Objective of the notebook:

In this notebook, let us explore the given dataset and make some inferences along the way. Also finally we will build a baseline light gbm model to get started.

Objective of the competition:

In this competition, we a're challenged to analyze a Google Merchandise Store (also known as GStore, where Google swag is sold) customer dataset to predict revenue per customer.

Code

About the dataset:

Similar to most other kaggle competitions, 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.

The data fields in the given files are

- 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.
- sessionId - A unique identifier for this visit to the store.
- 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).

Also it is important to note that some of the fields are in json format.

Thanks to this wonderful kernel (https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields/notebook) by Julian (https://www.kaggle.com/julian3833), we can convert all the json fields in the file to a flattened csv format which generally use in other competitions.

In [2]:

```
def load_df(csv_path='../input/train.csv', nrows=None):
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON
_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     nrows=nrows)

    for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn
in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=T
rue, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df.shape}")
    return df
```

In [3]:

```
%%time
train_df = load_df()
test_df = load_df("../input/test.csv")
```

In [4]:
```
train_df.head()
```

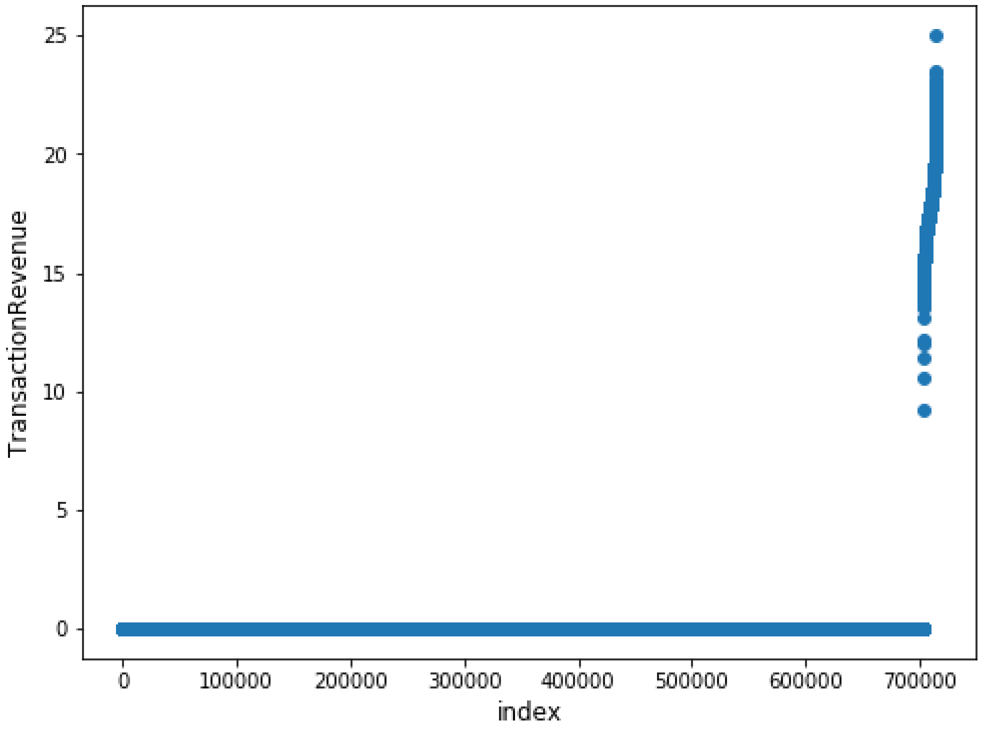
Out[4]:

	channelGrouping	date	fullVisitorId	sessionId	so
0	Organic Search	20160902	1131660440785968503	1131660440785968503_1472830385	No
1	Organic Search	20160902	377306020877927890	377306020877927890_1472880147	No
2	Organic Search	20160902	3895546263509774583	3895546263509774583_1472865386	No
3	Organic Search	20160902	4763447161404445595	4763447161404445595_1472881213	No
4	Organic Search	20160902	27294437909732085	27294437909732085_1472822600	No

Target Variable Exploration:

Since we are predicting the natural log of sum of all transactions of the user, let us sum up the transaction revenue at user level and take a log and then do a scatter plot.

In [5]:
```
train_df["totals.transactionRevenue"] = train_df["totals.transactionRevenue"].astype('float')
gdf = train_df.groupby("fullVisitorId")["totals.transactionRevenue"].sum().reset_index()

plt.figure(figsize=(8,6))
plt.scatter(range(gdf.shape[0]), np.sort(np.log1p(gdf["totals.transactionRevenue"].values)))
plt.xlabel('index', fontsize=12)
plt.ylabel('TransactionRevenue', fontsize=12)
plt.show()
```



Wow, This confirms the first two lines of the competition overview.

```
* The 80/20 rule has proven true for many businesses–only a small percentage
 of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies.
```

Infact in this case, the ratio is even less.

In [6]:
```
nzi = pd.notnull(train_df["totals.transactionRevenue"]).sum()
nzr = (gdf["totals.transactionRevenue"]>0).sum()
print("Number of instances in train set with non-zero revenue : ", nzi,
 " and ratio is : ", nzi / train_df.shape[0])
print("Number of unique customers with non-zero revenue : ", nzr, "and
 the ratio is : ", nzr / gdf.shape[0])
```

So the ratio of revenue generating customers to customers with no revenue is in the ratio os 1.3%

Since most of the rows have non-zero revenues, in the following plots let us have a look at the count of each category of the variable along with the number of instances where the revenue is not zero.

Number of visitors and common visitors:

Now let us look at the number of unique visitors in the train and test set and also the number of common visitors.

In [7]:
```
print("Number of unique visitors in train set : ",train_df.fullVisitorI
d.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(trai
n_df.fullVisitorId.unique()).intersection(set(test_df.fullVisitorId.un
ique())) ))
```

Columns with constant values:

Looks like there are quite a few features with constant value in the train set. Let us get the list of these features. As pointed by Svitlana in the comments below, let us not include the columns which has constant value and some null values.

In [8]:
```
const_cols = [c for c in train_df.columns if train_df[c].nunique(dropn
a=False)==1 ]
const_cols
```

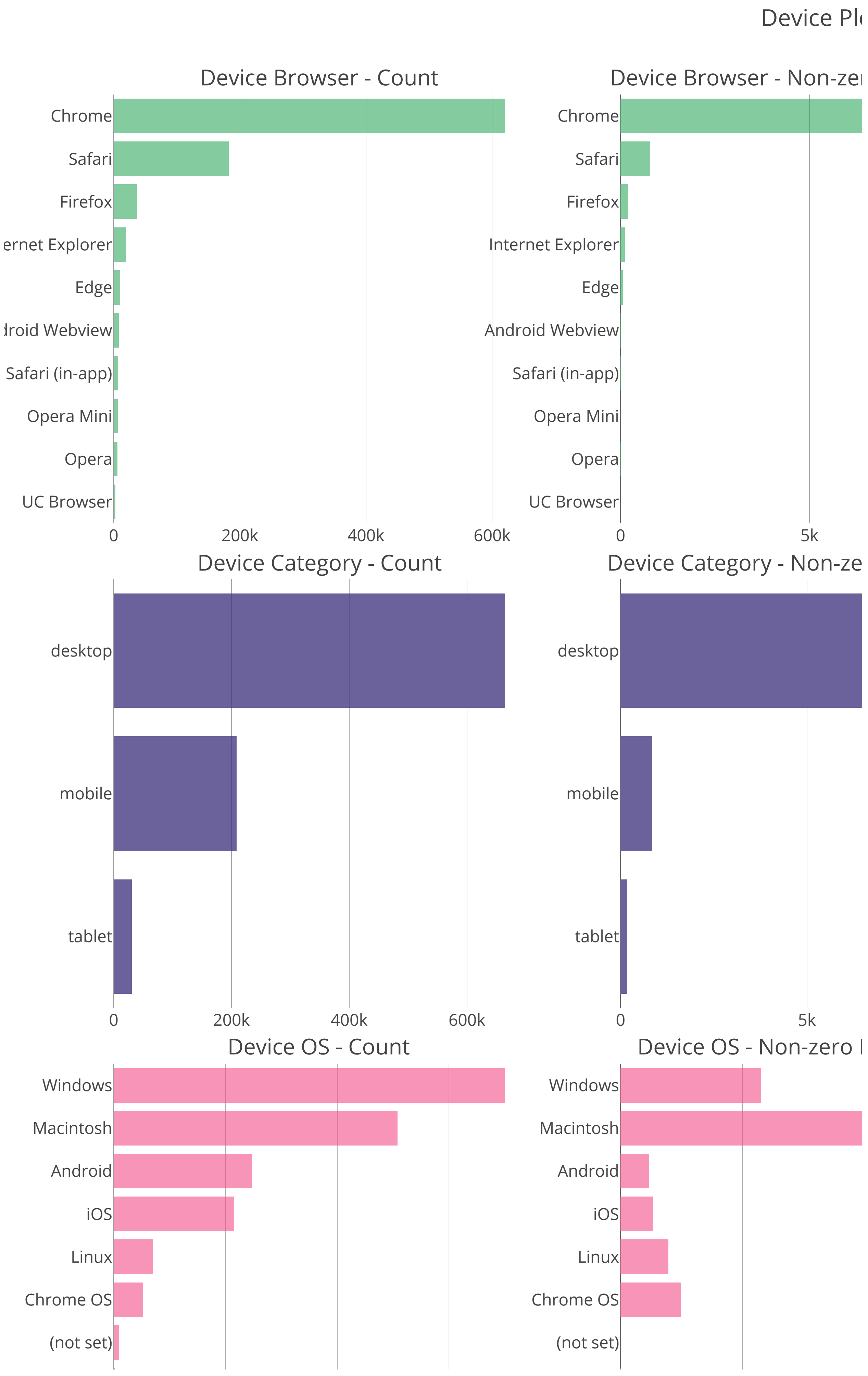
Out[8]:

They are quite a few. Since the values are constant, we can just drop them from our feature list and save some memory and time in our modeling process.

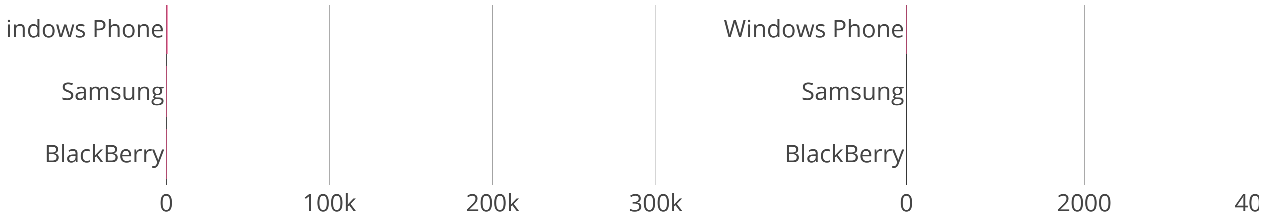
Device Information:

In [9]:

```
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

# Device Browser
cnt_srs = train_df.groupby('device.browser')['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)')

# Device Category
cnt_srs = train_df.groupby('device.deviceCategory')['totals.transaction
Revenue'].agg(['size', 'count', 'mean'])
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"]
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
trace4 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(71, 58,
 131, 0.8)')
trace5 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head
(10), 'rgba(71, 58, 131, 0.8)')
trace6 = horizontal_bar_chart(cnt_srs["mean"].head(10), 'rgba(71, 58,
 131, 0.8)')

# Operating system
cnt_srs = train_df.groupby('device.operatingSystem')['totals.transactio
nRevenue'].agg(['size', 'count', 'mean'])
cnt_srs.columns = ["count", "count of non-zero revenue", "mean"]
cnt_srs = cnt_srs.sort_values(by="count", ascending=False)
trace7 = horizontal_bar_chart(cnt_srs["count"].head(10), 'rgba(246, 78,
 139, 0.6)')
trace8 = horizontal_bar_chart(cnt_srs["count of non-zero revenue"].head
(10),'rgba(246, 78, 139, 0.6)')
trace9 = horizontal_bar_chart(cnt_srs["mean"].head(10),'rgba(246, 78, 1
39, 0.6)')

# Creating two subplots
fig = tools.make_subplots(rows=3, cols=3, vertical_spacing=0.04,
                          subplot_titles=["Device Browser - Count", "De
vice Browser - Non-zero Revenue Count", "Device Browser - Mean Revenue"
,
                                          "Device Category - Count",
"Device Category - Non-zero Revenue Count", "Device Category - Mean Rev
enue",
                                          "Device OS - Count", "Device
 OS - Non-zero Revenue Count", "Device OS - Mean Revenue"])

fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 1, 3)
fig.append_trace(trace4, 2, 1)
fig.append_trace(trace5, 2, 2)
fig.append_trace(trace6, 2, 3)
fig.append_trace(trace7, 3, 1)
fig.append_trace(trace8, 3, 2)
fig.append_trace(trace9, 3, 3)

fig['layout'].update(height=1200, width=1200, paper_bgcolor='rgb(233,2
33,233)', title="Device Plots")
py.iplot(fig, filename='device-plots')
```



Device Pl
Device Browser - Count
Device Browser - Non-ze
Chrome
Safari
Firefox
ernet Explorer
Internet Explorer
Edge
droid Webview
Android Webview
Safari (in-app)
Opera Mini
Opera
UC Browser
0
200k
400k
600k
5k
Device Category - Count
Device Category - Non-ze
desktop
mobile
tablet
Device OS - Count
Device OS - Non-zero
Windows
Macintosh
Android
iOS
Linux
Chrome OS
(not set)

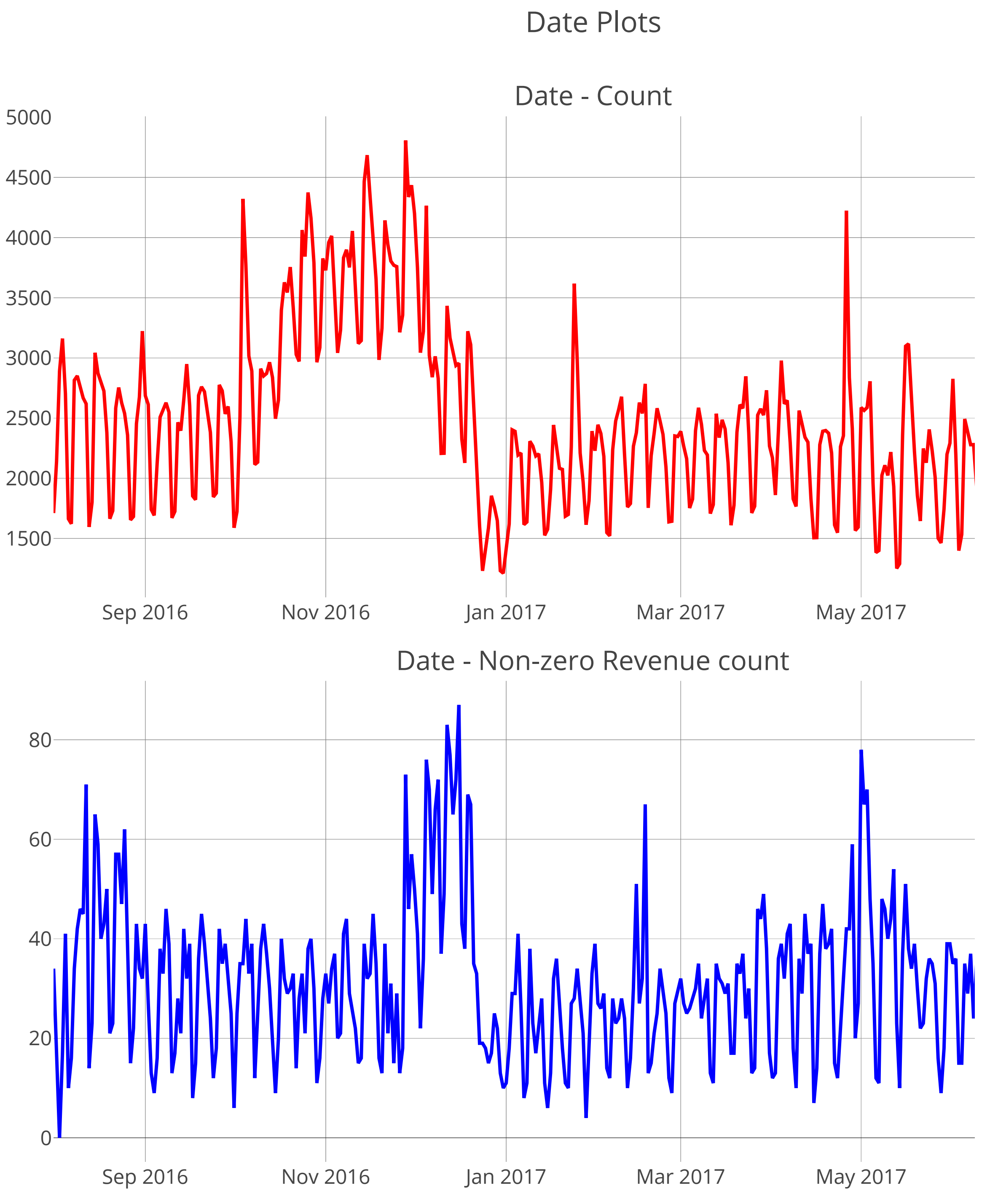


Inferences:

- Device browser distribution looks similar on both the count and count of non-zero revenue plots
- On the device category front, desktop seem to have higher percentage of non-zero revenue counts compared to mobile devices.
- In device operating system, though the number of counts is more from windows, the number of counts where revenue is not zero is more for Macintosh.
- Chrome OS also has higher percentage of non-zero revenue counts
- On the mobile OS side, iOS has more percentage of non-zero revenue counts compared to Android

Date Exploration:

Code

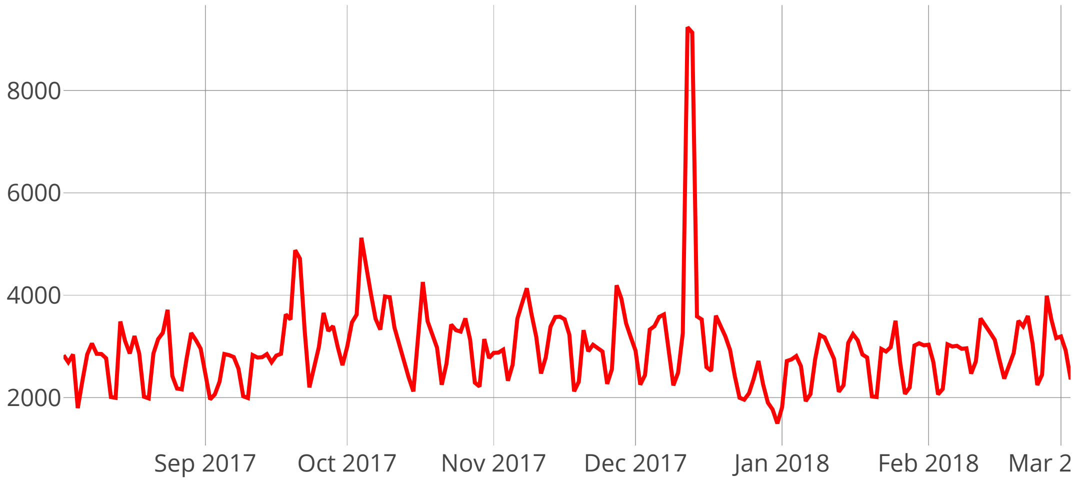

Date Plots
Date - Count
5000
4500
4000
3500
3000
2500
2000
1500
Sep 2016
Nov 2016
Jan 2017
Mar 2017
May 2017
Date - Non-zero Revenue count
80
60
40
20
0
Sep 2016
Nov 2016
Jan 2017
Mar 2017
May 2017

Inferences:

- We have data from 1 Aug, 2016 to 31 July, 2017 in our training dataset
- In Nov 2016, though there is an increase in the count of visitors, there is no increase in non-zero revenue counts during that time period (relative to the mean).

Code

Dates in Test set

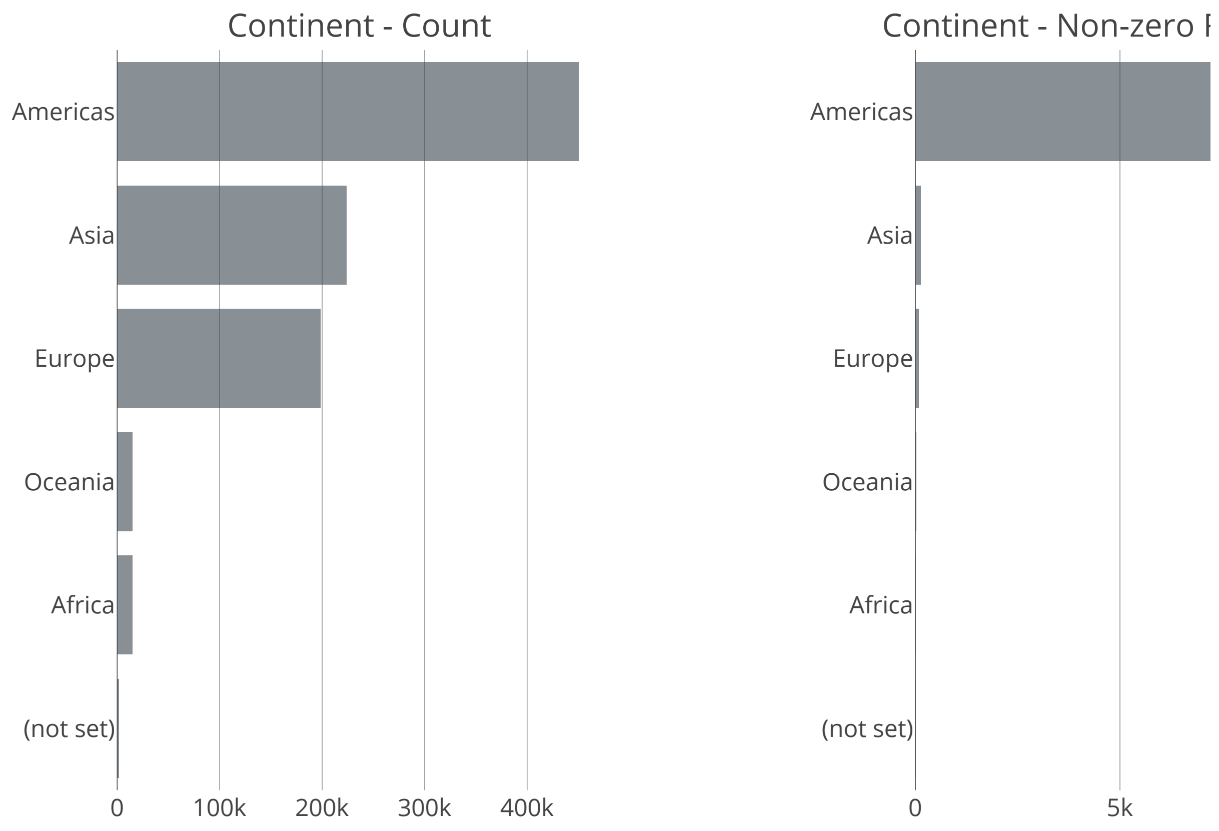


In the test set, we have dates from 2 Aug, 2017 to 30 Apr, 2018. So there are no common dates between train and test set. So it might be a good idea to do time based validation for this dataset.

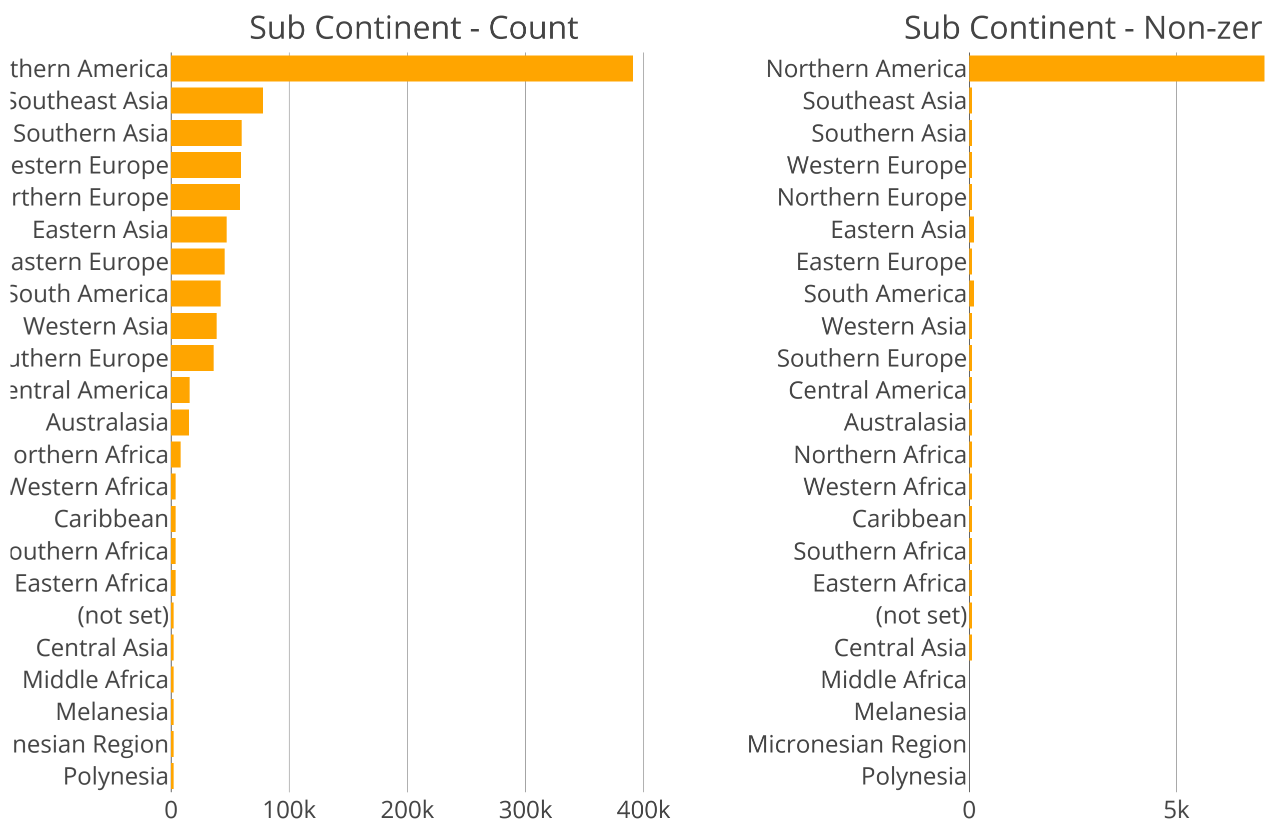
Geographic Information:

Code

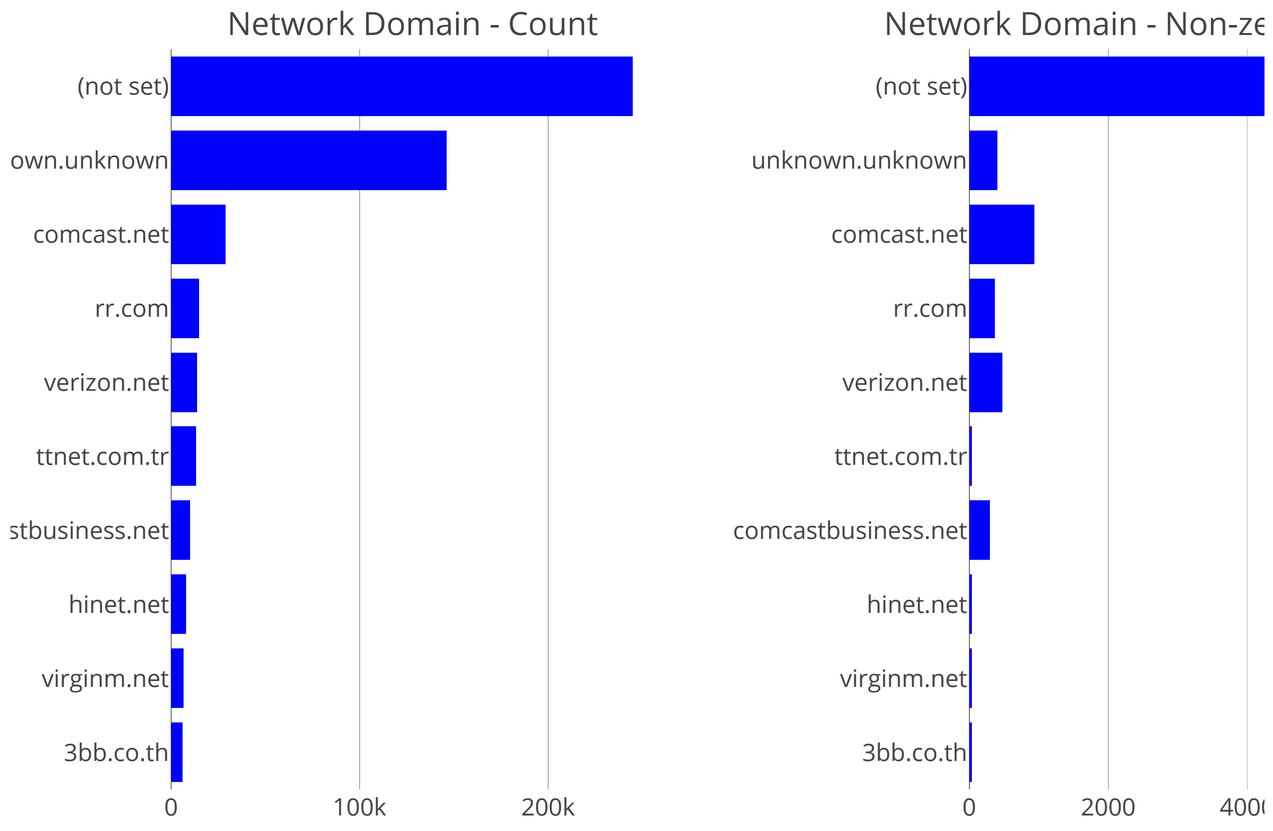
Geography



Continent - Count
Americas
Asia
Europe
Oceania
Africa
(not set)
0
100k
200k
300k
400k
Continent - Non-zero
Americas
Asia
Europe
Oceania
Africa
(not set)
0
5k



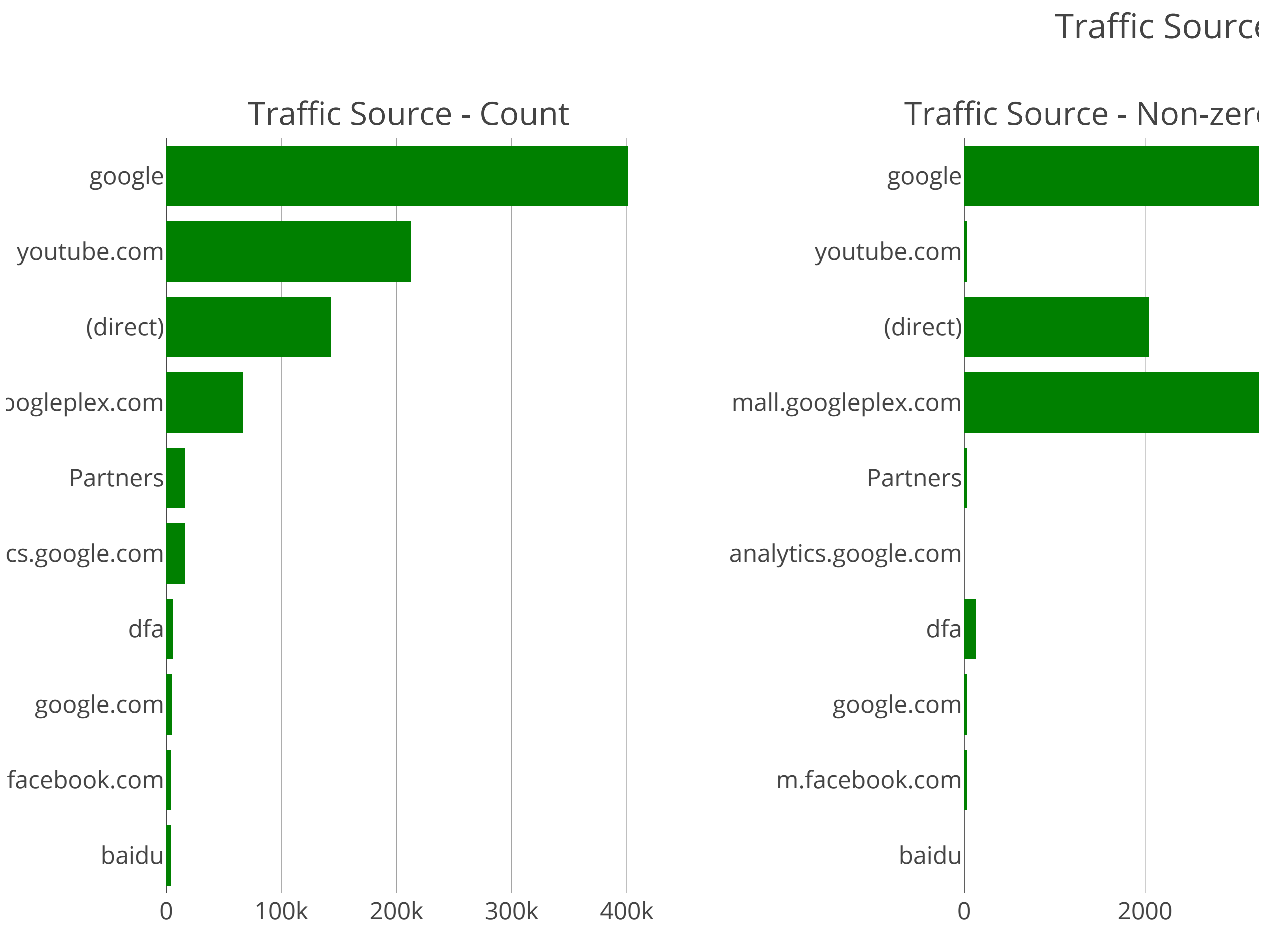
Sub Continent - Count
thern America
Southeast Asia
Southern Asia
estern Europe
rthern Europe
Eastern Asia
astern Europe
South America
Western Asia
uthern Europe
entral America
Australasia
orthern Africa
Western Africa
Caribbean
outhern Africa
Eastern Africa
(not set)
Central Asia
Middle Africa
Melanesia
nesian Region
Polynesia
0
100k
200k
300k
400k
Sub Continent - Non-zer
Northern America
Southeast Asia
Southern Asia
Western Europe
Northern Europe
Eastern Asia
Eastern Europe
South America
Western Asia
Southern Europe
Central America
Australasia
Northern Africa
Western Africa
Caribbean
Southern Africa
Eastern Africa
(not set)
Central Asia
Middle Africa
Melanesia
Micronesian Region
Polynesia
0
5k

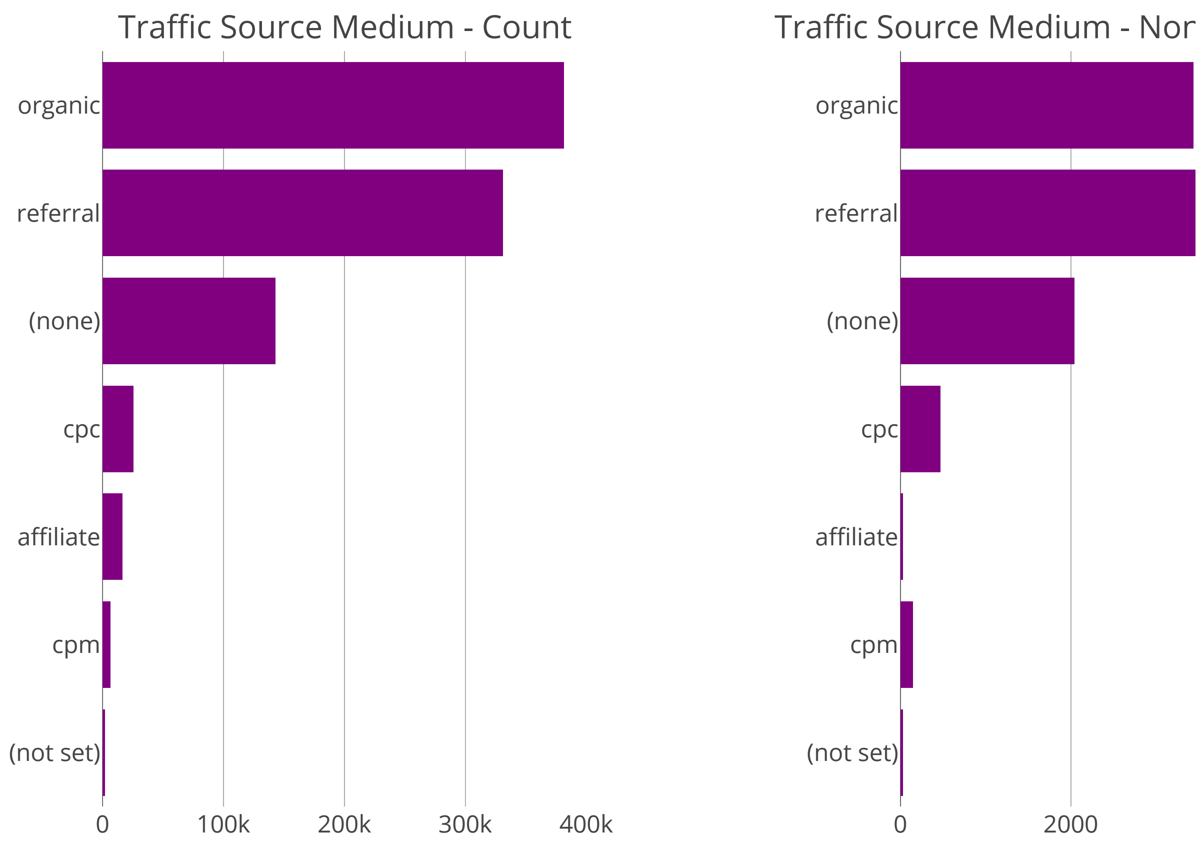


Inferences:

- On the continent plot, we can see that America has both higher number of counts as well as highest number of counts where the revenue is non-zero
- Though Asia and Europe has high number of counts, the number of non-zero revenue counts from these continents are comparatively low.
- We can infer the first two points from the sub-continents plot too.
- If the network domain is "unknown.unknown" rather than "(not set)", then the number of counts with non-zero revenue tend to be lower.

Traffic Source:


Traffic Source
Traffic Source - Count
Traffic Source - Non-zer
google
youtube.com
(direct)
ogleplex.com
Partners
cs.google.com
dfa
google.com
facebook.com
baidu
mall.googleplex.com
analytics.google.com
m.facebook.com
0
100k
200k
300k
400k
2000


Traffic Source Medium - Count
Traffic Source Medium - Nor
organic
referral
(none)
cpc
affiliate
cpm
(not set)
0
100k
200k
300k
400k
2000

Inferences:

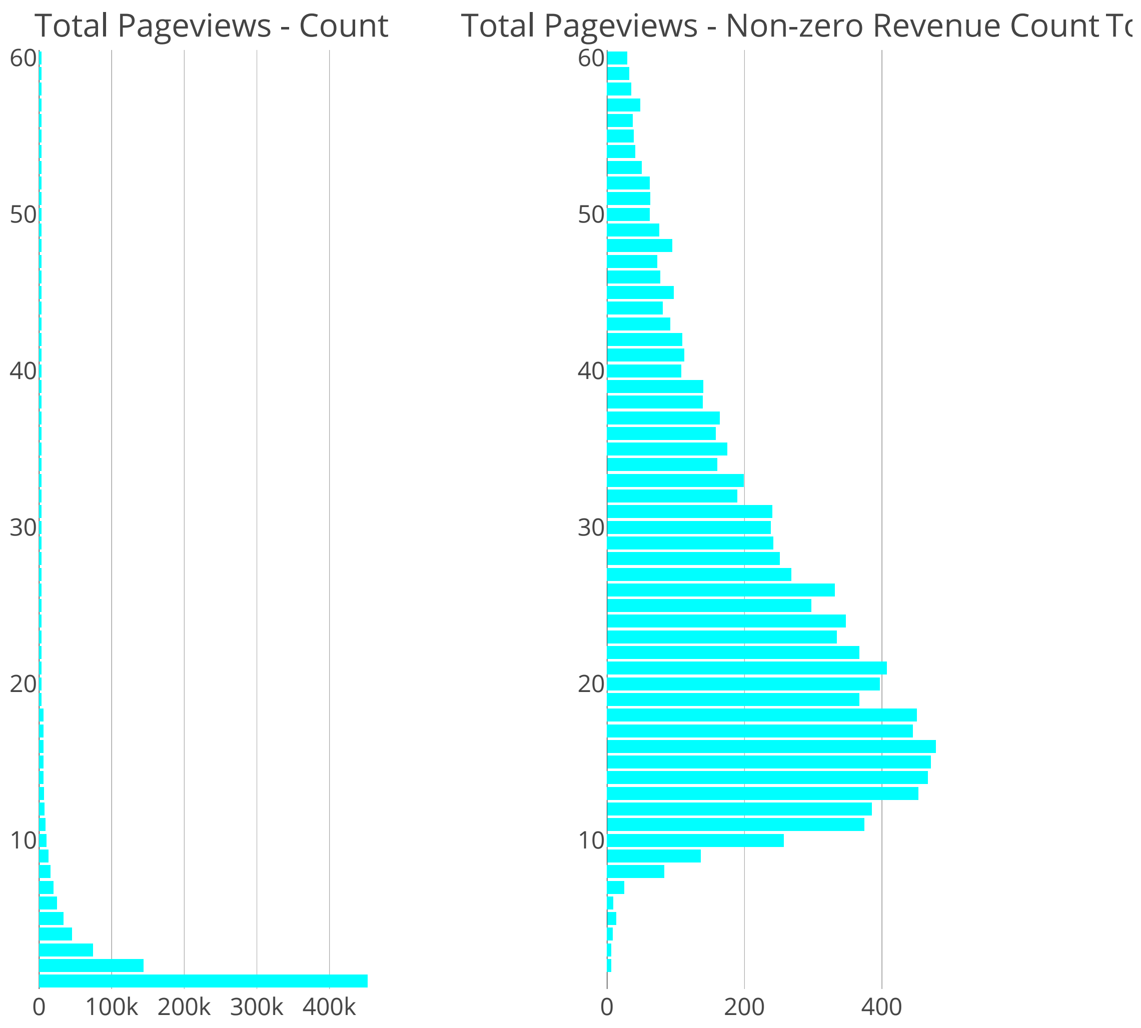
- In the traffic source plot, though Youtube has high number of counts in the dataset, the number of non-zero revenue counts are very less.
- Google plex has a high ratio of non-zero revenue count to total count in the traffic source plot.
- On the traffic source medium, "referral" has more number of non-zero revenue count compared to "organic" medium.

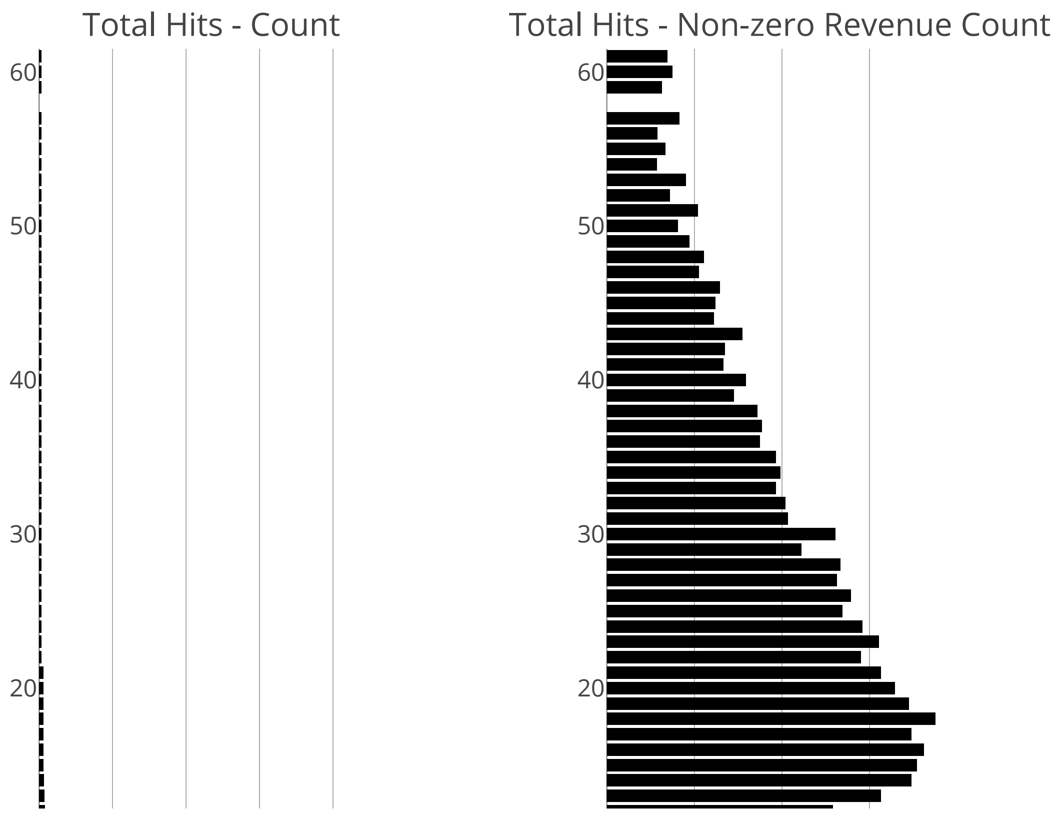
Visitor Profile:

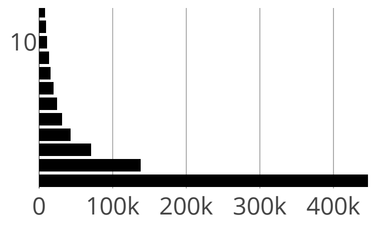
Now let us look at the visitor profile variables like number of pageviews by the visitor, number of hits by the visitor and see how they look.

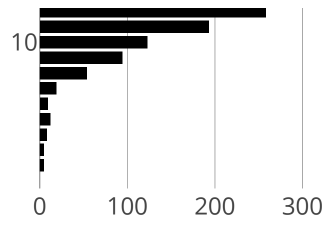
Code

Visitor Profile Plots









Inferences:

- Both these variables look very predictive
- Count plot shows decreasing nature i.e. we have a very high total count for less number of hits and page views per visitor transaction and the overall count decreases when the number of hits per visitor transaction increases.
- On the other hand, we can clearly see that when the number of hits / pageviews per visitor transaction increases, we see that there is a high number of non-zero revenue counts.

Baseline Model:

Now let us build a baseline model on this dataset. Before we start building models, let us look at the variable names which are there in train dataset and not in test dataset.

In [15]:
```
print("Variables not in test but in train : ", set(train_df.columns).di
fference(set(test_df.columns)))
```

So apart from target variable, there is one more variable "trafficSource.campaignCode" not present in test dataset. So we need to remove this variable while building models. Also we can drop the constant variables which we got earlier.

Also we can remove the "sessionId" as it is a unique identifier of the visit.

In [16]:
```
cols_to_drop = const_cols + ['sessionId']

train_df = train_df.drop(cols_to_drop + ["trafficSource.campaignCode"],
 axis=1)
test_df = test_df.drop(cols_to_drop, axis=1)
```

Now let us create development and validation splits based on time to build the model. We can take the last two months as validation sample.

In [17]:

```
# Impute 0 for missing target values
train_df["totals.transactionRevenue"].fillna(0, inplace=True)
train_y = train_df["totals.transactionRevenue"].values
train_id = train_df["fullVisitorId"].values
test_id = test_df["fullVisitorId"].values


# label encode the categorical variables and convert the numerical varia
bles to float
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.adwordsClickInfo.adNetworkType",
            "trafficSource.adwordsClickInfo.gclId",
            "trafficSource.adwordsClickInfo.page",
            "trafficSource.adwordsClickInfo.slot", "trafficSource.campa
ign",
            "trafficSource.keyword", "trafficSource.medium",
            "trafficSource.referralPath", "trafficSource.source",
            'trafficSource.adwordsClickInfo.isVideoAd', 'trafficSource.
isTrueDirect']
for col in cat_cols:
    print(col)
    lbl = preprocessing.LabelEncoder()
    lbl.fit(list(train_df[col].values.astype('str')) + list(test_df[co
l].values.astype('str')))
    train_df[col] = lbl.transform(list(train_df[col].values.astype('st
r')))
    test_df[col] = lbl.transform(list(test_df[col].values.astype('str'
)))


num_cols = ["totals.hits", "totals.pageviews", "visitNumber", "visitSta
rtTime", 'totals.bounces',  'totals.newVisits']
for col in num_cols:
    train_df[col] = train_df[col].astype(float)
    test_df[col] = test_df[col].astype(float)

# Split the train dataset into development and valid based on time
dev_df = train_df[train_df['date']<=datetime.date(2017,5,31)]
val_df = train_df[train_df['date']>datetime.date(2017,5,31)]
```

```
dev_y = np.log1p(dev_df["totals.transactionRevenue"].values)
val_y = np.log1p(val_df["totals.transactionRevenue"].values)

dev_X = dev_df[cat_cols + num_cols]
val_X = val_df[cat_cols + num_cols]
test_X = test_df[cat_cols + num_cols]
```

In [18]:
```
# custom function to run light gbm model
def run_lgb(train_X, train_y, val_X, val_y, test_X):
    params = {
        "objective" : "regression",
        "metric" : "rmse",
        "num_leaves" : 30,
        "min_child_samples" : 100,
        "learning_rate" : 0.1,
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.5,
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1
    }

    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    model = lgb.train(params, lgtrain, 1000, valid_sets=[lgval], early
_stopping_rounds=100, verbose_eval=100)

    pred_test_y = model.predict(test_X, num_iteration=model.best_iterat
ion)
    return pred_test_y, model

# Training the model #
pred_test, model = run_lgb(dev_X, dev_y, val_X, val_y, test_X)
```

There are some negative values in the predictions. So let us make them 0. Also let us sum up the prediction of multiple instances of a vistor to get one value per visitor. Finally let us take log value of these predictions.

In [19]:
```
sub_df = pd.DataFrame({"fullVisitorId":test_id})
pred_test[pred_test<0] = 0
sub_df["PredictedLogRevenue"] = np.expm1(pred_test)
sub_df = sub_df.groupby("fullVisitorId")["PredictedLogRevenue"].sum().
reset_index()
sub_df.columns = ["fullVisitorId", "PredictedLogRevenue"]
sub_df["PredictedLogRevenue"] =
np.log1p(sub_df["PredictedLogRevenue"])
sub_df.to_csv("baseline_lgb.csv", index=False)
```

In [20]:
```
sub_df.head()
```

Out[20]:

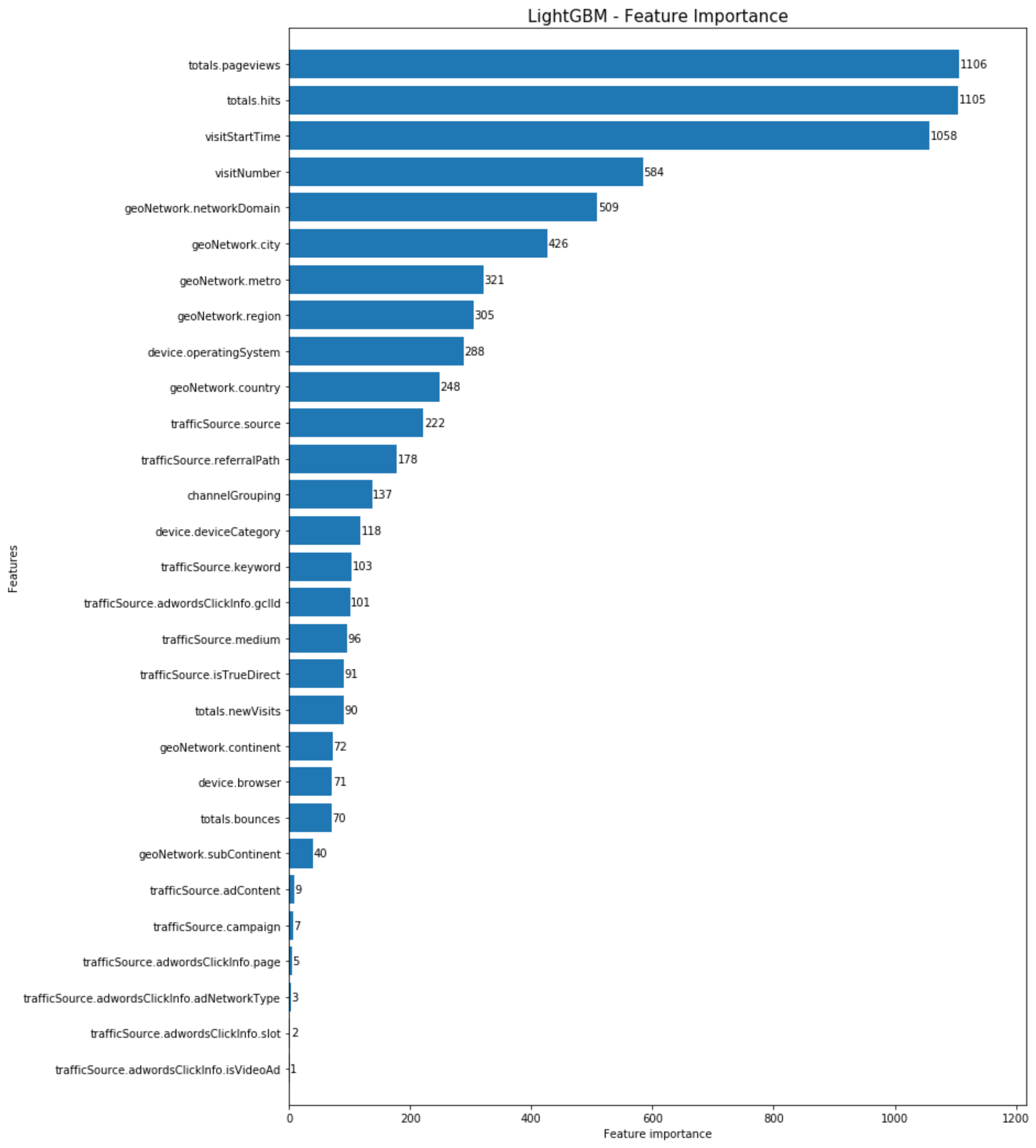
	fullVisitorId	PredictedLogRevenue
0	0000000259678714014	0.516011
1	0000049363351866189	0.000000
2	0000053049821714864	0.007906
3	0000059488412965267	0.048398
4	0000085840370633780	0.011108

This model with a **valid score of 1.69**

Feature Importance:

Now let us have a look at the important features of the light gbm model.

In [21]:
```
fig, ax = plt.subplots(figsize=(12,18))
lgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax)
ax.grid(False)
plt.title("LightGBM - Feature Importance", fontsize=15)
plt.show()
```



"totals.pageviews" turn out to be the most important feature followed by "totals.hits" and "visitStartTime".