If you want see another interesting Kernels please check here https://www.kaggle.com/kabure/kernels

***Please, don't forget to upvote this Kernel to keep me motivated !***

English is not my native language, so sorry for any error.

# Google Analytics Customer Revenue Prediction

## Presenting the initial data:

**Data Fields: fullVisitorIdv** - 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).

First of all, the data are becoming in Json format, so we will need to handle with it and I will use a chunk that I saw in almost all kernel of this competition.

# Objectives:

- I will explore if we have some difference between the browser and if browser is significant to predict sells.
- Which countrys and continents have more acesses and sales ? How it's distributed?!
- Which type of device are most normal in our dataset?
- What's the mobile % of accesses?
- Which is the most frequent Operational System?
- What's the most frequent channelGrouping ?
- Whats the most frequent Weekdays, months, days, year with highest accesses and revenue?

And another bunch of ideas that I will have when start exploring.

## Importing necessary librarys

In [ ]:
```
# Necessary librarys
import os # it's a operational system library, to set some informations
import random # random is to generate random values

import pandas as pd # to manipulate data frames
import numpy as np # to work with matrix
from scipy.stats import kurtosis, skew # it's to explore some statistics of numerical values

import matplotlib.pyplot as plt # to graphics plot
import seaborn as sns # a good library to graphic plots
import squarify # to better understand proportion of categorys - it's a treemap layout algorithm

# Importing librarys to use on interactive graphs
from plotly.offline import init_notebook_mode, iplot, plot
import plotly.graph_objs as go

import json # to convert json in df
from pandas.io.json import json_normalize # to normalize the json file

# to set a style to all graphs
plt.style.use('fivethirtyeight')
init_notebook_mode(connected=True)
```

## Some columns are in Json format so it will be necessary to handle with this problem.

I will use a chunk code inspiration that almost all kernels are using I dont know who did first, but I got on SRK kernel and I did some modifications

In [ ]:
```
columns = ['device', 'geoNetwork', 'totals', 'trafficSource'] # Columns that have json format

dir_path = "../input/" # you can change to your local

# p is a fractional number to skiprows and read just a random sample of the our dataset.
p = 0.07 # *** In this case we will use 50% of data set *** #

#Code to transform the json format columns in table
def json_read(df):
    #joining the [ path + df received]
    data_frame = dir_path + df

    #Importing the dataset
    df = pd.read_csv(data_frame,
                     converters={column: json.loads for column in columns}, # loading the json columns properly
                     dtype={'fullVisitorId': 'str'}, # transforming this column to string
                     skiprows=lambda i: i>0 and random.random() > p)# Number of rows that will be imported randomly

    for column in columns: #loop to finally transform the columns in data frame
        #It will normalize and set the json to a table
        column_as_df = json_normalize(df[column])
        # here will be set the name using the category and subcategory of json columns
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        # after extracting the values, let drop the original columns
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)

    # Printing the shape of dataframes that was imported
    print(f"Loaded {os.path.basename(data_frame)}. Shape: {df.shape}")
    return df # returning the df after importing and transforming
```

# Importing the datasets

In [ ]:
```
%%time
# %%time is used to calculate the timing of code chunk execution #

# We will import the data using the name and extension that will be concatenated with dir_path
df_train = json_read("train.csv")
# The same to test dataset
#df_test = json_read("test.csv")
```

Nice.
After the import and transformation, we have 54 columns.
Now, let's see our data and handle with problemns that we will find

In [ ]:
```
# This command shows the first 5 rows of our dataset
df_train.head()
```

It's interesting because we can see that **SessionId** has the **fullVisitorId** and **VisitStartTime** and **visitId**

Also, the date column we need to transform in datetime format and extract another datetime informations contained in the columns that I quoted above

## Knowing the missing values

In [ ]:
```
# code chunk that I saw in Gabriel Preda kernel
def missing_values(data):
    total = data.isnull().sum().sort_values(ascending = False) # getting the sum of null values and ordering
    percent = (data.isnull().sum() / data.isnull().count() * 100 ).sort_values(ascending = False) #getting the percent and order of null
    df = pd.concat([total, percent], axis=1, keys=['Total', 'Percent']) # Concatenating the total and percent
    print("Total columns at least one Values: ")
    print (df[~(df['Total'] == 0)]) # Returning values of nulls different of 0

    print("\n Total of Sales % of Total: ", round((df_train[df_train['totals.transactionRevenue'] != np.nan]['totals.transactionRevenue'].count() / len(df_train['totals.transactionRevenue']) * 100),4))

    return
```

In [ ]:
```
# calling the missing values function
missing_values(df_train)
```

### Nice. We can see that we have:

Our target have just 1.3% of non-null values
6 columns with 97%+ of missing values
4 columns with 50%+ of missing values
1 column with 22.22%
1 column with 0.004%

We will explore to understand what it looks like

## Let's take a look on datatypes of all columns

- If you want see the code click in "code"
- If you want see the ouput click in "output"

In [ ]:
```
print(df_train.info())
```

### Nice!

Data Types contained in our dataframe:

- bool(1)
- int64(4)
- object(49)

## Creating the function to handle with date

In [ ]:
```
# library of datetime
from datetime import datetime

# This function is to extract date features
def date_process(df):
    df["date"] = pd.to_datetime(df["date"], format="%Y%m%d") # seting the column as pandas datetime
    df["_weekday"] = df['date'].dt.weekday #extracting week day
    df["_day"] = df['date'].dt.day # extracting day
    df["_month"] = df['date'].dt.month # extracting day
    df["_year"] = df['date'].dt.year # extracting day
    df['_visitHour'] = (df['visitStartTime'].apply(lambda x: str(datetime.fromtimestamp(x).hour))).astype(int)

    return df #returning the df after the transformations
```

In [ ]:
```
### Calling the function
```

In [ ]:
```
df_train = date_process(df_train) #calling the function that we created above

df_train.head(n=2) #printing the first 2 rows of our dataset
```

## Before look the unique values in each column, I will drop the constant values that is not useful and will make the df lighter

for it, I will need to give some attention to numerical values

Defining some functions that I will use to call clean the data

- If you want see, click in "code"

In [ ]:
```
def FillingNaValues(df):    # fillna numeric feature
    df['totals.pageviews'].fillna(1, inplace=True).astype(int) #filling NA's with 1
    df['totals.newVisits'].fillna(0, inplace=True).astype(int) #filling NA's with 0
    df['totals.bounces'].fillna(0, inplace=True).astype(int)   #filling NA's with 0
    df["totals.transactionRevenue"] = df["totals.transactionRevenue"].fillna(0.0).astype(float) #filling NA with zero
    df['trafficSource.isTrueDirect'].fillna(False, inplace=True) # filling boolean with False
    df['trafficSource.adwordsClickInfo.isVideoAd'].fillna(True, inplace=True) # filling boolean with True
    df_train.loc[df_train['geoNetwork.city'] == "(not set)", 'geoNetwork.city'] = np.nan
    df_train['geoNetwork.city'].fillna("NaN", inplace=True)

    return df #return the transformed dataframe
```

In [ ]:
```
def NumericalColumns(df):    # fillna numeric feature
    df['totals.pageviews'].fillna(1, inplace=True) #filling NA's with 1
    df['totals.newVisits'].fillna(0, inplace=True) #filling NA's with 0
    df['totals.bounces'].fillna(0, inplace=True)   #filling NA's with 0
    df['trafficSource.isTrueDirect'].fillna(False, inplace=True) # filling boolean with False
    df['trafficSource.adwordsClickInfo.isVideoAd'].fillna(True, inplace=True) # filling boolean with True
    df["totals.transactionRevenue"] = df["totals.transactionRevenue"].fillna(0.0).astype(float) #filling NA with zero
    df['totals.pageviews'] = df['totals.pageviews'].astype(int) # setting numerical column as integer
    df['totals.newVisits'] = df['totals.newVisits'].astype(int) # setting numerical column as integer
    df['totals.bounces'] = df['totals.bounces'].astype(int)  # setting numerical column as integer
    df["totals.hits"] = df["totals.hits"].astype(float) # setting numerical to float
    df['totals.visits'] = df['totals.visits'].astype(int) # seting as int

    return df #return the transformed dataframe
```

Normalize

In [ ]:
```
from sklearn import preprocessing

def Normalizing(df):
    # Use MinMaxScaler to normalize the column
    df["totals.hits"] =  (df['totals.hits'] - min(df['totals.hits'])) / (max(df['totals.hits'])  - min(df['totals.hits']))
    # normalizing the transaction Revenue
    df['totals.transactionRevenue'] = df_train['totals.transactionRevenue'].apply(lambda x: np.log1p(x))
    # return the modified df
    return df
```

### Let's investigate some constant columns

In [ ]:
```
# We will takeoff all columns where we have a unique value (constants)
# It is useful because this columns don't give us none information
discovering_consts = [col for col in df_train.columns if df_train[col].nunique() == 1]

# printing the total of columns dropped and the name of columns
print("Columns with just one value: ", len(discovering_consts), "columns")
print("Name of constant columns: \n", discovering_consts)
```

In [ ]:
```
#Here are all columns that the unique value is 'not available in demo dataset'

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

# It's useul to we have notion that might we have 23 constant columns

- Below I will set a function to better investigate our data and correctly categorize them

In [ ]:
```
# seting the function to show
def knowningData(df, data_type=object, limit=3): #seting the function with df,
    n = df.select_dtypes(include=data_type) #selecting the desired data type
    for column in n.columns: #initializing the loop
        print("##############################################")
        print("Name of column ", column, ': \n', "Uniques: ", df[column].unique()[:limit], "\n",
              " | ## Total nulls: ", (round(df[column].isnull().sum() / len(df[column]) * 100,2)),
              " | ## Total unique values: ", df_train.nunique()[column]) #print the data and % of nulls)
        # print("Percentual of top 3 of: ", column)
        # print(round(df[column].value_counts()[:3] / df[column].value_counts().sum() * 100,2))
        print("##############################################")
```

### I will by object data Type.

Click on "Output" to see the result

In [ ]:
```
# calling our function: object is default
knowningData(df_train)
```

Printing Integers

In [ ]:
```
knowningData(df_train, data_type=int)
```

Printing Float

In [ ]:
```
knowningData(df_train, data_type=float)
```

We haven't float datatype yet.

## I will drop some of this features and fillna or missing in some of them

In [ ]:
```
to_drop = ["socialEngagementType",'device.browserVersion', 'device.browserSize', 'device.flashVersion', 'device.language',
           'device.mobileDeviceBranding', 'device.mobileDeviceInfo', 'device.mobileDeviceMarketingName', 'device.mobileDeviceModel',
           'device.mobileInputSelector', 'device.operatingSystemVersion', 'device.screenColors', 'device.screenResolution',
           'geoNetwork.cityId', 'geoNetwork.latitude', 'geoNetwork.longitude','geoNetwork.networkLocation',
           'trafficSource.adwordsClickInfo.criteriaParameters', 'trafficSource.adwordsClickInfo.gclId', 'trafficSource.campaign',
           'trafficSource.adwordsClickInfo.page', 'trafficSource.referralPath', 'trafficSource.adwordsClickInfo.slot',
           'trafficSource.adContent', 'trafficSource.keyword']
```

In [ ]:
```
df_train.drop(to_drop, axis=1, inplace=True)
```

In [ ]:
```
print("Total features dropped: ", len(to_drop))
print("Shape after dropping: ", df_train.shape)
```

In [ ]:
```
# call the function to transform the numerical columns
df_train = NumericalColumns(df_train)

# Call the function that will normalize some features
df_train = Normalizing(df_train)
```

# looking if we have any mistake on

for c in dummy_feaures: if c in to_drop: print(c)

In [ ]:
```
```

### Let's see the unique values in our dataset.

- if you want see click in "output"

In [ ]:
```
# We will takeoff all columns where we have a unique value
# It is useful because this columns don't give us none information
clean_consts = [col for col in df_train.columns if df_train[col].nunique() == 1]


# this function drop all constant columns, inplacing the data
df_train.drop('trafficSource.adwordsClickInfo.adNetworkType', axis=1, inplace=True)

# printing the total of columns dropped and the name of columns
print("This useful action will drop: ", len(clean_consts), "columns")
print("All dropped columns: \n", clean_consts)
```

The output show us totals.visits and trafficSource.adwordsClickInfo.adNetworkType, but totals,visits can be useful, so I will drop just trafficSource feature

In [ ]:
```
df_train.nunique()
```

Excellent. Now we don't have more constant values

### Based on this output I will select and set a variable with all features by category

In [ ]:
```
'trafficSource.adwordsClickInfo.adNetworkType'
```

In [ ]:
```
dummy_feaures =['channelGrouping', 'device.browser', 'device.deviceCategory', 'geoNetwork.city', 'device.operatingSystem',
                'trafficSource.medium', 'trafficSource.source',
                'geoNetwork.continent', 'geoNetwork.country', 'geoNetwork.metro', 'geoNetwork.networkDomain', 'geoNetwork.region',
                'geoNetwork.subContinent']


numericals = ['totals.visits', '_visitHour', '_day', '_month', '_weekday']
```

# First, let see the distribuition of transactions Revenues

I will start exploring the quantile

In [ ]:
```
# Printing some statistics of our data
print("Transaction Revenue Min Value: ",
      df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"].min()) # printing the min value
print("Transaction Revenue Mean Value: ",
      df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"].mean()) # mean value
print("Transaction Revenue Median Value: ",
      df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"].median()) # median value
print("Transaction Revenue Max Value: ",
      df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"].max()) # the max value

# It I did to plot the quantiles but are not working
#print(round(df_train['totals.transactionRevenue'].quantile([.025,.25,.5,.75,.975]),2))

# seting the figure size of our plots
plt.figure(figsize=(14,5))

# Subplot allow us to plot more than one
# in this case, will be create a subplot grid of 2 x 1
plt.subplot(1,2,1)
# seting the distribuition of our data and normalizing using np.log on values highest than 0 and +
# also, we will set the number of bins and if we want or not kde on our histogram
ax = sns.distplot(np.log(df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"] + 0.01), bins=40, kde=True)
ax.set_xlabel('Transaction RevenueLog', fontsize=15) #seting the xlabel and size of font
ax.set_ylabel('Distribuition', fontsize=15) #seting the ylabel and size of font
ax.set_title("Distribution of Revenue Log", fontsize=20) #seting the title and size of font

# setting the second plot of our grid of graphs
plt.subplot(1,2,2)
# ordering the total of users and seting the values of transactions to understanding
plt.scatter(range(df_train.shape[0]), np.sort(df_train['totals.transactionRevenue'].values))
plt.xlabel('Index', fontsize=15) # xlabel and size of words
plt.ylabel('Revenue value', fontsize=15) # ylabel and size of words
plt.title("Revenue Value Distribution", fontsize=20) # Setting Title and fontsize

plt.show()
```

Nice distribuition... We have very high values on the Transactions Revenue.

## I will see the kurtosis and Skewness of Transaction Revenue

Skew and Kurtosis: 2 Important Statistics terms you need to know

## Skewness

It is the degree of distortion from the symmetrical bell curve or the normal distribution. It measures the lack of symmetry in data distribution. It differentiates extreme values in one versus the other tail. A symmetrical distribution will have a skewness of 0.

**Positive Skewness** means when the tail on the right side of the distribution is longer or fatter. The mean and median will be greater than the mode.
**Negative Skewness** is when the tail of the left side of the distribution is longer or fatter than the tail on the right side. The mean and median will be less than the mode.

#### So, when is the skewness too much?

The rule of thumb seems to be:
If the skewness is between -0.5 and 0.5, the data are fairly symmetrical.
If the skewness is between -1 and -0.5(negatively skewed) or between 0.5 and 1(positively skewed), the data are moderately skewed.
If the skewness is less than -1(negatively skewed) or greater than 1(positively skewed), the data are highly skewed.

## Kurtosis

Kurtosis is all about the tails of the distribution — not the peakedness or flatness. It is used to describe the extreme values in one versus the other tail.
It is actually the measure of outliers present in the distribution.

**High kurtosis** in a data set is an indicator that data has heavy tails or outliers. If there is a high kurtosis, then, we need to investigate why do we have so many outliers. It indicates a lot of things, maybe wrong data entry or other things. Investigate!
**Low kurtosis** in a data set is an indicator that data has light tails or lack of outliers. If we get low kurtosis(too good to be true), then also we need to investigate and trim the dataset of unwanted results

In [ ]:
```
print('Excess kurtosis of normal distribution (should be 0): {}'.format(
    kurtosis(df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"])))
print( 'Skewness of normal distribution (should be 0): {}'.format(
    skew((df_train[df_train['totals.transactionRevenue'] > 0]["totals.transactionRevenue"]))))
```

Our data are fairly symmetrical skewed and have a High Kurtosis.

I will see how many outliers we have on this dataset.

### Function that I created to find the map outlier values

- Click on "code" to see the function

In [ ]:
```
def CalcOutliers(df_num):
    '''

    Leonardo Ferreira 20/10/2018
    Set a numerical value and it will calculate the upper, lower and total number of outliers
    It will print a lot of statistics of the numerical feature that you set on input

    '''
    # calculating mean and std of the array
    data_mean, data_std = np.mean(df_num), np.std(df_num)

    # seting the cut line to both higher and lower values
    # You can change this value
    cut = data_std * 3

    #Calculating the higher and lower cut values
    lower, upper = data_mean - cut, data_mean + cut

    # creating an array of lower, higher and total outlier values
    outliers_lower = [x for x in df_num if x < lower]
    outliers_higher = [x for x in df_num if x > upper]
    outliers_total = [x for x in df_num if x < lower or x > upper]

    # array without outlier values
    outliers_removed = [x for x in df_num if x > lower and x < upper]

    print('Identified lowest outliers: %d' % len(outliers_lower)) # printing total number of values in lower cut of outliers
    print('Identified upper outliers: %d' % len(outliers_higher)) # printing total number of values in higher cut of outliers
    print('Identified outliers: %d' % len(outliers_total)) # printing total number of values outliers of both sides
    print('Non-outlier observations: %d' % len(outliers_removed)) # printing total number of non outlier values
    print("Total percentual of Outliers: ", round((len(outliers_total) / len(outliers_removed) )*100, 4)) # Percentual of outliers in points

    return
```

In [ ]:
```
CalcOutliers(df_train['totals.transactionRevenue']) # Call
```

In [ ]:
```
CalcOutliers(df_train['totals.pageviews']) # Call
```

# Device Browsers

In [ ]:
```
# the top 10 of browsers represent % of total
print("Percentual of Browser usage: ")
print(df_train['device.browser'].value_counts()[:7] ) # printing the top 7 percentage of browsers

# seting the graph size
plt.figure(figsize=(14,6))

# Let explore the browser used by users
sns.countplot(df_train[df_train['device.browser']\
                       .isin(df_train['device.browser']\
                             .value_counts()[:10].index.values)]['device.browser'], palette="hls") # It's a module to count the category's
plt.title("TOP 10 Most Frequent Browsers", fontsize=20) # Adding Title and seting the size
plt.xlabel("Browser Names", fontsize=16) # Adding x label and seting the size
plt.ylabel("Count", fontsize=16) # Adding y label and seting the size
plt.xticks(rotation=45) # Adjust the xticks, rotating the labels

plt.show() #use plt.show to render the graph that we did above
```

### In our top 5 browsers we have more than 94% of total

- TOP 1 - CHROME - 69,08%
- TOP 2 - SAFARI - 20,04%
- TOP 3 - FIREFOX - 3,77%

Nothing new under the sun... Chrome is the most used followed by Safari and firefox.

## What if we cross the Revenue and Browser?

In [ ]:
```
plt.figure(figsize=(13,6)) #figure size

#It's another way to plot our data. using a variable that contains the plot parameters
g1 = sns.boxenplot(x='device.browser', y='totals.transactionRevenue',
                   data=df_train[(df_train['device.browser'].isin((df_train['device.browser'].value_counts()[:10].index.values))) &
                                  df_train['totals.transactionRevenue'] > 0])
g1.set_title('Browsers Name by Transactions Revenue', fontsize=20) # title and fontsize
g1.set_xticklabels(g1.get_xticklabels(),rotation=45) # It's the way to rotate the xticks when we use variable to our graphs
g1.set_xlabel('Device Names', fontsize=18) # Xlabel
g1.set_ylabel('Trans Revenue(log) Dist', fontsize=18) #Ylabel

plt.show()
```

I think that it's very insightful information.

Chrome have highest values in general but the highest value of transactions was did on Firefox.
We can see a "small" but consistent sells in Safari. Also IE and Edge give some results to Google;

## Let's see the Channel Grouping

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

In [ ]:
```
# the top 10 of browsers represent % of total
print("Percentual of Channel Grouping used: ")
print((df_train['channelGrouping'].value_counts()[:5])) # printing the top 7 percentage of browsers

# seting the graph size
plt.figure(figsize=(14,7))

# let explore the browser used by users
sns.countplot(df_train["channelGrouping"], palette="hls") # It's a module to count the category's
plt.title("Channel Grouping Count", fontsize=20) # seting the title size
plt.xlabel("Channel Grouping Name", fontsize=18) # seting the x label size
plt.ylabel("Count", fontsize=18) # seting the y label size

plt.show() #use plt.show to render the graph that we did above
```

The TOP 5 Grouping Channels represents 97% of total values. Respectivelly:

- TOP 1 => Organic Search - 42.99%
- TOP 2 => Social - 24.39%
- TOP 3 => Direct - 15.42%
- TOP 4 => Referral - 11.89%
- TOP 5 => Paid Search - 2.55%

I have a new insight that I will explore furthuer. How wich channel are distributed by browsers?

## Crossing Channel Grouping x Browsers

In [ ]:
```
## I will use the crosstab to explore two categorical values

# At index I will use set my variable that I want analyse and cross by another
crosstab_eda = pd.crosstab(index=df_train['channelGrouping'], normalize=True,
                           # at this line, I am using the isin to select just the top 5 of browsers
                           columns=df_train[df_train['device.browser'].isin(df_train['device.browser']\
                                                                            .value_counts()[:5].index.values)]['device.browser'])
# Ploting the crosstab that we did above
crosstab_eda.plot(kind="bar",    # select the bar to plot the count of categoricals
                  figsize=(14,7), # adjusting the size of graphs
                  stacked=True)   # code to unstack
plt.title("Channel Grouping % for which Browser", fontsize=20) # seting the title size
plt.xlabel("The Channel Grouping Name", fontsize=18) # seting the x label size
plt.ylabel("Count", fontsize=18) # seting the y label size
plt.xticks(rotation=0)
plt.show() # rendering
```

Very cool! Interesting patterns

# Operational System

In [ ]:
```
# the top 5 of browsers represent % of total
print("Percentual of Operational System: ")
print(df_train['device.operatingSystem'].value_counts()[:5]) # printing the top 7 percentage of browsers

# seting the graph size
plt.figure(figsize=(14,7))

# let explore the browser used by users
sns.countplot(df_train["device.operatingSystem"], palette="hls") # It's a module to count the category's
plt.title("Operational System used Count", fontsize=20) # seting the title size
plt.xlabel("Operational System Name", fontsize=16) # seting the x label size
plt.ylabel("OS Count", fontsize=16) # seting the y label size
plt.xticks(rotation=45) # Adjust the xticks, rotating the labels

plt.show() #use plt.show to render the graph that we did above
```

The TOP 5 of Operational System corresponds to 96%.

TOP 1 => Windows - 38.75%
TOP 2 => Macintosh - 28.04%
TOP 3 => Android - 14.15%
TOP 4 => iOS - 11.75%
TOP 5 => Linux - 3.91%

It's very interestign to me. In my country macbook isn't the most common SO. I will investigate further the SO by Country's

## Now let's investigate the most used brower by Operational System

In [ ]:
```
# At index I will use isin to substitute the loop and get just the values with more than 1%
crosstab_eda = pd.crosstab(index=df_train[df_train['device.operatingSystem']\
                                          .isin(df_train['device.operatingSystem']\
                                                .value_counts()[:6].index.values)]['device.operatingSystem'],

                           # at this line, I am using the isin to select just the top 5 of browsers
                           columns=df_train[df_train['device.browser'].isin(df_train['device.browser']\
                                                                            .value_counts()[:5].index.values)]['device.browser'])
# Ploting the crosstab that we did above
crosstab_eda.plot(kind="bar",    # select the bar to plot the count of categoricals
                  figsize=(14,7), # adjusting the size of graphs
                  stacked=True)   # code to unstack
plt.title("Most frequent OS's by Browsers of users", fontsize=22) # adjusting title and fontsize
plt.xlabel("Operational System Name", fontsize=19) # adjusting x label and fontsize
plt.ylabel("Count OS", fontsize=19) # adjusting y label and fontsize
plt.xticks(rotation=0) # Adjust the xticks, rotating the labels

plt.show() # rendering
```

### Cool!

It's visually clear to see that chrome is the most used in all OS, less in iOS, that is a mobile OS.

I will see if we can see a diference between the Revenues of transactions are different

## I will explore the distribuition of transaction Revenue by each OS

In [ ]:
```
(sns.FacetGrid(df_train[(df_train['device.operatingSystem']\
                         .isin(df_train['device.operatingSystem']\
                               .value_counts()[:6].index.values)) & df_train['totals.transactionRevenue'] > 0],
               hue='device.operatingSystem', height=5, aspect=2)
  .map(sns.kdeplot, 'totals.transactionRevenue', shade=True)
 .add_legend()
)
plt.show()
```

Cool, we can have a better understanding of the distribution of Revenue by OS

## Let's investigate the Device Category

In [ ]:
```
# the top 5 of browsers represent % of total
print("Percentual of Operational System: ")
print(round(df_train['device.deviceCategory'].value_counts() / len(df_train['device.deviceCategory']) * 100, 2)) # printing the top 7 percentage of browsers

# seting the graph size
plt.figure(figsize=(14,5))

plt.subplot(1,2,1)
# let explore the browser used by users
sns.countplot(df_train["device.deviceCategory"], palette="hls") # It's a module to count the category's
plt.title("Device Category Count", fontsize=20) # seting the title size
plt.xlabel("Device Category", fontsize=18) # seting the x label size
plt.ylabel("Count", fontsize=16) # seting the y label size
plt.xticks(fontsize=18) # Adjust the xticks, rotating the labels

plt.subplot(1,2,2)
sns.boxenplot(x="device.deviceCategory", y = 'totals.transactionRevenue',
              data=df_train[df_train['totals.transactionRevenue'] > 0], palette="hls") # It's a module to count the category's
plt.title("Device Category Revenue Distribuition", fontsize=20) # seting the title size
plt.xlabel("Device Category", fontsize=18) # seting the x label size
plt.ylabel("Revenue(Log)", fontsize=16) # seting the y label size
plt.xticks(fontsize=18) # Adjust the xticks, rotating the labels

plt.subplots_adjust(hspace = 0.9, wspace = 0.5)

plt.show() #use plt.show to render the graph that we did above
```

In percentual, we can see that :

- desktop represents 73.5%
- mobile represents 23.12%
- tablet represents 3.38%

I thought that the Revenue is almost all did by desktops. Let's explore it further.

## Let's see the difference distribution between Devices

In [ ]:
```
(sns.FacetGrid(df_train[df_train['totals.transactionRevenue'] > 0],
               hue='device.deviceCategory', height=5, aspect=2)
  .map(sns.kdeplot, 'totals.transactionRevenue', shade=True)
 .add_legend()
)
plt.show()
```

We have We can see the distribuition of

## Now, lets investigate the Device Category by Browsers

In [ ]:

```python
# At index I will use isin to substitute the loop and get just the values with more than 1%
crosstab_eda = pd.crosstab(index=df_train['device.deviceCategory'], # at this line, I am using the isin to select just the top 5 of browsers
                           columns=df_train[df_train['device.operatingSystem']\
                                            .isin(df_train['device.operatingSystem']\
                                                  .value_counts()[:6].index.values)]['device.operatingSystem'])
# Ploting the crosstab that we did above
crosstab_eda.plot(kind="bar",    # select the bar to plot the count of categoricals
                 figsize=(14,7), # adjusting the size of graphs
                 stacked=True)   # code to unstack
plt.title("Most frequent OS's by Device Categorys of users", fontsize=22) # adjusting title and fontsize
plt.xlabel("Device Name", fontsize=19)                # adjusting x label and fontsize
plt.ylabel("Count Device x OS", fontsize=19)                     # adjusting y label and fontsize
plt.xticks(rotation=0)                                         # Adjust the xticks, rotating the labels


plt.show() # rendering
```

Very interesting values.

## SubContinent

In [ ]:

```python
# the top 8 of browsers represent % of total
print("Description of SubContinent count: ")
print(df_train['geoNetwork.subContinent'].value_counts()[:8]) # printing the top 7 percentage of browsers

# seting the graph size
plt.figure(figsize=(16,7))

# let explore the browser used by users
sns.countplot(df_train[df_train['geoNetwork.subContinent']\
                       .isin(df_train['geoNetwork.subContinent']\
                             .value_counts()[:15].index.values)]['geoNetwork.subContinent'], palette="hls") # It's a module to count the category's
plt.title("TOP 15 most frequent SubContinents", fontsize=20) # seting the title size
plt.xlabel("subContinent Names", fontsize=18) # seting the x label size
plt.ylabel("SubContinent Count", fontsize=18) # seting the y label size
plt.xticks(rotation=45) # Adjust the xticks, rotating the labels

plt.show() #use plt.show to render the graph that we did above
```

WoW, We have a very high number of users from North America.

TOP 5 regions are equivalent of almost 70% +- of total

TOP 1 => Northern America - 44.18%
TOP 2 => Southeast Asia - 8.29%
TOP 3 => Northern Europe - 6.73%
TOP 4 => Southern Asia - 6.33%
TOP 5 => Western Europe - 6.23%

### Let's cross the SubContinent by Browser

In [ ]:

```python
## I will use the crosstab to explore two categorical values

# At index I will use isin to substitute the loop and get just the values with more than 1%
crosstab_eda = pd.crosstab(index=df_train[df_train['geoNetwork.subContinent']\
                                          .isin(df_train['geoNetwork.subContinent']\
                                                .value_counts()[:10].index.values)]['geoNetwork.subContinent'],

                           # at this line, I am using the isin to select just the top 5 of browsers
                           columns=df_train[df_train['device.browser'].isin(df_train['device.browser']\
                                                                            .value_counts()[:5].index.values)]['device.browser'])
# Ploting the crosstab that we did above
crosstab_eda.plot(kind="bar",    # select the bar to plot the count of categoricals
                 figsize=(16,7), # adjusting the size of graphs
                 stacked=True)   # code to unstack
plt.title("TOP 10 Most frequent Subcontinents by Browsers used", fontsize=22) # adjusting title and fontsize
plt.xlabel("Subcontinent Name", fontsize=19) # adjusting x label and fontsize
plt.ylabel("Count Subcontinent", fontsize=19) # adjusting y label and fontsize
plt.xticks(rotation=45) # Adjust the xticks, rotating the labels
plt.legend(loc=1, prop={'size': 12}) # to

plt.show() # rendering
```

Nice, this graph is very insightful. The North America have a low ratio of Safari x Chrome... I thought that it was the contrary

Firefox have a relative high presence in North America too.

In [ ]:

```python
print('train date:', min(df_train['date']), 'to', max(df_train['date']))
```

In [ ]:

```python
year = df_train['_year'].value_counts()         # counting the Year with value counts
month = df_train['_month'].value_counts()      # coutning months
weeday = df_train['_weekday'].value_counts()    # Couting weekday
day = df_train['_day'].value_counts()              # counting Day
date = df_train['date'].value_counts()           # Counting date
```

## INTERACTIVE DATE FEATURES

### First I will explore revenue and number of visits by day

In [ ]:

```python
# I saw and take a lot of inspiration to this interactive plots in kernel:
# https://www.kaggle.com/jsaguiar/complete-exploratory-analysis-all-columns
# I learned a lot in this kernel and I will implement and adapted some ideas

#seting some static color options
color_op = ['#5527A0', '#BB93D7', '#834CF7', '#6C941E', '#93EAEA', '#7425FF', '#F2098A', '#7E87AC',
            '#EBE36F', '#7FD394', '#49C35D', '#3058EE', '#44FDCF', '#A38F85', '#C4CEE0', '#B63A05',
            '#4856BF', '#F0DB1B', '#9FDBD9', '#B123AC']

# Visits by time train

# couting all entries by date to get number of visits by each date
dates_temp = df_train['date'].value_counts().to_frame().reset_index().sort_values('index')
# renaming the columns to apropriate names
dates_temp = dates_temp.rename(columns = {"date" : "visits"}).rename(columns = {"index" : "date"})

# creating the first trace with the necessary parameters
trace = go.Scatter(x=dates_temp.date.astype(str), y=dates_temp.visits,
                   opacity = 0.8, line = dict(color = color_op[3]), name= 'Visits by day')

# Below we will get the total values by Transaction Revenue Log by date
dates_temp_sum = df_train.groupby('date')['totals.transactionRevenue'].sum().to_frame().reset_index()

# using the new dates_temp_sum we will create the second trace
trace1 = go.Scatter(x=dates_temp_sum.date.astype(str), line = dict(color = color_op[1]), name="RevenueLog by day",
                        y=dates_temp_sum['totals.transactionRevenue'], opacity = 0.8)

# Getting the total values by Transactions by each date
dates_temp_count = df_train[df_train['totals.transactionRevenue'] > 0].groupby('date')['totals.transactionRevenue'].count().to_frame().reset_index()

# using the new dates_temp_count we will create the third trace
trace2 = go.Scatter(x=dates_temp_count.date.astype(str), line = dict(color = color_op[5]), name="Sellings by day",
                        y=dates_temp_count['totals.transactionRevenue'], opacity = 0.8)

#creating the layout the will allow us to give an title and
# give us some interesting options to handle with the outputs of graphs
layout = dict(
    title= "Informations by Date",
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1, label='1m', step='month', stepmode='backward'),
                dict(count=3, label='3m', step='month', stepmode='backward'),
                dict(count=6, label='6m', step='month', stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(visible = True),
        type='date'
    )
)

# creating figure with the both traces and layout
fig = dict(data= [trace, trace1, trace2], layout=layout)

#rendering the graphs
iplot(fig) #it's an equivalent to plt.show()
```

#### Creating an Sofistcated interactive graphics to better understanding of date features

To see the code click in "code".

### SELECT THE OPTION:

In [ ]:

```python
# Setting the first trace
trace1 = go.Histogram(x=df_train["_year"],
                      name='Year Count')

# Setting the second trace
trace2 = go.Histogram(x=df_train["_month"],
                name='Month Count')

# Setting the third trace
trace3 = go.Bar(y=day.values,
                x=day.index.values,
                name='Day Count')

# Setting the fourth trace
trace4 = go.Bar(y=weeday.values,
                x=weeday.index.values,
                name='Weekday Count')

# puting all traces in the same "array of graphics" to we render it below
data = [trace1, trace2, trace4, trace3]

#Creating the options to be posible we use in our
updatemenus = list([
    dict(active=-1,
         x=-0.15,
         buttons=list([
             dict(
                 label = 'Years Count',
                 method = 'update',
                 args = [{'visible': [True, False, False, False,False]},
                         {'title': 'Count of Year'}]),
             dict(
                 label = 'Months Count',
                 method = 'update',
                 args = [{'visible': [False, True, False, False,False]},
                         {'title': 'Count of Months'}]),
             dict(
                 label = 'WeekDays Count',
                 method = 'update',
                 args = [{'visible': [False, False, True, False, False]},
                         {'title': 'Count of WeekDays'}]),
            dict(
                label = 'Days Count ',
                method = 'update',
                args = [{'visible': [False, False, False, True,False]},
                        {'title': 'Count of Day'}]) ])
    )
])


layout = dict(title='The percentual Distribuitions of Date Features (Select from Dropdown)',
              showlegend=False,
              updatemenus=updatemenus,
#               xaxis = dict(
#                   type="category"
#                       ),
              barmode="group"
             )
fig = dict(data=data, layout=layout)
print("SELECT BELOW: ")
iplot(fig)
```

**** How can I set order to my year, months and days? ****

### Very Cool graphs.

WE can see that the number of access are clearly downing by the through the time.

- The months with highest accesses are October and November.
- On the Weekend the trafic is lower than other days.
- The 5 days with highest number of accesses is 1 and 5
- Considering the full count of dates, we can see that the days with highest accesses are almost all in november/2016

#### Let's investigate the VisitHour and weekday to see if we can find some interesting patterns

In [ ]:

```python
date_sales = ['_visitHour', '_weekday'] #seting the desired

cm = sns.light_palette("green", as_cmap=True)
pd.crosstab(df_train[date_sales[0]], df_train[date_sales[1]],
            values=df_train["totals.transactionRevenue"], aggfunc=[np.sum]).style.background_gradient(cmap = cm)

# tab.columns.levels[1] = ["Sun", "Mon", "Thu", "wed", "Thi","Fri","Sat"]
```

Very interesting, we can see that from 17 to 20 hour we have the highest numbers of

## I will use a interesting graphic called Squarify

- I will apply it in feature Country to discovery where the user access the store

In [ ]:

```python
number_of_colors = 20 # total number of different collors that we will use

# Here I will generate a bunch of hexadecimal colors
color = ["#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)])
             for i in range(number_of_colors)]
```

### Exploring Countrys

In [ ]:

```python
country_tree = df_train["geoNetwork.country"].value_counts() #counting the values of Country

print("Description most frequent countrys: ")
print(country_tree[:15]) #printing the 15 top most

country_tree = round((df_train["geoNetwork.country"].value_counts()[:30] \
                       / len(df_train['geoNetwork.country']) * 100),2)

plt.figure(figsize=(14,5))
g = squarify.plot(sizes=country_tree.values, label=country_tree.index,
                  value=country_tree.values,
                  alpha=.4, color=color)
g.set_title("'TOP 30 Countrys - % size of total",fontsize=20)
g.set_axis_off()
plt.show()
```

USA have a very highest value than another countrys.

Below I will take a look on cities and find for the highest revenues from them

### Now, I will look on City feature and see the principal cities in the dataset

In [ ]:

```python
df_train.loc[df_train["geoNetwork.city"] == "not available in demo dataset", 'geoNetwork.city'] = np.nan
```

In [ ]:

```python
city_tree = df_train["geoNetwork.city"].value_counts() #counting

print("Description most frequent Citys: " )
print(city_tree[:15])

city_tree = round((city_tree[:30] / len(df_train['geoNetwork.city']) * 100),2)

plt.figure(figsize=(14,5))
g = squarify.plot(sizes=city_tree.values, label=city_tree.index,
                  value=city_tree.values,
                  alpha=.4, color=color)
g.set_title("'TOP 30 Citys - % size of total",fontsize=20)
g.set_axis_off()
plt.show()
```

Nicelly distributed clients that accessed the store. (non set) have 3.81% of total, so I dont will consider in top five, but it was the top 2 most frequent.

The top 5 are:

- Montain View
- New York
- San Francisco
- Sunnyvale
- London

And in terms of money, how the Countrys and Cities are ?

---

#### Creating a function with plotly to better investigate the dataset

- Click in "code" to see the commented code

In [ ]:

```python
def PieChart(df_colum, title, limit=15):
    """
    This function helps to investigate the proportion of visits and total of transction revenue
    by each category
    """

    count_trace = df_train[df_colum].value_counts()[:limit].to_frame().reset_index()
    rev_trace = df_train.groupby(df_colum)["totals.transactionRevenue"].sum().nlargest(10).to_frame().reset_index()

    trace1 = go.Pie(labels=count_trace['index'], values=count_trace[df_colum], name= "% Acesses", hole= .5,
                    hoverinfo="label+percent+name", showlegend=True,domain= {'x': [0, .48]},
                    marker=dict(colors=color))

    trace2 = go.Pie(labels=rev_trace[df_colum],
                    values=rev_trace['totals.transactionRevenue'], name="% Revenue", hole= .5,
                    hoverinfo="label+percent+name", showlegend=False, domain= {'x': [.52, 1]})

    layout = dict(title= title, height=450, font=dict(size=15),
                  annotations = [
                      dict(
                          x=.25, y=.5,
                          text='Visits',
                          showarrow=False,
                          font=dict(size=20)
                      ),
                      dict(
                          x=.80, y=.5,
                          text='Revenue',
                          showarrow=False,
                          font=dict(size=20)
                      )
        ])

    fig = dict(data=[trace1, trace2], layout=layout)
    iplot(fig)
```

### Device Category feature

In [ ]:

```python
PieChart("device.deviceCategory", "Device Category")
```

### I will apply the Prie Chart in Country's again

In [ ]:

```python
# call the function
PieChart("geoNetwork.city", "Top Cities by Accesses and Revenue", limit=12)
```

- New York is responsible by 14% of visits and 31% of revenues.
- Montain view have 19% in visists but just 16% of revenues
- Chicago have just 3.5% of visits but have a high significance in revenues

### Seeing again Channel Grouping more specified

In [ ]:

```python
PieChart("channelGrouping", "Channel Grouping Visits and Revenues")
```

It's interesting to note that Referral have a less number of Visits but is responsible for almost 40% of revenues**

#### Months in pizza graph

### Let's see the NetWork Domain

- I will plot visits and revenues by each category, including the non-set and unknown accesses and revenues

In [ ]:

```python
PieChart('geoNetwork.networkDomain', "Network Domain")
```

Wow, another very cool information.

- (not set) domain have almost 50% of total visits and 62% of Revenues.
- Unknown is responsible by 28% of visits but just 2.70% of Revenues
- comcast.net have 5.5% of visits and 7.4% Revenues.

Let's take a look on Mobile and Browser proportions

In [ ]:

```python
PieChart("device.deviceCategory", "Device Category")
```

The absolutelly high part of revenues are from Desktop Devices

### Trafic Source Medium

In [ ]:

```python
PieChart("trafficSource.medium", "Trafic Source - Medium")
```

- Organic have highest number of visits but is the third in revenues
- Referral have almost 40% in both Visits and Revenues
- The none category have almost 16% of visits but almost 40% of revenues

Now I will take a look on trafficSource section, the Source to access the store

In [ ]:

```python
PieChart('trafficSource.source', "Visits and Revenue by TOP Sources", limit=8)
```

We have a high number of visits from youtube but 0 sales.
the mall.googleplex is have a low number of access but have the highest value in revenues

## I will continue this notebook! Votes up the kernel and stay tuned to next updates

In [ ]:

```python
df_train.corr()['totals.transactionRevenue']
```

Seeing the crosstab with heatmap

In [ ]:

```python
country_repayment = ['channelGrouping', '_weekday'] #seting the desired

cm = sns.light_palette("green", as_cmap=True)
pd.crosstab(df_train[country_repayment[0]], df_train[country_repayment[1]],
            values=df_train["totals.transactionRevenue"], aggfunc=[np.sum]).style.background_gradient(cmap = cm)

# tab.columns.levels[1] = ["Sun", "Mon", "Thu", "wed", "Thi","Fri","Sat"]
```

### Geolocation plot to visually understand the data

In [ ]:

```python
# Counting total visits by countrys
countMaps = pd.DataFrame(df_train['geoNetwork.country'].value_counts()).reset_index()
countMaps.columns=['country', 'counts'] #renaming columns
countMaps = countMaps.reset_index().drop('index', axis=1) #reseting index and droping the column

data = [ dict(
        type = 'choropleth',
        locations = countMaps['country'],
        locationmode = 'country names',
        z = countMaps['counts'],
        text = countMaps['country'],
        autocolorscale = False,
        marker = dict(
            line = dict (
                color = 'rgb(180,180,180)',
                width = 0.5
            ) ),
        colorbar = dict(
            autotick = False,
            tickprefix = '',
            title = 'Number of Visits'),
      ) ]

layout = dict(
    title = 'Couting Visits Per Country',
    geo = dict(
        showframe = False,
        showcoastlines = True,
        projection = dict(
            type = 'Mercator'
        )
    )
)

figure = dict( data=data, layout=layout )
iplot(figure, validate=False, filename='map-countrys-count')
```

### Total Revenues by Country

In [ ]:

```python
# I will crete a variable of Revenues by country sum
sumRevMaps = df_train[df_train['totals.transactionRevenue'] > 0].groupby("geoNetwork.country")["totals.transactionRevenue"].count().to_frame().reset_index()
sumRevMaps.columns = ["country", "count_sales"] # renaming columns
sumRevMaps = sumRevMaps.reset_index().drop('index', axis=1) #reseting index and drop index column

data = [ dict(
        type = 'choropleth',
        locations = sumRevMaps['country'],
        locationmode = 'country names',
        z = sumRevMaps['count_sales'],
        text = sumRevMaps['country'],
        autocolorscale = False,
        marker = dict(
            line = dict (
                color = 'rgb(180,180,180)',
                width = 0.5
            ) ),
        colorbar = dict(
            autotick = False,
            tickprefix = '',
            title = 'Count of Sales'),
      ) ]

layout = dict(
    title = 'Total Sales by Country',
    geo = dict(
        showframe = False,
        showcoastlines = True,
        projection = dict(
            type = 'Mercator'
        )
    )
)

figure = dict( data=data, layout=layout )

iplot(figure, validate=False, filename='map-countrys-total')
```

#### Some tests that I am doing to try find interesting feature engineering approaches

In [ ]:

```python
df_train['month_unique_user_count'] = df_train.groupby('_month')['fullVisitorId'].transform('nunique')
df_train['day_unique_user_count'] = df_train.groupby('_day')['fullVisitorId'].transform('nunique')
df_train['weekday_unique_user_count'] = df_train.groupby('_weekday')['fullVisitorId'].transform('nunique')


df_train['traf_sourc_browser_count'] = df_train.groupby(['trafficSource.medium', 'device.browser'])['totals.pageviews'].transform('nunique')
df_train['Id_browser_pageviews_sumprod'] = df_train.groupby(['fullVisitorId', 'device.browser'])['totals.pageviews'].transform('cumprod')
df_train['Id_browser_hits_sumprod'] = df_train.groupby(['fullVisitorId', 'device.browser'])['totals.hits'].transform('cumprod')
df_train['Id_browser_hits_sumprod'] = df_train.groupby(['fullVisitorId', 'device.browser'])['totals.hits'].transform('cumprod')
df_train['Id_browser_hits_sumprod_mob'] = df_train.groupby(['fullVisitorId', 'device.browser', 'device.isMobile'])['totals.hits'].transform('sum')

df_train['Id_networkDomain_hits'] = df_train.groupby(['fullVisitorId', 'geoNetwork.networkDomain'])['totals.hits'].transform('var')
df_train['Id_networkDomain_country_hits'] = df_train.groupby(['fullVisitorId', 'geoNetwork.networkDomain', 'geoNetwork.country'])['totals.hits'].transform('unique')
```

In [ ]:

```python
df_train[["totals.transactionRevenue", 'Id_browser_hits_sumprod', 'Id_networkDomain_hits','Id_networkDomain_country_hits', 'Id_browser_hits_sumprod_mob']].corr()
```

### Preprocessing the fulldataset and creating new features

In [ ]:

```python
aggs = {
    'date': ['min', 'max'],
    'totals.hits': ['sum', 'min', 'max', 'mean', 'median'],
    'totals.pageviews': ['sum', 'min', 'max', 'mean', 'median'],
    'totals.bounces': ['sum', 'mean', 'median'],
    'totals.newVisits': ['sum', 'mean', 'median']
}

# Previous applications categorical features
cat_aggregations = {}

for cat in dummy_feaures:
    cat_aggregations[cat] = ['min', 'max', 'mean']

prev_agg = df_train.groupby('fullVisitorId').agg({**aggs})

prev_agg.columns = pd.Index(['Agg_' + e[0] + "_" + e[1].upper() for e in prev_agg.columns.tolist()])
```

In [ ]:

```python
prev_agg
```

In [ ]:

```python
new_columns = [
        k + '_' + agg for k in aggs.keys() for agg in aggs[k]
    ]
new_columns
```

In [ ]:

```python
dummy_feaures
```

In [ ]:

```python
### Testing some grouping approaches
```

In [ ]:

```python
df_train['cumcount'] = df_train.groupby('fullVisitorId').cumcount() + 1
```

Some tests to feature engineering

In [ ]:

```python
aggs = {
    'date': ['min', 'max'],
    'totals.transactionRevenue': ['sum', 'size'],
    'totals.hits': ['sum', 'min', 'max', 'count', 'median'],
    'totals.pageviews': ['sum', 'min', 'max', 'mean', 'median'],
    'totals.bounces': ['sum', 'mean', 'median'],
    'totals.newVisits': ['sum', 'mean', 'median']
}

# Previous applications categorical features
cat_aggregations = {}

for cat in dummy_feaures:
    cat_aggregations[cat] = ['min', 'max', 'mean']

prev_agg = df_train.groupby('fullVisitorId').agg({**aggs})

prev_agg.head()
```

## I will continue working on this kernel, stay tuned

** Please, if you liked this kernel don't forget to votes up and give your feedback **

In [ ]:

```python
prev_agg.columns = ["_".join(x) for x in prev_agg.columns.ravel()]
```

In [ ]:

```python
prev_agg.head()
```

In [ ]:

```python

```