Capstone Project

<u>REPORT</u>

The Capstone Project is based on the domain of telecom sector. This project is a work on telecom data and is required to develop a model to help the company run different marketing campaigns and data monetization activities around it.

1. Problem Statement

Suppose XYZ Company, a telecom giant, wants to conduct some customer-driven campaigns as per certain users' gender and age preferences. However, the company may not rely on the Aadhaar information provided by the users at the time of purchasing SIM cards. So, in this Capstone Project, you will be **predicting the demographics of certain users (gender and age) based on their app download and usage behavior.**

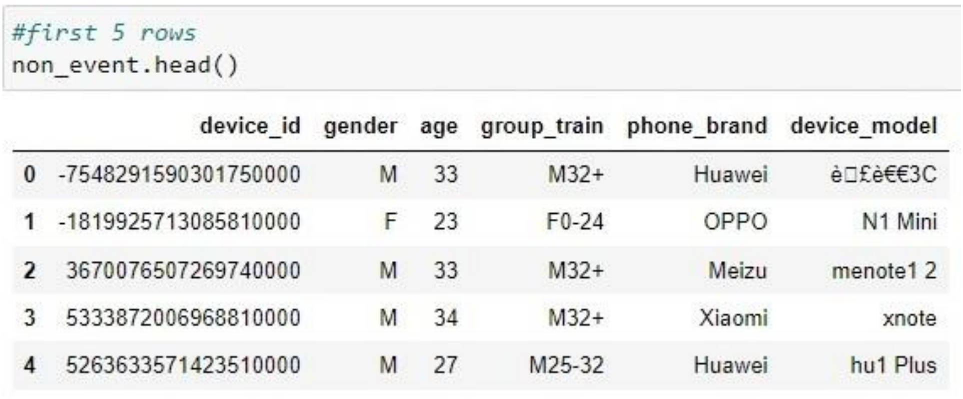
Use Case: Target-specific ad campaigns improving the customer experience and also open avenues for revenue generation through cross and upsell activities

2. Data

There are 4 main datasets that we deal with in this Capstone. In the initial stage itself two of them are merged and formed a new dataset, thus resulting in the 3 datasets that are used throughout this project

Datasets:

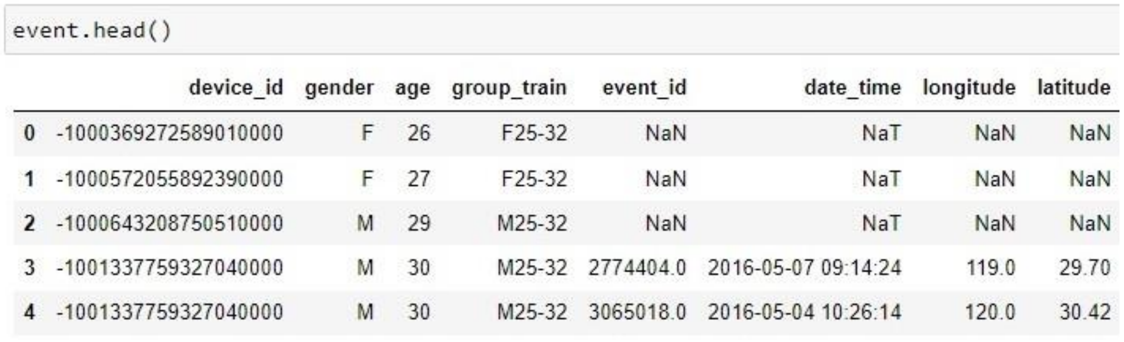
1. Non_event dataset - User activity is unavailable, only the device information is provided



```
#first 5 rows
non_event.head()
```

	device_id	gender	age	group_train	phone_brand	device_model
0	-7548291590301750000	M	33	M32+	Huawei	è□£è€€3C
1	-1819925713085810000	F	23	F0-24	OPPO	N1 Mini
2	3670076507269740000	M	33	M32+	Meizu	menote1 2
3	5333872006968810000	M	34	M32+	Xiaomi	xnote
4	5263633571423510000	M	27	M25-32	Huawei	hu1 Plus

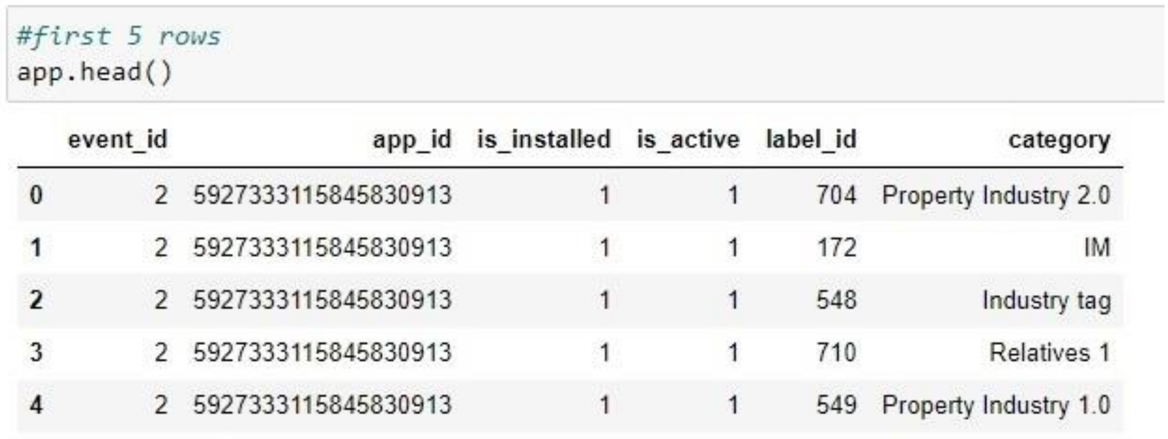
2. Event dataset - User allows the application to trigger events that collect their usage behaviour



```
event.head()
```

	device_id	gender	age	group_train	event_id	date_time	longitude	latitude
0	-1000369272589010000	F	26	F25-32	NaN	NaT	NaN	NaN
1	-1000572055892390000	F	27	F25-32	NaN	NaT	NaN	NaN
2	-1000643208750510000	M	29	M25-32	NaN	NaT	NaN	NaN
3	-1001337759327040000	M	30	M25-32	2774404.0	2016-05-07 09:14:24	119.0	29.70
4	-1001337759327040000	M	30	M25-32	3065018.0	2016-05-04 10:26:14	120.0	30.42

3. App dataset – Application information (app category, installed/active etc)

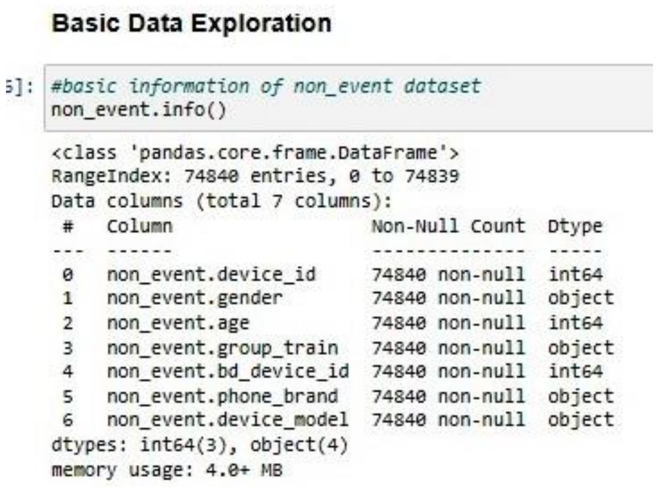


```
#first 5 rows
app.head()
```

	event_id	app_id	is_installed	is_active	label_id	category
0	2	5927333115845830913	1	1	704	Property Industry 2.0
1	2	5927333115845830913	1	1	172	IM
2	2	5927333115845830913	1	1	548	Industry tag
3	2	5927333115845830913	1	1	710	Relatives 1
4	2	5927333115845830913	1	1	549	Property Industry 1.0

2.1 Data Cleaning

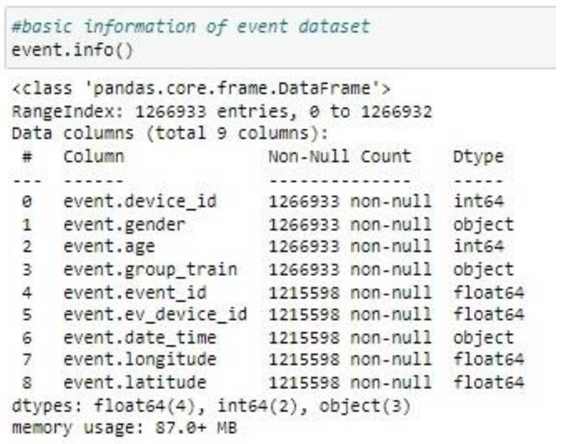
1. Basic data exploration, removing columns and renaming columns



Basic Data Exploration

```
5]: #basic information of non_event dataset
    non_event.info()
```

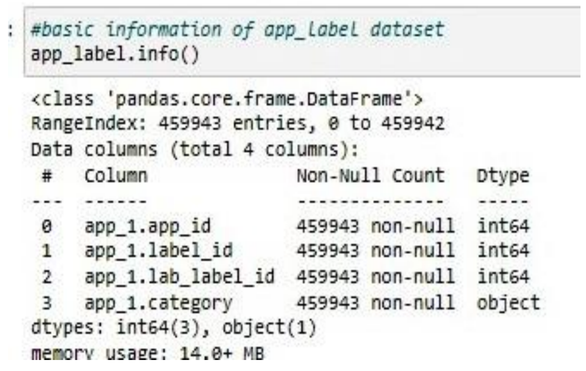
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74840 entries, 0 to 74839
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   non_event.device_id    74840 non-null  int64
 1   non_event.gender       74840 non-null  object
 2   non_event.age          74840 non-null  int64
 3   non_event.group_train  74840 non-null  object
 4   non_event.bd_device_id 74840 non-null  int64
 5   non_event.phone_brand  74840 non-null  object
 6   non_event.device_model 74840 non-null  object
dtypes: int64(3), object(4)
memory usage: 4.0+ MB
```



```
#basic information of event dataset
event.info()
```

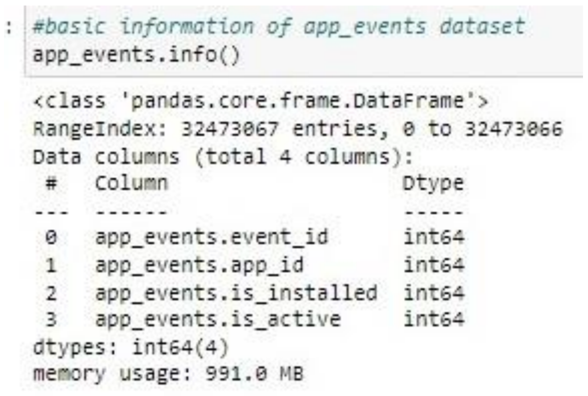
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1266933 entries, 0 to 1266932
Data columns (total 9 columns):
 #   Column             Non-Null Count    Dtype  
---  ------             --------------    -----  
 0   event.device_id    1266933 non-null  int64  
 1   event.gender       1266933 non-null  object 
 2   event.age          1266933 non-null  int64  
 3   event.group_train  1266933 non-null  object 
 4   event.event_id     1215598 non-null  float64
 5   event.ev_device_id 1215598 non-null  float64
 6   event.date_time    1215598 non-null  object 
 7   event.longitude    1215598 non-null  float64
 8   event.latitude     1215598 non-null  float64
dtypes: float64(4), int64(2), object(3)
memory usage: 87.0+ MB
```

The datasets given below are merged to form the final dataset called app



```
#basic information of app_label dataset
app_label.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 459943 entries, 0 to 459942
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   app_1.app_id       459943 non-null  int64 
 1   app_1.label_id     459943 non-null  int64 
 2   app_1.lab_label_id 459943 non-null  int64 
 3   app_1.category     459943 non-null  object
dtypes: int64(3), object(1)
memory usage: 14.0+ MB
```



```
#basic information of app_events dataset
app_events.info()
```

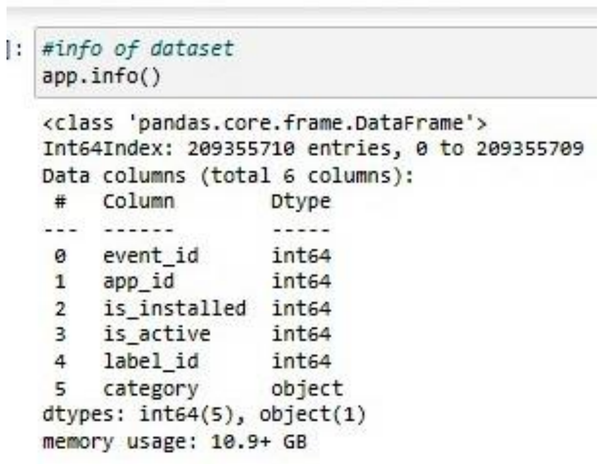
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32473067 entries, 0 to 32473066
Data columns (total 4 columns):
 #   Column                  Dtype
---  ------                  -----
 0   app_events.event_id      int64
 1   app_events.app_id        int64
 2   app_events.is_installed  int64
 3   app_events.is_active     int64
dtypes: int64(4)
memory usage: 991.0 MB
```

The basic data exploration shows the following findings:

- Non_event, event and app_label datasets have repeated columns, which is a result of the sql left joins done as a part of ETL. This need to be removed.
- The columns need to be renamed, since it contains the table names along with the actual column names.
- App_label and app_events data need to be joined to form a single dataset(app_data)

The columns are removed and renamed accordingly and is taken forward to other data cleaning steps

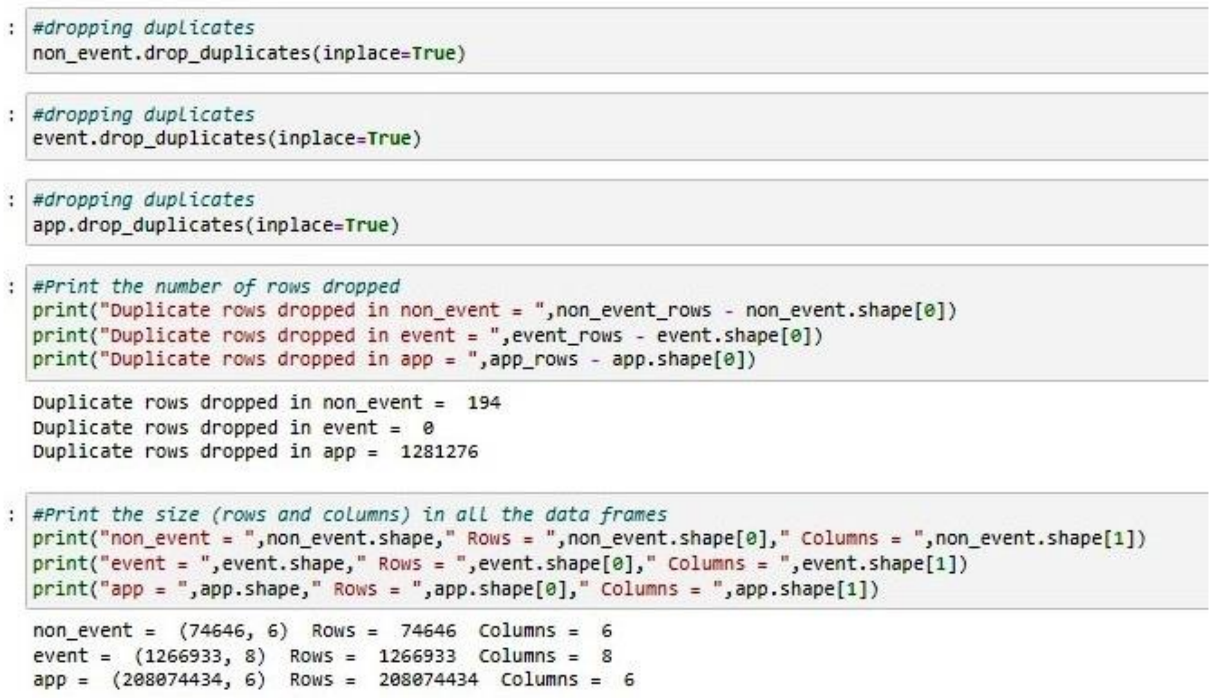
App_labels and app_events datasets are merged to form the app dataset. The basic data exploration changes that has been reflected on this merged data's information



```
]: #info of dataset
   app.info()

   <class 'pandas.core.frame.DataFrame'>
   Int64Index: 209355710 entries, 0 to 209355709
   Data columns (total 6 columns):
    #   Column        Dtype
   ---  ------        -----
    0   event_id      int64
    1   app_id        int64
    2   is_installed  int64
    3   is_active     int64
    4   label_id      int64
    5   category      object
   dtypes: int64(5), object(1)
   memory usage: 10.9+ GB
```

2. Dropping duplicates

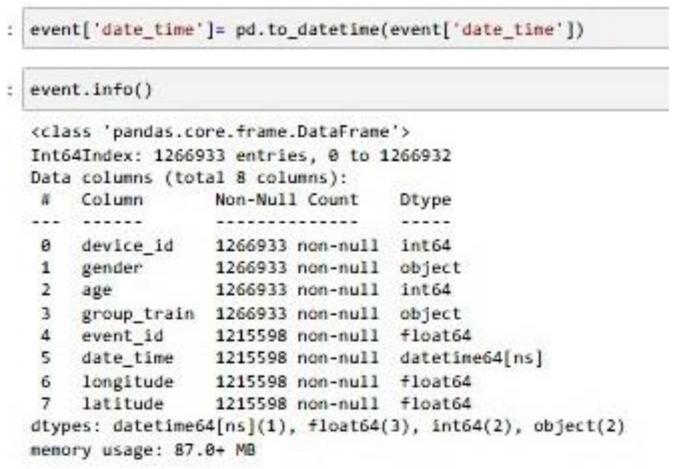
The presence of duplicates was checked and is dropped and the final data shape is as given below



```
: #dropping duplicates
  non_event.drop_duplicates(inplace=True)

: #dropping duplicates
  event.drop_duplicates(inplace=True)

: #dropping duplicates
  app.drop_duplicates(inplace=True)

: #Print the number of rows dropped
  print("Duplicate rows dropped in non_event = ",non_event_rows - non_event.shape[0])
  print("Duplicate rows dropped in event = ",event_rows - event.shape[0])
  print("Duplicate rows dropped in app = ",app_rows - app.shape[0])

  Duplicate rows dropped in non_event =  194
  Duplicate rows dropped in event =  0
  Duplicate rows dropped in app =  1281276

: #Print the size (rows and columns) in all the data frames
  print("non_event = ",non_event.shape," Rows = ",non_event.shape[0]," Columns = ",non_event.shape[1])
  print("event = ",event.shape," Rows = ",event.shape[0]," Columns = ",event.shape[1])
  print("app = ",app.shape," Rows = ",app.shape[0]," Columns = ",app.shape[1])

  non_event =  (74646, 6)  Rows =  74646  Columns =  6
  event =  (1266933, 8)  Rows =  1266933  Columns =  8
  app =  (208074434, 6)  Rows =  208074434  Columns =  6
```

3. Changing datatypes

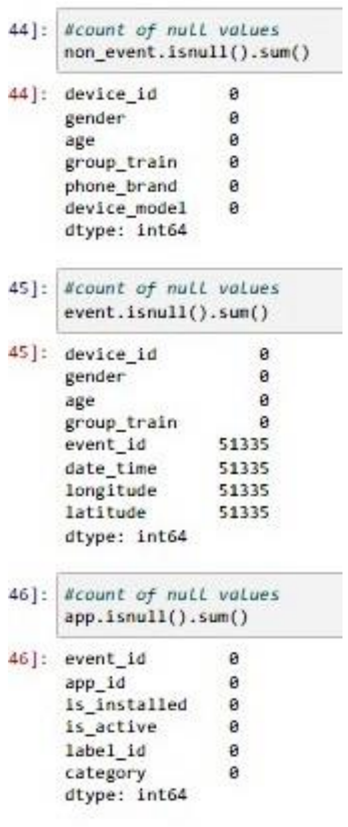
   From the above explorations, it can seen that date_type in events table need to be converted to data_time data type



```
: event['date_time']= pd.to_datetime(event['date_time'])

: event.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 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   date_time    1215598 non-null  datetime64[ns]
 6   longitude    1215598 non-null  float64
 7   latitude     1215598 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(2), object(2)
memory usage: 87.0+ MB
```

4. Missing value treatment

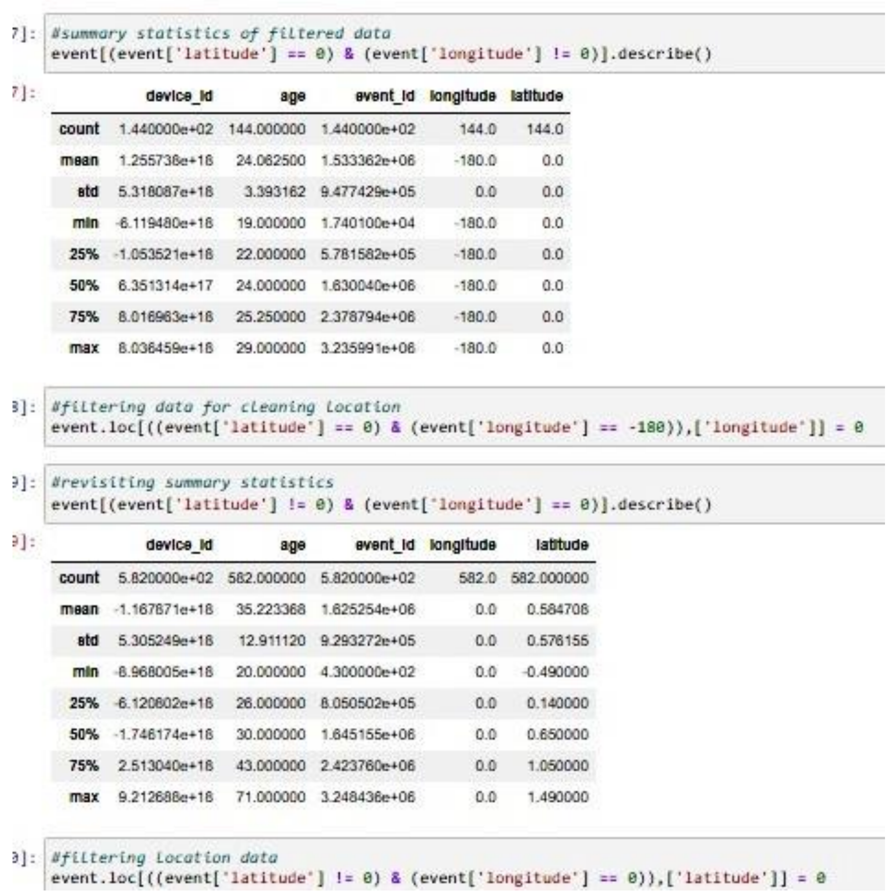
   The check for null values shows the following results:



```
44]: #count of null values
     non_event.isnull().sum()

44]: device_id       0
     gender          0
     age             0
     group_train     0
     phone_brand     0
     device_model    0
     dtype: int64

45]: #count of null values
     event.isnull().sum()

45]: device_id          0
     gender             0
     age                0
     group_train        0
     event_id       51335
     date_time      51335
     longitude      51335
     latitude       51335
     dtype: int64

46]: #count of null values
     app.isnull().sum()

46]: event_id        0
     app_id          0
     is_installed    0
     is_active       0
     label_id        0
     category        0
     dtype: int64
```

It can be seen that events datasets have missing values. But it cannot be considered as the usual missing value entity. The events are generated when an app is used and it triggers an event. If the users have restricted permission to gain information over the app, there won't be any event data available. Imputation is not the ideal way to deal here. So it is going to be let be as it is and will be filtered out accordingly, when considering data with app permissions and not.

It can also be seen that there are there are values of latitude and longitude as 0, -1, 1,180 and -180. These values are illogical in this project's context and so are treated as missing data. They are replaced with all 0 values.



```
#summary statistics of filtered data
event[(event['latitude'] == 0) & (event['longitude'] != 0)].describe()
```

	device_id	age	event_id	longitude	latitude
count	1.440000e+02	144.000000	1.440000e+02	144.0	144.0
mean	1.255736e+18	24.062500	1.533362e+06	-180.0	0.0
std	5.318087e+18	3.393162	9.477429e+05	0.0	0.0
min	-6.119480e+18	19.000000	1.740100e+04	-180.0	0.0
25%	-1.053521e+18	22.000000	5.781582e+05	-180.0	0.0
50%	6.351314e+17	24.000000	1.630040e+06	-180.0	0.0
75%	8.016963e+18	25.250000	2.378794e+06	-180.0	0.0
max	8.036459e+18	29.000000	3.235991e+06	-180.0	0.0

```
#filtering data for cleaning location
event.loc[((event['latitude'] == 0) & (event['longitude'] == -180)),['longitude']] = 0
```

```
#revisiting summary statistics
event[(event['latitude'] != 0) & (event['longitude'] == 0)].describe()
```

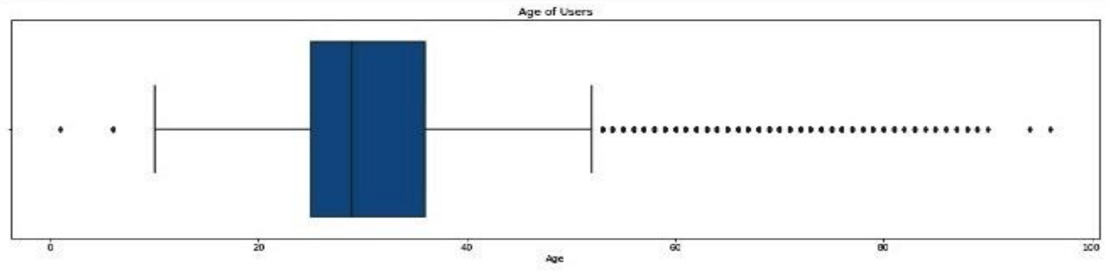
	device_id	age	event_id	longitude	latitude
count	5.820000e+02	582.000000	5.820000e+02	582.0	582.000000
mean	-1.167671e+18	35.223368	1.625254e+06	0.0	0.584708
std	5.305249e+18	12.911120	9.293272e+05	0.0	0.576155
min	-8.968005e+18	20.000000	4.300000e+02	0.0	-0.490000
25%	-6.120602e+18	26.000000	8.050502e+05	0.0	0.140000
50%	-1.746174e+18	30.000000	1.645155e+06	0.0	0.650000
75%	2.513040e+18	43.000000	2.423760e+06	0.0	1.050000
max	9.212688e+18	71.000000	3.248436e+06	0.0	1.490000

```
#filtering location data
event.loc[((event['latitude'] != 0) & (event['longitude'] == 0)),['latitude']] = 0
```

5. Outlier treatment

   Age variable in both non_events and events data have outliers. It is properly dealt with.

```
#box plot for age
fig = plt.figure(figsize=(20, 5))
sns.boxplot(data = non_event, x = non_event['age'])
plt.xlabel("Age")
plt.title("Age of Users")
plt.show()
```



The above boxplots clearly shows the presence of outliers.

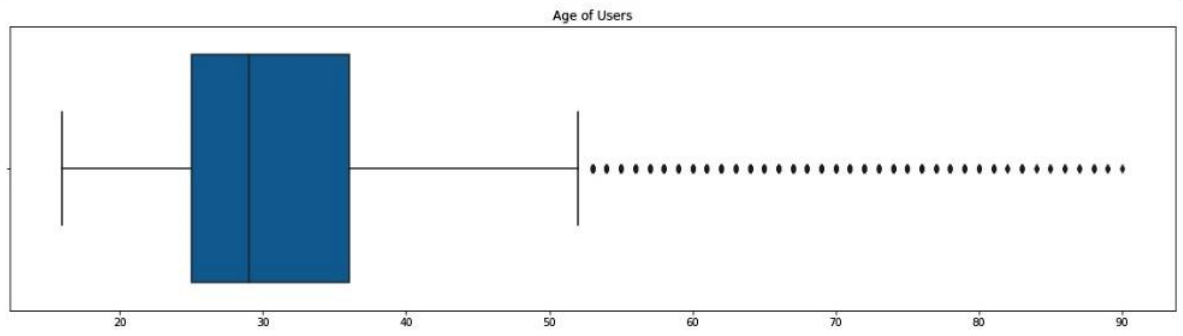
```
#dropping rows with outliers
non_event=non_event[~(non_event['age']>90)]
```

```
non_event=non_event[~(non_event['age']<16)]
```

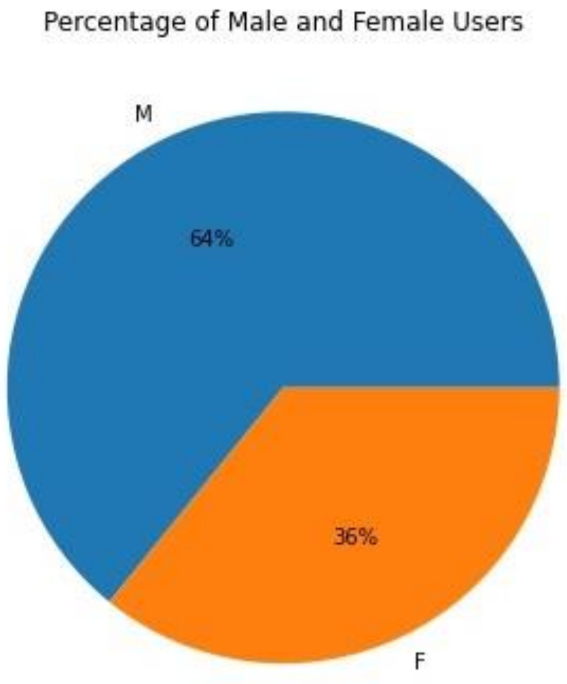
3. Exploratory Data Analysis

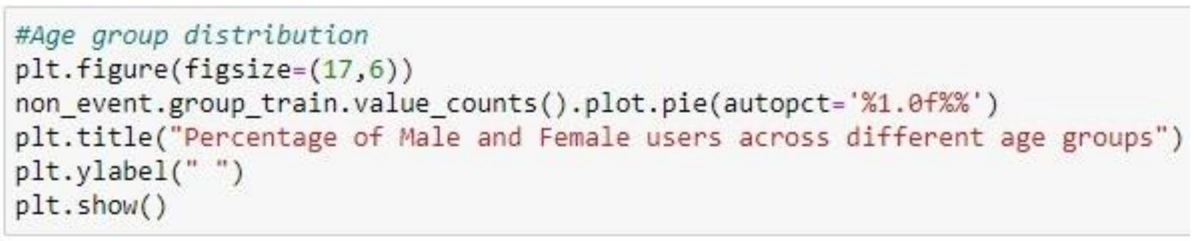
3.1. Plot appropriate graphs representing the distribution of age and gender in the data set

```
#Replotting age variable
fig = plt.figure(figsize=(20, 5))
sns.boxplot(data = non_event, x = non_event['age'])
plt.xlabel("Age")
plt.title("Age of Users")
plt.show()
```

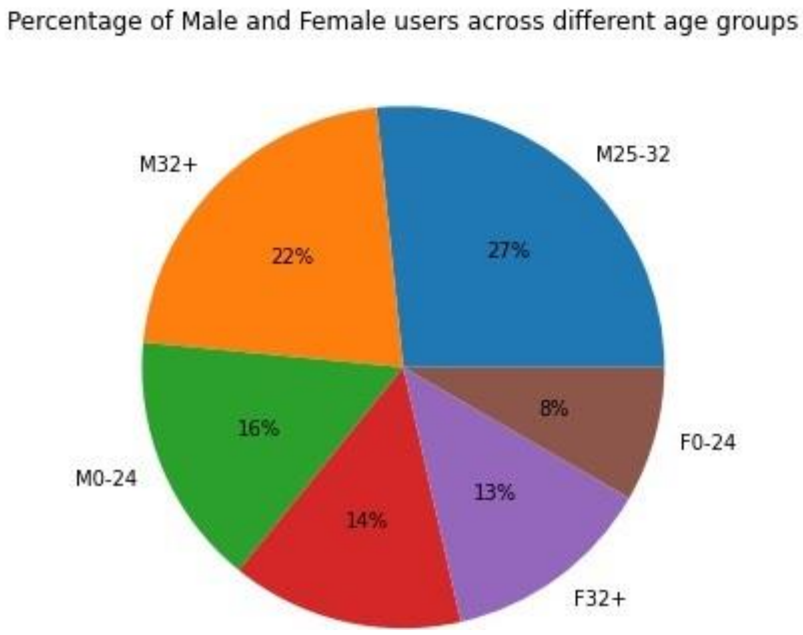


```
#gender distribution
plt.figure(figsize=(17,6))
non_event.gender.value_counts().plot.pie(autopct='%1.0f%%')
plt.title("Percentage of Male and Female Users")
plt.ylabel(" ")
plt.show()
```



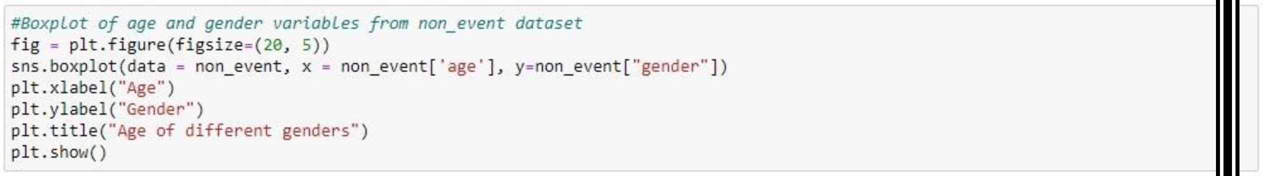


```
#Age group distribution
plt.figure(figsize=(17,6))
non_event.group_train.value_counts().plot.pie(autopct='%1.0f%%')
plt.title("Percentage of Male and Female users across different age groups")
plt.ylabel(" ")
plt.show()
```

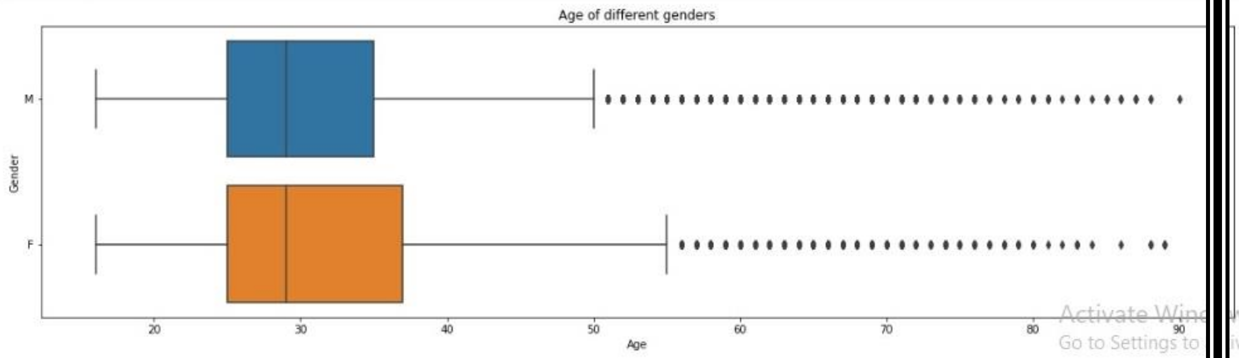


- The mean age of users is 29 years old
- Majority of the users are males, about 64%. Around 36% of users are females.
- Most of the male and female users belong to 25-33.
- The least number of users in both genders belong to 0-24 age group.

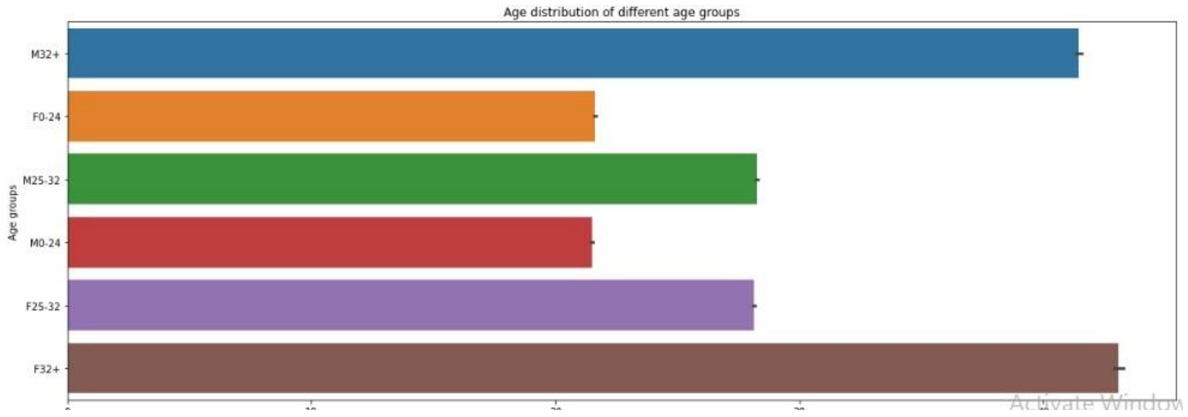
3.2 Boxplot analysis for gender and age



```
#Boxplot of age and gender variables from non_event dataset
fig = plt.figure(figsize=(20, 5))
sns.boxplot(data = non_event, x = non_event['age'], y=non_event["gender"])
plt.xlabel("Age")
plt.ylabel("Gender")
plt.title("Age of different genders")
plt.show()
```

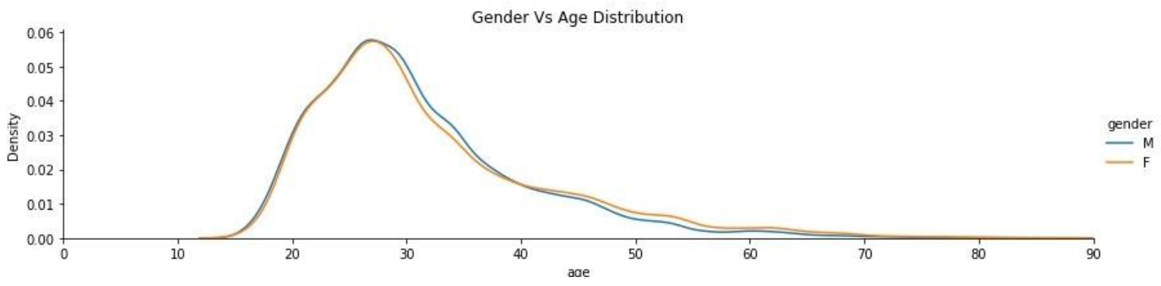


```
#Barplot of age group distribution
fig = plt.figure(figsize=(20,7))
sns.barplot(data = non_event, x = non_event['age'], y=non_event["group_train"])
plt.xlabel("Age")
plt.ylabel("Age groups")
plt.title("Age distribution of different age groups")
plt.show()
```

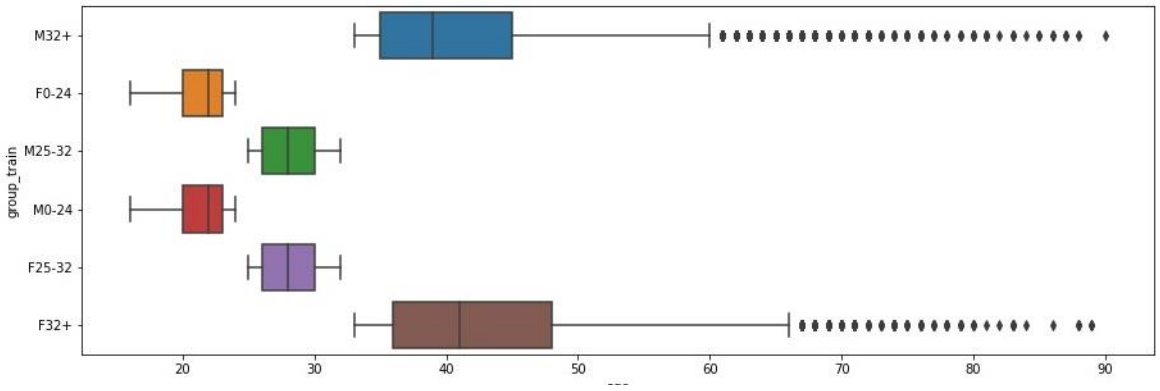


```
plt.figure(figsize=(15,5))
facet = sns.FacetGrid(non_event, hue="gender",aspect=4, hue_order=['M', 'F'])
facet.map(sns.kdeplot,'age')
facet.set(xlim=(0, non_event['age'].max()))
facet.add_legend()
plt.title("Gender Vs Age Distribution")
plt.show()
```

```
<Figure size 1080x360 with 0 Axes>
```

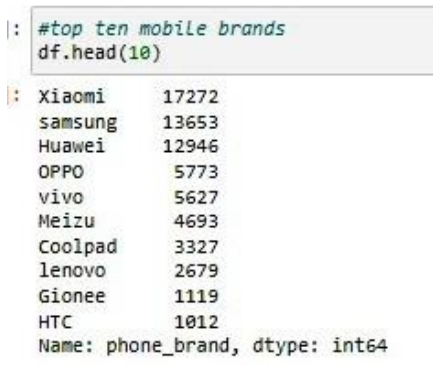


```
plt.figure(figsize=(15,5))
sns.boxplot(data = non_event, x = non_event['age'],y=non_event['group_train'])
plt.show()
```



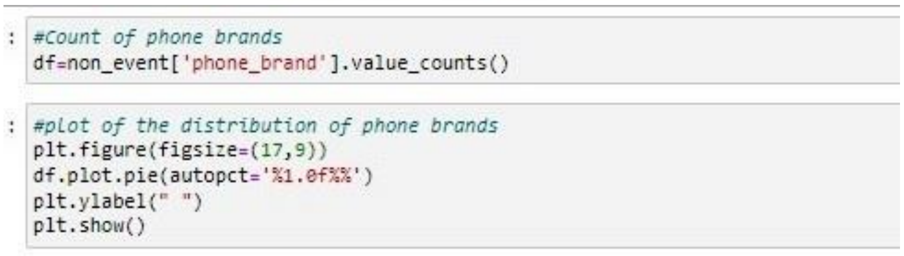
- The mean age of both genders are same, And both lie at 29 years old.
- The oldest (age wise) users are females.
- The age of genders follow a normal distribution.
- Other than the 32+ age group, all age groups across genders have same mean age.
- 32+ age group males have a mean age of 38 and females have a mean age of 41 years old.

3.3 Distribution of phone brands



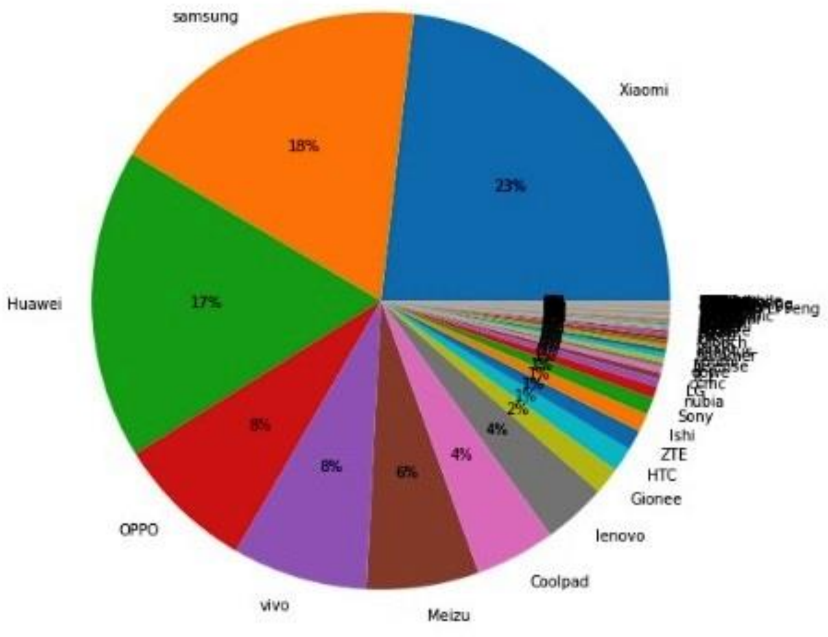
```
]: #top ten mobile brands
   df.head(10)
```

```
]: Xiaomi     17272
   samsung    13653
   Huawei     12946
   OPPO        5773
   vivo        5627
   Meizu       4693
   Coolpad     3327
   lenovo      2679
   Gionee      1119
   HTC         1012
   Name: phone_brand, dtype: int64
```



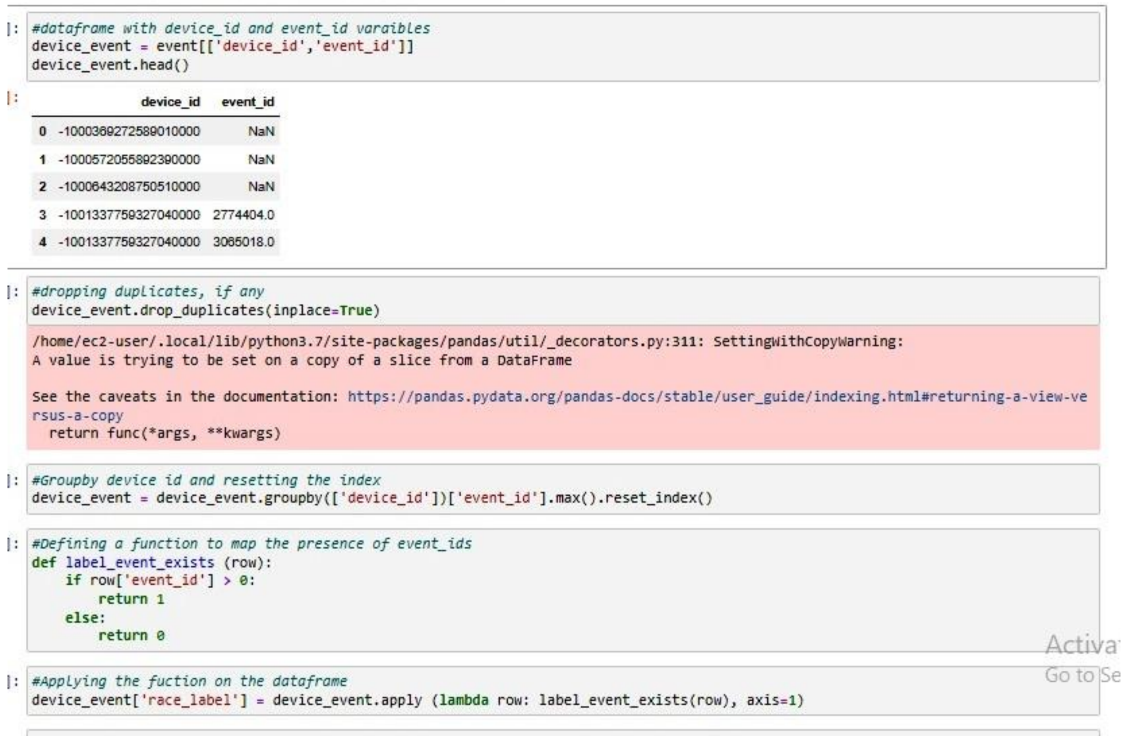
```
: #Count of phone brands
  df=non_event['phone_brand'].value_counts()
```

```
: #plot of the distribution of phone brands
  plt.figure(figsize=(17,9))
  df.plot.pie(autopct='%1.0f%%')
  plt.ylabel(" ")
  plt.show()
```



- Most of the users use the phone brand Xiaomi, followed by Samsung and Huawei

3.4 Plot the percentage of the device_ids with and without event data



```
]: #dataframe with device_id and event_id varaibles
   device_event = event[['device_id','event_id']]
   device_event.head()
```

]:

	device_id	event_id
0	-1000369272589010000	NaN
1	-1000572055892390000	NaN
2	-1000643208750510000	NaN
3	-1001337759327040000	2774404.0
4	-1001337759327040000	3065018.0

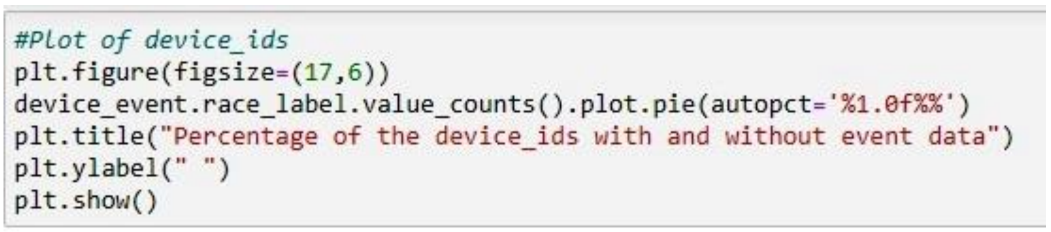
```
]: #dropping duplicates, if any
   device_event.drop_duplicates(inplace=True)
```

```
/home/ec2-user/.local/lib/python3.7/site-packages/pandas/util/_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  return func(*args, **kwargs)
```

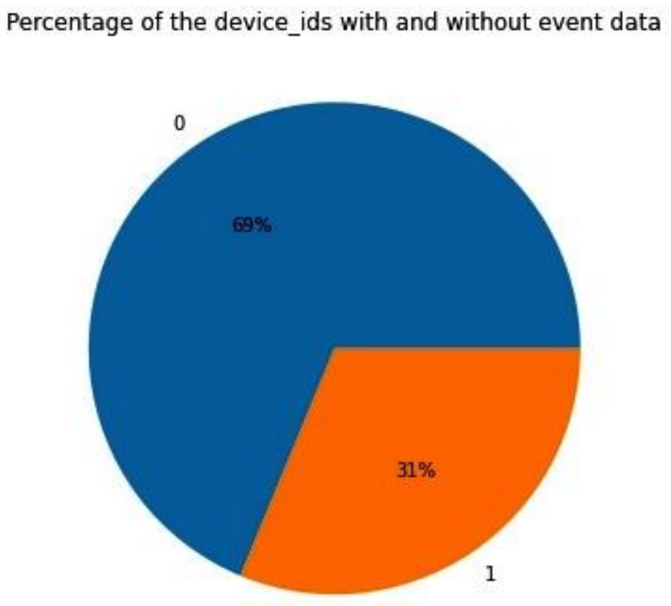
```
]: #Groupby device id and resetting the index
   device_event = device_event.groupby(['device_id'])['event_id'].max().reset_index()
```

```
]: #Defining a function to map the presence of event_ids
   def label_event_exists (row):
       if row['event_id'] > 0:
           return 1
       else:
           return 0
```

```
]: #Applying the fuction on the dataframe
   device_event['race_label'] = device_event.apply (lambda row: label_event_exists(row), axis=1)
```

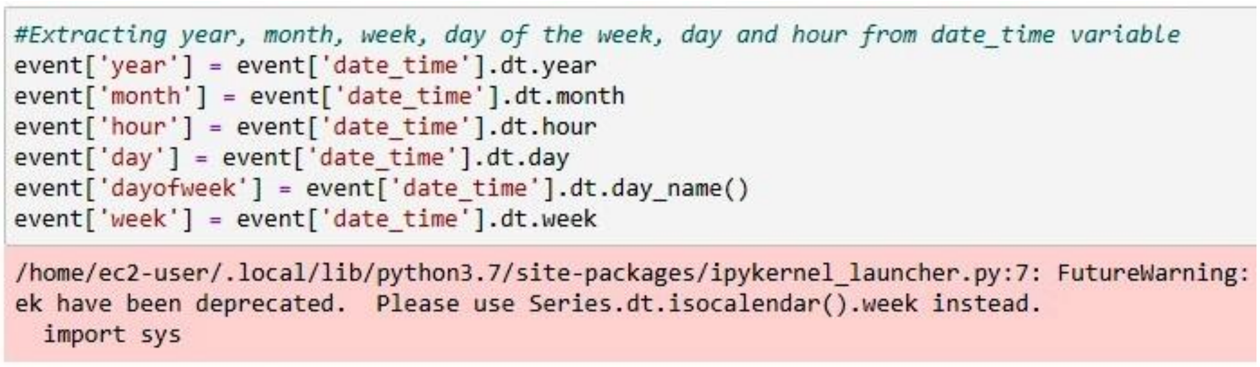


```
#Plot of device_ids
plt.figure(figsize=(17,6))
device_event.race_label.value_counts().plot.pie(autopct='%1.0f%%')
plt.title("Percentage of the device_ids with and without event data")
plt.ylabel(" ")
plt.show()
```



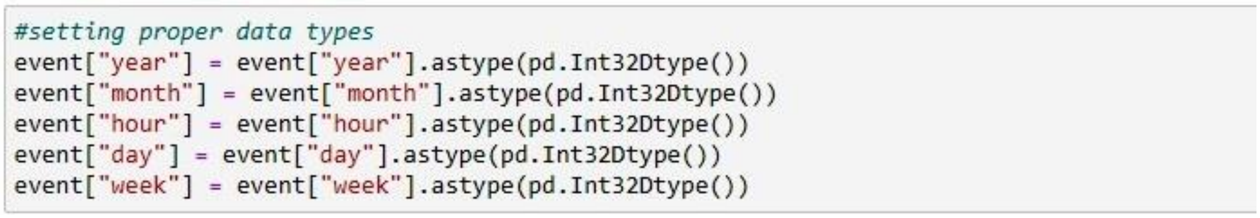
- Around 69% of devices have restricted permission to access info by apps

3.5 Plot a graph representing the distribution of events over different days of a week

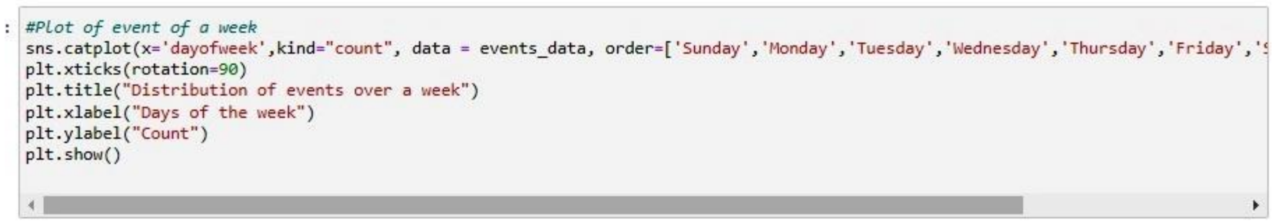


```
#Extracting year, month, week, day of the week, day and hour from date_time variable
event['year'] = event['date_time'].dt.year
event['month'] = event['date_time'].dt.month
event['hour'] = event['date_time'].dt.hour
event['day'] = event['date_time'].dt.day
event['dayofweek'] = event['date_time'].dt.day_name()
event['week'] = event['date_time'].dt.week
```

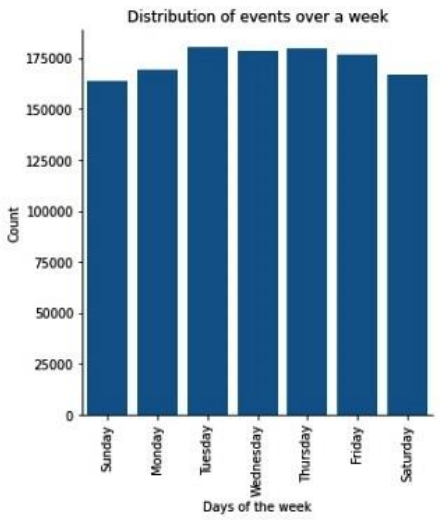
```
/home/ec2-user/.local/lib/python3.7/site-packages/ipykernel_launcher.py:7: FutureWarning:
ek have been deprecated.  Please use Series.dt.isocalendar().week instead.
  import sys
```



```
#setting proper data types
event["year"] = event["year"].astype(pd.Int32Dtype())
event["month"] = event["month"].astype(pd.Int32Dtype())
event["hour"] = event["hour"].astype(pd.Int32Dtype())
event["day"] = event["day"].astype(pd.Int32Dtype())
event["week"] = event["week"].astype(pd.Int32Dtype())
```

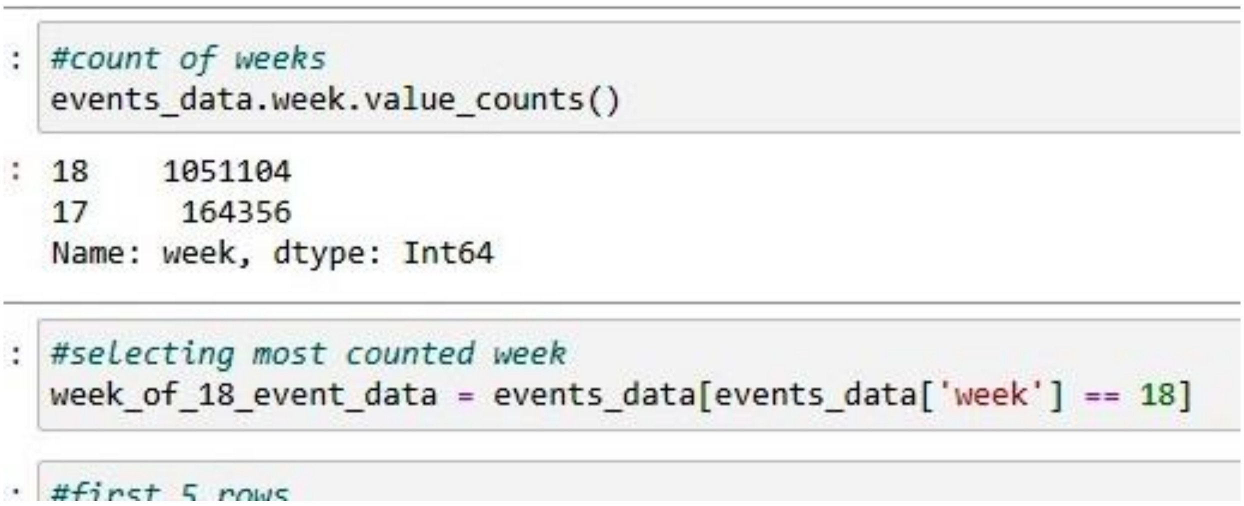


```
#Plot of event of a week
sns.catplot(x='dayofweek',kind="count", data = events_data, order=['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','S
plt.xticks(rotation=90)
plt.title("Distribution of events over a week")
plt.xlabel("Days of the week")
plt.ylabel("Count")
plt.show()
```



- Mid-week is where there is more number of events triggered.
- Most events are triggered on Tuesdays followed by Thursdays and Wednesdays

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

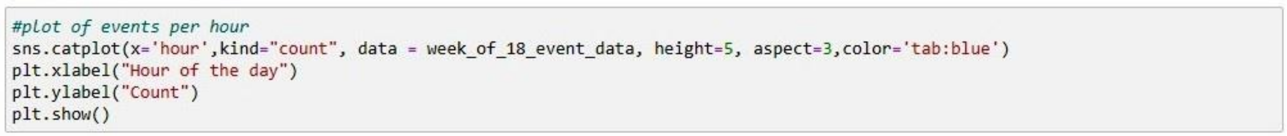


```
#count of weeks
events_data.week.value_counts()
```

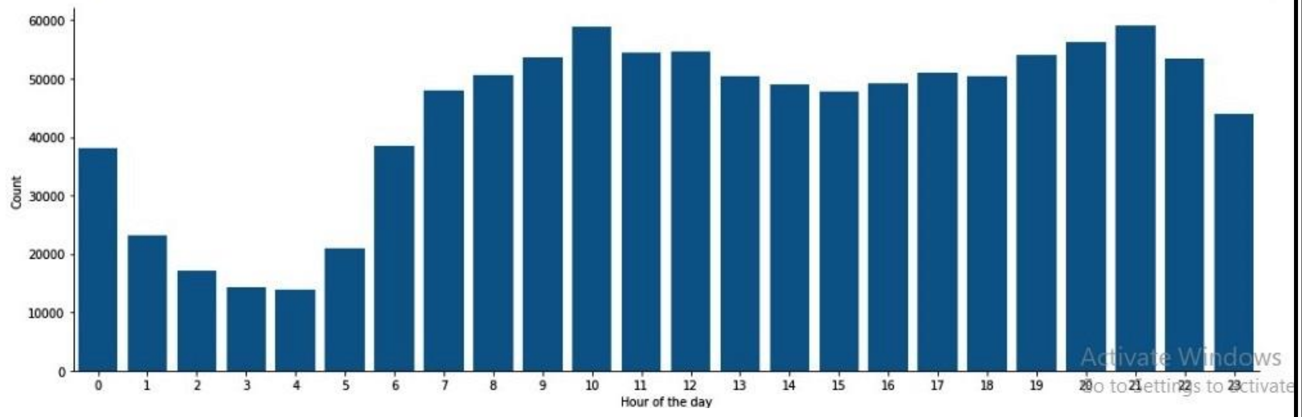
```
18    1051104
17     164356
Name: week, dtype: Int64
```

```
#selecting most counted week
week_of_18_event_data = events_data[events_data['week'] == 18]
```

```
#first 5 rows
```



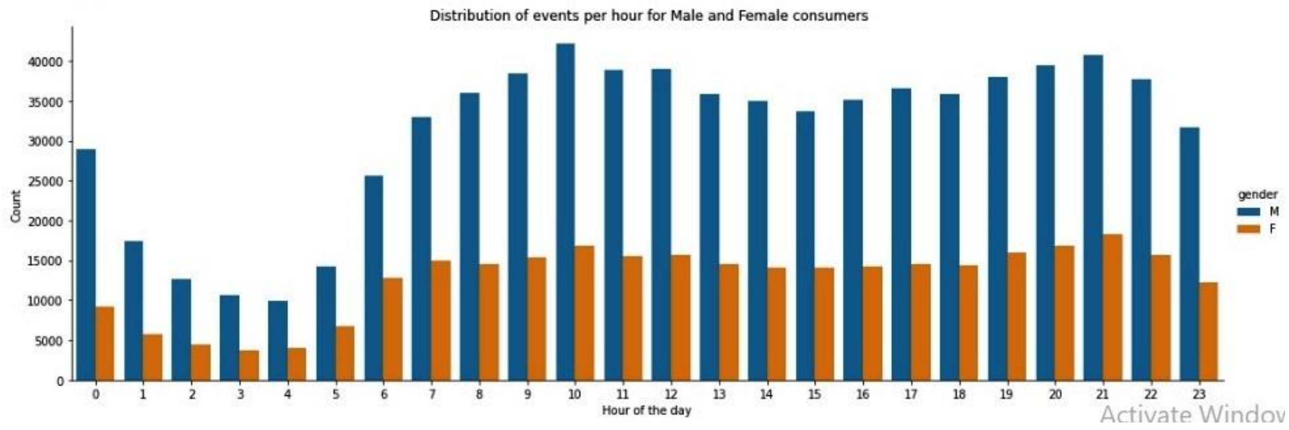
```
#plot of events per hour
sns.catplot(x='hour',kind="count", data = week_of_18_event_data, height=5, aspect=3,color='tab:blue')
plt.xlabel("Hour of the day")
plt.ylabel("Count")
plt.show()
```



- Throughout one week, most number of events is triggered during 10 in the morning and 10 at night, both around 60K in count
- Other than early hours of 1 am till 5 am, there are at least 38K events triggered.

3.7 **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.**]

```
#plot of events per hour for male and female consumers
sns.catplot(x='hour',kind="count",hue="gender", data = week_of_18_event_data, height=5, aspect=3)
plt.title(" Distribution of events per hour for Male and Female consumers")
plt.xlabel("Hour of the day")
plt.ylabel("Count")
plt.show()
```



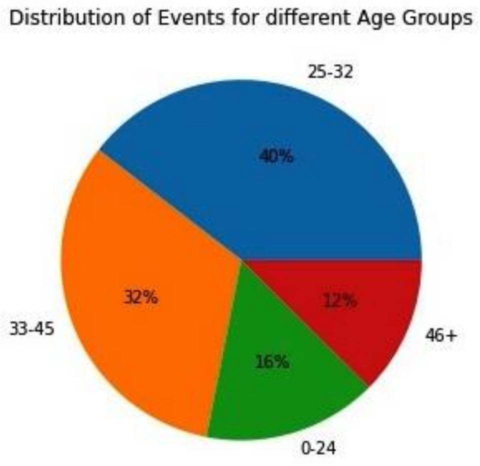
- Less number of events is triggered by female users, which in turn is the result of less number of female users.
- The phone usage pattern remains the same for both the genders, high number of events during 10 in the morning and 10 at night and lest number of events during early hours of 1 am till 5 am.

3.8 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+**]

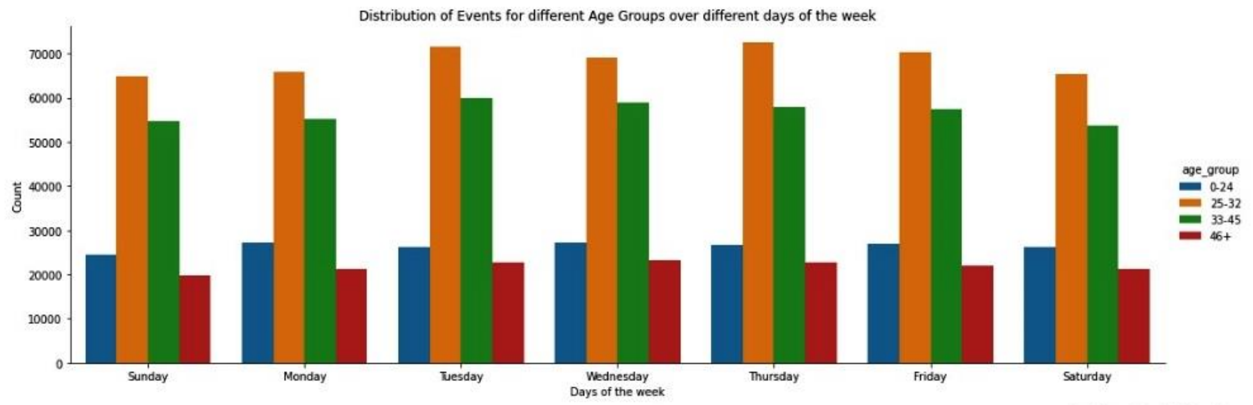
```
:  #Defining a function to make different age groups
   def age_group_generater (row):
       if row['age'] >= 0 and row['age'] <= 24:
           return '0-24'
       elif row['age'] >= 25 and row['age'] <= 32:
           return '25-32'
       elif row['age'] >= 33 and row['age'] <= 45:
           return '33-45'
       elif row['age'] >= 46:
           return '46+'
```

```
:  #applying the fuction
   event['age_group'] = event.apply (lambda row: age_group_generater(row), axis=1)
```

```
#Plot of percentage of different age group consumers
plt.figure(figsize=(15,5))
event.age_group.value_counts().plot.pie(autopct='%1.0f%%')
plt.title("Distribution of Events for different Age Groups")
plt.ylabel(" ")
plt.show()
```

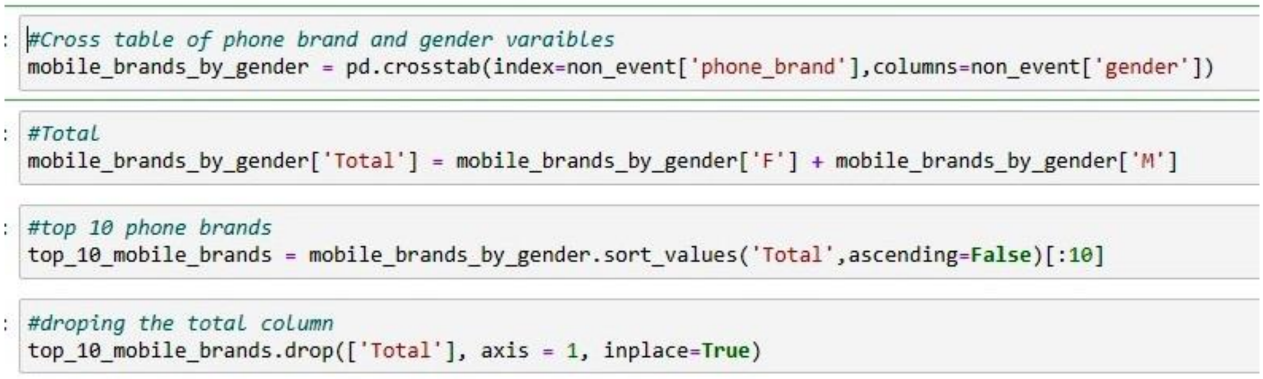


```
#plot of distibution of events across different age groups
sns.catplot(x='dayofweek',kind="count",hue='age_group',hue_order=['0-24', '25-32', '33-45','46+'], data = event, height=5, aspect
plt.title("Distribution of Events for different Age Groups over different days of the week")
plt.xlabel("Days of the week")
plt.ylabel("Count")
plt.show()
```



- There is significant difference in the distribution of events among age groups, across the week
- Most number of events is triggered by age group 25-32, followed by age group 33-45
- There is a huge difference of about 50K number of events triggered by age groups of 25+ and age group less than 25.
- The age group 0-24 exhibits more or less a steady pattern throughout the week and so is the 46+ age group
- The 25-32 age group shows the highest number of events across the week with a steady pattern followed by 33-45 age group

3.9 Stacked bar chart for the top 10 mobile brands across male and female consumers

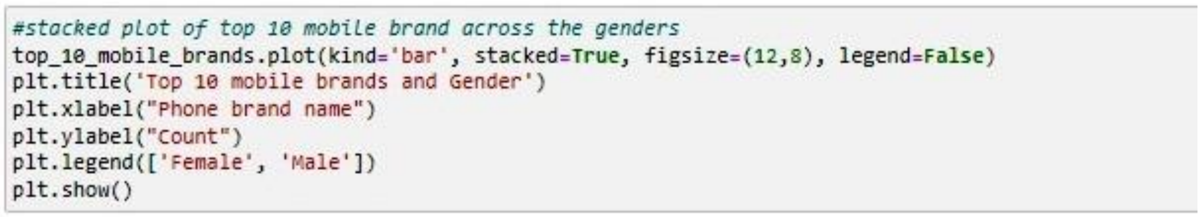


```
#Cross table of phone brand and gender varaibles
mobile_brands_by_gender = pd.crosstab(index=non_event['phone_brand'],columns=non_event['gender'])
```

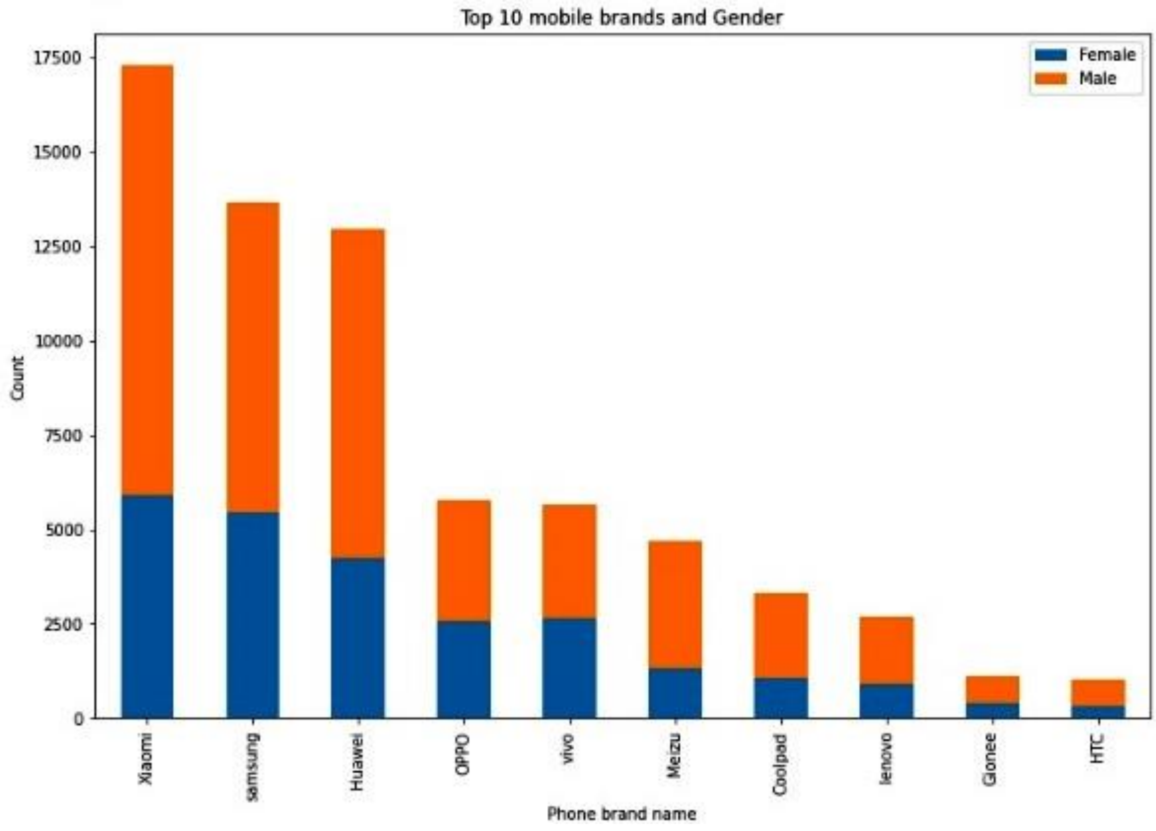
```
#Total
mobile_brands_by_gender['Total'] = mobile_brands_by_gender['F'] + mobile_brands_by_gender['M']
```

```
#top 10 phone brands
top_10_mobile_brands = mobile_brands_by_gender.sort_values('Total',ascending=False)[:10]
```

```
#droping the total column
top_10_mobile_brands.drop(['Total'], axis = 1, inplace=True)
```

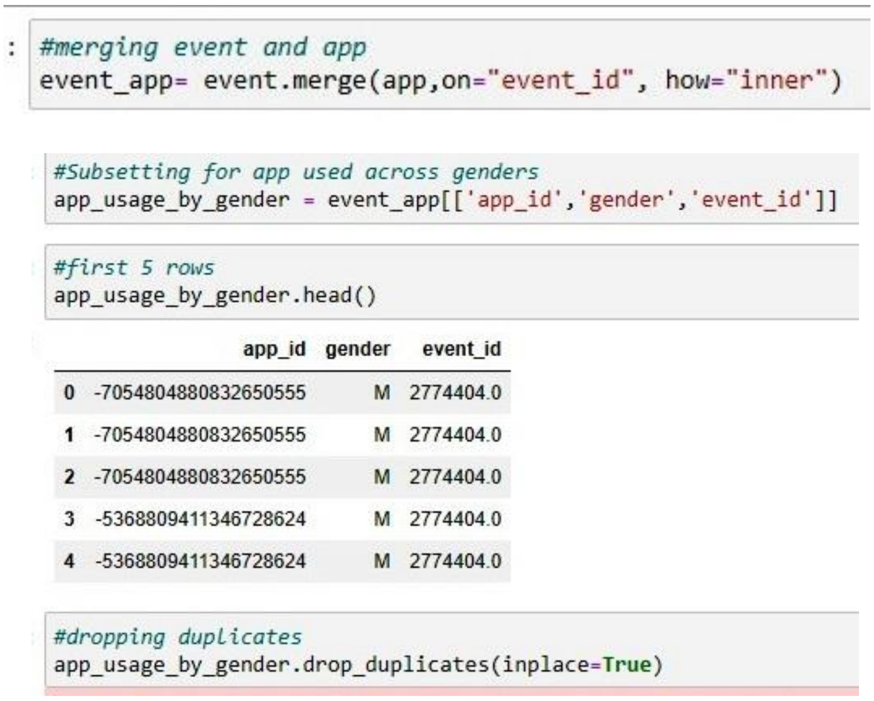


```
#stacked plot of top 10 mobile brand across the genders
top_10_mobile_brands.plot(kind='bar', stacked=True, figsize=(12,8), legend=False)
plt.title('Top 10 mobile brands and Gender')
plt.xlabel("Phone brand name")
plt.ylabel("Count")
plt.legend(['Female', 'Male'])
plt.show()
```



- The stacked bar chart is in agreement with the previously given pie chart of the phone brands, with Xiaomi being the top brand among the male and female users, followed by Samsung and Huawei

3.10 A chart representing the ten frequently used applications and their respective male and female percentage



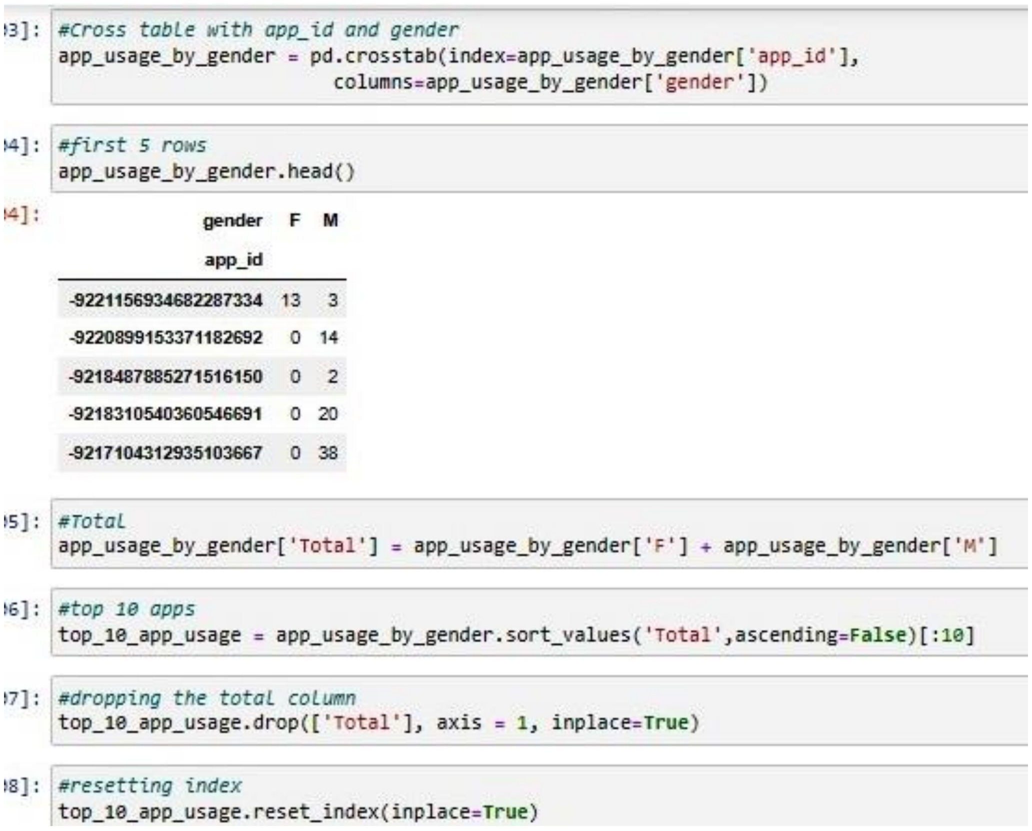
```
:  #merging event and app
   event_app= event.merge(app,on="event_id", how="inner")
```

```
#Subsetting for app used across genders
app_usage_by_gender = event_app[['app_id','gender','event_id']]
```

```
#first 5 rows
app_usage_by_gender.head()
```

	app_id	gender	event_id
0	-7054804880832650555	M	2774404.0
1	-7054804880832650555	M	2774404.0
2	-7054804880832650555	M	2774404.0
3	-5368809411346728624	M	2774404.0
4	-5368809411346728624	M	2774404.0

```
#dropping duplicates
app_usage_by_gender.drop_duplicates(inplace=True)
```



```
3]: #Cross table with app_id and gender
    app_usage_by_gender = pd.crosstab(index=app_usage_by_gender['app_id'],
                                      columns=app_usage_by_gender['gender'])
```

```
4]: #first 5 rows
    app_usage_by_gender.head()
```

4]:

gender	F	M
app_id		
-9221156934682287334	13	3
-9220899153371182692	0	14
-9218487885271516150	0	2
-9218310540360546691	0	20
-9217104312935103667	0	38

```
5]: #Total
    app_usage_by_gender['Total'] = app_usage_by_gender['F'] + app_usage_by_gender['M']
```

```
6]: #top 10 apps
    top_10_app_usage = app_usage_by_gender.sort_values('Total',ascending=False)[:10]
```

```
7]: #dropping the total column
    top_10_app_usage.drop(['Total'], axis = 1, inplace=True)
```

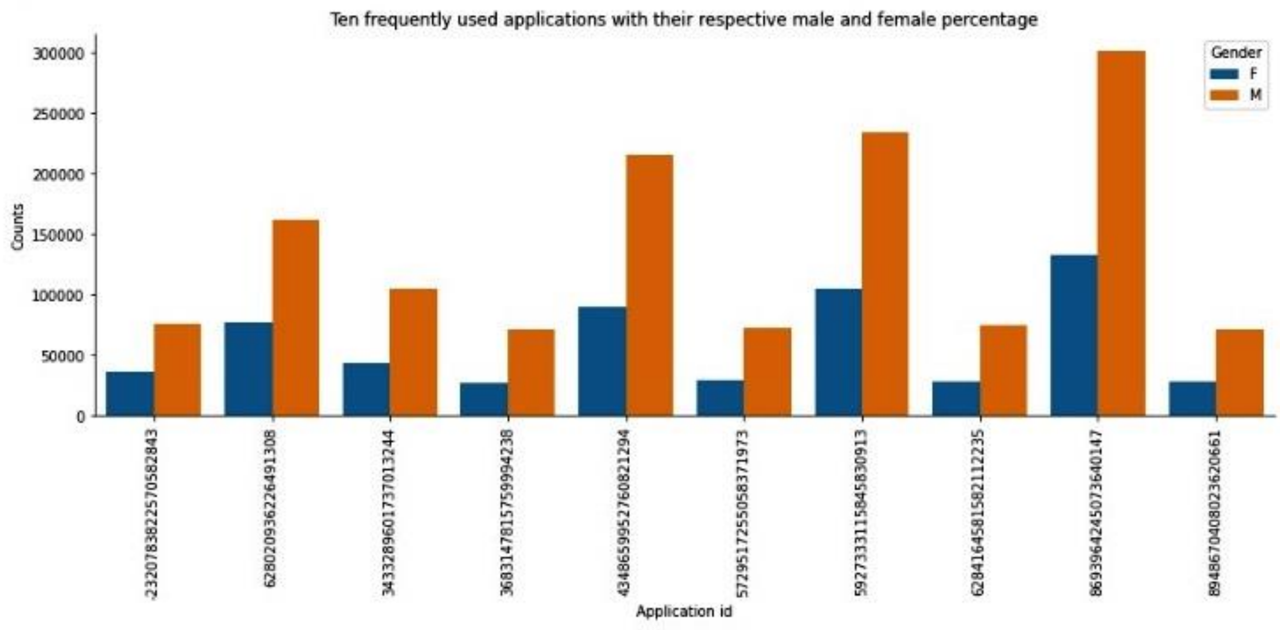
```
8]: #resetting index
    top_10_app_usage.reset_index(inplace=True)
```

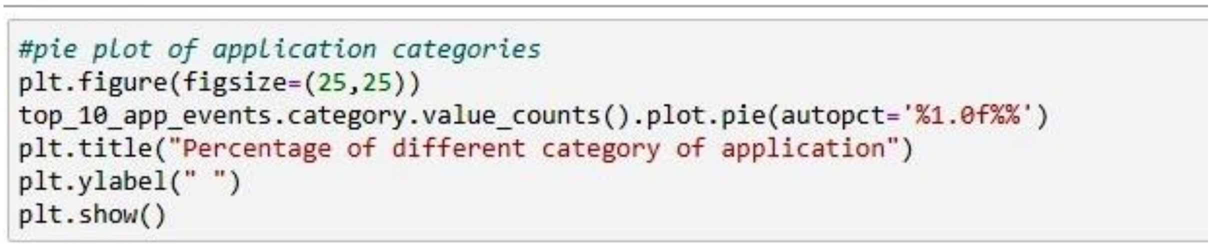
```
#Rearranging
top_10_app_usage_tidy = top_10_app_usage.melt(id_vars='app_id').rename(columns=str.title)
```

```
#dataframe
top_10_app_usage_tidy
```

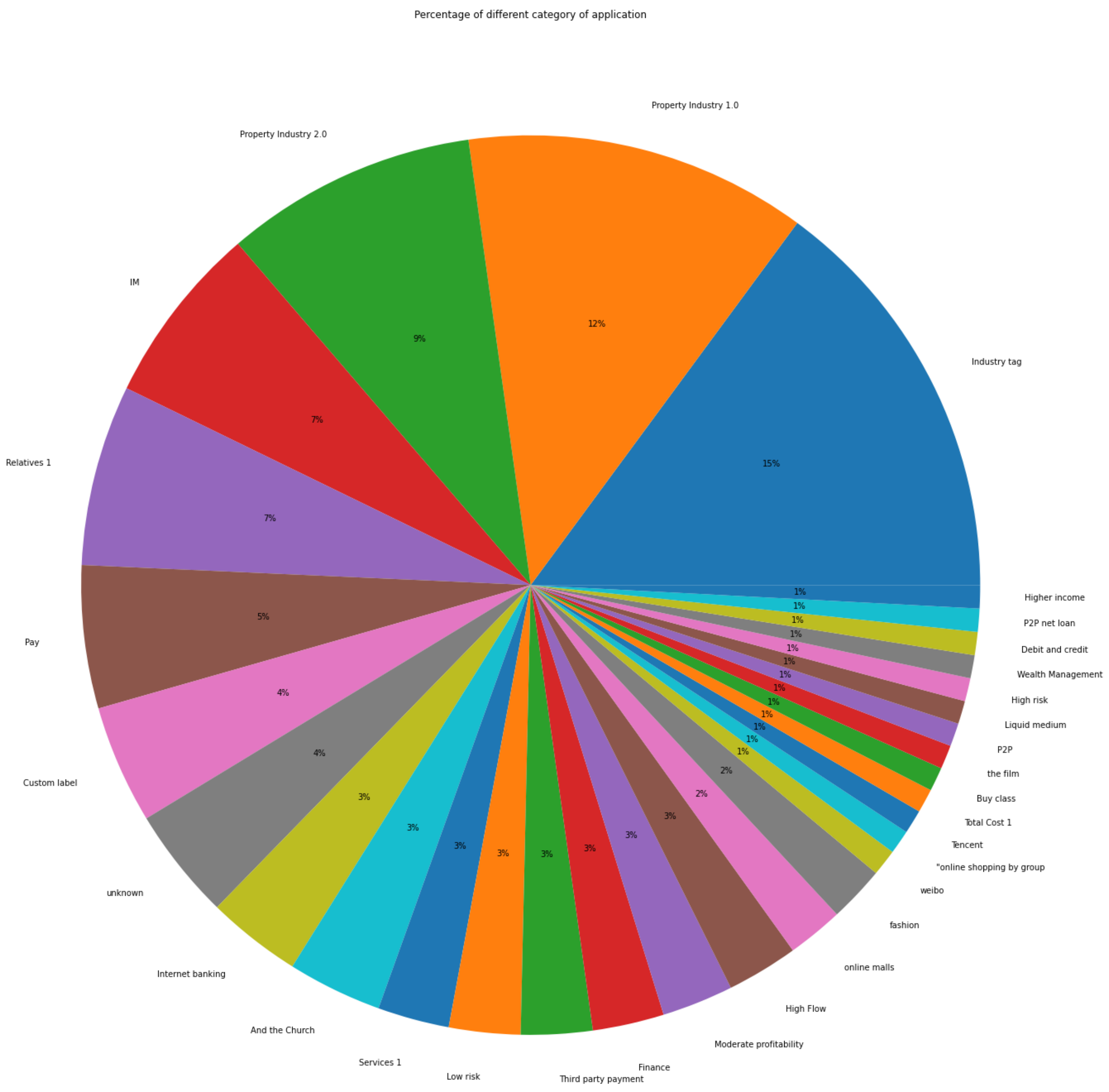
	App_Id	Gender	Value
0	8693964245073640147	F	132424
1	5927333115845830913	F	105256
2	4348659952760821294	F	89215
3	628020936226491308	F	77113
4	3433289601737013244	F	42763
5	-2320783822570582843	F	36471
6	6284164581582112235	F	28718
7	5729517255058371973	F	29903
8	8948670408023620661	F	28540
9	3683147815759994238	F	26965
10	8693964245073640147	M	300568
11	5927333115845830913	M	233327
12	4348659952760821294	M	215489

```
#plot of 10 frequently used app across genders
fig, ax1 = plt.subplots(figsize=(15, 5))
sns.barplot(x='App_Id', y='Value', hue='Gender', data=top_10_app_usage_tidy, ax=ax1)
plt.title("Ten frequently used applications with their respective male and female percentage")
plt.xlabel("Application id")
plt.ylabel("Counts")
plt.xticks(rotation=90, ha='center')
sns.despine(fig)
```



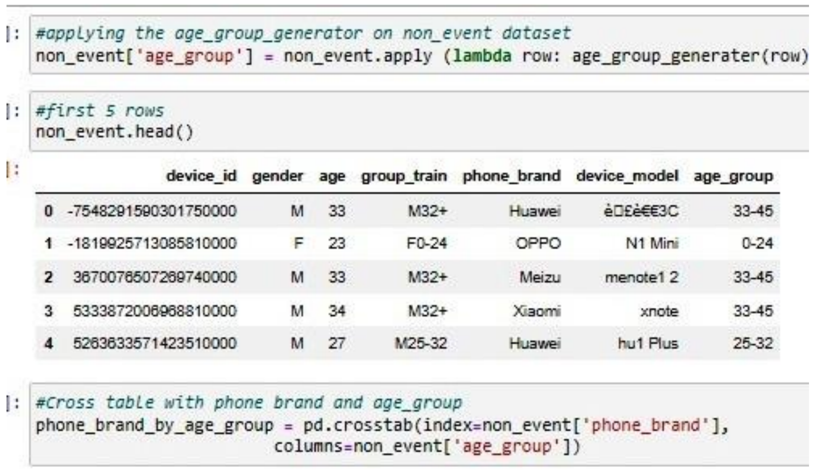


```
#pie plot of application categories
plt.figure(figsize=(25,25))
top_10_app_events.category.value_counts().plot.pie(autopct='%1.0f%%')
plt.title("Percentage of different category of application")
plt.ylabel(" ")
plt.show()
```



- The above shown app categories are the most used apps. Industry and property apps being most used followed by finances, internet banking and games
- Male users are more which an expected outcome.

3.11 List the top 10 mobile phone brands bought by customers by age groups



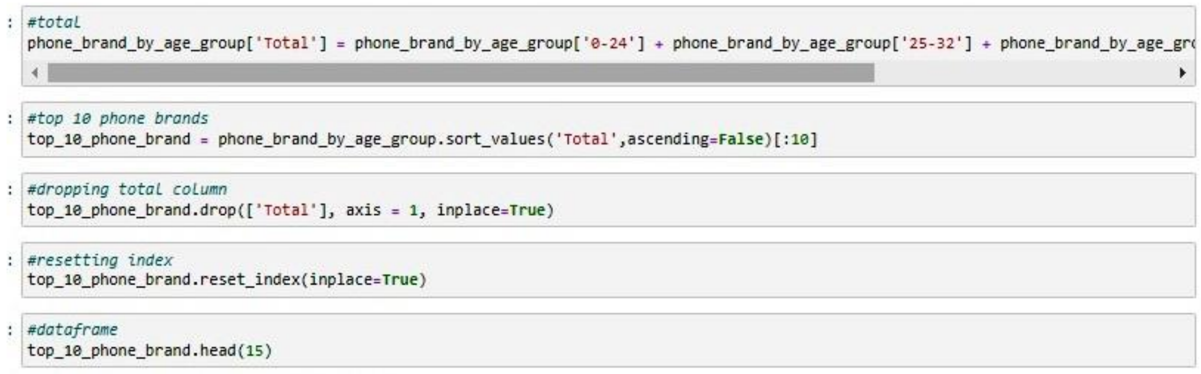
```
]: #applying the age_group_generator on non_event dataset
   non_event['age_group'] = non_event.apply (lambda row: age_group_generater(row)
```

```
]: #first 5 rows
   non_event.head()
```

]:

	device_id	gender	age	group_train	phone_brand	device_model	age_group
0	-7548291590301750000	M	33	M32+	Huawei	è□£è€€3C	33-45
1	-1819925713085810000	F	23	F0-24	OPPO	N1 Mini	0-24
2	3670076507269740000	M	33	M32+	Meizu	menote1 2	33-45
3	5333872006968810000	M	34	M32+	Xiaomi	xnote	33-45
4	5263633571423510000	M	27	M25-32	Huawei	hu1 Plus	25-32

```
]: #Cross table with phone brand and age_group
   phone_brand_by_age_group = pd.crosstab(index=non_event['phone_brand'],
                                          columns=non_event['age_group'])
```



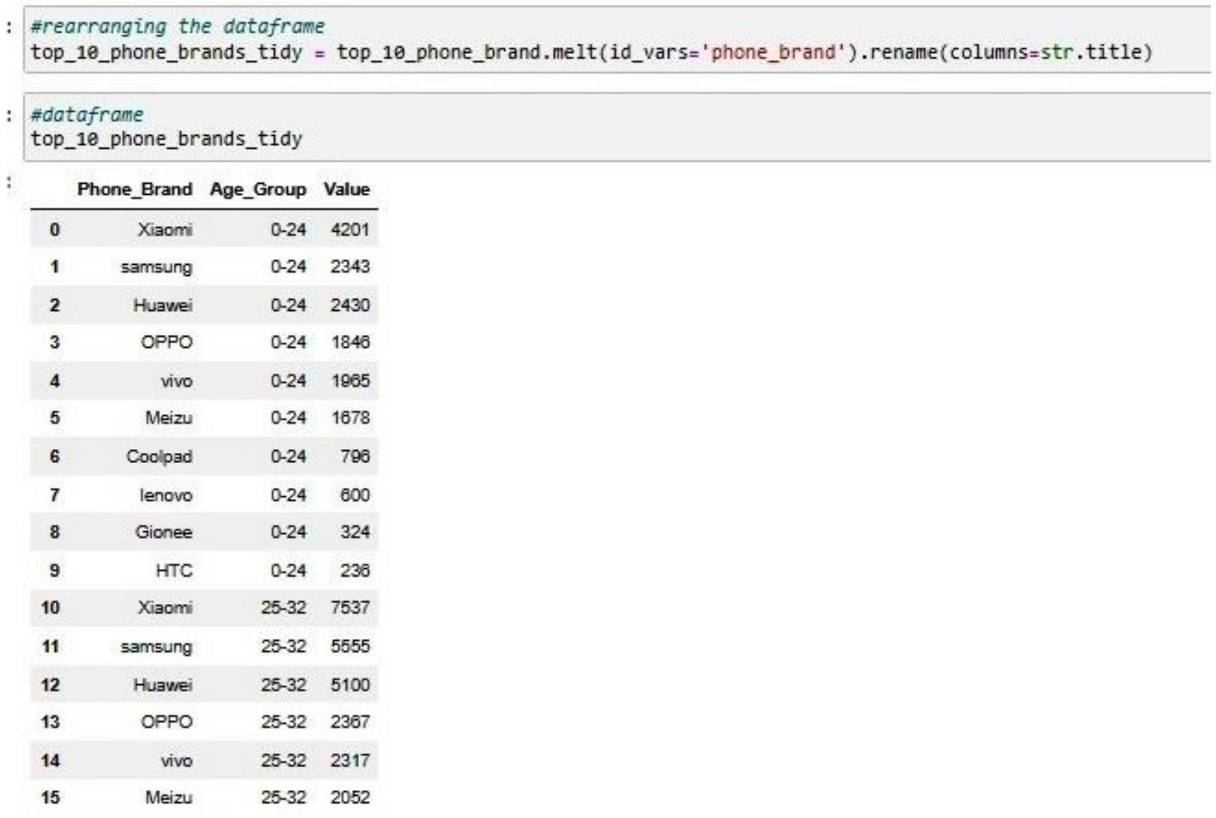
```
: #total
  phone_brand_by_age_group['Total'] = phone_brand_by_age_group['0-24'] + phone_brand_by_age_group['25-32'] + phone_brand_by_age_gr
```

```
: #top 10 phone brands
  top_10_phone_brand = phone_brand_by_age_group.sort_values('Total',ascending=False)[:10]
```

```
: #dropping total column
  top_10_phone_brand.drop(['Total'], axis = 1, inplace=True)
```

```
: #resetting index
  top_10_phone_brand.reset_index(inplace=True)
```

```
: #dataframe
  top_10_phone_brand.head(15)
```



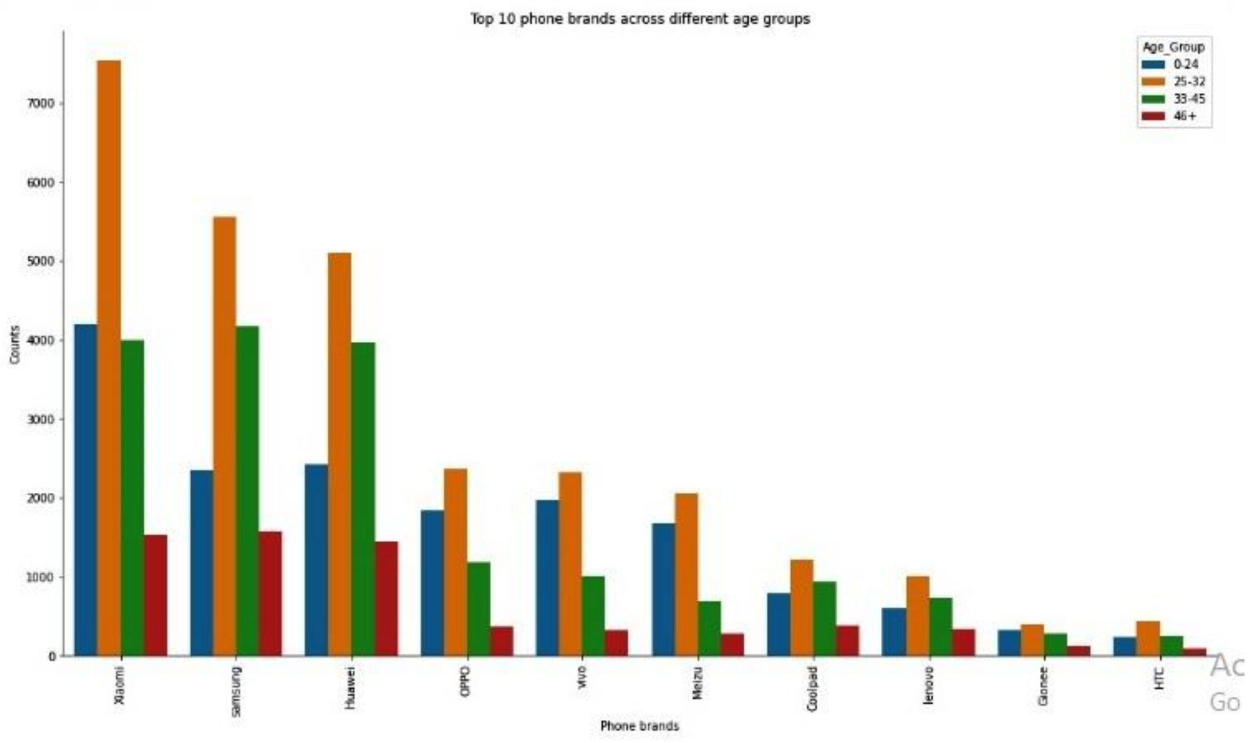
```
: #rearranging the dataframe
  top_10_phone_brands_tidy = top_10_phone_brand.melt(id_vars='phone_brand').rename(columns=str.title)
```

```
: #dataframe
  top_10_phone_brands_tidy
```

:

	Phone_Brand	Age_Group	Value
0	Xiaomi	0-24	4201
1	samsung	0-24	2343
2	Huawei	0-24	2430
3	OPPO	0-24	1846
4	vivo	0-24	1965
5	Meizu	0-24	1678
6	Coolpad	0-24	796
7	lenovo	0-24	600
8	Gionee	0-24	324
9	HTC	0-24	236
10	Xiaomi	25-32	7537
11	samsung	25-32	5555
12	Huawei	25-32	5100
13	OPPO	25-32	2367
14	vivo	25-32	2317
15	Meizu	25-32	2052

```
#plot of top 10 phone brands across different age groups
fig, ax1 = plt.subplots(figsize=(18, 10))
sns.barplot(x='Phone_Brand', y='Value', hue='Age_Group', data=top_10_phone_brands_tidy, ax=ax1)
plt.title("Top 10 phone brands across different age groups")
plt.xlabel("Phone brands")
plt.ylabel("Counts")
plt.xticks(rotation=90, ha='center')
sns.despine(fig)
```



```
top_10_phone_brand_for_0_24 = phone_brand_by_age_group.sort_values('0-24',ascending=False)[:10]
top_10_phone_brand_for_0_24.head(10)
```

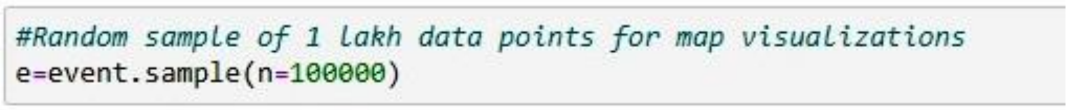
age_group	0-24	25-32	33-45	46+
phone_brand				
Xiaomi	4201	7537	4000	1534
Huawei	2430	5100	3964	1452
samsung	2343	5555	4175	1580
vivo	1965	2317	1013	332
OPPO	1846	2367	1183	377
Meizu	1678	2052	687	276
Coolpad	796	1208	942	381
lenovo	600	1005	738	336
Gionee	324	400	278	117
HTC	236	436	251	89

- Pricing plays a key role and we can see that mobile brands with competitive process are being purchased more.
- In all the brands, age group of 25-32 bought more phones, but the distribution of 0-24 and 33-45 is varied based on prices.
- In general observation, 0-24 age-group people got more phones from the company whose prices are comparatively less.

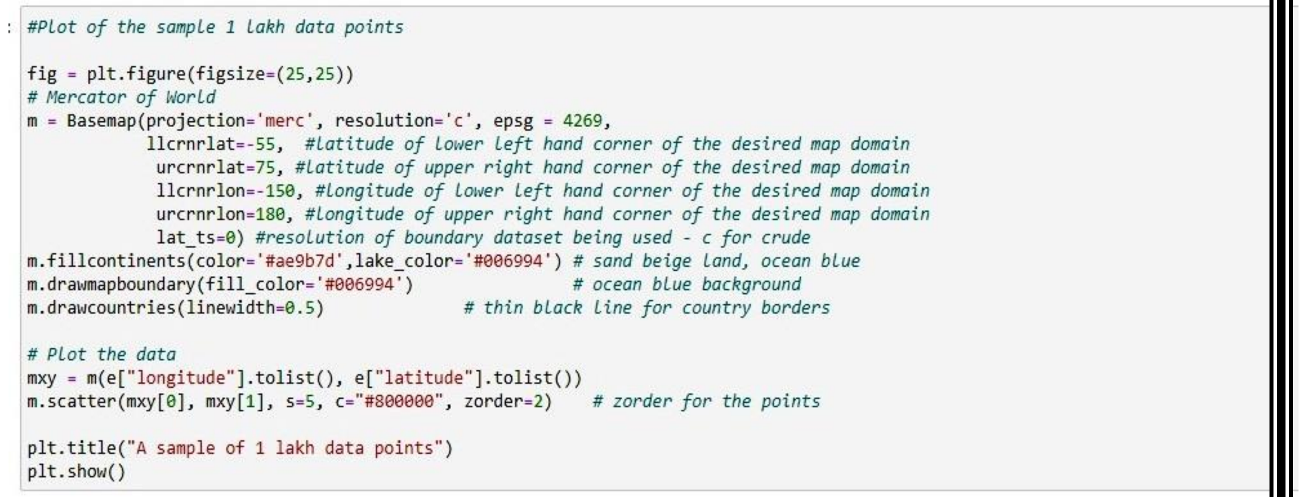
- As an example take Samsung and Huawei, those brands are old and has more brand value compared to Xiaomi, Oppo etc.
- For higher age group, brand value also played a key role and having an income also plays a role.
- The new generation is towards home brand and budget friendly phones, whereas higher age group is more towards brand value.
- As most of the data is from china, we can say Xiaomi is purchased more as it is a home country brand

4. Geospatial Visualization

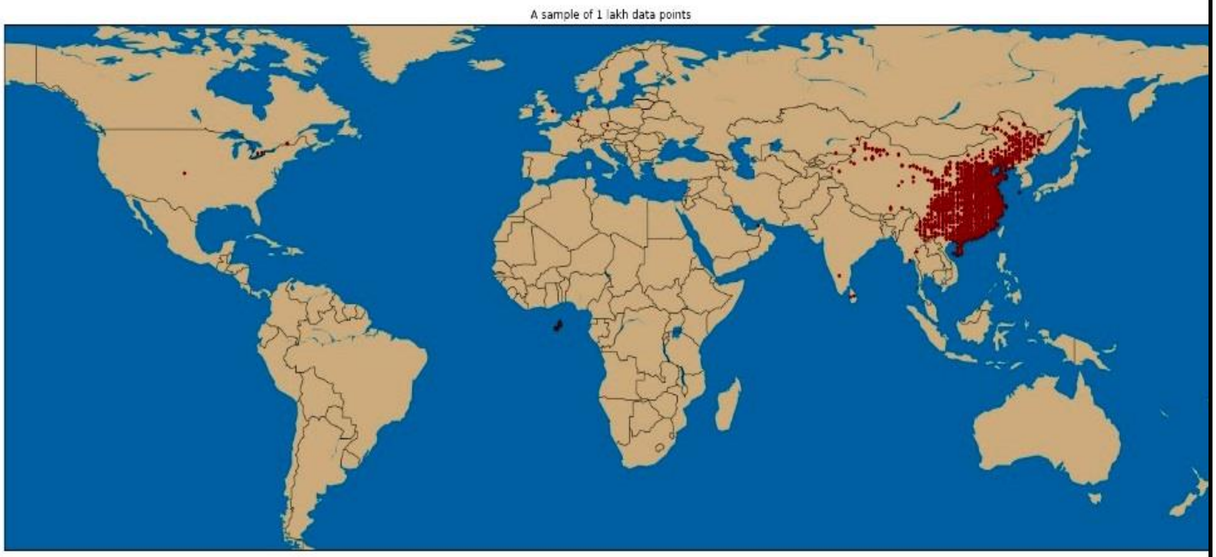
4.1 Plot the visualization plot for a sample of 1 lakh data points



```
#Random sample of 1 lakh data points for map visualizations
e=event.sample(n=100000)
```

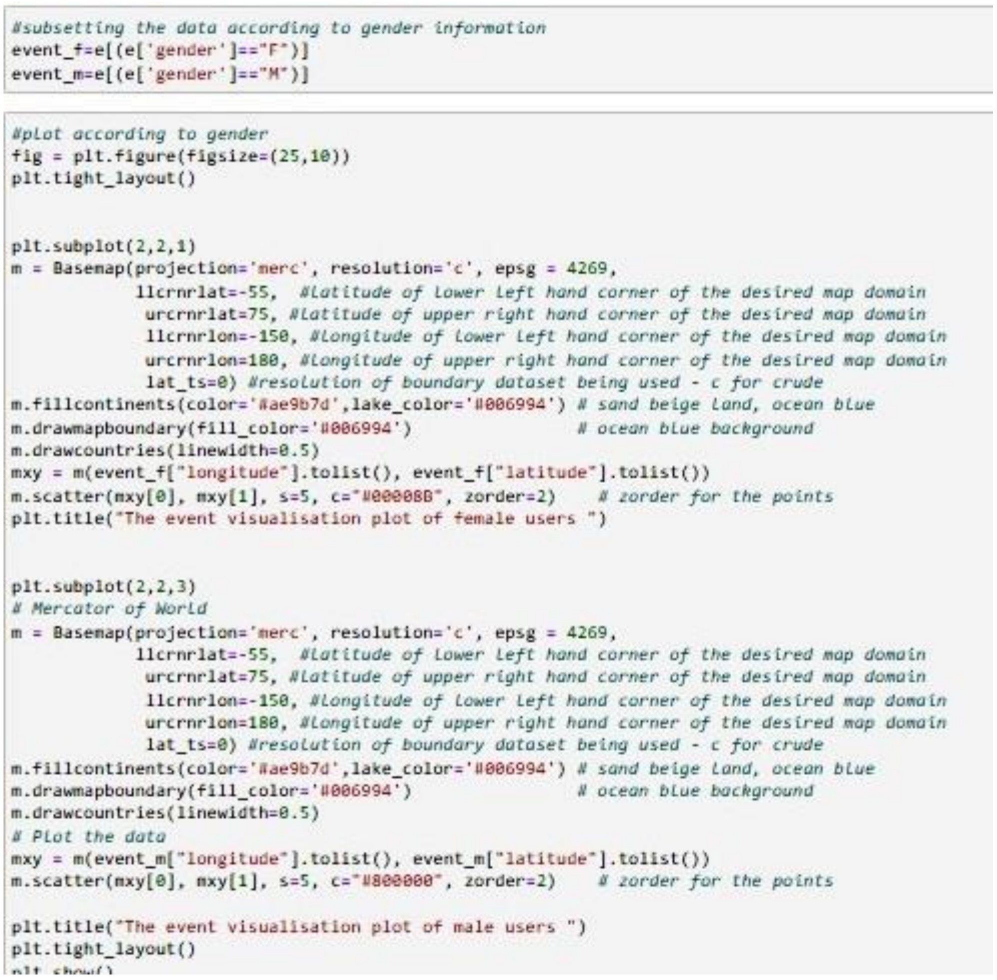


```
#Plot of the sample 1 lakh data points

fig = plt.figure(figsize=(25,25))
# Mercator of World
m = Basemap(projection='merc', resolution='c', epsg = 4269,
            llcrnrlat=-55,  #latitude of lower left hand corner of the desired map domain
             urcrnrlat=75, #latitude of upper right hand corner of the desired map domain
             llcrnrlon=-150, #longitude of lower left hand corner of the desired map domain
             urcrnrlon=180, #longitude of upper right hand corner of the desired map domain
             lat_ts=0) #resolution of boundary dataset being used - c for crude
m.fillcontinents(color='#ae9b7d',lake_color='#006994') # sand beige land, ocean blue
m.drawmapboundary(fill_color='#006994')             # ocean blue background
m.drawcountries(linewidth=0.5)              # thin black line for country borders

# Plot the data
mxy = m(e["longitude"].tolist(), e["latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#800000", zorder=2)    # zorder for the points

plt.title("A sample of 1 lakh data points")
plt.show()
```



- Most of the data points are in and around China region

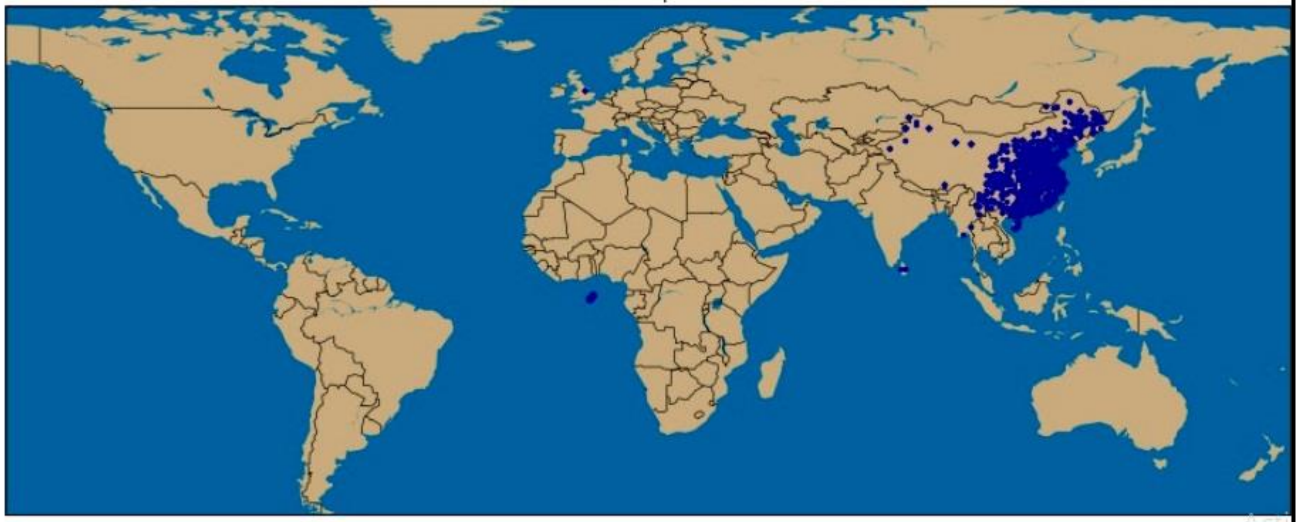
4.2 Compare the event visualization plots based on the users' gender information



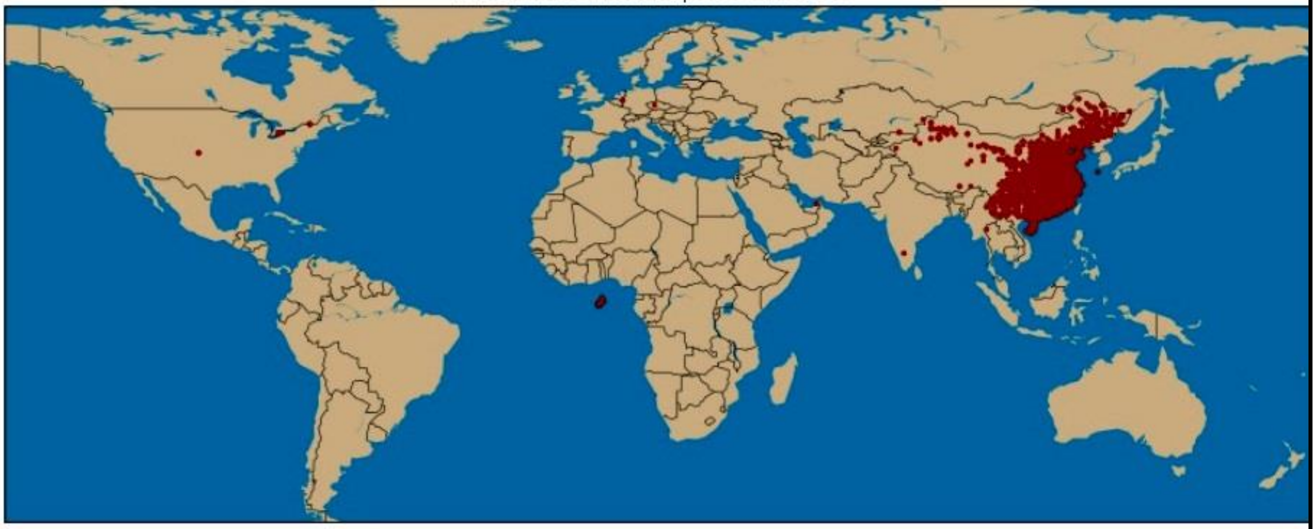
```
#subsetting the data according to gender information
event_f=e[(e['gender']=="F")]
event_m=e[(e['gender']=="M")]
```

```
#plot according to gender
fig = plt.figure(figsize=(25,10))
plt.tight_layout()


plt.subplot(2,2,1)
m = Basemap(projection='merc', resolution='c', epsg = 4269,
            llcrnrlat=-55,  #Latitude of lower left hand corner of the desired map domain
             urcrnrlat=75, #Latitude of upper right hand corner of the desired map domain
             llcrnrlon=-150, #Longitude of lower left hand corner of the desired map domain
             urcrnrlon=180, #Longitude of upper right hand corner of the desired map domain
             lat_ts=0) #resolution of boundary dataset being used - c for crude
m.fillcontinents(color='#ae9b7d',lake_color='#006994') # sand beige land, ocean blue
m.drawmapboundary(fill_color='#006994')              # ocean blue background
m.drawcountries(linewidth=0.5)
mxy = m(event_f["longitude"].tolist(), event_f["latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#00008B", zorder=2)    # zorder for the points
plt.title("The event visualisation plot of female users ")


plt.subplot(2,2,3)
# Mercator of World
m = Basemap(projection='merc', resolution='c', epsg = 4269,
            llcrnrlat=-55,  #Latitude of lower left hand corner of the desired map domain
             urcrnrlat=75, #Latitude of upper right hand corner of the desired map domain
             llcrnrlon=-150, #Longitude of lower left hand corner of the desired map domain
             urcrnrlon=180, #Longitude of upper right hand corner of the desired map domain
             lat_ts=0) #resolution of boundary dataset being used - c for crude
m.fillcontinents(color='#ae9b7d',lake_color='#006994') # sand beige land, ocean blue
m.drawmapboundary(fill_color='#006994')              # ocean blue background
m.drawcountries(linewidth=0.5)
# Plot the data
mxy = m(event_m["longitude"].tolist(), event_m["latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#800000", zorder=2)    # zorder for the points

plt.title("The event visualisation plot of male users ")
plt.tight_layout()
plt.show()
```

The event visualisation plot of female users

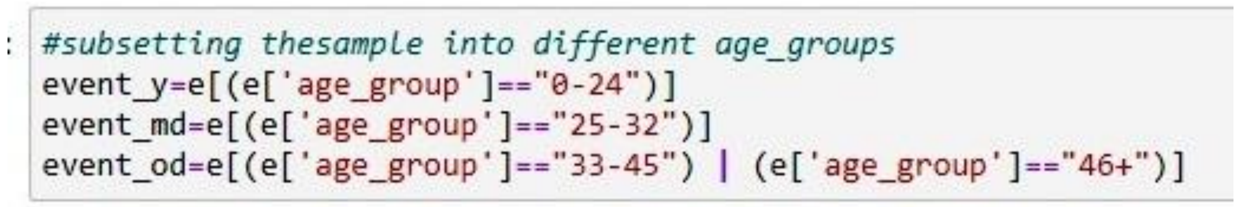


The event visualisation plot of male users

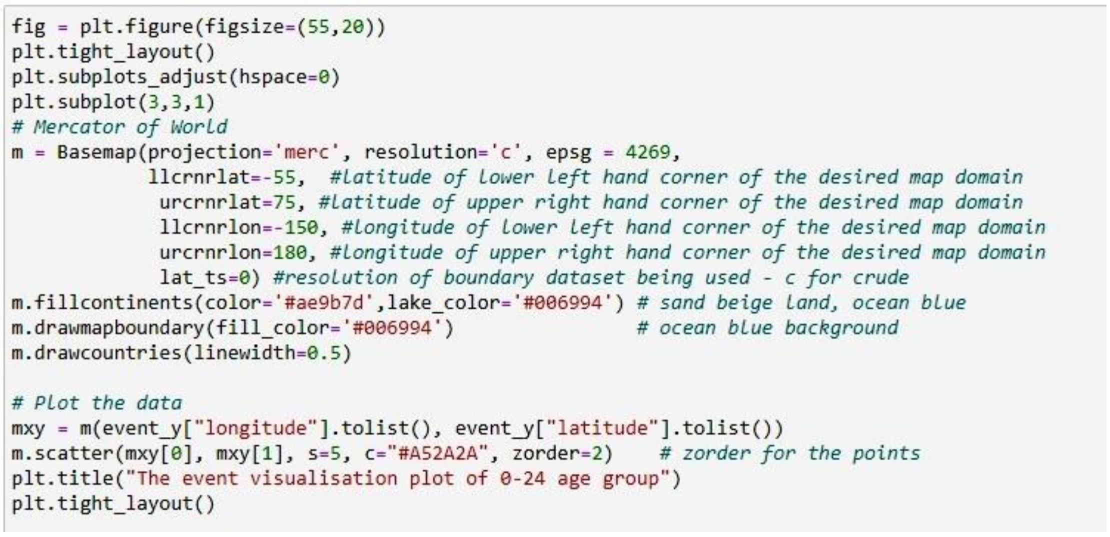


- It is quite evident that the result is in agreement with the results of EDA, that most users are males
- Male users are using the services overseas too as indicated by the visualization

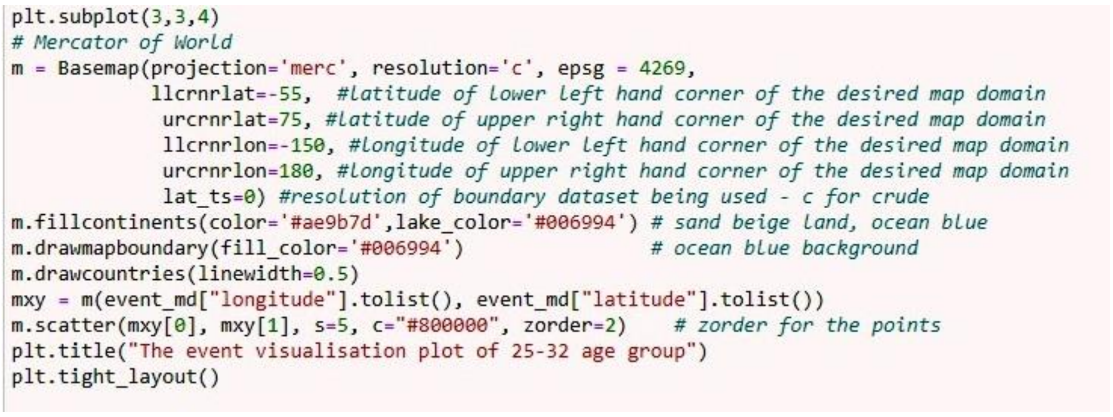
4.3 Compare the event visualization plots based on age groups



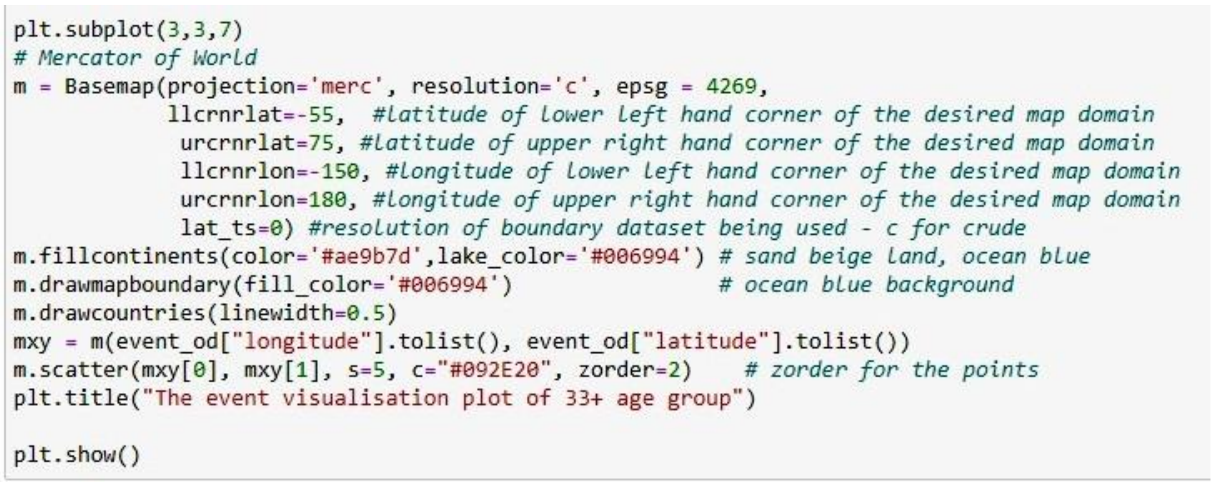
```
#subsetting thesample into different age_groups
event_y=e[(e['age_group']=="0-24")]
event_md=e[(e['age_group']=="25-32")]
event_od=e[(e['age_group']=="33-45") | (e['age_group']=="46+")]
```



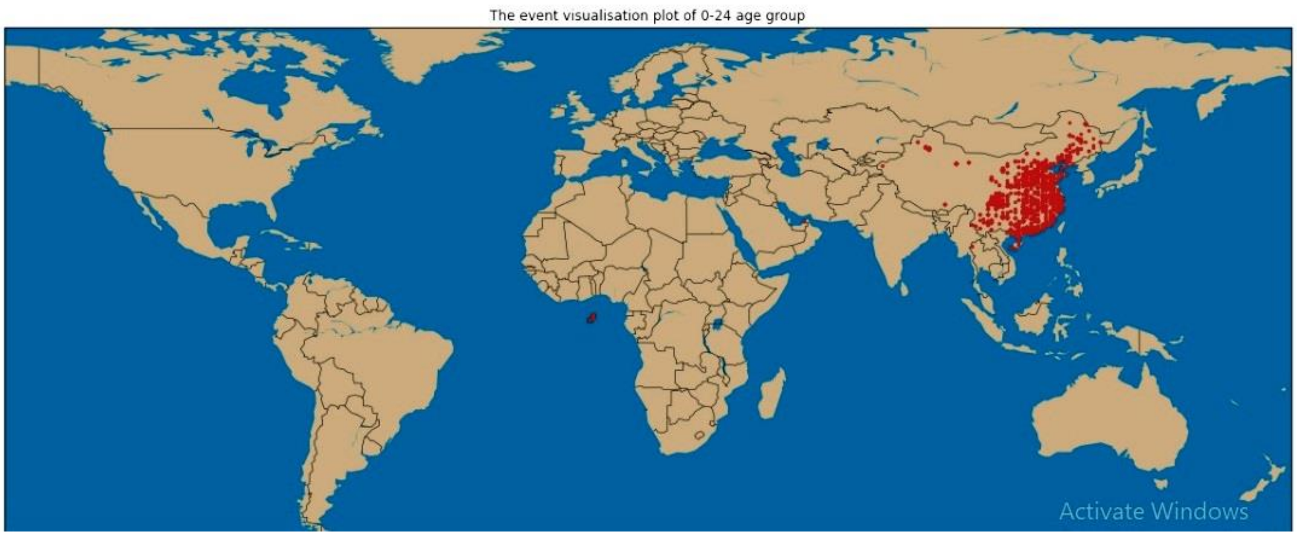
```
fig = plt.figure(figsize=(55,20))
plt.tight_layout()
plt.subplots_adjust(hspace=0)
plt.subplot(3,3,1)
# Mercator of World
m = Basemap(projection='merc', resolution='c', epsg = 4269,
            llcrnrlat=-55,  #latitude of lower left hand corner of the desired map domain
             urcrnrlat=75, #latitude of upper right hand corner of the desired map domain
             llcrnrlon=-150, #longitude of lower left hand corner of the desired map domain
             urcrnrlon=180, #longitude of upper right hand corner of the desired map domain
             lat_ts=0) #resolution of boundary dataset being used - c for crude
m.fillcontinents(color='#ae9b7d',lake_color='#006994') # sand beige land, ocean blue
m.drawmapboundary(fill_color='#006994')                # ocean blue background
m.drawcountries(linewidth=0.5)

# Plot the data
mxy = m(event_y["longitude"].tolist(), event_y["latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#A52A2A", zorder=2)    # zorder for the points
plt.title("The event visualisation plot of 0-24 age group")
plt.tight_layout()
```

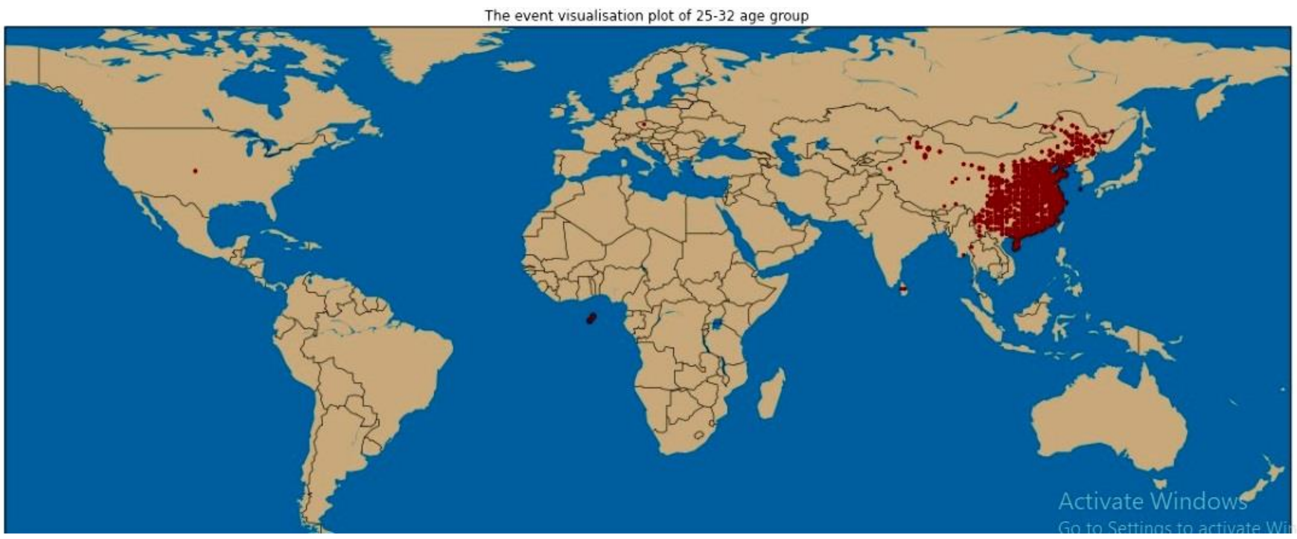


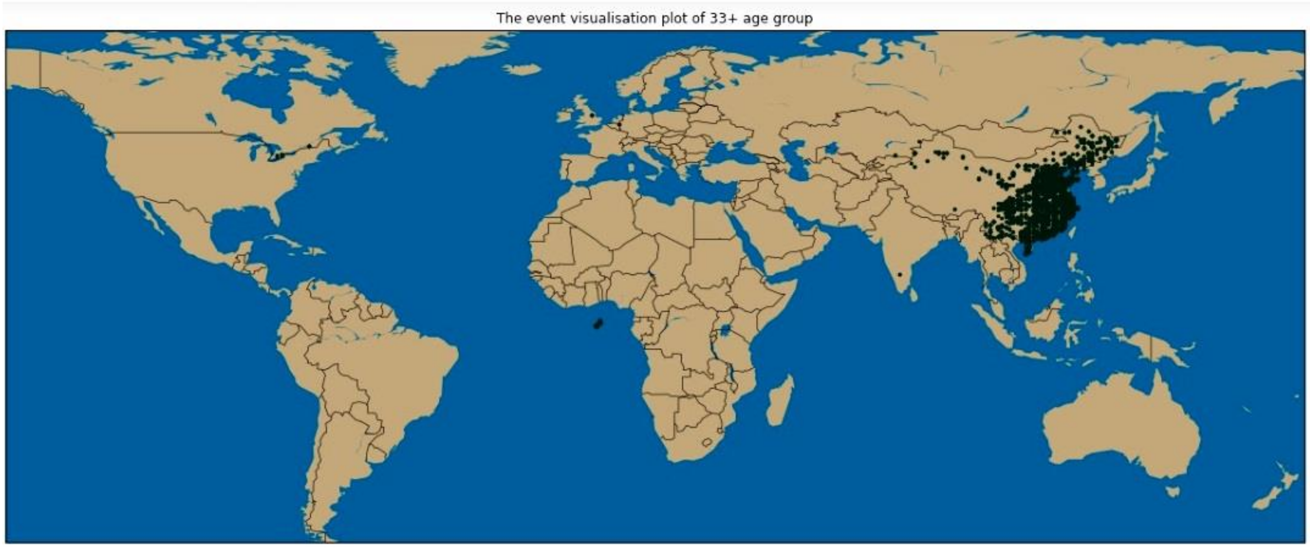
```
plt.subplot(3,3,4)
# Mercator of World
m = Basemap(projection='merc', resolution='c', epsg = 4269,
            llcrnrlat=-55,  #latitude of lower left hand corner of the desired map domain
             urcrnrlat=75, #latitude of upper right hand corner of the desired map domain
             llcrnrlon=-150, #longitude of lower left hand corner of the desired map domain
             urcrnrlon=180, #longitude of upper right hand corner of the desired map domain
             lat_ts=0) #resolution of boundary dataset being used - c for crude
m.fillcontinents(color='#ae9b7d',lake_color='#006994') # sand beige land, ocean blue
m.drawmapboundary(fill_color='#006994')                # ocean blue background
m.drawcountries(linewidth=0.5)
mxy = m(event_md["longitude"].tolist(), event_md["latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#800000", zorder=2)    # zorder for the points
plt.title("The event visualisation plot of 25-32 age group")
plt.tight_layout()
```



```
plt.subplot(3,3,7)
# Mercator of World
m = Basemap(projection='merc', resolution='c', epsg = 4269,
            llcrnrlat=-55,  #latitude of lower left hand corner of the desired map domain
             urcrnrlat=75, #latitude of upper right hand corner of the desired map domain
             llcrnrlon=-150, #longitude of lower left hand corner of the desired map domain
             urcrnrlon=180, #longitude of upper right hand corner of the desired map domain
             lat_ts=0) #resolution of boundary dataset being used - c for crude
m.fillcontinents(color='#ae9b7d',lake_color='#006994') # sand beige land, ocean blue
m.drawmapboundary(fill_color='#006994')                # ocean blue background
m.drawcountries(linewidth=0.5)
mxy = m(event_od["longitude"].tolist(), event_od["latitude"].tolist())
m.scatter(mxy[0], mxy[1], s=5, c="#092E20", zorder=2)    # zorder for the points
plt.title("The event visualisation plot of 33+ age group")

plt.show()
```




The event visualisation plot of 25-32 age group
Activate Windows
Go to Settings to activate Wi



- The plots points that the 25-32 age groups have the more number of users followed by the 33+ age group.

- The least number of users are in the age group of 0-24. This again comes in agreement with the EDA results
- The number users in overseas is more for age group 24-32 and 32+, which again can be explained to the fact that these age group tend to travel or relocate for further studies and career prospects.

5. Insights from EDA and geospatial visualizations

- The mean age of users is 29 years old
- Majority of the users are males, about 64%. Around 36% of users are females.
- Most of the male and female users belong to 25-33.
- The least number of users in both genders belong to 0-24 age group.
- The mean age of both genders are same, And both lie at 29 years old.
- The oldest (age wise) users are females.
- The age of genders follow a normal distribution.
- Other than the 32+ age group, all age groups across genders have same mean age.
- 32+ age group males have a mean age of 38 and females have a mean age of 41 years old.
- Most of the users use the phone brand Xiaomi, followed by Samsung and Huawei
- Around 69% of devices have restricted permission to access info by apps
- Mid-week is where there is more number of events triggered.
- Most events are triggered on Tuesdays followed by Thursdays and Wednesdays
- Throughout one week, most number of events is triggered during 10 in the morning and 10 at night, both around 60K in count
- Other than early hours of 1 am till 5 am, there are at least 38K events triggered
- Less number of events is triggered by female users, which in turn is the result of less number of female users.
- The phone usage pattern remains the same for both the genders, high number of events during 10 in the morning and 10 at night and lest number of events during early hours of 1 am till 5 am.
- There is significant difference in the distribution of events among age groups, across the week

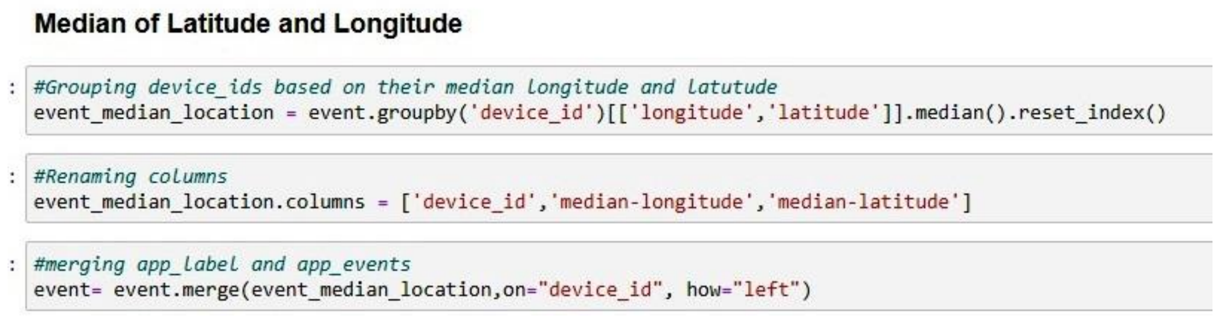
- Most number of events is triggered by age group 25-32, followed by age group 33-45
- There is a huge difference of about 50K number of events triggered by age groups of 25+ and age group less than 25.
- The age group 0-24 exhibits more or less a steady pattern throughout the week and so is the 46+ age group
- The 25-32 age group shows the highest number of events across the week with a steady pattern followed by 33-45 age group
- The stacked bar chart is in agreement with the previously given pie chart of the phone brands, with Xiaomi being the top brand among the male and female users, followed by Samsung and Huawei
- The above shown app categories are the most used apps. Industry and property apps being most used followed by finances, internet banking and games
- Male users are more which an expected outcome
- Pricing plays a key role and we can see that mobile brands with competitive process are being purchased more.
- In all the brands, age group of 25-32 bought more phones, but the distribution of 0-24 and 33-45 is varied based on prices.
- In general observation, 0-24 age-group people got more phones from the company whose prices are comparatively less.
- As an example take Samsung and Huawei, those brands are old and has more brand value compared to Xiaomi, Oppo etc.
- For higher age group, brand value also played a key role and having an income also plays a role.
- The new generation is towards home brand and budget friendly phones, whereas higher age group is more towards brand value.
- As most of the data is from china, we can say Xiaomi is purchased more as it is a home country brand
- Most of the data points are in and around China region
- It is quite evident that the result is in agreement with the results of EDA, that most users are males
- Male users are using the services overseas too as indicated by the visualization
- The plots points that the 25-32 age groups have the more number of users followed by the 33+ age group.
- The least number of users are in the age group of 0-24. This again comes in agreement with the EDA results

- The number users in overseas is more for age group 24-32 and 32+, which again can be explained to the fact that these age group tend to travel or relocate for further studies and career prospects.

6. Feature Engineering

New features were added for all three basic datasets and a few are dropped in light of the new features.

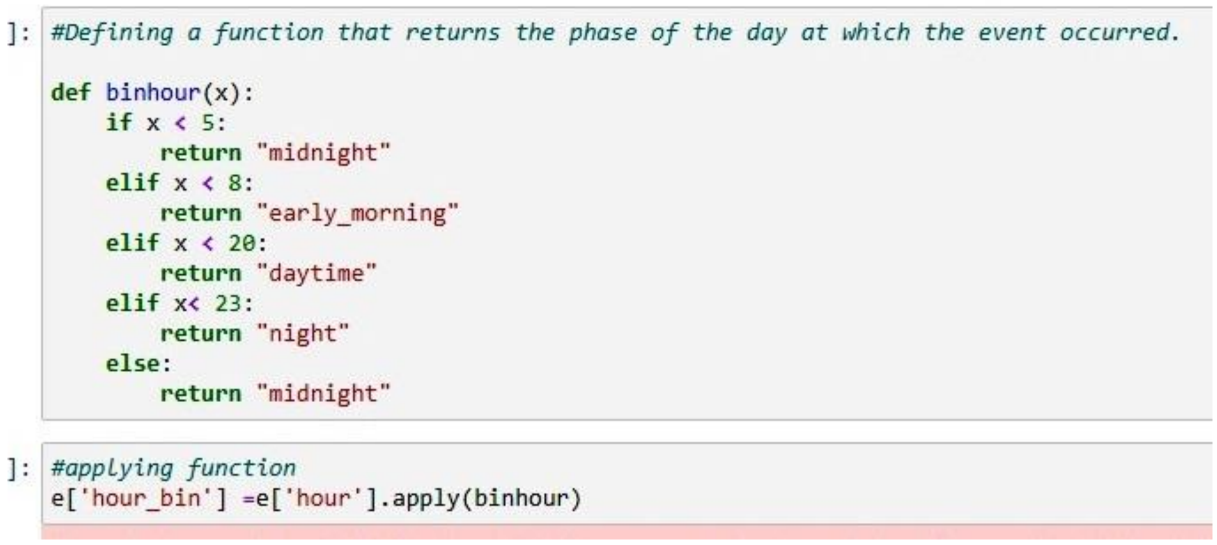
- The median of latitude and longitudes are taken in event dataset. This does help in imputing some of the missing data information, since some of the apps may have user permissions.



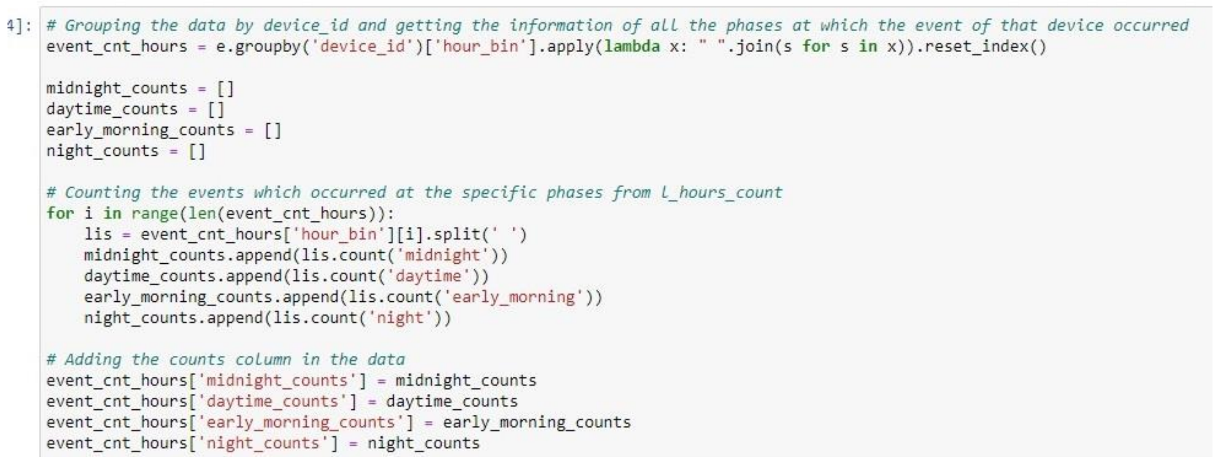
Median of Latitude and Longitude

```
: #Grouping device_ids based on their median longitude and latutude
  event_median_location = event.groupby('device_id')[['longitude','latitude']].median().reset_index()

: #Renaming columns
  event_median_location.columns = ['device_id','median-longitude','median-latitude']

: #merging app_label and app_events
  event= event.merge(event_median_location,on="device_id", how="left")
```

- The number of events at different points on the day calculated in events dataset- ie during daytime, afternoon, night etc. This feature will give an idea regarding the age of the users. We can take a guess that young people tend to use phones more and as a result trigger more number of events throughout the day

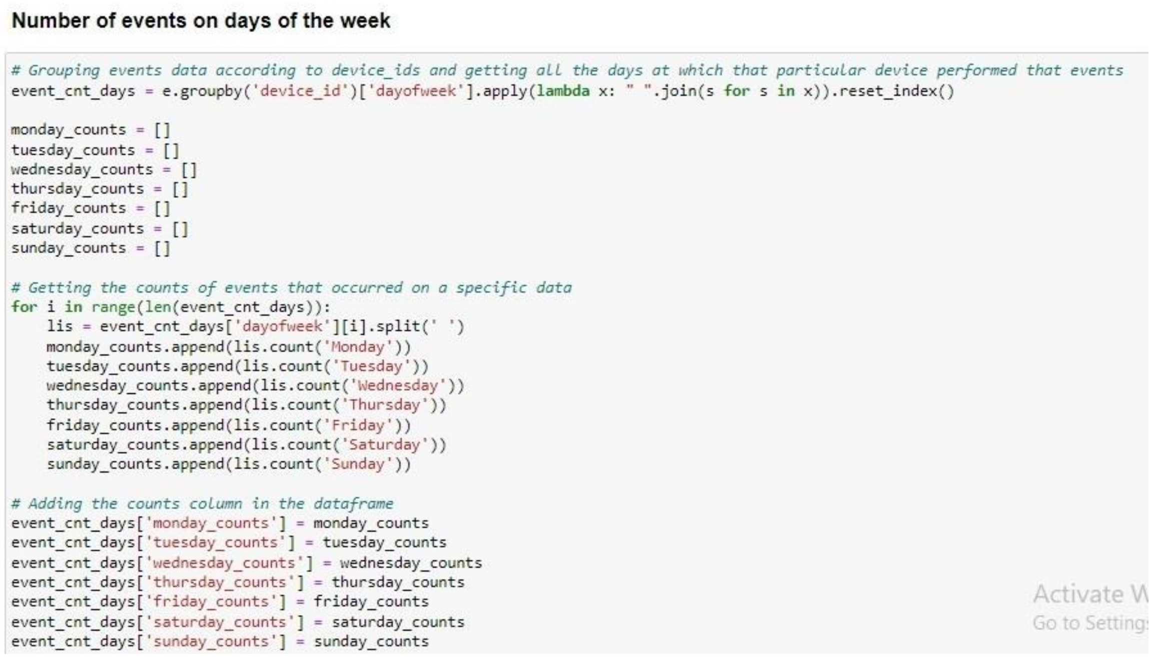


```
]: #Defining a function that returns the phase of the day at which the event occurred.

   def binhour(x):
       if x < 5:
           return "midnight"
       elif x < 8:
           return "early_morning"
       elif x < 20:
           return "daytime"
       elif x< 23:
           return "night"
       else:
           return "midnight"

]: #applying function
   e['hour_bin'] =e['hour'].apply(binhour)
```



```
4]: # Grouping the data by device_id and getting the information of all the phases at which the event of that device occurred
    event_cnt_hours = e.groupby('device_id')['hour_bin'].apply(lambda x: " ".join(s for s in x)).reset_index()

    midnight_counts = []
    daytime_counts = []
    early_morning_counts = []
    night_counts = []

    # Counting the events which occurred at the specific phases from l_hours_count
    for i in range(len(event_cnt_hours)):
        lis = event_cnt_hours['hour_bin'][i].split(' ')
        midnight_counts.append(lis.count('midnight'))
        daytime_counts.append(lis.count('daytime'))
        early_morning_counts.append(lis.count('early_morning'))
        night_counts.append(lis.count('night'))

    # Adding the counts column in the data
    event_cnt_hours['midnight_counts'] = midnight_counts
    event_cnt_hours['daytime_counts'] = daytime_counts
    event_cnt_hours['early_morning_counts'] = early_morning_counts
    event_cnt_hours['night_counts'] = night_counts
```

- The number of events throughout the days of the week is calculated in events dataset. This feature again can help us to determine the age of users, since people who are working tend to use phones less at the start of the week and increase as weekend nears.



Number of events on days of the week

```
# Grouping events data according to device_ids and getting all the days at which that particular device performed that events
event_cnt_days = e.groupby('device_id')['dayofweek'].apply(lambda x: " ".join(s for s in x)).reset_index()

monday_counts = []
tuesday_counts = []
wednesday_counts = []
thursday_counts = []
friday_counts = []
saturday_counts = []
sunday_counts = []

# Getting the counts of events that occurred on a specific data
for i in range(len(event_cnt_days)):
    lis = event_cnt_days['dayofweek'][i].split(' ')
    monday_counts.append(lis.count('Monday'))
    tuesday_counts.append(lis.count('Tuesday'))
    wednesday_counts.append(lis.count('Wednesday'))
    thursday_counts.append(lis.count('Thursday'))
    friday_counts.append(lis.count('Friday'))
    saturday_counts.append(lis.count('Saturday'))
    sunday_counts.append(lis.count('Sunday'))

# Adding the counts column in the dataframe
event_cnt_days['monday_counts'] = monday_counts
event_cnt_days['tuesday_counts'] = tuesday_counts
event_cnt_days['wednesday_counts'] = wednesday_counts
event_cnt_days['thursday_counts'] = thursday_counts
event_cnt_days['friday_counts'] = friday_counts
event_cnt_days['saturday_counts'] = saturday_counts
event_cnt_days['sunday_counts'] = sunday_counts
```

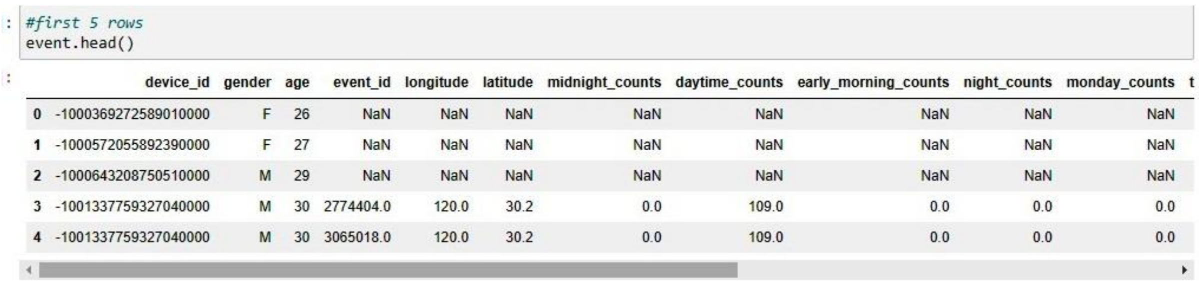
- Top 10 phone brands are kept and rest is converted to other in non_event dataset. The top 10 phone brands covers almost 90% of data. This is done for facilitating in encoding and thus reducing noise in data. This top brands tend to have a dependence on age group-which is one of the target variable in this project.

- Top 150 device models are kept and rest is converted to other in non_event dataset. This selection covers 85% of data. This is also done for facilitating in encoding and reducing noise in data.

- Top 50 active app_categories are kept and rest is converted into other in the app data. This is one of the most important feature which can determine the gender of the users, which is a target variable in this project.

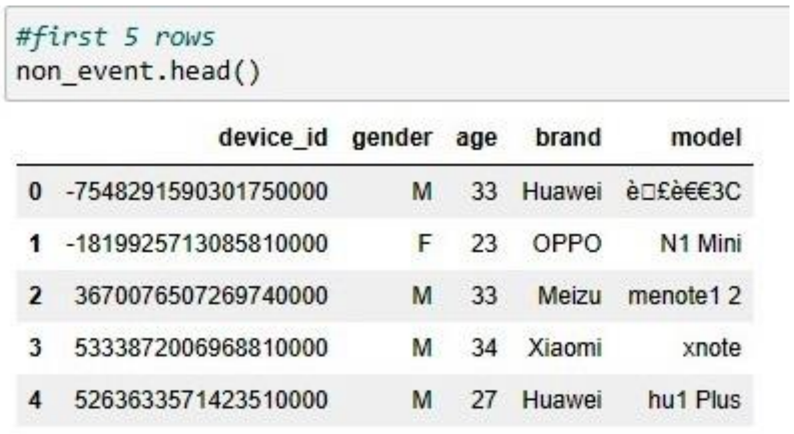
7. Data Preparation

Repeated, unnecessary and similar data representing columns are dropped from all datasets. . The following are the final datasets before merging to prepare the final dataset for modeling purposes.



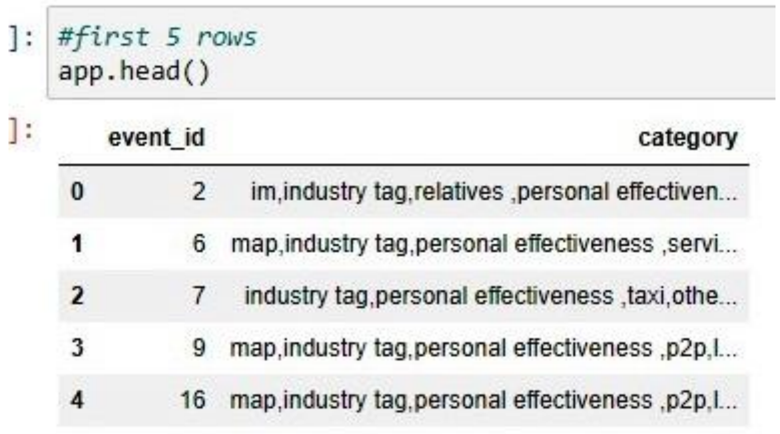
```
#first 5 rows
event.head()
```

	device_id	gender	age	event_id	longitude	latitude	midnight_counts	daytime_counts	early_morning_counts	night_counts	monday_counts	t
0	-1000369272589010000	F	26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	-1000572055892390000	F	27	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	-1000643208750510000	M	29	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	-1001337759327040000	M	30	2774404.0	120.0	30.2	0.0	109.0	0.0	0.0	0.0	
4	-1001337759327040000	M	30	3065018.0	120.0	30.2	0.0	109.0	0.0	0.0	0.0	



```
#first 5 rows
non_event.head()
```

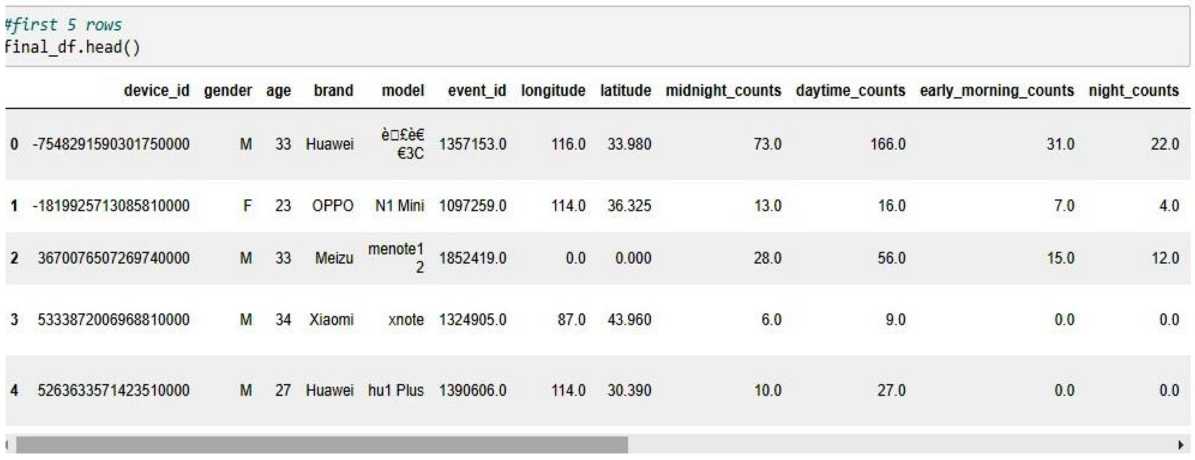
	device_id	gender	age	brand	model
0	-7548291590301750000	M	33	Huawei	è□£è€€3C
1	-1819925713085810000	F	23	OPPO	N1 Mini
2	3670076507269740000	M	33	Meizu	menote1 2
3	5333872006968810000	M	34	Xiaomi	xnote
4	5263633571423510000	M	27	Huawei	hu1 Plus



```
#first 5 rows
app.head()
```

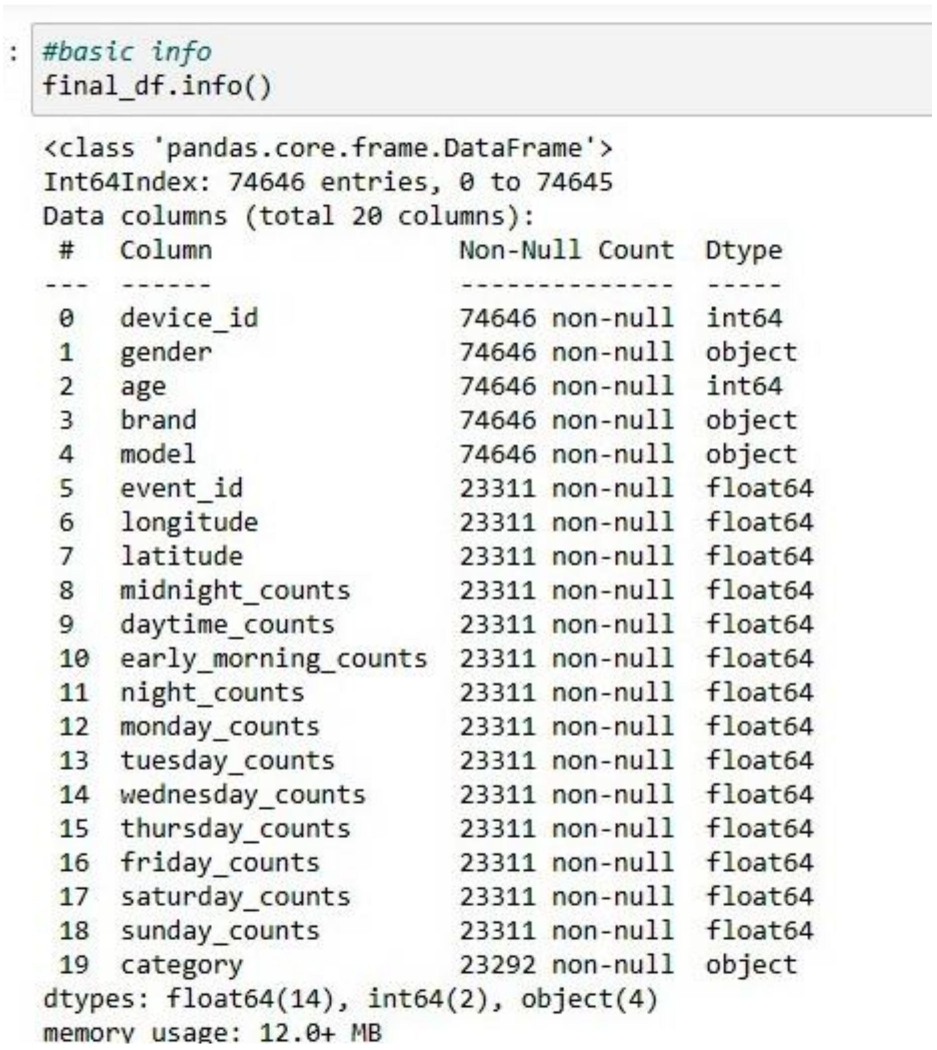
	event_id	category
0	2	im,industry tag,relatives ,personal effectiven...
1	6	map,industry tag,personal effectiveness ,servi...
2	7	industry tag,personal effectiveness ,taxi,othe...
3	9	map,industry tag,personal effectiveness ,p2p,l...
4	16	map,industry tag,personal effectiveness ,p2p,l...

Events data is merged with app data. Duplicates are dropped and unnecessary columns are dropped. This resulting dataset is merged with non_event data. The merges are left and is done on device_id. The final dataset is as below:



```
#first 5 rows
final_df.head()
```

	device_id	gender	age	brand	model	event_id	longitude	latitude	midnight_counts	daytime_counts	early_morning_counts	night_counts
0	-7548291590301750000	M	33	Huawei	è□£è€ €3C	1357153.0	116.0	33.980	73.0	166.0	31.0	22.0
1	-1819925713085810000	F	23	OPPO	N1 Mini	1097259.0	114.0	36.325	13.0	16.0	7.0	4.0
2	3670076507269740000	M	33	Meizu	menote1 2	1852419.0	0.0	0.000	28.0	56.0	15.0	12.0
3	5333872006968810000	M	34	Xiaomi	xnote	1324905.0	87.0	43.960	6.0	9.0	0.0	0.0
4	5263633571423510000	M	27	Huawei	hu1 Plus	1390606.0	114.0	30.390	10.0	27.0	0.0	0.0



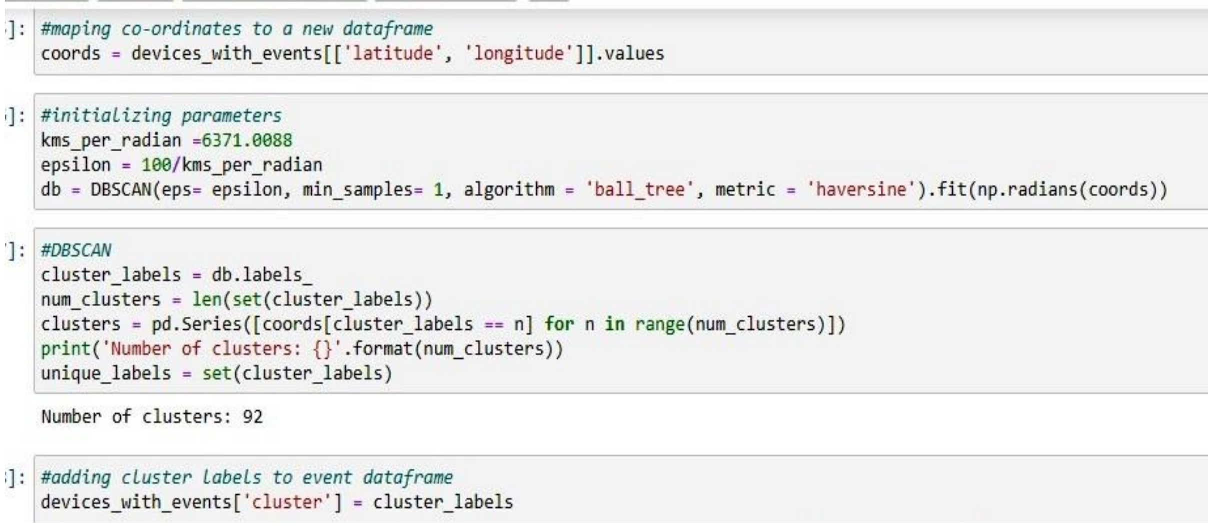
```
#basic info
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74646 entries, 0 to 74645
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   device_id             74646 non-null  int64  
 1   gender                74646 non-null  object 
 2   age                   74646 non-null  int64  
 3   brand                 74646 non-null  object 
 4   model                 74646 non-null  object 
 5   event_id              23311 non-null  float64
 6   longitude             23311 non-null  float64
 7   latitude              23311 non-null  float64
 8   midnight_counts       23311 non-null  float64
 9   daytime_counts        23311 non-null  float64
 10  early_morning_counts  23311 non-null  float64
 11  night_counts          23311 non-null  float64
 12  monday_counts         23311 non-null  float64
 13  tuesday_counts        23311 non-null  float64
 14  wednesday_counts      23311 non-null  float64
 15  thursday_counts       23311 non-null  float64
 16  friday_counts         23311 non-null  float64
 17  saturday_counts       23311 non-null  float64
 18  sunday_counts         23311 non-null  float64
 19  category              23292 non-null  object 
dtypes: float64(14), int64(2), object(4)
memory usage: 12.0+ MB
```

8. DBSCAN Clustering

DBSCAN Clustering algorithm is run as a part of Data preprocessing. Here we have taken a 100 km/radian for calculating epsilon and min_sample=1 as parameter, because of the presence of users in overseas. These users are crucial for age prediction.

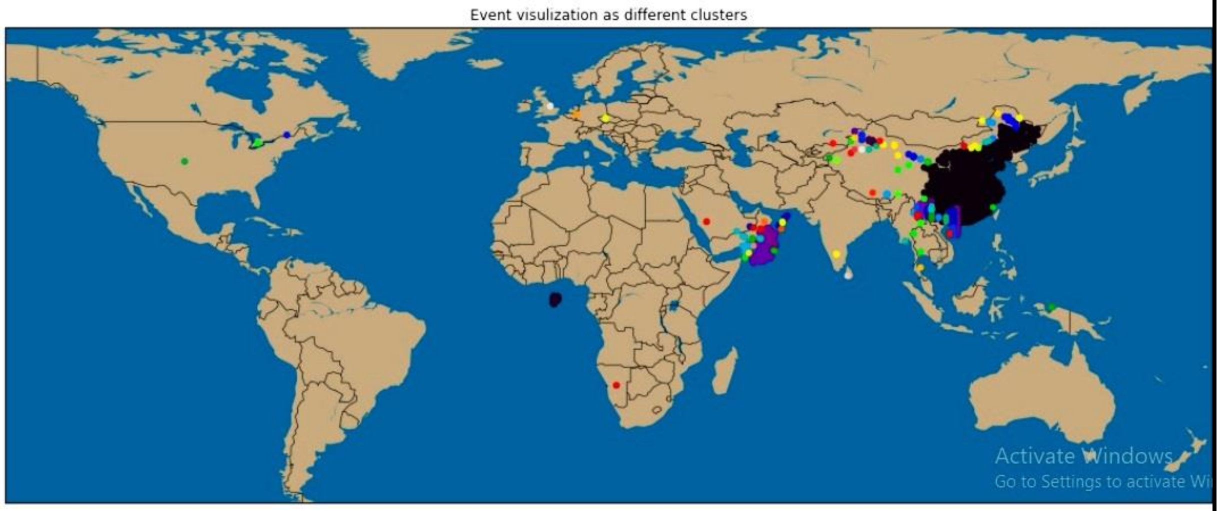
DBSCAN is run on data with events data and so the final dataset is filtered and used in the modeling.



```
]: #maping co-ordinates to a new dataframe
   coords = devices_with_events[['latitude', 'longitude']].values

]: #initializing parameters
   kms_per_radian =6371.0088
   epsilon = 100/kms_per_radian
   db = DBSCAN(eps= epsilon, min_samples= 1, algorithm = 'ball_tree', metric = 'haversine').fit(np.radians(coords))

]: #DBSCAN
   cluster_labels = db.labels_
   num_clusters = len(set(cluster_labels))
   clusters = pd.Series([coords[cluster_labels == n] for n in range(num_clusters)])
   print('Number of clusters: {}'.format(num_clusters))
   unique_labels = set(cluster_labels)

   Number of clusters: 92

]: #adding cluster labels to event dataframe
   devices_with_events['cluster'] = cluster_labels
```

As we can see the model gave 92 clusters. These are added back into the final dataset afterwards to be used instead of location data.

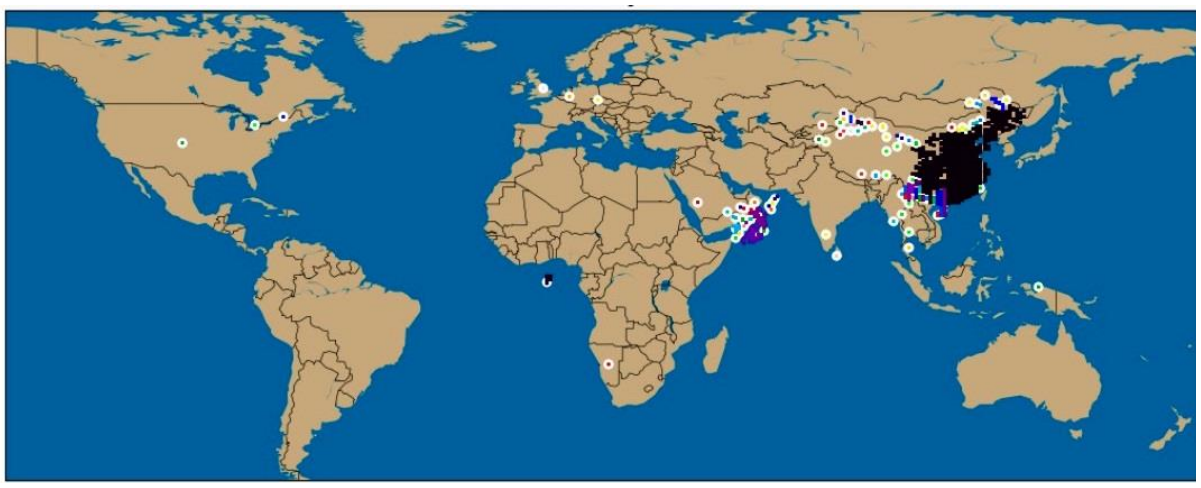
The clusters are visualized



Event visulization as different clusters
Activate Windows
Go to Settings to activate Wi

.

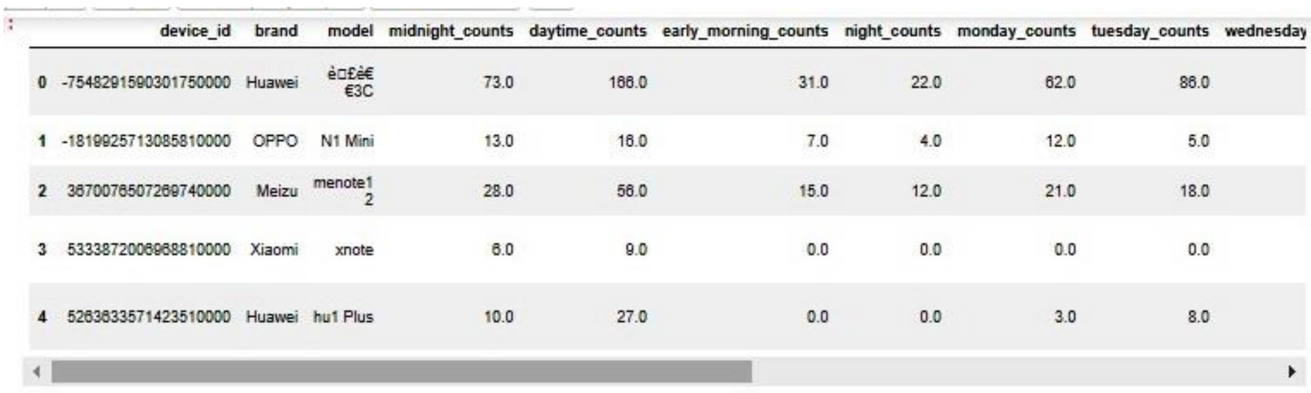
The clusters along with their centroids



Once the clusters are found, they are merged back into the final dataset. This is yet another great feature, since it represents location based data of the users. This is more comfortable to handle than the latitude, longitude data that can cause noise in data.

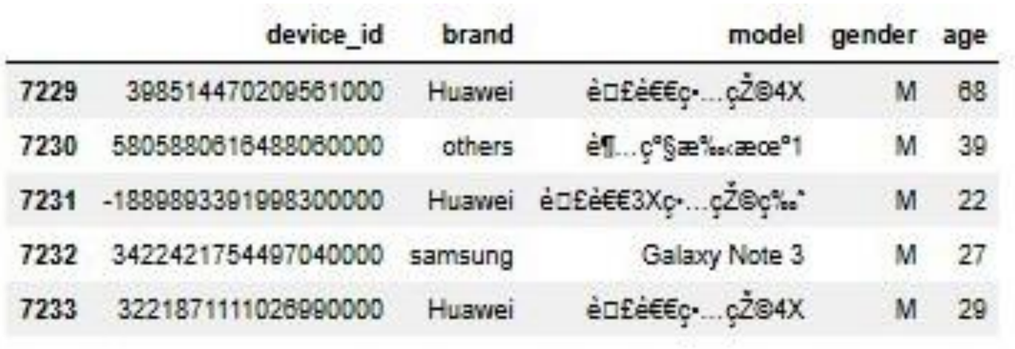
The final dataset is prepared by integrating cluster data and dropping location data along with some other. The final dataset is then split to cater for the two scenarios that we are considering for the project.

- **Scenario 1**: The user allows the application to trigger events that collect their usage behaviour.



	device_id	brand	model	midnight_counts	daytime_counts	early_morning_counts	night_counts	monday_counts	tuesday_counts	wednesday
0	-7548291590301750000	Huawei	è□£è€ €3C	73.0	166.0	31.0	22.0	62.0	86.0	
1	-1819925713085810000	OPPO	N1 Mini	13.0	16.0	7.0	4.0	12.0	5.0	
2	3670076507269740000	Meizu	menote1 2	28.0	56.0	15.0	12.0	21.0	18.0	
3	5333872006968810000	Xiaomi	xnote	6.0	9.0	0.0	0.0	0.0	0.0	
4	5263633571423510000	Huawei	hu1 Plus	10.0	27.0	0.0	0.0	3.0	8.0	

- **Scenario 2**: User activity is unavailable, thereby leaving us with only the device information. Therefore, the event data will not be available.



	device_id	brand	model	gender	age
7229	398514470209561000	Huawei	è□£è€€ç•…çŽ®4X	M	68
7230	5805880616488060000	others	è¶…ç°§æ‰‹æœº1	M	39
7231	-1888893391998300000	Huawei	è□£è€€3Xç•…çŽ®ç‰ˆ	M	22
7232	3422421754497040000	samsung	Galaxy Note 3	M	27
7233	3221871111026990000	Huawei	è□£è€€ç•…çŽ®4X	M	29

9. Model Development

9.1 Age Prediction

Model is developed on the merged data of all three basic datasets and the cluster labels provided by the DBSCAN modeling.

9.1.1 Scenario 1 (with user information)

Data Preprocessing:

- The category column which has app categories separated by comma as a single data point is split and made into individual columns using MultiLabelBinarizer. The resulting columns resemble dummy variable encoding.
- All the categorical variables are converted to the data type category for encoding purposes.
- Dummy variable encoding is done using pd.get_dummies
- The data is split into train and test as per instructed.
- The test dataset is converted into a csv file for further use in model deloyment
- X_train. y_train, X_test, y_test is spit afterwards
- **Feature Scaling** is done for numerical variables. The MinMaxscalar is used since the rest of the data is encoded and is binary format.

Model Building

- The target variable "age" is a continuous variable, so as a starting point and for simplicity, linear regression algorithm is considered. The modeling can also be done using multiclass logistic regression algorithm. Since most of the variables are categorical, logistic regression may get more complicated to execute. So to avoid the complexity and ease of understanding the metrics and execution the basic classic linear regression algorithm is opted for modeling.
- The model is built with statsmodels.api GLS . After initial training the model is evaluated on the test set and the metrics were calculated.
- For better model performance an XGBoost regressor is trained. It is run with GridSearchCV to attain the best parameters for training the model. After initial training the model is evaluated on the test set and the metrics were calculated.
- Once more for better performance and accuracy, StackingCV Regressor is trained. In this initially a linear regression is done followed by a Random forest regression. The random forest regression is again run with GridSearchCV to bet the optimum parameter. Both these regressors are run and their result is fed to the XGBoost regressor and the model is trained. After initial training the model is evaluated on the test set and the metrics were calculated.
- Once the model is build, the best model is pickled for model deployment.

9.1.2 Scenario 2 (without user information)

Data Preprocessing:

- The category column which has app categories separated by comma as a single data point is split and made into individual columns using MultiLabelBinarizer. The resulting columns resemble dummy variable encoding.
- All the categorical variables are converted to the data type category for encoding purposes.
- Dummy variable encoding is done using pd.get_dummies
- The data is split into train and test as per instructed.
- X_train. y_train, X_test, y_test is spit afterwards

Model Building

- The target variable "age" is a continuous variable, so as a starting point and for simplicity, linear regression algorithm is considered. The modeling can also be done using multiclass logistic regression algorithm. Since most of the variables are categorical, logistic regression may get more complicated to execute. So to avoid the complexity and ease of understanding the metrics and execution the basic classic linear regression algorithm is opted for modeling.
- The model is built with statsmodels.api GLS . After initial training the model is evaluated on the test set and the metrics were calculated.
- For better model performance an XGBoost regressor is trained. It is run with GridSearchCV to attain the best parameters for training the model. After initial training the model is evaluated on the test set and the metrics were calculated.
- Once more for better performance and accuracy, StackingCV Regressor is trained. In this initially a linear regression is done followed by a Random forest regression. The random forest regression is again run with GridSearchCV to bet the optimum parameter. Both these regressors are run and their result is fed to the XGBoost regressor and the model is trained. After initial training the model is evaluated on the test set and the metrics were calculated.

9.2 Gender Prediction

Model is developed on the merged data of all three basic datasets and the cluster labels   provided by the DBSCAN modeling.

9.2.1 Scenario 1 (with user information)

Data Preprocessing:

- A dictionary is created for mapping the values for gender- 0 for females and 1 for males.
- The category column which has app categories separated by comma as a single data point is split and made into individual columns using MultiLabelBinarizer. The resulting columns resemble dummy variable encoding.

- All the categorical variables are converted to the data type category for encoding purposes.
- Dummy variable encoding is done using pd.get_dummies
- The data is split into train and test as per instructed.
- The test dataset is converted into a csv file for further use in model deloyment
- X_train. y_train, X_test, y_test is spit afterwards
- **Feature Scaling** is done for numerical variables. The StandardScalar is used.

Model Building

- The target variable "gender" is a categorical variable and a binary one, so logistic regression algorithm is used for training..
- The model is built with statsmodels.api GLM model . After initial training the model is evaluated on the test set and the metrics were calculated.
- For better model performance an XGBoostClassifierr is trained. It is run with GridSearchCV to attain the best parameters for training the model. After initial training the model is evaluated on the test set and the metrics were calculated.
- Once more for better performance and accuracy, StackingCV Classifier is trained. In this initially a logistic regression is done followed by a Random forest classification. The random forest regression is again run with GridSearchCV to bet the optimum parameter. Both these classifiers are run and their resulting probabilities are fed to the XGBoost classifier and the model is trained. After initial training the model is evaluated on the test set and the metrics were calculated.
- Once the model is build, the best model is pickled for model deployment.

9.1.2 Scenario 2 (without user information)

Data Preprocessing:

- A dictionary is created for mapping the values for gender- 0 for females and 1 for males.
- The category column which has app categories separated by comma as a single data point is split and made into individual columns using MultiLabelBinarizer. The resulting columns resemble dummy variable encoding.

- All the categorical variables are converted to the data type category for encoding purposes.
- Dummy variable encoding is done using pd.get_dummies
- The data is split into train and test as per instructed.
- The test dataset is converted into a csv file for further use in model deloyment
- X_train. y_train, X_test, y_test is spit afterwards

Model Building

- The target variable "gender" is a categorical variable and a binary one, so logistic regression algorithm is used for training.
- The model is built with statsmodels.api GLM model . After initial training the model is evaluated on the test set and the metrics were calculated.
- For better model performance an XGBoostClassifierr is trained. It is run with GridSearchCV to attain the best parameters for training the model. After initial training the model is evaluated on the test set and the metrics were calculated.
- Once more for better performance and accuracy, StackingCV Classifier is trained. In this initially a logistic regression is done followed by a Random forest classification. The random forest regression is again run with GridSearchCV to bet the optimum parameter. Both these classifiers are run and their resulting probabilities are fed to the XGBoost classifier and the model is trained. After initial training the model is evaluated on the test set and the metrics were calculated’

10. Model evaluation

10.1 Age Prediction

Models	Metrics	Scenario 1		Scenario 2	
		Train	Test	Train	Test
Linear Regression	**R2 Squared**	0.099	0.1260	0.02413	0.02878
	RMSE	9.300	9.327	9.7616	9.6041
	PPD	-5.2836	-6.324365	-7.660	-7.577
XGBoost Regression	**R2 Squared**	0.20377	0.1998	0.03632	0.0305527
	RMSE	8.7433	8.764	9.7004	9.59539
	PPD	-4.910	-5.478	-7.78899	-7.7157
StackingCV Regression	**R2 Squared**	0.224	0.221	0.03731	0..02768
	RMSE	8.627	8.6444	9.6954	9.7319
	PPD	-4.9652	-4.487	-7.7425	-7.227

Findings:

- The results of linear regression are pretty poor considering the low r squared value and less rmse value.
- The result of XGBoost model is better compared to linear regression but not so good performing. And is still very poor values of r squared and rmse.
- Stacking model has given best results as compared all. No results are good and acceptable, but out of three, stacking model is opted for deployment.

The low performance can be taken as a strong indicator that the user information provided to the telecom company and the real user are two different individuals. So in the context of this project this disparity can be considered as a real world result.

10.2 Gender Prediction

Scenario 1

Logistic Regression

```
Confusion matrix: [1140,  848],
                   [1297, 2502]
Accuracy: 62.93%
True Positive: 2502
True Negative: 1140
False Positive: 848
False Negative: 1297
Sensitivity: 65.86%
Specificity: 57.34%
Male predictive value: 74.69%
Female predictive value: 46.78%
Precision: 74.69%
Recall: 65.86%
F1 Score: 70.00%
KS is 26.400000000000002% at decile 6
```

With simple logistice regression model we got the male prediction value of ~75% and female prediction value of ~47% with an accuracy of ~63%

XGBoost Classification

```
Confusion matrix [[ 468 1520]
                  [ 332 3467]]
Accuracy: 68.00%
True Positive: 3467
True Negative: 468
False Positive: 1520
False Negative: 332
Sensitivity: 91.26%
Specificity: 23.54%
Male predictive value: 69.52%
Female predictive value: 58.50%
Precision: 69.52%
Recall: 91.26%
```

F1 Score: 78.92%

In logistic regression we got a male prediction score of 73% and female prediction score of 47%. But with xgboost we got a male prediction score of 70% and female prediction score of 58%.

Even though male prediction score is reduced by a small ammount, female prediction score is increased drastically. And accuracy and f1 score is also increased. So, out of these two. I prefer xgboost.

StackingCV Classifier

```
Confusion Matrix [[ 418 1570]
                  [ 284 3515]]
```

Accuracy: 67.96%
True Positive: 3515
True Negative: 418
False Positive: 1570
False Negative: 284
Sensitivity: 92.52%
Specificity: 21.03%
Male predictive value: 69.12%
Female predictive value: 59.54%
Precision: 69.12%
Recall: 92.52%
F1 Score: 79.13%

Stacking is more preferred since we have got ~69% male predictive rate and ~60% female predictive rate which is higher than logistic regression and xgboost alone.

The Accuracy is also ~68% with F1 score of ~79%

Scenario 2

Logistic Regression

```
Confusion Matrix: [[3211,  527],
                   [5164, 1366]])
```

Accuracy: 44.58%
True Positive: 1366

```
True Negative: 3211
False Positive: 527
False Negative: 5164
Sensitivity: 20.92%
Specificity: 85.90%
Male predictive value: 72.16%
Female predictive value: 38.34%
Precision: 72.16%
Recall: 20.92%
F1 Score: 32.43%
```

With simple logistic regression model we got the male prediction value of ~72% and female prediction value of ~38% with an accuracy of ~45%

XGBoost Classifier

```
Confusion Matrix: [[  83 3655]
                   [  59 6471]]
Accuracy: 63.83%
True Positive: 6471
True Negative: 83
False Positive: 3655
False Negative: 59
Sensitivity: 99.10%
Specificity: 2.22%
Male predictive value: 63.90%
Female predictive value: 58.45%
Precision: 63.90%
Recall: 99.10%
F1 Score: 77.70%
```

In logistic regression we got a male prediction score of 72% and female prediction score of 38%. With xgboost we gor male prediction rate as 64% and female prediction rate as 58%.

Even though male prediction score is reduced by a small amount, female prediction score is increased drastically. And accuracy and f1 score is also increased. So, out of these two. I prefer xgboost.

StackingCVClassifier

```
Confusion Matrix [[1207 2531]
                  [1431 5099]]
Accuracy: 61.41%
True Positive: 5099
True Negative: 1207
False Positive: 2531
False Negative: 1431
Sensitivity: 78.09%
Specificity: 32.29%
Male predictive value: 66.83%
Female predictive value: 45.75%
Precision: 66.83%
Recall: 78.09%
F1 Score: 72.02%
```

I prefer stacking, we have got ~67% male predictive rate and ~46% female predictive rate. The Accuracy is also ~61% with F1 score of ~72%

11. Contribution of each Student

Both the students contributed equally towards the project

Krishna Ramadas	Harindranath Reddy
Basic EDA , Geospatial visualization	EDA , EDA Questions
Feature engineering, Visualization of Clusters	Feature engineering and DBSCAN modeling
Age prediction Model (S1 and S2)	Gender Prediction model (S1 andS2)
Model Deployment Execution and Documentation	Model Deployment code and other necessary files preparation