Google Analytics Customer Revenue Prediction

Problem statement:

Predict how much GStore customers will spend

Data Overview

Taken data from Google Analytics Customer Revenue Prediction on kaggle https://www.kaggle.com/c/ga-customer-revenue-prediction/overview/description (https://www.kaggle.com/c/ga-customer-revenue-prediction/overview/description)

We are given two datasets:-

```
- train.csv

- test.csv
```

Each row in the dataset is one visit to the store. We are predicting the natural log of the sum of all transactions per user.

Train Dataset

Contains user transactions from August 1st 2016 to April 30th 2018.

Test Dataset

Contains user transactions from May 1st 2018 to October 15th 2018.

Private LB is being calculated on the future-looking timeframe of 12/1/18 to 1/31/19 - for those same set of users.

Data Fields

fullVisitorId- A unique identifier for each user of the Google Merchandise Store.

channelGrouping - The channel via which the user came to the Store.

date - The date on which the user visited the Store.

device - The specifications for the device used to access the Store.

geoNetwork - This section contains information about the geography of the user.

socialEngagementType - Engagement type, either "Socially Engaged" or "Not Socially Engaged".

totals - This section contains aggregate values across the session.

trafficSource - This section contains information about the Traffic Source from which the session originated.

visitId - An identifier for this session. This is part of the value usually stored as the _utmb cookie. This is only unique to the user. For a completely unique ID, you should use a combination of fullVisitorId and visitId.

visitNumber - The session number for this user. If this is the first session, then this is set to 1.

visitStartTime - The timestamp (expressed as POSIX time).

hits - This row and nested fields are populated for any and all types of hits. Provides a record of all page visits.

customDimensions - This section contains any user-level or session-level custom dimensions that are set for a session. This is a repeated field and has an entry for each dimension that is set.

totals - This set of columns mostly includes high-level aggregate data.

Real-world/Business objectives and constraints.

- No low-latency requirement.
- Some of the fields are in json format.
- Only a very small percentage of customers produce most of the revenue.

Performance metric for supervised learning:

- Minimize RMSE.

RMSE

Solution:-

- We will try to implement the winner's solution for this competition - https://www.kaggle.com/c/ga-customer-revenue-prediction/discussion/82614 (https://www.kaggle.com/c/ga-customer-revenue-prediction/discussion/82614)

Two key features of solution :-

- Framing train features in timeframe of 168 days (Window of test data - May 1st 2018 to October 15th 2018.)
- Solving this as Classification + Regression

  => Classification :- Whether the customer will return to shop in future time-window of 62 days (Number of days calculated from 12/1/18 to 1/31/19)

  => Regression :- Predict total transaction revenue from customer.

  => Final value would be the product above two values

  Solving the problem as Classification + Regression is motivated by Hurdle Model(https://seananderson.ca/2014/05/18/gamma-hurdle/ (https://seananderson.ca/2014/05/18/gamma-hurdle/)).

Hurdle Model :-

=> This model is preferred way of solving problem where target variable has more number of zeroes than a value.

=> It recommends to solve problem by

```
- Classification whether the value is going to non-zero or not
- And then predict the amount.
```

The solution implemented for this challenge is based on above model.

1. Import Required Libraries

In [2]:
```
%pip install dask[dataframe] --upgrade  --user
```

```
Requirement already up-to-date: dask[dataframe] in ./.local/lib/python3.5/sit
e-packages (2.6.0)
Collecting toolz>=0.7.3; extra == "dataframe"
  Downloading https://files.pythonhosted.org/packages/22/8e/037b9ba5c6a5739ef
0dcde60578c64d49f45f64c5e5e886531bfbc39157f/toolz-0.10.0.tar.gz (49kB)
     |████████████████████████████████| 51kB 1.4MB/s eta 0:00:011
Requirement already satisfied, skipping upgrade: pandas>=0.21.0; extra == "da
taframe" in /usr/local/lib/python3.5/dist-packages (from dask[dataframe]) (0.
25.3)
Requirement already satisfied, skipping upgrade: numpy>=1.13.0; extra == "dat
aframe" in /usr/local/lib/python3.5/dist-packages (from dask[dataframe]) (1.1
7.4)
Requirement already satisfied, skipping upgrade: cloudpickle>=0.2.1; extra ==
"dataframe" in /usr/local/lib/python3.5/dist-packages (from dask[dataframe])
(1.1.1)
Requirement already satisfied, skipping upgrade: fsspec>=0.5.1; extra == "dat
aframe" in /usr/local/lib/python3.5/dist-packages (from dask[dataframe]) (0.
6.1)
Collecting partd>=0.3.10; extra == "dataframe"
  Downloading https://files.pythonhosted.org/packages/44/e1/68dbe731c9c067655
bff1eca5b7d40c20ca4b23fd5ec9f3d17e201a6f36b/partd-1.1.0-py3-none-any.whl
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/local/l
ib/python3.5/dist-packages (from pandas>=0.21.0; extra == "dataframe"->dask[d
ataframe]) (2019.3)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in /u
sr/local/lib/python3.5/dist-packages (from pandas>=0.21.0; extra == "datafram
e"->dask[dataframe]) (2.8.1)
Collecting locket
  Downloading https://files.pythonhosted.org/packages/d0/22/3c0f97614e0be8386
542facb3a7dcfc2584f7b83608c02333bced641281c/locket-0.2.0.tar.gz
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/p
ython3.5/dist-packages (from python-dateutil>=2.6.1->pandas>=0.21.0; extra ==
"dataframe"->dask[dataframe]) (1.13.0)
Building wheels for collected packages: toolz, locket
  Building wheel for toolz (setup.py) ... done
  Created wheel for toolz: filename=toolz-0.10.0-cp35-none-any.whl size=57159
sha256=bbbd986710b8eb194a40c5bdb8a32087667fe2585a6e087ccc4922ccc4bc9350
  Stored in directory: /home/priyadarshi_cse/.cache/pip/wheels/e1/8b/65/3294e
5b727440250bda09e8c0153b7ba19d328f661605cb151
  Building wheel for locket (setup.py) ... done
  Created wheel for locket: filename=locket-0.2.0-cp35-none-any.whl size=4480
sha256=4b373aa2192853d88901d27d6fdb97c36b673394d1b95c726d7285a70b9b89c7
  Stored in directory: /home/priyadarshi_cse/.cache/pip/wheels/26/1e/e8/4fa23
6ec931b1a0cdd61578e20d4934d7bf188858723b84698
Successfully built toolz locket
Installing collected packages: toolz, locket, partd
Successfully installed locket-0.2.0 partd-1.1.0 toolz-0.10.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
#Importing Libraries

import warnings
warnings.filterwarnings("ignore")
import pandas as pd #pandas to create small dataframes
import json         #json library would be using to parse JSON Columns
from pandas.io.json import json_normalize #Library to normalize semi-structure
d JSON data into a flat table.
import os           #Library to use system level variable.
import matplotlib.pylab as plt #Plotting
import numpy as np  #Do aritmetic operations on arrays
import plotly.graph_objects as go #Graphing library.
import gc           #Garbage Collector interface
gc.enable() #Enable automatic garbage collection.
import lightgbm as lgb # Light GBM model
from sklearn.model_selection import RandomizedSearchCV #Hypertune parameters f
or model
from datetime import datetime, timedelta #The datetime module supplies classes
for manipulating dates and times.
from sklearn import preprocessing #Will use this library to label encode categ
orical features.
```

1. Train and Test Data Loading

Constraints with Train & Test Dataset :-

```
=> Train Dataset is of 25.4 GB. So, we have used Google Cloud instance with 50 GB m
ain memory to process it.
=> While Reading the dataset, we would be using Chunksize of 100000 to optimize per
formance.
=> Also, will drop unneccessary columns (columns with constant value or those which
adds minimal value) to reduce memory usage.
```

In [1]:
```
#Defining Function to load train and test Dataframe. Also, parsing the JSON Co
lumns


def load_df(csv_path='train.csv', nrows=None, feats=[]):
    #Columns in JSON Format.
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    temp_df = pd.DataFrame() #new df to store processed data and would be retu
red from this function

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON_COLUMNS
}, #Parsing JSON Columns from input file
                     dtype={'fullVisitorId': 'str', # Taking fullVisitorId as
 String, inorder to find multiple records for same user. As mentioned in Kaggl
e Competition details.
                            'channelGrouping': 'str',
                            'visitId':'int',
                            'visitNumber':'int',
                            'visitStartTime':'int'},
                            nrows=nrows,
                            parse_dates=['date'],
                            chunksize=100000) #Reading the train.csv using Pan
das Dataframe.

    for df_chunk in df:    #Process entire dataset, considering chunk of 10000
0 at a time
        df_chunk.reset_index(drop = True,inplace = True)  #Reset the index of
 the DataFrame, and use the default one instead.
        for column in JSON_COLUMNS: #Process each of the columns of dataset
            column_as_df = json_normalize(df_chunk[column]) #Normalize semi-st
ructured JSON data into a flat table.
            cols = []    #List to store the columns present in JSON field
            for subcol in column_as_df.columns:   #Process each column of JSON
fields
                cols.append(column + "." +  subcol)  #Defining columns for dat
aset as JSON field name.sub columns present under the JSON field.

            column_as_df.columns = cols #Adding the column name to dataset cre
ated above from JSON fields.
            df_chunk = df_chunk.drop(column, axis=1).merge(column_as_df, right
_index=True, left_index=True) #Dropping JSON column and adding columns extract
ed from JSON field.

        print("Loaded {path}. Shape: {shape}".format(path=os.path.basename(csv
_path),shape=df_chunk.shape)) #Shape of Loaded table.
        if len(feats) == 0:
            feats = df_chunk.columns    #populate feats with useful features p
assed on to the called function.
        use_df = df_chunk[feats] #Filter out useful features from dataset
        del df_chunk #delete this chunk of data read after choosing useful fea
tures from them.
        gc.collect()
        temp_df = pd.concat([temp_df, use_df], axis = 0).reset_index(drop = Tr
ue) #concatenate the dataframe generated in this iteration with already stored
```

```
one.
        print(temp_df.shape) #print the shape of input dataset
    return temp_df #return the processed dataset generated from input file
```

In [3]:
```
useful_feats = []
```

In [4]:
```
#Loading train dataframe. Initialing loading only 100k data, just to figure-ou
t useful features.
%time train_df = load_df("train_v2.csv",100000,useful_feats)
```

```
Loaded train_v2.csv. Shape: (100000, 59)
(100000, 59)
CPU times: user 23.6 s, sys: 2.98 s, total: 26.5 s
Wall time: 33.3 s
```

In [5]:
```
#Sample of train dataset
train_df.head()
```

Out[5]:

	channelGrouping	customDimensions	date	fullVisitorId	hits	socialEngage
0	Organic Search	[{'index': '4', 'value': 'EMEA'}]	2017-10-16	3162355547410993243	[{'hitNumber': '1', 'time': '0', 'hour': '17',...	Not Socially
1	Referral	[{'index': '4', 'value': 'North America'}]	2017-10-16	8934116514970143966	[{'hitNumber': '1', 'time': '0', 'hour': '10',...	Not Socially
2	Direct	[{'index': '4', 'value': 'North America'}]	2017-10-16	7992466427990357681	[{'hitNumber': '1', 'time': '0', 'hour': '17',...	Not Socially
3	Organic Search	[{'index': '4', 'value': 'EMEA'}]	2017-10-16	9075655783635761930	[{'hitNumber': '1', 'time': '0', 'hour': '9', ...	Not Socially
4	Organic Search	[{'index': '4', 'value': 'Central America'}]	2017-10-16	6960673291025684308	[{'hitNumber': '1', 'time': '0', 'hour': '14',...	Not Socially

5 rows × 59 columns

In [6]:
```
#Column list in train dataset
train_df.columns
```

Out[6]:
```
Index(['channelGrouping', 'customDimensions', 'date', 'fullVisitorId', 'hit
s',
       'socialEngagementType', 'visitId', 'visitNumber', 'visitStartTime',
       'device.browser', 'device.browserSize', 'device.browserVersion',
       'device.deviceCategory', 'device.flashVersion', 'device.isMobile',
       'device.language', 'device.mobileDeviceBranding',
       'device.mobileDeviceInfo', 'device.mobileDeviceMarketingName',
       'device.mobileDeviceModel', 'device.mobileInputSelector',
       'device.operatingSystem', 'device.operatingSystemVersion',
       'device.screenColors', 'device.screenResolution', 'geoNetwork.city',
       'geoNetwork.cityId', 'geoNetwork.continent', 'geoNetwork.country',
       'geoNetwork.latitude', 'geoNetwork.longitude', 'geoNetwork.metro',
       'geoNetwork.networkDomain', 'geoNetwork.networkLocation',
       '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.criteriaParameters',
       'trafficSource.adwordsClickInfo.gclId',
       'trafficSource.adwordsClickInfo.isVideoAd',
       'trafficSource.adwordsClickInfo.page',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.isTrueDirect', 'trafficSource.keyword',
       'trafficSource.medium', 'trafficSource.referralPath',
       'trafficSource.source'],
      dtype='object')
```

In [5]:
```
#Identify columns having same value for all transactions.
const_cols = [c for c in train_df.columns if train_df[c].nunique()==1 ]
const_cols
```

Out[5]:
```
['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',
 'totals.bounces',
 'totals.newVisits',
 'totals.visits',
 'trafficSource.adwordsClickInfo.criteriaParameters',
 'trafficSource.adwordsClickInfo.isVideoAd',
 'trafficSource.isTrueDirect']
```

In [8]:
```
#Columns having complex data and doesn't seem to add much value. There are oth
er columns having count of total number of hits.
train_df['hits'][1]
```

Out[8]:


```
'[{\'hitNumber\': \'1\', \'time\': \'0\', \'hour\': \'10\', \'minute\': \'51
\', \'isInteraction\': True, \'isEntrance\': True, \'referer\': \'https://sit
es.google.com/a/google.com/transportation/mtv-services/bikes/bike2workmay2016
\', \'page\': {\'pagePath\': \'/home\', \'hostname\': \'shop.googlemerchandis
estore.com\', \'pageTitle\': \'Home\', \'searchKeyword\': \'jersey\', \'searc
hCategory\': \'(not set)\', \'pagePathLevel1\': \'/home\', \'pagePathLevel2
\': \'\', \'pagePathLevel3\': \'\', \'pagePathLevel4\': \'\'}, \'appInfo\':
{\'screenName\': \'shop.googlemerchandisestore.com/home\', \'landingScreenNam
e\': \'shop.googlemerchandisestore.com/home\', \'exitScreenName\': \'shop.goo
glemerchandisestore.com/asearch.html\', \'screenDepth\': \'0\'}, \'exceptionI
nfo\': {\'isFatal\': True}, \'product\': [], \'promotion\': [{\'promoId\':
\'Apparel Row 1\', \'promoName\': \'Apparel\', \'promoCreative\': \'home_main
_link_apparel.jpg\', \'promoPosition\': \'Row 1\'}, {\'promoId\': \'Backpacks
Row 2 Combo\', \'promoName\': \'Backpacks\', \'promoCreative\': \'home_bags_g
oogle_2.jpg\', \'promoPosition\': \'Row 2 Combo\'}, {\'promoId\': \'Mens T-Sh
irts Row 3-1\', \'promoName\': \'Mens T-Shirts\', \'promoCreative\': \'mens-t
shirts.jpg\', \'promoPosition\': \'Row 3-1\'}, {\'promoId\': \'Womens T-Shirt
s Row 3-2\', \'promoName\': \'Womens T-Shirts\', \'promoCreative\': \'womens-
tshirts.jpg\', \'promoPosition\': \'Row 3-2\'}, {\'promoId\': \'Office Row 5
Color Combo\', \'promoName\': \'Office\', \'promoCreative\': \'green_row_link
_to_office.jpg\', \'promoPosition\': \'Row 5 Color Combo\'}, {\'promoId\':
\'Drinkware Row 4 Color Combo\', \'promoName\': \'Drinkware\', \'promoCreativ
e\': \'red_row_hydrate.jpg\', \'promoPosition\': \'Row 4 Color Combo\'}, {\'p
romoId\': \'Google Brand Row 7-1\', \'promoName\': \'Google Brand\', \'promoC
reative\': \'home_lower_google_500.jpg\', \'promoPosition\': \'Brand Row 7-1
\'}, {\'promoId\': \'YouTube Brand Row 7-2\', \'promoName\': \'YouTube Brand
\', \'promoCreative\': \'home_lower_youtube_500.jpg\', \'promoPosition\': \'B
rand Row 7-2\'}, {\'promoId\': \'Android Brand Row 7-3\', \'promoName\': \'An
driod Brand\', \'promoCreative\': \'home_lower_android_500.jpg\', \'promoPosi
tion\': \'Brand Row 7-3\'}], \'promotionActionInfo\': {\'promoIsView\': Tru
e}, \'eCommerceAction\': {\'action_type\': \'0\', \'step\': \'1\'}, \'experim
ent\': [], \'customVariables\': [], \'customDimensions\': [], \'customMetrics
\': [], \'type\': \'PAGE\', \'social\': {\'socialNetwork\': \'(not set)\',
\'hasSocialSourceReferral\': \'No\', \'socialInteractionNetworkAction\': \' :
\'}, \'contentGroup\': {\'contentGroup1\': \'(not set)\', \'contentGroup2\':
\'(not set)\', \'contentGroup3\': \'(not set)\', \'contentGroup4\': \'(not se
t)\', \'contentGroup5\': \'(not set)\', \'previousContentGroup1\': \'(entranc
e)\', \'previousContentGroup2\': \'(entrance)\', \'previousContentGroup3\':
\'(entrance)\', \'previousContentGroup4\': \'(entrance)\', \'previousContentG
roup5\': \'(entrance)\'}, \'dataSource\': \'web\', \'publisher_infos\': []},
{\'hitNumber\': \'2\', \'time\': \'27844\', \'hour\': \'10\', \'minute\': \'5
2\', \'isInteraction\': True, \'isExit\': True, \'page\': {\'pagePath\': \'/a
search.html\', \'hostname\': \'shop.googlemerchandisestore.com\', \'pageTitle
\': \'Store search results\', \'pagePathLevel1\': \'/asearch.html\', \'pagePa
thLevel2\': \'\', \'pagePathLevel3\': \'\', \'pagePathLevel4\': \'\'}, \'tran
saction\': {\'currencyCode\': \'USD\'}, \'item\': {\'currencyCode\': \'USD
\'}, \'appInfo\': {\'screenName\': \'shop.googlemerchandisestore.com/asearch.
html\', \'landingScreenName\': \'shop.googlemerchandisestore.com/home\', \'ex
itScreenName\': \'shop.googlemerchandisestore.com/asearch.html\', \'screenDep
th\': \'0\'}, \'exceptionInfo\': {\'isFatal\': True}, \'product\': [{\'produc
tSKU\': \'GGOEGAAX0104\', \'v2ProductName\': "Google Men\'s 100% Cotton Short
Sleeve Hero Tee White", \'v2ProductCategory\': \'(not set)\', \'productVarian
t\': \'(not set)\', \'productBrand\': \'(not set)\', \'productPrice\': \'1699
0000\', \'localProductPrice\': \'16990000\', \'isImpression\': True, \'custom
Dimensions\': [], \'customMetrics\': [], \'productListName\': \'Search Result
s\', \'productListPosition\': \'1\'}, {\'productSKU\': \'GGOEGAAX0105\', \'v2
ProductName\': "Google Men\'s 100% Cotton Short Sleeve Hero Tee Black", \'v2P
```

roductCategory\': \'(not set)\', \'productVariant\': \'(not set)\', \'product
Brand\': \'(not set)\', \'productPrice\': \'16990000\', \'localProductPrice
\': \'16990000\', \'isImpression\': True, \'customDimensions\': [], \'customM
etrics\': [], \'productListName\': \'Search Results\', \'productListPosition
\': \'2\'}, {\'productSKU\': \'GGOEGAAX0106\', \'v2ProductName\': "Google Men
\'s 100% Cotton Short Sleeve Hero Tee Navy", \'v2ProductCategory\': \'(not se
t)\', \'productVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'p
roductPrice\': \'16990000\', \'localProductPrice\': \'16990000\', \'isImpress
ion\': True, \'customDimensions\': [], \'customMetrics\': [], \'productListNa
me\': \'Search Results\', \'productListPosition\': \'3\'}, {\'productSKU\':
\'GGOEGAAX0279\', \'v2ProductName\': "Google Women\'s Short Sleeve Hero Tee W
hite", \'v2ProductCategory\': \'(not set)\', \'productVariant\': \'(not set)
\', \'productBrand\': \'(not set)\', \'productPrice\': \'16990000\', \'localP
roductPrice\': \'16990000\', \'isImpression\': True, \'customDimensions\':
[], \'customMetrics\': [], \'productListName\': \'Search Results\', \'product
ListPosition\': \'4\'}, {\'productSKU\': \'GGOEGAAX0291\', \'v2ProductName\':
"Google Women\'s Short Sleeve Hero Tee Sky Blue", \'v2ProductCategory\': \'(n
ot set)\', \'productVariant\': \'(not set)\', \'productBrand\': \'(not set)
\', \'productPrice\': \'18990000\', \'localProductPrice\': \'18990000\', \'is
Impression\': True, \'customDimensions\': [], \'customMetrics\': [], \'produc
tListName\': \'Search Results\', \'productListPosition\': \'5\'}, {\'productS
KU\': \'GGOEGAAX0278\', \'v2ProductName\': "Google Women\'s Short Sleeve Hero
Tee Black", \'v2ProductCategory\': \'(not set)\', \'productVariant\': \'(not
set)\', \'productBrand\': \'(not set)\', \'productPrice\': \'16990000\', \'lo
calProductPrice\': \'16990000\', \'isImpression\': True, \'customDimensions
\': [], \'customMetrics\': [], \'productListName\': \'Search Results\', \'pro
ductListPosition\': \'6\'}, {\'productSKU\': \'GGOEGAAX0297\', \'v2ProductNam
e\': "Google Women\'s Short Sleeve Hero Tee Red Heather", \'v2ProductCategory
\': \'(not set)\', \'productVariant\': \'(not set)\', \'productBrand\': \'(no
t set)\', \'productPrice\': \'18990000\', \'localProductPrice\': \'18990000
\', \'isImpression\': True, \'customDimensions\': [], \'customMetrics\': [],
\'productListName\': \'Search Results\', \'productListPosition\': \'7\'},
{\'productSKU\': \'GGOEGAAX0107\', \'v2ProductName\': "Google Men\'s 100% Cot
ton Short Sleeve Hero Tee Red", \'v2ProductCategory\': \'(not set)\', \'produ
ctVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'productPrice
\': \'16990000\', \'localProductPrice\': \'16990000\', \'isImpression\': Tru
e, \'customDimensions\': [], \'customMetrics\': [], \'productListName\': \'Se
arch Results\', \'productListPosition\': \'8\'}, {\'productSKU\': \'GGOEGAAX0
280\', \'v2ProductName\': "Google Women\'s Short Sleeve Hero Tee Grey", \'v2P
roductCategory\': \'(not set)\', \'productVariant\': \'(not set)\', \'product
Brand\': \'(not set)\', \'productPrice\': \'16990000\', \'localProductPrice
\': \'16990000\', \'isImpression\': True, \'customDimensions\': [], \'customM
etrics\': [], \'productListName\': \'Search Results\', \'productListPosition
\': \'9\'}, {\'productSKU\': \'GGOEGAAX0289\', \'v2ProductName\': "Google Wom
en\'s Short Sleeve Hero Dark Grey", \'v2ProductCategory\': \'(not set)\', \'p
roductVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'productPri
ce\': \'18990000\', \'localProductPrice\': \'18990000\', \'isImpression\': Tr
ue, \'customDimensions\': [], \'customMetrics\': [], \'productListName\': \'S
earch Results\', \'productListPosition\': \'10\'}, {\'productSKU\': \'GGOEGAA
X0281\', \'v2ProductName\': "Google Women\'s Short Sleeve Badge Tee Grey",
\'v2ProductCategory\': \'(not set)\', \'productVariant\': \'(not set)\', \'pr
oductBrand\': \'(not set)\', \'productPrice\': \'16990000\', \'localProductPr
ice\': \'16990000\', \'isImpression\': True, \'customDimensions\': [], \'cust
omMetrics\': [], \'productListName\': \'Search Results\', \'productListPositi
on\': \'11\'}, {\'productSKU\': \'GGOEGAAX0746\', \'v2ProductName\': "Google
Women\'s Short Sleeve Badge Tee Navy", \'v2ProductCategory\': \'(not set)\',
\'productVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'product

Price\': \'16990000\', \'localProductPrice\': \'16990000\', \'isImpression\': True, \'customDimensions\': [], \'customMetrics\': [], \'productListName\': \'Search Results\', \'productListPosition\': \'12\'}, {\'productSKU\': \'GGOE GAAX0324\', \'v2ProductName\': "Android Men\'s Short Sleeve Tri-blend Hero Tee Grey", \'v2ProductCategory\': \'(not set)\', \'productVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'productPrice\': \'18990000\', \'localProductPrice\': \'18990000\', \'isImpression\': True, \'customDimensions\': [], \'customMetrics\': [], \'productListName\': \'Search Results\', \'productListPosition\': \'13\'}, {\'productSKU\': \'GGOEGAAX0326\', \'v2ProductName\': "Google Men\'s Short Sleeve Badge Tee Charcoal", \'v2ProductCategory\': \'(not set)\', \'productVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'productPrice\': \'18990000\', \'localProductPrice\': \'18990000\', \'isImpression\': True, \'customDimensions\': [], \'customMetrics\': [], \'productListName\': \'Search Results\', \'productListPosition\': \'14\'}, {\'productSKU\': \'GGOEGAAX0323\', \'v2ProductName\': "Google Men\'s Short Sleeve Hero Tee Charcoal", \'v2ProductCategory\': \'(not set)\', \'productVariant\': \'(not set)\', \'productBrand\': \'(not set)\', \'productPrice\': \'18990000\', \'localProductPrice\': \'18990000\', \'isImpression\': True, \'customDimensions\': [], \'customMetrics\': [], \'productListName\': \'Search Results\', \'productListPosition\': \'15\'}], \'promotion\': [], \'eCommerceAction\': {\'action_type\': \'0\', \'step\': \'1\'}, \'experiment\': [], \'customVariables\': [], \'customDimensions\': [], \'customMetrics\': [], \'type\': \'PAGE\', \'social\': {\'socialNetwork\': \'(not set)\', \'hasSocialSourceReferral\': \'No\', \'socialInteractionNetworkAction\': \' : \'}, \'contentGroup\': {\'contentGroup1\': \'(not set)\', \'contentGroup2\': \'(not set)\', \'contentGroup3\': \'(not set)\', \'contentGroup4\': \'(not set)\', \'contentGroup5\': \'(not set)\', \'previousContentGroup1\': \'(not set)\', \'previousContentGroup2\': \'(not set)\', \'previousContentGroup3\': \'(not set)\', \'previousContentGroup4\': \'(not set)\', \'previousContentGroup5\': \'(not set)\'}, \'dataSource\': \'web\', \'publisher_infos\': []}]'

In [6]:

```
#List of Columns to be dropped.
cols_to_drop = const_cols + ['customDimensions'] + ['hits'] + ["trafficSource.adwordsClickInfo.adNetworkType",
            "trafficSource.adwordsClickInfo.gclId", "trafficSource.adwordsClickInfo.slot",
            "trafficSource.adwordsClickInfo.page"]

train_df = train_df.drop(cols_to_drop, axis=1)
```

In [10]:

```
#Initialize useful features with all the columns in train-set and later will remove the unwanted ones.
useful_feats = list(train_df.columns)
```

In [7]:

```
#generate list of useful features, removing the columns those add minimal value.
useful_feats = list(filter(lambda col: col not in cols_to_drop, useful_feats))
```

In [15]:
```
#List of useful features generated from above.
list(useful_feats)
```

Out[15]:
```
['channelGrouping',
 'date',
 'fullVisitorId',
 'visitId',
 'visitNumber',
 'visitStartTime',
 'device.browser',
 'device.deviceCategory',
 'device.isMobile',
 'device.operatingSystem',
 'geoNetwork.city',
 'geoNetwork.continent',
 'geoNetwork.country',
 'geoNetwork.metro',
 'geoNetwork.networkDomain',
 'geoNetwork.region',
 'geoNetwork.subContinent',
 'totals.hits',
 'totals.pageviews',
 'totals.sessionQualityDim',
 'totals.timeOnSite',
 'totals.totalTransactionRevenue',
 'totals.transactionRevenue',
 'totals.transactions',
 'trafficSource.adContent',
 'trafficSource.adwordsClickInfo.slot',
 'trafficSource.campaign',
 'trafficSource.keyword',
 'trafficSource.medium',
 'trafficSource.referralPath',
 'trafficSource.source']
```

In [11]:
```
#Loading train dataframe and only the useful features.
%time train_df = load_df("train_v2.csv", feats = useful_feats)
```

```
Loaded train_v2.csv. Shape: (100000, 59)
(100000, 31)
Loaded train_v2.csv. Shape: (100000, 60)
(200000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(300000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(400000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(500000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(600000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(700000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(800000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(900000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1000000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1100000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1200000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1300000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1400000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1500000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1600000, 31)
Loaded train_v2.csv. Shape: (100000, 59)
(1700000, 31)
Loaded train_v2.csv. Shape: (8337, 59)
(1708337, 31)
CPU times: user 8min 2s, sys: 34.8 s, total: 8min 37s
Wall time: 10min 22s
```

In [16]:
```
train_df['totals.timeOnSite'][1000]
```

Out[16]:
```
nan
```

In [14]:
```
#Below code ran to identify memory allocated to different variable. So that we can identify any unused variable and delete those to free some.
import sys

# These are the usual ipython objects, including this one you are creating
ipython_vars = ['In', 'Out', 'exit', 'quit', 'get_ipython', 'ipython_vars']

# Get a sorted list of the objects and their sizes
sorted([(x, sys.getsizeof(globals().get(x))) for x in dir() if not x.startswith('_') and x not in sys.modules and x not in ipython_vars], key=lambda x: x[1], reverse=True)
```

Out[14]:
```
[('train_df', 2823104511),
 ('cols_to_drop', 272),
 ('const_cols', 264),
 ('json_normalize', 136),
 ('load_df', 136),
 ('go', 80),
 ('np', 80),
 ('pd', 80),
 ('plt', 80)]
```

In [12]:
```
#Loading test dataframe.
%time test_df = load_df("test_v2.csv", feats = useful_feats)
```

```
Loaded test_v2.csv. Shape: (100000, 59)
(100000, 31)
Loaded test_v2.csv. Shape: (100000, 59)
(200000, 31)
Loaded test_v2.csv. Shape: (100000, 59)
(300000, 31)
Loaded test_v2.csv. Shape: (100000, 59)
(400000, 31)
Loaded test_v2.csv. Shape: (1589, 59)
(401589, 31)
CPU times: user 2min 2s, sys: 7.71 s, total: 2min 10s
Wall time: 2min 47s
```

In [18]:
```
train_df.columns
```

Out[18]:
```
Index(['channelGrouping', 'date', 'fullVisitorId', 'visitId', 'visitNumber',
       'visitStartTime', 'device.browser', 'device.deviceCategory',
       'device.isMobile', 'device.operatingSystem', 'geoNetwork.city',
       'geoNetwork.continent', 'geoNetwork.country', 'geoNetwork.metro',
       'geoNetwork.networkDomain', 'geoNetwork.region',
       'geoNetwork.subContinent', 'totals.hits', 'totals.pageviews',
       'totals.sessionQualityDim', 'totals.timeOnSite',
       'totals.totalTransactionRevenue', 'totals.transactionRevenue',
       'totals.transactions', 'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.keyword', 'trafficSource.medium',
       'trafficSource.referralPath', 'trafficSource.source'],
      dtype='object')
```

In [19]:
```
test_df.columns
```

Out[19]:
```
Index(['channelGrouping', 'date', 'fullVisitorId', 'visitId', 'visitNumber',
       'visitStartTime', 'device.browser', 'device.deviceCategory',
       'device.isMobile', 'device.operatingSystem', 'geoNetwork.city',
       'geoNetwork.continent', 'geoNetwork.country', 'geoNetwork.metro',
       'geoNetwork.networkDomain', 'geoNetwork.region',
       'geoNetwork.subContinent', 'totals.hits', 'totals.pageviews',
       'totals.sessionQualityDim', 'totals.timeOnSite',
       'totals.totalTransactionRevenue', 'totals.transactionRevenue',
       'totals.transactions', 'trafficSource.adContent',
       'trafficSource.adwordsClickInfo.slot', 'trafficSource.campaign',
       'trafficSource.keyword', 'trafficSource.medium',
       'trafficSource.referralPath', 'trafficSource.source'],
      dtype='object')
```

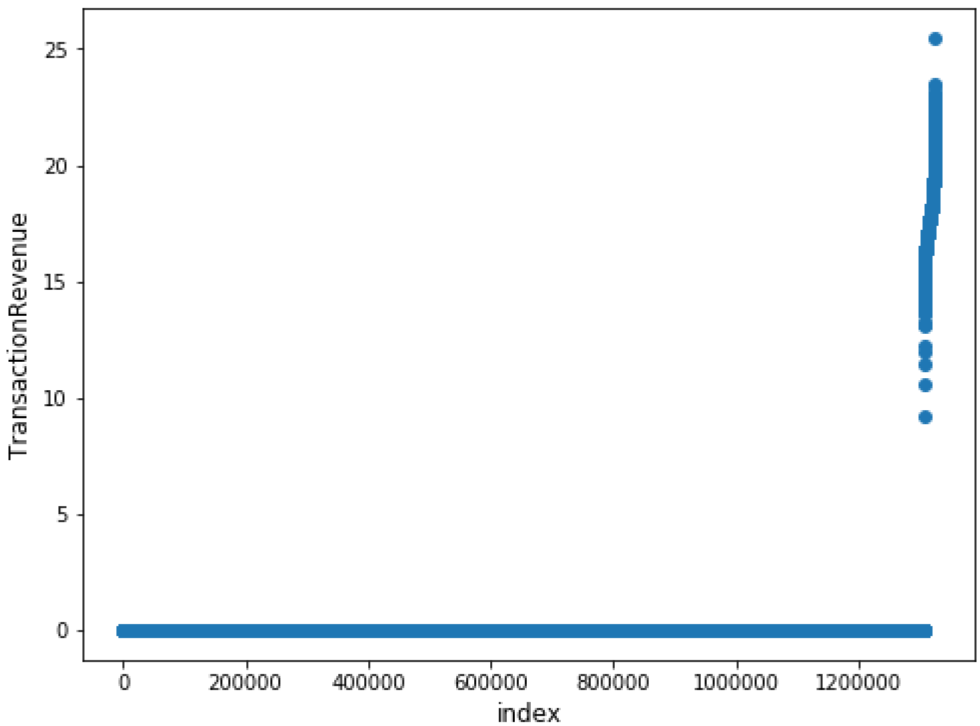
In [20]:
```
print("Shape of train set", train_df.shape[0], train_df.shape[1])
print("Shape of test set", test_df.shape[0], test_df.shape[1] )
```

```
Shape of train set 1708337 31
Shape of test set 401589 31
```

2. Exploratory Data Analysis

2.1. Distribution of Target Variable(totals.transactionRevenue) in training data set

In [64]:
```
#Plotting transactionRevenue per user in train-set.
#Source - https://www.kaggle.com/sudalairajkumar/simple-exploration-baseline-g
a-customer-revenue
train_df["totals.transactionRevenue"] = train_df["totals.transactionRevenue"].
astype('float')  #Converting the transaction revenue field to float type.
gdf = train_df.groupby("fullVisitorId")["totals.transactionRevenue"].sum().res
et_index() #Summing up all the transaction revenue for an user id.

gdf_nrows = gdf.shape[0]

plt.figure(figsize=(8,6))
plt.scatter(range(gdf_nrows), np.sort(np.log1p(gdf["totals.transactionRevenue"
].values))) #for all the users plotting log of above summed up transaction rev
enue values.
plt.xlabel('index', fontsize=12)
plt.ylabel('TransactionRevenue', fontsize=12)
plt.show()
```



In [66]:
```
#Number/ Ratio of customers having transactionRevenue greater than zero.
nzi = pd.notnull(train_df["totals.transactionRevenue"]).sum() #nzi - Count of
 all instances/ transactions having transaction revenue not null.
nzr = (gdf["totals.transactionRevenue"]>0).sum() #nzi - Count of all users hav
ing transacton revenue great than zero.
print("Number of instances in train set with non-zero revenue : ", nzi, ", tot
al number of transactions :", train_df.shape[0], " and ratio is : ", nzi / tra
in_df.shape[0])
print("Number of unique customers with non-zero revenue : ", nzr, ", total num
ber of users",  gdf_nrows, "and the ratio is : ", nzr / gdf_nrows)
```

```
Number of instances in train set with non-zero revenue :  18514 , total numbe
r of transactions : 1708337  and ratio is :  0.010837440153786987
Number of unique customers with non-zero revenue :  16141 , total number of u
sers 1323730 and the ratio is :  0.012193574218307359
```

Observation -

Ratio of revenue generating customers to customers with no revenue is in the ratio of 1.21%

2.2 Number of visitors and common visitors in train and test set:

In [67]:
```
print("Number of unique visitors in train set : ",train_df.fullVisitorId.nunique(), " out of rows : ",train_df.shape[0])
print("Number of unique visitors in test set : ",test_df.fullVisitorId.nunique(), " out of rows : ",test_df.shape[0])
print("Number of common visitors in train and test set : ",len(set(train_df.fullVisitorId.unique()).intersection(set(test_df.fullVisitorId.unique())) ))
```

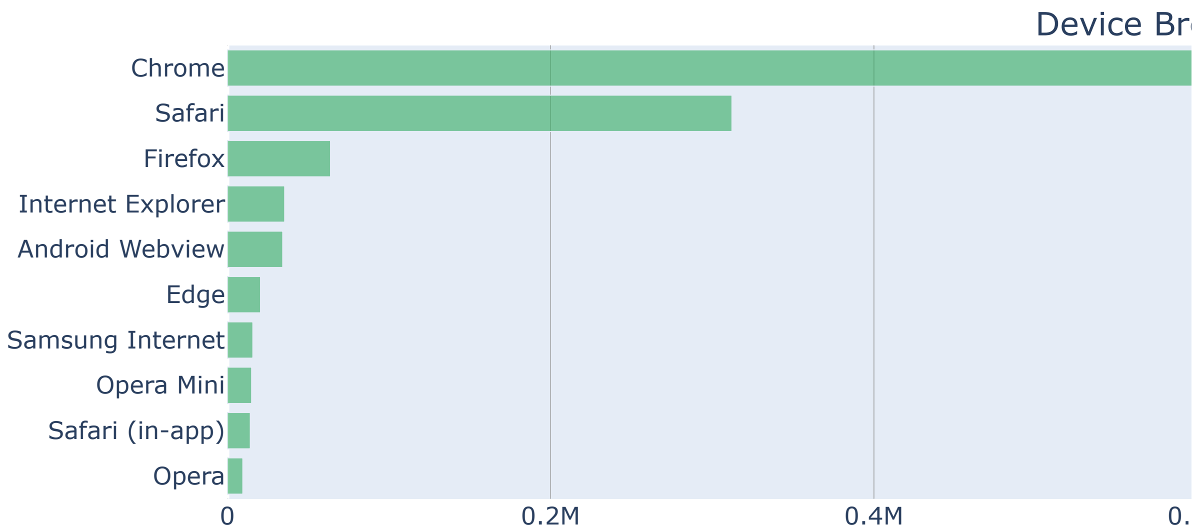
```
Number of unique visitors in train set :  1323730  out of rows :  1708337
Number of unique visitors in test set :  296530  out of rows :  401589
Number of common visitors in train and test set :  2759
```

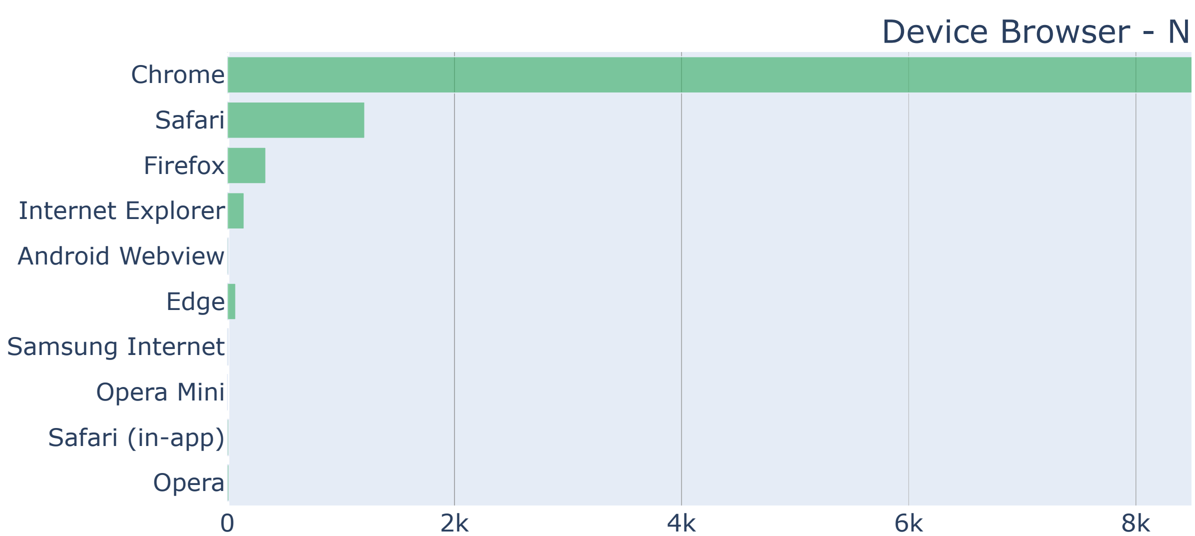
2.3 Gemerate Plots of Some of the key features to visualize data:

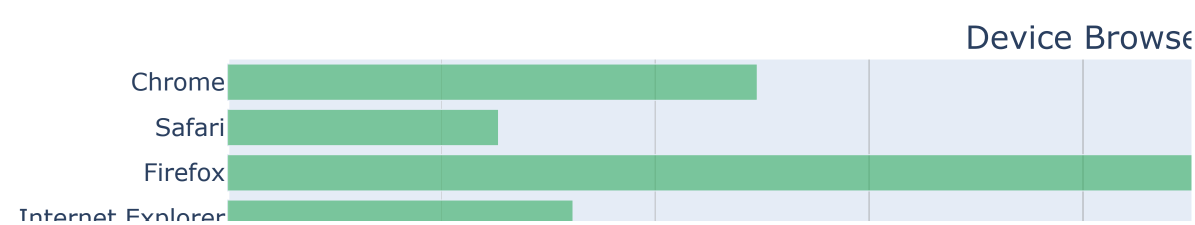
In [68]:
```
#Function to generate horizontal bar chart.
# cnt_srs = Value for each unique item
# Color of bar

def horizontal_bar_chart(cnt_srs, color):
    trace = go.Bar(
        y=cnt_srs.index[::-1],
        x=cnt_srs.values[::-1],
        showlegend=False,
        orientation = 'h',
        marker=dict(
            color=color,
        ),
    )
    return trace
```

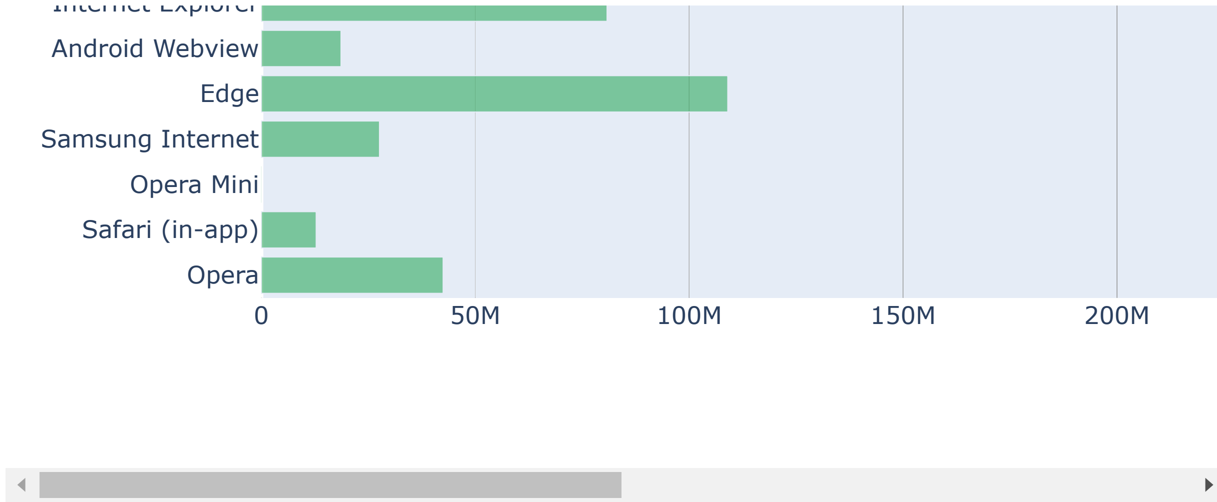
In [69]:
```
# #Plotting Device Browser of transactions.
from plotly.offline import init_notebook_mode, iplot
from plotly.subplots import make_subplots

cnt_srs = train_df.groupby('device.browser')['totals.transactionRevenue'].agg
(['size', 'count', 'mean'])
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"]
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
trace1 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(50, 171, 96, 0.
6)')
trace2 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head(10),
'rgba(50, 171, 96, 0.6)')
trace3 = horizontal_bar_chart(cnt_srs["mean"].head(10), 'rgba(50, 171, 96, 0.
6)')

# Creating two subplots
fig = make_subplots(rows=3, cols=1,
                          subplot_titles=["Device Browser - Count", "Device Br
owser - Non-zero Revenue Count", "Device Browser - Mean Revenue"])


fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)

fig['layout'].update(height=1200, width=1200, paper_bgcolor='rgb(233,233,233)'
, title="Device Plots")
iplot(fig, filename='device-plots')
```

Device Plots









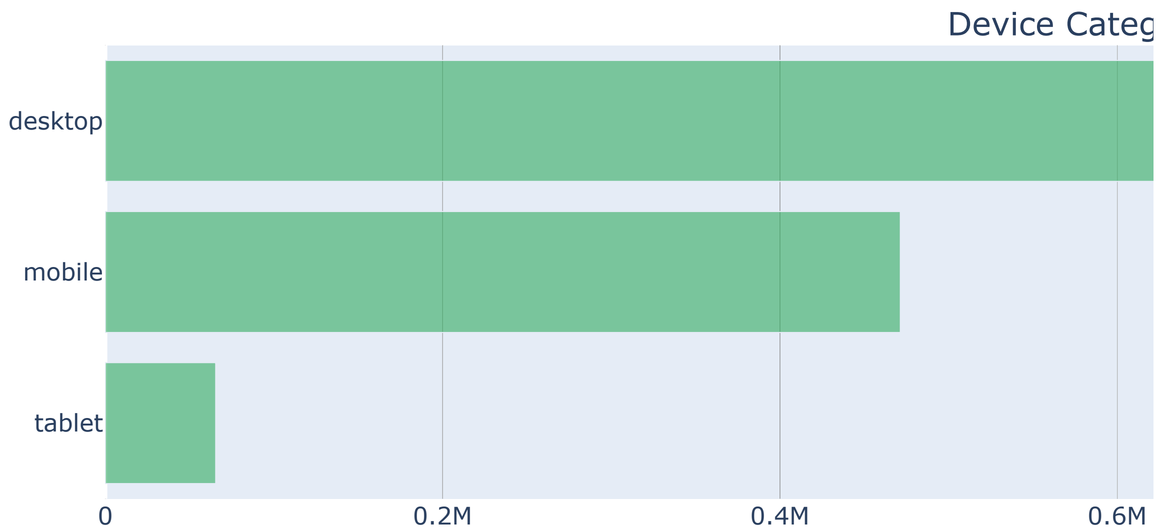
Observation -

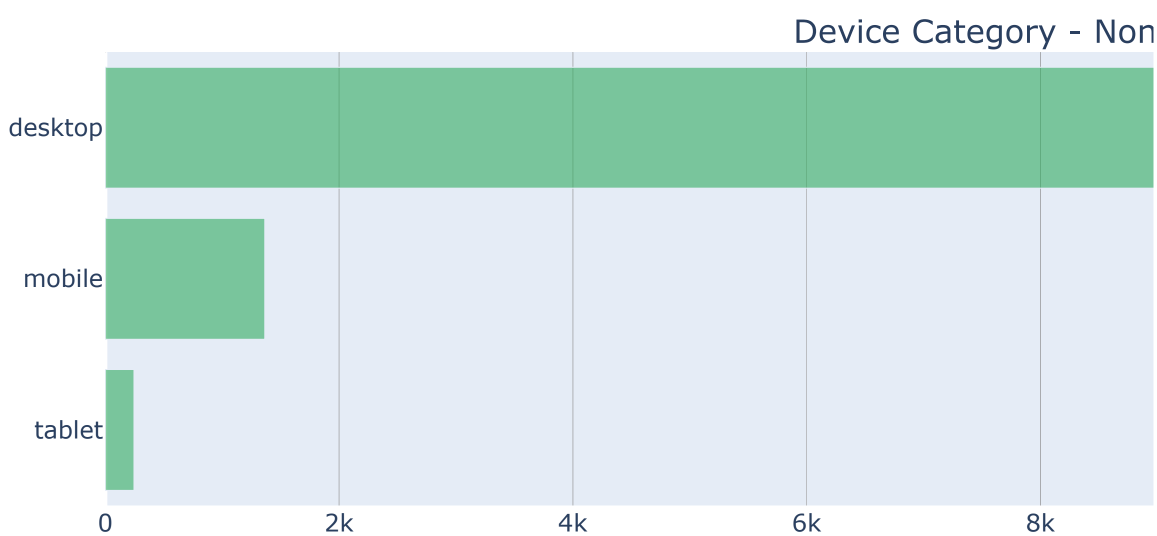
=> Chrome seems to be widely used browser.

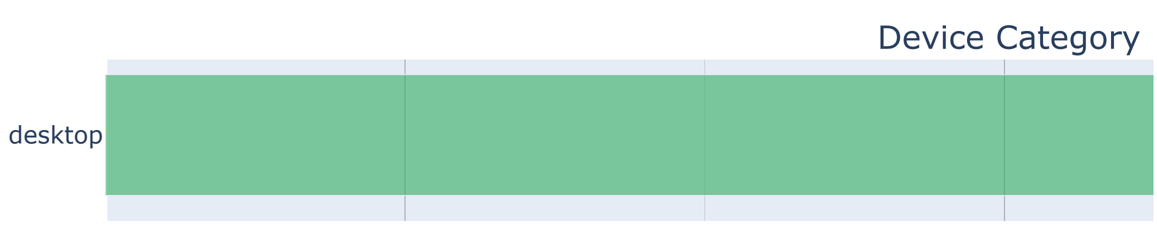
=> Whereas, mean revenue is highest on Firefox.

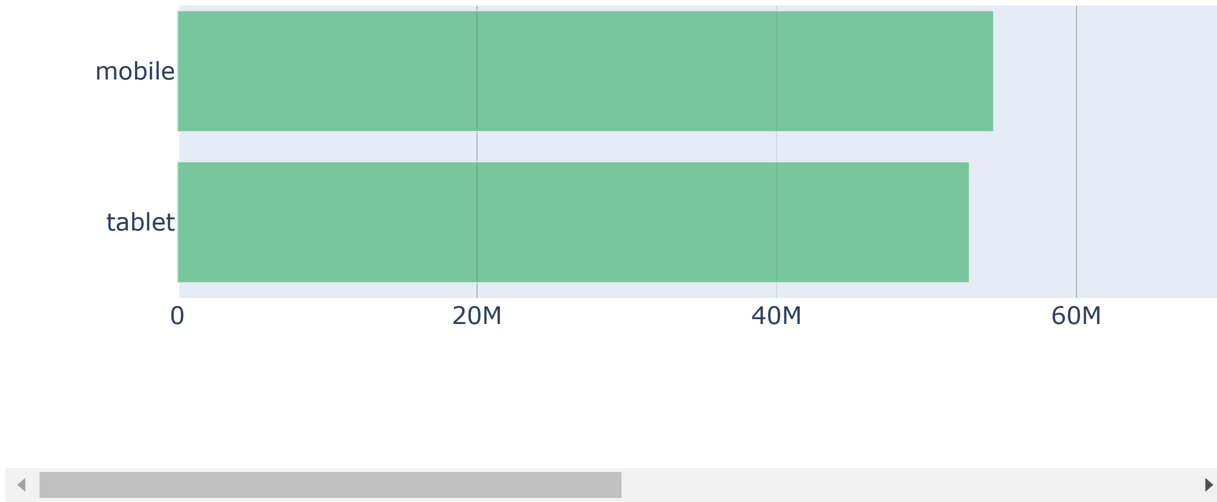
In [70]:
```
#Plotting Device of transactions.
cnt_srs = train_df.groupby('device.deviceCategory')['totals.transactionRevenu
e'].agg(['size', 'count', 'mean'])
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"]
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
trace1 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(50, 171, 96, 0.
6)')
trace2 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head(10),
'rgba(50, 171, 96, 0.6)')
trace3 = horizontal_bar_chart(cnt_srs["mean"].head(10), 'rgba(50, 171, 96, 0.
6)')

# Creating subplots
fig = make_subplots(rows=3, cols=1,
                          subplot_titles=["Device Category - Count", "Device C
ategory - Non-zero Revenue Count", "Device Category - Mean Revenue"])


fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)

fig['layout'].update(height=1200, width=1200, paper_bgcolor='rgb(233,233,233)'
, title="Device Plots")
iplot(fig, filename='device-plots')
```

Device Plots







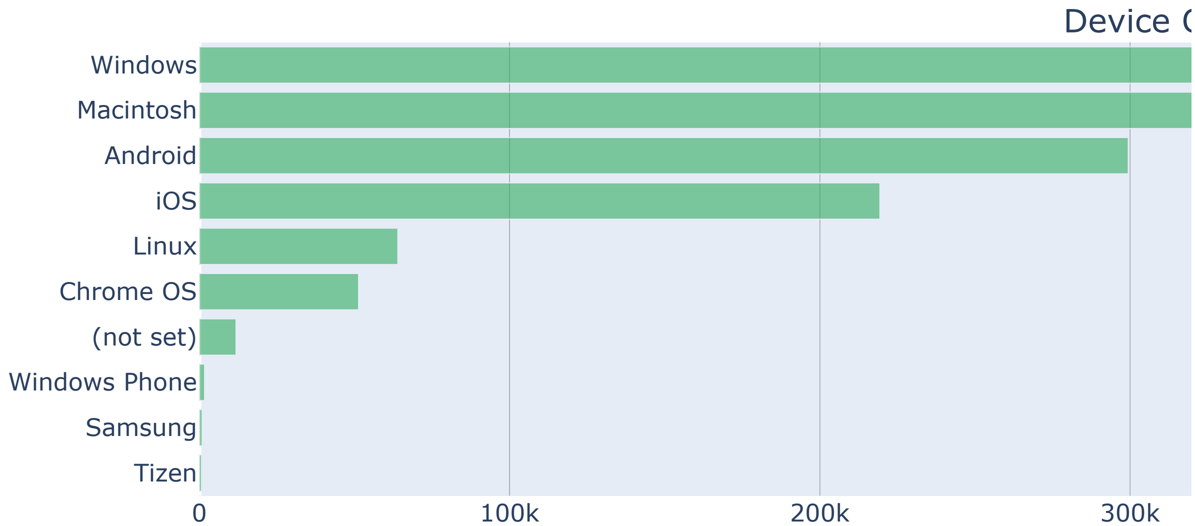


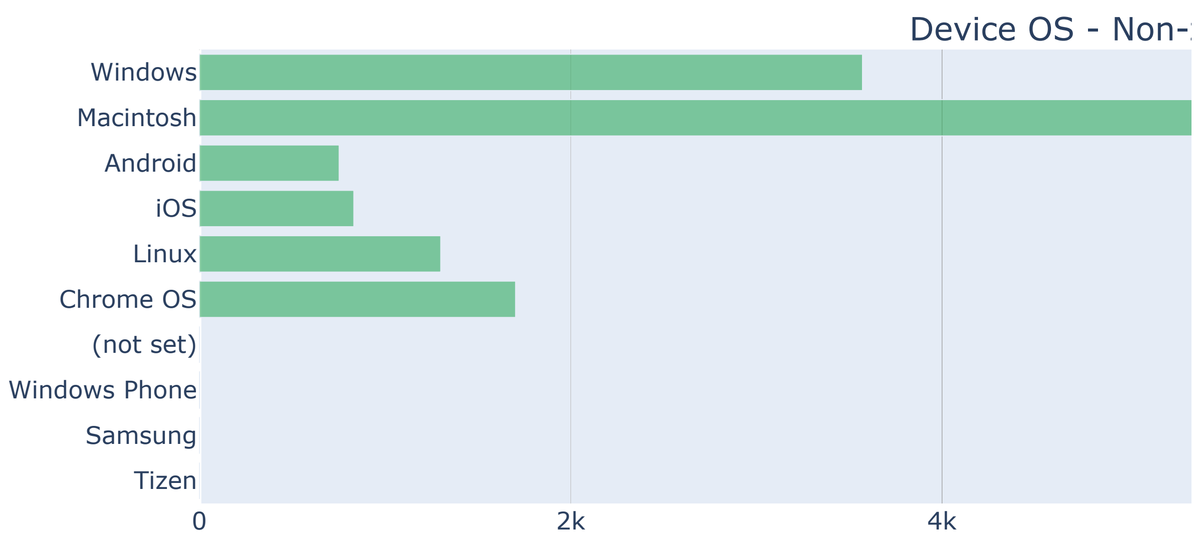
Observation -

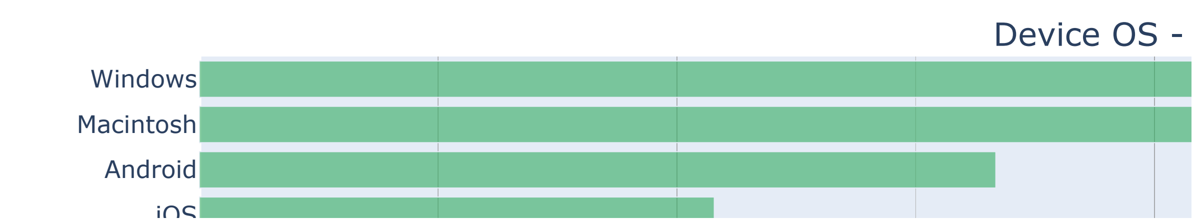
=> Maximum transactions were done from Desktop

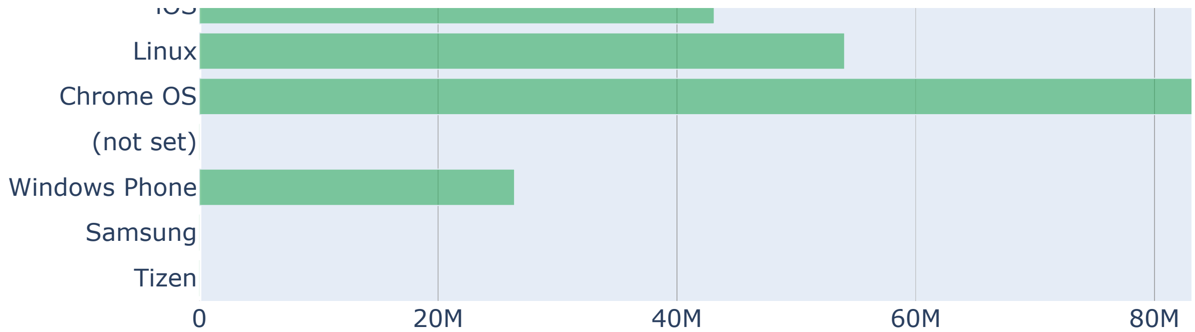
In [71]:
```
#Plotting OS of transactions.
cnt_srs = train_df.groupby('device.operatingSystem')['totals.transactionRevenu
e'].agg(['size', 'count', 'mean'])
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"]
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
trace1 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(50, 171, 96, 0.
6)')
trace2 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head(10),
'rgba(50, 171, 96, 0.6)')
trace3 = horizontal_bar_chart(cnt_srs["mean"].head(10), 'rgba(50, 171, 96, 0.
6)')

# Creating subplots
fig = make_subplots(rows=3, cols=1,
                          subplot_titles=["Device OS - Count", "Device OS - No
n-zero Revenue Count", "Device OS - Mean Revenue"])


fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)

fig['layout'].update(height=1200, width=1200, paper_bgcolor='rgb(233,233,233)'
, title="Device Plots")
iplot(fig, filename='device-plots')
```

Device Plots



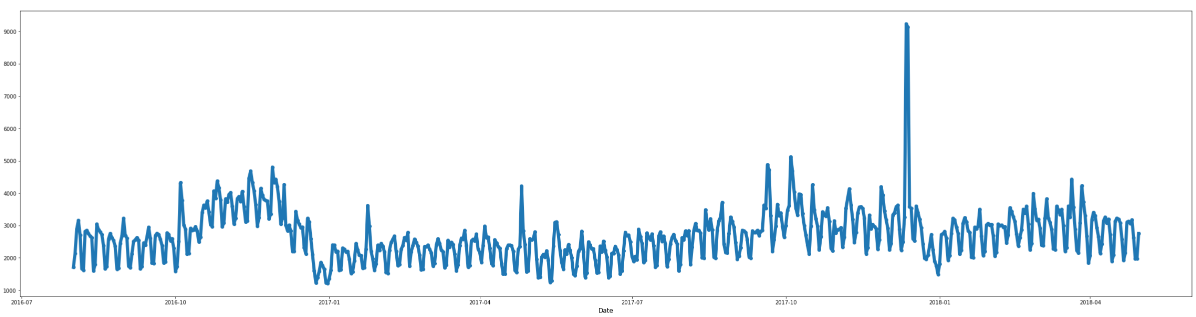






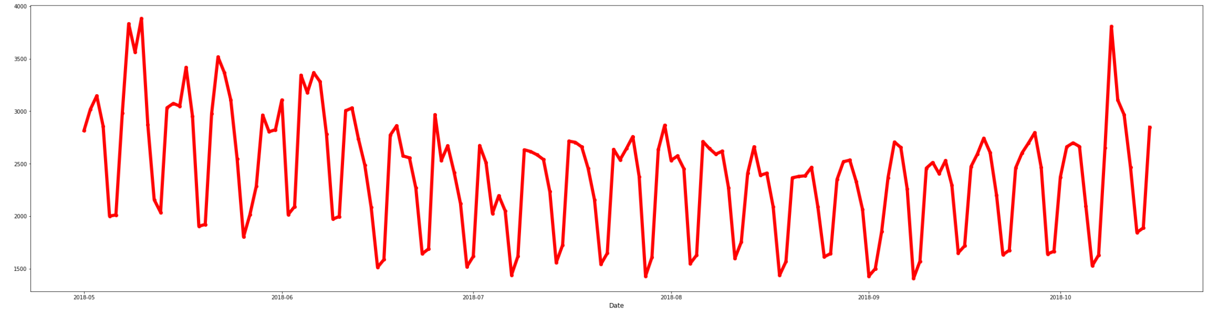
In [21]:
```
#Formatting date field
train_df["date"] = pd.to_datetime(train_df["date"], infer_datetime_format=True
, format="%Y%m%d")
```

In [74]:
```
#Plotting transaction count for a given date over the train window.
plt.figure(figsize=(40,10))

df_groupedby_date = train_df.groupby('date').count()
df_groupedby_date.reset_index(inplace=True)
plt.plot_date(x=df_groupedby_date['date'], y=df_groupedby_date['fullVisitorId'
],linestyle='solid',linewidth=6)
plt.xlabel('Date',fontsize=12)
plt.autoscale(True)
plt.show()
```



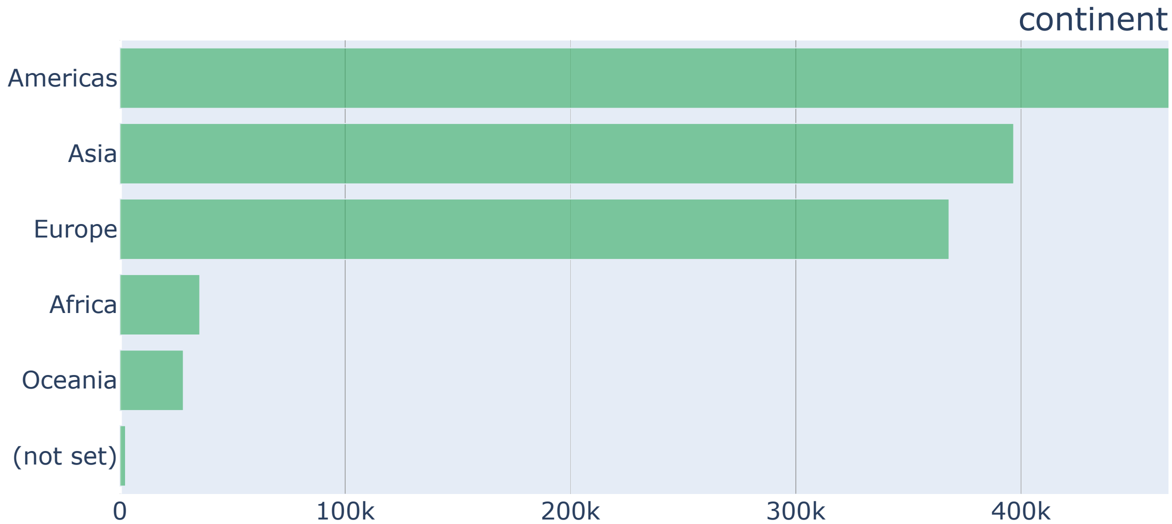
In [22]:
```
#Formatting date field
test_df["date"] = pd.to_datetime(test_df["date"], infer_datetime_format=True,
format="%Y%m%d")
```

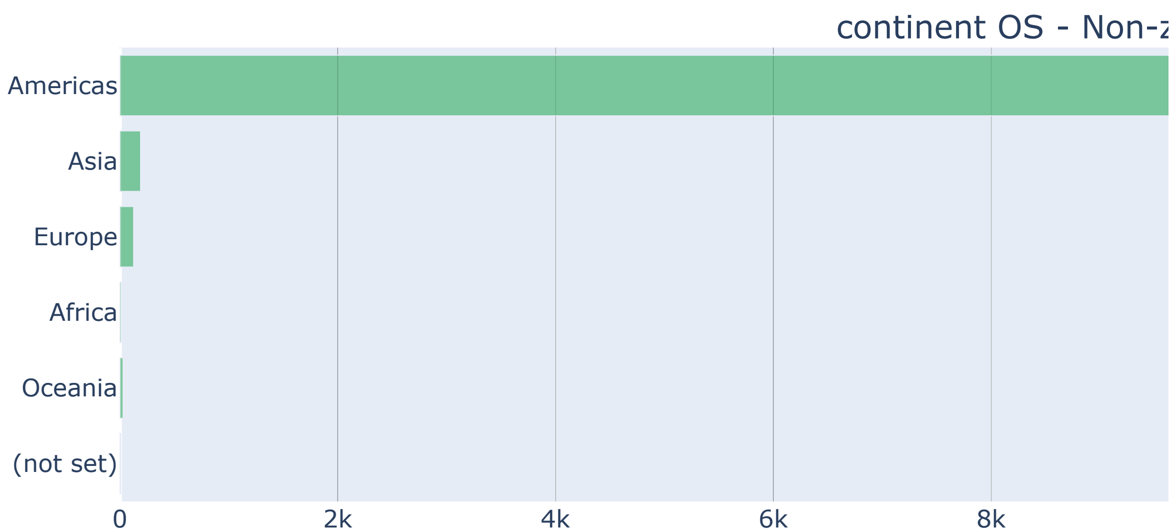
In [76]:
```
#Plotting transaction count for a given date over the test window.
plt.figure(figsize=(40,10))

df_groupedby_date = test_df.groupby('date').count()
df_groupedby_date.reset_index(inplace=True)
plt.plot_date(x=df_groupedby_date['date'], y=df_groupedby_date['fullVisitorId'
],linestyle='solid',linewidth=6, color='red')
plt.xlabel('Date',fontsize=12)
plt.autoscale(True)
plt.show()
```

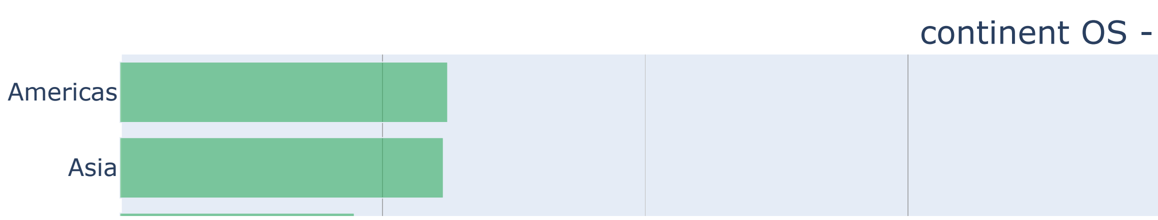


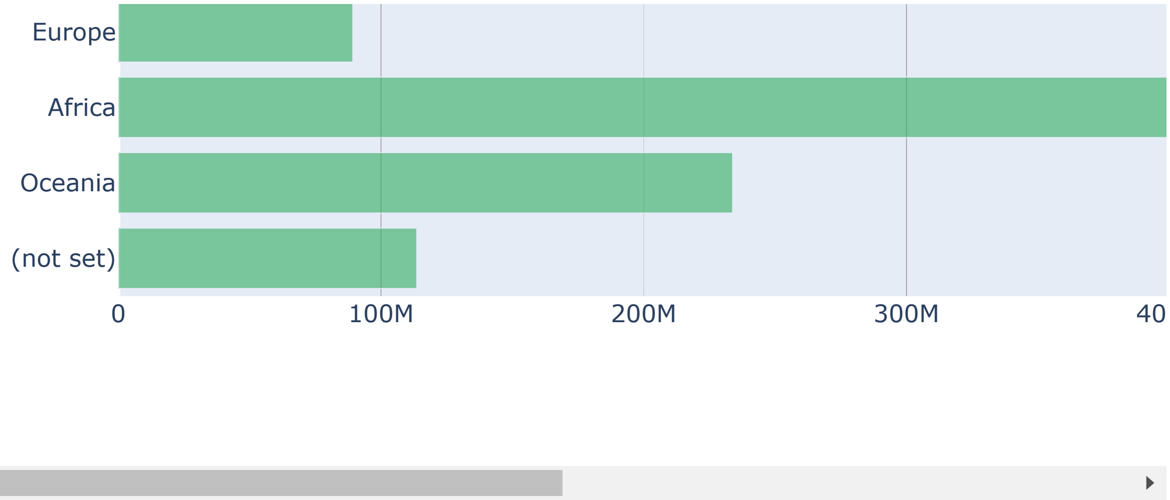
In [77]:
```
#Plotting Continent of transactions.
cnt_srs = train_df.groupby('geoNetwork.continent')['totals.transactionRevenue'
].agg(['size', 'count', 'mean'])
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"]
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
trace1 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(50, 171, 96, 0.
6)')
trace2 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head(10),
'rgba(50, 171, 96, 0.6)')
trace3 = horizontal_bar_chart(cnt_srs["mean"].head(10), 'rgba(50, 171, 96, 0.
6)')

# Creating subplots
fig = make_subplots(rows=3, cols=1,
                          subplot_titles=["continent - Count", "continent OS -
Non-zero Revenue Count", "continent OS - Mean Revenue"])


fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)

fig['layout'].update(height=1200, width=1200, paper_bgcolor='rgb(233,233,233)'
, title="Device Plots")
iplot(fig, filename='continent-plots')
```

Device Plots









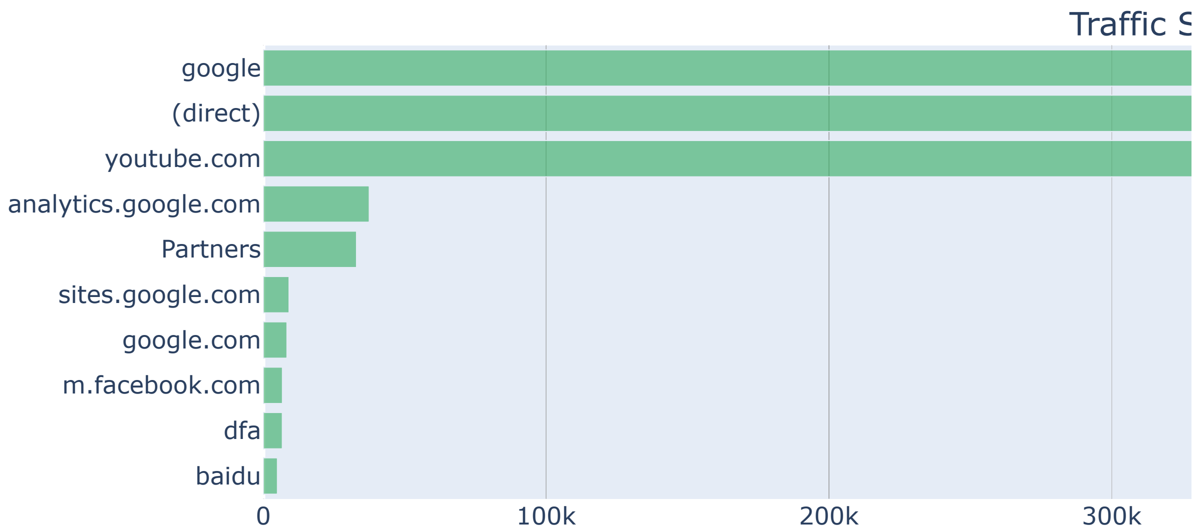
Observation -

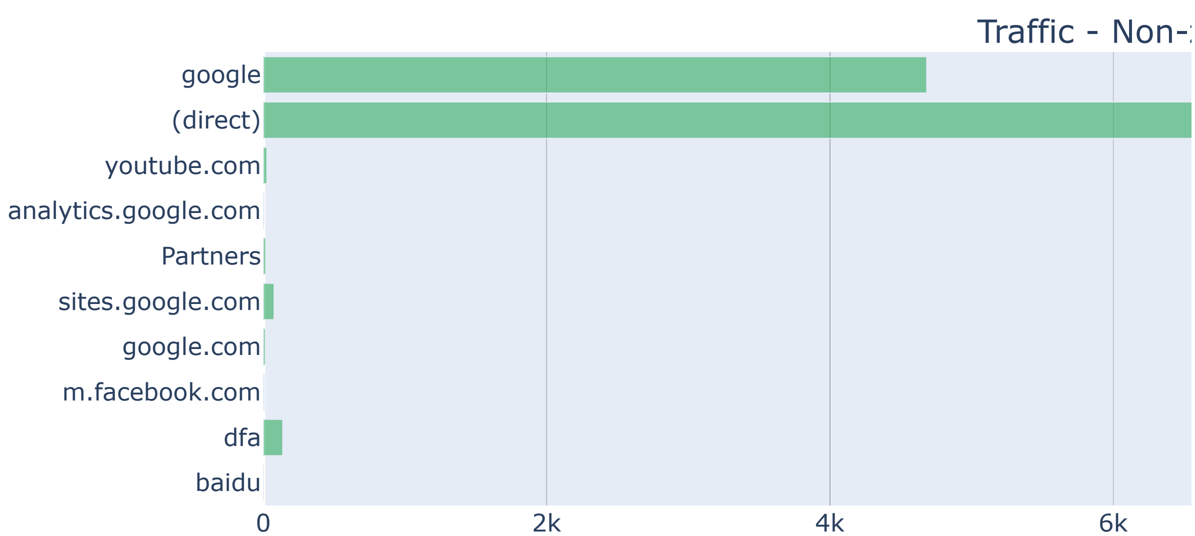
=> Maximum transactions were from Americas.

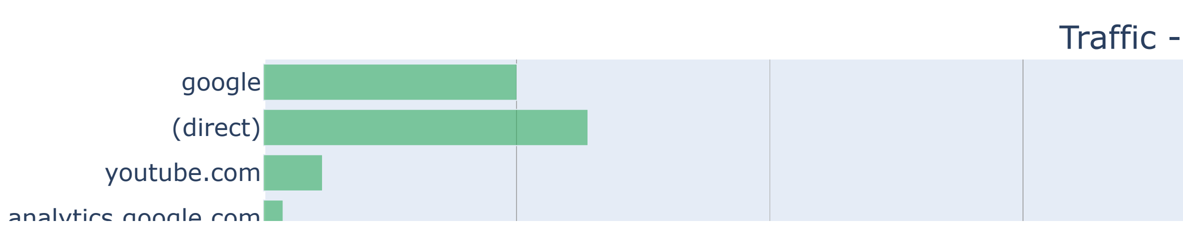
=> But the mean value of each transaction was highest in Africa.

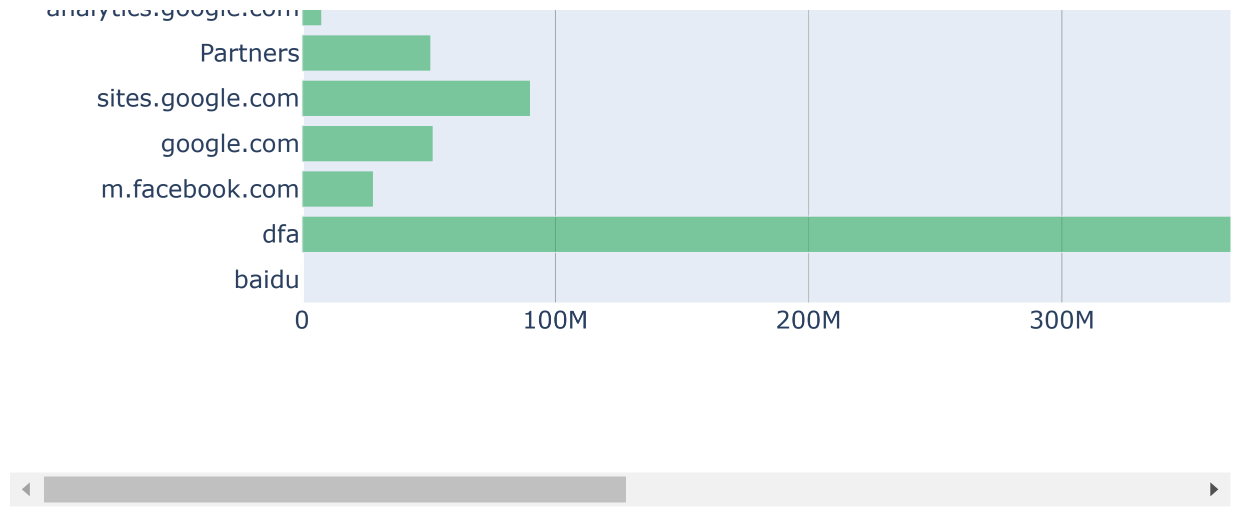
In [78]:
```
#Plotting Source of transaction
cnt_srs = train_df.groupby('trafficSource.source')['totals.transactionRevenue'
].agg(['size', 'count', 'mean']) #Groupby source and caluclate count and mean.
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"] #naming the c
olumns
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)  #Sorting the value
s on count
trace1 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(50, 171, 96, 0.
6)') #Bar chart for top-10 values.
trace2 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head(10),
'rgba(50, 171, 96, 0.6)') #Bar chart for top-10 values.
trace3 = horizontal_bar_chart(cnt_srs["mean"].head(10), 'rgba(50, 171, 96, 0.
6)') #Bar chart for top-10 values.

# Creating subplots
fig = make_subplots(rows=3, cols=1,
                          subplot_titles=["Traffic Source - Count", "Traffic -
Non-zero Revenue Count", "Traffic - Mean Revenue"])


fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)

fig['layout'].update(height=1200, width=1200, paper_bgcolor='rgb(233,233,233)'
, title="Source Plots")
iplot(fig, filename='Traffic Source-plots')
```

Device Plots









Observation -

=> Maximum transactions happened thru Google.

3. Featurization

3.1 Impute Missing Values

In [13]:
```
# Impute 0 for missing target values
train_df["totals.transactionRevenue"].fillna(0, inplace=True)
```

3.2 Convert Boolean Features

In [32]:
```
#Convert Boolean Features
train_df['device.isMobile'] = train_df['device.isMobile'].astype(bool)
test_df['device.isMobile'] = test_df['device.isMobile'].astype(bool)
```

3.3 Convert Numerical Features to Float

In [15]:
```
#convert the numerical variables to float
num_cols = ["totals.hits", "totals.pageviews", "visitNumber", "visitStartTime"
,'totals.timeOnSite','totals.transactions', 'totals.totalTransactionRevenue' ]
for col in num_cols:
    train_df[col] = train_df[col].astype(float)
    test_df[col] = test_df[col].astype(float)
```

3.4 Label Encode Categorical Features

In [16]:
```
# label encode the categorical variables and
cat_cols = ["channelGrouping", "device.browser",
            "device.deviceCategory", "device.operatingSystem",
            "geoNetwork.city", "geoNetwork.continent",
            "geoNetwork.country", "geoNetwork.metro",
            "geoNetwork.networkDomain", "geoNetwork.region",
            "geoNetwork.subContinent", "trafficSource.adContent",
            "trafficSource.campaign",
            "trafficSource.keyword", "trafficSource.medium", "totals.sessionQu
alityDim",
            "trafficSource.referralPath", "trafficSource.source", "trafficSour
ce.adwordsClickInfo.slot"]

for col in cat_cols:
    print(col)
    lbl = preprocessing.LabelEncoder()
    lbl.fit(list(train_df[col].values.astype('str')) + list(test_df[col].value
s.astype('str')))
    train_df[col] = lbl.transform(list(train_df[col].values.astype('str')))
    test_df[col] = lbl.transform(list(test_df[col].values.astype('str')))
```

```
channelGrouping
device.browser
device.deviceCategory
device.operatingSystem
geoNetwork.city
geoNetwork.continent
geoNetwork.country
geoNetwork.metro
geoNetwork.networkDomain
geoNetwork.region
geoNetwork.subContinent
trafficSource.adContent
trafficSource.campaign
trafficSource.keyword
trafficSource.medium
trafficSource.referralPath
trafficSource.source
trafficSource.adwordsClickInfo.slot
```

3.5 Featurization of Train Data

In [18]:
```
#Generate a frame with featurizations aggregating all the transactions for tha
t customer
#Source - https://github.com/HuanZhang999/GoogleAnalyticsCustomerRevenuePredic
tion/blob/master/1-%20create_train.ipynb

def getTimeFramewithFeatures(df, k=1):
    #Splitting the dataframe in the time  window of 168 days. 168 was the numb
er of days in test data.

    #Filter the rows from dataframe having dates between given window
    #K determines the frame number
    # Training dataset has dates from - August 1st 2016 to April 30th 2018.
    #for Frame-1(k=1), it would take dates from August 1st 2016 till January 1
5, 2017. (Added 168 to starting date)
    #for Frame-2(k=2), it would take dates from January 16, 2017 till July 2,
 2017. (Added 168 to starting date)
    #for Frame-3(k=3), it would take dates from July 3, 2017 till December 17,
2017. (Added 168 to starting date)
    #for Frame-3(k=4), it would take dates from December 18, 2017 till June 4,
2018. (Added 168 to starting date)
    #December 1st 2018 to January 31st 2019
    tf = df.loc[(df['date'] >= min(df['date']) + timedelta(days=168*(k-1)))
              & (df['date'] < min(df['date']) + timedelta(days=168*k))]

    #Fetch the visitor id for the users those returned in 62 days window after
46 days from the frame-end date.
    #This was done replicate the real world scenario where we have given data
 till October 15th 2018 and need to
    #determine if user returns after 46 days (December 1st 2018) and in the  s
tarting December 1st 2018 and in the
    #time-frame of 62 days from December 1st 2018 to January 31st 2019
    tf_fvid = set(df.loc[(df['date'] >= min(df['date']) + timedelta(days=168*k
+ 46 ))
                         & (df['date'] < min(df['date']) + timedelta(days=168*k
+ 46 + 62))]['fullVisitorId'])

    tf_returned = tf[tf['fullVisitorId'].isin(tf_fvid)]

    #Creating test data set with future timeframe of 62 days after 46 days fro
m end date of current timeframe
    #For e.g. - for Frame-1(k=1),August 1st 2016 till January 15, 2017. (168 d
ays),
    # test-set would start after 46 days from end date of current frame (Janua
ry 15, 2017) :- March 2, 2017
    #and would be for 62 days :- (March 2, 2017 till May 3, 2017)
    tf_tst = df[df['fullVisitorId'].isin(set(tf_returned['fullVisitorId']))
             & (df['date'] >= min(df['date']) + timedelta(days=168*k + 46))
             & (df['date'] < min(df['date']) + timedelta(days=168*k + 46 + 62
))]

    #Calculating target variable totals_transaction_revenue per customer from
 the test-set calculated above.
    tf_target = tf_tst.groupby('fullVisitorId')[['totals.totalTransactionReven
ue']].sum().apply(np.log1p, axis=1).reset_index()

    #Setting returned flag to 1, for customers present in test set
```

```
    tf_target['ret'] = 1
    tf_target.rename(columns={'totals.totalTransactionRevenue': 'target'}, inplace=True)

    #Creating dataframe for the users not present in test-set created above. This set of users signify those which haven't returned
    #for shopping in test window.
    tf_nonret = pd.DataFrame()
    tf_nonret['fullVisitorId'] = list(set(tf['fullVisitorId']) - tf_fvid)

    #Setting target variable and returned flag to 0, as customer hasn't returned for shopping in given future time window.
    tf_nonret['target'] = 0
    tf_nonret['ret'] = 0

    #Creating test-set combining values for both the customers those who returned as well as those who haven't.
    tf_target = pd.concat([tf_target, tf_nonret], axis=0).reset_index(drop=True)

    #Below max and min date would be used to generate some date based featurizations
    tf_maxdate = max(tf['date'])
    tf_mindate = min(tf['date'])
    #for the users present in current frame of train-set, calculating all the features.
    tf = tf.groupby('fullVisitorId').agg({
            'geoNetwork.networkDomain': {'networkDomain': lambda x: x.dropna().max()}, #max value of network domain
            'geoNetwork.city': {'city': lambda x: x.dropna().max()},  #max value of city
            'device.operatingSystem': {'operatingSystem': lambda x: x.dropna().max()},  #max value of Operating System
            'geoNetwork.metro': {'metro': lambda x: x.dropna().max()},  #max value of metro
            'geoNetwork.region': {'region': lambda x: x.dropna().max()},   #max vaue of region
            'channelGrouping': {'channelGrouping': lambda x: x.dropna().max()},  #max value of channel grouping
            'trafficSource.referralPath': {'referralPath': lambda x: x.dropna().max()},  #max value of referral path
            'geoNetwork.country': {'country': lambda x: x.dropna().max()}, #max value of country
            'trafficSource.source': {'source': lambda x: x.dropna().max()}, #max value of source
            'trafficSource.medium': {'medium': lambda x: x.dropna().max()}, #max value of medium
            'trafficSource.keyword': {'keyword': lambda x: x.dropna().max()}, #max value of keyboard
            'device.browser':  {'browser': lambda x: x.dropna().max()},  #max value of browser
            'device.deviceCategory': {'deviceCategory': lambda x: x.dropna().max()}, #max of device category
            'geoNetwork.continent': {'continent': lambda x: x.dropna().max()}, #max of continent value
            'totals.timeOnSite': {'timeOnSite_sum': lambda x: x.dropna().sum(),     #sum timeonsite
```

```
                                  'timeOnSite_min': lambda x: x.dropna().min
(),     #min timeonsite
                                  'timeOnSite_max': lambda x: x.dropna().max
(),     #max timeonsite
                                  'timeOnSite_mean': lambda x: x.dropna().mean
()},  #mean timeonsite
            'totals.pageviews': {'pageviews_sum': lambda x: x.dropna().sum(),
#sum of page views
                                 'pageviews_min': lambda x: x.dropna().min(),
#min of page views
                                 'pageviews_max': lambda x: x.dropna().max(),
#max of page views
                                 'pageviews_mean': lambda x: x.dropna().mean
()},  #mean of page views
            'totals.hits': {'hits_sum': lambda x: x.dropna().sum(),     #sum o
f hits
                            'hits_min': lambda x: x.dropna().min(),     #min o
f hits
                            'hits_max': lambda x: x.dropna().max(),     #max o
f hits
                            'hits_mean': lambda x: x.dropna().mean()},  #mean
 of hits
            'visitStartTime': {'visitStartTime_counts': lambda x: x.dropna().c
ount()}, #Count of visitStartTime
            'totals.sessionQualityDim': {'sessionQualityDim': lambda x: x.drop
na().max()}, #Max value of sessionQualityDim
            'device.isMobile': {'isMobile': lambda x: x.dropna().max()}, #Max
 value of isMobile
            'visitNumber': {'visitNumber_max' : lambda x: x.dropna().max()},
#Maximum number of visits.
            'totals.totalTransactionRevenue':  {'totalTransactionRevenue_sum':
lambda x:x.dropna().sum()}, #summation of all the transaction amounts.
            'totals.transactions' : {'transactions' : lambda x:x.dropna().sum
()}, #Summation of all the transaction counts.
            'date': {'first_ses_from_the_period_start': lambda x: x.dropna().m
in() - tf_mindate, #first shopping session for customer after the period end d
ate for current frame.
                     'last_ses_from_the_period_end': lambda x: tf_maxdate - x.
dropna().max(), #Last shopping session for customer before the period end date
for current frame.
                     'interval_dates': lambda x: x.dropna().max() - x.dropna()
.min(),  #interval calculated as the latest date on which customer visited - o
ldest date on which they visited.
                     'unqiue_date_num': lambda x: len(set(x.dropna())) }, #Uni
que number of dates customer visited.
                    })

    #Drop the parent level of features. for e.g. drop geoNetwork.networkDomain
and keep only 'networkDomain' which stores max value from the group.
    tf.columns = tf.columns.droplevel()
    #merging the two dataframe tf having features and tf_target having target
 variables.
    tf = pd.merge(tf, tf_target, left_on='fullVisitorId', right_on='fullVisito
rId')
    return tf
```

In [19]:
```
#Concatenate the trainn and test to create total dataframe. We are concatenati
ng as we are generating featues whether customer returned in future test windo
w. So, for that we need test data
tot_df = pd.concat([train_df, test_df], axis=0).reset_index()
```

In [ ]:
```
#Featurize 1st and second frame from train set
print('Get 1st train part')
%time tr1 = getTimeFramewithFeatures(tot_df, k=1)
tr1.to_pickle('tr1_clean')

print('Get 2nd train part')
%time tr2 = getTimeFramewithFeatures(tot_df, k=2)
tr2.to_pickle('tr2_clean')
```

```
Get 1st train part
CPU times: user 1h 49min 38s, sys: 1min 17s, total: 1h 50min 56s
Wall time: 1h 48min 32s
Get 2nd train part
CPU times: user 1h 24min 16s, sys: 54.3 s, total: 1h 25min 11s
Wall time: 1h 23min 26s
Get 3rd train part
```

In [20]:
```
#Featurize 3rd and 4th frame from train set
print('Get 3rd train part')
%time tr3 = getTimeFramewithFeatures(tot_df, k=3)
tr3.to_pickle('tr3_clean')

print('Get 4th train part')
%time tr4 = getTimeFramewithFeatures(tot_df, k=4)
tr4.to_pickle('tr4_clean')
```

```
Get 3rd train part
CPU times: user 1h 52min 24s, sys: 1min 46s, total: 1h 54min 11s
Wall time: 1h 51min 50s
Get 4th train part
CPU times: user 1h 46min 11s, sys: 1min 9s, total: 1h 47min 20s
Wall time: 1h 45min 21s
```

In [27]:
```
#Read stored featured training dataframes
tr1 = pd.read_pickle("tr1_clean")
```

In [28]:
```
#Read stored featured training dataframes.
tr2 = pd.read_pickle("tr2_clean")
```

In [29]:
```
#Shape of all the training dataframes.
tr1.shape, tr2.shape, tr3.shape, tr4.shape
```

Out[29]:
```
((377186, 39), (288869, 39), (385318, 39), (366202, 39))
```

3.6 Featurization of Test Data

In [35]:
```
### Construction of the test-set (by analogy as train-set)
print('Load test data')
#test data would have all transactions done after 01 May 2018
tr5 = tot_df[tot_df['date'] >= pd.to_datetime(20180501, infer_datetime_format=
True, format="%Y%m%d")]
#Below max and min date would be used to generate some featurizations
tf_maxdate = max(tr5['date'])    #maxmimum date in this frame
tf_mindate = min(tr5['date'])    #minmiun date in this dataframe
```

```
Load test data
```

In [36]:
```
#Generate features aggregating all the transactions for a visitor/customer.
#Source - https://github.com/HuanZhang999/GoogleAnalyticsCustomerRevenuePredic
tion/blob/master/1-%20create_train.ipynb

tr5 = tr5.groupby('fullVisitorId').agg({              #aggregate features for
 each visitor/ customer
            'geoNetwork.networkDomain': {'networkDomain': lambda x: x.dropna()
.max()},  #max value of network domain
            'geoNetwork.city': {'city': lambda x: x.dropna().max()},
#max value of city
            'device.operatingSystem': {'operatingSystem': lambda x: x.dropna()
.max()},  #max value of Operating System
            'geoNetwork.metro': {'metro': lambda x: x.dropna().max()},
#max value of metro
            'geoNetwork.region': {'region': lambda x: x.dropna().max()},
#max vaue of region
            'channelGrouping': {'channelGrouping': lambda x: x.dropna().max
()},          #max value of channel grouping
            'trafficSource.referralPath': {'referralPath': lambda x: x.dropna
().max()},  #max value of referral path
            'geoNetwork.country': {'country': lambda x: x.dropna().max()},
#max value of country
            'trafficSource.source': {'source': lambda x: x.dropna().max()},
#max value of source
            'trafficSource.medium': {'medium': lambda x: x.dropna().max()},
#max value of medium
            'trafficSource.keyword': {'keyword': lambda x: x.dropna().max()},
#max value of keyboard
            'device.browser':  {'browser': lambda x: x.dropna().max()},
#max value of browser
            'device.deviceCategory': {'deviceCategory': lambda x: x.dropna().m
ax()}, #max of device category
            'geoNetwork.continent': {'continent': lambda x: x.dropna().max()},
#max of continent value
            'totals.timeOnSite': {'timeOnSite_sum': lambda x: x.dropna().sum
(),      #sum timeonsite
                                  'timeOnSite_min': lambda x: x.dropna().min
(),      #min timeonsite
                                  'timeOnSite_max': lambda x: x.dropna().max
(),      #max timeonsite
                                  'timeOnSite_mean': lambda x: x.dropna().mean
()},   #mean timeonsite
            'totals.pageviews': {'pageviews_sum': lambda x: x.dropna().sum(),
#sum of page views
                                 'pageviews_min': lambda x: x.dropna().min(),
#min of page views
                                 'pageviews_max': lambda x: x.dropna().max(),
#max of page views
                                 'pageviews_mean': lambda x: x.dropna().mean
()}, #mean of page views
            'totals.hits': {'hits_sum': lambda x: x.dropna().sum(),     #sum o
f hits
                            'hits_min': lambda x: x.dropna().min(),     #min o
f hits
                            'hits_max': lambda x: x.dropna().max(),     #max o
f hits
```

```
                          'hits_mean': lambda x: x.dropna().mean()},  #mean
 of hits
            'visitStartTime': {'visitStartTime_counts': lambda x: x.dropna().c
ount()}, #Count of visitStartTime
            'totals.sessionQualityDim': {'sessionQualityDim': lambda x: x.drop
na().max()}, #Max value of sessionQualityDim
            'device.isMobile': {'isMobile': lambda x: x.dropna().max()},   #Ma
x value of isMobile
            'visitNumber': {'visitNumber_max' : lambda x: x.dropna().max()},
#Maximum number of visits.
            'totals.totalTransactionRevenue':  {'totalTransactionRevenue_sum':
lambda x:x.dropna().sum()}, #summation of all the transaction amounts.
            'totals.transactions' : {'transactions' : lambda x:x.dropna().sum
()},   #Summation of all the transaction counts.
            'date': {'first_ses_from_the_period_start': lambda x: x.dropna().m
in() - tf_mindate,  #first shopping session for customer after the period end
 date for current frame.
                     'last_ses_from_the_period_end': lambda x: tf_maxdate - x.
dropna().max(), #Last shopping session for customer before the period end date
for current frame.
                     'interval_dates': lambda x: x.dropna().max() - x.dropna()
.min(), #interval calculated as the latest date on which customer visited - ol
dest date on which they visited.
                     'unqiue_date_num': lambda x: len(set(x.dropna())) }, #Uni
que number of dates customer visited.
                    })
tr5.columns = tr5.columns.droplevel() #Drop the parent level of features. for
 e.g. drop geoNetwork.networkDomain and keep only 'networkDomain' which stores
max value from the group.
```

In [ ]:
```
#Set the target varaibles to nan for test data
tr5['target'] = np.nan
tr5['ret'] = np.nan
```

In [37]:
```
#Save the preprocessed test data frame.
tr5.to_pickle('tr5_clean')
```

In [38]:
```
#Concatenate all the dataframes created above to build final feature set.
final_df = pd.concat([tr1, tr2, tr3, tr4, tr5], axis=0, sort=False).reset_inde
x(drop=True)
```

In [ ]:
```
#Convert the date calulcated field in days format
final_df['interval_dates'] = final_df['interval_dates'].dt.days
final_df['first_ses_from_the_period_start'] = final_df['first_ses_from_the_per
iod_start'].dt.days
final_df['last_ses_from_the_period_end'] = final_df['last_ses_from_the_period_
end'].dt.days
```

In [39]:
```
#Save final featurized data
final_df.to_pickle('train_and_test_clean')
```

4. Read Final preprocessed features and build train and test Dataframes.

In [3]:
```
#Read final features after preprocessing
final_df = pd.read_pickle("train_and_test_clean")
```

In [4]:
```
#Classifying train dataset as the records whether target variable(transaction
 amount) is unknown.
train_df = final_df[final_df['target'].notnull()]
```

In [5]:
```
#Classifying test dataset as the records whether target variable(transaction a
mount) is unknown.
test_df = final_df[final_df['target'].isnull()]
```

In [6]:
```
#Final train set after featurization.
train_df.head()
```

Out[6]:

	fullVisitorId	pageviews_max	pageviews_sum	pageviews_mean	pageviews_min	met
0	0000010278554503158	8.0	8.0	8.0	8.0	1
1	0000020424342248747	13.0	13.0	13.0	13.0	
2	000005103959234087	8.0	8.0	8.0	8.0	1
3	0000093957001069502	2.0	2.0	2.0	2.0	1
4	0000114156543135683	1.0	1.0	1.0	1.0	

5 rows × 39 columns

5. Hyperparameter Tuning for the Classification Model to predict whether customer would return during test window

In [5]:

```
# Create parameters to search
gridParams = {
    'learning_rate': [0.005,0.01,0.015],    #Learning rate
    'n_estimators': [40,100,200],           #number of boosting iterations
    'num_leaves': [6,8,12,15,16],           #number of leaves in full tree
    'boosting_type' : ['gbdt'],
    'objective' : ['binary'],               #Binary Classification model to pr
edict whether customer will return during test window
    'metric' : ['binary_logloss'],          #Performance metric as "Binary Log
loss"
    'colsample_bytree' : [0.6, 0.8, 1],     #LightGBM will select 80% of featu
res before training each tree
    'subsample' : [0.7,0.9, 1],             #this will randomly select part of
data without resampling
    'reg_alpha' : [0,1],                    #L1 regularization
    'reg_lambda' : [0,1],                   #L2 regularization
    'max_leaves': [128,256,512],            #Maximum number of nodes to be add
ed.
    'min_child_samples' : [1,20]            #Minimum number of data points nee
ded in a child (leaf) node.
     }
```

In [8]:

```
#Define LightGBM Classifier model
model = lgb.LGBMClassifier()
```

In [8]:
```
#RandomizedSearchCV to hypertune the parameters
grid = RandomizedSearchCV(model, gridParams,
                    cv=3,
                    n_jobs=1)

# Run the Randomsearch cv on the train dataset to find tuned hyperparameters
%time grid.fit(train_df.drop(target_cols, axis=1),train_df['ret'])
```

```
CPU times: user 21min 7s, sys: 36.8 s, total: 21min 43s
Wall time: 12min 10s
```

Out[8]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=LGBMClassifier(boosting_type='gbdt',
                                            class_weight=None,
                                            colsample_bytree=1.0,
                                            importance_type='split',
                                            learning_rate=0.1, max_depth=-1,
                                            min_child_samples=20,
                                            min_child_weight=0.001,
                                            min_split_gain=0.0,
                                            n_estimators=100, n_jobs=-1,
                                            num_leaves=31, objective=None,
                                            random_state=None, reg_alpha=0.0,
                                            reg_lambda=0.0, sile...
                                        'learning_rate': [0.005, 0.01, 0.01
5],
                                        'max_leaves': [128, 256, 512],
                                        'metric': ['binary_logloss'],
                                        'min_child_samples': [1, 20],
                                        'n_estimators': [40, 100, 200],
                                        'num_leaves': [6, 8, 12, 15, 16],
                                        'objective': ['binary'],
                                        'reg_alpha': [0, 1],
                                        'reg_lambda': [0, 1],
                                        'subsample': [0.7, 0.9, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [9]:
```
# Print the best parameters found
print(grid.best_params_)
print(grid.best_score_)
```

```
{'colsample_bytree': 0.8, 'n_estimators': 200, 'learning_rate': 0.01, 'object
ive': 'binary', 'min_child_samples': 20, 'reg_alpha': 1, 'max_leaves': 256,
'reg_lambda': 1, 'boosting_type': 'gbdt', 'metric': 'binary_logloss', 'subsam
ple': 0.7, 'num_leaves': 16}
0.9938521771334851
```

6. Hyperparameter Tuning for the Regression Model to predict transaction amount

In [14]:
```
# Create parameters to be tuned
gridParams = {
    'learning_rate': [0.005,0.01,0.015],   #Learning rate
    'n_estimators': [40,100,200],          #number of boosting iterations
    'num_leaves': [6,8,12,15,16],          #number of leaves in full tree
    'boosting_type' : ['gbdt'],
    'objective' : ['regression'],          #Regression model to predict transa
ction amount
    'metric' : ['rmse'],                   #Performance metric as "RMSE
    'colsample_bytree' : [0.6, 0.8, 1],    #LightGBM will select 80% of featur
es before training each tree
    'subsample' : [0.7,0.9, 1],            #this will randomly select part of
 data without resampling
    'reg_alpha' : [0,1],                   #L1 regularization
    'reg_lambda' : [0,1],                  #L2 regularization
    'max_leaves': [128,256,512],           #Maximum number of nodes to be adde
d.
    'min_child_samples' : [1,20]           #Minimum number of data points need
ed in a child (leaf) node.
    }
```

In [10]:
```
#Define LightGBM Regressor model
model = lgb.LGBMRegressor()
```

In [20]:
```
#RandomizedSearchCV to hypertune the parameters
random_search = RandomizedSearchCV(model, gridParams,
                    cv=3,
                    n_jobs=1)

# Run the Randomsearch cv on the train dataset to find tuned hyperparameters
%time random_search.fit(train_df.drop(target_cols, axis=1)[train_df['ret']==
1], train_df['target'][train_df['ret']==1])
```

```
CPU times: user 15.8 s, sys: 240 ms, total: 16.1 s
Wall time: 8.83 s
```

Out[20]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=LGBMRegressor(boosting_type='gbdt',
                                           class_weight=None,
                                           colsample_bytree=1.0,
                                           importance_type='split',
                                           learning_rate=0.1, max_depth=-1,
                                           min_child_samples=20,
                                           min_child_weight=0.001,
                                           min_split_gain=0.0, n_estimators=1
00,
                                           n_jobs=-1, num_leaves=31,
                                           objective=None, random_state=None,
                                           reg_alpha=0.0, reg_lambda=0.0,
                                           silen...
                                        'learning_rate': [0.005, 0.01, 0.01
5],
                                        'max_leaves': [128, 256, 512],
                                        'metric': ['rmse'],
                                        'min_child_samples': [1, 20],
                                        'n_estimators': [40, 100, 200],
                                        'num_leaves': [6, 8, 12, 15, 16],
                                        'objective': ['regression'],
                                        'reg_alpha': [0, 1],
                                        'reg_lambda': [0, 1],
                                        'subsample': [0.7, 0.9, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [21]:
```
# Print the best parameters found
print(random_search.best_params_)
print(random_search.best_score_)
```

```
{'colsample_bytree': 0.8, 'n_estimators': 200, 'learning_rate': 0.01, 'object
ive': 'regression', 'min_child_samples': 1, 'reg_alpha': 1, 'max_leaves': 12
8, 'reg_lambda': 1, 'boosting_type': 'gbdt', 'metric': 'rmse', 'subsample':
1, 'num_leaves': 8}
0.07429798613085208
```

7. Run Final Model with Hyper tuned Parameters and final dataset after featurization

In [7]:
```
#Parameters for Classification model to predict whether customer would return
 during test window after hyper-parameter tuning.
params_lgb1 = {
        "objective" : "binary",               #Binary Classification model to
predict whether customer will return during test window
        "metric" : "binary_logloss",          #Performance metric as "Binary
 Logloss"
        "max_leaves": 256,                    #Maximum number of nodes to be
 added.
        "num_leaves" : 16,                    #number of leaves in full tree
        "min_child_samples" : 20,             #Minimum number of data points
 needed in a child (leaf) node.
        "learning_rate" : 0.01,               #Learning rate
        "subsample" : 0.7,                    #this will randomly select part
of data without resampling
        "colsample_bytree" : 0.8,             #LightGBM will select 80% of fe
atures before training each tree
        "bagging_frequency" : 1,              #Perform bagging at every k ite
ration
        "n_estimators" : 200,                 #number of boosting iterations
        "reg_alpha" : 1,                      #L1 regularization
        "reg_lambda": 1,                      #L2 regularization
        "boosting_type" : "gbdt"}
```

In [8]:
```
#Parameters for Regression model to predict transaction amount returned after
 hyper-parameter tuning.
params_lgb2 = {
        "objective" : "regression",                #Regression model to predi
ct transaction amount
        "metric" : "rmse",                         #Performance metric as "RM
SE"
        "max_leaves": 128,                         #Maximum number of nodes t
o be added.
        "num_leaves" : 8,                          #number of leaves in full
 tree
        "min_child_samples" : 1,                   #Minimum number of data po
ints needed in a child (leaf) node.
        "learning_rate" : 0.01,                    #Learning rate
        "subsample" : 1,                           #this will randomly select
part of data without resampling
        "colsample_bytree" : 0.8,                  #LightGBM will select 80%
 of features before training each tree
        "bagging_frequency" : 1,                   #Perform bagging at every
 k iteration
        "n_estimators" : 200,                      #number of boosting iterat
ions
        "reg_alpha" : 1,                           #L1 regularization
        "reg_lambda": 1,                           #L2 regularization
        "boosting_type" : "gbdt"}
```

In [14]:
```
target_cols = ['target', 'ret', 'fullVisitorId', 'sessionQualityDim']  #Column
s to be dropped from final dataset

#Define dataset for Classification model to determine whether customer would r
eturn during test time window.
dtrain_ret = lgb.Dataset(train_df.drop(target_cols, axis=1), label=train_df['r
et'])

#Define dataset for Regression model, picking only the customers who returned
 during test time window.
dtrain_amt = lgb.Dataset(train_df.drop(target_cols, axis=1)[train_df['ret']==1
],
                         label=train_df['target'][train_df['ret']==1])
```

In [26]:
```
#Running Lightgbm model for 10 iterations and took average of those.
#Source :- https://www.kaggle.com/kostoglot/winning-solution
pr_lgb_sum = 0    #Variable to store predictions.
print('Training and predictions')
for i in range(10):     #Running the model for 10 iterations and would be taki
ng average of those as final value.
    print('Interation number ', i)

    #Classification model to predict whether customer will return in test wind
ow.
    lgb_model1 = lgb.train(params_lgb1, dtrain_ret)
    pr_lgb = lgb_model1.predict(test_df.drop(target_cols, axis=1))
    lgb_model1.save_model('lgb_model1_itr_' + str(i) + '.txt' )

    #Classification model to predict the transaction amount for the customers
 who returned in that time window.
    lgb_model2 = lgb.train(params_lgb2, dtrain_amt)
    pr_lgb_ret = lgb_model2.predict(test_df.drop(target_cols, axis=1))
    lgb_model2.save_model('lgb_model2_itr_' + str(i) + '.txt' )

    #Calculating final prediction as product of above two amounts.
    pr_lgb_sum = pr_lgb_sum + pr_lgb*pr_lgb_ret

#Taking average value from above iterations the model was run.
pr_final2 = pr_lgb_sum/10
```

```
Training and predictions
Interation number  0
Interation number  1
Interation number  2
Interation number  3
Interation number  4
Interation number  5
Interation number  6
Interation number  7
Interation number  8
Interation number  9
```

In [23]:
```
test_df.columns
```

Out[23]:
```
Index(['fullVisitorId', 'pageviews_max', 'pageviews_sum', 'pageviews_mean',
       'pageviews_min', 'metro', 'source', 'region', 'browser', 'referralPat
h',
       'deviceCategory', 'operatingSystem', 'isMobile', 'networkDomain',
       'medium', 'hits_sum', 'hits_max', 'hits_min', 'hits_mean',
       'unqiue_date_num', 'interval_dates', 'last_ses_from_the_period_end',
       'first_ses_from_the_period_start', 'keyword', 'visitStartTime_counts',
       'timeOnSite_mean', 'timeOnSite_min', 'timeOnSite_max', 'timeOnSite_su
m',
       'channelGrouping', 'transactions', 'city', 'continent', 'country',
       'sessionQualityDim', 'totalTransactionRevenue_sum', 'visitNumber_max',
       'target', 'ret'],
      dtype='object')
```

In [25]:
```
i=1
```

Out[25]:
```
<lightgbm.basic.Booster at 0x7ff8e923f978>
```

In [22]:
```
#Saving the model
lgb_model1.save_model('lgb_model1.txt')
lgb_model2.save_model('lgb_model2.txt')
```

Out[22]:
```
<lightgbm.basic.Booster at 0x7ff8e923f588>
```

In [20]:
```
#Loading the saved models
lgb_model1 = lgb.Booster(model_file='lgb_model1.txt')
lgb_model2 = lgb.Booster(model_file='lgb_model2.txt')
```

In [28]:
```
#Shape of predictions variable
pr_final2.shape
```

Out[28]:
```
(296530,)
```

In [27]:
```
#Writting predictions in csv file
#Format - fullVisitorId    PredictedLogRevenue
#   0000018966949534117               0.005551
pred_df = pd.DataFrame({"fullVisitorId":test_df["fullVisitorId"].values})
pred_df["PredictedLogRevenue"] = pr_final2
pred_df.columns = ["fullVisitorId", "PredictedLogRevenue"]
pred_df.to_csv("pred_lgb_2.csv", index=False)
```

In [22]:
```
#Sample predictions for some test records.
pred_df.head()
```

Out[22]:

	fullVisitorId	PredictedLogRevenue
0	0000018966949534117	0.005551
1	0000039738481224681	0.002812
2	0000073585230191399	0.002449
3	0000087588448856385	0.001243
4	0000149787903119437	0.001197

8. Steps Followed

- Loaded Train and test datasets from the provided files.
- As the data was huge, so initially took 100k train data, did some feature analysis and loaded only useful features from full train and test data.
- Did exploratory data analysis on key features and made observations.
- Featurization:-
  - Imputed missing Transaction revenue with 0 wherever it was missing.
  - Converted Numerical features to float.
  - Label Encoded Categorical variables.
  - We featurized train and test data into time windows of 168 days (168 is the number of days in test window)

```
 - Test window - May 1st 2018 to October 15th 2018. - 168 days (One frame of data)
 - Train Window - August 1st 2016 to April 30th 2018. (638 days = 638/ 168 = 4 timeframes.)
```

  - In each time window, we have aggregated features at a unqiue customer level(Differentiated by visitor id)
  - Concatenated all the timeframe of features generated above and built train and test dataframe.
- Ran final Model -
  - We took LightGBM as the Machine Learning model. As we had huge data to train, Light GBM provided faster training speed, higher efficiency, low memory usage, better accuracy.
  - We used to models to do the prediction :-

```
 - Model-1 :- Classification Model
              Used to classify whether customer would return for shopping in given future time window.
 - Model-2 :- Regression Model
              Used to predict the transaction revenue amount for customers which returned for shopping.
```

  - We hypertuned the above data using RandomizedSearchCV.
  - The final prediction was calculated as the product of below two components

```
     => probability the customer would return for shopping and
     => The predicted transaction amount revenue.
```

  - Ran 10 iterations for the final model and took average of those.
- Written predictions for test data in "pred_lgb_2.csv"

9. Results

- Above submission resulted in Private Score of 0.88298 which could have ranked 5th (Out of 1089 submissions) in Private Leadership board.

Result

In [ ]:

```
```

In [ ]:

```
```