [  60, 1011, 1171]])
```

iii) Classification Report

```
print(classification_report(y_test_data_age_scenario_1, y_preds_age1))
```

```
              precision    recall  f1-score   support

           1       0.33      0.06      0.10      1202
           2       0.43      0.54      0.48      2384
           3       0.45      0.52      0.48      2242

    accuracy                           0.43      5828
   macro avg       0.40      0.37      0.36      5828
weighted avg       0.42      0.43      0.40      5828
```

iv) Multi class Log Loss Score

```
ypred_prob = stack_age_1.predict_proba(x_test_data_age_scenario_1)
log_loss(y_test_data_age_scenario_1, ypred_prob)
```

```
25]: 1.0614931971696382
```

Scenario -2 :

a. Logistic Regression

i) Accuracy

```
score_age_2 = logisticRegr_age_2.score(x_test_data_age_scenario_2, y_test_data_age_scenario_2)
print(score_age_2)
```

```
0.40855083755356447
```

ii) Confusion Matrix

```
confusion_matrix(y_test_data_age_scenario_2, y_preds1)
```

```
array([[   0, 2660,    0],
       [   0, 4195,    0],
       [   0, 3413,    0]])
```

iii) Classification Report

```
y_preds1= logisticRegr_age_2.predict(x_test_data_age_scenario_2)
print(classification_report(y_test_data_age_scenario_2, y_preds1))
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00      2660
           2       0.41      1.00      0.58      4195
           3       0.00      0.00      0.00      3413

    accuracy                           0.41     10268
   macro avg       0.14      0.33      0.19     10268
weighted avg       0.17      0.41      0.24     10268
```

iv) Multi class Log Loss Score

```
ypred_prob = logisticRegr_age_2.predict_proba(x_test_data_age_scenario_2)
log_loss(y_test_data_age_scenario_2, ypred_prob)
```

```
1.080971227199126
```

b. XGBosst

i) Accuracy

```
best_model_age_2.score(x_test_data_age_scenario_2, y_test_data_age_scenario_2)
```

```
0.4152707440592131
```

ii) Confusion Matrix

```
confusion_matrix(y_test_data_age_scenario_2, y_preds_age_2)
```

```
array([[  47, 2293,  320],
       [  54, 3561,  580],
       [  36, 2721,  656]])
```

iii) Classification Report

```
print(classification_report(y_test_data_age_scenario_2, y_preds_age_2))
```

```
              precision    recall  f1-score   support

           1       0.34      0.02      0.03      2660
           2       0.42      0.85      0.56      4195
           3       0.42      0.19      0.26      3413

    accuracy                           0.42     10268
   macro avg       0.39      0.35      0.29     10268
weighted avg       0.40      0.42      0.32     10268
```

iv) Multi class Log Loss Score

```
ypred_prob = best_model_age_2.predict_proba(x_test_data_age_scenario_2)
log_loss(y_test_data_age_scenario_2, ypred_prob)
```

```
]: 1.0619213066133117
```

c. Stacking

i) Accuracy

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_2],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_age_scenario_2, y_train_data_age_scenario_2, cv=3, scoring='ac
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.41 (+/- 0.00) [lr]
Accuracy: 0.41 (+/- 0.00) [Random Forest]
Accuracy: 0.41 (+/- 0.00) [StackingClassifier]
```

ii) Confusion Matrix

```
stack_age_2= stacking_2.fit(x_train_data_age_scenario_2, y_train_data_age_scenario_2)
y_preds_age2= stack_age_2.predict(x_test_data_age_scenario_2)
confusion_matrix(y_test_data_age_scenario_2, y_preds_age2)
```

```
]: array([[ 297, 1885,  478],
         [ 294, 2991,  910],
         [ 175, 2300,  938]])
```

iii) Classification Report

```
print(classification_report(y_test_data_age_scenario_2, y_preds_age2))
```

```
              precision    recall  f1-score   support

           1       0.39      0.11      0.17      2660
           2       0.42      0.71      0.53      4195
           3       0.40      0.27      0.33      3413

    accuracy                           0.41     10268
   macro avg       0.40      0.37      0.34     10268
weighted avg       0.40      0.41      0.37     10268
```

iv) Multi class Log Loss Score

```
ypred_prob = stack_age_2.predict_proba(x_test_data_age_scenario_2)
log_loss(y_test_data_age_scenario_2, ypred_prob)
```

```
]: 1.077680039827258
```

Evaluation metric for Gender Prediction:

Below are the models that were built for age prediction for both Scenario 1 and Scenario 2.

Scenario-1:

a. Logistic Regression

i) Accuracy

```
score_gender = logisticRegr_gender.score(x_test_data_gender_scenario_1, y_test_data_gender_scenario_1)
print(score_gender)
```

```
0.6561427590940289
```

ii) Confusion Matrix (F1 Score, Precision, Recall)

```
y_preds1= logisticRegr_gender.predict(x_test_data_gender_scenario_1)
print(classification_report(y_test_data_gender_scenario_1, y_preds1))
```

```
              precision    recall  f1-score   support

           0       0.48      0.02      0.04      2001
           1       0.66      0.99      0.79      3827

    accuracy                           0.66      5828
   macro avg       0.57      0.50      0.41      5828
weighted avg       0.60      0.66      0.53      5828
```

```
confusion_matrix(y_test_data_gender_scenario_1, y_preds1)
```

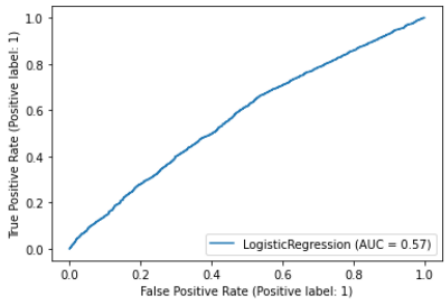
```
array([[  41, 1960],
       [  44, 3783]])
```

iii) ROC Curve and AUC

```
##roc_auc_score
ypred_prob = logisticRegr_gender.predict_proba(x_test_data_gender_scenario_1)
roc_auc_score(y_test_data_gender_scenario_1, ypred_prob[:, 1])
```

```
0.5732674556372193
```

```
## ROC Curve
metrics.plot_roc_curve(logisticRegr_gender, x_test_data_gender_scenario_1, y_test_data_gender_scenario_1)
plt.show()
```



b. XGBoost

i) Accuracy

```
best_model_gender.score(x_test_data_gender_scenario_1, y_test_data_gender_scenario_1)
```

```
0.662834591626663
```

ii) Confusion Matrix (F1 Score, Precision, Recall)

```
##Confusion matric
confusion_matrix(y_test_data_gender_scenario_1, y_preds1)
```

```
array([[ 232, 1769],
       [ 196, 3631]])
```

```
## Classification report
print(classification_report(y_test_data_gender_scenario_1, y_preds1))
```

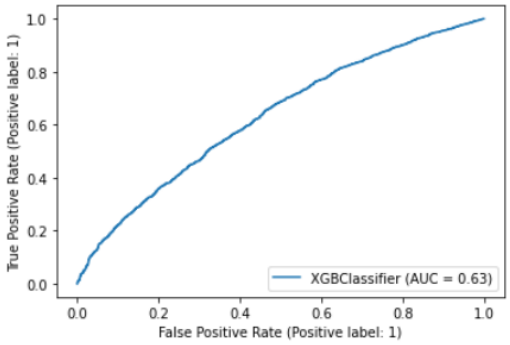
```
              precision    recall  f1-score   support

           0       0.54      0.12      0.19      2001
           1       0.67      0.95      0.79      3827

    accuracy                           0.66      5828
   macro avg       0.61      0.53      0.49      5828
weighted avg       0.63      0.66      0.58      5828
```

iii) ROC Curve and AUC

```
##roc_auc_score
ypred_prob = best_model_gender.predict_proba(x_test_data_gender_scenario_1)
roc_auc_score(y_test_data_gender_scenario_1, ypred_prob[:, 1])
```

```
0.6305244294497643
```

```
## ROC Curve
metrics.plot_roc_curve(best_model_gender, x_test_data_gender_scenario_1, y_test_data_gender_scenario_1)
plt.show()
```



c. Stacking

i) Accuracy

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_1],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_gender_scenario_1, y_train_data_gender_scenario_1, cv=5, scori
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.65 (+/- 0.00) [lr]
Accuracy: 0.65 (+/- 0.00) [Random Forest]
Accuracy: 0.64 (+/- 0.01) [StackingClassifier]
```

ii) Confusion Matrix (F1 Score, Precision, Recall)

```
stack_gender_1= stacking_1.fit(x_train_data_gender_scenario_1, y_train_data_gender_scenario_1)
y_preds_gender1= stack_gender_1.predict(x_test_data_gender_scenario_1)
confusion_matrix(y_test_data_gender_scenario_1, y_preds_gender1)
```

```
38]: array([[ 241, 1760],
       [ 284, 3543]])
```

```
print(classification_report(y_test_data_gender_scenario_1, y_preds_gender1))
```

```
              precision    recall  f1-score   support

           0       0.46      0.12      0.19      2001
           1       0.67      0.93      0.78      3827

    accuracy                           0.65      5828
   macro avg       0.56      0.52      0.48      5828
weighted avg       0.60      0.65      0.58      5828
```

iii) ROC AUC Score

```
##roc_auc_score
ypred_prob = stack_gender_1.predict_proba(x_test_data_gender_scenario_1)
roc_auc_score(y_test_data_gender_scenario_1, ypred_prob[:, 1])
```

```
77]: 0.5766640458187421
```

Scenario -2 :

a. Logistic Regression

i) Accuracy

```
score_gender_2 = logisticRegr_gender_2.score(x_test_data_gender_scenario_2, y_test_data_gender_scenario_2)
print(score_gender_2)
```

```
0.635956369302688
```

ii) Confusion Matrix (F1 Score, Precision, Recall)

```
##classification report
y_preds1= logisticRegr_gender_2.predict(x_test_data_gender_scenario_2)
print(classification_report(y_test_data_gender_scenario_2, y_preds1))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3738
           1       0.64      1.00      0.78      6530

    accuracy                           0.64     10268
   macro avg       0.32      0.50      0.39     10268
weighted avg       0.40      0.64      0.49     10268
```

```
##confusion matrix
confusion_matrix(y_test_data_gender_scenario_2, y_preds1)
```

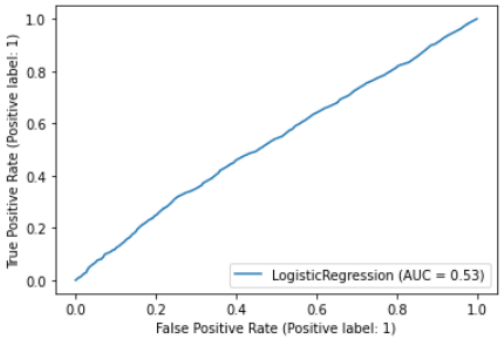
```
36]: array([[   0, 3738],
       [   0, 6530]])
```

iii) ROC Curve and AUC

```
##roc_auc_score
ypred_prob = logisticRegr_gender_2.predict_proba(x_test_data_gender_scenario_2)
roc_auc_score(y_test_data_gender_scenario_2, ypred_prob[:, 1])
```

```
]: 0.5320357661105635
```

```
## ROC Curve
metrics.plot_roc_curve(logisticRegr_gender_2, x_test_data_gender_scenario_2, y_test_data_gender_scenario_2)
plt.show()
```



b. XGBoost

i) Accuracy

```
best_model_gender_2.score(x_test_data_gender_scenario_2, y_test_data_gender_scenario_2)
```

```
34]: 0.637709388391118
```

ii) Confusion Matrix (F1 Score, Precision, Recall)

```
confusion_matrix(y_test_data_gender_scenario_2, y_preds_gender_2)
```

```
]: array([[  40, 3698],
         [  22, 6508]])
```

```
print(classification_report(y_test_data_gender_scenario_2, y_preds_gender_2))
```

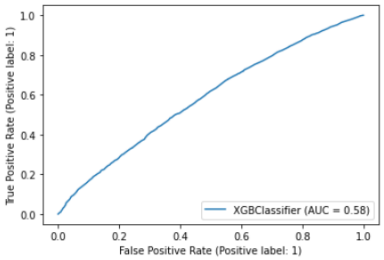
```
              precision    recall  f1-score   support

           0       0.65      0.01      0.02      3738
           1       0.64      1.00      0.78      6530

    accuracy                           0.64     10268
   macro avg       0.64      0.50      0.40     10268
weighted avg       0.64      0.64      0.50     10268
```

iii) ROC Curve and AUC

```
##roc_auc_score
ypred_prob = best_model_gender_2.predict_proba(x_test_data_gender_scenario_2)
roc_auc_score(y_test_data_gender_scenario_2, ypred_prob[:, 1])
```

```
1]: 0.5831312164213897
```

```
## ROC Curve
metrics.plot_roc_curve(best_model_gender_2, x_test_data_gender_scenario_2, y_test_data_gender_scenario_2)
plt.show()
```



c. Stacking

i) Accuracy

```
# Do CV
for clf, label in zip([clf1, clf2, stacking_2],
                      ['lr',
                       'Random Forest',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train_data_gender_scenario_2, y_train_data_gender_scenario_2, cv=3, scori
    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: 0.62 (+/- 0.00) [StackingClassifier]
```

ii) Confusion Matrix (F1 Score, Precision, Recall)

```
stack_gender_2= stacking_2.fit(x_train_data_gender_scenario_2, y_train_data_gender_scenario_2)
y_preds_gender2= stack_gender_2.predict(x_test_data_gender_scenario_2)
confusion_matrix(y_test_data_gender_scenario_2, y_preds_gender2)
```

```
3]: array([[ 239, 3499],
           [ 306, 6224]])
```

```
print(classification_report(y_test_data_gender_scenario_2, y_preds_gender2))
```

```
              precision    recall  f1-score   support

           0       0.44      0.06      0.11      3738
           1       0.64      0.95      0.77      6530

    accuracy                           0.63     10268
   macro avg       0.54      0.51      0.44     10268
weighted avg       0.57      0.63      0.53     10268
```

iii) ROC Curve and AUC

```
##roc_auc_score
ypred_prob = stack_gender_2.predict_proba(x_test_data_gender_scenario_2)
roc_auc_score(y_test_data_gender_scenario_2, ypred_prob[:, 1])
```

```
)4]: 0.5475360868920413
```

Conclusion of Best Models for Scenario - 1:

Age Prediction -> Out of 3 modes, XGBoost model gave the best accuracy score, therefore it's the best model for predicting age for this data.

Gender Predition -> Out of 3 models, Logistic Regression model gave the best accuracy score therefore it's the best model for this data.

The above two model will be dumped and used to build and deploy the Model in EC2 using Flask