Self Case study 1: Google Analytics Customer Revenue Prediction

Predict how much GStore customers will spend

About Gstore

Challeng is to analyze a Google Merchandise Store (also known as GStore, where Google swag is sold) customer dataset to predict revenue per customer. Hopefully, the outcome will be more actionable operational changes and a better use of marketing budgets for those companies who choose to use data analysis on top of GA data.

Problem Statement:

To analyze a Google Merchandise Store (also known as GStore, where Google swag is sold) customer dataset to predict revenue per customer. The analysis on the customer data can be used to make better marketing budgets for those companies who choose to use data analysis on top of GA data.

Dataset:

Google is being activelt tracking the user, their search keywords to the demographic reports to know who are the customers, what are their puchase patterns, their likes and dislikes to collect all the possible datas from the users. There data are than analysed to traget to specific customers for specific products and services so as to increse the revenue and to provide better products and services to the customers.

This dataset provided by Google:

- **Train_v2.csv** - Training set - contains user transactions from August 1st 2016 to April 30th 2018.
- **test_v2.csv** - Test set - contains user transactions from May 1st 2018 to October 15th 2018.
- **sample_submission_v2.csv** - Submission file in the correct format. Contains all fullVisitorIds in test_v2.csv.

The data is shared in big query and csv format. The csv files contains some filed with json objects.

,

DataFields

- **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** - The Geography of the user.
- **socialEngagementType** - Engagement type, either "Socially Engaged" or "Not Socially Engaged".
- **totals** - Aggregate values across the session.
- **trafficSource** - Traffic Source from which the session originated
- **visitId** - An identifier for this session. This is only unique to the user. For a completely unique ID 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** - Provides a record of all page visits.
- **customDimensions** - 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** - Includes high-level aggregate data.

What am I predicting?

PredictedLogRevenue each of these fullVisitorIds for the timeframe of December 1st 2018 to January 31st 2019.

Type of Machine learning problem:

- To predict the logRevenue, hence this is a regression problem
- Minimize the root mean square error

Performance Matrix:

Source: https://www.kaggle.com/c/ga-customer-revenue-prediction/overview/evaluation (https://www.kaggle.com/c/ga-customer-revenue-prediction/overview/evaluation)

Metric(s):

- RMSE (Root Mean Squared Error)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

References:

- https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields/notebook.

Importing Libraries

In [1]:

```
import pandas as pd
import json
from pandas.io.json import json_normalize
import pickle as pkl
import numpy as np
import matplotlib.pyplot as plt
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
from plotly import tools
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from wordcloud import WordCloud
from sklearn.preprocessing import OneHotEncoder
import math

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
import lightgbm as lgb
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from prettytable import PrettyTable
```

Importing Dataset

In [2]:

```
#reading csv files
train_data = pd.read_csv('train_v2.csv')
test_data = pd.read_csv('test_v2.csv')
```

Train data

In [3]:

```
train_data.head() #diplaying top 5 rows of train data
```

Out[3]:

	channelGrouping	date	device	fullVisitorId	geoNetwork	
0	Organic Search	20160902	{"browser": "Chrome", "browserVersion": "not a...	1131660440785968503	{"continent": "Asia", "subContinent": "Western...	11316604
1	Organic Search	20160902	{"browser": "Firefox", "browserVersion": "not ...	377306020877927890	{"continent": "Oceania", "subContinent": "Aust...	3773060
2	Organic Search	20160902	{"browser": "Chrome", "browserVersion": "not a...	3895546263509774583	{"continent": "Europe", "subContinent": "South...	38955462
3	Organic Search	20160902	{"browser": "UC Browser", "browserVersion": "n...	4763447161404445595	{"continent": "Asia", "subContinent": "Southea...	47634471
4	Organic Search	20160902	{"browser": "Chrome", "browserVersion": "not a...	27294437909732085	{"continent": "Europe", "subContinent": "North...	272944

Test data

In [4]:

```
test_data.head()  #diplaying top 5 datapoints of test data
```

Out[4]:

	channelGrouping	date	device	fullVisitorId	geoNetwork	
0	Organic Search	20171016	{"browser": "Chrome", "browserVersion": "not a...	6167871330617112363	{"continent": "Asia", "subContinent": "Southea...	61678713
1	Organic Search	20171016	{"browser": "Chrome", "browserVersion": "not a...	0643697640977915618	{"continent": "Europe", "subContinent": "South...	06436976
2	Organic Search	20171016	{"browser": "Chrome", "browserVersion": "not a...	6059383810968229466	{"continent": "Europe", "subContinent": "Weste...	60593838
3	Organic Search	20171016	{"browser": "Safari", "browserVersion": "not a...	2376720078563423631	{"continent": "Americas", "subContinent": "Nor...	23767200
4	Organic Search	20171016	{"browser": "Safari", "browserVersion": "not a...	2314544520795440038	{"continent": "Americas", "subContinent": "Nor...	23145445

1. JSON features in the train and test data: 'device', 'geoNetwork', 'totals', 'trafficSource'.(JSON is a syntax for storing and exchanging data.)
2. We first need to convert these columns.
3. Specify the datatype of fullVisitorIs as string.

In [5]:

```
#https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields/notebook.
json_columns = ['device', 'geoNetwork','totals', 'trafficSource']  #these 4 columns are of
def load_dataframe(filename):
    df = pd.read_csv(filename, converters={column: json.loads for column in json_columns},
                     dtype={'fullVisitorId': 'str'}) #loading json columns and specifying t

    for column in json_columns:
        column_as_df = json_normalize(df[column])  #normalizing json columns
        column_as_df.columns = [f"{column}_{subcolumn}" for subcolumn in column_as_df.colum
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    return df
```

In [6]:

```
train_data_conv = load_dataframe('train.csv')
test_data_conv = load_dataframe('test.csv')
```

Train data

In [7]:

```
train_data_conv.head(3)
```

Out[7]:

	channelGrouping	date	fullVisitorId	sessionId	socialE
0	Organic Search	20160902	1131660440785968503	1131660440785968503_1472830385	Not
1	Organic Search	20160902	377306020877927890	377306020877927890_1472880147	Not
2	Organic Search	20160902	3895546263509774583	3895546263509774583_1472865386	Not

3 rows × 55 columns

Test data

In [8]:

```
test_data_conv.head(3)
```

Out[8]:

	channelGrouping	date	fullVisitorId	sessionId	socialE
0	Organic Search	20171016	6167871330617112363	6167871330617112363_1508151024	Not
1	Organic Search	20171016	0643697640977915618	0643697640977915618_1508175522	Not
2	Organic Search	20171016	6059383810968229466	6059383810968229466_1508143220	Not

3 rows × 53 columns

In [9]:

```
#saving the converted dataframes in csv files
train_data_conv.to_csv('train_data_conv.csv')
test_data_conv.to_csv('test_data_conv.csv')
```

Exploratory Data Analysis

The Dataset containes data of either numeric type or Categorical data.

1. Columns that contains Numerical data in train and test datasets are:

In [10]:

```
numeric_features_train = train_data_conv.select_dtypes(include=[np.number]) #getting the co
print("In Train data",numeric_features_train.columns)
print("-"*100)
numeric_features_test = test_data_conv.select_dtypes(include=[np.number]) #getting the colu
print("In Test data",numeric_features_test.columns)
```

```
In Train data Index(['date', 'visitId', 'visitNumber', 'visitStartTime'], dt
ype='object')
----------------------------------------------------------------------------
------------------------
In Test data Index(['date', 'visitId', 'visitNumber', 'visitStartTime'], dty
pe='object')
```

2. Columns that contains Categorical data in train and test datasets are:

In [11]:

```
categorical_features_train = train_data_conv.select_dtypes(include=[np.object]) #getting th
print("In Train data",numeric_features_train.columns)
print("-"*100)
categorical_features_test = test_data_conv.select_dtypes(include=[np.object]) #getting the
print("In Test data",numeric_features_test.columns)
```

```
In Train data Index(['date', 'visitId', 'visitNumber', 'visitStartTime'], dt
ype='object')
----------------------------------------------------------------------------
------------------------
In Test data Index(['date', 'visitId', 'visitNumber', 'visitStartTime'], dty
pe='object')
```

3. Getting the percentage of the Missing (NaN) Values of all the features in train and test data

In [12]:

```
#https://www.kaggle.com/kailex/r-eda-for-gstore-glm-keras-xgb
features = train_data_conv.select_dtypes(include=[np.number,np.object]) #getting all the fe
total_null = features.isnull().sum().sort_values(ascending=False)  #sorting the features in
null_percentage = (features.isnull().sum()/features.isnull().count()).sort_values(ascending
missing_data = pd.concat([total_null, null_percentage], axis=1,join='outer', keys=['Total M
missing_data.index.name ='Features'  #indexing the datframe as 'Features'
missing_data
```

Out[12]:

Features	Total Missing Count	% of Total Observations
trafficSource_campaignCode	903652	99.999889
trafficSource_adContent	892707	98.788694
totals_transactionRevenue	892138	98.725728
trafficSource_adwordsClickInfo.isVideoAd	882193	97.625195
trafficSource_adwordsClickInfo.adNetworkType	882193	97.625195
trafficSource_adwordsClickInfo.slot	882193	97.625195
trafficSource_adwordsClickInfo.page	882193	97.625195
trafficSource_adwordsClickInfo.gclId	882092	97.614018
trafficSource_isTrueDirect	629648	69.678073
trafficSource_referralPath	572712	63.377425
trafficSource_keyword	502929	55.655102
totals_bounces	453023	50.132407
totals_newVisits	200593	22.198012
totals_pageviews	100	0.011066
sessionId	0	0.000000
device_operatingSystem	0	0.000000
device_mobileDeviceMarketingName	0	0.000000
device_mobileDeviceInfo	0	0.000000
device_mobileInputSelector	0	0.000000
device_mobileDeviceModel	0	0.000000
date	0	0.000000
device_mobileDeviceBranding	0	0.000000
device_operatingSystemVersion	0	0.000000
device_browserVersion	0	0.000000
device_browserSize	0	0.000000
socialEngagementType	0	0.000000
device_browser	0	0.000000
visitStartTime	0	0.000000
visitNumber	0	0.000000

Features	Total Missing Count	% of Total Observations
visitId	0	0.000000
device_language	0	0.000000
fullVisitorId	0	0.000000
device_flashVersion	0	0.000000
geoNetwork_region	0	0.000000
device_screenColors	0	0.000000
device_screenResolution	0	0.000000
trafficSource_adwordsClickInfo.criteriaParameters	0	0.000000
trafficSource_medium	0	0.000000
trafficSource_source	0	0.000000
trafficSource_campaign	0	0.000000
totals_hits	0	0.000000
totals_visits	0	0.000000
geoNetwork_networkLocation	0	0.000000
geoNetwork_longitude	0	0.000000
geoNetwork_latitude	0	0.000000
geoNetwork_networkDomain	0	0.000000
geoNetwork_cityId	0	0.000000
geoNetwork_city	0	0.000000
geoNetwork_metro	0	0.000000
geoNetwork_country	0	0.000000
geoNetwork_subContinent	0	0.000000
geoNetwork_continent	0	0.000000
device_deviceCategory	0	0.000000
channelGrouping	0	0.000000

In [13]:

```
features = test_data_conv.select_dtypes(include=[np.number,np.object]) #getting all the fea
total_null = features.isnull().sum().sort_values(ascending=False)  #sorting the features in
null_percentage = (features.isnull().sum()/features.isnull().count()).sort_values(ascending
missing_data = pd.concat([total_null, null_percentage], axis=1,join='outer', keys=['Total M
missing_data.index.name ='Features'  #indexing the datframe as 'Features'
missing_data
```

Out[13]:

Features	Total Missing Count	% of Total Observations
trafficSource_adContent	750893	93.315264
trafficSource_adwordsClickInfo.isVideoAd	750870	93.312406
trafficSource_adwordsClickInfo.adNetworkType	750870	93.312406
trafficSource_adwordsClickInfo.slot	750870	93.312406
trafficSource_adwordsClickInfo.page	750870	93.312406
trafficSource_adwordsClickInfo.gclId	750822	93.306441
trafficSource_referralPath	569361	70.755850
trafficSource_isTrueDirect	544171	67.625428
trafficSource_keyword	391032	48.594479
totals_bounces	383736	47.687788
totals_newVisits	200314	24.893499
totals_pageviews	139	0.017274
socialEngagementType	0	0.000000
device_operatingSystem	0	0.000000
device_mobileDeviceMarketingName	0	0.000000
device_mobileDeviceInfo	0	0.000000
device_mobileInputSelector	0	0.000000
date	0	0.000000
device_mobileDeviceModel	0	0.000000
device_mobileDeviceBranding	0	0.000000
device_operatingSystemVersion	0	0.000000
device_browserSize	0	0.000000
sessionId	0	0.000000
device_browserVersion	0	0.000000
device_browser	0	0.000000
visitStartTime	0	0.000000
visitNumber	0	0.000000
visitId	0	0.000000
device_language	0	0.000000

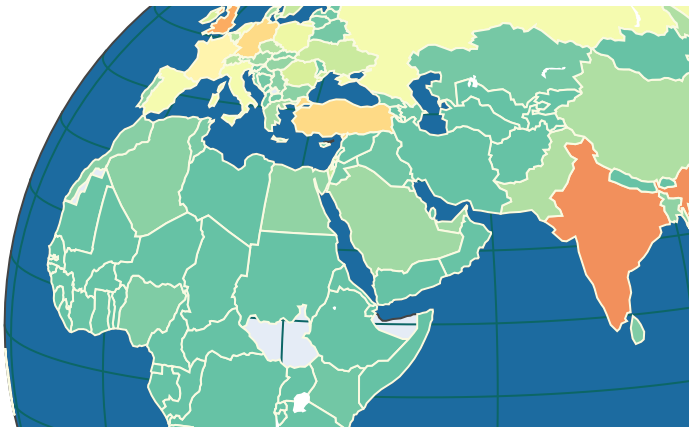
Features	Total Missing Count	% of Total Observations
fullVisitorId	0	0.000000
device_flashVersion	0	0.000000
geoNetwork_country	0	0.000000
device_screenColors	0	0.000000
device_screenResolution	0	0.000000
trafficSource_adwordsClickInfo.criteriaParameters	0	0.000000
trafficSource_medium	0	0.000000
trafficSource_source	0	0.000000
trafficSource_campaign	0	0.000000
totals_hits	0	0.000000
totals_visits	0	0.000000
geoNetwork_networkLocation	0	0.000000
geoNetwork_longitude	0	0.000000
geoNetwork_latitude	0	0.000000
geoNetwork_networkDomain	0	0.000000
geoNetwork_cityId	0	0.000000
geoNetwork_city	0	0.000000
geoNetwork_metro	0	0.000000
geoNetwork_region	0	0.000000
geoNetwork_subContinent	0	0.000000
geoNetwork_continent	0	0.000000
device_deviceCategory	0	0.000000
channelGrouping	0	0.000000

4. Visulaizing customers' distribution(location) countrywise

In [14]:

```
#https://www.kaggle.com/kabure/exploring-the-consumer-patterns-ml-pipeline
#how to create rotating globe in python - https://towardsdatascience.com/a-complete-guide-t
# RGB color code - https://www.w3schools.com/colors/colors_rgb.asp?color=rgb(0,%20191,%2025
colorscale = [[0, 'rgb(102,194,165)'], [0.0005, 'rgb(102,194,165)'],
              [0.01, 'rgb(171,221,164)'], [0.02, 'rgb(230,245,152)'],
              [0.04, 'rgb(255,255,191)'], [0.05, 'rgb(254,224,139)'],
              [0.10, 'rgb(253,174,97)'], [0.25, 'rgb(213,62,79)'], [1.0, 'rgb(158,1,66)']]

data = [ dict(
        type = 'choropleth',
        autocolorscale = False,
        colorscale = colorscale,
        showscale = True,
        locations = train_data_conv["geoNetwork_country"].value_counts().index,
        locationmode = 'country names',
        z = train_data_conv["geoNetwork_country"].value_counts().values,
        marker = dict(
            line = dict(color = 'rgb(250,250,225)', width = 1)),
            colorbar = dict( title = 'Customer Visits ')
            )
       ]

layout = dict(
    height=600,
    title = 'Countrywise Customer Visit Distribution',
    geo = dict(
        showframe = True,
        showocean = True,
        oceancolor = 'rgb(28,107,160)',
        projection = dict(
        type = 'orthographic',
            rotation = dict(
                    lon = 50,
                    lat = 10),
        ),
        lonaxis =  dict(
                showgrid = True,
                gridcolor = 'rgb(12, 102, 102)'
            ),
        lataxis = dict(
                showgrid = True,
                gridcolor = 'rgb(12, 102, 102)'
                )
            ),
        )
fig = dict(data=data, layout=layout)
iplot(fig)
```

Countrywise Customer Visit Distribution

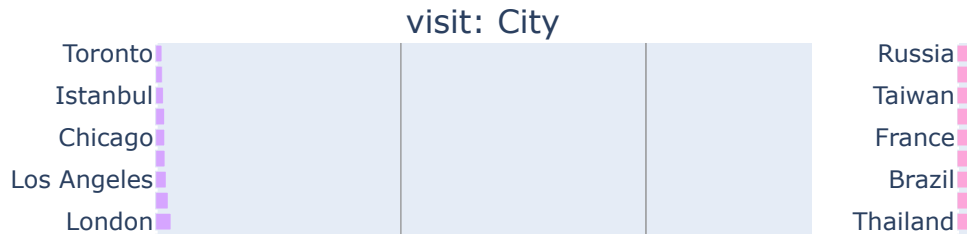


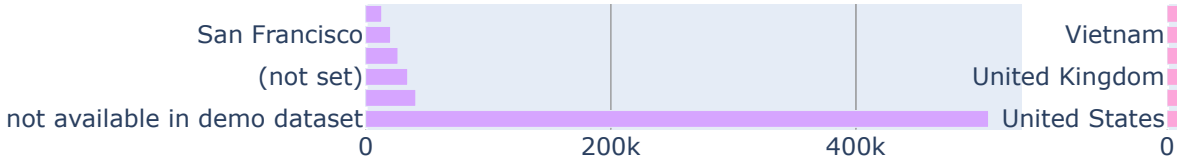
5. Visualizing the mean revenue generated city, country, subcontinent and continent wise

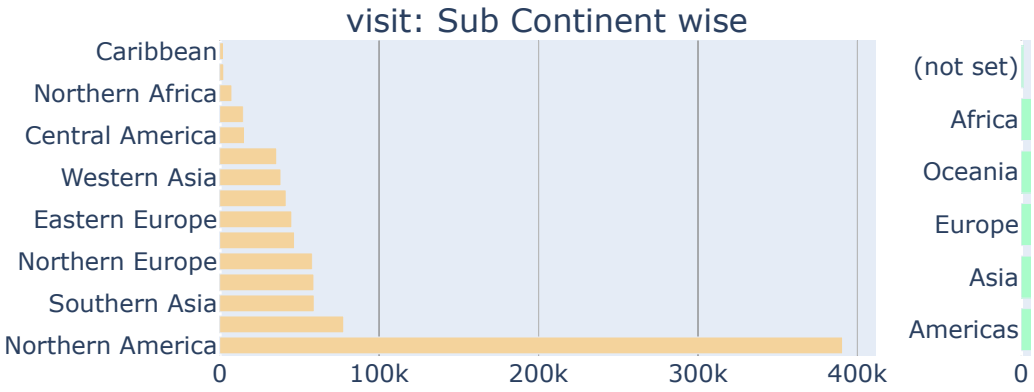
In [15]:

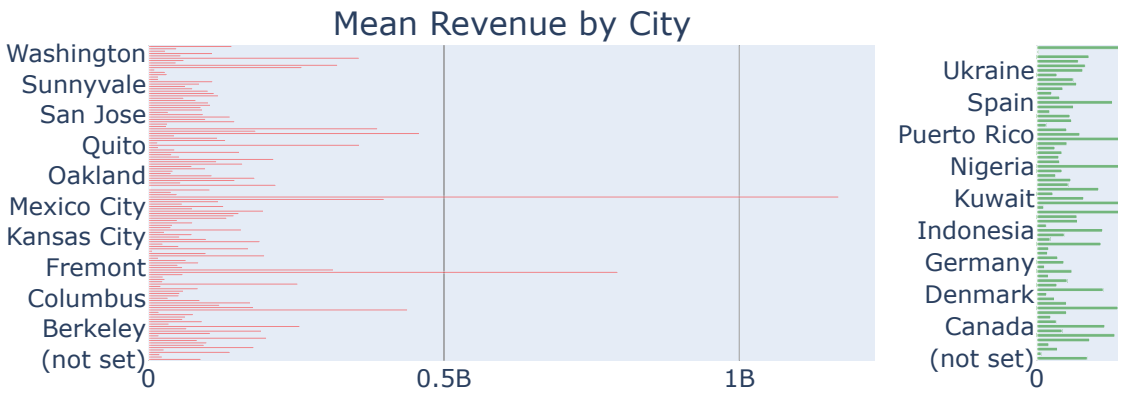
```
#https://plot.ly/python/choropleth-maps/
geo_cols = ["geoNetwork_city", "geoNetwork_country", "geoNetwork_subContinent", "geoNetwork
colors = ["#d6a5ff", "#fca6da", "#f4d39c", "#a9fcca"]
traces = []
for i, col in enumerate(geo_cols):
    t = train_data_conv[col].value_counts()
    traces.append(go.Bar(marker=dict(color=colors[i]),orientation="h", y = t.index[:15], x

fig = tools.make_subplots(rows=2, cols=2,
                          subplot_titles=["visit: City", "visit: Country wise","visit: Sub
                          , print_grid=False)
fig.append_trace(traces[0], 1, 1)
fig.append_trace(traces[1], 1, 2)
fig.append_trace(traces[2], 2, 1)
fig.append_trace(traces[3], 2, 2)

fig['layout'].update(height=600,width=1000, showlegend=False)
iplot(fig)

train_data_conv["totals_transactionRevenue"] = train_data_conv["totals_transactionRevenue"]

fig = tools.make_subplots(rows=2, cols=2, subplot_titles=["Mean Revenue by City", "Mean Rev

colors = ["red", "green", "purple","blue"]
trs = []
for i, col in enumerate(geo_cols):
    tmp = train_data_conv.groupby(col).agg({"totals_transactionRevenue": "mean"}).reset_ind
    tmp = tmp.dropna()
    tr = go.Bar(x = tmp["Mean Revenue"], orientation="h", marker=dict(opacity=0.5, color=co
    trs.append(tr)

fig.append_trace(trs[0], 1, 1)
fig.append_trace(trs[1], 1, 2)
fig.append_trace(trs[2], 2, 1)
fig.append_trace(trs[3], 2, 2)

fig['layout'].update(height=600,width=1000, showlegend=False)
iplot(fig)
```

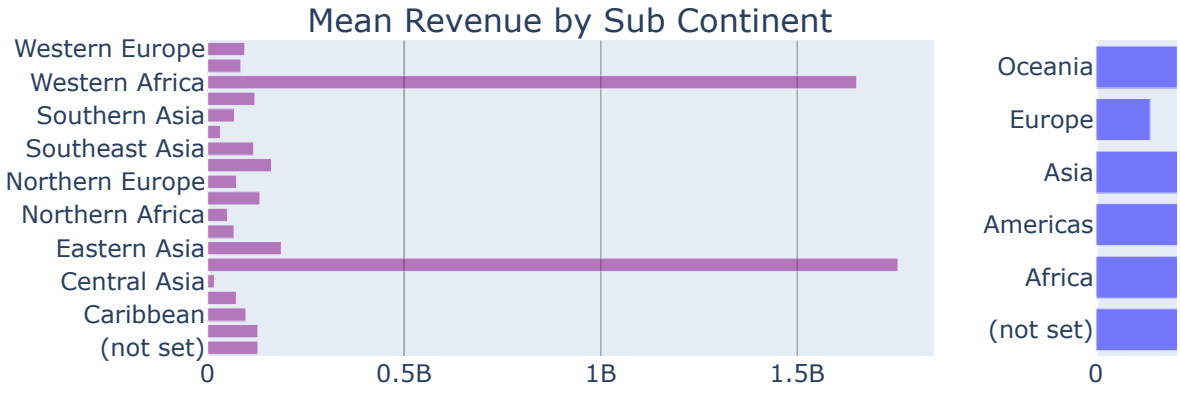
```
C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages\plotly\tools.py:465: De
precationWarning:

plotly.tools.make_subplots is deprecated, please use plotly.subplots.make_su
bplots instead
```




San Francisco
(not set)
not available in demo dataset
0
200k
400k
Vietnam
United Kingdom
United States
0


visit: Sub Continent wise
Caribbean
Northern Africa
Central America
Western Asia
Eastern Europe
Northern Europe
Southern Asia
Northern America
0
100k
200k
300k
400k
(not set)
Africa
Oceania
Europe
Asia
Americas
0


Mean Revenue by City
Washington
Sunnyvale
San Jose
Quito
Oakland
Mexico City
Kansas City
Fremont
Columbus
Berkeley
(not set)
0
0.5B
1B
Ukraine
Spain
Puerto Rico
Nigeria
Kuwait
Indonesia
Germany
Denmark
Canada
(not set)
0


Mean Revenue by Sub Continent
Western Europe
Western Africa
Southern Asia
Southeast Asia
Northern Europe
Northern Africa
Eastern Asia
Central Asia
Caribbean
(not set)
0
0.5B
1B
1.5B
Oceania
Europe
Asia
Americas
Africa
(not set)
0

6. Visulaizing customers visit and Monthly revenue by date

In [16]:

```
#https://docs.python.org/3/library/datetime.html
#adding datetime column in data
def add_date_features(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"])

    df["month"]   = df['date'].dt.month  #getting month
    df["day"]     = df['date'].dt.day   #getting day
    df["weekday"] = df['date'].dt.weekday   #getting date
    return df
```

In [17]:

```
train_data_conv = add_date_features(train_data_conv)
```

In [18]:

```
test_data_conv = add_date_features(test_data_conv)
```

Visits by date

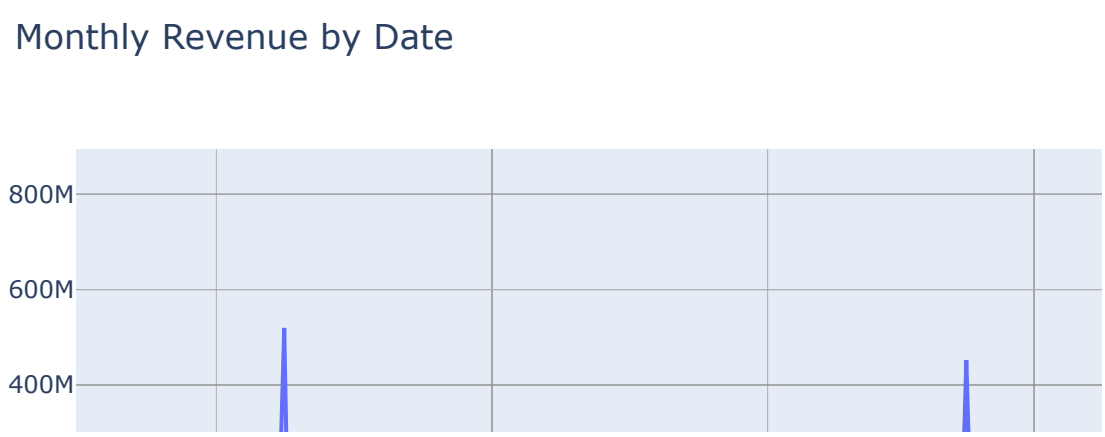
In [19]:

```
# Visualization for Visits by date
#https://plot.ly/python/choropleth-maps/
tmp = train_data_conv['date'].value_counts().to_frame().reset_index().sort_values('index')
tmp = tmp.rename(columns = {"index" : "date", "date" : "visits"})

tr = go.Scatter(mode="lines", x = tmp["date"].astype(str), y = tmp["visits"])
layout = go.Layout(title="Visits by Date", height=400)
fig = go.Figure(data = [tr], layout = layout)
iplot(fig)
```

Monthly revenue by date

In [20]:

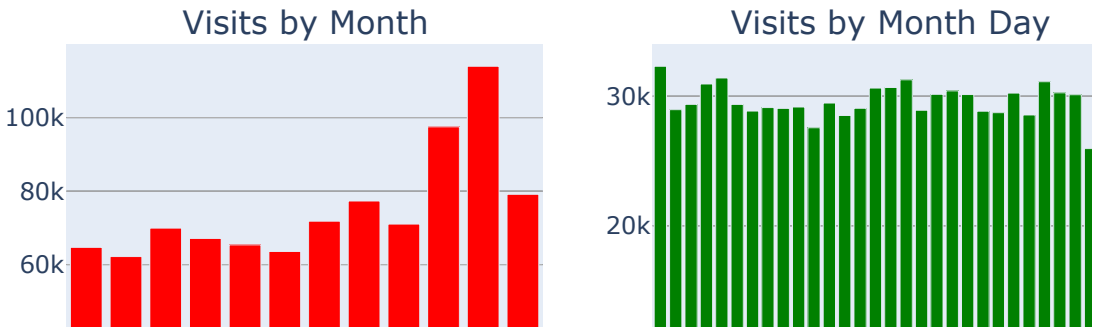
```
# Visualization for Visits by monthly revenue
#https://plot.ly/python/choropleth-maps/
tmp = train_data_conv.groupby("date").agg({"totals_transactionRevenue" : "mean"}).reset_ind
tmp = tmp.rename(columns = {"date" : "dateX", "totals_transactionRevenue" : "mean_revenue"}
tr = go.Scatter(mode="lines", x = tmp["dateX"].astype(str), y = tmp["mean_revenue"])
layout = go.Layout(title="Monthly Revenue by Date", height=400)
fig = go.Figure(data = [tr], layout = layout)
iplot(fig)
```



7. Visualizing visits by Month, Day of Month and Day of week

In [21]:

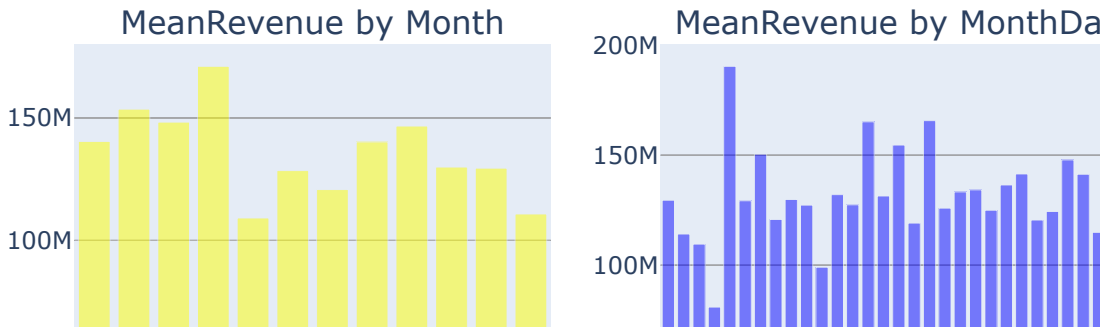
```
#https://plot.ly/python/choropleth-maps/
fig = tools.make_subplots(rows=1, cols=3, subplot_titles=["Visits by Month", "Visits by Mon
trs = []
for i,col in enumerate(["month", "day", "weekday"]):
    t = train_data_conv[col].value_counts()
    tr = go.Bar(x = t.index, marker=dict(color=colors[i]), y = t.values)
    trs.append(tr)

fig.append_trace(trs[0], 1, 1)
fig.append_trace(trs[1], 1, 2)
fig.append_trace(trs[2], 1, 3)
fig['layout'].update(height=400, showlegend=False)
iplot(fig)
```



8. Visulaizing Revenue by Month, Day of month and Day of week

In [22]:

```
tmp1 = train_data_conv.groupby('month').agg({"totals_transactionRevenue" : "mean"}).reset_i
tmp2 = train_data_conv.groupby('day').agg({"totals_transactionRevenue" : "mean"}).reset_ind
tmp3 = train_data_conv.groupby('weekday').agg({"totals_transactionRevenue" : "mean"}).reset

fig = tools.make_subplots(rows=1, cols=3, subplot_titles=["MeanRevenue by Month", "MeanReve
tr1 = go.Bar(x = tmp1.month, marker=dict(color="yellow", opacity=0.5), y = tmp1.totals_tran
tr2 = go.Bar(x = tmp2.day, marker=dict(color="blue", opacity=0.5), y = tmp2.totals_transact
tr3 = go.Bar(x = tmp3.weekday, marker=dict(color="violet", opacity=0.5), y = tmp3.totals_tr

fig.append_trace(tr1, 1, 1)
fig.append_trace(tr2, 1, 2)
fig.append_trace(tr3, 1, 3)
fig['layout'].update(height=400, showlegend=False)
iplot(fig)
```



9. Visualizing visitors profile

In [23]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

agg_dict = {}
for col in ["totals_bounces", "totals_hits", "totals_newVisits", "totals_pageviews", "total
    train_data_conv[col] = train_data_conv[col].astype('float')
    agg_dict[col] = "sum"
tmp = train_data_conv.groupby("fullVisitorId").agg(agg_dict).reset_index()
tmp.head()
```

Out[23]:

	fullVisitorId	totals_bounces	totals_hits	totals_newVisits	totals_pageviews	totals_t
0	0000010278554503158	0.0	11.0	1.0	8.0	
1	0000020424342248747	0.0	17.0	1.0	13.0	
2	0000027376579751715	0.0	6.0	1.0	5.0	
3	0000039460501403861	0.0	2.0	1.0	2.0	
4	0000040862739425590	0.0	5.0	1.0	5.0	

In [24]:

```
non_zero = tmp[tmp["totals_transactionRevenue"] > 0]["totals_transactionRevenue"]
print ("There are " + str(len(non_zero)) + " visitors in the train dataset having non zero
```

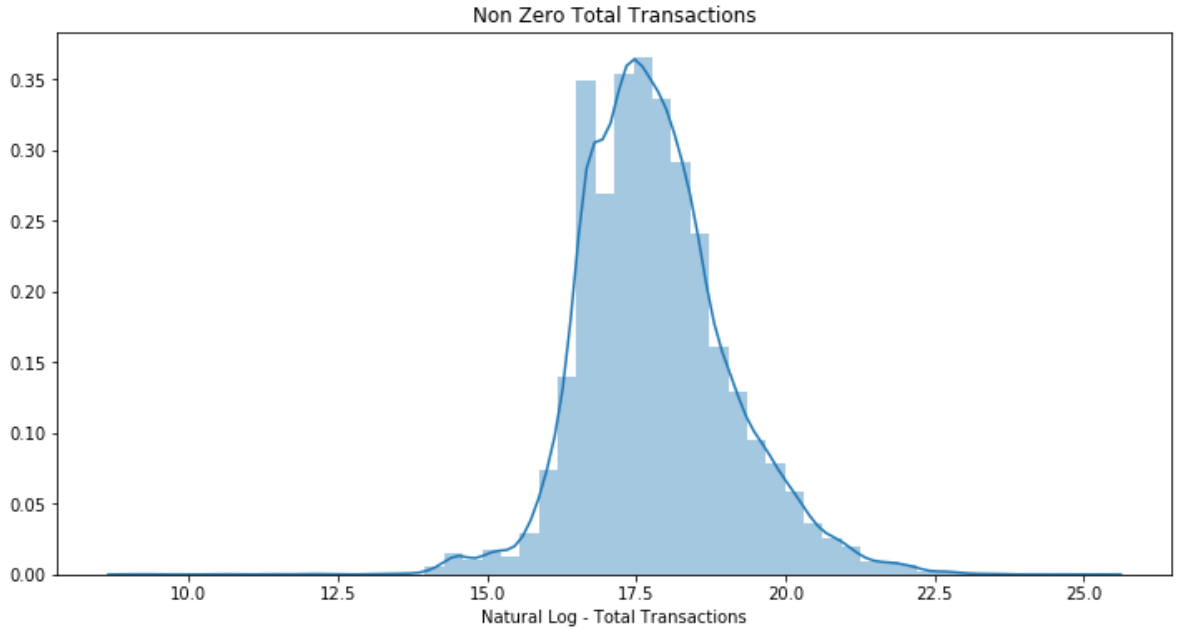
```
There are 9996 visitors in the train dataset having non zero total transacti
on revenue
```

In [25]:

```
#converting the values to log-values to chec for log-normal
import math
log_non_zero=[math.log(i) for i in non_zero]
```

In [26]:

```
plt.figure(figsize=(12,6))
sns.distplot((log_non_zero))
plt.title("Non Zero Total Transactions");
plt.xlabel("Natural Log - Total Transactions");
```



In [27]:

```
### 10. Number of customers common on both train and test data
print("Number of unique visitors in train set : ",train_data_conv.fullVisitorId.nunique(),
print("Number of unique visitors in test set : ",test_data_conv.fullVisitorId.nunique(), "
print("Number of common visitors in train and test set : ",len(set(train_data_conv.fullVisi
```

```
Number of unique visitors in train set :  714167  out of rows :  903653
Number of unique visitors in test set :  617242  out of rows :  804684
Number of common visitors in train and test set :  7679
```

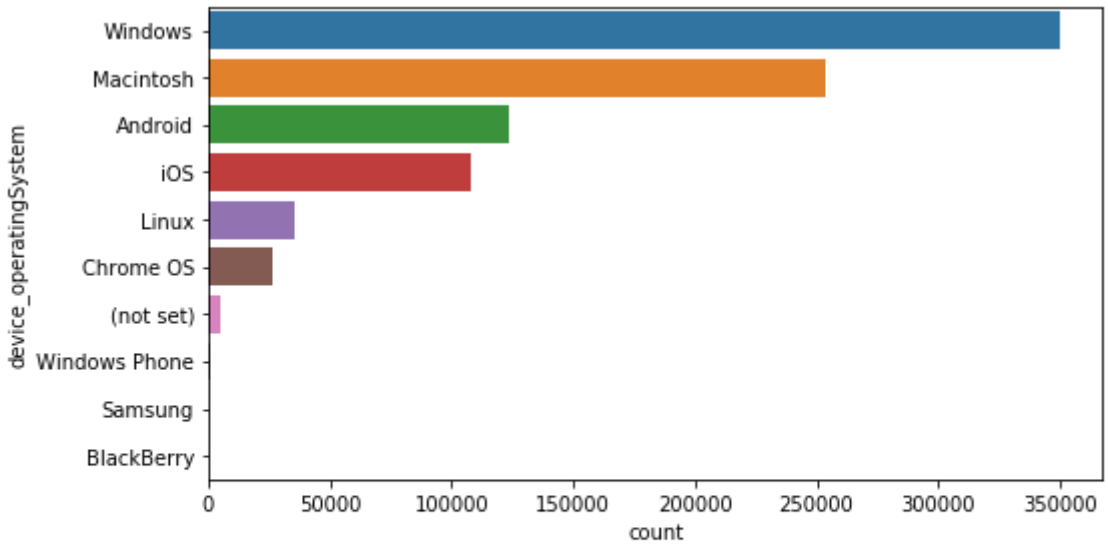
11. Visualizing device

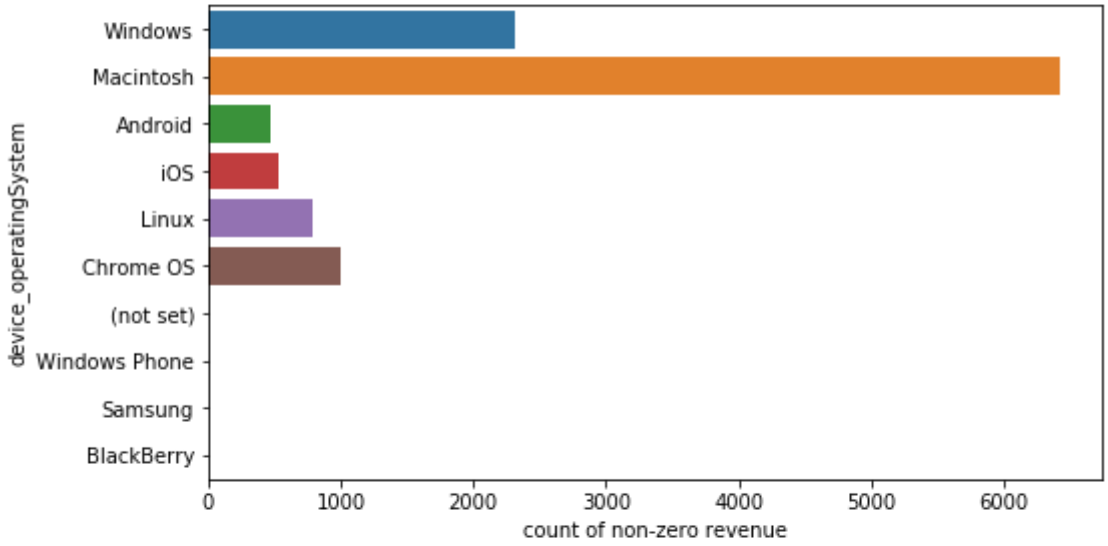
In [28]:

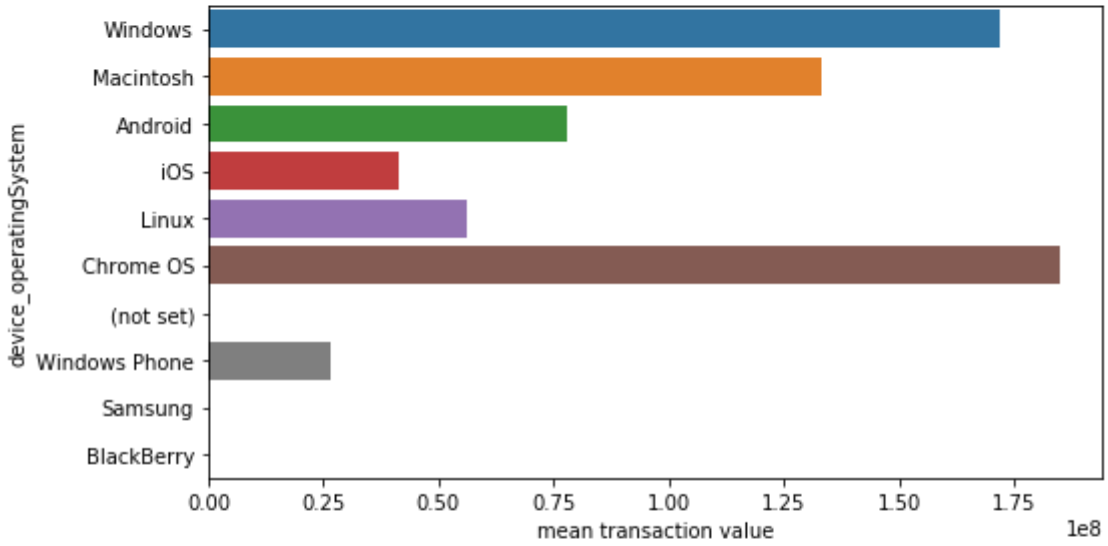
```
def category_plots(col):
    a = train_data_conv.loc[:,[col, 'totals_transactionRevenue']]
    a['totals_transactionRevenue'] = a['totals_transactionRevenue'].replace(0.0, np.nan)

    cnt_srs = a.groupby(col)['totals_transactionRevenue'].agg(['size','count','mean'])
    cnt_srs.columns = ["count", 'count of non-zero revenue', "mean transaction value"]

    cnt_srs['total_revenue'] = cnt_srs['count of non-zero revenue']*cnt_srs['mean transacti
    cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
    plt.figure(figsize=(8, 20))
    plt.subplot(4,1,1)
    sns.barplot(x=cnt_srs['count'].head(10), y=cnt_srs.index[:10])
    plt.subplot(4,1,2)
    sns.barplot(x=cnt_srs['count of non-zero revenue'].head(10), y=cnt_srs.index[:10])
    plt.subplot(4,1,3)
    sns.barplot(x=cnt_srs['mean transaction value'].head(10), y=cnt_srs.index[:10])
    plt.subplot(4,1,4)
    sns.barplot(x=cnt_srs['total_revenue'].head(10), y=cnt_srs.index[:10])

#cnt_srs
```

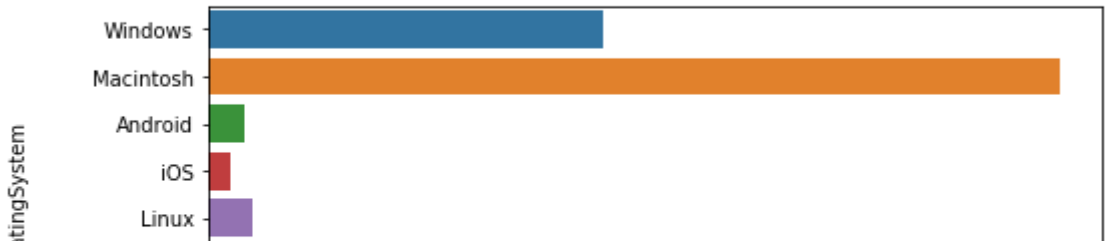
In [29]:

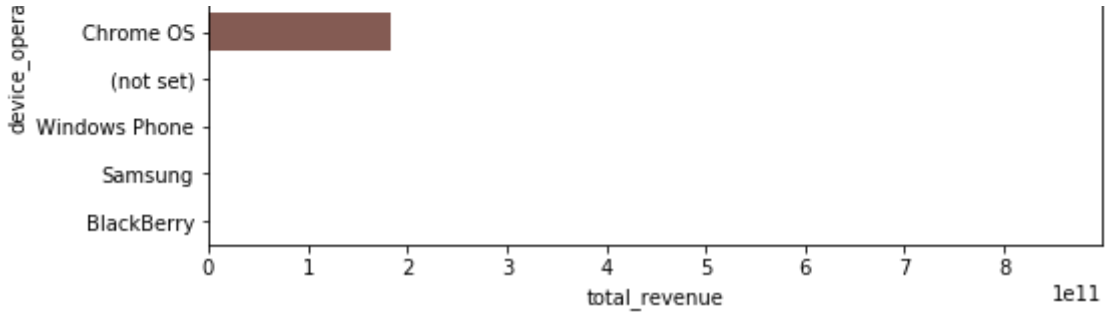
```
category_plots('device_operatingSystem')
```







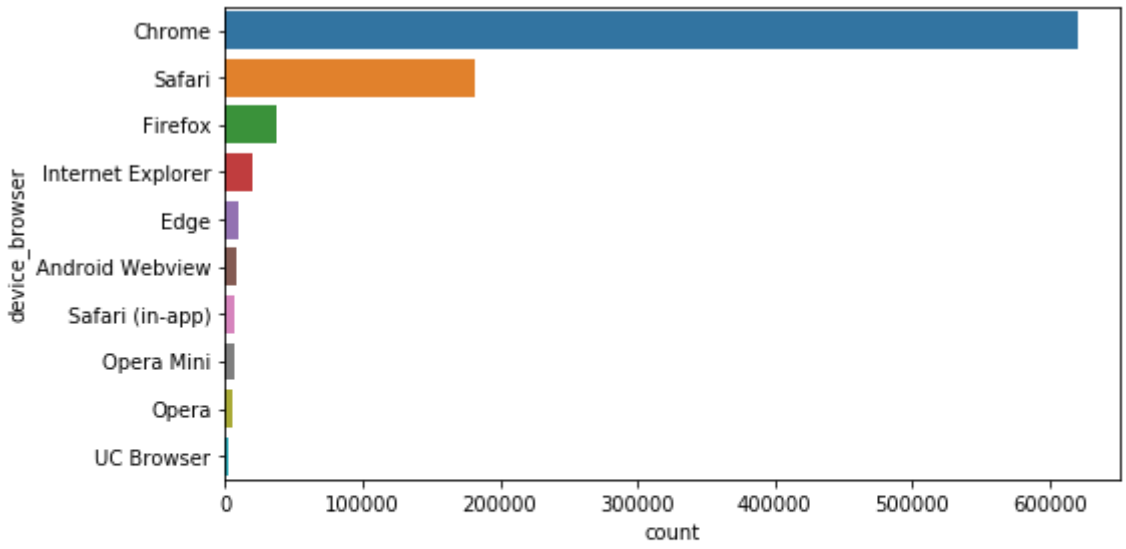


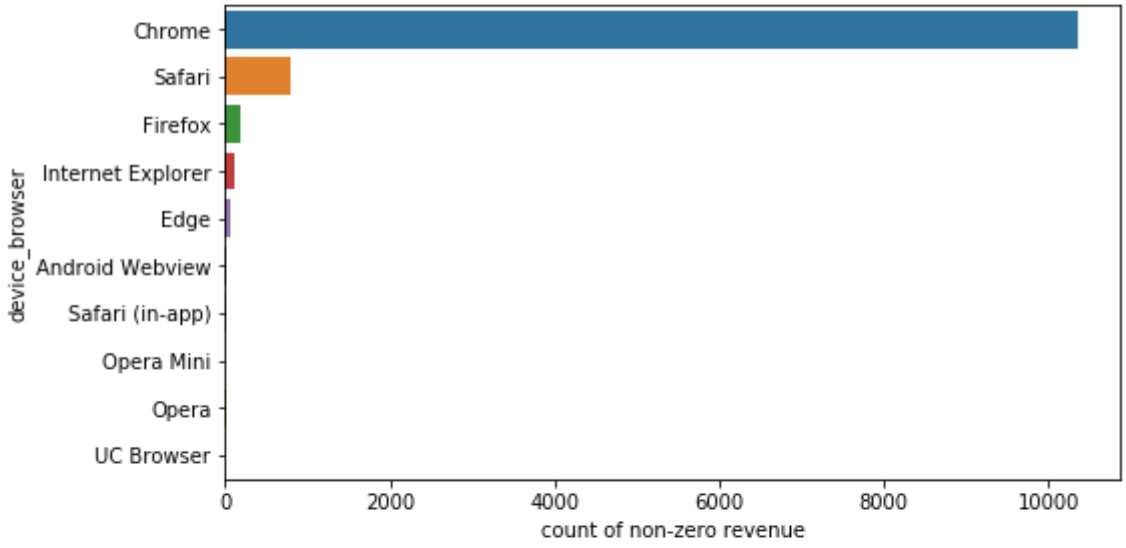


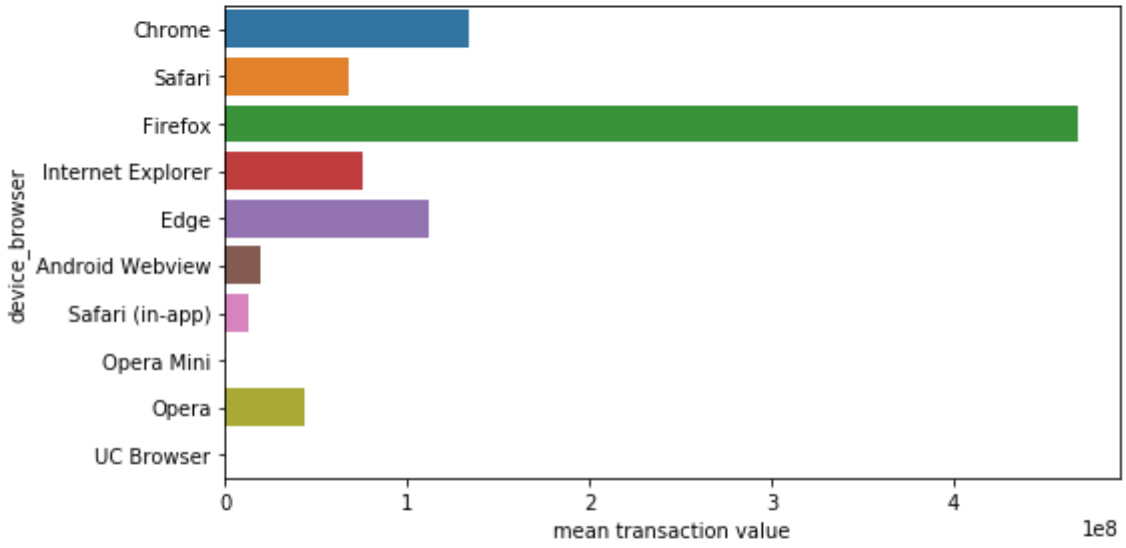
Visulaizing device browser

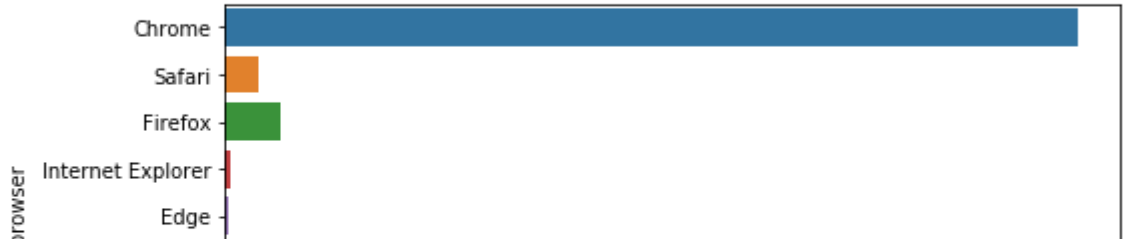
In [30]:

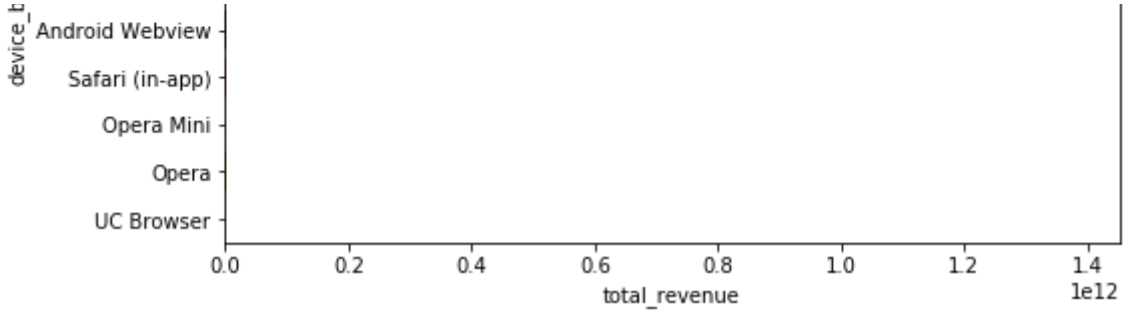
```
category_plots('device_browser')
```









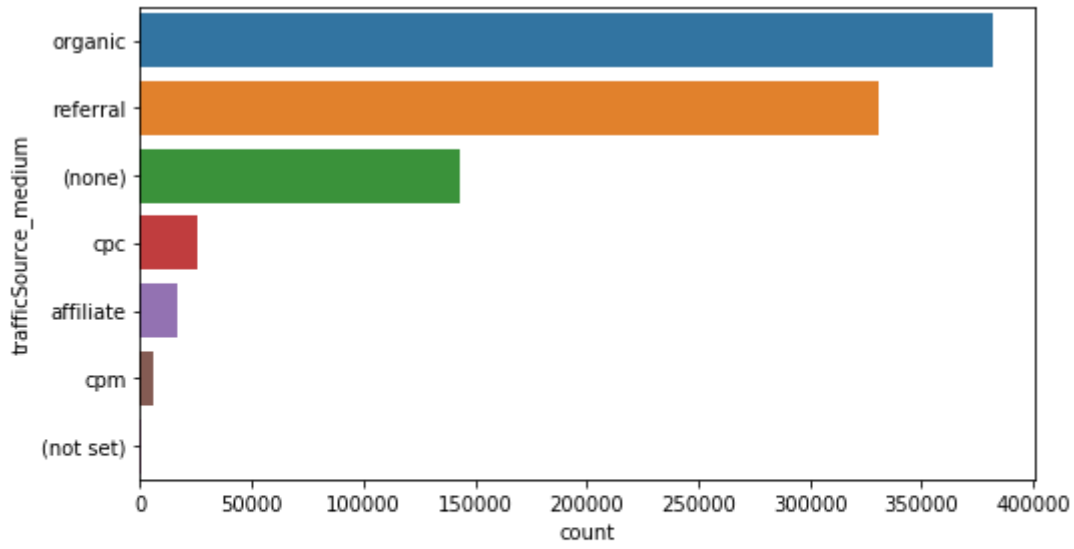


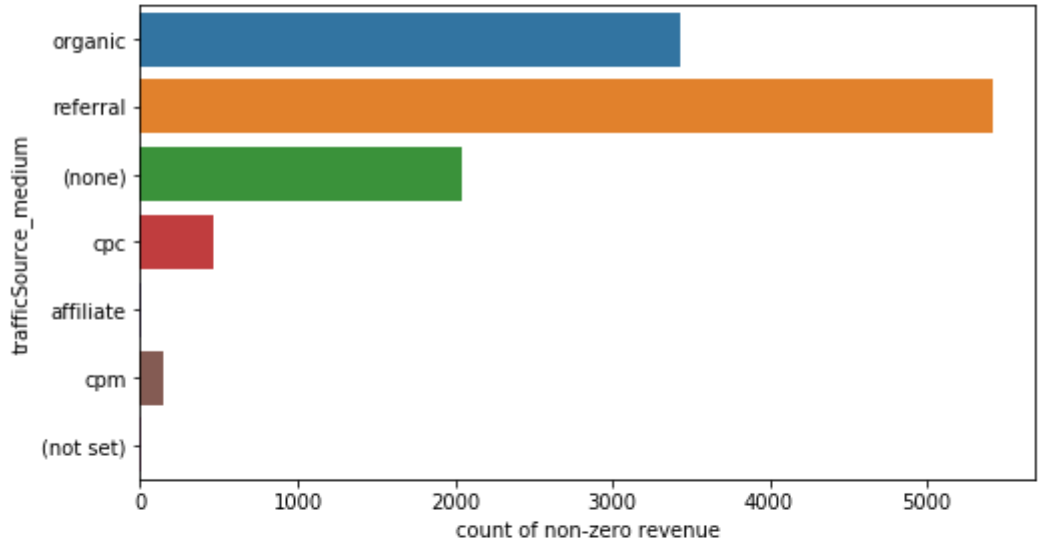
12. Visualizing Traffic source

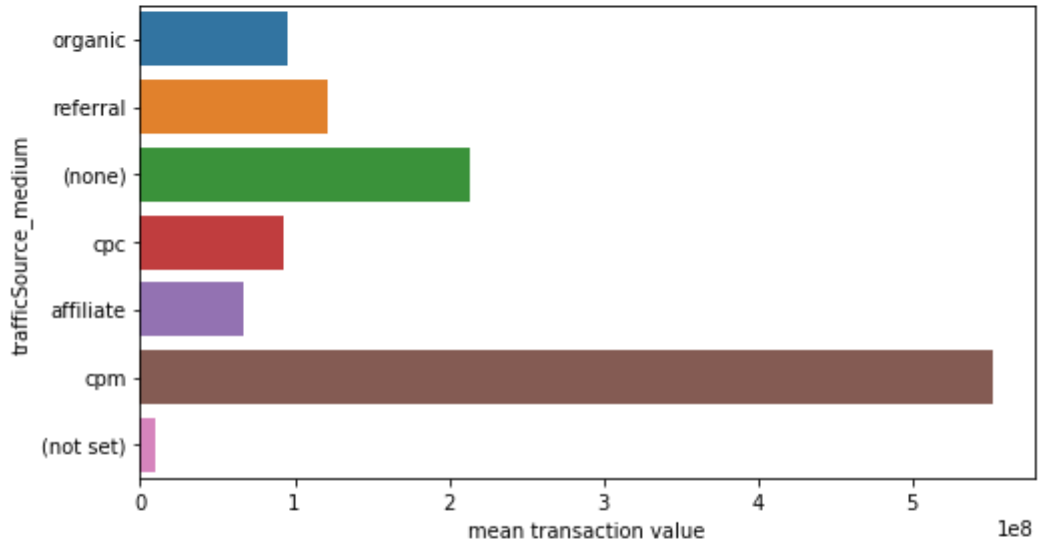
Traffic source medium

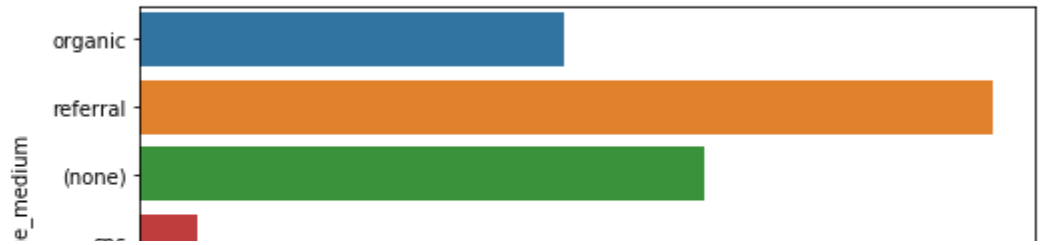
In [31]:

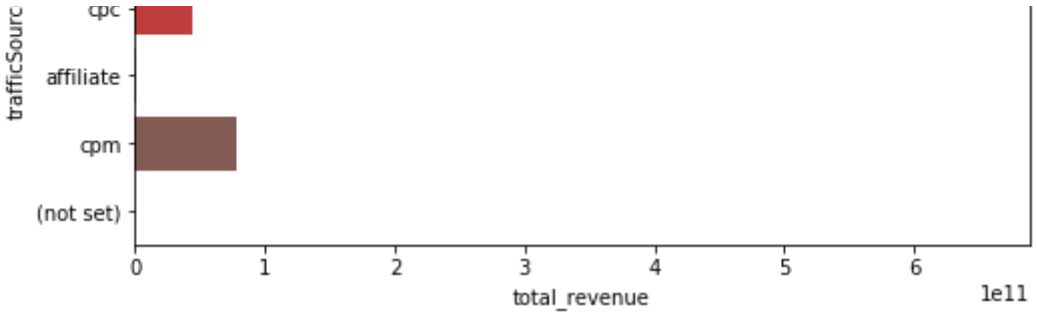
```
category_plots('trafficSource_medium')
```







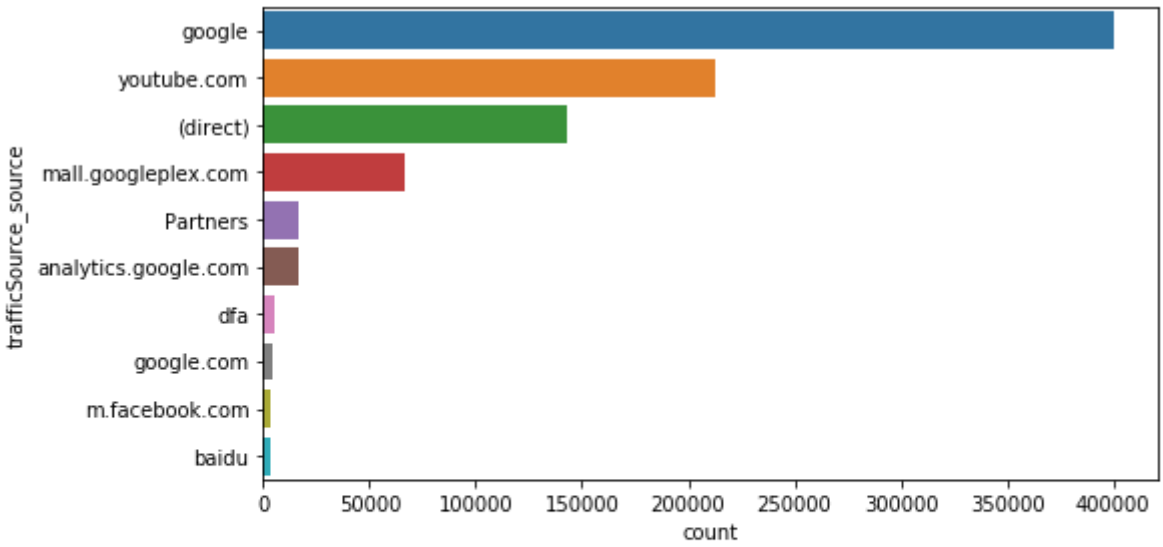


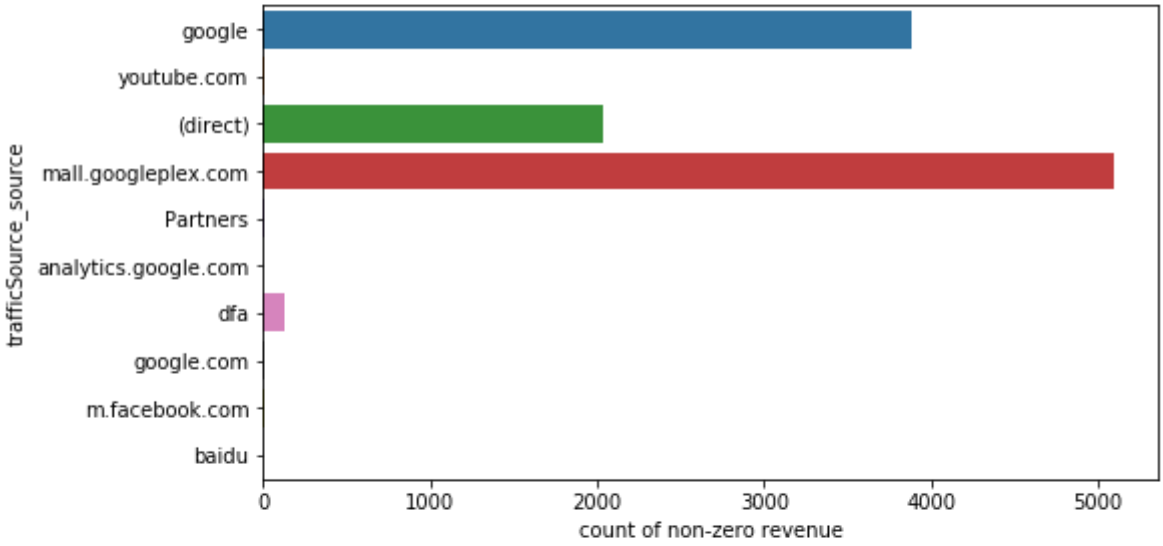


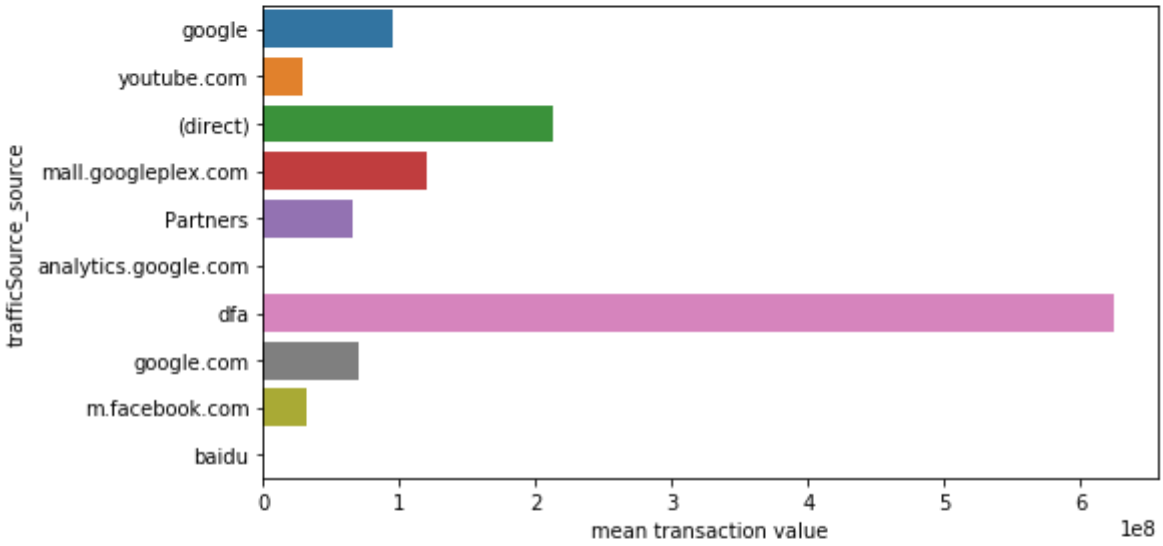
Visualizing Traffic Source

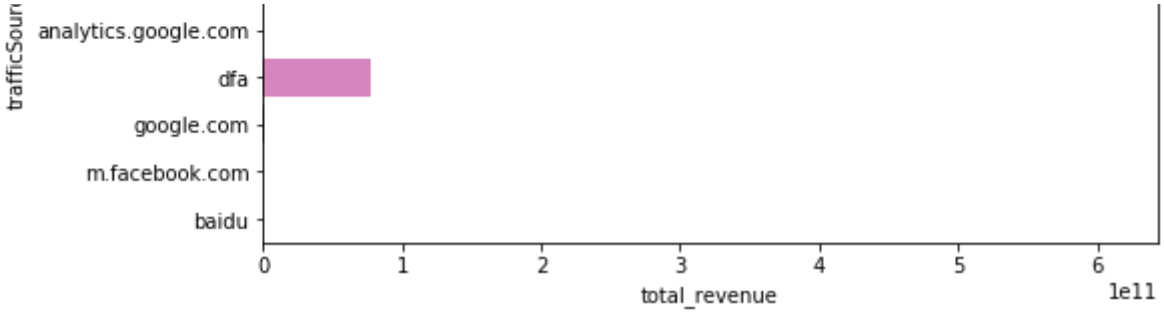
In [32]:

```
category_plots('trafficSource_source')
```





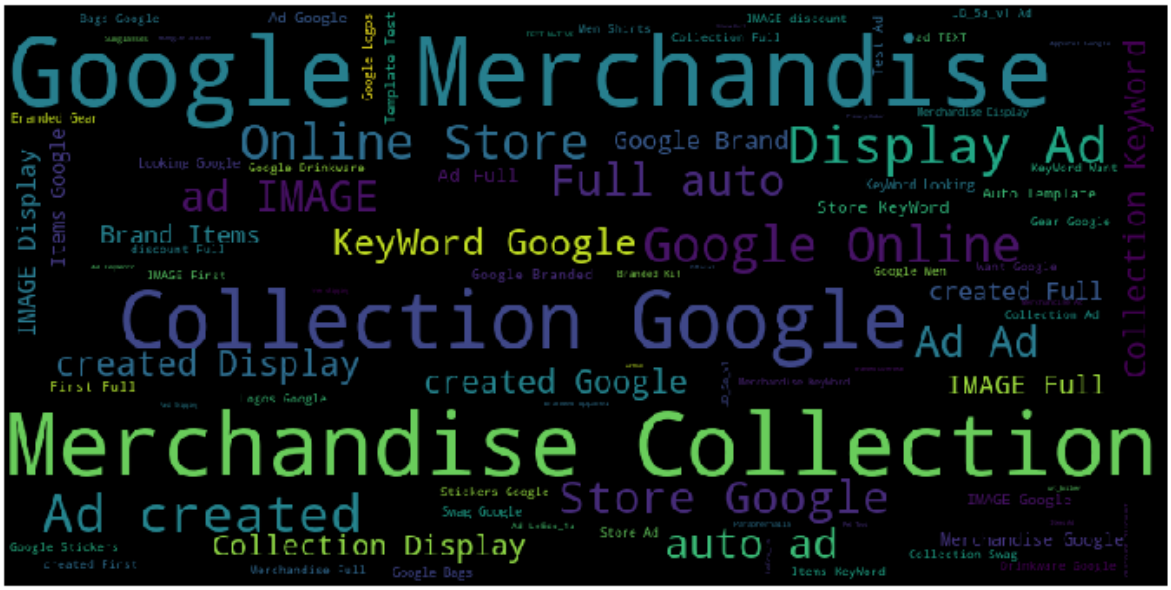




13. Visualizing google AD content

In [33]:

```
ad_content = train_data_conv['trafficSource_adContent'].fillna('')
wordcloud2 = WordCloud(width=800, height=400).generate(' '.join(ad_content))
plt.figure( figsize=(12,9))
plt.imshow(wordcloud2)
plt.axis("off")
plt.show()
```



12. Revenue generating customers vs Non-Revenue generating customers

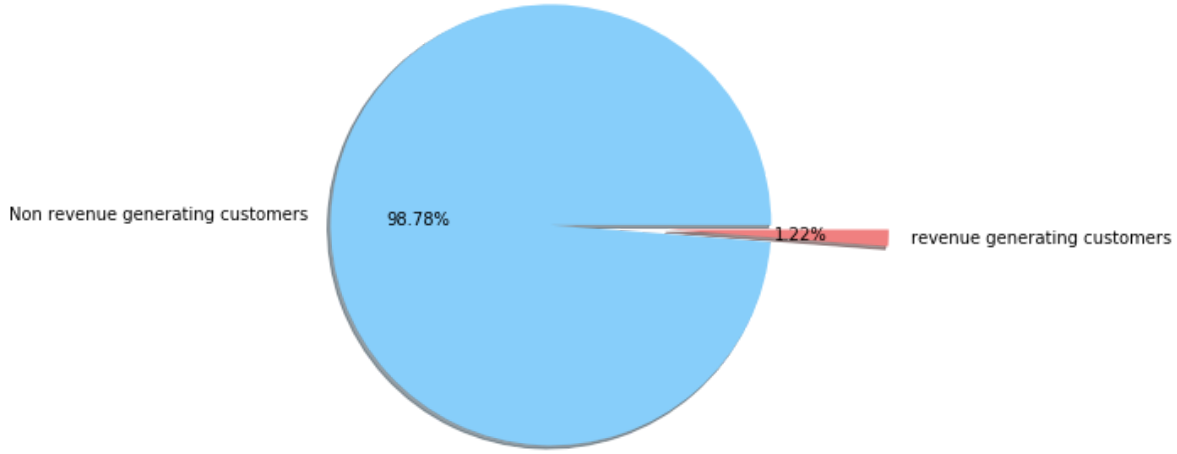
In [34]:

```
gdf = train_data_conv.groupby("fullVisitorId")["totals_transactionRevenue"].sum().reset_ind
nrc = gdf[gdf['totals_transactionRevenue']==0]
rc = gdf[gdf['totals_transactionRevenue']>0]
print("The number of nonrevenue customers are ", len(nrc))
print("The number of revenue generating customers are ", len(rc))
print("The ratio of revenue generating customers is {0:0.4}%".format(len(rc)/len(gdf)*100))
```

```
The number of nonrevenue customers are  704171
The number of revenue generating customers are  9996
The ratio of revenue generating customers is 1.4%
```

In [35]:

```
labels = ['Non revenue generating customers','revenue generating customers']
values = [1307589,16141]

plt.axis("equal")
plt.pie(values, labels=labels, radius=1.5, autopct="%0.2f%%",shadow=True, explode=[0,0.8],
plt.show()
```



Conclusion from the above data analysis

- Most of the Gstore customers are from US, India and Canada
- Customer visits are more from US (Country), American (Continent), Northern America (SubContiinent)
- Number of Visitors to America is nearly 39times greater than Africa.But, Africa is the one from which most of transactionRevenue is generated followed by Asian continent.
- Anguilla is the country from where highest revenue is generated.(Which is in Africa region) followed by Curaco.
- Visits are more during November and May
- Monthly revenue generated are more from March - April - May
- There are 9996 visitors in the train dataset having non zero total transaction revenue
- Number of common visitors in train and test set : 7679
- The ratio of revenue generating customers is 1.4%
- There are more customers who access gstore having WindowsOS
- Most of the revenue generated by MacOS users.
- Most of the visits done by Chrome User
- FireFox browser is the one from which most of transactionRevenue is generated. This might be possible because all of them who visit the website doesn't mean they will purchase.We can assume as, the one who used firefox browser had made purchases.
- Desktop is the one which is used by most of the visitors. Most of transaction is generated by it when compared to other devices.
- Most of the visitors trafficSource came through campaign 'not set', this might be possible as there will be many unknown sources
- But through 'cpm' medium , highest transactionRevenue is generated.
- Most of the revenue generated by the user who uses Chome browser
- The best traffic medium is "Referal"

- Mall.googleplex.com is the best traffic source by which most revenue is generated
- Google, Youtube and direct are the source of more visitors
- Google ad mainly contain keyword - "Google Merchandise Collection"

Data Preprocessing

Removie the columns that are having

1. ID
2. NA
3. Data and visitstarttime

Dropping coluns in train dataset

In [36]:

```
## find constant columns
constant_columns = [column for column in train_data_conv.columns if train_data_conv[column]

#dropping constant columns
train_data_conv_dropped = train_data_conv.drop(columns=constant_columns)

#Sorting by date to perform time based slicing
sorted_train_data_conv_dropped = train_data_conv_dropped.sort_values(by='date',ascending=Tr

## non relevant columns
non_relevant = ["visitNumber", "date", "fullVisitorId", "sessionId", "visitId", "visitStart

## Drpiing non relevant columns
train = sorted_train_data_conv_dropped.drop(columns=non_relevant)

print("Before dropping the columns: ", train_data_conv.shape)
print("After dropping the columns: ", train.shape)
```

```
Before dropping the columns:  (903653, 58)
After dropping the columns:  (903653, 33)
```

Dropiing columns in test dataset

In [37]:

```
## find constant columns
constant_columns = [column for column in test_data_conv.columns if test_data_conv[column].n

#dropping constant columns
test_data_conv_dropped = test_data_conv.drop(columns=constant_columns)

#Sorting by date to perform time based slicing
sorted_test_data_conv_dropped = test_data_conv_dropped.sort_values(by='date',ascending=True

## non relevant columns
non_relevant = ["visitNumber", "date", "fullVisitorId", "sessionId", "visitId", "visitStart

## Drpiing non relevant columns
test = sorted_test_data_conv_dropped.drop(columns=non_relevant)

print("Before dropping the columns: ", test_data_conv.shape)
print("After dropping the columns: ", test.shape)
```

```
Before dropping the columns:  (804684, 56)
After dropping the columns:  (804684, 31)
```

Getting attributes which are in train data but not in test dataset

In [38]:

```
print("Column in train but not in test : ", set(train_data.columns).difference(set(test.col
```

```
Column in train but not in test :  {'visitStartTime', 'visitId', 'sessionI
d', 'fullVisitorId', 'trafficSource', 'geoNetwork', 'date', 'visitNumber',
'device', 'socialEngagementType', 'totals'}
```

In [39]:

```
# removing above columns in train data
train = train.drop(columns='trafficSource_campaignCode')
```

Encoding Categorical values

In [59]:

```
categorical_features_train = (train.select_dtypes(include=[np.object]))
print("In Train data",categorical_features_train.columns)
```

```
In Train data Index([], dtype='object')
```

In [41]:

```
categorical_columns = [column for column in train.columns if not column.startswith('total')
categorical_columns = [column for column in categorical_features_train if column not in con

for column in categorical_columns:

    le = LabelEncoder()
    train_values = list(train[column].values.astype(str))
    test_values = list(test[column].values.astype(str))

    le.fit(train_values + test_values)

    train[column] = le.transform(train_values)
    test[column] = le.transform(test_values)
```

In [42]:

```
#this is boolean attribute
train['device_isMobile'] = le.transform(train_values)
test['device_isMobile'] = le.transform(test_values)
```

Numerical features: Filling NA with 0.0 and normalizing the numerical data

In [43]:

```
#filling the NA values in totals column
def normalize_numerical_columns(dataframe, isTrainDataset = True):
    dataframe["totals_hits"] = dataframe["totals_hits"].astype(float)
    #dataframe["totals.hits"] = (dataframe["totals.hits"] - min(dataframe["totals.hits"]))

    dataframe["totals_pageviews"] = dataframe["totals_pageviews"].astype(float)
    #dataframe["totals.pageviews"] = (dataframe["totals.pageviews"] - min(dataframe["totals

    dataframe["totals_bounces"] = dataframe["totals_bounces"].astype(float)
    dataframe["totals_newVisits"] = dataframe["totals_newVisits"].astype(float)


    if isTrainDataset:
        dataframe["totals_transactionRevenue"] = dataframe["totals_transactionRevenue"].fil
    return dataframe


train = normalize_numerical_columns(train)
test = normalize_numerical_columns(test,isTrainDataset=False)
```

Preprocessed train and test dataset

In [57]:

```
train.to_csv("Preprocessed_train.csv",index=False)
train.head()
```

Out[57]:

channelGrouping	device_browser	device_operatingSystem	device_isMobile	device_deviceCateg
2	35	7	62	
2	35	20	62	
6	35	7	62	
2	35	20	62	
2	72	7	62	

5 rows × 31 columns

In [58]:

```
#test.to_csv("Preprocessed_test.csv",index=False)
test_data = pd.read_csv("Preprocessed_test.csv")
test_data.head()
```

Out[58]:

	channelGrouping	device_browser	device_operatingSystem	device_isMobile	device_deviceCat
0	4	72	23	62	
1	7	72	7	62	
2	7	35	20	62	
3	7	35	20	62	
4	7	35	20	62	

5 rows × 31 columns

Since the train data is sorted date wise hence Performing time based slicing to the train data in 80-20 ratio

In [2]:

```
train = pd.read_csv('Preprocessed_train.csv', index_col=0)
X_test = pd.read_csv('Preprocessed_test.csv', index_col=0)
```

In [3]:

```
temp = round((train.shape[0])*0.8)
X_train = train[:temp]
X_cv = train[temp+1:]

y_train = np.log1p(X_train['totals_transactionRevenue'])
X_train = X_train.drop(['totals_transactionRevenue'],axis=1)
y_cv = np.log1p(X_cv['totals_transactionRevenue'])
X_cv = X_cv.drop(['totals_transactionRevenue'],axis=1)


print("After splitting:")
print("Train: ",X_train.shape,y_train.shape)
print("CV: ",X_cv.shape,y_cv.shape)
print("Test: ",X_test.shape)
```

```
After splitting:
Train:  (722922, 29) (722922,)
CV:  (180730, 29) (180730,)
Test:  (804684, 30)
```

In [4]:

```
X_train = X_train.fillna(0).astype('float32')
X_cv = X_cv.fillna(0).astype('float32')
X_test = X_test.fillna(0).astype('float32')
```

For Visulaizing feature imprtance

In [5]:

```
#https://chrisalbon.com/machine_learning/trees_and_forests/feature_importance/
def plot_feat_imp(model):
    importances = model.feature_importances_

    indices = np.argsort(importances)[::-1]

    # Rearrange feature names so they match the sorted feature importances
    names = [X_train.columns[i] for i in indices]

    # Create plot
    plt.figure()

    # Create plot title
    plt.title("Feature Importance")

    # Add bars
    plt.bar(range(30), importances[indices])

    # Add feature names as x-axis labels
    plt.xticks(range(30),names, rotation=90)

    # Show plot
    plt.show()
```

Machine learning models

Light GBM

In [39]:

```
import lightgbm as lgb
from bayes_opt import BayesianOptimization
```

In [69]:

```
import lightgbm as lgb

lgb_params = {"objective" : "regression", "metric" : "rmse",
              "num_leaves" : 50, "learning_rate" : 0.02,
              "bagging_fraction" : 0.75, "feature_fraction" : 0.8, "bagging_frequency" : 9}

lgb_train = lgb.Dataset(X_train, label=y_train)
lgb_val = lgb.Dataset(X_cv, label=y_cv)
model = lgb.train(lgb_params, lgb_train, 700, valid_sets=[lgb_val], early_stopping_rounds=1
```

```
Training until validation scores don't improve for 150 rounds
[20]    valid_0's rmse: 1.95704
[40]    valid_0's rmse: 1.86112
[60]    valid_0's rmse: 1.80591
[80]    valid_0's rmse: 1.77429
[100]   valid_0's rmse: 1.75523
[120]   valid_0's rmse: 1.74355
[140]   valid_0's rmse: 1.7358
[160]   valid_0's rmse: 1.73037
[180]   valid_0's rmse: 1.72668
[200]   valid_0's rmse: 1.72505
[220]   valid_0's rmse: 1.72369
[240]   valid_0's rmse: 1.72294
[260]   valid_0's rmse: 1.72221
[280]   valid_0's rmse: 1.72208
[300]   valid_0's rmse: 1.72173
[320]   valid_0's rmse: 1.72167
[340]   valid_0's rmse: 1.72197
[360]   valid_0's rmse: 1.72209
[380]   valid_0's rmse: 1.72192
[400]   valid_0's rmse: 1.72198
[420]   valid_0's rmse: 1.72222
[440]   valid_0's rmse: 1.72252
Early stopping, best iteration is:
[307]   valid_0's rmse: 1.72167
```

Linear Regression model

In [51]:

```
lr_reg=LinearRegression()

param_grid = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False
grid_clf_acc = GridSearchCV(lr_reg, param_grid = param_grid)
grid_clf_acc.fit(X_train,  y_train)

print(grid_clf_acc.best_estimator_)
print(grid_clf_acc.best_params_)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fal
se)
{'copy_X': True, 'fit_intercept': True, 'normalize': False}
```

In [52]:

```
lr_reg=LinearRegression(copy_X= True, fit_intercept=True, normalize= False)
lr_reg.fit(X_train, y_train)

y_pred = lr_reg.predict(X_train)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = lr_reg.predict(X_cv)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)
```

```
Train Root Mean Squared Error: 1.7830931064852178
Validation Root Mean Squared Error: 1.921973286588651
```

Gradient Boosting model

In [55]:

```
regr = ensemble.GradientBoostingRegressor()

param_grid = {"max_depth": [3,5,7,8],
              "learning_rate": [0.001,0.01,0.1,0.5,1],
              "n_estimators":[20,50,100, 120, 150, 200, 250, 300]
              }

# run randomized search
random_search = RandomizedSearchCV(regr, param_distributions=param_grid)
random_search.fit(X_train, y_train)

print(random_search.best_params_)
```

```
[22:42:52] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[23:05:17] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[23:27:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[23:45:48] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:03:45] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:21:38] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:26:03] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
```

In [56]:

```
print(random_search.cv_results_)
print(random_search.best_params_)
```

```
{'mean_fit_time': array([1175.89219131,  262.81834207,  468.71439581,  24
7.78635502,
        360.56631112,  106.04157023,  369.67409596,  385.81286306,
        582.75471988,  105.76499772]), 'std_fit_time': array([1.15444146e+
02, 7.53371681e-01, 3.31792372e+00, 1.56776635e+00,
       1.66416508e+00, 4.15945400e-01, 3.28929963e+00, 1.21758240e+00,
       3.19339326e+00, 9.51309568e-02]), 'mean_score_time': array([9.35976
987, 1.15156999, 3.8644443 , 1.38140564, 2.34076157,
       0.46022358, 1.60283728, 2.02775812, 6.86246924, 0.39199848]), 'std_
score_time': array([1.10452051, 0.0143809 , 0.07004821, 0.03502578, 0.0287
5194,
       0.01510045, 0.0249983 , 0.01614348, 0.06422487, 0.00608929]), 'para
m_n_estimators': masked_array(data=[700, 500, 300, 200, 300, 200, 700, 50
0, 300, 200],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_max_depth': masked_array(data=[10, 3, 1
0, 8, 8, 3, 3, 5, 12, 3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_learning_rate': masked_array(data=[0.1,
0.1, 1, 0.01, 0.1, 0.1, 0.01, 0.001, 1, 0.01],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'n_estimators': 700, 'max_depth': 1
0, 'learning_rate': 0.1}, {'n_estimators': 500, 'max_depth': 3, 'learning_
rate': 0.1}, {'n_estimators': 300, 'max_depth': 10, 'learning_rate': 1},
{'n_estimators': 200, 'max_depth': 8, 'learning_rate': 0.01}, {'n_estimato
rs': 300, 'max_depth': 8, 'learning_rate': 0.1}, {'n_estimators': 200, 'ma
x_depth': 3, 'learning_rate': 0.1}, {'n_estimators': 700, 'max_depth': 3,
'learning_rate': 0.01}, {'n_estimators': 500, 'max_depth': 5, 'learning_ra
te': 0.001}, {'n_estimators': 300, 'max_depth': 12, 'learning_rate': 1},
{'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.01}], 'split0_tes
t_score': array([ 0.12592107,  0.25314175, -0.4327011 ,  0.25360882,  0.21
278614,
        0.24479884,  0.22280093,  0.17411958, -0.44148072,  0.21592104]),
'split1_test_score': array([ 0.23705499,  0.31416664, -0.31538502,  0.2956
6155,  0.2970651 ,
        0.30569818,  0.28618077,  0.15556589, -0.25904025,  0.25613017]),
'split2_test_score': array([ 0.24313162,  0.34008794, -0.33448929,  0.3111
1121,  0.31052148,
        0.32863338,  0.3066833 ,  0.16054092, -0.27906414,  0.27347838]),
'split3_test_score': array([ 0.19564096,  0.31744838, -0.55241121,  0.3127
6883,  0.28856914,
        0.31069481,  0.30084996,  0.16976087, -0.40428093,  0.26696678]),
'split4_test_score': array([ 0.30165795,  0.33038298, -0.00474581,  0.3105
373 ,  0.33895755,
        0.31948728,  0.29594571,  0.1525025 , -0.0093298 ,  0.25014733]),
'mean_test_score': array([ 0.22068132,  0.31104554, -0.32794648,  0.296737
54,  0.28957988,
        0.3018625 ,  0.28249214,  0.16249795, -0.27863917,  0.25252874]),
'std_test_score': array([0.0581994 , 0.0304006 , 0.18221812, 0.02242892,
0.04202626,
```

```
       0.02958478, 0.03059368, 0.00824142, 0.15179925, 0.02003052]), 'rank
_test_score': array([ 7,  1, 10,  3,  4,  2,  5,  8,  9,  6])}
{'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.1}
```

In [74]:

```
from sklearn import ensemble
reg = ensemble.GradientBoostingRegressor(n_estimators=500, learning_rate=0.1, max_depth=3,
reg.fit(X_train, y_train)
```

```
      Iter       Train Loss   Remaining Time
         1           3.7104           9.54m
         2           3.5682           9.47m
         3           3.4466           9.48m
         4           3.3470           9.46m
         5           3.2651           9.44m
         6           3.1962           9.44m
         7           3.1385           9.40m
         8           3.0907           9.36m
         9           3.0500           9.34m
        10           3.0163           9.33m
        20           2.8580           9.12m
        30           2.8038           8.93m
        40           2.7710           8.76m
        50           2.7465           8.56m
        60           2.7215           8.38m
        70           2.7033           8.21m
        80           2.6852           8.01m
        90           2.6728           7.82m
       100           2.6624           7.63m
       200           2.5909           5.71m
       300           2.5544           3.80m
       400           2.5293           1.90m
       500           2.5084           0.00s
```

Out[74]:

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_ms
e',
                          init=None, learning_rate=0.1, loss='ls', max_depth
=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=Non
e,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=500,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=42, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=1, warm_start=Fal
se)
```

RandomForest Model

In [53]:

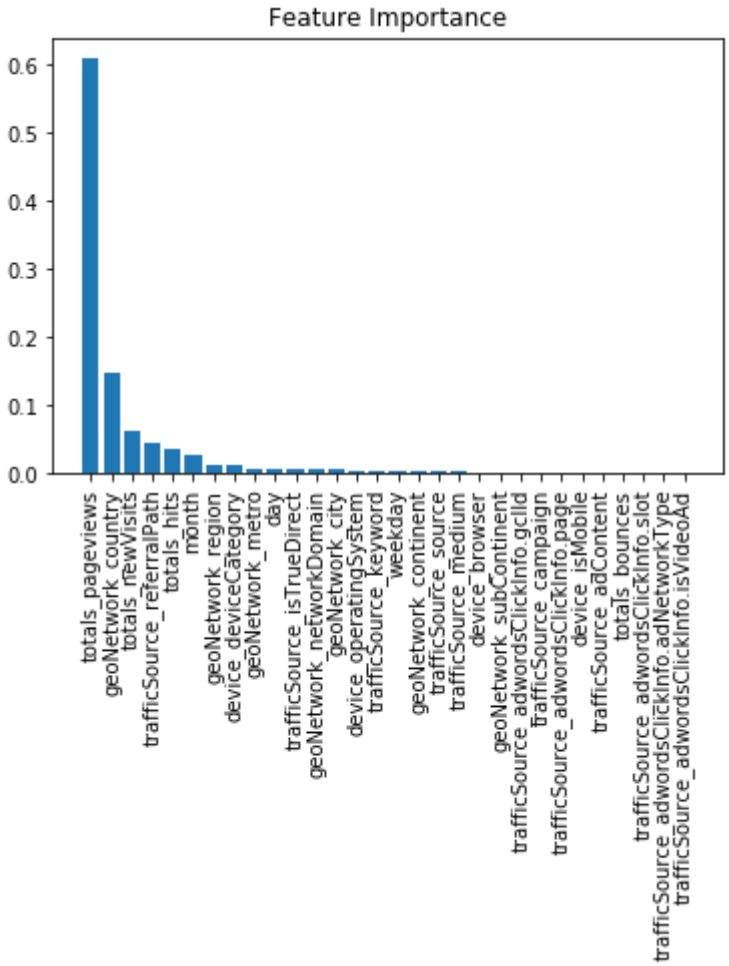
```
regr = RandomForestRegressor()

param_grid = {"max_depth": [3,5,7,8],
              "min_samples_split": sp_randint(2, 10),
              "min_samples_leaf": sp_randint(1, 10),
              "n_estimators":[20,50,100, 120, 150, 200, 250, 300]
              }

# run randomized search
random_search = RandomizedSearchCV(regr, param_distributions=param_grid)
random_search.fit(X_train, y_train)

print(random_search.best_params_)
```

```
{'max_depth': 8, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimator
s': 250}
```

In [54]:

```
regr = RandomForestRegressor(max_depth=8, min_samples_leaf=4, min_samples_split= 5, n_estim
regr.fit(X_train, y_train)

y_pred = regr.predict(X_train)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = regr.predict(X_cv)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)

plot_feat_imp(regr)
```

```
Train Root Mean Squared Error: 1.6116429405186015
Validation Root Mean Squared Error: 1.7362785104593284
```



XGBoost Model

In [55]:

```
x_model = xgb.XGBRegressor()
param_dist = {"max_depth": [3,5,8,10,12],
              "n_estimators" : [200, 300, 500, 700],
              "learning_rate":[0.001,0.01,0.1,1]
              }

random_search = RandomizedSearchCV(x_model, param_distributions=param_dist)
random_search.fit(X_train, y_train)
```

```
[00:39:15] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:43:38] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:51:34] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:59:31] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[01:07:22] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[01:15:11] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[01:23:01] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
```

In [56]:

```
print(random_search.cv_results_)
print(random_search.best_params_)
```

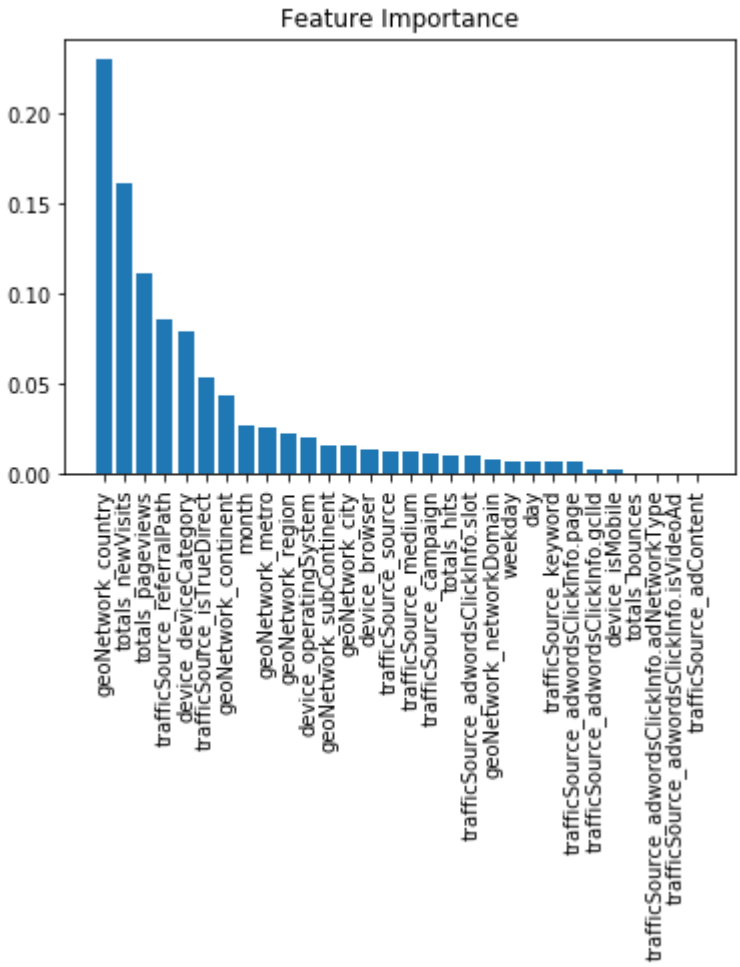
```
{'mean_fit_time': array([1175.89219131,  262.81834207,  468.71439581,  247.7
8635502,
        360.56631112,  106.04157023,  369.67409596,  385.81286306,
        582.75471988,  105.76499772]), 'std_fit_time': array([1.15444146e+0
2, 7.53371681e-01, 3.31792372e+00, 1.56776635e+00,
       1.66416508e+00, 4.15945400e-01, 3.28929963e+00, 1.21758240e+00,
       3.19339326e+00, 9.51309568e-02]), 'mean_score_time': array([9.3597698
7, 1.15156999, 3.8644443 , 1.38140564, 2.34076157,
       0.46022358, 1.60283728, 2.02775812, 6.86246924, 0.39199848]), 'std_sc
ore_time': array([1.10452051, 0.0143809 , 0.07004821, 0.03502578, 0.0287519
4,
       0.01510045, 0.0249983 , 0.01614348, 0.06422487, 0.00608929]), 'param_
n_estimators': masked_array(data=[700, 500, 300, 200, 300, 200, 700, 500, 30
0, 200],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_max_depth': masked_array(data=[10, 3, 10,
8, 8, 3, 3, 5, 12, 3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_learning_rate': masked_array(data=[0.1, 0.
1, 1, 0.01, 0.1, 0.1, 0.01, 0.001, 1, 0.01],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'n_estimators': 700, 'max_depth': 10,
'learning_rate': 0.1}, {'n_estimators': 500, 'max_depth': 3, 'learning_rat
e': 0.1}, {'n_estimators': 300, 'max_depth': 10, 'learning_rate': 1}, {'n_es
timators': 200, 'max_depth': 8, 'learning_rate': 0.01}, {'n_estimators': 30
0, 'max_depth': 8, 'learning_rate': 0.1}, {'n_estimators': 200, 'max_depth':
3, 'learning_rate': 0.1}, {'n_estimators': 700, 'max_depth': 3, 'learning_ra
te': 0.01}, {'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.001},
{'n_estimators': 300, 'max_depth': 12, 'learning_rate': 1}, {'n_estimators':
200, 'max_depth': 3, 'learning_rate': 0.01}], 'split0_test_score': array([
0.12592107,  0.25314175, -0.4327011 ,  0.25360882,  0.21278614,
        0.24479884,  0.22280093,  0.17411958, -0.44148072,  0.21592104]), 's
plit1_test_score': array([ 0.23705499,  0.31416664, -0.31538502,  0.2956615
5,  0.2970651 ,
        0.30569818,  0.28618077,  0.15556589, -0.25904025,  0.25613017]), 's
plit2_test_score': array([ 0.24313162,  0.34008794, -0.33448929,  0.3111112
1,  0.31052148,
        0.32863338,  0.3066833 ,  0.16054092, -0.27906414,  0.27347838]), 's
plit3_test_score': array([ 0.19564096,  0.31744838, -0.55241121,  0.3127688
3,  0.28856914,
        0.31069481,  0.30084996,  0.16976087, -0.40428093,  0.26696678]), 's
plit4_test_score': array([ 0.30165795,  0.33038298, -0.00474581,  0.3105373
,  0.33895755,
        0.31948728,  0.29594571,  0.1525025 , -0.0093298 ,  0.25014733]), 'm
ean_test_score': array([ 0.22068132,  0.31104554, -0.32794648,  0.29673754,
0.28957988,
        0.3018625 ,  0.28249214,  0.16249795, -0.27863917,  0.25252874]), 's
td_test_score': array([0.0581994 , 0.0304006 , 0.18221812, 0.02242892, 0.042
02626,
       0.02958478, 0.03059368, 0.00824142, 0.15179925, 0.02003052]), 'rank_t
```

```
est_score': array([ 7,  1, 10,  3,  4,  2,  5,  8,  9,  6])}
{'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.1}
```

In [58]:

```
x_model = xgb.XGBRegressor(learning_rate =0.1,n_estimators=500,max_depth=3)

x_model.fit(X_train, y_train)

y_pred = x_model.predict(X_train)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = x_model.predict(X_cv)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)

plot_feat_imp(x_model)
```

```
[08:28:57] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/obje
ctive/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:sq
uarederror.
Train Root Mean Squared Error: 1.585291931379857
Validation Root Mean Squared Error: 1.7314629332886489
```



Feature engineering

1. Mean hits per networkdomain

In [6]:

```
X_train['mean_hits_per_networkDomain'] = X_train.groupby('geoNetwork_networkDomain')['total

X_cv['mean_hits_per_networkDomain'] = X_cv.groupby('geoNetwork_networkDomain')['totals_hits

X_test['mean_hits_per_networkDomain'] = X_test.groupby('geoNetwork_networkDomain')['totals_
```

2. Mean page views per networkdomain

In [7]:

```
X_train['mean_pageViews_per_networkDomain'] = X_train.groupby('geoNetwork_networkDomain')['

X_cv['mean_pageViews_per_networkDomain'] = X_cv.groupby('geoNetwork_networkDomain')['totals

X_test['mean_pageViews_per_networkDomain'] = X_test.groupby('geoNetwork_networkDomain')['to
```

In [26]:

```
X_train.columns
```

Out[26]:

```
Index(['device_operatingSystem', 'device_isMobile', 'device_deviceCategory',
       'geoNetwork_continent', 'geoNetwork_subContinent', 'geoNetwork_countr
y',
       'geoNetwork_region', 'geoNetwork_metro', 'geoNetwork_city',
       'geoNetwork_networkDomain', 'totals_hits', 'totals_pageviews',
       'totals_bounces', 'totals_newVisits', 'trafficSource_campaign',
       'trafficSource_source', 'trafficSource_medium', 'trafficSource_keywor
d',
       'trafficSource_isTrueDirect', 'trafficSource_referralPath',
       'trafficSource_adwordsClickInfo.page',
       'trafficSource_adwordsClickInfo.slot',
       'trafficSource_adwordsClickInfo.gclId',
       'trafficSource_adwordsClickInfo.adNetworkType',
       'trafficSource_adwordsClickInfo.isVideoAd', 'trafficSource_adConten
t',
       'month', 'day', 'weekday', 'mean_hits_per_networkDomain',
       'mean_pageViews_per_networkDomain'],
      dtype='object')
```

In [10]:

```
X_test.columns
```

Out[10]:

```
Index(['device_browser', 'device_operatingSystem', 'device_isMobile',
       'device_deviceCategory', 'geoNetwork_continent',
       'geoNetwork_subContinent', 'geoNetwork_country', 'geoNetwork_region',
       'geoNetwork_metro', 'geoNetwork_city', 'geoNetwork_networkDomain',
       'totals_hits', 'totals_pageviews', 'totals_newVisits', 'totals_bounce
s',
       'trafficSource_campaign', 'trafficSource_source',
       'trafficSource_medium', 'trafficSource_keyword',
       'trafficSource_isTrueDirect', 'trafficSource_adwordsClickInfo.page',
       'trafficSource_adwordsClickInfo.slot',
       'trafficSource_adwordsClickInfo.gclId',
       'trafficSource_adwordsClickInfo.adNetworkType',
       'trafficSource_adwordsClickInfo.isVideoAd',
       'trafficSource_referralPath', 'trafficSource_adContent', 'month', 'da
y',
       'weekday', 'mean_hits_per_networkDomain',
       'mean_pageViews_per_networkDomain'],
      dtype='object')
```

Decicion tree with 2 new features added

In [48]:

```
from sklearn.tree import DecisionTreeRegressor
regr = DecisionTreeRegressor()

param_grid = {"max_depth": [3,5,7,8],
              "min_samples_split": sp_randint(2, 10)}

# run randomized search
random_search = RandomizedSearchCV(regr, param_distributions=param_grid)
random_search.fit(X_train, y_train)

print(random_search.best_params_)
```

```
{'max_depth': 7, 'min_samples_split': 3}
```

In [49]:

```
regr = DecisionTreeRegressor(max_depth=7, min_samples_leaf=3)
regr.fit(X_train, y_train)

y_pred = regr.predict(X_train)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = regr.predict(X_cv)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)
```

```
Train Root Mean Squared Error: 1.6556350802859183
Validation Root Mean Squared Error: 1.771264478785601
```

Random Forest with 2 new features added

In [61]:

```
regr = RandomForestRegressor()

param_grid = {"max_depth": [3,5,7,8],
              "min_samples_split": sp_randint(2, 10),
              "min_samples_leaf": sp_randint(1, 10),
              "n_estimators":[20,50,100, 120, 150, 200, 250, 300]
              }

# run randomized search
random_search = RandomizedSearchCV(regr, param_distributions=param_grid)
random_search.fit(X_train, y_train)

print(random_search.best_params_)
```

```
{'max_depth': 7, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimator
s': 300}
```

In [9]:

```
regr = RandomForestRegressor(max_depth=7, min_samples_leaf=4, min_samples_split= 5, n_estim
regr.fit(X_train, y_train)

y_pred = regr.predict(X_train)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = regr.predict(X_cv)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)
```

```
Train Root Mean Squared Error: 1.6340411444059175
Validation Root Mean Squared Error: 1.7438964897306692
```

XGBoost with two new features added

In [55]:

```
x_model = xgb.XGBRegressor()
param_dist = {"max_depth": [3,5,8,10,12],
              "n_estimators" : [200, 300, 500, 700],
              "learning_rate":[0.001,0.01,0.1,1]
              }

random_search = RandomizedSearchCV(x_model, param_distributions=param_dist)
random_search.fit(X_train, y_train)
```

```
[22:42:52] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[23:05:17] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[23:27:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[23:45:48] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:03:45] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:21:38] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
jective/regression_obj.cu:152: reg:linear is now deprecated in favor of re
g:squarederror.
[00:26:03] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ob
```

In [56]:

```
print(random_search.cv_results_)
print(random_search.best_params_)
```

```
{'mean_fit_time': array([1175.89219131,  262.81834207,  468.71439581,  247.7
8635502,
        360.56631112,  106.04157023,  369.67409596,  385.81286306,
        582.75471988,  105.76499772]), 'std_fit_time': array([1.15444146e+0
2, 7.53371681e-01, 3.31792372e+00, 1.56776635e+00,
       1.66416508e+00, 4.15945400e-01, 3.28929963e+00, 1.21758240e+00,
       3.19339326e+00, 9.51309568e-02]), 'mean_score_time': array([9.3597698
7, 1.15156999, 3.8644443 , 1.38140564, 2.34076157,
       0.46022358, 1.60283728, 2.02775812, 6.86246924, 0.39199848]), 'std_sc
ore_time': array([1.10452051, 0.0143809 , 0.07004821, 0.03502578, 0.0287519
4,
       0.01510045, 0.0249983 , 0.01614348, 0.06422487, 0.00608929]), 'param_
n_estimators': masked_array(data=[700, 500, 300, 200, 300, 200, 700, 500, 30
0, 200],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_max_depth': masked_array(data=[10, 3, 10,
8, 8, 3, 3, 5, 12, 3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_learning_rate': masked_array(data=[0.1, 0.
1, 1, 0.01, 0.1, 0.1, 0.01, 0.001, 1, 0.01],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'n_estimators': 700, 'max_depth': 10,
'learning_rate': 0.1}, {'n_estimators': 500, 'max_depth': 3, 'learning_rat
e': 0.1}, {'n_estimators': 300, 'max_depth': 10, 'learning_rate': 1}, {'n_es
timators': 200, 'max_depth': 8, 'learning_rate': 0.01}, {'n_estimators': 30
0, 'max_depth': 8, 'learning_rate': 0.1}, {'n_estimators': 200, 'max_depth':
3, 'learning_rate': 0.1}, {'n_estimators': 700, 'max_depth': 3, 'learning_ra
te': 0.01}, {'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.001},
{'n_estimators': 300, 'max_depth': 12, 'learning_rate': 1}, {'n_estimators':
200, 'max_depth': 3, 'learning_rate': 0.01}], 'split0_test_score': array([
0.12592107,  0.25314175, -0.4327011 ,  0.25360882,  0.21278614,
        0.24479884,  0.22280093,  0.17411958, -0.44148072,  0.21592104]), 's
plit1_test_score': array([ 0.23705499,  0.31416664, -0.31538502,  0.2956615
5,  0.2970651 ,
        0.30569818,  0.28618077,  0.15556589, -0.25904025,  0.25613017]), 's
plit2_test_score': array([ 0.24313162,  0.34008794, -0.33448929,  0.3111112
1,  0.31052148,
        0.32863338,  0.3066833 ,  0.16054092, -0.27906414,  0.27347838]), 's
plit3_test_score': array([ 0.19564096,  0.31744838, -0.55241121,  0.3127688
3,  0.28856914,
        0.31069481,  0.30084996,  0.16976087, -0.40428093,  0.26696678]), 's
plit4_test_score': array([ 0.30165795,  0.33038298, -0.00474581,  0.3105373
,  0.33895755,
        0.31948728,  0.29594571,  0.1525025 , -0.0093298 ,  0.25014733]), 'm
ean_test_score': array([ 0.22068132,  0.31104554, -0.32794648,  0.29673754,
0.28957988,
        0.3018625 ,  0.28249214,  0.16249795, -0.27863917,  0.25252874]), 's
td_test_score': array([0.0581994 , 0.0304006 , 0.18221812, 0.02242892, 0.042
02626,
       0.02958478, 0.03059368, 0.00824142, 0.15179925, 0.02003052]), 'rank_t
```

```
est_score': array([ 7,  1, 10,  3,  4,  2,  5,  8,  9,  6])}
{'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.1}
```

In [38]:

```
x_model = xgb.XGBRegressor(learning_rate =0.1,n_estimators=500,max_depth=3)

x_model.fit(X_train.values, y_train)

y_pred = x_model.predict(X_train.values)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = x_model.predict(X_cv.values)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)
```

```
C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages\xgboost\core.py:587: Fu
tureWarning:

Series.base is deprecated and will be removed in a future version


[00:39:18] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/obje
ctive/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:sq
uarederror.
Train Root Mean Squared Error: 1.586328663503075
Validation Root Mean Squared Error: 1.7316440303161333
```

Ensemble model

In [107]:

```
from mlxtend.regressor import StackingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
```

In [9]:

```
lr = LinearRegression()
svr_lin = SVR(kernel='linear')
ridge = Ridge(random_state=1)
svr_rbf = SVR(kernel='rbf')

x_model = StackingRegressor(regressors=[svr_lin, lr, ridge],
                            meta_regressor=svr_rbf)

x_model.fit(X_train, y_train)

y_pred = x_model.predict(X_train.values)
mse =mean_squared_error(y_train, y_pred)
rmse = math.sqrt(mse)
print("Train Root Mean Squared Error:", rmse)

y_pred = x_model.predict(X_cv.values)
mse =mean_squared_error(y_cv, y_pred)
rmse = math.sqrt(mse)
print("Validation Root Mean Squared Error:", rmse)
```

```
Train Root Mean Squared Error: 1.6340411444059175
Validation Root Mean Squared Error: 1.7438964897306692
```

Deep learning model

In [8]:

```
# Importing libraries
from keras.models import Sequential
from keras.layers import CuDNNLSTM, LSTM
from keras.layers.core import Dense, Dropout
from keras.layers import BatchNormalization
from keras.initializers import he_normal
from keras.regularizers import l2
from keras.layers import Activation,Conv1D,MaxPool1D, Dense, Dropout, Flatten
from keras.layers import MaxPooling1D
```

```
Using TensorFlow backend.
```

In [9]:

```
X_train = np.resize(X_train, new_shape=(722922, 32,1))
X_cv = np.resize(X_cv, new_shape=(180730,32,1))
```

In [10]:

```
import keras
from keras.callbacks import Callback, EarlyStopping,ModelCheckpoint
from keras.layers import Flatten, Reshape
from keras.layers import concatenate, GRU, Input
from keras.models import Model
from time import time
from keras.callbacks import TensorBoard

# define CNN model
inputs = Input(shape=(32,1),name="other_input")
x = Conv1D(filters=32,kernel_size=3,strides=1)(inputs)
x = CuDNNLSTM(64,return_sequences=True)(x)
x = CuDNNLSTM(128,return_sequences=True)(x)

x = Flatten()(x)

output = Dense(1, activation = 'sigmoid')(x)

model = Model(inputs, output)

#To visualize, run -  tensorboard --log_dir=logs/ in command prompt

tensorboard = TensorBoard(log_dir="logs/".format(time))

model.compile(loss="mean_squared_error", optimizer=keras.optimizers.Adam(lr=0.0006,decay =
print(model.summary())
```

```
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is depreca
ted. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:74: The name tf.get_default_graph is de
precated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is dep
recated. Please use tf.random.uniform instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\optimizers.py:790: The name tf.train.Optimizer is deprecated. Please
use tf.compat.v1.train.Optimizer instead.


_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
other_input (InputLayer)     (None, 32, 1)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 30, 32)            128
_________________________________________________________________
cu_dnnlstm_1 (CuDNNLSTM)     (None, 30, 64)            25088
_________________________________________________________________
cu_dnnlstm_2 (CuDNNLSTM)     (None, 30, 128)           99328
_________________________________________________________________
flatten_1 (Flatten)          (None, 3840)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 3841
=================================================================
```

```
Total params: 128,385
Trainable params: 128,385
Non-trainable params: 0
_________________________________________________________________
None
```

In [11]:

```
model.fit(X_train, y_train, epochs=15, verbose=1, batch_size=100, validation_data=(X_cv, y_
```

```
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:986: The name tf.assign_add is deprecat
ed. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:973: The name tf.assign is deprecated.
Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:2741: The name tf.Session is deprecate
d. Please use tf.compat.v1.Session instead.

Train on 722922 samples, validate on 180730 samples
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:174: The name tf.get_default_session is
deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:181: The name tf.ConfigProto is depreca
ted. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:190: The name tf.global_variables is de
precated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:199: The name tf.is_variable_initialize
d is deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\backend\tensorflow_backend.py:206: The name tf.variables_initializer
is deprecated. Please use tf.compat.v1.variables_initializer instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\callbacks.py:850: The name tf.summary.merge_all is deprecated. Please
use tf.compat.v1.summary.merge_all instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\callbacks.py:853: The name tf.summary.FileWriter is deprecated. Pleas
e use tf.compat.v1.summary.FileWriter instead.

Epoch 1/15
722922/722922 [==============================] - 71s 98us/step - loss: 3.885
9 - val_loss: 4.5374
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages
\keras\callbacks.py:995: The name tf.Summary is deprecated. Please use tf.co
mpat.v1.Summary instead.

Epoch 2/15
722922/722922 [==============================] - 67s 93us/step - loss: 3.884
9 - val_loss: 4.5365
Epoch 3/15
722922/722922 [==============================] - 68s 93us/step - loss: 3.884
7 - val_loss: 4.5350
Epoch 4/15
722922/722922 [==============================] - 68s 93us/step - loss: 3.884
5 - val_loss: 4.5358
```

```
Epoch 5/15
722922/722922 [==============================] - 68s 93us/step - loss: 3.884
3 - val_loss: 4.5376
Epoch 6/15
722922/722922 [==============================] - 68s 94us/step - loss: 3.884
1 - val_loss: 4.5365
Epoch 7/15
722922/722922 [==============================] - 67s 93us/step - loss: 3.884
0 - val_loss: 4.5361
Epoch 8/15
722922/722922 [==============================] - 68s 93us/step - loss: 3.883
8 - val_loss: 4.5367
Epoch 9/15
722922/722922 [==============================] - 68s 94us/step - loss: 3.883
7 - val_loss: 4.5351
Epoch 10/15
722922/722922 [==============================] - 68s 94us/step - loss: 3.883
5 - val_loss: 4.5353
Epoch 11/15
722922/722922 [==============================] - 69s 95us/step - loss: 3.883
4 - val_loss: 4.5360
Epoch 12/15
722922/722922 [==============================] - 69s 95us/step - loss: 3.883
2 - val_loss: 4.5364
Epoch 13/15
722922/722922 [==============================] - 69s 96us/step - loss: 3.883
1 - val_loss: 4.5358
Epoch 14/15
722922/722922 [==============================] - 69s 95us/step - loss: 3.883
0 - val_loss: 4.5358
Epoch 15/15
722922/722922 [==============================] - 70s 96us/step - loss: 3.882
8 - val_loss: 4.5356
```

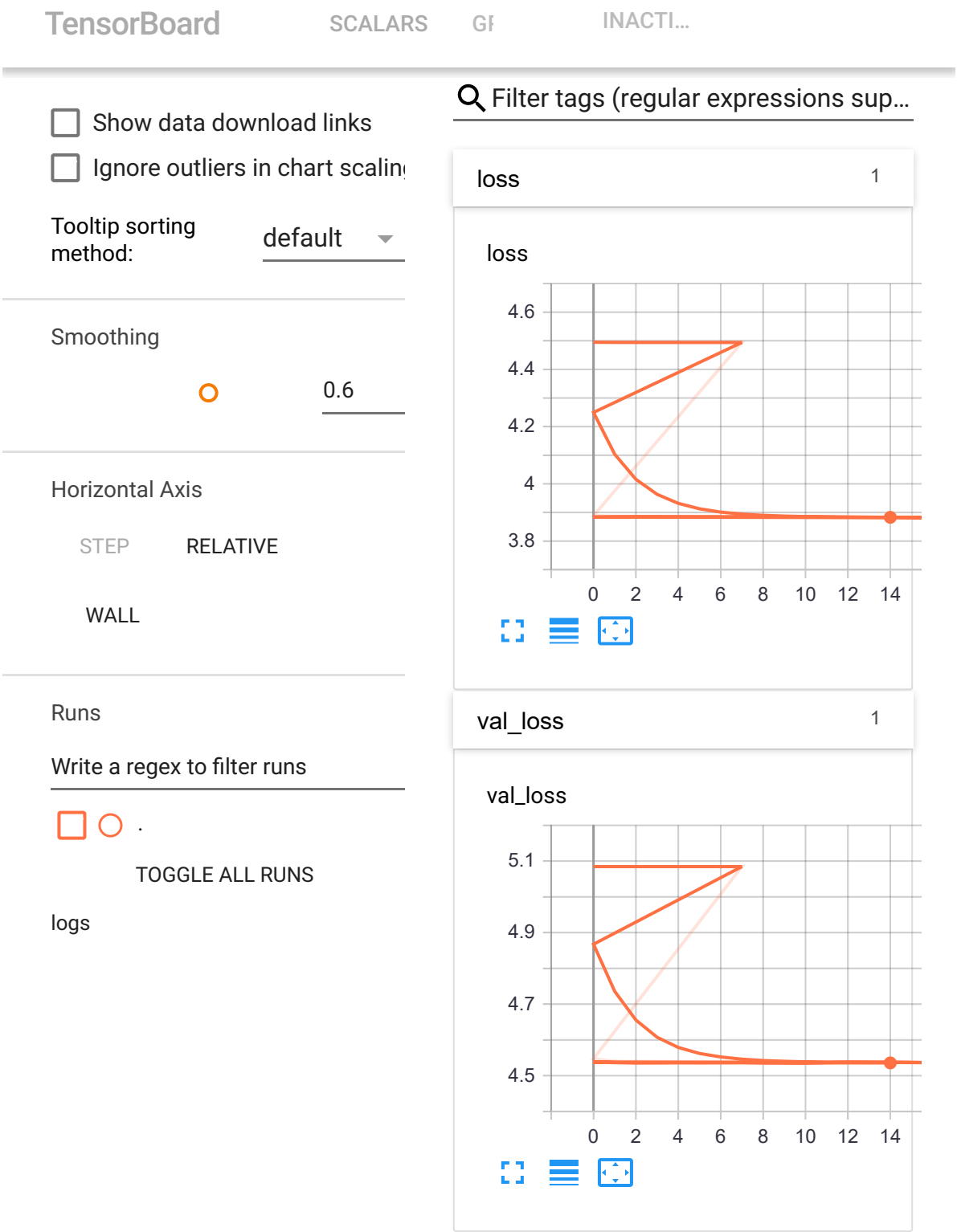
Out[11]:

```
<keras.callbacks.History at 0x205ecec1c88>
```

In [26]:

```
%load_ext tensorboard
# Start TENSORBOARD
%tensorboard --logdir logs --port=8008
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Reusing TensorBoard on port 8008 (pid 12752), started 0:02:55 ago. (Use '!ki
ll 12752' to kill it.)
```



Pickling the best model -- XGBRegressor

In [39]:

```
x_model.feature_names_ = list(X_train.columns)
```

In [40]:

```
# save the model to disk
filename = 'final_model.pkl'
pkl.dump(x_model, open(filename, 'wb'))
```

Model summarization

Approach 1: Basic features

In [28]:

```
ptable = PrettyTable()
ptable.title = "Conclusion from all the models: Aproach 1"
ptable.field_names = ['Model Name','Train rmse','Test rmse']
ptable.add_row(["Light gbm","1.71","1.75"])
ptable.add_row(["Linear Regression","1.78","1.92"])
ptable.add_row(["GBDT","2.50","2.58"])
ptable.add_row(["Random forest","1.61","1.73"])
ptable.add_row(["XGBoost","1.58","1.73"])
print(ptable)
```

```
+-------------------+------------+-----------+
|     Model Name    | Train rmse | Test rmse |
+-------------------+------------+-----------+
|     Light gbm     |    1.71    |    1.75   |
| Linear Regression |    1.78    |    1.92   |
|        GBDT       |    2.50    |    2.58   |
|   Random forest   |    1.61    |    1.73   |
|      XGBoost      |    1.58    |    1.73   |
+-------------------+------------+-----------+
```

Approach 2: Adding two more features

In [29]:

```
ptable = PrettyTable()
ptable.title = "Conclusion from all the models: Aproach 1"
ptable.field_names = ['Model Name','Train rmse','Test rmse']
ptable.add_row(["Decision tree","1.65","1.77"])
ptable.add_row(["Random forest","1.63","1.74"])
ptable.add_row(["XGBoost","1.58","1.73"])
ptable.add_row(["Ensemble","1.63","1.74"])
ptable.add_row(["Deep Learning Model (CNN+RNN)","-","4.53"])
print(ptable)
```

```
+-------------------------------+------------+-----------+
|           Model Name          | Train rmse | Test rmse |
+-------------------------------+------------+-----------+
|         Decision tree         |    1.65    |    1.77   |
|         Random forest         |    1.63    |    1.74   |
|            XGBoost            |    1.58    |    1.73   |
|            Ensemble           |    1.63    |    1.74   |
| Deep Learning Model (CNN+RNN) |     -      |    4.53   |
+-------------------------------+------------+-----------+
```

Pipeline:

You just need to paas the raw data (.csv file) in th get prediction class and prediction results will get saved in your local disk

In [10]:

```
#https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields/notebook.
json_columns = ['device', 'geoNetwork','totals', 'trafficSource']  #these 4 columns are of
def load_dataframe(filename):
    df = pd.read_csv(filename, converters={column: json.loads for column in json_columns},
                     dtype={'fullVisitorId': 'str'}) #loading json columns and specifying t

    for column in json_columns:
        column_as_df = json_normalize(df[column])  #normalizing json columns
        column_as_df.columns = [f"{column}_{subcolumn}" for subcolumn in column_as_df.colum
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    return df
```

In [14]:

```
#https://docs.python.org/3/library/datetime.html
#adding datetime column in data
def add_date_features(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"])

    df["month"]   = df['date'].dt.month  #getting month
    df["day"]     = df['date'].dt.day   #getting day
    df["weekday"] = df['date'].dt.weekday   #getting date
    return df
```

In [17]:

```
def normalize_numerical_columns(dataframe, isTrainDataset = True):
    dataframe["totals_hits"] = dataframe["totals_hits"].astype(float)
    #dataframe["totals.hits"] = (dataframe["totals.hits"] - min(dataframe["totals.hits"]))

    dataframe["totals_pageviews"] = dataframe["totals_pageviews"].astype(float)
    #dataframe["totals.pageviews"] = (dataframe["totals.pageviews"] - min(dataframe["totals

    dataframe["totals_bounces"] = dataframe["totals_bounces"].astype(float)
    dataframe["totals_newVisits"] = dataframe["totals_newVisits"].astype(float)


    if isTrainDataset:
        dataframe["totals_transactionRevenue"] = dataframe["totals_transactionRevenue"].fil
    return dataframe
```

In [18]:

```
def get_predictions(file,model,save_in):
    filename = file


    test_data = load_dataframe(filename)

    test_data = add_date_features(test_data)

    test_data["totals_hits"] = test_data["totals_hits"].astype(float)
    test_data["totals_pageviews"] = test_data["totals_pageviews"].astype(float)
    test_data["totals_bounces"] = test_data["totals_bounces"].astype(float)
    test_data["totals_newVisits"] = test_data["totals_newVisits"].astype(float)



    constant_columns = [column for column in test_data.columns if test_data[column].nunique

    #dropping constant columns
    test_data = test_data.drop(columns=constant_columns)

    #Sorting by date to perform time based slicing
    test_data = test_data.sort_values(by='date',ascending=True)

    submission = pd.DataFrame()
    submission["fullVisitorId"] = test_data["fullVisitorId"]
    submission['predictedLogRevenue'] = np.nan
    ## non relevant columns
    non_relevant = ["visitNumber", "date", "fullVisitorId", "sessionId", "visitId", "visitS

    ## Droping non relevant columns
    test = test_data.drop(columns=non_relevant)

    categorical_features_test = (test.select_dtypes(include=[np.object]))
    categorical_columns = [column for column in test.columns if not column.startswith('tota
    categorical_columns = [column for column in categorical_features_test if column not in

    for column in categorical_columns:

        le = LabelEncoder()

        test_values = list(test[column].values.astype(str))

        le.fit(test_values)

        test[column] = le.transform(test_values)

    test['device_isMobile'] = le.transform(test_values)



    test = normalize_numerical_columns(test,isTrainDataset=False)

    test = test.fillna(0).astype('float32')

    test['mean_pageViews_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['to
    test['mean_hits_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['totals_

    test = test.drop('channelGrouping',axis=1)
```

```
    # load the model from disk
    loaded_model = pkl.load(open(model, 'rb'))

    test.columns = loaded_model.feature_names_
    submission['predictedLogRevenue'] = np.log(loaded_model.predict(test.values))
    submission.to_csv(save_in)
```

In [19]:

```
get_predictions('test.csv','final_model.pkl','predictions.csv')
```

```
[21:24:18] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/obje
ctive/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:sq
uarederror.

C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages\ipykernel_launcher.py:6
5: RuntimeWarning:

invalid value encountered in log
```

In [22]:

```
def get_predictions_and_comapre_results(file,model):
    filename = file


    test_data = load_dataframe(filename)

    test_data = add_date_features(test_data)

    test_data["totals_hits"] = test_data["totals_hits"].astype(float)
    test_data["totals_pageviews"] = test_data["totals_pageviews"].astype(float)
    test_data["totals_bounces"] = test_data["totals_bounces"].astype(float)
    test_data["totals_newVisits"] = test_data["totals_newVisits"].astype(float)



    constant_columns = [column for column in test_data.columns if test_data[column].nunique

    #dropping constant columns
    test_data = test_data.drop(columns=constant_columns)

    #Sorting by date to perform time based slicing
    test_data = test_data.sort_values(by='date',ascending=True)


    ## non relevant columns
    non_relevant = ["visitNumber", "date", "fullVisitorId", "sessionId", "visitId", "visitS

    ## Droping non relevant columns
    test = test_data.drop(columns=non_relevant)

    categorical_features_test = (test.select_dtypes(include=[np.object]))
    categorical_columns = [column for column in test.columns if not column.startswith('tota
    categorical_columns = [column for column in categorical_features_test if column not in

    for column in categorical_columns:

        le = LabelEncoder()

        test_values = list(test[column].values.astype(str))

        le.fit(test_values)

        test[column] = le.transform(test_values)

    test['device_isMobile'] = le.transform(test_values)



    test = normalize_numerical_columns(test,isTrainDataset=False)

    test = test.fillna(0).astype('float32')

    test['mean_pageViews_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['to
    test['mean_hits_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['totals_

    test = test.drop('channelGrouping',axis=1)


    # load the model from disk
```

```
loaded_model = pkl.load(open(model, 'rb'))

test.columns = loaded_model.feature_names_

predictions = np.log(loaded_model.predict(test.values))

mse =mean_squared_error(truth_value, predictions)
rmse = math.sqrt(mse)
print("Root Mean Squared Error:", rmse)

print("-"*70)

importances = loaded_model.feature_importances_

indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [test.columns[i] for i in indices]

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(30), importances[indices])

# Add feature names as x-axis labels
plt.xticks(range(30),names, rotation=90)

# Show plot
plt.show()
```

---Done