Task1_Modelbuilding

1. Top five rows of the data set at the beginning of the analysis

In [6]:
```
app_events.head(5)
```
Out[6]:

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

In [19]:
```
app_events_meta_data.head(5)
```
Out[19]:

	app_id	label_id	category
1	7324884708820027918	251	Finance
2	-4494216993218550286	251	Finance
3	6058196446775239644	406	unknown
4	6058196446775239644	407	DS_P2P net loan
5	8694625920731541625	406	unknown

In [11]:
```
train_event_data.head(5)
```
Out[11]:

	device_id	gender	age	group_train	event_id	datetimestamp	latitude	longitude
0	-7548291590301750000	M	33	M32+	2369465.00	2016-05-03 15:55:35	33.98	116.79
1	-7548291590301750000	M	33	M32+	1080869.00	2016-05-03 06:07:16	33.98	116.79
2	-7548291590301750000	M	33	M32+	1079338.00	2016-05-04 03:28:02	33.98	116.79
3	-7548291590301750000	M	33	M32+	1078881.00	2016-05-04 02:53:08	33.98	116.79
4	-7548291590301750000	M	33	M32+	1068711.00	2016-05-03 15:59:35	33.98	116.79

In [22]:
```
train_mobile_brand.head(5)
```
Out[22]:

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

2. List of data cleaning techniques applied such as missing value treatment

a. We used missing and null value analysis to check the health of the data

```
In [32]: new_train_event_data.isnull().sum()

Out[32]: device_id          0
         gender             0
         age                0
         group_train        0
         event_id        7700
         datetimestamp   7700
         latitude        7700
         longitude       7700
         Event_status       0
         dtype: int64
```

```
In [33]: app_events_meta_data.isnull().sum()

Out[33]: app_id      0
         label_id    0
         category    0
         dtype: int64
```

```
In [34]: train_mobile_brand.isnull().sum()

Out[34]: device_id       0
         gender          0
         age             0
         group_train     0
         phone_brand     0
         device_model    0
         dtype: int64
```


```
In [32]: new_train_event_data.isnull().sum()

Out[32]: device_id          0
         gender             0
         age                0
         group_train        0
         event_id        7700
         datetimestamp   7700
         latitude        7700
         longitude       7700
         Event_status       0
         dtype: int64
```


will not be dropping these value because these are those record which have no event data but have device data . which will be used later for EDA purpose

b. Converted “datetimestamp” column to “to_datetime” datatype

The dataframe train_event_data has columns to be converted to right datatype

```
In [23]: ## Column datetimestamp from string to timestamp
         new_train_event_data['datetimestamp'] = pd.to_datetime(new_train_event_data['datetimestamp'])
```

```
In [24]: new_train_event_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 190040 entries, 1261416 to 846458
Data columns (total 9 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   device_id      190040 non-null  int64
 1   gender         190040 non-null  object
 2   age            190040 non-null  int64
 3   group_train    190040 non-null  object
 4   event_id       182340 non-null  float64
 5   datetimestamp  182340 non-null  datetime64[ns]
 6   latitude       182340 non-null  float64
 7   longitude      182340 non-null  float64
 8   Event_status   190040 non-null  int32
dtypes: datetime64[ns](1), float64(3), int32(1), int64(2), object(2)
memory usage: 13.8+ MB
```

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

a. extraction of day, hour, day_ name and month features from dateimestamp column

b. Age group column based of age column

```
In [35]: new_train_event_data['day'] = new_train_event_data['datetimestamp'].dt.weekday
         new_train_event_data['hour'] = new_train_event_data['datetimestamp'].dt.hour
         new_train_event_data['month'] = new_train_event_data['datetimestamp'].dt.month
         new_train_event_data['day_name'] = new_train_event_data['datetimestamp'].dt.day_name()
```

```
In [36]: def func(x):
             if x < 25:
                 return "0-24"
             elif x < 33:
                 return "25-32"
             elif x < 46:
                 return "33-45"
             else:
                 return '45+'
```

```
In [37]: new_train_event_data['age_group'] = new_train_event_data['age'].apply(func)
```

c. count_events_perday feature based on the 'device_id','day','month' columns

d. median of latitude and longitude for normalizing line so that we can plot geo spatial visualizations accurately.

Some device IDs have multiple events in a day or over a period of days.

Find the average number of events – find the percentage of time the mobile phone was active by calculating the number of events for a device ID.

```
In [91]: final_event_data['count_events_perday'] = final_event_data.groupby(['device_id','day','month']).event_id.transform('count')
```

```
In [92]: final_event_data['lat_median'] = final_event_data.groupby(['device_id','day','month']).latitude.transform('median')
```

```
In [93]: final_event_data['long_median'] = final_event_data.groupby(['device_id','day','month']).longitude.transform('median')
```

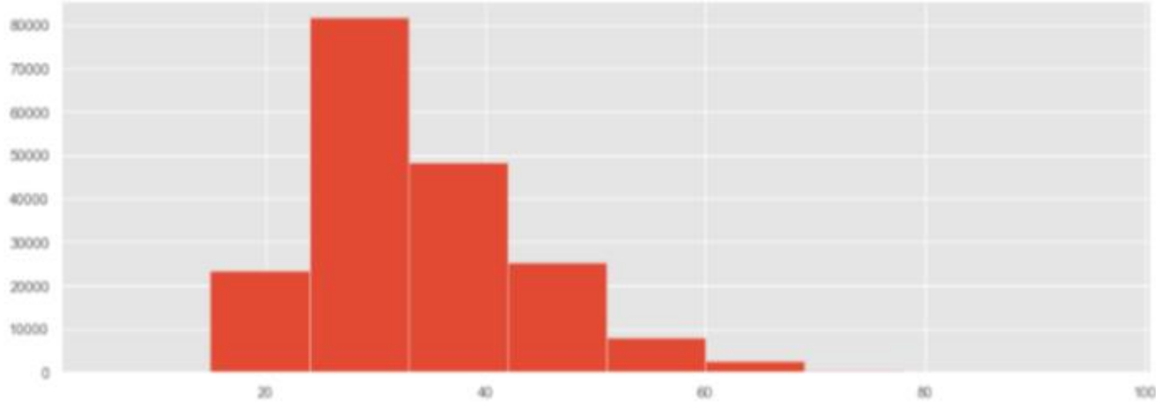
4. **Outputs to the various EDA and Visualisation codes along with the corresponding results and the insights gathered from each EDA and visualisation**

a. Basic Visualization for checking distribution of features for Age & Gender

Univariate Analysis

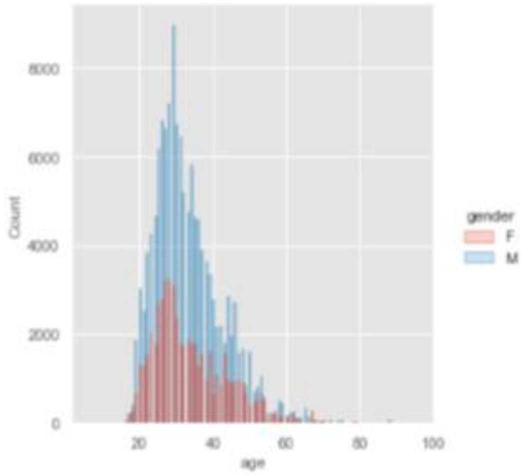
1. Age

In [41]:
```
plt.figure(figsize=(15,5))
plt.hist(new_train_event_data['age'], bins=10)
plt.show()
```



In [42]:
```
plt.figure(figsize=(15,5))
sns.displot(new_train_event_data, x="age", hue="gender", element="step")
plt.show()
```

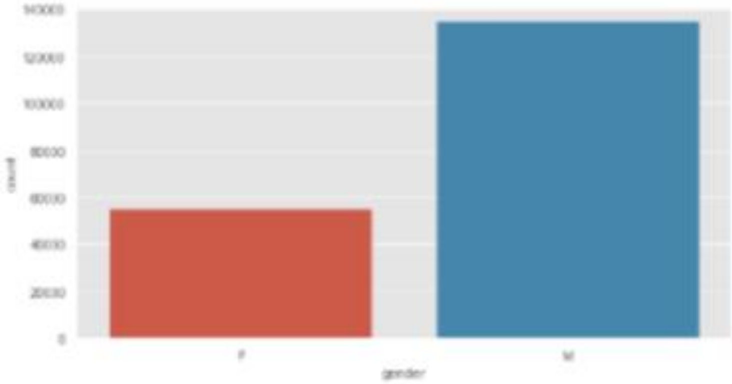
```
<Figure size 1080x360 with 0 Axes>
```



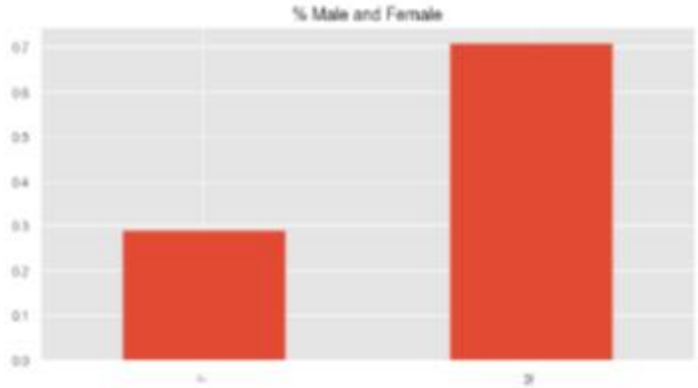
Analysis: Most of data falls between the age 25 - 45

2. Gender

In [43]:
```
plt.figure(figsize=(10,5))
sns.countplot(x ='gender', data = new_train_event_data)
plt.show()
```

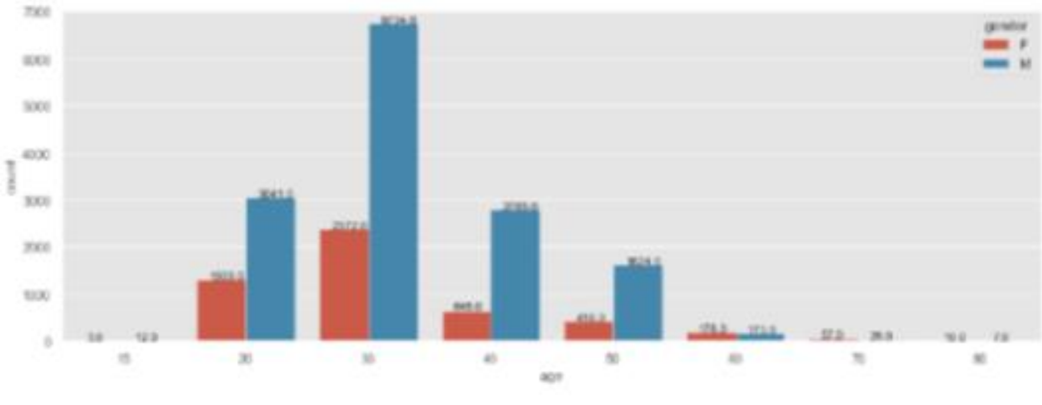


In [44]:
```
plt.figure(figsize=(10,5))
new_train_event_data.gender.value_counts(normalize=True,ascending=True).plot.bar(title="% Male and Female")
plt.show()
```



In [45]:
```
print("Unique age numbers    : ",new_train_event_data.age.unique())
print("\n train group with age : ",new_train_event_data.group_train.unique())
print("\n")
plt.figure(figsize=(15,5))
ax = sns.countplot(data = new_train_event_data, x="age", hue='gender', order = [15,20,30,40,50,60,70,80])
for p in ax.patches:
   ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.10, p.get_height()+0.01))
plt.show()
```

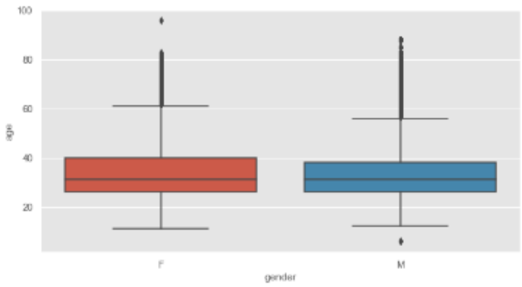
```
Unique age numbers    :  [40 22 34 27 33 32 28 30 29 39 37 20 25 42 38 23 35 60 47 43 26 75 21 24
 48 49 41 72 64 19 45 50 31 18 36 46 51 65 56 44 77 52 58 53 17 68 54 96
 63 62 16 11 59 66 55 71 61 67 57 76 14 13 74 83 69 73 15 12 80 70  6 79
 81 78 85 82 88]

 train group with age :  ['F32+' 'M0-24' 'M25-32' 'M32+' 'F25-32' 'F0-24']
```



Bivariate Analysis

Boxplot

In [46]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x='gender', y='age', data=new_train_event_data)
plt.show()
```



Analysis:

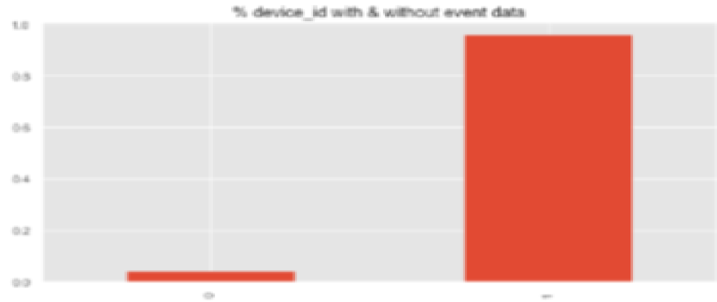
1. The above boxplots show that most of the people are under the age between 25-40.

b. Trend Data analysis and Visualisation

EDA for Trend data

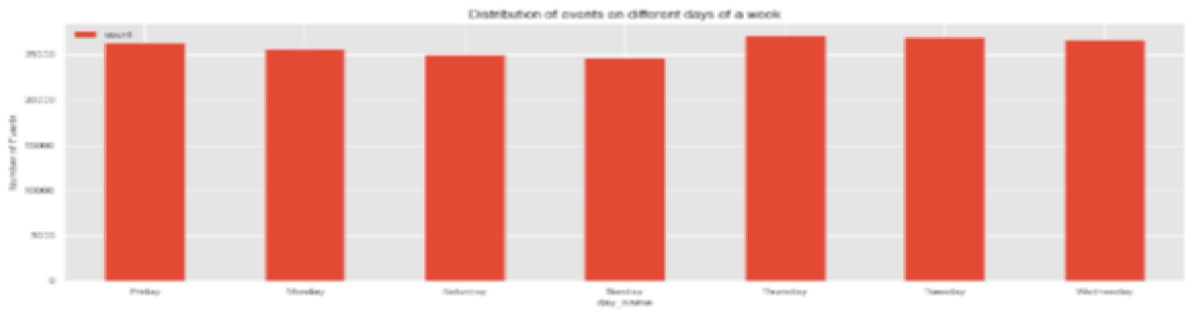
percentage of device_ids with and without event data

In [47]:
```
plt.figure(figsize=(10,5))
new_train_event_data.Event_status.value_counts(normalize=True,ascending=True).plot.bar(title="% device_id with & without event d
plt.show()
```



Graph representing the distribution of events on different days of a week

In [48]:
```
new_train_event_data.groupby('day_name').size().reset_index(name='count').plot(kind='bar', x='day_name',title='Distribution of e
plt.xticks(rotation=0)
plt.show()
```



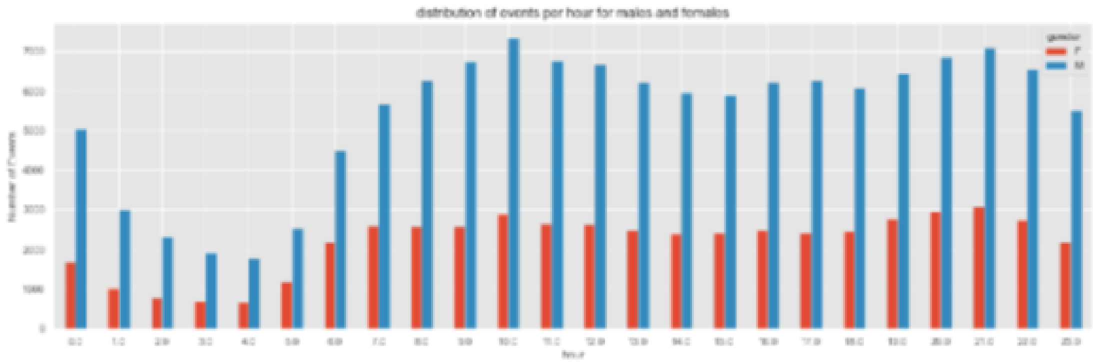
```
In [49]: start_date = '2016-05-01 00:00:00'
         end_date = '2016-05-07 23:59:00'
         mask = (new_train_event_data['datetimestamp'] >= start_date) & (new_train_event_data['datetimestamp'] <= end_date)
         one_week_df = new_train_event_data.loc[mask]
```

Graph representing the distribution of events per hour (for one-week data)

```
In [50]: one_week_df.groupby('hour').size().reset_index(name='count').plot(kind='bar', title='distribution of events per hour(one week da
         plt.xticks(rotation=0)
         plt.show()
```

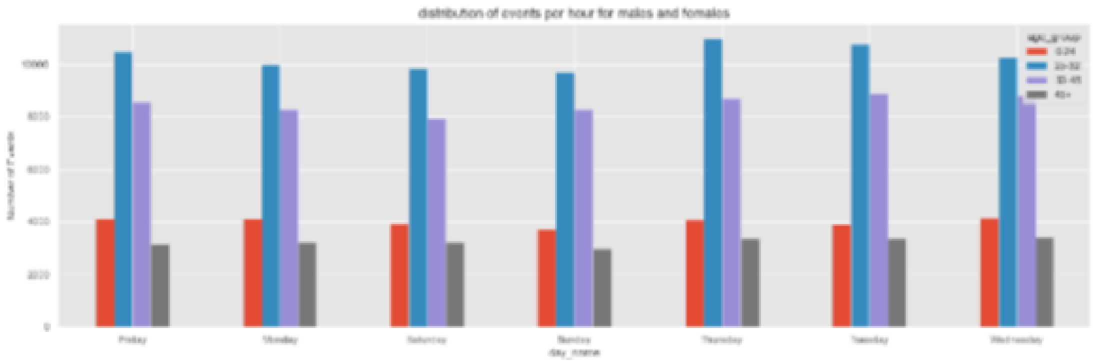
The difference in the distribution of events per hour for males and females (Show the difference using an appropriate chart for one week's data)

```
In [51]: one_week_df.groupby(['hour','gender']).size().unstack().plot(kind='bar', title='distribution of events per hour for males and fe
         plt.xticks(rotation=0)
         plt.show()
```



distribution of events for different age groups over different days of week? (Consider the age groups as 0–24, 25–32, 33–45, 46+]

```
In [52]: one_week_df.groupby(['day_name','age_group']).size().unstack().plot(kind='bar', title='distribution of events per hour for males
         plt.xticks(rotation=0)
         plt.show()
```



c. Phone brands and their distribution among app_id's, gender and age

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

```
In [55]: plt.figure(figsize=(20,15))
         counts.plot(kind='bar', stacked=True)
         plt.legend(bbox_to_anchor=(1.31,1))
         plt.show()
```

<Figure size 1440x1080 with 0 Axes>

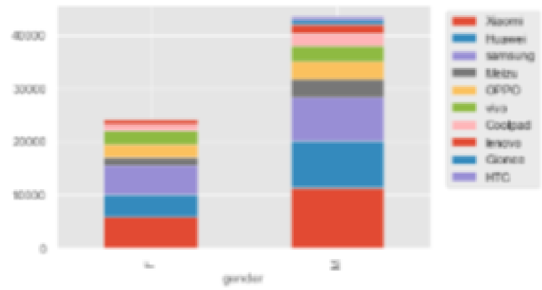
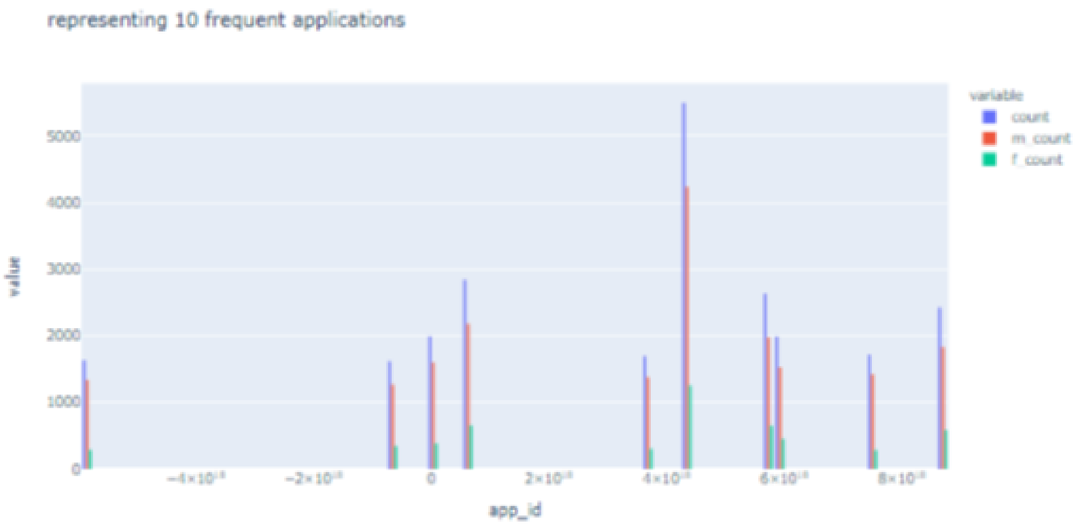


Chart representing 10 frequent applications and the corresponding percentage of male and female consumers

```
In [85]: fig = px.bar(
             data_frame = df_plot,
             x = "app_id",
             y = ['count', 'm_count', 'f_count'],
             opacity = 0.9,
             orientation = "v",
             barmode = 'group',
             title='representing 10 frequent applications',
         )
         fig.show()
```



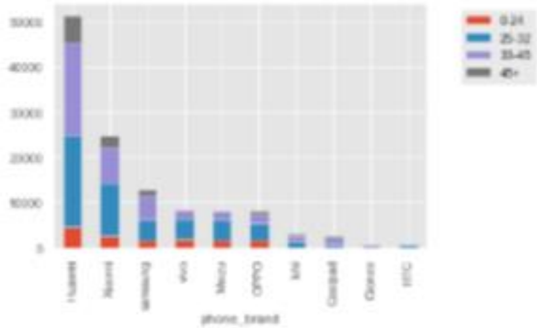
Top 10 mobile phone brands by age groups (Consider the age groups as 0–24, 25–32, 33–45, 46+.) with Event_data

In [86]:
```
## Event_data
popularity  = final_event_data.groupby(['phone_brand','age_group']).phone_brand.count().nlargest(30).unstack()
print(popularity)
```

```
age_group        0-24     25-32     33-45      45+
phone_brand
Huawei        4553.00  20515.00  20179.00  6135.00
Xiaomi        2561.00  11784.00   7828.00  2586.00
samsung       1641.00   4485.00   5445.00  1215.00
vivo          1947.00   4542.00   1988.00      NaN
Meizu         1712.00   4512.00   1874.00      NaN
OPPO          1618.00   3806.00   2839.00   596.00
lshi              NaN   1435.00   1313.00   514.00
Coolpad           NaN    818.00   1313.00   606.00
Gionee            NaN       NaN    725.00      NaN
HTC               NaN    650.00       NaN      NaN
```

In [87]:
```
plt.figure(figsize=(20,15))
popularity.plot(kind='bar', stacked=True)
plt.legend(bbox_to_anchor=(1.31,1))
plt.show()
```

```
<Figure size 1440x1080 with 0 Axes>
```



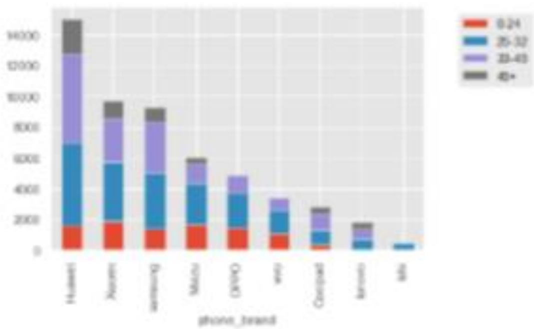
Top 10 mobile phone brands by age groups (Consider the age groups as 0–24, 25–32, 33–45, 46+.) with non_Event_data

In [88]:
```
## Event_data
popularity  = final_non_event_data.groupby(['phone_brand','age_group']).phone_brand.count().nlargest(30).unstack()
print(popularity)
```

```
age_group       0-24    25-32    33-45      45+
phone_brand
Huawei       1603.00  5324.00  5824.00  2290.00
Xiaomi       1854.00  3861.00  2840.00  1199.00
samsung      1358.00  3604.00  3354.00   969.00
Meizu        1665.00  2703.00  1225.00   391.00
OPPO         1431.00  2268.00  1163.00      NaN
vivo         1051.00  1573.00   755.00      NaN
Coolpad       365.00   930.00  1109.00   419.00
lenovo           NaN   733.00   622.00   465.00
lshi             NaN   430.00      NaN      NaN
```

In [89]:
```
plt.figure(figsize=(20,15))
popularity.plot(kind='bar', stacked=True)
plt.legend(bbox_to_anchor=(1.31,1))
plt.show()
```

```
<Figure size 1440x1080 with 0 Axes>
```



5. **<u>Geospatial visualisations along with the insights gathered from this visualisation</u>**

The median latitude and longitude calculated to see the distribution of events, Gender and age across the globe

Advanced visualization

Note will be using 80k data points as due to local computation strength we have taken only 10% data

Plot the visualization plot for a sample of 80k data points

In [100]:
```
#visualization plot for a sample of 80k data points.
temp = final_event_data.sample(n=80000)
plt.figure(1, figsize=(12, 6))
m1 = Basemap(projection='merc',llcrnrlat=-60,urcrnrlat=65,llcrnrlon=-180,urcrnrlon=180,lat_ts=0,resolution='c')
m1.fillcontinents(color='#191919', lake_color='#000000')
m1.drawmapboundary(fill_color='#000000')
m1.drawcountries(linewidth=0.1, color='w')

mxy = m1(temp['long_median'].tolist(), temp['lat_median'].tolist())
m1.scatter(mxy[0], mxy[1], s=3, c='#1292db', lw=0, alpha=1, zorder=5)
plt.title('Overall View of Events')
plt.show()

del temp
del m1
```

Overall View of Events

Compare the event visualization plots based on the users' gender information. (This can be done on the sample of 80k data points.)

```
In [101]: # visualization plots based on the gender information with sample of 80k data points

colors = ['tab:red', 'tab:green']
#colors = ListedColormap(['red', 'green'])

gender = ['F', 'M']

temp = final_event_data.sample(n=80000)
plt.figure(1, figsize=(12, 6))
m1 = Basemap(projection='merc',llcrnrlat=-60,urcrnrlat=65,llcrnrlon=-180,urcrnrlon=180,lat_ts=0,resoluti
m1.fillcontinents(color='#191919', lake_color='#000000')
m1.drawmapboundary(fill_color='#000000')
m1.drawcountries(linewidth=0.1, color='w')
i=0
for g in gender:
    temp1 = temp[temp['gender'] == g]
    mxy = m1(temp1['long_median'].tolist(), temp1['lat_median'].tolist())
    scatter = m1.scatter(mxy[0], mxy[1], s=3, c=colors[i] , lw=0, alpha=1, zorder=5, label= g)
    i+=1


plt.title('Gender based view of Events')
plt.legend(loc='lower left');

plt.show()
del temp
del temp1
del m1
```

Gender based view of Events

Compare the event visualization plots based on the following age groups

In [102]:
```
# visualization plot based on the following age groups:0-24 , 25-32 and 32+ for 80k data points

temp = final_event_data.sample(n=80000)
temp['age_group'] = pd.cut(x = temp['age'], bins=[0,24,31,100], \
                           labels = ['0-24', '25-32', '32+'])

colors = ['tab:blue', 'tab:orange', 'tab:green']
labels = [ '0-24','25-32', '32+']

plt.figure(1, figsize=(12, 6))
m1 = Basemap(projection='merc',llcrnrlat=-60,urcrnrlat=65,llcrnrlon=-180,urcrnrlon=180,lat_ts=0, resolution='c')
m1.fillcontinents(color='#191919', lake_color='#000000')
m1.drawmapboundary(fill_color='#000000')
m1.drawcountries(linewidth=0.1, color='w')
i=0
for label in labels:
    temp1 = temp[temp['age_group'] == label]
    mxy = m1(temp['long_median'].tolist(), temp['lat_median'].tolist())
    scatter = m1.scatter(mxy[0], mxy[1], s=1, c= colors[i], alpha=1, zorder=1, label= label)
    i+=1

plt.title('Age Group based view of Events')

plt.legend(loc='lower left');

plt.show()
del temp
del temp1
del m1
```

Age Group based view of Events

6. <u>Results interpreting the clusters formed as part of DBSCAN Clustering and how the cluster information is being used</u>

Around 121 cluster were created based on the median latitude and longitude using haversine metric and ball_tree algorithm

DBSCAN clustering

In [103]:
```
## Clustering based on lat_mean and long_mean
temp = final_event_data[final_event_data['lat_median'] != 0]
coords = np.asmatrix(temp[['lat_median', 'long_median']].fillna(0))
kms_per_radian = 6371.0088
epsilon = 15 / kms_per_radian
db = DBSCAN(eps= epsilon, min_samples=125, algorithm='ball_tree', metric='haversine',n_jobs=-1).fit(np.radians(coords))
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))
temp['cluster'] = cluster_labels
temp.head(2)
```

```
Number of clusters: 119
```

Out[103]:

group_train	phone_brand	device_model	age_group	app_id	is_active	label_id	category	count_events_perday	lat_median	long_median	cluster
M25-32	Xiaomi	MI 4	25-32	-5305696816021977482	0.00	178	music	284	25.10	99.16	0
M25-32	Xiaomi	MI 4	25-32	9112463114311278255	0.00	179	video	284	25.10	99.16	0

7. A brief summary of any additional subtask that was performed and may have improved the data cleaning and feature generation step

a. Scaling of the data for normalization

```
In [113]: from sklearn.preprocessing import MinMaxScaler
          scaler = MinMaxScaler()
```

```
In [114]: # Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
          num_vars = ['hour', 'lat_median', 'long_median', 'cluster']

          final_event_data[num_vars] = scaler.fit_transform(final_event_data[num_vars])
```

b. Is_installed column removal as it was not giving any insight

```
memory usage: 41.0+ MB

In [68]: final_event_data['is_installed'].value_counts().sort_values(ascending=False)

Out[68]: 1.00    128659
         Name: is_installed, dtype: int64
```

is installed always have 1 and no 0, it's not adding any value hence dropping.

8. All the data preparation steps that were used before applying the ML algorithm

a. Merging of all datasets

```
In [58]: merged_events = event_data.merge(train_mobile_brand, on='device_id', how='left')
```

```
dtype: float64

In [61]: merged_app = new_app_events.merge(app_events_meta_data, on='app_id', how='left')
```

```
In [66]: final_event_data = merged_events.merge(merged_app, on='event_id', how='left')
```

```
In [77]: final_non_event_data = non_event_data.merge(train_mobile_brand, on='device_id', how='left')
```

b. Splitting data to event and non_event data (scenario-1 and scenario-2)

Event_data & Non_event_data

```
In [56]: event_data = new_train_event_data[~(((new_train_event_data.longitude == 0)&(new_train_event_data.latitude ==0))|((new_train_even
```

```
In [57]: non_event_data = new_train_event_data[(((new_train_event_data.longitude == 0)&(new_train_event_data.latitude ==0))|((new_train_e
```

9. 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.)

a. DBScan

DBSCAN clustering

In [103]:
```
## Clustering based on lat_mean and long_mean
temp = final_event_data[final_event_data['lat_median'] != 0]
coords = np.asmatrix(temp[['lat_median', 'long_median']].fillna(0))
kms_per_radian = 6371.0088
epsilon = 15 / kms_per_radian
db = DBSCAN(eps= epsilon, min_samples=125, algorithm='ball_tree', metric='haversine',n_jobs=-1).fit(np.radians(coords))
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))
temp['cluster'] = cluster_labels
temp.head(2)
```

```
Number of clusters: 119
```

Out[103]:

group_train	phone_brand	device_model	age_group	app_id	is_active	label_id	category	count_events_perday	lat_median	long_median	cluster
M25-32	Xiaomi	MI 4	25-32	-5305696816021977482	0.00	178	music	284	25.10	99.16	0
M25-32	Xiaomi	MI 4	25-32	9112463114311278255	0.00	179	video	284	25.10	99.16	0

b. Train- Test split based on device_id

In [125]:
```
df_event = final_event_data.merge(train_test_split, on=['device_id'], how='inner')
```

In [130]:
```
train_data = df_event[(df_event.train_test_flag == "train")]
```

In [134]:
```
test_data = df_event[(df_event.train_test_flag == "test")]
```

c. DictVectorizer is used to convert column categorical data to sparse matrix data using pipeline. All the below screenshots of each model will show this.
d. Scenario-1 Event Data Modelling (Gender Prediction – Logistic, RandomForestClassifier and XGBoostClassifer) & (Age Prediction – Linear, RandomForestRegressor and XGBoostRegressor)

We will convert the data to dictionary format

Then, make use DicVectorisation method for Sparsing categorical data in pipe line itself of every model so we need not have to fit and fransform separately

In [147]:
```
vec = DictVectorizer(sparse=True)
```

In [148]:
```
X_dict = input_train_evet_data.to_dict(orient='records')
```

In [149]:
```
Xtest_dict = input_test_evet_data.to_dict(orient='records')
```

Logistic and RandomForest model for Gender prediction

In [150]:
```
lr_gender= LogisticRegression(penalty='l1',solver='saga',random_state=0)
```

In [151]:
```
lr_gender_pipe = make_pipeline(vec, lr_gender)
lr_gender_pipe.fit(X_dict, ytrain_gender)
```

Out[151]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('logisticregression',
                 LogisticRegression(penalty='l1', random_state=0,
                                    solver='saga'))])
```

Train accuracy and Test accuracy metrices

In [152]:
```
y_train_pred = lr_gender_pipe.predict(X_dict)
```

In [153]:
```
print("train accuracy: ",metrics.accuracy_score(ytrain_gender, y_train_pred))
```

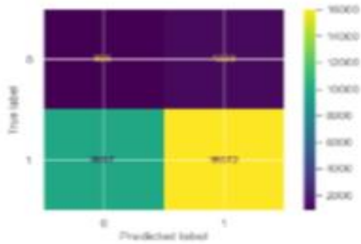
```
train accuracy:  0.6104581000193088
```

Logistic Regression Gender Model Building & Evaluation

Accuracy & Confusion matrix

In [154]:
```
y_pred = lr_gender_pipe.predict(Xtest_dict)
print("test accuracy",metrics.accuracy_score(ytest_gender, y_pred))
# Plot and print confusion matrix
cnf_matrix = metrics.confusion_matrix(ytest_gender, y_pred)
print(cnf_matrix)
metrics.plot_confusion_matrix(lr_gender_pipe, Xtest_dict, ytest_gender)
plt.show()
```

```
test accuracy 0.6011092150170648
[[  836  1223]
 [ 9997 16072]]
```



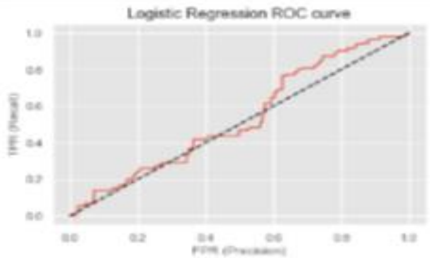
F1 Score, Precision, Recall

In [155]:
```
print(classification_report(ytest_gender, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.08      0.41      0.13      2059
           1       0.93      0.62      0.74     26069

    accuracy                           0.60     28128
   macro avg       0.50      0.51      0.44     28128
weighted avg       0.87      0.60      0.70     28128
```

ROC curve and AUC

In [156]:
```
lr_probability =lr_gender_pipe.predict_proba(Xtest_dict)[:,1]
fpr, tpr,_ = roc_curve(ytest_gender, lr_probability,)
plt.title('Logistic Regression ROC curve')
plt.xlabel('FPR (Precision)')
plt.ylabel('TPR (Recall)')
plt.plot(fpr,tpr)
plt.plot((0,1), ls='dashed',color='black')
plt.show()
```



Random Forest Gender Model Building & Evaluation

```
In [157]: RF_gender = RandomForestClassifier(random_state=42, n_estimators=10,max_depth=4,n_jobs=-1)
```

```
In [158]: RF_gender_pipe = make_pipeline(vec, RF_gender)
```

```
In [159]: RF_gender_pipe.fit(X_dict, ytrain_gender)
```

```
Out[159]: Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                          ('randomforestclassifier',
                           RandomForestClassifier(max_depth=4, n_estimators=10, n_jobs=-1,
                                                  random_state=42))])
```

Random Forest Train accuracy and Test accuracy metrices

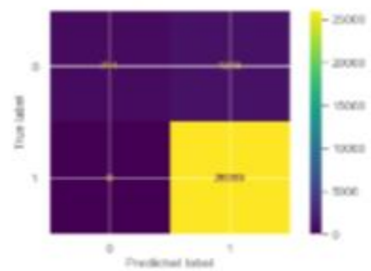
```
In [160]: y_train_pred = RF_gender_pipe.predict(X_dict)
```

```
In [161]: print("train accuracy: ",metrics.accuracy_score(ytrain_gender, y_train_pred))
```

```
train accuracy:  0.9249372465726974
```

```
In [162]: y_pred = RF_gender_pipe.predict(Xtest_dict)
          print("test accuracy: ", metrics.accuracy_score(ytest_gender, y_pred))
          # Plot and print confusion matrix
          cnf_matrix = metrics.confusion_matrix(ytest_gender, y_pred)
          print(cnf_matrix)
          metrics.plot_confusion_matrix(RF_gender_pipe, Xtest_dict, ytest_gender)
          plt.show()
```

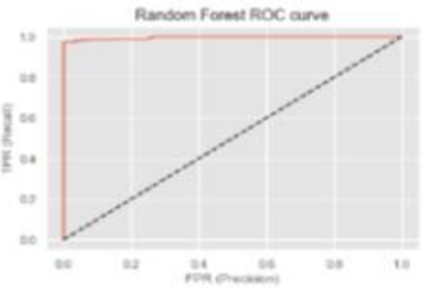
```
test accuracy:  0.9532494311717861
[[  744  1315]
 [    0 26069]]
```




```
In [163]: print(classification_report(ytest_gender, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.36	0.53	2059
1	0.95	1.00	0.98	26069
accuracy			0.95	28128
macro avg	0.98	0.68	0.75	28128
weighted avg	0.96	0.95	0.94	28128

```
In [164]: lr_probability =RF_gender_pipe.predict_proba(Xtest_dict)[:,1]
          fpr, tpr,_ = roc_curve(ytest_gender, lr_probability,)
          plt.title('Random Forest ROC curve')
          plt.xlabel('FPR (Precision)')
          plt.ylabel('TPR (Recall)')
          plt.plot(fpr,tpr)
          plt.plot((0,1), ls='dashed',color='black')
          plt.show()
```



XGB Preparation for meta leaner

Only using the parameter that are possible to withstand the local system computation

In [165]:
```
params = {
    'min_child_weight': [1, 5],
    'gamma': [0.5, 1],
    'subsample': [0.6, 0.8],
    'max_depth': [3, 4],
    'n_estimators': [5,6],
    'learning_rate': [0.1, 0.2]
 }
```

In [166]:
```
xgbClass = XGBClassifier()
```

In [167]:
```
XGB = GridSearchCV(xgbClass, params, n_jobs =-1,return_train_score=True, verbose=True)
```

In [168]:
```
stacking_gender = StackingCVClassifier(classifiers=[lr_gender, RF_gender], meta_classifier=XGB, use_probas=True, cv=3)
```

In [169]:
```
# Fit on train data
stacking_gender_pipe = make_pipeline(vec, stacking_gender)
stacking_gender_pipe.fit(X_dict, ytrain_gender)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
```

Out[169]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('stackingcvclassifier',
                 StackingCVClassifier(classifiers=[LogisticRegression(penalty='l1',
                                                                      random_state=0,
                                                                      solver='saga'),
                                                   RandomForestClassifier(max_depth=4,
                                                                          n_estimators=10,
                                                                          n_jobs=-1,
                                                                          random_state=42)],
                                      cv=3,
                                      meta_classifier=GridSearchCV(estimator=XGBClassifier(base_score=None,
                                                                                           booster=None,
                                                                                           callbacks=No...
                                                                                           min_child_weight=None,
                                                                                           missing=nan,
                                                                                           monotone_constraints=None,
                                                                                           n_estimators=100,
                                                                                           n_jobs=None,
                                                                                           num_parallel_tree=None,
                                                                                           predictor=None,
                                                                                           random_state=None, ...),
                                                                   n_jobs=-1,
                                                                   param_grid={'gamma': [0.5,
                                                                                         1],
                                                                               'learning_rate': [0.1,
                                                                                                 0.2],
                                                                               'max_depth': [3,
                                                                                             4],
                                                                               'min_child_weight': [1,
                                                                                                    5],
                                                                               'n_estimators': [5,
                                                                                                6],
                                                                               'subsample': [0.6,
                                                                                             0.8]},
                                                                   return_train_score=True,
                                                                   verbose=True),
                                      use_probas=True))])
```

Train and Test Accuracy scores metrices

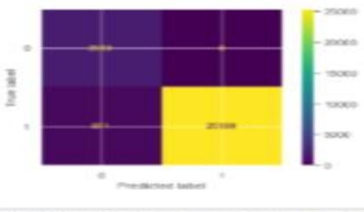
In [170]:
```
y_train_pred = stacking_gender_pipe.predict(X_dict)
```

In [171]:
```
print("train accuracy: ",metrics.accuracy_score(ytrain_gender, y_train_pred))
```

```
train accuracy:  0.9850477891484842
```

In [172]:
```
y_pred = stacking_gender_pipe.predict(Xtest_dict)
print("test accuracy: ",metrics.accuracy_score(ytest_gender, y_pred))
# Plot and print confusion matrix
cnf_matrix = metrics.confusion_matrix(ytest_gender, y_pred)
print(cnf_matrix)
metrics.plot_confusion_matrix(stacking_gender_pipe, Xtest_dict, ytest_gender)
plt.show()
```

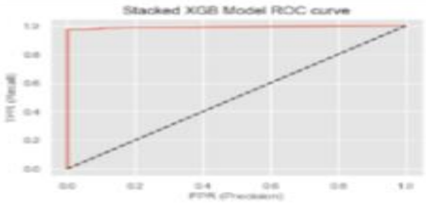
```
test accuracy:  0.9761447667804323
[[ 2059     0]
 [  671 25398]]
```



In [173]:
```
print(classification_report(ytest_gender, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      2059
           1       1.00      0.97      0.99     26069

    accuracy                           0.98     28128
   macro avg       0.88      0.99      0.92     28128
weighted avg       0.98      0.98      0.98     28128
```

In [174]:
```
lr_probability =stacking_gender_pipe.predict_proba(Xtest_dict)[:,1]
fpr, tpr,_ = roc_curve(ytest_gender, lr_probability,)
plt.title('Stacked XGB Model ROC curve')
plt.xlabel('FPR (Precision)')
plt.ylabel('TPR (Recall)')
plt.plot(fpr,tpr)
plt.plot((0,1), ls='dashed',color='black')
plt.show()
```



Scenario-1 Linear Regression And RandomForest model for age prediction

Linear Regression Age Model Building & Evaluation

In [175]:
```
lr_age= LinearRegression()
```

In [176]:
```
# Fit on train data
lr_age_pipe = make_pipeline(vec, lr_age)
lr_age_pipe.fit(X_dict, ytrain_age)
```

Out[176]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('linearregression', LinearRegression())])
```

In [177]:
```
ypred = lr_age_pipe.predict(Xtest_dict)
mse = mean_squared_error(ytest_age, ypred)
r_squared = r2_score(ytest_age, ypred)
rsme = math.sqrt(mse)
```

In [178]:
```
print('Mean_Squared_Error :' ,mse)
print('Root Squared mean_Error :' ,rsme)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 288.79967556956956
Root Squared mean_Error : 16.99410708362077
r_square_value : -2.1738318858362673
```

Random forest Regression Age Model Building & Evaluation

In [179]:
```
RF_age = RandomForestRegressor(random_state=42,n_estimators = 100, min_samples_split = 15, min_samples_leaf= 4, max_features = '
```

In [180]:
```
# Fit on train data
RF_age_pipe = make_pipeline(vec, RF_age)
RF_age_pipe.fit(X_dict, ytrain_age)
```

Out[180]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('randomforestregressor',
                 RandomForestRegressor(max_depth=150, max_features='sqrt',
                                       min_samples_leaf=4, min_samples_split=15,
                                       random_state=42))])
```

In [181]:
```
ypred = RF_age_pipe.predict(Xtest_dict)
mse = mean_squared_error(ytest_age, ypred)
r_squared = r2_score(ytest_age, ypred)
rsme = math.sqrt(mse)
```

In [182]:
```
print('Mean_Squared_Error :' ,mse)
print('Root Squared mean_Error :' ,rsme)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 177.99699800358486
Root Squared mean_Error : 13.34155156207796
r_square_value : -0.9561398294047976
```

Scenario-1 Stack model for age - Model Building and Evaluation

In [183]:
```
xgbreg = XGBRegressor()
XGBREG = GridSearchCV(xgbreg, params, n_jobs =-1,return_train_score=True, verbose=True)
```

In [184]:
```
stacking_age = StackingCVRegressor(regressors=[lr_age,RF_age], meta_regressor=XGBREG, cv=3)
```

In [185]:
```
# Fit on train data
stacking_age_pipe = make_pipeline(vec, stacking_age)
stacking_age_pipe.fit(X_dict, ytrain_age)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
```

Out[185]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('stackingcvregressor',
                 StackingCVRegressor(cv=3,
                                     meta_regressor=GridSearchCV(estimator=XGBRegressor(base_score=None,
                                                                                        booster=None,
                                                                                        callbacks=None,
                                                                                        colsample_bylevel=None,
                                                                                        colsample_bynode=None,
                                                                                        colsample_bytree=None,
                                                                                        early_stopping_rounds=None,
                                                                                        enable_categorical=False,
                                                                                        eval_metric=None,
                                                                                        feature_types=None,...
                                                                                        random_state=None, ...),
                                                                 n_jobs=-1,
                                                                 param_grid={'gamma': [0.5,
                                                                                       1],
                                                                             'learning_rate': [0.1,
                                                                                               0.2],
                                                                             'max_depth': [3,
                                                                                           4],
                                                                             'min_child_weight': [1,
                                                                                                  5],
                                                                             'n_estimators': [5,
                                                                                              6],
                                                                             'subsample': [0.6,
                                                                                           0.8]},
                                                                 return_train_score=True,
                                                                 verbose=True),
                                     regressors=[LinearRegression(),
                                                 RandomForestRegressor(max_depth=150,
                                                                       max_features='sqrt',
                                                                       min_samples_leaf=4,
                                                                       min_samples_split=15,
                                                                       random_state=42)]))])
```

In [186]:
```
ypred = stacking_age_pipe.predict(Xtest_dict)
mse = mean_squared_error(ytest_age, ypred)
r_squared = r2_score(ytest_age, ypred)
rsme = math.sqrt(mse)
```

In [187]:
```
print('Mean_Squared_Error :' ,mse)
print('Root Squared mean_Error :' ,rsme)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 494.3100565904452
Root Squared mean_Error : 22.233084729529665
r_square_value : -4.432336501078789
```

e. Scenario-2 non_Event Data Modelling (Gender Prediction – Logistic, RandomForestClassifier and XGBoostClassifer) & (Age Prediction – Linear, RandomForestRegressor and XGBoostRegressor)

Scenario -2 Logistic Regression Model Building & evaluation

```
In [206]: Scenario2_lr_gender= LogisticRegression(penalty='l1',solver='saga',random_state=0)
```

```
In [207]: Scenario2_lr_gender_pipe = make_pipeline(vec, Scenario2_lr_gender)
          Scenario2_lr_gender_pipe.fit(X_dict_nonevent, ytrain_gender_nonevent)
```

```
Out[207]: Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                          ('logisticregression',
                           LogisticRegression(penalty='l1', random_state=0,
                                              solver='saga'))])
```

Train and Test accuracy metrices

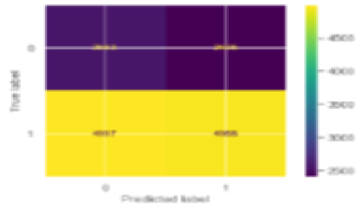
```
In [208]: y_train_pred = Scenario2_lr_gender_pipe.predict(X_dict_nonevent)
```

```
In [209]: print("Train accuracy: ",metrics.accuracy_score(ytrain_gender_nonevent, y_train_pred))
```

```
Train accuracy:  0.5106910039113429
```

```
In [210]: y_pred = Scenario2_lr_gender_pipe.predict(Xtest_dict_nonevent)
          print("Test accuracy: ",metrics.accuracy_score(ytest_gender_nonevent, y_pred))
          # Plot and print confusion matrix
          cnf_matrix = metrics.confusion_matrix(ytest_gender_nonevent, y_pred)
          print(cnf_matrix)
          metrics.plot_confusion_matrix(Scenario2_lr_gender_pipe, Xtest_dict_nonevent, ytest_gender_nonevent)
          plt.show()
```

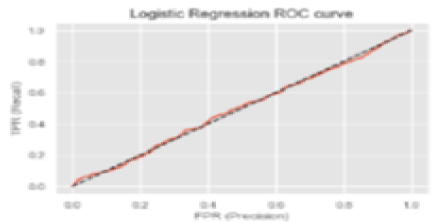
```
Test accuracy:  0.5043192928413S8
[[2563 2405]
 [4997 4968]]
```




```
In [211]: print(classification_report(ytest_gender_nonevent, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.34      0.52      0.41      4968
           1       0.67      0.50      0.57      9965

    accuracy                           0.50     14933
   macro avg       0.51      0.51      0.49     14933
weighted avg       0.56      0.50      0.52     14933
```

```
In [212]: lr_probability =Scenario2_lr_gender_pipe.predict_proba(Xtest_dict_nonevent)[:,1]
          fpr, tpr,_ = roc_curve(ytest_gender_nonevent, lr_probability,)
          plt.title('Logistic Regression ROC curve')
          plt.xlabel('FPR (Precision)')
          plt.ylabel('TPR (Recall)')
          plt.plot(fpr,tpr)
          plt.plot((0,1), ls='dashed',color='black')
          plt.show()
```



Scenario-2 Random Forest model build and evaluation (gender)

In [213]:
```
Scenario2_RF_g = RandomForestClassifier(random_state=42, n_estimators=10,max_depth=4,n_jobs=-1)
```

In [214]:
```
Scenario2_RF_g_pipe = make_pipeline(vec, Scenario2_RF_g)
Scenario2_RF_g_pipe.fit(X_dict_nonevent, ytrain_gender_nonevent)
```

Out[214]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('randomforestclassifier',
                 RandomForestClassifier(max_depth=4, n_estimators=10, n_jobs=-1,
                                        random_state=42))])
```

Train and Test acuuracy metrices

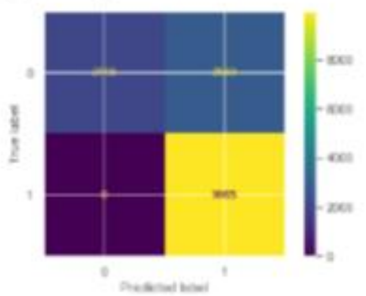
In [215]:
```
y_train_pred = Scenario2_RF_g_pipe.predict(X_dict_nonevent)
```

In [216]:
```
print("Train accuracy: ",metrics.accuracy_score(ytrain_gender_nonevent, y_train_pred))
```

```
Train accuracy:  0.8126249456757931
```

In [217]:
```
y_pred = Scenario2_RF_g_pipe.predict(Xtest_dict_nonevent)
print("Test accuracy: ",metrics.accuracy_score(ytest_gender_nonevent, y_pred))
# Plot and print confusion matrix
cnf_matrix = metrics.confusion_matrix(ytest_gender_nonevent, y_pred)
print(cnf_matrix)
metrics.plot_confusion_matrix(Scenario2_RF_g_pipe, Xtest_dict_nonevent, ytest_gender_nonevent)
plt.show()
```

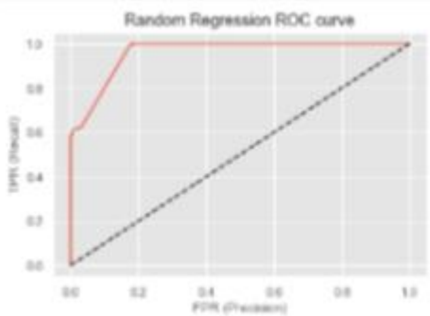
```
Test accuracy:  0.8091475256144111
[[2118 2850]
 [   0 9965]]
```



In [218]:
```
print(classification_report(ytest_gender_nonevent, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.43      0.60      4968
           1       0.78      1.00      0.87      9965

    accuracy                           0.81     14933
   macro avg       0.89      0.71      0.74     14933
weighted avg       0.85      0.81      0.78     14933
```

In [219]:
```
lr_probability =Scenario2_RF_g_pipe.predict_proba(Xtest_dict_nonevent)[:,1]
fpr, tpr,_ = roc_curve(ytest_gender_nonevent, lr_probability,)
plt.title('Random Regression ROC curve')
plt.xlabel('FPR (Precision)')
plt.ylabel('TPR (Recall)')
plt.plot(fpr,tpr)
plt.plot((0,1), ls='dashed',color='black')
plt.show()
```



Scenario-2 Stacking Model Gender (Building and Evaluation)

In [220]:
```
## Stacking gender
Scenario2_stacking_gender = StackingCVClassifier(classifiers=[Scenario2_lr_gender, Scenario2_Rf_g], meta_classifier=XGB, use_pro
```

In [221]:
```
# Fit on train data
Scenario2_stacking_gender_pipe = make_pipeline(vec, Scenario2_stacking_gender)
Scenario2_stacking_gender_pipe.fit(X_dict_nonevent, ytrain_gender_nonevent)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
```

Out[221]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('stackingcvclassifier',
                 StackingCVClassifier(classifiers=[LogisticRegression(penalty='l1',
                                                                      random_state=0,
                                                                      solver='saga'),
                                                   RandomForestClassifier(max_depth=4,
                                                                          n_estimators=10,
                                                                          n_jobs=-1,
                                                                          random_state=42)],
                                      cv=3,
                                      meta_classifier=GridSearchCV(estimator=XGBClassifier(base_score=None,
                                                                                           booster=None,
                                                                                           callbacks=No...
                                                                                           min_child_weight=None,
                                                                                           missing=nan,
                                                                                           monotone_constraints=None,
                                                                                           n_estimators=100,
                                                                                           n_jobs=None,
                                                                                           num_parallel_tree=None,
                                                                                           predictor=None,
                                                                                           random_state=None, ...),
                                                                   n_jobs=-1,
                                                                   param_grid={'gamma': [0.5,
                                                                                         1],
                                                                               'learning_rate': [0.1,
                                                                                                 0.2],
                                                                               'max_depth': [3,
                                                                                             4],
                                                                               'min_child_weight': [1,
                                                                                                    5],
                                                                               'n_estimators': [5,
                                                                                                6],
                                                                               'subsample': [0.6,
                                                                                             0.8]},
                                                                   return_train_score=True,
                                                                   verbose=True),
                                      use_probas=True))])
```

Train and Test accuracy metrices

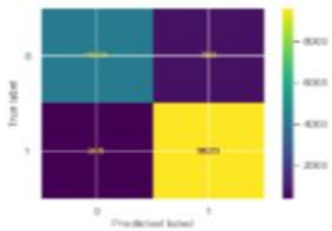
In [222]:
```
y_train_pred = Scenario2_stacking_gender_pipe.predict(X_dict_nonevent)
```

In [223]:
```
print("Train accuracy: ",metrics.accuracy_score(ytrain_gender_nonevent, y_train_pred))
```

```
Train accuracy:  0.9308996088657105
```

In [224]:
```
y_pred = Scenario2_stacking_gender_pipe.predict(Xtest_dict_nonevent)
print("Test Accuracy: ",metrics.accuracy_score(ytest_gender_nonevent, y_pred))
# Plot and print confusion matrix
cnf_matrix = metrics.confusion_matrix(ytest_gender_nonevent, y_pred)
print(cnf_matrix)
metrics.plot_confusion_matrix(Scenario2_stacking_gender_pipe, Xtest_dict_nonevent, ytest_gender_nonevent)
plt.show()
```

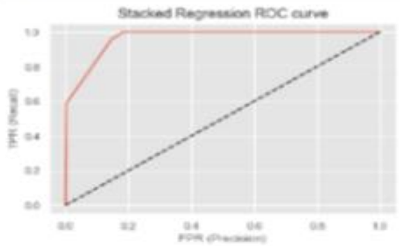
```
Test Accuracy:  0.9254001205384049
[[4199  769]
 [ 345 9620]]
```



In [225]:
```
print(classification_report(ytest_gender_nonevent, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.85      0.88      4968
           1       0.93      0.97      0.95      9965

    accuracy                           0.93     14933
   macro avg       0.93      0.91      0.91     14933
weighted avg       0.93      0.93      0.92     14933
```

In [226]:
```
lr_probability =Scenario2_stacking_gender_pipe.predict_proba(Xtest_dict_nonevent)[:,1]
fpr, tpr,_ = roc_curve(ytest_gender_nonevent, lr_probability,)
plt.title('Stacked Regression ROC curve')
plt.xlabel('FPR (Precision)')
plt.ylabel('TPR (Recall)')
plt.plot(fpr,tpr)
plt.plot((0,1), ls='dashed',color='black')
plt.show()
```



Scenario-2 Linear Regression And RandomForest model for age prediction

Scenario-2 Linear Regression Model Building & Evaluation (age)

In [227]:
```
Sceanrio2_lr_age= LinearRegression()
```

In [228]:
```
# Fit on train data
Sceanrio2_lr_age_pipe = make_pipeline(vec, Sceanrio2_lr_age)
Sceanrio2_lr_age_pipe.fit(X_dict_nonevent, ytrain_age_nonevent)
```

Out[228]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('linearregression', LinearRegression())])
```

In [229]:
```
ypred = Sceanrio2_lr_age_pipe.predict(Xtest_dict_nonevent)
mse = mean_squared_error(ytest_age_nonevent, ypred)
r_squared = r2_score(ytest_age_nonevent, ypred)
rsme = math.sqrt(mse)
print('Mean_Squared_Error :' ,mse)
print('Root Squared mean_Error :' ,rsme)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 94.05383169996855
Root Squared mean_Error : 9.698135475439006
r_square_value : -0.0009663985791381613
```

Scenario-2 Random forest Regression model building & evaluation(age)

In [230]:
```
Sceanrio2_RF_a = RandomForestRegressor(random_state=0, n_estimators=20,max_depth=100,min_samples_split =10, min_samples_leaf= 15
```

In [231]:
```
# Fit on train data
Sceanrio2_RF_age_pipe = make_pipeline(vec, Sceanrio2_RF_a)
Sceanrio2_RF_age_pipe.fit(X_dict_nonevent, ytrain_age_nonevent)
```

Out[231]:
```
Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('randomforestregressor',
                 RandomForestRegressor(max_depth=100, min_samples_leaf=15,
                                       min_samples_split=10, n_estimators=20,
                                       n_jobs=-1, random_state=0))])
```

In [232]:
```
ypred = Sceanrio2_RF_age_pipe.predict(Xtest_dict_nonevent)
mse = mean_squared_error(ytest_age_nonevent, ypred)
r_squared = r2_score(ytest_age_nonevent, ypred)
rsme = math.sqrt(mse)
print('Mean_Squared_Error :' ,mse)
print('Root Squared mean_Error :' ,rsme)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 33.215236754523524
Root Squared mean_Error : 5.7632661533650795
r_square_value : 0.6465073744343799
```

Scenario-2 Stacking model building & evaluation(age)

```
In [233]: Scenario2_stacking_age = StackingCVRegressor(regressors=[Sceanrio2_lr_age,Sceanrio2_RF_a], meta_regressor=XGBREG, cv=3)
```

```
In [234]: # Fit on train data
          Scenario2_stacking_age_pipe = make_pipeline(vec, Scenario2_stacking_age)
          Scenario2_stacking_age_pipe.fit(X_dict_nonevent, ytrain_age_nonevent)

Fitting 5 folds for each of 64 candidates, totalling 320 fits

Out[234]: Pipeline(steps=[('dictvectorizer', DictVectorizer()),
                ('stackingcvregressor',
                 StackingCVRegressor(cv=3,
                                     meta_regressor=GridSearchCV(estimator=XGBRegressor(base_score=None,
                                                                                        booster=None,
                                                                                        callbacks=None,
                                                                                        colsample_bylevel=None,
                                                                                        colsample_bynode=None,
                                                                                        colsample_bytree=None,
                                                                                        early_stopping_rounds=None,
                                                                                        enable_categorical=False,
                                                                                        eval_metric=None,
                                                                                        feature_types=None,...
                                                                                        random_state=None, ...),
                                                                 n_jobs=-1,
                                                                 param_grid={'gamma': [0.5,
                                                                                       1],
                                                                             'learning_rate': [0.1,
                                                                                               0.2],
                                                                             'max_depth': [3,
                                                                                           4],
                                                                             'min_child_weight': [1,
                                                                                                  5],
                                                                             'n_estimators': [5,
                                                                                              6],
                                                                             'subsample': [0.6,
                                                                                           0.8]},
                                                                 return_train_score=True,
                                                                 verbose=True),
                                     regressors=[LinearRegression(),
                                                 RandomForestRegressor(max_depth=100,
                                                                       min_samples_leaf=15,
                                                                       min_samples_split=10,
                                                                       n_estimators=20,
                                                                       n_jobs=-1,
                                                                       random_state=0)]))])
```

```
In [235]: ypred = Scenario2_stacking_age_pipe.predict(Xtest_dict_nonevent)
          mse = mean_squared_error(ytest_age_nonevent, ypred)
          r_squared = r2_score(ytest_age_nonevent, ypred)
          rsme = math.sqrt(mse)
          print('Mean_Squared_Error :' ,mse)
          print('Root Squared mean_Error :' ,rsme)
          print('r_square_value :',r_squared)

Mean_Squared_Error : 110.27068493460831
Root Squared mean_Error : 10.500984950689546
r_square_value : -0.17355399995856077
```

Type Markdown and LaTeX: α^2

10. <u>The reason for using regression or classification for age prediction</u>

a. Age prediction with classification would have let for multiclassification whose interpretation is lot more complex in nature and the end results would have been in categorical in nature were as I wanted to evaluate the model and then assign category so that right campaign could be assigned.

11. <u>The outcomes of the evaluation metrics (results for both Scenario 1 and Scenario 2 must be shown separately). ----- Please refer the scrennshots of point 9 which displays all metics along with models</u>