Json flattening

In [1]:

```
import os
os.chdir('D:/MSBA Classes/Fall/PREDITIVE/predictive project')
import json
import numpy as np
import pandas as pd
from pandas.io.json import json_normalize
```

Loading the train data in chunks

In [2]:

```
file = 'train_v2.csv'
JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']
df_chunk = pd.read_csv(file, chunksize=1000000,
                       dtype={'fullVisitorId': 'str'})
```

Concating the chunks into single Dataframe

In [3]:

```
chunk_list = []

for chunk in df_chunk:
    chunk_list.append(chunk)

train = pd.concat(chunk_list)
```

Removing the unwanted columns

In [4]:

```
train = train.drop(['customDimensions','hits'], axis = 1)
```

In [6]:

```
train.shape
```

Out[6]:

```
(1708337, 11)
```

In [7]:

```
train.head()
```

Out[7]:

	channelGrouping	date	device	fullVisitorId	geoNetwork	socialEngagementType	totals	
0	Organic Search	20171016	{"browser": "Firefox", "browserVersion": "not ...	3162355547410993243	{"continent": "Europe", "subContinent": "Weste...	Not Socially Engaged	{"visits": "1", "hits": "1", "pageviews": "1",...	{"car "s
1	Referral	20171016	{"browser": "Chrome", "browserVersion": "not a...	8934116514970143966	{"continent": "Americas", "subContinent": "Nor...	Not Socially Engaged	{"visits": "1", "hits": "2", "pageviews": "2",...	"/a/google.c

	channelGrouping	date	device	fullVisitorId	geoNetwork	socialEngagementType	totals	
2	Direct	20171016	{"browser": "Chrome", "browserVersion": "not a...	7992466427990357681	{"continent": "Americas", "subContinent": "Nor...	Not Socially Engaged	{"visits": "1", "hits": "2", "pageviews": "2",...	{"car "
3	Organic Search	20171016	{"browser": "Chrome", "browserVersion": "not a...	9075655783635761930	{"continent": "Asia", "subContinent": "Western...	Not Socially Engaged	{"visits": "1", "hits": "2", "pageviews": "2",...	{"car "s
4	Organic Search	20171016	{"browser": "Chrome", "browserVersion": "not a...	6960673291025684308	{"continent": "Americas", "subContinent": "Cen...	Not Socially Engaged	{"visits": "1", "hits": "2", "pageviews": "2",...	{"car "s

Exporting the train data(reduced size) to csv

In [8]:

```
train.to_csv(r'D:/MSBA Classes/Fall/PREDITIVE/predictive project/train_json.csv')
```

Flattening of Json columns in train data

In [9]:

```
df = pd.read_csv('train_json.csv',
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     )
for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
```

In [10]:

```
df.shape
```

Out[10]:

```
(1708337, 59)
```

In [11]:

```
df.head()
```

Out[11]:

	Unnamed: 0	channelGrouping	date	fullVisitorId	socialEngagementType	visitId	visitNumber	visitStartTime	devi
0	0	Organic Search	20171016	3162355547410993243	Not Socially Engaged	1508198450	1	1508198450	
1	1	Referral	20171016	8934116514970143966	Not Socially Engaged	1508176307	6	1508176307	
2	2	Direct	20171016	7992466427990357681	Not Socially Engaged	1508201613	1	1508201613	
3	3	Organic Search	20171016	9075655783635761930	Not Socially Engaged	1508169851	1	1508169851	
4	4	Organic Search	20171016	6960673291025684308	Not Socially Engaged	1508190552	1	1508190552	

5 rows × 59 columns

In [ ]:

```
df.to_csv(r'D:/MSBA Classes/Fall/PREDITIVE/predictive project/train.csv')
```

Loading the test data in chunks

In [12]:

```
file1 = 'test_v2.csv'
JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']
df_chunk = pd.read_csv(file1, chunksize=1000000,
                       dtype={'fullVisitorId': 'str'})
```

Concating the chunks into single Dataframe

In [13]:

```
chunk_list1 = []

for chunk in df_chunk:
    chunk_list1.append(chunk)

test = pd.concat(chunk_list1)
```

Removing the unwanted columns

In [14]:

```
test = test.drop(['customDimensions','hits'], axis = 1)
```

In [16]:

```
test.shape
```

Out[16]:

```
(401589, 11)
```

In [17]:

```
test.head()
```

Out[17]:

	channelGrouping	date	device	fullVisitorId	geoNetwork	socialEngagementType	totals	trafficSour
0	Organic Search	20180511	{"browser": "Chrome", "browserVersion": "not a...	7460955084541987166	{"continent": "Asia", "subContinent": "Souther...	Not Socially Engaged	{"visits": "1", "hits": "4", "pageviews": "3",...	{"referralPat "(not se "campaign (nc
1	Direct	20180511	{"browser": "Chrome", "browserVersion": "not a...	460252456180441002	{"continent": "Americas", "subContinent": "Nor...	Not Socially Engaged	{"visits": "1", "hits": "4", "pageviews": "3",...	{"referralPat "(not se "campaign (nc
2	Organic Search	20180511	{"browser": "Chrome", "browserVersion": "not a...	3461808543879602873	{"continent": "Americas", "subContinent": "Nor...	Not Socially Engaged	{"visits": "1", "hits": "4", "pageviews": "3",...	{"referralPat "(not se "campaign (nc
3	Direct	20180511	{"browser": "Chrome", "browserVersion": "not a...	975129477712150630	{"continent": "Americas", "subContinent": "Nor...	Not Socially Engaged	{"visits": "1", "hits": "5", "pageviews": "4",...	{"referralPat "(not se "campaign (nc
4	Organic Search	20180511	{"browser": "Internet Explorer", "browserVersi...	8381672768065729990	{"continent": "Americas", "subContinent": "Nor...	Not Socially Engaged	{"visits": "1", "hits": "5", "pageviews": "4",...	{"referralPat "(not se "campaign (nc

Exporting the test data(reduced size) to csv

In [18]:

```
test.to_csv(r'D:/MSBA Classes/Fall/PREDITIVE/predictive project/test_json.csv')
```

Flattening of Json columns in test data

In [19]:

```
df2 = pd.read_csv('test_json.csv',
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     )
for column in JSON_COLUMNS:
        column_as_df = json_normalize(df2[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df2 = df2.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
```

In [20]:

```
df2.shape
```

Out[20]:

```
(401589, 58)
```

In [21]:

```
df2.head()
```

Out[21]:

	Unnamed: 0	channelGrouping	date	fullVisitorId	socialEngagementType	visitId	visitNumber	visitStartTime	devi
0	0	Organic Search	20180511	7460955084541987166	Not Socially Engaged	1526099341	2	1526099341	
1	1	Direct	20180511	460252456180441002	Not Socially Engaged	1526064483	166	1526064483	
2	2	Organic Search	20180511	3461808543879602873	Not Socially Engaged	1526067157	2	1526067157	
3	3	Direct	20180511	975129477712150630	Not Socially Engaged	1526107551	4	1526107551	
4	4	Organic Search	20180511	8381672768065729990	Not Socially Engaged	1526060254	1	1526060254	

5 rows × 58 columns

In [ ]:

```
df2.to_csv(r'D:/MSBA Classes/Fall/PREDITIVE/predictive project/test.csv')
```

Predictive-GA Prediction_latest

Data Cleaning & Transformation

In [1]:

```
# import packages

import pandas as pd
import numpy as np
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
```

In [2]:

```python
# Read train data
train = pd.read_csv("train.csv", low_memory = False)
```

In [3]:

```python
pd.set_option('display.max_columns', 500)
```

In [4]:

```python
train.head()
```

Out[4]:

	Unnamed: 0	Unnamed: 0.1	channelGrouping	date	fullVisitorId	socialEngagementType	visitId	visitNumber	visitStai
0	0	0	Organic Search	20171016	3162355547410993243	Not Socially Engaged	1508198450	1	15081
1	1	1	Referral	20171016	8934116514970143966	Not Socially Engaged	1508176307	6	15081
2	2	2	Direct	20171016	7992466427990357681	Not Socially Engaged	1508201613	1	15082
3	3	3	Organic Search	20171016	9075655783635761930	Not Socially Engaged	1508169851	1	15081
4	4	4	Organic Search	20171016	6960673291025684308	Not Socially Engaged	1508190552	1	15081

In [5]:

```python
# Read test data
test = pd.read_csv("test.csv", low_memory = False)
```

In [6]:

```python
test.head()
```

Out[6]:

	Unnamed: 0	Unnamed: 0.1	channelGrouping	date	fullVisitorId	socialEngagementType	visitId	visitNumber	visitStai
0	0	0	Organic Search	20180511	7460955084541987166	Not Socially Engaged	1526099341	2	15260
1	1	1	Direct	20180511	460252456180441002	Not Socially Engaged	1526064483	166	15260
2	2	2	Organic Search	20180511	3461808543879602873	Not Socially Engaged	1526067157	2	15260
3	3	3	Direct	20180511	975129477712150630	Not Socially Engaged	1526107551	4	15261
4	4	4	Organic Search	20180511	8381672768065729990	Not Socially Engaged	1526060254	1	15260

In [7]:

```python
# Drop unnamed columns
train = train.drop(['Unnamed: 0', 'Unnamed: 0.1'], axis=1)
test = test.drop(['Unnamed: 0', 'Unnamed: 0.1'], axis=1)
```

In [8]:

```
# Find data shape
print('Train shape:', train.shape)
print('Test shape:', test.shape)
```

```
Train shape: (1708337, 58)
Test shape: (401589, 57)
```

In [9]:

```
# Find data type
print(train.dtypes.value_counts())
```

```
object     42
float64     9
int64       6
bool        1
dtype: int64
```

In [10]:

```
col_with_constant = pd.DataFrame({'uniq_counts': [col for col in train.columns if train[col].nunique() == 1]})
print("Columns with constant value: ", len(col_with_constant), "columns")
print("Name of constant columns: \n", col_with_constant)
```

```
Columns with constant value:  24 columns
Name of constant columns: 
                                        uniq_counts
0                           socialEngagementType
1                          device.browserVersion
2                             device.browserSize
3                 device.operatingSystemVersion
4                    device.mobileDeviceBranding
5                       device.mobileDeviceModel
6                      device.mobileInputSelector
7                        device.mobileDeviceInfo
8               device.mobileDeviceMarketingName
9                            device.flashVersion
10                               device.language
11                           device.screenColors
12                       device.screenResolution
13                            geoNetwork.cityId
14                          geoNetwork.latitude
15                         geoNetwork.longitude
16                   geoNetwork.networkLocation
17                                totals.visits
18                               totals.bounces
19                             totals.newVisits
20  trafficSource.adwordsClickInfo.criteriaParameters
21                   trafficSource.isTrueDirect
22       trafficSource.adwordsClickInfo.isVideoAd
23                   trafficSource.campaignCode
```

In [11]:

```
constant_cols = col_with_constant['uniq_counts'].tolist()
constant_df = train[constant_cols]
constant_df.head()
```

Out[11]:

	socialEngagementType	device.browserVersion	device.browserSize	device.operatingSystemVersion	device.mobileDeviceBranding	dev
0	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	
1	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	
2	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	

	socialEngagementType	device.browserVersion	device.browserSize	device.operatingSystemVersion	device.mobileDeviceBranding	dev
3	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	
4	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	

In [12]:

```
drop = ['totals.visits', 'totals.bounces', 'totals.newVisits']
for i in drop:
    constant_cols.remove(i)
```

In [13]:

```
# drop columns with constant value from train data
train = train.drop(columns = constant_cols, axis=1)
```

In [14]:

```
# drop columns with constant value from test data
col_with_constant_t = pd.DataFrame({'uniq_counts': [col for col in test.columns if test[col].nuniqu
e() == 1]})
print("Columns with constant value: ", len(col_with_constant_t), "columns")
print("Name of constant columns: \n", col_with_constant_t)
```

```
Columns with constant value:  23 columns
Name of constant columns: 
                                           uniq_counts
0                                socialEngagementType
1                                  device.browserSize
2                               device.browserVersion
3                                 device.flashVersion
4                                     device.language
5                         device.mobileDeviceBranding
6                             device.mobileDeviceInfo
7                    device.mobileDeviceMarketingName
8                            device.mobileDeviceModel
9                          device.mobileInputSelector
10                      device.operatingSystemVersion
11                                device.screenColors
12                            device.screenResolution
13                                  geoNetwork.cityId
14                                geoNetwork.latitude
15                               geoNetwork.longitude
16                         geoNetwork.networkLocation
17                                     totals.bounces
18                                   totals.newVisits
19                                      totals.visits
20    trafficSource.adwordsClickInfo.criteriaParameters
21             trafficSource.adwordsClickInfo.isVideoAd
22                         trafficSource.isTrueDirect
```

In [15]:

```
constant_cols_t = col_with_constant_t['uniq_counts'].tolist()
constant_df_t = test[constant_cols_t]
constant_df_t.head()
```

Out[15]:

	socialEngagementType	device.browserSize	device.browserVersion	device.flashVersion	device.language	device.mobileDeviceBranding
0	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo datase
1	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo datase
2	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo datase
3	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo datase

	socialEngagementType	device.browserSize	device.browserVersion	device.flashVersion	device.language	device.mobileDeviceBranding
4	Not Socially Engaged	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset	not available in demo dataset

In [94]:

```
constant_cols_t
```

Out[94]:

```
['socialEngagementType',
 'device.browserSize',
 'device.browserVersion',
 'device.flashVersion',
 'device.language',
 'device.mobileDeviceBranding',
 'device.mobileDeviceInfo',
 'device.mobileDeviceMarketingName',
 'device.mobileDeviceModel',
 'device.mobileInputSelector',
 'device.operatingSystemVersion',
 'device.screenColors',
 'device.screenResolution',
 'geoNetwork.cityId',
 'geoNetwork.latitude',
 'geoNetwork.longitude',
 'geoNetwork.networkLocation',
 'trafficSource.adwordsClickInfo.criteriaParameters',
 'trafficSource.adwordsClickInfo.isVideoAd',
 'trafficSource.isTrueDirect']
```

In [16]:

```
drop = ['totals.visits', 'totals.bounces', 'totals.newVisits']
for i in drop:
    constant_cols_t.remove(i)

# drop columns with constant value from train data
test = test.drop(columns = constant_cols_t, axis=1)
```

In [17]:

```
test.head()
```

Out[17]:

	channelGrouping	date	fullVisitorId	visitId	visitNumber	visitStartTime	device.browser	device.deviceCategory
0	Organic Search	20180511	7460955084541987166	1526099341	2	1526099341	Chrome	mobile
1	Direct	20180511	460252456180441002	1526064483	166	1526064483	Chrome	desktop
2	Organic Search	20180511	3461808543879602873	1526067157	2	1526067157	Chrome	desktop
3	Direct	20180511	975129477712150630	1526107551	4	1526107551	Chrome	mobile
4	Organic Search	20180511	8381672768065729990	1526060254	1	1526060254	Internet Explorer	tablet

In [18]:

```
# Print data shape after dropping constant columns
print('Train shape:', train.shape)
print('Test shape:', test.shape)
```

```
Train shape: (1708337, 37)
Test shape: (401589, 37)
```

In [19]:

```
# Find if there's missing value in data
```

```python
def find_missing(data):
    #find the features that have missing values
    count_missing=data.isnull().sum().values
    total=data.shape[0]
    ratio_missing=count_missing/total
    return pd.DataFrame({'missing_count':count_missing,'missing_ratio':ratio_missing},
                                    index=data.columns.values)

train_missing=find_missing(train)
test_missing=find_missing(test)

#merge the train and test missing ratio
train_missing.reset_index()[['index','missing_ratio']].merge(test_missing.reset_index()[['index','
missing_ratio']],on='index',how='left')\
                   .rename(columns={'index':'columns','missing_ratio_x':'train_missing_ratio','missi
ng_ratio_y':'test_missing_ratio'})\
                   .sort_values(['train_missing_ratio','test_missing_ratio'],ascending=False).query(
'train_missing_ratio>0')
```

Out[19]:

	columns	train_missing_ratio	test_missing_ratio
25	totals.transactionRevenue	0.989163	0.988560
26	totals.totalTransactionRevenue	0.989163	0.988560
24	totals.transactions	0.989136	0.984300
32	trafficSource.adContent	0.962105	0.000000
33	trafficSource.adwordsClickInfo.page	0.955937	0.973592
34	trafficSource.adwordsClickInfo.slot	0.955937	0.973592
36	trafficSource.adwordsClickInfo.adNetworkType	0.955937	0.973592
35	trafficSource.adwordsClickInfo.gclId	0.955850	0.973575
31	trafficSource.referralPath	0.668529	0.000000
30	trafficSource.keyword	0.616260	0.100167
23	totals.timeOnSite	0.511781	0.457398
20	totals.bounces	0.489809	0.545112
22	totals.sessionQualityDim	0.488940	0.000000
21	totals.newVisits	0.234677	0.287667
19	totals.pageviews	0.000140	0.000252

In [20]:

```python
# Find numerical columns in train data
numeric_features_train = train.select_dtypes(include=[np.number])
numeric_features_train.columns
```

Out[20]:

```
Index(['date', 'visitId', 'visitNumber', 'visitStartTime', 'totals.visits',
       'totals.hits', 'totals.pageviews', 'totals.bounces', 'totals.newVisits',
       'totals.sessionQualityDim', 'totals.timeOnSite', 'totals.transactions',
       'totals.transactionRevenue', 'totals.totalTransactionRevenue',
       'trafficSource.adwordsClickInfo.page'],
      dtype='object')
```

In [21]:

```python
# Find numerical columns in test data
numeric_features_test = test.select_dtypes(include=[np.number])
numeric_features_test.columns
```

Out[21]:

```
Index(['date', 'visitId', 'visitNumber', 'visitStartTime', 'totals.bounces',
       'totals.hits', 'totals.newVisits', 'totals.pageviews',
       'totals.sessionQualityDim', 'totals.timeOnSite',
       'totals.totalTransactionRevenue', 'totals.transactionRevenue',
```

```
       'totals.transactions', 'totals.visits',
       'trafficSource.adwordsClickInfo.page'],
      dtype='object')
```

In [22]:

```
# Find categorical columns in train data
categorical_features_train = train.select_dtypes(include=[np.object])
categorical_features_train.columns
```

Out[22]:

```
Index(['channelGrouping', 'fullVisitorId', 'device.browser',
       'device.operatingSystem', 'device.deviceCategory',
       'geoNetwork.continent', 'geoNetwork.subContinent', 'geoNetwork.country',
       'geoNetwork.region', 'geoNetwork.metro', 'geoNetwork.city',
       'geoNetwork.networkDomain', 'trafficSource.campaign',
       'trafficSource.source', 'trafficSource.medium', 'trafficSource.keyword',
       'trafficSource.referralPath', 'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.slot',
       'trafficSource.adwordsClickInfo.gclId',
       'trafficSource.adwordsClickInfo.adNetworkType'],
      dtype='object')
```

In [23]:

```
# Find categorical columns in test data
categorical_features_test = test.select_dtypes(include=[np.object])
categorical_features_test.columns
```

Out[23]:

```
Index(['channelGrouping', 'fullVisitorId', 'device.browser',
       'device.deviceCategory', 'device.operatingSystem', 'geoNetwork.city',
       'geoNetwork.continent', 'geoNetwork.country', 'geoNetwork.metro',
       'geoNetwork.networkDomain', 'geoNetwork.region',
       'geoNetwork.subContinent', 'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.adNetworkType',
       'trafficSource.adwordsClickInfo.gclId',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.keyword', 'trafficSource.medium',
       'trafficSource.referralPath', 'trafficSource.source'],
      dtype='object')
```

In [24]:

```
miss_cols = ['trafficSource.adContent', 'trafficSource.adContent',
'trafficSource.adwordsClickInfo.page',
'trafficSource.adwordsClickInfo.slot', 'trafficSource.adwordsClickInfo.adNetworkType',
'trafficSource.adwordsClickInfo.gclId', 'trafficSource.referralPath', 'trafficSource.keyword',
'totals.timeOnSite', 'totals.bounces', 'totals.sessionQualityDim', 'totals.newVisits',
'totals.pageviews']
```

In [25]:

```
len(miss_cols)
```

Out[25]:

```
13
```

In [26]:

```
num_cols = numeric_features_train.columns.tolist()
```

In [27]:

```
for col in miss_cols:
    print(col, pd.DataFrame(train[col].unique()))
```

```
trafficSource.adContent                                    0
0                              NaN
1   Placement Accessories 300 x 250
2          Google Merchandise Store
3                      Bags 300x250
4        Display Ad created 3/11/14
..                              ...
72                   GA Help Center
73                   Free Shipping!
74             Swag w/ Google Logos
75         Men's Apparel from Google
76                  Ad from 2/17/17

[77 rows x 1 columns]
```

```
trafficSource.adContent                                    0
0                              NaN
1   Placement Accessories 300 x 250
2          Google Merchandise Store
3                      Bags 300x250
4        Display Ad created 3/11/14
..                              ...
72                   GA Help Center
73                   Free Shipping!
74             Swag w/ Google Logos
75         Men's Apparel from Google
76                  Ad from 2/17/17

[77 rows x 1 columns]
```

```
trafficSource.adwordsClickInfo.page        0
0    NaN
1    1.0
2    3.0
3    2.0
4    5.0
5    6.0
6    4.0
7   14.0
8    7.0
9    8.0
10   9.0
11  12.0
trafficSource.adwordsClickInfo.slot                         0
0                     NaN
1                     Top
2                     RHS
3  Google Display Network
trafficSource.adwordsClickInfo.adNetworkType                  0
0              NaN
1    Google Search
2          Content
3  Search partners
trafficSource.adwordsClickInfo.gclId                                                  0
0                                                    NaN
1      Cj0KCQjwsZHPBRClARIsAC-VMPBHdNF2oMOgh6Xp6YhjXW...
2                            CODVoMjJ9tYCFUIvgQod_dsKEA
3      Cj0KCQjwsZHPBRClARIsAC-VMPA4CVJtDhu1lYkB0AR1hj...
4      Cj0KCQjwsZHPBRClARIsAC-VMPDlLD6kS4tmqFGZjMUqye...
...                                                  ...
59004  CjwKEAiA17LDBRDElqOGq8vR7m8SJAA1AC0_L8MUnlMT5A...
59005  CjwKEAiA17LDBRDElqOGq8vR7m8SJAA1AC0_aFaq6n49gX...
59006  CjwKEAiA17LDBRDElqOGq8vR7m8SJAA1AC0_F8I700U-81...
59007  CjwKEAiA17LDBRDElqOGq8vR7m8SJAA1AC0_HT9qtS8geJ...
59008  CjwKEAiA17LDBRDElqOGq8vR7m8SJAA1AC0_0wzbDX0fd-...

[59009 rows x 1 columns]
trafficSource.referralPath                                                   0
0                                                   NaN
1     /a/google.com/transportation/mtv-services/bike...
2                                           /offer/2145
3     /a/google.com/nest-vision/dropcam-field-tester...
4                                       /analytics/web/
...                                                 ...
3192               /intl/ar/yt/advertise/how-it-works/
3193                        /yt/lineups/ru/france.html
3194                                  /mail/mu/mp/118/
3195                                       /BB1QfReObs
3196                                  /mail/mu/mp/509/
```

```
[3197 rows x 1 columns]
trafficSource.keyword                                              0
0                         water bottle
1                                  NaN
2                       (not provided)
3      (Remarketing/Content targeting)
4                     6qEhsCssdK0z36ri
...                                ...
4542                         www googl
4543     global corporate merchandise
4544                           Mug pic
4545                    shorter google
4546           google shirts for sale

[4547 rows x 1 columns]
totals.timeOnSite            0
0         NaN
1        28.0
2        38.0
3         1.0
4        52.0
...       ...
4770  5564.0
4771  3587.0
4772  4665.0
4773  5381.0
4774  8811.0

[4775 rows x 1 columns]
totals.bounces      0
0  1.0
1  NaN
totals.sessionQualityDim          0
0      1.0
1      2.0
2      3.0
3      4.0
4      6.0
..     ...
96    95.0
97    97.0
98    99.0
99    98.0
100  100.0

[101 rows x 1 columns]
totals.newVisits      0
0  1.0
1  NaN
totals.pageviews          0
0      1.0
1      2.0
2      3.0
3      4.0
4      5.0
..     ...
226  232.0
227  429.0
228  213.0
229  151.0
230  186.0

[231 rows x 1 columns]
```

In [28]:

```
# Fill nan numeric value with zero
miss_col_num = ['trafficSource.adwordsClickInfo.page', 'totals.timeOnSite', 'totals.bounces', 'totals.sessionQualityDim', 'totals.newVisits',
               'totals.pageviews']
def fill_na_num(df):
    for col in miss_col_num:
        df[col] = train[col].fillna(0).astype(int)
    return df
```

In [29]:

```
train.head()
```

Out[29]:

	channelGrouping	date	fullVisitorId	visitId	visitNumber	visitStartTime	device.browser	device.operatingSystem
0	Organic Search	20171016	3162355547410993243	1508198450	1	1508198450	Firefox	Windows
1	Referral	20171016	8934116514970143966	1508176307	6	1508176307	Chrome	Chrome OS
2	Direct	20171016	7992466427990357681	1508201613	1	1508201613	Chrome	Android
3	Organic Search	20171016	9075655783635761930	1508169851	1	1508169851	Chrome	Windows
4	Organic Search	20171016	6960673291025684308	1508190552	1	1508190552	Chrome	Windows

In [30]:

```
cat_cols = ['channelGrouping',
 'device.browser',
 'device.isMobile',
 'device.operatingSystem',
 'device.deviceCategory',
 'geoNetwork.continent',
 'geoNetwork.subContinent',
 'geoNetwork.country',
 'geoNetwork.region',
 'geoNetwork.etro',
 'geoNetwork.city',
 'geoNetwork.networkDomain',
 'trafficSource.campaign',
 'trafficSource.source',
 'trafficSource.medium',
 'trafficSource.keyword',
 'trafficSource.referralPath',
 'trafficSource.adContent',
 'trafficSource.adwordsClickInfo.slot',
 'trafficSource.adwordsClickInfo.gclId',
 'trafficSource.adwordsClickInfo.adNetworkType']
```

In [31]:

```
train.head()
```

Out[31]:

	channelGrouping	date	fullVisitorId	visitId	visitNumber	visitStartTime	device.browser	device.operatingSystem
0	Organic Search	20171016	3162355547410993243	1508198450	1	1508198450	Firefox	Windows
1	Referral	20171016	8934116514970143966	1508176307	6	1508176307	Chrome	Chrome OS
2	Direct	20171016	7992466427990357681	1508201613	1	1508201613	Chrome	Android
3	Organic Search	20171016	9075655783635761930	1508169851	1	1508169851	Chrome	Windows
4	Organic Search	20171016	6960673291025684308	1508190552	1	1508190552	Chrome	Windows

In [32]:

```
test.head()
```

Out[32]:

	channelGrouping	date	fullVisitorId	visitId	visitNumber	visitStartTime	device.browser	device.deviceCategory
0	Organic Search	20180511	7460955084541987166	1526099341	2	1526099341	Chrome	mobile
1	Direct	20180511	460252456180441002	1526064483	166	1526064483	Chrome	desktop
2	Organic Search	20180511	3461808543879602873	1526067157	2	1526067157	Chrome	desktop
3	Direct	20180511	975129477712150630	1526107551	4	1526107551	Chrome	mobile
4	Organic Search	20180511	8381672768065729990	1526060254	1	1526060254	Internet Explorer	tablet

In [34]:

```
# Transfer datetime data
def date_converter(df):
    df['date'] = df['date'].astype(str)
    df["date"] = df["date"].apply(lambda x : x[:4] + "-" + x[4:6] + "-" + x[6:])
    df["date"] = pd.to_datetime(df["date"])

    return df
```

In [35]:

```
train_2 = date_converter(train)
```

In [36]:

```
test_2 = date_converter(test)
```

In [37]:

```
train_2 = test.reindex(sorted(train.columns), axis = 1)
test_2 = test.reindex(sorted(test.columns), axis = 1)
```

In [38]:

```
data = pd.concat([train_2, test_2])
```

In [39]:

```
data.head()
```

Out[39]:

	channelGrouping	date	device.browser	device.deviceCategory	device.isMobile	device.operatingSystem	fullVisitorId	geol
0	Organic Search	2018-05-11	Chrome	mobile	True	Android	7460955084541987166	
1	Direct	2018-05-11	Chrome	desktop	False	Macintosh	460252456180441002	S
2	Organic Search	2018-05-11	Chrome	desktop	False	Chrome OS	3461808543879602873	nc c
3	Direct	2018-05-11	Chrome	mobile	True	iOS	975129477712150630	
4	Organic Search	2018-05-11	Internet Explorer	tablet	True	Windows	8381672768065729990	

In [40]:

```
num_cols = ['totals.visits', 'totals.hits', 'totals.pageviews', 'totals.bounces', 'totals.newVisits
', 'totals.sessionQualityDim', 'totals.timeOnSite', 'totals.transactions',
'totals.transactionRevenue', 'totals.totalTransactionRevenue']
```

```
for col in num_cols:
    data[col] = data[col].fillna(0)
```

In [41]:

```
data['device.isMobile'] = data['device.isMobile'].map({True:1,False:0})
```

In [42]:

```
data.columns
```

Out[42]:

```
Index(['channelGrouping', 'date', 'device.browser', 'device.deviceCategory',
       'device.isMobile', 'device.operatingSystem', 'fullVisitorId',
       'geoNetwork.city', 'geoNetwork.continent', 'geoNetwork.country',
       'geoNetwork.metro', 'geoNetwork.networkDomain', 'geoNetwork.region',
       'geoNetwork.subContinent', 'totals.bounces', 'totals.hits',
       'totals.newVisits', 'totals.pageviews', 'totals.sessionQualityDim',
       'totals.timeOnSite', 'totals.totalTransactionRevenue',
       'totals.transactionRevenue', 'totals.transactions', 'totals.visits',
       'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.adNetworkType',
       'trafficSource.adwordsClickInfo.gclId',
       'trafficSource.adwordsClickInfo.page',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.keyword', 'trafficSource.medium',
       'trafficSource.referralPath', 'trafficSource.source', 'visitId',
       'visitNumber', 'visitStartTime'],
      dtype='object')
```

In [48]:

```
def advance_tranformation(data, k):
    date_min = data['date'].min()
    date_max = data['date'].max()
    df = data[(data['date'] >= date_min + timedelta(168 * (k-1))) & (data['date'] < date_min +
timedelta(days=168 * k))]
    df_future_id = pd.unique(data[(data['date'] >= date_min + timedelta(days = 168 * k + 46)) & (da
ta['date'] < date_min + timedelta(days=168 * k + 46 + 62))]['fullVisitorId'])
    df_return = df[df['fullVisitorId'].isin(df_future_id)]
    return_unique = pd.unique(df_return['fullVisitorId'])
    df_test = data[(data['fullVisitorId'].isin(return_unique))]
    df_test = df_test[(df_test['date'] >= date_min + timedelta(days = 168 * k + 46)) & (df_test['da
te'] < date_min + timedelta(days=168 * k + 46 + 62))]
    df_target = df_test.groupby('fullVisitorId').apply(lambda x:
np.log1p(sum(x['totals.transactionRevenue']))).reset_index()
    df_target['ret'] = 1
    df_target = df_target.rename(columns= {0: 'target'})
    df_not_return = df[df['fullVisitorId'].isin(df_future_id) == False]
    not_return_unique = pd.unique(df_not_return['fullVisitorId'])
    df_not_ret_df = pd.DataFrame({'fullVisitorId': not_return_unique, 'target': 0, 'ret': 0})
    target_df = pd.concat([df_target, df_not_ret_df])

    df_group = pd.DataFrame()

    df_group['channelGrouping'] = df.groupby('fullVisitorId').agg(lambda x: x['channelGrouping'].mo
de()[0])['channelGrouping']
    df_group['first_ses_from_the_period_start'] = df['date'].min() - date_min
    df_group['last_ses_from_the_period_end'] = date_max - df['date'].max()
    df_group['interval_dates'] = df['date'].max() - df['date'].min()
    df_group['unique_date_num'] = len(pd.unique(df['date']))
    df_group['maxVisitNum'] = df.groupby('fullVisitorId')['totals.visits'].agg("max")
    df_group['browser'] = df.groupby('fullVisitorId').agg(lambda x: x['device.browser'].mode()[0])[
'device.browser']
    df_group['operatingSystem'] = df.groupby('fullVisitorId').agg(lambda x:
x['device.operatingSystem'].mode()[0])['device.operatingSystem']
    df_group['deviceCategory'] = df.groupby('fullVisitorId').agg(lambda x:
x['device.deviceCategory'].mode()[0])['device.deviceCategory']
    df_group['continent'] = df.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.continent'].mod
e()[0])['geoNetwork.continent']
    df_group['subContinent'] = df.groupby('fullVisitorId').agg(lambda x:
x['geoNetwork.subContinent'].mode()[0])['geoNetwork.subContinent']
```

```
    df_group['country'] = df.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.country'].mode()[
0])['geoNetwork.country']
    df_group['region'] = df.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.region'].mode()[0]
)['geoNetwork.region']
    df_group['metro'] = df.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.metro'].mode()[0])[
'geoNetwork.metro']
    df_group['city'] = df.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.city'].mode()[0])['g
eoNetwork.city']
    df_group['networkDomain'] = df.groupby('fullVisitorId').agg(lambda x:
x['geoNetwork.networkDomain'].mode()[0])['geoNetwork.networkDomain']
    df_group['source'] = df.groupby('fullVisitorId').agg(lambda x: x['trafficSource.source'].mode()
[0])['trafficSource.source']
    df_group['medium'] = df.groupby('fullVisitorId').agg(lambda x: x['trafficSource.medium'].mode()
[0])['trafficSource.medium']
    df_group['isMobile']= df.groupby('fullVisitorId')['device.isMobile'].agg("sum")
    df_group['bounce_sessions'] = df.groupby('fullVisitorId')['totals.bounces'].agg("sum")
    df_group['hits_sum'] = df.groupby('fullVisitorId')['totals.hits'].agg("sum")
    df_group['hits_mean'] = df.groupby('fullVisitorId')['totals.hits'].agg("mean")
    df_group['pageviews_sum'] = df.groupby('fullVisitorId')['totals.pageviews'].agg("sum")
    df_group['pageviews_mean'] = df.groupby('fullVisitorId')['totals.pageviews'].agg("mean")
    df_group['session_cnt'] = len(df['visitStartTime'])
    df_group['transactionRevenue'] = df.groupby('fullVisitorId')['totals.transactionRevenue'].agg("
sum")
    df_group['transactions'] = df.groupby('fullVisitorId')['totals.transactions'].agg("sum")

    df_group = pd.merge(df_group, target_df, on='fullVisitorId', sort=False)


    return df_group
```

In [49]:

```
tr_1 = advance_tranformation(data, 1)
tr_2 = advance_tranformation(data, 2)
tr_3 = advance_tranformation(data, 3)
tr_4 = advance_tranformation(data, 4)
```

```
C:\Users\jiash\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_launcher.py:16:
FutureWarning: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  app.launch_new_instance()
```

In [58]:

```
tr_1.head()
```

Out[58]:

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique_date_r
0	0000018966949534117	Organic Search	0 days	0 days	167 days	
1	0000039738481224681	Direct	0 days	0 days	167 days	
2	0000073585230191399	Organic Search	0 days	0 days	167 days	
3	0000087588448856385	Organic Search	0 days	0 days	167 days	

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique_date_r
4	0000149787903119437	Organic Search	0 days	0 days	167 days	

In [92]:

```
tr_5 = data[data['date'] >= '2018-05-01']
tr_5.head()
```

Out[92]:

	channelGrouping	date	device.browser	device.deviceCategory	device.isMobile	device.operatingSystem	fullVisitorId	geol
0	Organic Search	2018-05-11	Chrome	mobile	1	Android	7460955084541987166	
1	Direct	2018-05-11	Chrome	desktop	0	Macintosh	460252456180441002	S
2	Organic Search	2018-05-11	Chrome	desktop	0	Chrome OS	3461808543879602873	nc c
3	Direct	2018-05-11	Chrome	mobile	1	iOS	975129477712150630	
4	Organic Search	2018-05-11	Internet Explorer	tablet	1	Windows	8381672768065729990	

In [93]:

```
# Build testing dataset
tr_5 = data[data['date'] >= '2018-05-01']
tr_5_maxdate = tr_5['date'].max()
tr_5_mindate = tr_5['date'].min()

df_group_5 = pd.DataFrame()

df_group_5['channelGrouping'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['channelGrouping'].mo
de()[0])['channelGrouping']
df_group_5['first_ses_from_the_period_start'] = tr_5['date'].min() - tr_5_mindate
df_group_5['last_ses_from_the_period_end'] = tr_5_maxdate - tr_5['date'].max()
df_group_5['interval_dates'] = tr_5['date'].max() - tr_5['date'].min()
df_group_5['unique_date_num'] = len(pd.unique(tr_5['date']))
df_group_5['maxVisitNum'] = tr_5.groupby('fullVisitorId')['totals.visits'].agg("max")
df_group_5['browser'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['device.browser'].mode()[0])[
'device.browser']
df_group_5['operatingSystem'] = tr_5.groupby('fullVisitorId').agg(lambda x:
x['device.operatingSystem'].mode()[0])['device.operatingSystem']
df_group_5['deviceCategory'] = tr_5.groupby('fullVisitorId').agg(lambda x:
x['device.deviceCategory'].mode()[0])['device.deviceCategory']
df_group_5['continent'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.continent'].mod
e()[0])['geoNetwork.continent']
df_group_5['subContinent'] = tr_5.groupby('fullVisitorId').agg(lambda x:
x['geoNetwork.subContinent'].mode()[0])['geoNetwork.subContinent']
df_group_5['country'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.country'].mode()[
0])['geoNetwork.country']
df_group_5['region'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.region'].mode()[0]
)['geoNetwork.region']
df_group_5['metro'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.metro'].mode()[0])[
'geoNetwork.metro']
df_group_5['city'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['geoNetwork.city'].mode()[0])['g
eoNetwork.city']
df_group_5['networkDomain'] = tr_5.groupby('fullVisitorId').agg(lambda x:
x['geoNetwork.networkDomain'].mode()[0])['geoNetwork.networkDomain']
df_group_5['source'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['trafficSource.source'].mode()
[0])['trafficSource.source']
df_group_5['medium'] = tr_5.groupby('fullVisitorId').agg(lambda x: x['trafficSource.medium'].mode()
[0])['trafficSource.medium']
df_group_5['isMobile']= tr_5.groupby('fullVisitorId')['device.isMobile'].agg("sum")
df_group_5['bounce_sessions'] = tr_5.groupby('fullVisitorId')['totals.bounces'].agg("sum")
df_group_5['hits_sum'] = tr_5.groupby('fullVisitorId')['totals.hits'].agg("sum")
df_group_5['hits_mean'] = tr_5.groupby('fullVisitorId')['totals.hits'].agg("mean")
df_group_5['pageviews_sum'] = tr_5.groupby('fullVisitorId')['totals.pageviews'].agg("sum")
df_group_5['pageviews_mean'] = tr_5.groupby('fullVisitorId')['totals.pageviews'].agg("mean")
df_group_5['session_cnt'] = len(tr_5['visitStartTime'])
df_group_5['transactionRevenue'] = tr_5.groupby('fullVisitorId')['totals.transactionRevenue'].agg(
"sum")
```

```
df_group_5['transactions'] = tr_5.groupby('fullVisitorId')['totals.transactions'].agg("sum")
df_group_5['target'] = np.nan
df_group_5['return'] = np.nan
```

In [95]:

```
tr_5.head()
```

Out[95]:

	channelGrouping	date	device.browser	device.deviceCategory	device.isMobile	device.operatingSystem	fullVisitorId	geoI
0	Organic Search	2018-05-11	Chrome	mobile	1	Android	7460955084541987166	
1	Direct	2018-05-11	Chrome	desktop	0	Macintosh	460252456180441002	S
2	Organic Search	2018-05-11	Chrome	desktop	0	Chrome OS	3461808543879602873	nc c
3	Direct	2018-05-11	Chrome	mobile	1	iOS	975129477712150630	
4	Organic Search	2018-05-11	Internet Explorer	tablet	1	Windows	8381672768065729990	

In [96]:

```
train_combined = pd.concat([tr_1, tr_2, tr_3, tr_4, tr_5], sort=False)
```

In [97]:

```
train_combined.columns
```

Out[97]:

```
Index(['fullVisitorId', 'channelGrouping', 'first_ses_from_the_period_start',
       'last_ses_from_the_period_end', 'interval_dates', 'unique_date_num',
       'maxVisitNum', 'browser', 'operatingSystem', 'deviceCategory',
       'continent', 'subContinent', 'country', 'region', 'metro', 'city',
       'networkDomain', 'source', 'medium', 'isMobile', 'bounce_sessions',
       'hits_sum', 'hits_mean', 'pageviews_sum', 'pageviews_mean',
       'session_cnt', 'transactionRevenue', 'transactions', 'ret', 'target',
       'date', 'device.browser', 'device.deviceCategory', 'device.isMobile',
       'device.operatingSystem', 'geoNetwork.city', 'geoNetwork.continent',
       'geoNetwork.country', 'geoNetwork.metro', 'geoNetwork.networkDomain',
       'geoNetwork.region', 'geoNetwork.subContinent', 'totals.bounces',
       'totals.hits', 'totals.newVisits', 'totals.pageviews',
       'totals.sessionQualityDim', 'totals.timeOnSite',
       'totals.totalTransactionRevenue', 'totals.transactionRevenue',
       'totals.transactions', 'totals.visits', 'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.adNetworkType',
       'trafficSource.adwordsClickInfo.gclId',
       'trafficSource.adwordsClickInfo.page',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.keyword', 'trafficSource.medium',
       'trafficSource.referralPath', 'trafficSource.source', 'visitId',
       'visitNumber', 'visitStartTime'],
      dtype='object')
```

In [98]:

```
drop_col = ['date', 'device.browser', 'device.deviceCategory', 'device.isMobile',
       'device.operatingSystem', 'geoNetwork.city', 'geoNetwork.continent',
       'geoNetwork.country', 'geoNetwork.metro', 'geoNetwork.networkDomain',
       'geoNetwork.region', 'geoNetwork.subContinent', 'totals.bounces',
       'totals.hits', 'totals.newVisits', 'totals.pageviews',
       'totals.sessionQualityDim', 'totals.timeOnSite',
       'totals.totalTransactionRevenue', 'totals.transactionRevenue',
       'totals.transactions', 'totals.visits', 'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.adNetworkType',
       'trafficSource.adwordsClickInfo.gclId',
       'trafficSource.adwordsClickInfo.page',
```

```
       'trafficSource.adwordsClickInfo.page',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.keyword', 'trafficSource.medium',
       'trafficSource.referralPath', 'trafficSource.source', 'visitId',
       'visitNumber', 'visitStartTime']
```

In [99]:

```python
train_combined_2 = train_combined.drop(columns=drop_col)
```

In [108]:

```python
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
c = ['city','browser','channelGrouping',"operatingSystem","continent",'subContinent','country',
     'region','metro','networkDomain','source','medium', 'deviceCategory']
for col in c:
    train_combined_2[col] = train_combined_2[col].fillna('na')
    train_combined_2[col] = le.fit_transform(train_combined_2[col].values)
```

In [109]:

```python
train_combined_2
```

Out[109]:

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique_
0	0000018966949534117	4	0 days	0 days	167 days	
1	0000039738481224681	2	0 days	0 days	167 days	
2	0000073585230191399	4	0 days	0 days	167 days	
3	0000087588448856385	4	0 days	0 days	167 days	
4	0000149787903119437	4	0 days	0 days	167 days	
...	...	...	...	...	...	
401584	6701149525099562370	4	NaT	NaT	NaT	
401585	6154541330147351453	4	NaT	NaT	NaT	
401586	6013469762773705448	4	NaT	NaT	NaT	
401587	4565378823441900999	4	NaT	NaT	NaT	
401588	3875690118293601911	2	NaT	NaT	NaT	

1099708 rows × 30 columns

In [110]:

```python
train_combined_2['interval_dates'] = pd.to_numeric(train_combined_2['interval_dates'], errors= 'coe
rce')
train_combined_2['first_ses_from_the_period_start'] =
pd.to_numeric(train_combined_2['first_ses_from_the_period_start'], errors= 'coerce')
train_combined_2['last_ses_from_the_period_end'] =
pd.to_numeric(train_combined_2['last_ses_from_the_period_end'], errors= 'coerce')
```

In [111]:

```python
train_combined_2.head()
```

Out[111]:

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique_
0	0000018966949534117	4	0	0	14428800000000000	
1	0000039738481224681	2	0	0	14428800000000000	
2	0000073585230191399	4	0	0	14428800000000000	
3	0000087588448856385	4	0	0	14428800000000000	

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique_(
4	0000149787903119437	4	0	0	14428800000000000	

In [112]:

```
# Divide train/ test
train = train_combined_2[train_combined_2['target'].isna()]
test = train_combined_2[train_combined_2['target'].isna() == False]
```

In [121]:

```
train.sort_values(by='first_ses_from_the_period_start')
```

Out[121]:

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	(
0	7460955084541987166	4	-9223372036854775808	-9223372036854775808	-9223372036854775808	
133857	9900874527485563945	4	-9223372036854775808	-9223372036854775808	-9223372036854775808	
133858	6037578503108830215	2	-9223372036854775808	-9223372036854775808	-9223372036854775808	
133859	8630812730608319988	4	-9223372036854775808	-9223372036854775808	-9223372036854775808	
133860	7827975973225241449	4	-9223372036854775808	-9223372036854775808	-9223372036854775808	
...	...	...	...	...	...	
267729	7476629900831012268	6	-9223372036854775808	-9223372036854775808	-9223372036854775808	
267730	1184221904466941690	5	-9223372036854775808	-9223372036854775808	-9223372036854775808	
267731	7875020615350219520	2	-9223372036854775808	-9223372036854775808	-9223372036854775808	
267733	8192330271104386623	4	-9223372036854775808	-9223372036854775808	-9223372036854775808	
401588	3875690118293601911	2	-9223372036854775808	-9223372036854775808	-9223372036854775808	

803178 rows × 30 columns

In [116]:

```
test.head()
```

Out[116]:

	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique_(
0	0000018966949534117	4	0	0	14428800000000000	
1	0000039738481224681	2	0	0	14428800000000000	
2	0000073585230191399	4	0	0	14428800000000000	
3	0000087588448856385	4	0	0	14428800000000000	
4	0000149787903119437	4	0	0	14428800000000000	

In [117]:

```
train.shape
```

Out[117]:

```
(803178, 30)
```

In [118]:

```
test.shape
```

Out[118]:

```
(296530, 30)
```

In [119]:

```
train.to_csv("train_final.csv")
```

In [120]:

```
test.to_csv("test_final.csv")
```

Predictive_randomforest_XGBoost

Import necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectKBest, chi2
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz
from sklearn import metrics
from sklearn.metrics import classification_report, mean_squared_error
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn import neighbors
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
import warnings
warnings.filterwarnings('ignore')
```

In [14]:

```
# All training data
df = pd.read_csv(r'C:\Users\ACER\Desktop\UMN\Predictive
Analysis\FinalProject\data\train_final.csv')
len(df)
df.head(5)
```

Out[14]:

	Unnamed: 0	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_dates	unique
0	1	10278554503158	Organic Search	80	87	0	
1	2	20424342248747	Organic Search	121	46	0	
2	3	5103959234087	Organic Search	20	147	0	
3	4	93957001069502	Organic Search	57	110	0	
4	5	114156543135683	Social	7	160	0	

5 rows × 41 columns

In [15]:

```
df = df.drop(columns = ['isVideoAd_mean',
                        'Unnamed: 0',
                        'isTrueDirect',
                        'hits_min',
                        'hits_max',
                        'hits_median',
                        'hits_sd',
                        'pageviews_min',
                        'pageviews_max',
                        'pageviews_median',
                        'pageviews_sd'])
```

In [16]:

```
len(set(df['networkDomain']))
```

Out[16]:

```
42722
```

In [17]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1417575 entries, 0 to 1417574
Data columns (total 30 columns):
fullVisitorId                     1417575 non-null object
channelGrouping                   1417575 non-null object
first_ses_from_the_period_start   1417575 non-null int64
last_ses_from_the_period_end      1417575 non-null int64
interval_dates                    1417575 non-null int64
unique_date_num                   1417575 non-null int64
maxVisitNum                       1417575 non-null int64
browser                           1417575 non-null object
operatingSystem                   1417575 non-null object
deviceCategory                    1417575 non-null object
continent                         1417575 non-null object
subContinent                      1417575 non-null object
country                           1417575 non-null object
region                            1417575 non-null object
metro                             1417575 non-null object
city                              1417575 non-null object
networkDomain                     1417575 non-null object
source                            1417575 non-null object
medium                            1417575 non-null object
isMobile                          1417575 non-null float64
bounce_sessions                   1417575 non-null int64
hits_sum                          1417575 non-null int64
hits_mean                         1417575 non-null float64
pageviews_sum                     1417575 non-null int64
pageviews_mean                    1417524 non-null float64
session_cnt                       1417575 non-null int64
transactionRevenue                1417575 non-null float64
transactions                      1417575 non-null int64
target                            1417575 non-null float64
ret                               1417575 non-null int64
dtypes: float64(5), int64(11), object(14)
memory usage: 324.5+ MB
```

In [18]:

```
df_test = pd.read_csv(r'C:\Users\ACER\Desktop\UMN\Predictive
Analysis\FinalProject\data\test_final.csv')
len(df_test)
df_test
```

Out[18]:

	Unnamed: 0	fullVisitorId	channelGrouping	first_ses_from_the_period_start	last_ses_from_the_period_end	interval_date
0	1417576	18966949534117	Organic Search	104	63	
1	1417577	39738481224681	Direct	43	124	
2	1417578	73585230191399	Organic Search	33	134	
3	1417579	87588448856385	Organic Search	36	131	
4	1417580	149787903119437	Organic Search	20	147	
...	...	...	...	...	...	
296525	1714101	9999862054614696520	Organic Search	7	160	
296526	1714102	9999898168621645223	Organic Search	67	100	
296527	1714103	999990167740728398	Direct	93	74	
296528	1714104	9999915620249883537	Organic Search	46	121	
296529	1714105	9999947552481876143	Organic Search	108	59	

296530 rows × 41 columns

In [19]:

```
df_test = df_test.drop(columns = ['isVideoAd_mean',
                            'Unnamed: 0',
                            'isTrueDirect',
                            'hits_min',
                            'hits_max',
                            'hits_median',
                            'hits_sd',
                            'pageviews_min',
                            'pageviews_max',
                            'pageviews_median',
                            'pageviews_sd'])
```

In [20]:

```
# use label encoding to convert categorical data to numeric one
c = ['city','browser','channelGrouping',"operatingSystem","continent",'subContinent','country',
                             'region','metro','networkDomain','source','medium','deviceCategory']
le = preprocessing.LabelEncoder()
for col in c:
    df[col] = le.fit_transform(df[col].values)
    df_test[col] = le.fit_transform(df_test[col].values)
```

In [21]:

```
x_train_return = df.drop(['fullVisitorId','target','ret'], axis = 1)
y_train_return = df['ret']

x_train_target = df[df['ret'] == 1].drop(['fullVisitorId','target','ret'], axis = 1)
y_train_target = df[df['ret'] == 1]['target']

x_test = df_test.drop(['fullVisitorId','target','ret'], axis = 1)
```

In [22]:

```
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 296530 entries, 0 to 296529
Data columns (total 27 columns):
channelGrouping                   296530 non-null int32
first_ses_from_the_period_start   296530 non-null int64
last_ses_from_the_period_end      296530 non-null int64
interval_dates                    296530 non-null int64
unique_date_num                   296530 non-null int64
maxVisitNum                       296530 non-null int64
```

```
browser                              296530 non-null int32
operatingSystem                      296530 non-null int32
deviceCategory                       296530 non-null int32
continent                            296530 non-null int32
subContinent                         296530 non-null int32
country                              296530 non-null int32
region                               296530 non-null int32
metro                                296530 non-null int32
city                                 296530 non-null int32
networkDomain                        296530 non-null int32
source                               296530 non-null int32
medium                               296530 non-null int32
isMobile                             296530 non-null float64
bounce_sessions                      296530 non-null int64
hits_sum                             296530 non-null int64
hits_mean                            296530 non-null float64
pageviews_sum                        296530 non-null int64
pageviews_mean                       296520 non-null float64
session_cnt                          296530 non-null int64
transactionRevenue                   296530 non-null float64
transactions                         296530 non-null int64
dtypes: float64(4), int32(13), int64(10)
memory usage: 46.4 MB
```

In [23]:

```
# Split train and validation data

# for target variable = return
x_train_return, x_valid_return, y_train_return, y_valid_return = train_test_split(x_train_return, y
_train_return, test_size=0.33)

# for target variable = target
x_train_target, x_valid_target, y_train_target, y_valid_target = train_test_split(x_train_target, y
_train_target, test_size=0.33)
```

In [676]:

```
# Normalized training dataset

# Normalize train_target
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(x_train_target)
x_train_target_scaled = scaler.transform(x_train_target)
x_valid_target_scaled = scaler.transform(x_valid_target)

# Normalize train_return
scaler = MinMaxScaler()
scaler.fit(x_train_return)
x_train_return_scaled = scaler.transform(x_train_return)
x_valid_return_scaled = scaler.transform(x_valid_return)

# Normalize test dataset
scaler = MinMaxScaler()
scaler.fit(x_test)
x_test_scaled = scaler.transform(x_test)
```

In [677]:

```
# Deal with nan values
np.argwhere(np.isnan(x_train_return_scaled))
x_train_return_scaled = np.nan_to_num(x_train_return_scaled)

np.argwhere(np.isnan(x_valid_return_scaled))
x_valid_return_scaled = np.nan_to_num(x_valid_return_scaled)

np.argwhere(np.isnan(x_test_scaled))
x_test_scaled = np.nan_to_num(x_test_scaled)
```

Model 1 : Random Forest

Using original features to see the model performance

In [678]:

```
# Download necessary library for random forest regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
```

In [680]:

```
# Build random forest classification model
clf = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=0)
clf.fit(x_train_return_scaled, y_train_return)
```

Out[680]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=3, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [679]:

```
# Build random forest regression model
reg = RandomForestRegressor(max_depth=10, n_estimators=100)
reg.fit(x_train_target_scaled, y_train_target)
```

Out[679]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [682]:

```
# Classification Report
y_pred_valid_return = clf.predict(x_valid_return_scaled)
print(classification_report(y_valid_return, y_pred_valid_return))
# Calculating R-square
print("Its R-square of random forest regression model is", reg.score(x_train_target_scaled,
y_train_target))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    464923
           1       0.00      0.00      0.00      2877

    accuracy                           0.99    467800
   macro avg       0.50      0.50      0.50    467800
weighted avg       0.99      0.99      0.99    467800

Its R-square of random forest regression model is 0.4786546091953813
```

In [683]:

```
# Calculating RMSE before tuning hyperparameters
y_pred_target = reg.predict(x_valid_target_scaled)
print("The RMSE of random forest regression model is",np.sqrt(mean_squared_error(y_valid_target, y
_pred_target)))
```

```
Its RMSE of random forest regression model is 4.00323956489714
```

Utilizing randomizedsearchCV to tune hyperparameters

In [684]:

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 500, num = 10)]

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 100, num = 11)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]

# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 233, 266, 300, 333, 366, 400, 433, 466, 500], 'max_features': ['auto',
'sqrt'], 'max_depth': [10, 19, 28, 37, 46, 55, 64, 73, 82, 91, 100, None], 'min_samples_split': [2
, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

In [685]:

```
# Use RandomizedSearchCV for best hyperparameters

# First create the base model to tune
reg2 = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
reg2_random = RandomizedSearchCV(estimator = reg2, param_distributions = random_grid, n_iter = 10,
cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
reg2_random.fit(x_train_target_scaled, y_train_target)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  1.2min finished
```

Out[685]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators='warn',
                                                   n_jobs=None, oob_score=False,
                                                   random_sta...
                   iid='warn', n_iter=10, n_jobs=-1,
```

```
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 19, 28, 37, 46, 55,
                                                      64, 73, 82, 91, 100,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 233, 266, 300,
                                                         333, 366, 400, 433,
                                                         466, 500]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [686]:

```
reg2_random.best_params_
```

Out[686]:

```
{'n_estimators': 200,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 10,
 'bootstrap': True}
```

In [687]:

```
best_random = RandomForestRegressor(n_estimators=200,
                                    min_samples_split=5,
                                    min_samples_leaf=2,
                                    max_features='sqrt',
                                    max_depth=10,
                                    bootstrap=True)

best_random.fit(x_train_target_scaled, y_train_target)
y_predF = best_random.predict(x_valid_target_scaled)
print("Its RMSE of tuned random forest model is",np.sqrt(mean_squared_error(y_valid_target,
y_predF)))
```

```
Its RMSE of tuned random forest model is 3.901847198007343
```

In [688]:

```
y_pred_target =  best_random.predict(x_test_scaled)
y_pred_target
```

Out[688]:

```
array([0.28238238, 0.3591359 , 0.02736908, ..., 0.25784802, 0.36411663,
       0.37251495])
```

In [689]:

```
df_test1 = pd.read_csv(r'C:\Users\ACER\Desktop\UMN\Predictive Analysis\FinalProject\data\test.csv'
,converters={'fullVisitorId': lambda x: str(x)})
```

In [691]:

```
#test
data = {'fullVisitorId': [i for i in df_test1.fullVisitorId], 'PredictedLogRevenue': [j for j in y_
pred_target]}
submission_random_forest = pd.DataFrame(data)
submission_random_forest
```

Out[691]:

	fullVisitorId	PredictedLogRevenue
0	0000018966949534117	0.282382

	fullVisitorId	PredictedLogRevenue
1	0000039738481224681	0.359136
2	0000073585230191399	0.027369
3	0000087588448856385	0.037537
4	0000149787903119437	0.004584
...	...	...
296525	9999862054614696520	0.964398
296526	9999898168621645223	0.036835
296527	999990167740728398	0.257848
296528	9999915620249883537	0.364117
296529	9999947552481876143	0.372515

296530 rows × 2 columns

In [692]:

```
import csv
submission_random_forest.to_csv(r'C:\Users\ACER\Desktop\UMN\Predictive Analysis\FinalProject\data\submission_random_forest.csv', index = False)
```

Utilizing feature selection and executing random forest model with tuned hyperparameters

In [693]:

```
#Feature Selection for classification
feature_importances_clf = pd.DataFrame(clf.feature_importances_,
                                       index = x_train_target.columns,
                                        columns=['importance']).sort_values('importance', ascending=Fal
e)
```

In [694]:

```
feature_importances_clf
```

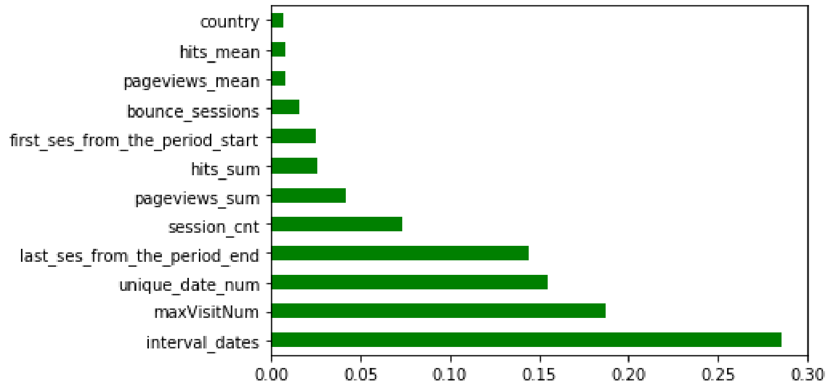
Out[694]:

	importance
interval_dates	0.285881
maxVisitNum	0.186963
unique_date_num	0.154433
last_ses_from_the_period_end	0.144428
session_cnt	0.072903
pageviews_sum	0.041733
hits_sum	0.025367
first_ses_from_the_period_start	0.025121
bounce_sessions	0.015758
pageviews_mean	0.008133
hits_mean	0.007734
country	0.007131
transactionRevenue	0.004848
source	0.004131
medium	0.003448
transactions	0.002705
region	0.002504
deviceCategory	0.001740
operatingSystem	0.001393
channelGrouping	0.001089

	importance
networkDomain	0.000966
city	0.000576
metro	0.000378
subContinent	0.000355
isMobile	0.000250
browser	0.000030
continent	0.000000

In [695]:

```
#plot graph of feature importances for better visualization
import matplotlib.pyplot as plt
feature_importances_clf2 = pd.Series(clf.feature_importances_, index=x_train_target.columns)
feature_importances_clf2.nlargest(12).plot(kind='barh',color='g')
plt.show()
```



In [697]:

```
feature_importances_clf[feature_importances_clf['importance']>= 0.05].index.tolist()
```

Out[697]:

```
['interval_dates',
 'maxVisitNum',
 'unique_date_num',
 'last_ses_from_the_period_end',
 'session_cnt']
```

In [698]:

```
#Feature Selection for classification
feature_importances_reg = pd.DataFrame(best_random.feature_importances_,
                                   index = x_train_target.columns,
                                    columns=['importance']).sort_values('importance', ascending=Fal
e)
```

In [699]:

```
feature_importances_reg
```

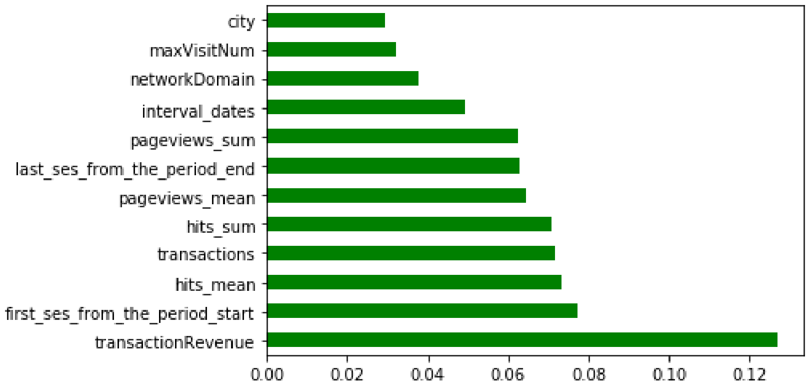
Out[699]:

	importance
transactionRevenue	0.127108
first_ses_from_the_period_start	0.077094
hits_mean	0.073249
transactions	0.071604
hits_sum	0.070696

	importance
pageviews_mean	0.064597
last_ses_from_the_period_end	0.063047
pageviews_sum	0.062329
interval_dates	0.049413
networkDomain	0.037815
maxVisitNum	0.032151
city	0.029512
session_cnt	0.026894
unique_date_num	0.025497
region	0.024016
operatingSystem	0.023880
metro	0.022023
channelGrouping	0.021965
bounce_sessions	0.018272
country	0.017622
source	0.014289
medium	0.013169
continent	0.009252
subContinent	0.007856
browser	0.006161
deviceCategory	0.005272
isMobile	0.005216

In [700]:

```
#plot graph of feature importances for better visualization
import matplotlib.pyplot as plt
feature_importances_reg2 = pd.Series(best_random.feature_importances_, index=x_train_target.columns
)
feature_importances_reg2.nlargest(12).plot(kind='barh',color='g')
plt.show()
```



In [701]:

```
feature_importances_reg[feature_importances_reg['importance']>= 0.05].index.tolist()
```

Out[701]:

```
['transactionRevenue',
 'first_ses_from_the_period_start',
 'hits_mean',
 'transactions',
 'hits_sum',
 'pageviews_mean',
 'last_ses_from_the_period_end',
 'pageviews_sum']
```

In [713]:

```
# Split train and validation data

# for target variable = return
x_train_return, x_valid_return, y_train_return, y_valid_return = train_test_split(x_train_return, y
_train_return, test_size=0.33)

# for target variable = target
x_train_target, x_valid_target, y_train_target, y_valid_target = train_test_split(x_train_target, y
_train_target, test_size=0.33)
```

In [714]:

```
# Select important features
x_train_return = x_train_return[['interval_dates',
                                  'maxVisitNum',
                                  'unique_date_num',
                                  'last_ses_from_the_period_end',
                                  'session_cnt']]
x_valid_return = x_valid_return[['interval_dates',
                                  'maxVisitNum',
                                  'unique_date_num',
                                  'last_ses_from_the_period_end',
                                  'session_cnt']]
x_train_target = x_train_target[['transactionRevenue',
                                  'first_ses_from_the_period_start',
                                  'hits_mean',
                                  'transactions',
                                  'hits_sum',
                                  'pageviews_mean',
                                  'last_ses_from_the_period_end',
                                  'pageviews_sum']]
x_valid_target = x_valid_target[['transactionRevenue',
                                  'first_ses_from_the_period_start',
                                  'hits_mean',
                                  'transactions',
                                  'hits_sum',
                                  'pageviews_mean',
                                  'last_ses_from_the_period_end',
                                  'pageviews_sum']]
x_test_return = x_test[['interval_dates',
                                  'maxVisitNum',
                                  'unique_date_num',
                                  'last_ses_from_the_period_end',
                                  'session_cnt']]

x_test_target = x_test[['transactionRevenue',
                                  'first_ses_from_the_period_start',
                                  'hits_mean',
                                  'transactions',
                                  'hits_sum',
                                  'pageviews_mean',
                                  'last_ses_from_the_period_end',
                                  'pageviews_sum']]
```

In [716]:

```
# Normalized training & testing dataset

# Normalize train_target
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(x_train_target)
x_train_target_scaled = scaler.transform(x_train_target)
x_valid_target_scaled = scaler.transform(x_valid_target)

# Normalize train_return
scaler = MinMaxScaler()
scaler.fit(x_train_return)
x_train_return_scaled = scaler.transform(x_train_return)
x_valid_return_scaled = scaler.transform(x_valid_return)

# Normalize test
```

```
scaler = MinMaxScaler()
scaler.fit(x_test_return)
x_test_return_scaled = scaler.transform(x_test_return)

scaler.fit(x_test_target)
x_test_target_scaled = scaler.transform(x_test_target)
```

In [717]:

```
# Dealing with NAN value
np.argwhere(np.isnan(x_train_return_scaled))
x_train_return_scaled = np.nan_to_num(x_train_return_scaled)

np.argwhere(np.isnan(x_valid_return_scaled))
x_valid_return_scaled = np.nan_to_num(x_valid_return_scaled)

np.argwhere(np.isnan(x_test_scaled))
x_test_return_scaled = np.nan_to_num(x_test_return_scaled)
x_test_target_scaled = np.nan_to_num(x_test_target_scaled)
```

In [718]:

```
# Build random forest model
# For classification model (target variable = return)
clf = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=0)
clf.fit(x_train_return_scaled, y_train_return)
```

Out[718]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=3, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [719]:

```
y_pred_valid_return = clf.predict(x_valid_return_scaled)
# Classification Report
print(classification_report(y_valid_return, y_pred_valid_return))

# For regression model (target variable = target)
reg = RandomForestRegressor(max_depth=10, n_estimators=100)
reg.fit(x_train_target_scaled, y_train_target)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     93699
           1       0.60      0.01      0.02       568

    accuracy                           0.99     94267
   macro avg       0.80      0.51      0.51     94267
weighted avg       0.99      0.99      0.99     94267
```

Out[719]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [720]:

```
# Calculating R-square
print("Its R-square of random forest regression model is", reg.score(x_train_target_scaled,
y_train_target))
```

```
Its R-square of random forest regression model is 0.722565384969677
```

In [721]:

```
best_random.fit(x_train_target_scaled, y_train_target)
y_predF_new = best_random.predict(x_valid_target_scaled)
print("Its RMSE of tuned random forest model is",np.sqrt(mean_squared_error(y_valid_target,
y_predF_new)))
```

```
Its RMSE of tuned random forest model is 3.474077726323697
```

In [722]:

```
y_pred_target_new = best_random.predict(x_test_target_scaled)
y_pred_target_new
```

Out[722]:

```
array([0.16788353, 0.16438741, 0.16438741, ..., 0.36746037, 0.25762755,
       0.53529752])
```

In [723]:

```
df_test1 = pd.read_csv(r'C:\Users\ACER\Desktop\UMN\Predictive Analysis\FinalProject\data\test.csv'
,converters={'fullVisitorId': lambda x: str(x)})
```

In [724]:

```
#test
data_random_forest = {'fullVisitorId': [i for i in df_test1.fullVisitorId], 'PredictedLogRevenue':
[j for j in y_pred_target_new]}
submission_random_forest_new = pd.DataFrame(data)
submission_random_forest_new
```

Out[724]:

	fullVisitorId	PredictedLogRevenue
0	0000018966949534117	0.282382
1	0000039738481224681	0.359136
2	0000073585230191399	0.027369
3	0000087588448856385	0.037537
4	0000149787903119437	0.004584
...	...	...
296525	9999862054614696520	0.964398
296526	9999898168621645223	0.036835
296527	999990167740728398	0.257848
296528	9999915620249883537	0.364117
296529	9999947552481876143	0.372515

296530 rows × 2 columns

In [725]:

```
submission_random_forest_new.to_csv(r'C:\Users\ACER\Desktop\UMN\Predictive
Analysis\FinalProject\data\submission_random_forest2.csv', index = False)
```

Model 2 : XGboost

In [726]:

```python
# Import necessary libraries
from sklearn import ensemble
import xgboost as xgb
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
```

In [727]:

```python
x_train_return = df.drop(['fullVisitorId','target','ret'], axis = 1)
y_train_return = df['ret']

x_train_target = df[df['ret'] == 1].drop(['fullVisitorId','target','ret'], axis = 1)
y_train_target = df[df['ret'] == 1]['target']

x_test = df_test.drop(['fullVisitorId','target','ret'], axis = 1)
```

In [728]:

```python
# Split train and validation data

# for target variable = return
x_train_return, x_valid_return, y_train_return, y_valid_return = train_test_split(x_train_return, y
_train_return, test_size=0.33)

# for target variable = target
x_train_target, x_valid_target, y_train_target, y_valid_target = train_test_split(x_train_target, y
_train_target, test_size=0.33)
```

In [729]:

```python
# Normalized training & testing dataset

# Normalize train_target
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(x_train_target)
x_train_target_scaled = scaler.transform(x_train_target)
x_valid_target_scaled = scaler.transform(x_valid_target)

# Normalize train_return
scaler = MinMaxScaler()
scaler.fit(x_train_return)
x_train_return_scaled = scaler.transform(x_train_return)
x_valid_return_scaled = scaler.transform(x_valid_return)

# Normalize test
scaler = MinMaxScaler()
scaler.fit(x_test)
x_test_scaled = scaler.transform(x_test)
```

In [730]:

```python
# Dealing with NAN value
np.argwhere(np.isnan(x_train_return_scaled))
x_train_return_scaled = np.nan_to_num(x_train_return_scaled)

np.argwhere(np.isnan(x_valid_return_scaled))
x_valid_return_scaled = np.nan_to_num(x_valid_return_scaled)

np.argwhere(np.isnan(x_test_scaled))
x_test_scaled = np.nan_to_num(x_test_scaled)
```

In [731]:

```python
# XGboost classification
clf = xgb.XGBClassifier(max_depth=10,
                        min_child_weight=1,
                        learning_rate=0.1,
                        n_estimators=500,
                        silent=True,
                        objective='binary:logistic',
                        gamma=0,
```

```
                              max_delta_step=0,
                              subsample=1,
                              colsample_bytree=1,
                              colsample_bylevel=1,
                              reg_alpha=0,
                              reg_lambda=0,
                              scale_pos_weight=1,
                              seed=1,
                              missing=None)
clf.fit(x_train_return_scaled, y_train_return, eval_metric='auc', verbose=True,
            eval_set=[(x_valid_return_scaled, y_valid_return)], early_stopping_rounds=5)

y_pred_valid_return = clf.predict(x_valid_return_scaled)
```

```
[0] validation_0-auc:0.840764
Will train until validation_0-auc hasn't improved in 5 rounds.
[1] validation_0-auc:0.867023
[2] validation_0-auc:0.876627
[3] validation_0-auc:0.877874
[4] validation_0-auc:0.877972
[5] validation_0-auc:0.879299
[6] validation_0-auc:0.879304
[7] validation_0-auc:0.880145
[8] validation_0-auc:0.881351
[9] validation_0-auc:0.883728
[10] validation_0-auc:0.884225
[11] validation_0-auc:0.884786
[12] validation_0-auc:0.884822
[13] validation_0-auc:0.884727
[14] validation_0-auc:0.884264
[15] validation_0-auc:0.884703
[16] validation_0-auc:0.884399
[17] validation_0-auc:0.884525
Stopping. Best iteration:
[12] validation_0-auc:0.884822
```

In [732]:

```
# CLassification Report
print(classification_report(y_valid_return, y_pred_valid_return))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    464957
           1       0.43      0.02      0.04      2843

    accuracy                           0.99    467800
   macro avg       0.71      0.51      0.52    467800
weighted avg       0.99      0.99      0.99    467800
```

In [754]:

```
# XGboost Regression
xg_reg = xgb.XGBRegressor(learning_rate=0.1,max_depth=10)
```

In [755]:

```
# fit the model with the training data
xg_reg.fit(x_train_target_scaled, y_train_target)
```

```
[21:20:52] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[755]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
```

```
             max_depth=10, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

In [756]:

```
# Calculating R-square
print("Its R-square of XGboost regression model is", xg_reg.score(x_train_target_scaled, y_train_target))

# Calculating RMSE for regression
y_pred_valid_target = xg_reg.predict(x_valid_target_scaled)
print("Its RMSE of XGboost regression model is",np.sqrt(mean_squared_error(y_pred_valid_target, y_valid_target)))
```

```
Its R-square of XGboost regression model is 0.9468226969141958
Its RMSE of XGboost regression model is 4.407437111752069
```

In [757]:

```
y_pred = xg_reg.predict(x_test_scaled)
```

In [758]:

```
y_pred
```

Out[758]:

```
array([ 0.4664757 ,  4.687464  ,  0.06096366, ...,  0.00505006,
        0.02956718, -0.2237975 ], dtype=float32)
```

In [759]:

```
df_test1 = pd.read_csv(r'C:\Users\ACER\Desktop\UMN\Predictive Analysis\FinalProject\data\test.csv'
,converters={'fullVisitorId': lambda x: str(x)})
```

In [760]:

```
data_xgboost = {'fullVisitorId': [i for i in df_test1.fullVisitorId], 'PredictedLogRevenue': [j for j in y_pred]}
submission_xgboost = pd.DataFrame(data_xgboost)
submission_xgboost
```

Out[760]:

	fullVisitorId	PredictedLogRevenue
0	0000018966949534117	0.466476
1	0000039738481224681	4.687464
2	0000073585230191399	0.060964
3	0000087588448856385	-0.021676
4	0000149787903119437	0.437586
...	...	...
296525	9999862054614696520	1.720398
296526	9999898168621645223	-0.039090
296527	999990167740728398	0.005050
296528	9999915620249883537	0.029567
296529	9999947552481876143	-0.223798

296530 rows × 2 columns

In [761]:

```
submission_xgboost.to_csv(r'C:\Users\ACER\Desktop\UMN\Predictive Analysis\FinalProject\data\submission_xgboost.csv', index = False)
```

In [762]:

```
submission_xgboost
```

Out[762]:

	fullVisitorId	PredictedLogRevenue
0	0000018966949534117	0.466476
1	0000039738481224681	4.687464
2	0000073585230191399	0.060964
3	0000087588448856385	-0.021676
4	0000149787903119437	0.437586
...	...	...
296525	9999862054614696520	1.720398
296526	9999898168621645223	-0.039090
296527	999990167740728398	0.005050
296528	9999915620249883537	0.029567
296529	9999947552481876143	-0.223798

296530 rows × 2 columns

Utilizing feature selection and executing XGboost model again

In [746]:

```
#Feature Selection for classification
feature_importances_xg_clf = pd.DataFrame(clf.feature_importances_,
                                    index = x_train_target.columns,
                                     columns=['importance']).sort_values('importance', ascending=Fal
e)
```

In [747]:

```
feature_importances_xg_clf
```

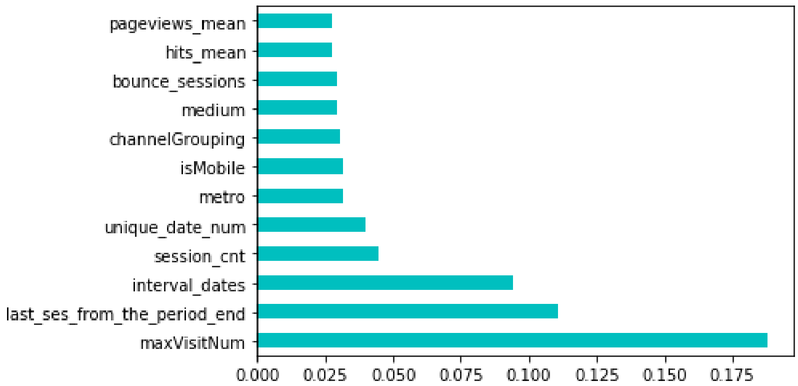
Out[747]:

	importance
maxVisitNum	0.188062
last_ses_from_the_period_end	0.110923
interval_dates	0.094427
session_cnt	0.044560
unique_date_num	0.039723
metro	0.031840
isMobile	0.031498
channelGrouping	0.030286
medium	0.029387
bounce_sessions	0.029100
hits_mean	0.027721
pageviews_mean	0.027500
hits_sum	0.026518
region	0.026465
pageviews_sum	0.025819

	importance
source	0.023397
transactionRevenue	0.023313
city	0.022876
first_ses_from_the_period_start	0.022041
networkDomain	0.020110
operatingSystem	0.019942
country	0.019692
transactions	0.019419
subContinent	0.018837
deviceCategory	0.018295
browser	0.015792
continent	0.012458

In [748]:

```
#plot graph of feature importances for better visualization
import matplotlib.pyplot as plt
feature_importances_xg_clf2 = pd.Series(clf.feature_importances_, index=x_train_target.columns)
feature_importances_xg_clf2.nlargest(12).plot(kind='barh',color='c')
plt.show()
```



In [749]:

```
# classification
feature_importances_xg_clf[feature_importances_xg_clf['importance']>= 0.035].index.tolist()
```

Out[749]:

```
['maxVisitNum',
 'last_ses_from_the_period_end',
 'interval_dates',
 'session_cnt',
 'unique_date_num']
```

In [767]:

```
#Feature Selection for regression
feature_importances_xg_reg = pd.DataFrame(xg_reg.feature_importances_,
                                    index = x_train_target.columns,
                                     columns=['importance']).sort_values('importance', ascending=Fal
e)
```

In [764]:

```
feature_importances_xg_reg
```

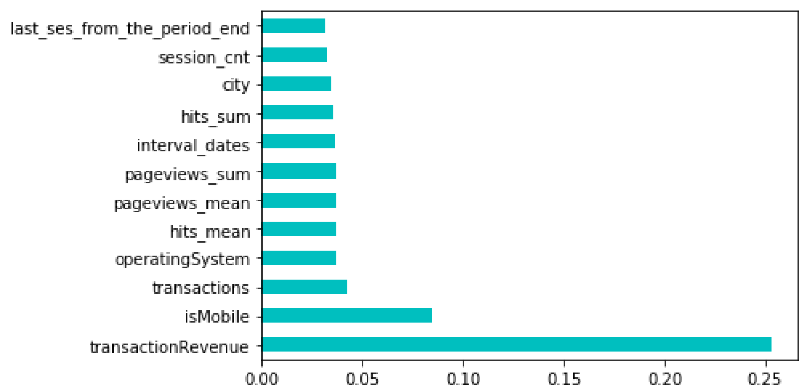
Out[764]:

	importance

	importance
transactionRevenue	0.253467
isMobile	0.085102
transactions	0.042302
operatingSystem	0.037384
hits_mean	0.037252
pageviews_mean	0.037240
pageviews_sum	0.036746
interval_dates	0.036145
hits_sum	0.035135
city	0.034437
session_cnt	0.032452
last_ses_from_the_period_end	0.031687
bounce_sessions	0.031650
metro	0.029143
region	0.026959
medium	0.026599
first_ses_from_the_period_start	0.025814
source	0.024834
unique_date_num	0.024397
maxVisitNum	0.023539
networkDomain	0.023277
country	0.019102
channelGrouping	0.014901
browser	0.011254
continent	0.009093
deviceCategory	0.006758
subContinent	0.003363

In [765]:

```
#plot graph of feature importances for better visualization
import matplotlib.pyplot as plt
feature_importances_xg_reg = pd.Series(xg_reg.feature_importances_, index=x_train_target.columns)
feature_importances_xg_reg.nlargest(12).plot(kind='barh',color='c')
plt.show()
```



In [768]:

```
# regression
feature_importances_xg_reg[feature_importances_xg_reg['importance']>= 0.035].index.tolist()
```

Out[768]:

```
['transactionRevenue',
```

```
 'isMobile',
 'transactions',
 'operatingSystem',
 'hits_mean',
 'pageviews_mean',
 'pageviews_sum',
 'interval_dates',
 'hits_sum']
```

In [769]:

```
# Split train and validation data
# for target variable = return
x_train_return, x_valid_return, y_train_return, y_valid_return = train_test_split(x_train_return, y
_train_return, test_size=0.33)

# for target variable = target
x_train_target, x_valid_target, y_train_target, y_valid_target = train_test_split(x_train_target, y
_train_target, test_size=0.33)
```

In [770]:

```
# Select important features for classification
x_train_return = x_train_return[['maxVisitNum',
                                  'last_ses_from_the_period_end',
                                  'interval_dates',
                                  'session_cnt',
                                  'unique_date_num']]
x_valid_return = x_valid_return[['maxVisitNum',
                                  'last_ses_from_the_period_end',
                                  'interval_dates',
                                  'session_cnt',
                                  'unique_date_num']]
x_test_return = x_test[['maxVisitNum',
                                  'last_ses_from_the_period_end',
                                  'interval_dates',
                                  'session_cnt',
                                  'unique_date_num']]
# Select important features for regression
x_train_target = x_train_target[['transactionRevenue',
                                  'isMobile',
                                  'transactions',
                                  'operatingSystem',
                                  'hits_mean',
                                  'pageviews_mean',
                                  'pageviews_sum',
                                  'interval_dates',
                                  'hits_sum']]
x_valid_target = x_valid_target[['transactionRevenue',
                                  'isMobile',
                                  'transactions',
                                  'operatingSystem',
                                  'hits_mean',
                                  'pageviews_mean',
                                  'pageviews_sum',
                                  'interval_dates',
                                  'hits_sum']]

x_test_target = x_test[['transactionRevenue',
                                  'isMobile',
                                  'transactions',
                                  'operatingSystem',
                                  'hits_mean',
                                  'pageviews_mean',
                                  'pageviews_sum',
                                  'interval_dates',
                                  'hits_sum']]
```

In [771]:

```
# Normalized training & testing dataset

# Normalize train_target
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
scaler.fit(x_train_target)
x_train_target_scaled = scaler.transform(x_train_target)
x_valid_target_scaled = scaler.transform(x_valid_target)

# Normalize train_return
scaler = MinMaxScaler()
scaler.fit(x_train_return)
x_train_return_scaled = scaler.transform(x_train_return)
x_valid_return_scaled = scaler.transform(x_valid_return)

# Normalize test
scaler = MinMaxScaler()
scaler.fit(x_test_target)
x_test_target_scaled = scaler.transform(x_test_target)
scaler.fit(x_test_return)
x_test_return_scaled = scaler.transform(x_test_return)
```

In [772]:

```
# Dealing with NAN value
np.argwhere(np.isnan(x_train_return_scaled))
x_train_return_scaled = np.nan_to_num(x_train_return_scaled)

np.argwhere(np.isnan(x_valid_return_scaled))
x_valid_return_scaled = np.nan_to_num(x_valid_return_scaled)

np.argwhere(np.isnan(x_test_scaled))
x_test_return_scaled = np.nan_to_num(x_test_return_scaled)
x_test_target_scaled = np.nan_to_num(x_test_target_scaled)
```

In [773]:

```
# XGboost classification
clf = xgb.XGBClassifier(max_depth=10,
                            min_child_weight=1,
                            learning_rate=0.1,
                            n_estimators=500,
                            silent=True,
                            objective='binary:logistic',
                            gamma=0,
                            max_delta_step=0,
                            subsample=1,
                            colsample_bytree=1,
                            colsample_bylevel=1,
                            reg_alpha=0,
                            reg_lambda=0,
                            scale_pos_weight=1,
                            seed=1,
                            missing=None)
clf.fit(x_train_return_scaled, y_train_return, eval_metric='auc', verbose=True,
            eval_set=[(x_valid_return_scaled, y_valid_return)], early_stopping_rounds=5)

y_pred_valid_return = clf.predict(x_valid_return_scaled)
```

```
[0] validation_0-auc:0.790371
Will train until validation_0-auc hasn't improved in 5 rounds.
[1] validation_0-auc:0.811361
[2] validation_0-auc:0.820967
[3] validation_0-auc:0.823629
[4] validation_0-auc:0.823931
[5] validation_0-auc:0.824745
[6] validation_0-auc:0.825439
[7] validation_0-auc:0.82547
[8] validation_0-auc:0.826124
[9] validation_0-auc:0.827535
[10] validation_0-auc:0.829742
[11] validation_0-auc:0.830777
[12] validation_0-auc:0.831048
[13] validation_0-auc:0.830361
[14] validation_0-auc:0.831438
[15] validation_0-auc:0.831822
[16] validation_0-auc:0.832153
[17] validation_0-auc:0.832076
[18] validation_0-auc:0.832266
```

```
[18] validation_0-auc:0.832266
[19] validation_0-auc:0.831917
[20] validation_0-auc:0.83158
[21] validation_0-auc:0.831914
[22] validation_0-auc:0.831441
[23] validation_0-auc:0.831114
Stopping. Best iteration:
[18] validation_0-auc:0.832266
```

In [774]:

```
# CLassification Report
print(classification_report(y_valid_return, y_pred_valid_return))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    311474
           1       0.36      0.02      0.04      1952

    accuracy                           0.99    313426
   macro avg       0.68      0.51      0.52    313426
weighted avg       0.99      0.99      0.99    313426
```

In [775]:

```
from sklearn.metrics import accuracy_score, confusion_matrix,
matthews_corrcoef,classification_report,roc_curve, auc, mean_squared_error, make_scorer, f1_score
from math import sqrt
from sklearn.model_selection import cross_val_score, train_test_split, KFold, GridSearchCV,
RandomizedSearchCV

xgb_estimator = xgb.XGBRegressor()

param_grid = {'max_depth': range(15),
              'n_estimators ':[50,100,150],
              'learning_rate': [0.01, 0.1,0.5,1]}
xgb_new = RandomizedSearchCV(xgb_estimator, param_distributions = param_grid, cv=3, verbose = 2)
xgb_new.fit(x_train_target_scaled, y_train_target)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] n_estimators =50, max_depth=8, learning_rate=0.1 ................
[21:26:34] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] . n_estimators =50, max_depth=8, learning_rate=0.1, total=   0.9s
[CV] n_estimators =50, max_depth=8, learning_rate=0.1 ................
[21:26:35] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.8s remaining:    0.0s
```

```
[CV] . n_estimators =50, max_depth=8, learning_rate=0.1, total=   0.4s
[CV] n_estimators =50, max_depth=8, learning_rate=0.1 ................
[21:26:35] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV] . n_estimators =50, max_depth=8, learning_rate=0.1, total=   0.3s
[CV] n_estimators =100, max_depth=12, learning_rate=0.1 .............
[21:26:36] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =100, max_depth=12, learning_rate=0.1, total=   0.5s
[CV] n_estimators =100, max_depth=12, learning_rate=0.1 .............
[21:26:36] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

```
reg:squarederror.
[CV]  n_estimators =100, max_depth=12, learning_rate=0.1, total=   0.6s
[CV] n_estimators =100, max_depth=12, learning_rate=0.1 ..............
[21:26:37] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =100, max_depth=12, learning_rate=0.1, total=   0.5s
[CV] n_estimators =150, max_depth=4, learning_rate=0.5 ...............
[21:26:37] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=4, learning_rate=0.5, total=   0.2s
[CV] n_estimators =150, max_depth=4, learning_rate=0.5 ...............
[21:26:37] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=4, learning_rate=0.5, total=   0.2s
[CV] n_estimators =150, max_depth=4, learning_rate=0.5 ...............
[21:26:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=4, learning_rate=0.5, total=   0.2s
[CV] n_estimators =100, max_depth=0, learning_rate=1 .................
[21:26:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV] .. n_estimators =100, max_depth=0, learning_rate=1, total=   0.1s
[CV] n_estimators =100, max_depth=0, learning_rate=1 .................
[21:26:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV] .. n_estimators =100, max_depth=0, learning_rate=1, total=   0.1s
[CV] n_estimators =100, max_depth=0, learning_rate=1 .................
[21:26:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV] .. n_estimators =100, max_depth=0, learning_rate=1, total=   0.1s
[CV] n_estimators =150, max_depth=9, learning_rate=0.5 ...............
[21:26:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=9, learning_rate=0.5, total=   0.4s
[CV] n_estimators =150, max_depth=9, learning_rate=0.5 ...............
[21:26:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=9, learning_rate=0.5, total=   0.4s
[CV] n_estimators =150, max_depth=9, learning_rate=0.5 ...............
[21:26:39] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=9, learning_rate=0.5, total=   0.4s
[CV] n_estimators =150, max_depth=8, learning_rate=0.1 ...............
[21:26:39] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=8, learning_rate=0.1, total=   0.3s
[CV] n_estimators =150, max_depth=8, learning_rate=0.1 ...............
[21:26:40] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=8, learning_rate=0.1, total=   0.4s
[CV] n_estimators =150, max_depth=8, learning_rate=0.1 ...............
[21:26:40] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =150, max_depth=8, learning_rate=0.1, total=   0.3s
[CV] n_estimators =50, max_depth=6, learning_rate=0.01 ...............
[21:26:40] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =50, max_depth=6, learning_rate=0.01, total=   0.3s
[CV] n_estimators =50, max_depth=6, learning_rate=0.01 ...............
[21:26:41] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
[CV]  n_estimators =50, max_depth=6, learning_rate=0.01, total=   0.3s
[CV] n_estimators =50, max_depth=6, learning_rate=0.01
```

```
[CV] n_estimators =50, max_depth=6, learning_rate=0.01 ...............
[21:26:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =50, max_depth=6, learning_rate=0.01, total=   0.3s
[CV] n_estimators =50, max_depth=4, learning_rate=0.5 ................
[21:26:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] . n_estimators =50, max_depth=4, learning_rate=0.5, total=   0.2s
[CV] n_estimators =50, max_depth=4, learning_rate=0.5 ................
[21:26:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] . n_estimators =50, max_depth=4, learning_rate=0.5, total=   0.2s
[CV] n_estimators =50, max_depth=4, learning_rate=0.5 ................
[21:26:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] . n_estimators =50, max_depth=4, learning_rate=0.5, total=   0.2s
[CV] n_estimators =100, max_depth=0, learning_rate=0.5 ..............
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =100, max_depth=0, learning_rate=0.5, total=   0.1s
[CV] n_estimators =100, max_depth=0, learning_rate=0.5 ..............
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =100, max_depth=0, learning_rate=0.5, total=   0.1s
[CV] n_estimators =100, max_depth=0, learning_rate=0.5 ..............
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =100, max_depth=0, learning_rate=0.5, total=   0.1s
[CV] n_estimators =150, max_depth=2, learning_rate=0.01 .............
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =150, max_depth=2, learning_rate=0.01, total=   0.1s
[CV] n_estimators =150, max_depth=2, learning_rate=0.01 .............
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =150, max_depth=2, learning_rate=0.01, total=   0.1s
[CV] n_estimators =150, max_depth=2, learning_rate=0.01 .............
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  n_estimators =150, max_depth=2, learning_rate=0.01, total=   0.1s
[21:26:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    8.2s finished
```

Out[775]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          importance_type='gain',
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100,
                                          n_jobs=1, nthread=None,
                                          objective='reg:linear',
                                          random_state=0, reg_alpha=0,
                                          reg_lambda=1, scale_pos_weight=1,
                                          seed=None, silent=None, subsample=1,
                                          verbosity=1),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'learning_rate': [0.01, 0.1, 0.5, 1],
                                        'max_depth': range(0, 15),
                                        'n_estimators ': [50, 100, 150]},
```

```
                   n_estimators': [50, 100, 150]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [776]:

```
xgb_best = xgb_new.best_params_
print('Best parameters found by grid search are:', xgb_best)
```

```
Best parameters found by grid search are: {'n_estimators ': 150, 'max_depth': 2, 'learning_rate':
0.01}
```

In [777]:

```
xgb_best = xgb.XGBRegressor(n_estimators = 150, max_depth = 2, learning_rate = 0.01)
xgb_best.fit(x_train_target_scaled, y_train_target)
# Calculating R-square
print("Its R-square of XGboost model is", xgb_best.score(x_train_target_scaled, y_train_target))

# Calculating RMSE for regression
y_pred_valid_target = xgb_best.predict(x_valid_target_scaled)
print("Its RMSE of XGboost model is",np.sqrt(mean_squared_error(y_pred_valid_target,
y_valid_target)))
```

```
[21:27:01] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
Its R-square of XGboost model is 0.12087742759953268
Its RMSE of XGboost model is 4.134724483858594
```

In [778]:

```
y_pred_new = xgb_best.predict(x_test_target_scaled)
```

In [779]:

```
y_pred_new
```

Out[779]:

```
array([0.68039453, 0.49469256, 0.38342154, ..., 0.49469256, 0.5691235 ,
       0.54492813], dtype=float32)
```

In [780]:

```
df_test1 = pd.read_csv(r'C:\Users\ACER\Desktop\UMN\Predictive Analysis\FinalProject\data\test.csv'
,converters={'fullVisitorId': lambda x: str(x)})
```

In [781]:

```
#test
data_xgboost_new = {'fullVisitorId': [i for i in df_test1.fullVisitorId], 'PredictedLogRevenue': [j
for j in y_pred_new]}
submission_xgboost_new = pd.DataFrame(data_xgboost_new)
submission_xgboost_new
```

Out[781]:

	fullVisitorId	PredictedLogRevenue
0	0000018966949534117	0.680395
1	0000039738481224681	0.494693
2	0000073585230191399	0.383422
3	0000087588448856385	0.544928
4	0000149787903119437	0.494693
...	...	...

	fullVisitorId	PredictedLogRevenue
296525	9999862054614696520	0.680395
296526	9999898168621645223	0.407617
296527	999990167740728398	0.494693
296528	9999915620249883537	0.569124
296529	9999947552481876143	0.544928

296530 rows × 2 columns

In [ ]:

```
submission_xgboost_new.to_csv(r'C:\Users\ACER\Desktop\UMN\Predictive
Analysis\FinalProject\data\submission_xgboost2.csv', index = False)
```

GA_lightgbm

In [1]:

```
import pandas as pd
# conda install -c conda-forge lightgbm
import lightgbm as lgb
import numpy as np
from sklearn.model_selection import train_test_split
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/__init__.py:48: UserWarning: Starting from version
2.2.1, the library file in distribution wheels for macOS is built by the Apple Clang (Xcode_8.3.3)
compiler.
This means that in case of installing LightGBM from PyPI via the ``pip install lightgbm`` command,
you don't need to install the gcc compiler anymore.
Instead of that, you need to install the OpenMP library, which is required for running LightGBM on
the system with the Apple Clang compiler.
You can install the OpenMP library by the following command: ``brew install libomp``.
  "You can install the OpenMP library by the following command: ``brew install libomp``.", UserWar
ning)
```

In [2]:

```
train = pd.read_csv('train_final.csv')
test = pd.read_csv('test_final.csv')
```

```
/anaconda3/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3049: DtypeWarning:
Columns (1) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [ ]:

```
pd.set_option('display.max_columns', 99)
train.head()
```

In [3]:

```
drops = ['isVideoAd_mean','Unnamed: 0','isTrueDirect','hits_min', 'hits_max',
         'hits_median','hits_sd', 'pageviews_min', 'pageviews_max','pageviews_median',
         'pageviews_sd']

train = train.drop(columns = drops)
test = test.drop(columns = drops)
```

In [4]:

```
cat = ['channelGrouping','browser','operatingSystem', 'deviceCategory', 'continent',
       'subContinent', 'country', 'region', 'metro', 'city', 'networkDomain', 'source','medium']

train[cat] = train[cat].astype('category')
```

```
test[cat] = test[cat].astype('category')
```

In [29]:

```
X_train_all = train.drop(['fullVisitorId','target','ret'], axis = 1)
y_train_all = train['ret']

X_train_ret = train[train['ret'] == 1].drop(['fullVisitorId','target','ret'], axis = 1)
y_train_ret = train[train['ret'] == 1]['target']

X_test = test.drop(['fullVisitorId','target','ret'], axis = 1)
```

In [30]:

```
X_train_re, X_test_re, y_train_re, y_test_re = train_test_split(X_train_ret, y_train_ret, test_size
=0.33, random_state=42)
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X_train_all, y_train_all, test_
size=0.33, random_state=42)
```

Hyperparameters Tuning

In [ ]:

```
estimator = lgb.LGBMClassifier()

param_grid = {'num_leaves': [40,50,70,90], 'bagging_fraction ' : [0.5,0.7,0.9], 'bagging_freq ': [9
, 10, 11],
    'feature_fraction': [0.4, 0.5, 0.6],
    'learning_rate': [0.001, 0.01, 0.1]
}

lbm_clf = RandomizedSearchCV(estimator, param_distributions = param_grid, cv=5, scoring = 'neg_log_
loss')
lbm_clf.fit(X_train_clf, y_train_clf)
print(lbm_clf.best_params_)
```

Train LightGBM Model

In [32]:

```
dtrain_clf = lgb.Dataset(X_train_clf, label = y_train_clf)
dtest_clf = lgb.Dataset(X_test_clf, label = y_test_clf)
dtrain_re = lgb.Dataset(X_train_re, label = y_train_re)
dtest_re = lgb.Dataset(X_test_re, label = y_test_re)
```

In [33]:

```
#Parameters of "isReturned" classificator
param_lgb2 = {"objective":"binary",
              "num_leaves" : 40,
              "learning_rate" : 0.1,
              "bagging_fraction" : 0.5,
              "feature_fraction" : 0.4,
              "bagging_frequency" : 11,
              "metric": "binary_logloss"}


#Parameters of "How_Much_Returned_Will_Pay" regressor
param_lgb3= {"objective" : "regression",
             "metric" : "rmse",
             "num_leaves" : 50,
             "learning_rate" : 0.01,
             "bagging_fraction" : 0.7,
             "feature_fraction" : 0.6,
             "bagging_frequency" : 9}
```

In [ ]:

```
re_model = lgb.train(param_lgb3, dtrain_re, 1200, valid_sets = [dtest_re],
                     verbose_eval=20, early_stopping_rounds=700, categorical_feature = cat)
```

In [35]:

```
X_train_clf = lgb.Dataset(X_train_all, label = y_train_all)
X_train_re = lgb.Dataset(X_train_re, label = y_train_re)
```

In [37]:

```
pr_lgb_sum = 0
for i in range(10):
    lgb_model1 = lgb.train(param_lgb2, X_train_clf, 65)
    pr_lgb = lgb_model1.predict(X_test)
    lgb_model2 = lgb.train(param_lgb3, X_train_re,  168)
    pr_lgb_ret = lgb_model2.predict(X_test)
    pr_lgb_sum = pr_lgb_sum + pr_lgb * pr_lgb_ret

pr_final2 = (pr_lgb_sum / 10).round(20)
```

In [ ]:

```
df = pd.read_csv('test (1).csv', nrows=1) # Just take the first row to extract the columns' names
col_str_dic = {column:str for column in list(df)}
df = pd.read_csv('test (1).csv', dtype=col_str_dic)

data = {'fullVisitorId': [i for i in df.fullVisitorId], 'PredictedLogRevenue': [j for j in pr_final
2]}
newsub = pd.DataFrame(data)

newsub.to_csv("submission_lgb.csv",index=False)
```

In [ ]:

```
newsub
```

Feature importance

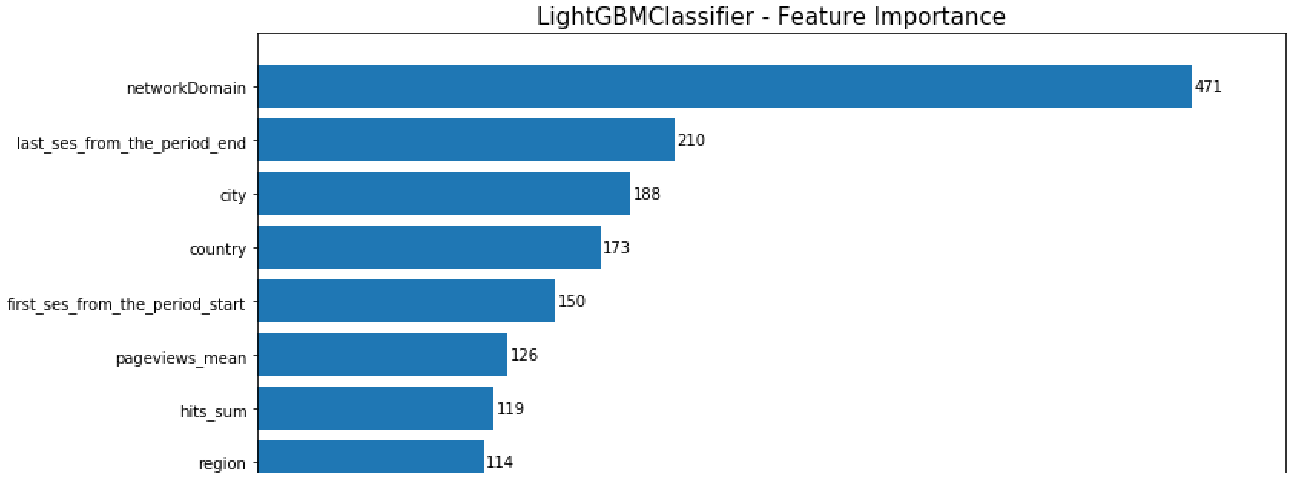
In [39]:

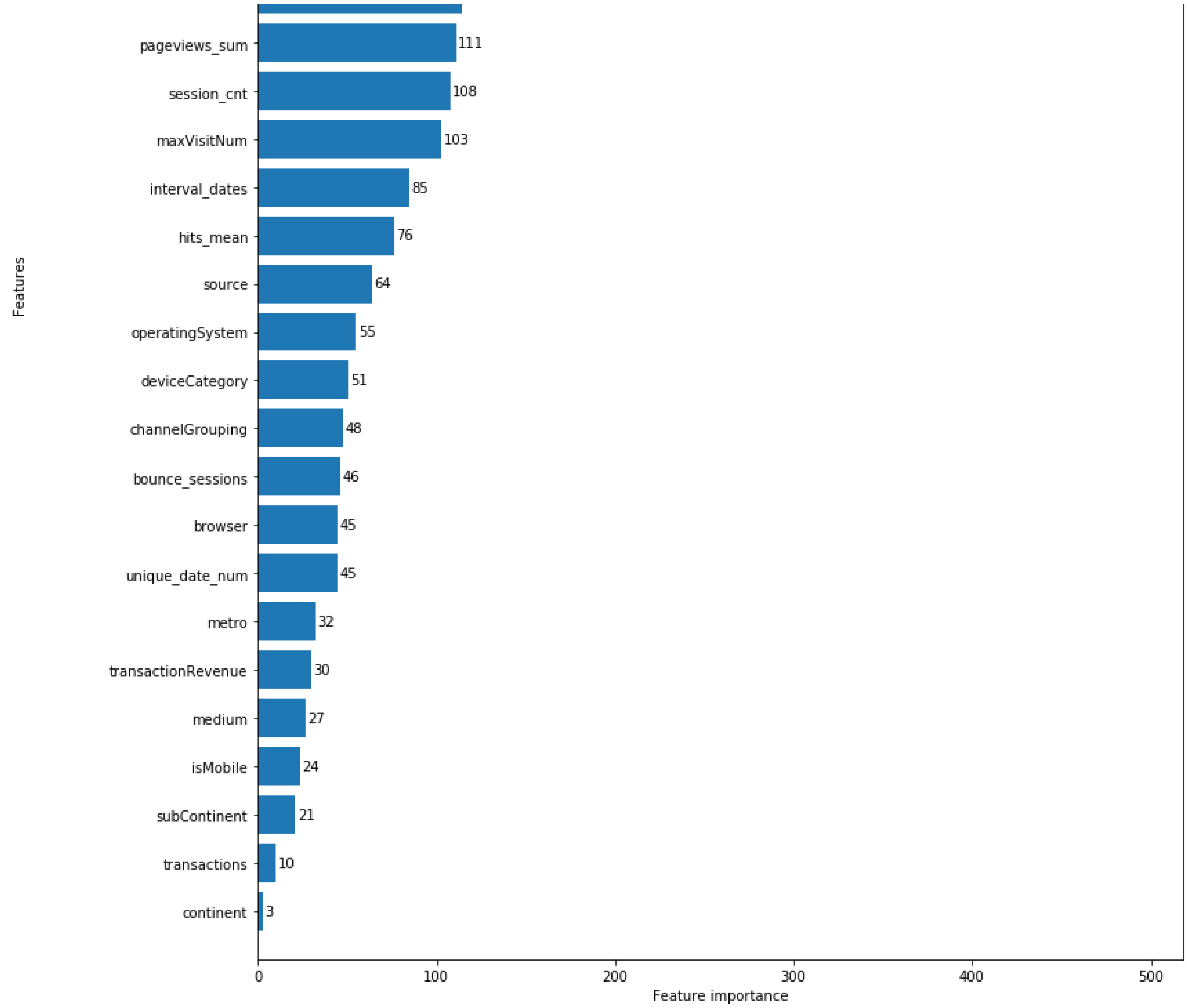
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12,18))
lgb.plot_importance(lgb_model1, max_num_features=50, height=0.8, ax=ax)
ax.grid(False)
plt.title("LightGBMClassifier - Feature Importance", fontsize=15)
plt.show
```

Out[39]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



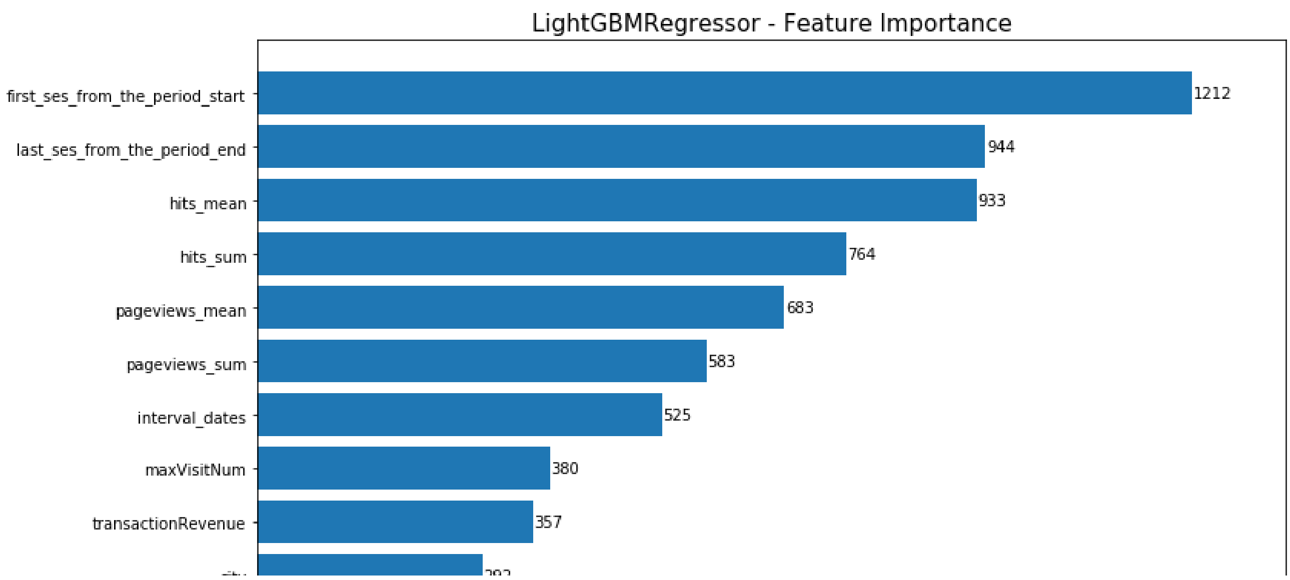


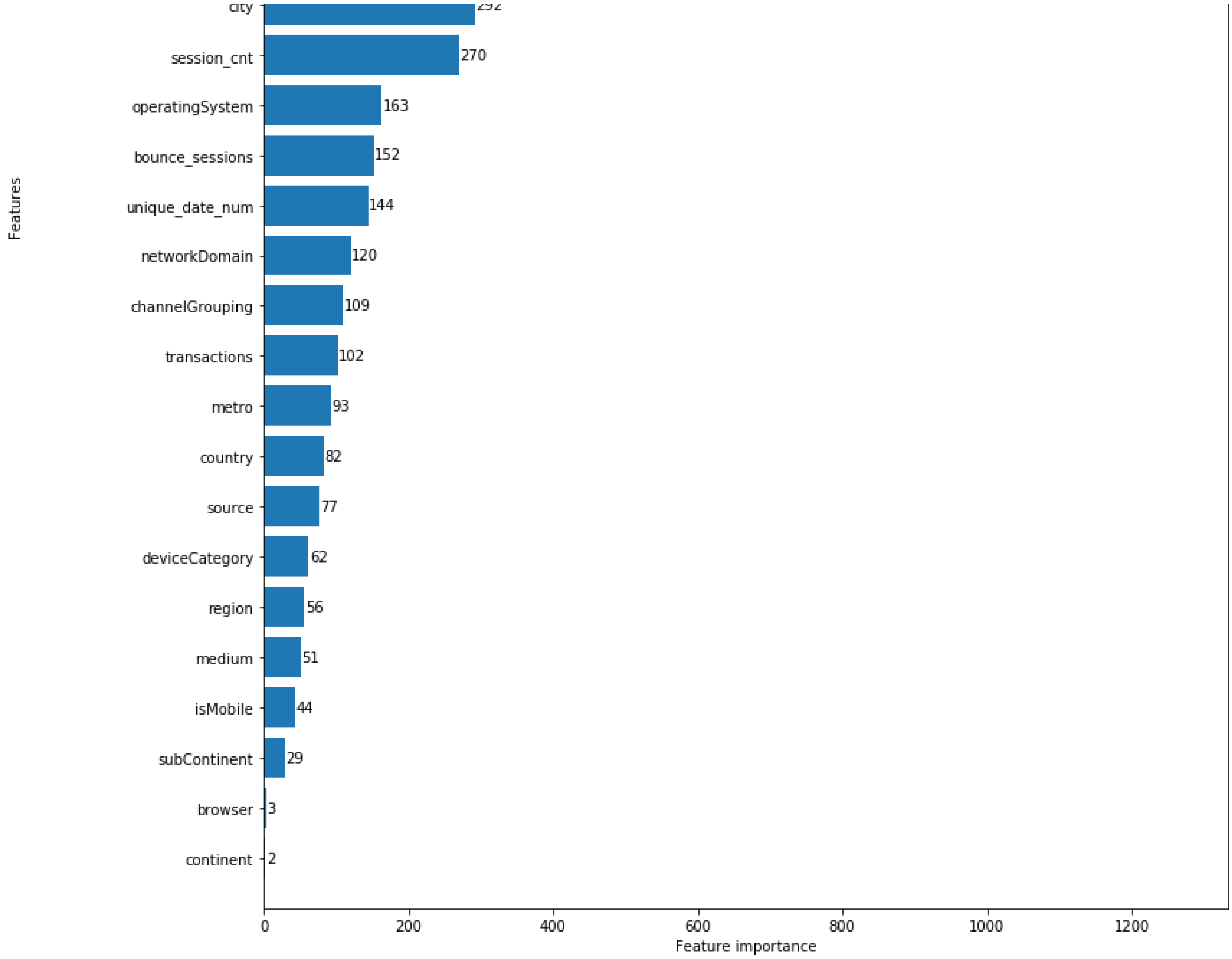
In [40]:

```
fig, ax = plt.subplots(figsize=(12,18))
lgb.plot_importance(lgb_model2, max_num_features=50, height=0.8, ax=ax)
ax.grid(False)
plt.title("LightGBMRegressor - Feature Importance", fontsize=15)
plt.show
```

Out[40]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```





Feature Selection

In [12]:

```
# For CLassification
important_features_for_clf = ['networkDomain',
'last_ses_from_the_period_end','city','country','first_ses_from_the_period_start',
'pageviews_mean','hits_sum', 'region','pageviews_sum','session_cnt','maxVisitNum']

X_train_all = X_train_all[important_features_for_clf]

X_test_clf_fs = X_test[important_features_for_clf]
```

In [13]:

```
# For Regression
important_features_for_re = ['first_ses_from_the_period_start',
                             'last_ses_from_the_period_end', 'hits_sum', 'pageviews_sum',
                             'hits_mean', 'pageviews_mean', 'interval_dates',
                             'transactionRevenue', 'maxVisitNum', 'city']

X_train_ret = X_train_ret[important_features_for_re]
X_test_re_fs = X_test[important_features_for_re]
```

In [14]:

```
X_train_re, X_test_re, y_train_re, y_test_re = train_test_split(X_train_ret, y_train_ret, test_size
=0.33, random_state=42)
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X_train_all, y_train_all, test_
size=0.33, random_state=42)
```

In [15]:

```python
import lightgbm as lgb
from sklearn.metrics import accuracy_score, confusion_matrix,
matthews_corrcoef,classification_report,roc_curve, auc, mean_squared_error, make_scorer, f1_score
from math import sqrt
from sklearn.model_selection import cross_val_score, train_test_split, KFold, GridSearchCV,
RandomizedSearchCV
estimator = lgb.LGBMRegressor()

param_grid = {'num_leaves': [40,50,70,90], 'bagging_fraction ' : [0.5,0.7,0.9], 'bagging_freq ': [9
, 10, 11],
    'feature_fraction': [0.4, 0.5, 0.6],
    'learning_rate': [0.001, 0.01, 0.1]
}

lbm_re = RandomizedSearchCV(estimator, param_distributions = param_grid, cv=5, scoring = 'neg_mean_
squared_error')
lbm_re.fit(X_train_re, y_train_re)
print(lbm_re.best_params_)
```

```
{'num_leaves': 40, 'learning_rate': 0.01, 'feature_fraction': 0.6, 'bagging_freq ': 9,
'bagging_fraction ': 0.9}
```

In [16]:

```python
estimator = lgb.LGBMClassifier()

param_grid = {'num_leaves': [40,50,70,90], 'bagging_fraction ' : [0.5,0.7,0.9], 'bagging_freq ': [9
, 10, 11],
    'feature_fraction': [0.4, 0.5, 0.6],
    'learning_rate': [0.001, 0.01, 0.1]
}

lbm_clf = RandomizedSearchCV(estimator, param_distributions = param_grid, cv=5, scoring = 'neg_log_
loss')
lbm_clf.fit(X_train_clf, y_train_clf)
print(lbm_clf.best_params_)
```

```
{'num_leaves': 40, 'learning_rate': 0.1, 'feature_fraction': 0.4, 'bagging_freq ': 10,
'bagging_fraction ': 0.9}
```

In [17]:

```python
dtrain_clf = lgb.Dataset(X_train_clf, label = y_train_clf)
dtest_clf = lgb.Dataset(X_test_clf, label = y_test_clf)
dtrain_re = lgb.Dataset(X_train_re, label = y_train_re)
dtest_re = lgb.Dataset(X_test_re, label = y_test_re)
```

In [18]:

```python
#Parameters of "isReturned" classificator
param_lgb2 = {"objective":"binary",
              "num_leaves" : 40,
              "learning_rate" : 0.01,
              "bagging_fraction" : 0.9,
              "feature_fraction" : 0.4,
              "bagging_frequency" : 10,
              "metric": "binary_logloss"}


#Parameters of "How_Much_Returned_Will_Pay" regressor
param_lgb3= {"objective" : "regression",
             "metric" : "rmse",
             "num_leaves" : 40,
             "learning_rate" : 0.01,
             "bagging_fraction" : 0.9,
             "feature_fraction" : 0.6,
             "bagging_frequency" : 9}
```

In [19]:

```python
cat_clf = ['country', 'region', 'city', 'networkDomain']
cat_re = ['city']
```

```
cat_re = ['city']
```

In [20]:

```
clf_model = lgb.train(param_lgb2, dtrain_clf, 1200, valid_sets = [dtest_clf],
                      verbose_eval=20, early_stopping_rounds=700, categorical_feature = cat_clf)
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/basic.py:1247: UserWarning: categorical_feature in
Dataset is overridden.
New categorical_feature is ['city', 'country', 'networkDomain', 'region']
  'New categorical_feature is {}'.format(sorted(list(categorical_feature))))
```

```
Training until validation scores don't improve for 700 rounds
[20] valid_0's binary_logloss: 0.0307216
[40] valid_0's binary_logloss: 0.0310039
[60] valid_0's binary_logloss: 0.0308969
[80] valid_0's binary_logloss: 0.030863
[100] valid_0's binary_logloss: 0.0308749
[120] valid_0's binary_logloss: 0.0310135
[140] valid_0's binary_logloss: 0.031066
[160] valid_0's binary_logloss: 0.0308067
[180] valid_0's binary_logloss: 0.0308714
[200] valid_0's binary_logloss: 0.0309511
[220] valid_0's binary_logloss: 0.0310635
[240] valid_0's binary_logloss: 0.0311462
[260] valid_0's binary_logloss: 0.031245
[280] valid_0's binary_logloss: 0.031302
[300] valid_0's binary_logloss: 0.0313704
[320] valid_0's binary_logloss: 0.0314337
[340] valid_0's binary_logloss: 0.0314795
[360] valid_0's binary_logloss: 0.0316051
[380] valid_0's binary_logloss: 0.0316744
[400] valid_0's binary_logloss: 0.0317165
[420] valid_0's binary_logloss: 0.0317842
[440] valid_0's binary_logloss: 0.0318475
[460] valid_0's binary_logloss: 0.0320409
[480] valid_0's binary_logloss: 0.0320953
[500] valid_0's binary_logloss: 0.0321497
[520] valid_0's binary_logloss: 0.0322124
[540] valid_0's binary_logloss: 0.0322738
[560] valid_0's binary_logloss: 0.0323355
[580] valid_0's binary_logloss: 0.0324449
[600] valid_0's binary_logloss: 0.0326138
[620] valid_0's binary_logloss: 0.0330917
[640] valid_0's binary_logloss: 0.0332603
[660] valid_0's binary_logloss: 0.0336325
[680] valid_0's binary_logloss: 0.0336671
[700] valid_0's binary_logloss: 0.0342849
[720] valid_0's binary_logloss: 0.034782
Early stopping, best iteration is:
[31] valid_0's binary_logloss: 0.0303679
```

In [21]:

```
re_model = lgb.train(param_lgb3, dtrain_re, 1200, valid_sets = [dtest_re],
                      verbose_eval=20, early_stopping_rounds=700, categorical_feature = cat_re)
```

```
/anaconda3/lib/python3.7/site-packages/lightgbm/basic.py:1247: UserWarning: categorical_feature in
Dataset is overridden.
New categorical_feature is ['city']
  'New categorical_feature is {}'.format(sorted(list(categorical_feature))))
```

```
Training until validation scores don't improve for 700 rounds
[20] valid_0's rmse: 4.04981
[40] valid_0's rmse: 4.00071
[60] valid_0's rmse: 3.96611
[80] valid_0's rmse: 3.94231
[100] valid_0's rmse: 3.93009
[120] valid_0's rmse: 3.92147
[140] valid_0's rmse: 3.91509
[160] valid_0's rmse: 3.91328
[180] valid_0's rmse: 3.91308
[200] valid_0's rmse: 3.91404
```

```
[220] valid_0's rmse: 3.91632
[240] valid_0's rmse: 3.92015
[260] valid_0's rmse: 3.92487
[280] valid_0's rmse: 3.92773
[300] valid_0's rmse: 3.93017
[320] valid_0's rmse: 3.93444
[340] valid_0's rmse: 3.93721
[360] valid_0's rmse: 3.94164
[380] valid_0's rmse: 3.94625
[400] valid_0's rmse: 3.95134
[420] valid_0's rmse: 3.95596
[440] valid_0's rmse: 3.95859
[460] valid_0's rmse: 3.9618
[480] valid_0's rmse: 3.96535
[500] valid_0's rmse: 3.96945
[520] valid_0's rmse: 3.97164
[540] valid_0's rmse: 3.97523
[560] valid_0's rmse: 3.97868
[580] valid_0's rmse: 3.98178
[600] valid_0's rmse: 3.9841
[620] valid_0's rmse: 3.98718
[640] valid_0's rmse: 3.98912
[660] valid_0's rmse: 3.99216
[680] valid_0's rmse: 3.99595
[700] valid_0's rmse: 3.99871
[720] valid_0's rmse: 4.00107
[740] valid_0's rmse: 4.00339
[760] valid_0's rmse: 4.00601
[780] valid_0's rmse: 4.00715
[800] valid_0's rmse: 4.00976
[820] valid_0's rmse: 4.01134
[840] valid_0's rmse: 4.01405
[860] valid_0's rmse: 4.01721
[880] valid_0's rmse: 4.01904
Early stopping, best iteration is:
[186] valid_0's rmse: 3.91213
```

In [22]:

```
X_train_clf = lgb.Dataset(X_train_all, label = y_train_all)
X_train_re = lgb.Dataset(X_train_ret, label = y_train_ret)
```

In [25]:

```
pr_lgb_sum = 0
for i in range(10):
    lgb_model1 = lgb.train(param_lgb2, X_train_clf, 345)
    pr_lgb = lgb_model1.predict(X_test_clf_fs)
    lgb_model2 = lgb.train(param_lgb3, X_train_re,  165)
    pr_lgb_ret = lgb_model2.predict(X_test_re_fs)
    pr_lgb_sum = pr_lgb_sum + pr_lgb * pr_lgb_ret

pr_final2 = pr_lgb_sum / 10
```

In [26]:

```
df = pd.read_csv('test (1).csv', nrows=1) # Just take the first row to extract the columns' names
col_str_dic = {column:str for column in list(df)}
df = pd.read_csv('test (1).csv', dtype=col_str_dic)

data = {'fullVisitorId': [i for i in df.fullVisitorId], 'PredictedLogRevenue': [j for j in pr_final
2]}
newsub = pd.DataFrame(data)

newsub.to_csv("submission_lgb_fs.csv",index=False)
```

In [ ]: