Automating web analytics through Python

Ruthger Righart

Email: rrighart@googlemail.com

Website: https://rrighart.github.io (https://rrighart.github.io)

Python modules

In the current blog **Python 2** was used. Please note that the code may be slightly different for Python 3. The following Python modules were used:

In [1]:
```
import pandas as pd
import numpy as np
from google2pandas import *
import matplotlib as mpl
import matplotlib.pyplot as plt
import sys
from geonamescache import GeonamesCache
from geonamescache.mappers import country
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
from mpl_toolkits.basemap import Basemap
from scipy import stats
import matplotlib.patches as mpatches
import plotly
import plotly.plotly as py
from IPython.display import Image
from plotly import tools
from plotly.graph_objs import *
```

1. Introduction

Web analytics is a fascinating domain and important target of data science. Seeing where people come from (geographical information), what they do (behavioral analyses), how they visit (device: mobile, tablet or workstation), and when they visit your website (time-related info, frequency etc), are all different metrics of webtraffic that have potential business value. Google Analytics is one of the available open-source tools that is highly used and well-documented.

The current blog deals with the case how to implement web analytics in Python. I am enthusiastic about the options that are available inside Google Analytics. Google Analytics has a rich variety of metrics and dimensions available. It has a good visualization and an intuitive Graphic User Interface (GUI). However, in certain situations it makes sense to automate webanalytics and add advanced statistics and visualizations. In the current blog, I will show how to do that using Python.

As an example, I will present the traffic analyses of my own website, for one of my blogs (https://rrighart.github.io/Webscraping/ (https://rrighart.github.io/Webscraping/)). Note however that many of the implementation steps are quite similar for conventional (non-GitHub) websites. So please do stay here and do not despair if GitHub is not your cup of tea.

2. The end in mind

As the options are quite extensive, it is best to start with the end in mind. In other words, for what purpose do we use webtraffic analyses?[1].

1. What is the effect of a campaign on webtraffic? Concretely, publishing a link (i.e., my blog *"Webscraping and beyond"*) on a reputated site for colleague developers, does this have a substantial impact on the number of visitors? To reach this goal, much like an experiment, I performed a *single* intervention of publishing a link at http://www.datatau.com (http://www.datatau.com) .
2. Where do the visitors come from, is this a large range of countries, from different continents, or is it rather limited to one or a few countries?
3. What kind of devices do my visitors use? (mobile, desktop, or tablet).

To realise these goals, what follows are the analytical steps needed from data acquisition to visualization. If at this moment you are not able to run Google Analytics for your own website but want to nevertheless reproduce the data analyses in Python (starting below at section 8), I advice to load the DataFrames (df1, df2, df3, df3a, df3b) from my GitHub site using the following code (change the "df"-filename accordingly):

In [2]:
```
url = 'https://raw.githubusercontent.com/RRighart/GA/master/df1.csv'
df1 = pd.read_csv(url, parse_dates=True, delimiter=",", decimal=",")
```

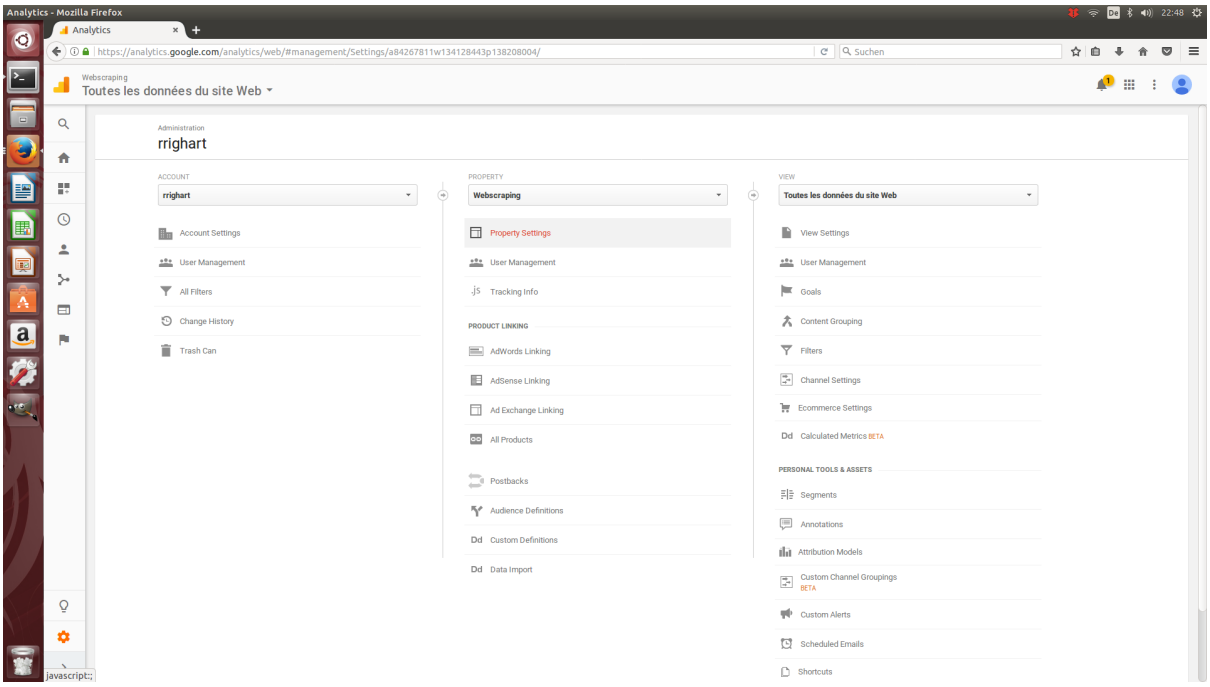
3. Add a website to your Google Analytics account

You need to first subscribe to Google Analytics and add your website[2]:

- Select **Admin**
- In the dropdown menu **Property**, select **Create new property**. You need to give the name of your website. The URL has the following format for GitHub sites: https://yourname.github.io/projectname/ (https://yourname.github.io/projectname/) (in my case, it is for example https://rrighart.github.io/Webscraping/ (https://rrighart.github.io/Webscraping/)).
- Do not forget to set the reporting timezone right. This is very important if you want to research the time of the day that your customers come visit your website.
- After confirming the other steps, you'll receive a Universal Analytics (UA) tracking -ID, which has the following format, UA-xxxxxxxx-x, where the x are numbers.

In [3]:
```
Image("Fig1.png")
```

Out[3]:



4. Tracking-ID and code

If you need to find back the tracking-ID later, the code can be found at **Tracking Info** and then **Tracking Code**. Under the header **Website Tracking**, Google Analytics will give a script that needs to be pasted in your website. It is essential to set the code right to prevent for example half or double counting[3].

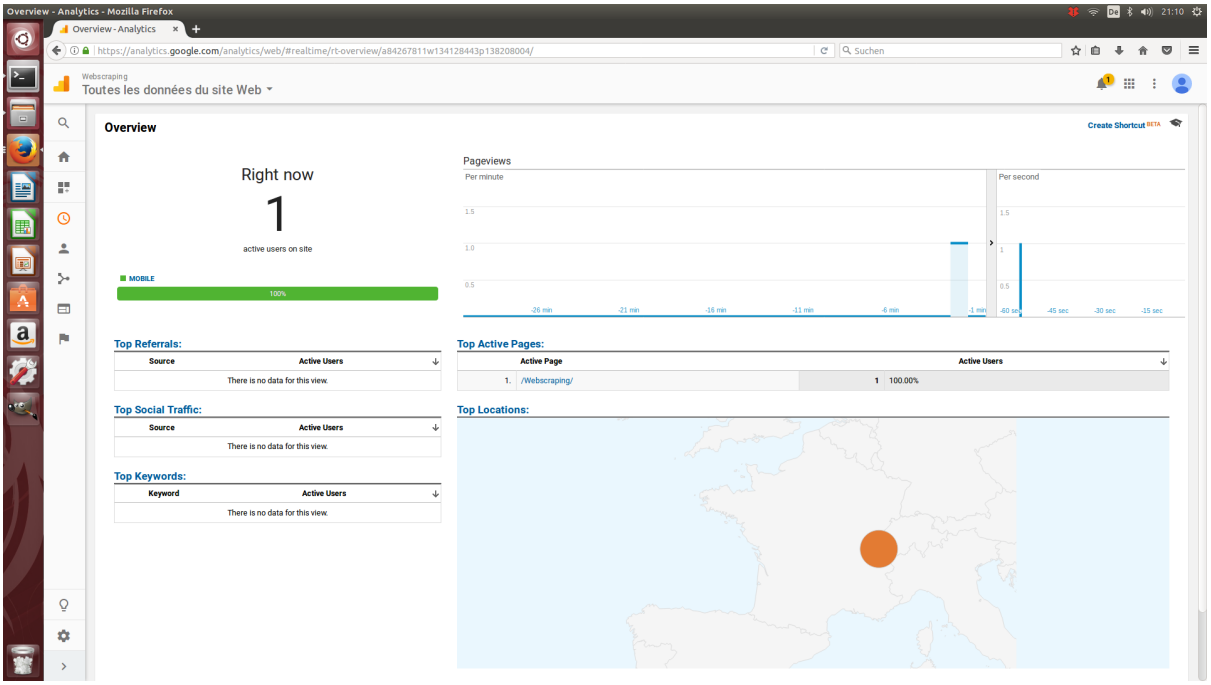
5. Check the connection

Now you have the tracking code pasted in your website, Google Analytics is able to collect traffic data. The „official“ way to inspect if there is a connection in Google Analytics is to select **Tracking Info**, **Tracking Code**, and under status, push the button **Send test traffic**. This will open up your website.

However, a more real life way to do this is to visit your website yourself, using for example your mobile phone. In Google Analytics select **Home**, **Real-time**, and **Overview**. If you just visited your website of interest, you should see under **pageviews** that there is *"Right now 1 active users on site"* (of course this could be >1 if at the same moment there were other visitors). Additionally, you may want to check the geographical map and see if *your place* is highlighted. If you leave your website, the active users section should return to zero (or go one down). If this works, you are ready to start webtraffic analyses as soon as your first visitors drop in.

In [4]:
```
Image("Fig2.png")
```

Out[4]:



6. Query Explorer

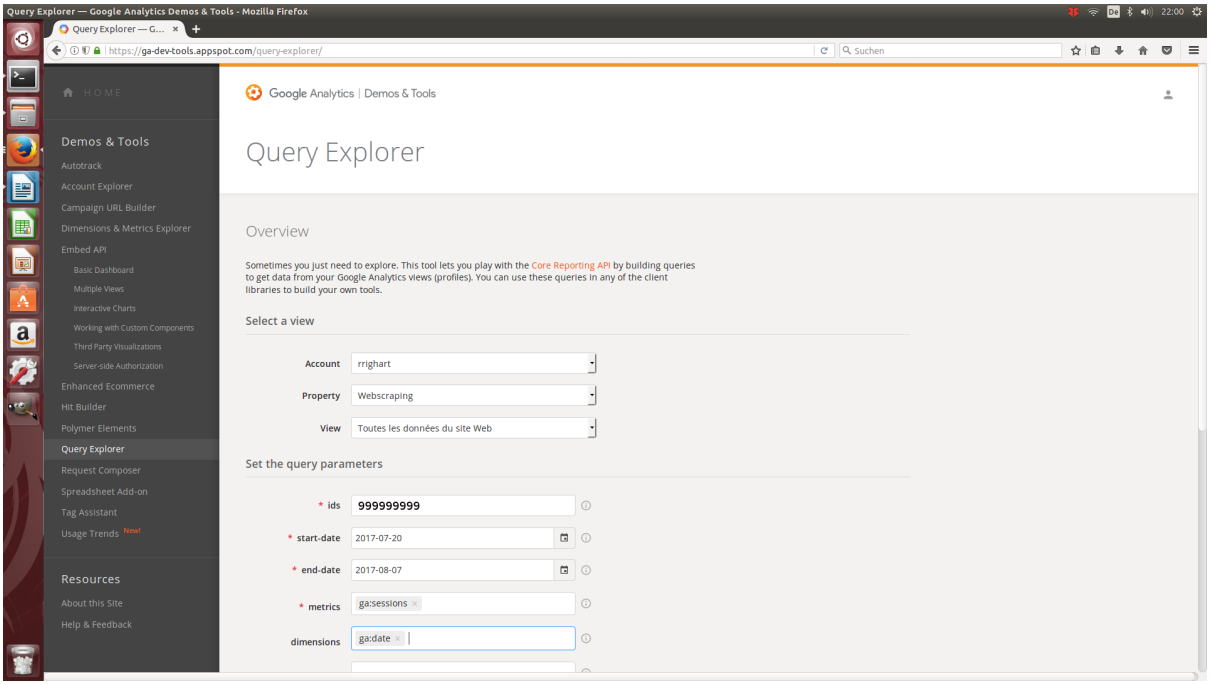
So how to start webtraffic analyses? One option is to visualize traffic in Google Analytics itself. Another option is Query Explorer. Query Explorer is a GUI tool that gives a very quick impression of your data, combining different metrics and dimensions at once. It is also very helpful for preparing the Python code needed for data extraction (more about this later). Follow the next steps:

- Log-in with your Google account at: https://ga-dev-tools.appspot.com/query-explorer/ (https://ga-dev-tools.appspot.com/query-explorer/)
- Select under **property** the webpage that you want to check (in my case „Webscraping“).
- Select **view** to choose between extracting all data, desktop or mobile.
- Select **ids**: this is the „ga:“ code that corresponds with your property.
- Fill-in a **start-date**. Here we select '2017-07-20' (this is one day before I started campaigning at www.datatau.com).
- Fill-in **End-date**: '2017-08-07'.
- **Metrics**: select 'ga:sessions'.
- **Dimensions**: select 'ga:date'.

Note that *number of sessions* is different from *number of visitors*. The difference is that the same visitors may return several times at the same website, resulting in a higher number of sessions. For the time being, leave all the other fields empty. When you hit the button **Run Query** this should return a spreadsheet with number of sessions, for each day in your time-window.

In [5]:
```
Image("Fig3.png")
```

Out[5]:



7. Get your data in Python

A major advantage of using Python is that you can automate data extraction, customize settings and build your own platform, statistics, predictive analytics, and visualizations, if you desire all in a single script. Regarding visualizations, it would be possible to build for example dynamic geographic maps in Python, showing how the flux of visitors changed locally and globally from day-to-day, week-to-week.

Google2pandas[4] is a tool that transfers data from Google Analytics to Python into a Pandas DataFrame. From there you could proceed further making for example statistics and visualizations. The most important steps to enable this are:

- Getting permission from Google Analytics API[2]
- Install **Google2pandas** and **Pandas**.
- Copy the following code in Jupyter notebook. Run it and verify if you obtained the right DataFrame. This should give data that are identical to those in Query Explorer or the data that can be displayed in Google Analytics. So we started collecting the data at July 20, and ended at August 7. There are by the way a number of handy options to select **start_date** and **end_date**, such as '7daysAgo' (you can change the *number* to your likings, for ex. '10daysAgo') or 'today'. These options come in very useful if you want to regularly extract and analyze a same time period. For example, if you want to make a report every thursday morning going one week back ("7daysAgo"), you could in principal run everytime the same script without changing anything.

In [6]:
```
df1 = []

conn = GoogleAnalyticsQuery(secrets='/your-directory/ga-creds/client_secret.json', token_file_name='/your-directory/ga-creds/analytics.dat')

query = {
'ids' : '999999999',
'metrics' : 'sessions',
'dimensions' : 'date',
'start_date' : '2017-07-20',
'end_date' : '2017-08-07'
}

df1, metadata = conn.execute_query(**query)
print(df1)
```

	date	sessions
0	20170720	4
1	20170721	147
2	20170722	125
3	20170723	77
4	20170724	104
5	20170725	57
6	20170726	63
7	20170727	326
8	20170728	277
9	20170729	93
10	20170730	59
11	20170731	96
12	20170801	118
13	20170802	67
14	20170803	34
15	20170804	28
16	20170805	16
17	20170806	20
18	20170807	22

8. Plotting number of sessions as a function of date

To answer the first question -- what is the effect of a campaign on the webtraffic? -- we will analyze if campaigning had a sizeable effect on the number of sessions. First, to improve readability of the resulting plot, we will modify the date string that will be in the x-axis. Therefore, we will remove the year part, and we will reverse the order of day and month[5].

In [7]:
```
df1.date = df1.date.replace({'2017':''}, regex=True)
```

In [8]:
```
df1.date = df1.date.map(lambda x: str(x)[2:]) + '-' + df1.date.map(lambda x: str(x)[:2])
```

In [9]:
```
df1.head(5)
```

Out[9]:

	date	sessions
0	20-07	4
1	21-07	147
2	22-07	125
3	23-07	77
4	24-07	104

Checking the DataFrame **df1** we can see that the date column now has less bits of information. Before plotting the number of sessions, let us view some summary statistics. The number of sessions was 91 on average in the inspected period (with a max. of 326).

In [10]:
```
df1['sessions'].describe()
```

Out[10]:
```
count     19.000000
mean      91.210526
std       84.652610
min        4.000000
25%       31.000000
50%       67.000000
75%      111.000000
max      326.000000
Name: sessions, dtype: float64
```

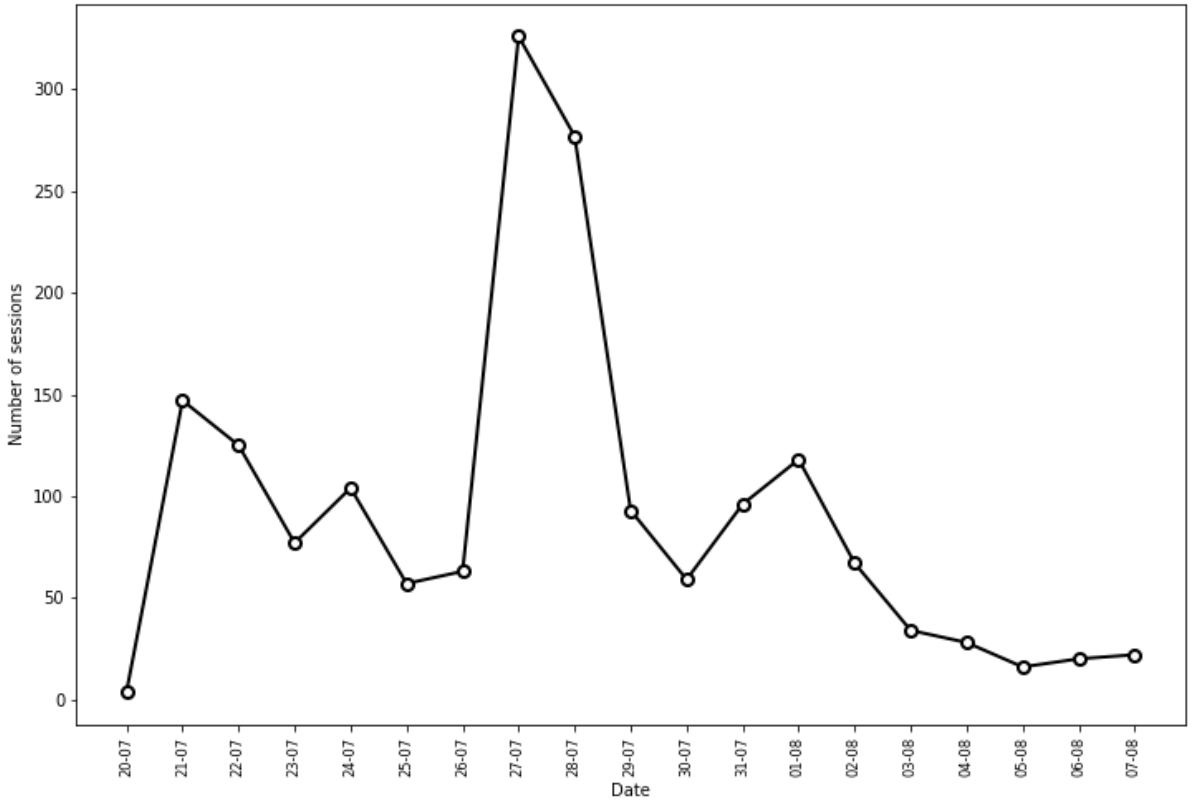
Total number of sessions during the selected time window was 1733.

In [11]:
```
sum(df1.sessions)
```

Out[11]:
```
1733
```

Now we are going to plot the number of sessions (*y*-axis) as a function of date (*x*-axis). Remind that a link to the blog was published at 21-07-2017. Next to the observation that the number of visitors increased substantially after the publication date, there is an additional boost at July 27. I do not have a definite explanation for this second boost. One admittedly speculative explanation is that the blog had in the meanwhile received several "likes" at the site *datatau*, and this in turn may have attracted other visitors. That the number of sessions is decreasing after a certain time is explained by the fact that the link is slowly falling off the *datatau* main page, as new blogs are dropping in. This means that some time after publication people are less likely to see and visit it.

In [12]:
```
f1 = plt.figure(figsize=(12,8))
plt.plot(df1.sessions, '-ko', lw = 2, markerfacecolor = 'white', markersize = 7, markeredgewidth = 2)
plt.xticks(range(len(df1.sessions)), df1.date, size='small', rotation='vertical')
plt.xlabel('Date')
plt.ylabel('Number of sessions')
plt.show()
```



9. Geographic mapping

To investigate the second question -- where do visitors come from? -- a choropleth map can be used. In this case, a world map is used that displays the number of visitors per country, using different color scales. For this purpose, we make a new DataFrame **df2** that sorts the countries on number of sessions.

In [13]:
```
df2 = []

conn = GoogleAnalyticsQuery(secrets='/your-directory/ga-creds/client_secret.json', token_file_name='/your-directory/ga-creds/analytics.dat')

query = {\
'ids' : '999999999',
'metrics' : 'sessions',
'dimensions' : 'country',
'sort' : '-sessions',
'start_date' : '2017-07-20',
'end_date' : '2017-08-07'
}

df2, metadata = conn.execute_query(**query)
```

The top 20 countries are the following:

In [14]:
```
df2.head(20)
```

Out[14]:

	country	sessions
0	United States	589
1	India	95
2	Germany	91
3	United Kingdom	77
4	Australia	54
5	France	50
6	Canada	47
7	Brazil	44
8	Poland	44
9	South Korea	40
10	Spain	40
11	Russia	39
12	China	32
13	Netherlands	32
14	Vietnam	29
15	Italy	28
16	Ukraine	28
17	Hungary	22
18	Singapore	20
19	Japan	19

It turns out that visitors come from 80 countries:

In [15]:
```
len(df2)
```

Out[15]: 80

We are now going to use the choroplethmap. This is a bit of code and an excellent blog going in more detail about this method can be found elsewhere[6].

In [16]:
```
reload(sys)
sys.setdefaultencoding("utf-8")
```

In [17]:
```
shapefile =  'ne_10m_admin_0_countries'
num_colors = 9
title = 'Visitors in period from July 20'
imgfile = '.png'
description = '"
Number of sessions were obtained by Google Analytics. Author: R. Righart"'.strip()
```

We are going to make a list called **cnt** consisting of country abbreviations that we will put in the index of **df2**:

In [18]:
```
cnt = []
mapper = country(from_key='name', to_key='iso3')
for i in range(0,len(df2)):
    A = mapper(df2.country[i])
    cnt.append(A)
```

In [19]:
```
df2.index = cnt
```

In [20]:
```
df2.head(5)
```

Out[20]:

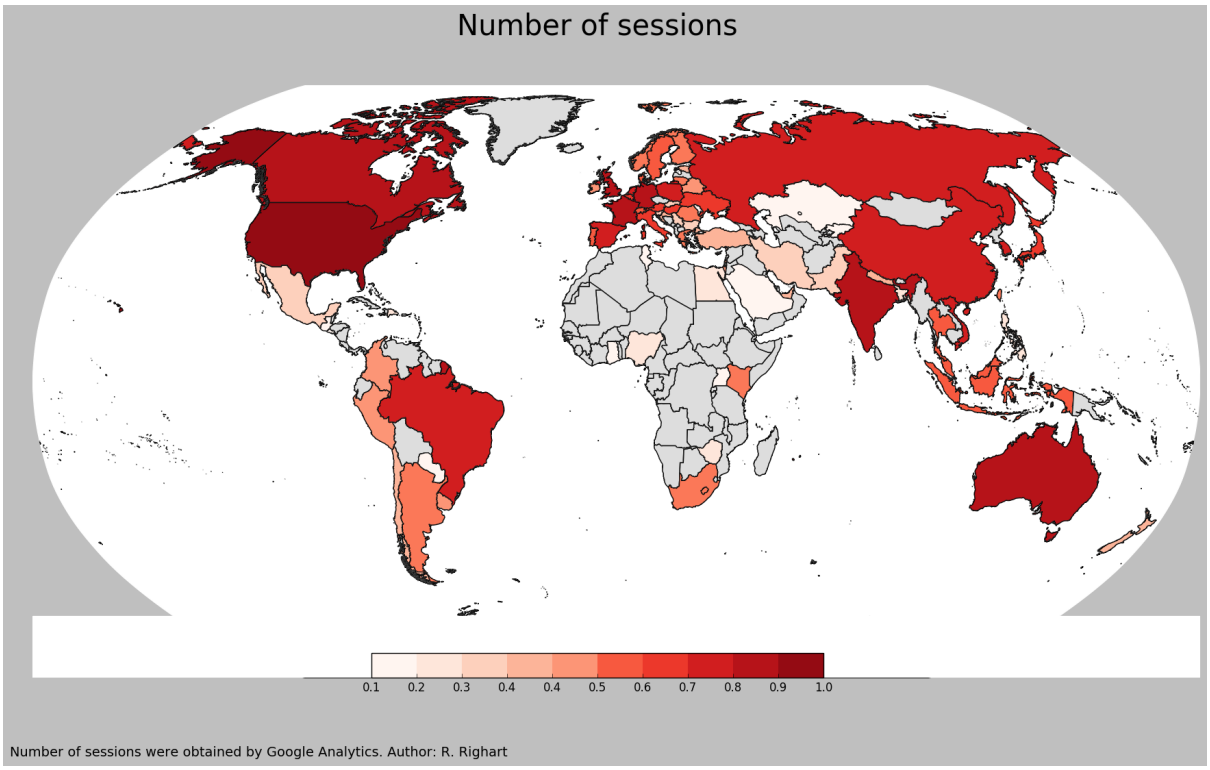
	country	sessions
USA	United States	589
IND	India	95
DEU	Germany	91
GBR	United Kingdom	77
AUS	Australia	54

Using for sessions the *absolute* values did not give a clear color distribution. Most countries had quite similar values with only a few countries having higher values, and for this reason most countries fell in the same color scale. Therefore, I decided to convert the values to percentiles, benefitting from the **Scipy** package[7]. The resulting map demonstrates that visitors do not come from a local geographic area, but they come from a wide variety of countries around the world.

In [21]:
```
values = df2.sessions
values = stats.rankdata(values, "average")/len(values)
```

In [22]:
```
num_colors = 11
cm = plt.get_cmap('Reds')
nw = float("{0:.2f}".format(num_colors))
scheme = [cm(i / nw) for i in range(num_colors)]
bins = np.linspace(values.min(), values.max(), num_colors)
df2['bin'] = np.digitize(values, bins) - 1
```

In [23]:
```
mpl.style.use('classic')
fig = plt.figure(figsize=(22, 12))

ax = fig.add_subplot(111, axisbg='w', frame_on=False)
fig.suptitle('Number of sessions', fontsize=30, y=.95)

m = Basemap(lon_0=0, projection='robin')
m.drawmapboundary(color='w')

m.readshapefile(shapefile, 'units', color='#444444', linewidth=.2)
for info, shape in zip(m.units_info, m.units):
    iso3 = info['ADM0_A3']
    if iso3 not in df2.index:
        color = '#dddddd'
    else:
        color = scheme[df2.ix[iso3]['bin']]

    patches = [Polygon(np.array(shape), True)]
    pc = PatchCollection(patches)
    pc.set_facecolor(color)
    ax.add_collection(pc)

ax.axhspan(0, 1000 * 1800, facecolor='w', edgecolor='w', zorder=2)
ax_legend = fig.add_axes([0.35, 0.14, 0.3, 0.03], zorder=3)
cmap = mpl.colors.ListedColormap(scheme)
cb = mpl.colorbar.ColorbarBase(ax_legend, cmap=cmap, ticks=bins, boundaries=bins, orientation='horizontal')
cb.ax.set_xticklabels([str(round(i, 1)) for i in bins])
plt.annotate(description, xy=(-.8, -3.2), size=14, xycoords='axes fraction')
plt.savefig(imgfile, bbox_inches='tight', pad_inches=.2)
plt.show()
```

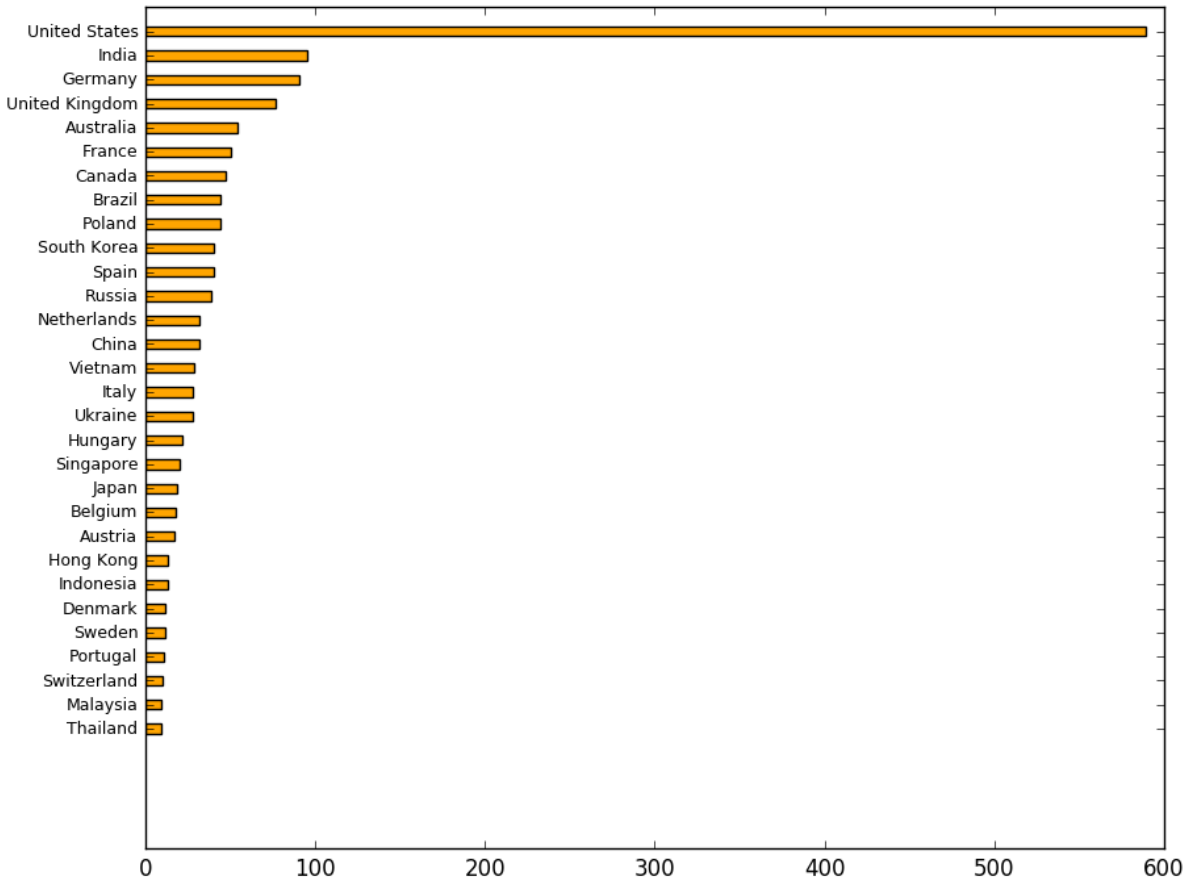


More detail is possible here, such as regional or city maps. An example of a city map can be found elsewhere[8]. Region and city dimensions can be extracted from Google Analytics. As mentioned before, it is best to explore the available "metrics" and "dimensions" in Query Explorer before implementing it in Python.

To get a better impression of the real number of sessions from "top 30" countries, we could make a barplot.

In [24]:
```
ndd = df2.head(30)
```

In [25]:
```
fig = plt.figure(figsize=(10, 8), facecolor='w')
fig.subplots_adjust(wspace=0.2)

ax1 = plt.subplot(1,1,1)
barwd = 0.6
r1=range(len(ndd))
ax1.barh(r1, ndd.sort_values(by='sessions', ascending=True).sessions, height=0.4, align="center", color="orange")
ax1.set_yticks(r1)
ax1.set_yticklabels(ndd.sort_values(by='sessions', ascending=True).country, size=9)
plt.show()
```



10. Device

The third question --what kind of devices do my visitors use? -- can be best answered using a piechart, since it nicely illustrates the proportions.

In [26]:
```
df3 = []

conn = GoogleAnalyticsQuery(secrets='/your-directory/ga-creds/client_secret.json', token_file_name='/your-directory/ga-creds/analytics.dat')

query = {
'ids' : '999999999',
'metrics' : 'sessions',
'dimensions' : 'deviceCategory',
'sort' : '-sessions',
'start_date' : '2017-07-20',
'end_date' : '2017-08-07'
}

df3, metadata = conn.execute_query(**query)
```

In [27]:
```
df3
```

Out[27]:

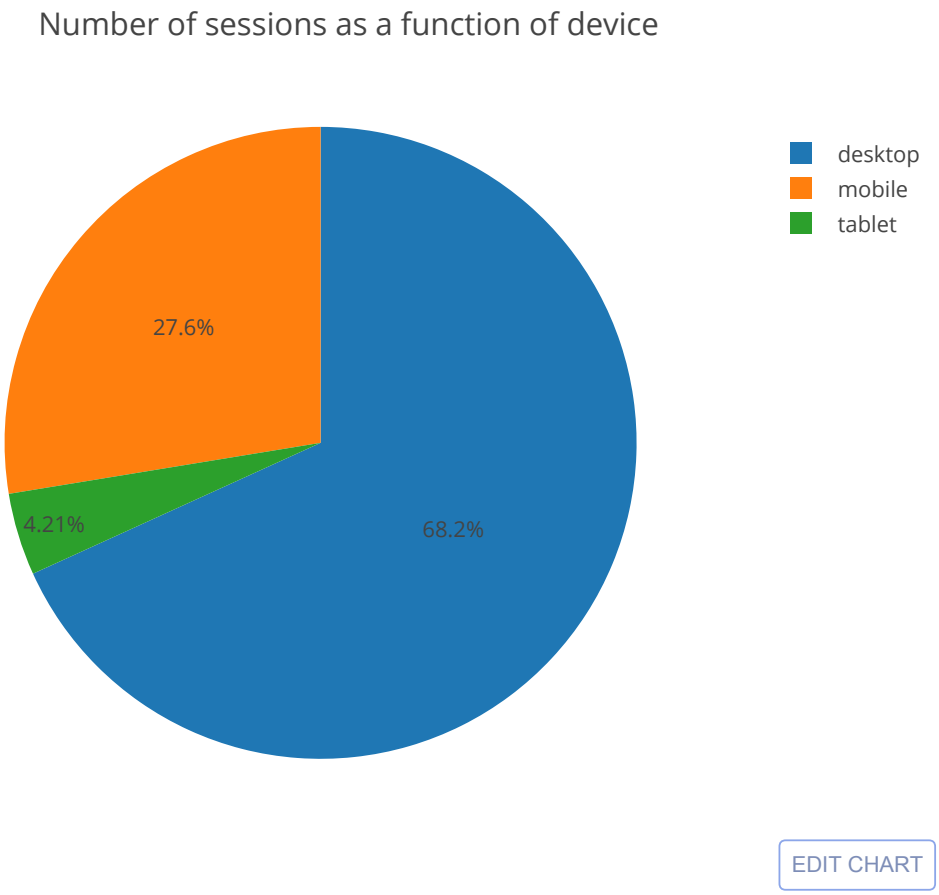
	deviceCategory	sessions
0	desktop	1182
1	mobile	478
2	tablet	73

For this purpose we are going to use Plotly, but one could equally well do this using for example Matplotlib. Please be aware that you would need a so-called *api-key* to do this, which you will receive upon registration at the Plotly website.

In [28]:
```
plotly.tools.set_credentials_file(username='rrighart', api_key='999999999')
```

In [29]:
```
fig = {
  'data': [{'labels': df3.deviceCategory,
            'values': df3.sessions,
            'type': 'pie'}],
  'layout': {'title': 'Number of sessions as a function of device'}
   }

py.iplot(fig)
```

Out[29]:



The pie chart shows that a large majority of the visitors used a desktop. Does this change from one week to another week? It would be possible to show the change in device use over time using multiple pie charts. First, we make two datasets, a dataset for the first and second week, **df3a** and **df3b** respectively. The code is a bit extensive, it is advisable to use a loop when you have several timepoints.

In [30]:
```
df3a = []

conn = GoogleAnalyticsQuery(secrets='/your-directory/ga-creds/client_secret.json', token_file_name='/your-directory/ga-creds/analytics.dat')

query = {
'ids' : '999999999',
'metrics' : 'sessions',
'dimensions' : 'deviceCategory',
'sort' : '-sessions',
'start_date' : '2017-07-20',
'end_date' : '2017-07-26'
}

df3a, metadata = conn.execute_query(**query)

df3b = []

conn = GoogleAnalyticsQuery(secrets='/your-directory/ga-creds/client_secret.json', token_file_name='/your-directory/ga-creds/analytics.dat')

query = {
'ids' : '999999999',
'metrics' : 'sessions',
'dimensions' : 'deviceCategory',
'sort' : '-sessions',
'start_date' : '2017-07-27',
'end_date' : '2017-08-02'
}

df3b, metadata = conn.execute_query(**query)
```

In [31]:
```
df3a
```

Out[31]:

	deviceCategory	sessions
0	desktop	423
1	mobile	136
2	tablet	18

In [32]:
```
df3b
```

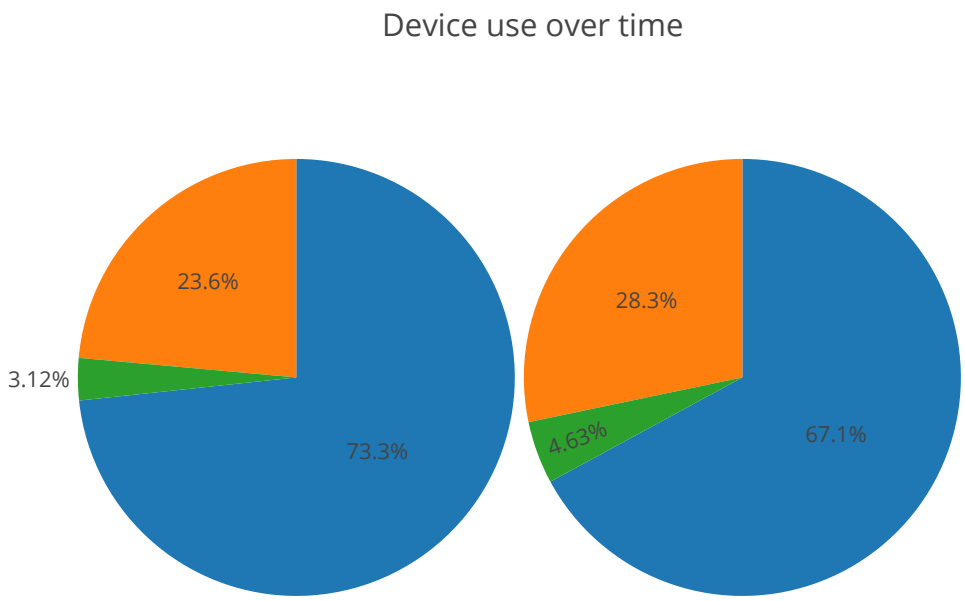
Out[32]:

	deviceCategory	sessions
0	desktop	695
1	mobile	293
2	tablet	48

Now the goal is to display the piecharts for the 1st and 2nd week. It would be important to use the domain parameter to set the position of the charts. Further parameters can be found in the official Plotly documentation[9].

In [33]:
```
fig = {
  'data': [
    {
      'labels': df3a.deviceCategory,
      'values': df3a.sessions,
      'type': 'pie',
      'domain': {'x': [0, .48], 'y': [.21, 1]},
      'name': '1st Week'
    },
    {
      'labels': df3b.deviceCategory,
      'values': df3b.sessions,
      'type': 'pie',
      'domain': {'x': [.49, .97], 'y': [.21, 1]},
      'name': '2nd Week'
    }
  ],
  'layout': {'title': 'Device use over time',
          'showlegend': False}
}

py.iplot(fig)
```

Out[33]:



EDIT CHART

The number of mobile sessions increased slightly while the number of desktop sessions decreased. If this is just noise or a *real* shift in sessions could be probably best evaluated using more timepoints.

11. Save data

If you want to save the DataFrames for later use:

In [34]:
```
df1.to_csv('df1.csv', sep=",", index=False)
df2.to_csv('df2.csv', sep=",", index=False)
df3.to_csv('df3.csv', sep=",", index=False)
df3a.to_csv('df3a.csv', sep=",", index=False)
df3b.to_csv('df3b.csv', sep=",", index=False)
```

12. Closing words

Webtraffic allows a wide variety of potentially interesting analyses. Lots of other measures can be explored. For example, is webtraffic different during certain hours or weekdays?, what is the duration that visitors stay on the website?, from which websites are users referred? is there a change in geographic distribution over time?, just to name a few.

Bringing webtraffic data to Python is rewarding for several reasons. Python can automate basic to advanced analyses. After programming the analysis pipeline, a single button press can produce the desired analyses and visualizations in a report. And last but not least, periodical updates can be produced on a regular basis, customized to your goals.

Notes & References

1. My goal was to track webtraffic to my GitHub site, specifically my blog pages (the so-called gh-pages in GitHub). GitHub has its own webtraffic stats for the master branch, where developers typically share their software tools, scripts etc. The current blog will only deal with Google Analytics.
2. Subscribing and how to get your site detected by Google Analytics. http://www.ryanpraski.com/google-analytics-reporting-api-python-tutorial/ (http://www.ryanpraski.com/google-analytics-reporting-api-python-tutorial/)
3. Setting the tracking code right. http://www.seerinteractive.com/blog/audit-this-why-it-matters-where-you-put-the-google-analytics-tracking-code/ (http://www.seerinteractive.com/blog/audit-this-why-it-matters-where-you-put-the-google-analytics-tracking-code/)
4. Google2pandas. https://github.com/panalysis/Google2Pandas (https://github.com/panalysis/Google2Pandas)
5. A perhaps more appropriate way is converting the column to datetime. As we only use this variable as an *x*-axis in the plot, I have chosen the shorter alternative.
6. Choropleth mapping using Basemap. http://ramiro.org/notebook/basemap-choropleth/ (http://ramiro.org/notebook/basemap-choropleth/)
7. Conversion to percentiles. https://stackoverflow.com/questions/12414043/map-each-list-value-to-its-corresponding-percentile (https://stackoverflow.com/questions/12414043/map-each-list-value-to-its-corresponding-percentile)
8. City map. https://rrighart.github.io/City/#q24 (https://rrighart.github.io/City/#q24)
9. Plotly subplot option. https://plot.ly/python/pie-charts/ (https://plot.ly/python/pie-charts/)

Acknowledgements for CSS style go to jckantor