MunichDataGeeks_Playing_with_data

August 5, 2014

1 Munich DataGeeks - Playing with the Meetup Data

Given that Datageeks Meetup is about cool things to do with data, let's see what we can do with a bit of processing to the data we have available.

Let's start by getting nice defaults and setting up some helpful code. I based the style of using the recommendations and code on the Harvard course on Data Science http://cs109.org/ - Totally recommended resource on learning both Data Science and how to do it with IPtyhon Notebooks.

```
In [62]: #import basic tools and change default colors

         %load_ext autoreload
         %autoreload 2

         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import matplotlib as mpl
         import brewer2mpl
         from matplotlib import rcParams

         # Change the default colors
         dark2_colors = brewer2mpl.get_map('Paired', 'Qualitative', 7).mpl_colors

         rcParams['figure.figsize'] = (10, 10)
         rcParams['figure.dpi'] = 200
         rcParams['axes.color_cycle'] = dark2_colors
         rcParams['lines.linewidth'] = 2
         rcParams['axes.facecolor'] = 'white'
         rcParams['font.size'] = 10
         rcParams['patch.edgecolor'] = 'white'
         rcParams['patch.facecolor'] = dark2_colors[0]
         rcParams['font.family'] = 'StixGeneral'

         %matplotlib inline
         #%matplotlib qt
```

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

1.1 RSVP Comparison

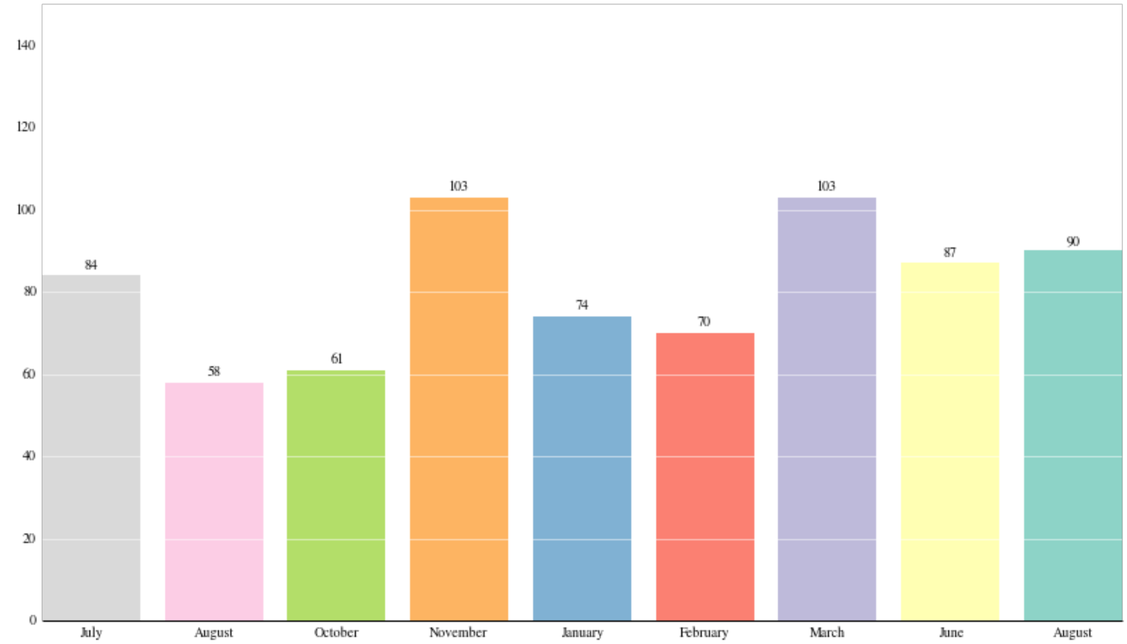
We already had multiple meetups since we started the meetup group. Let's have a look at the trends.

First prepare the data . . .

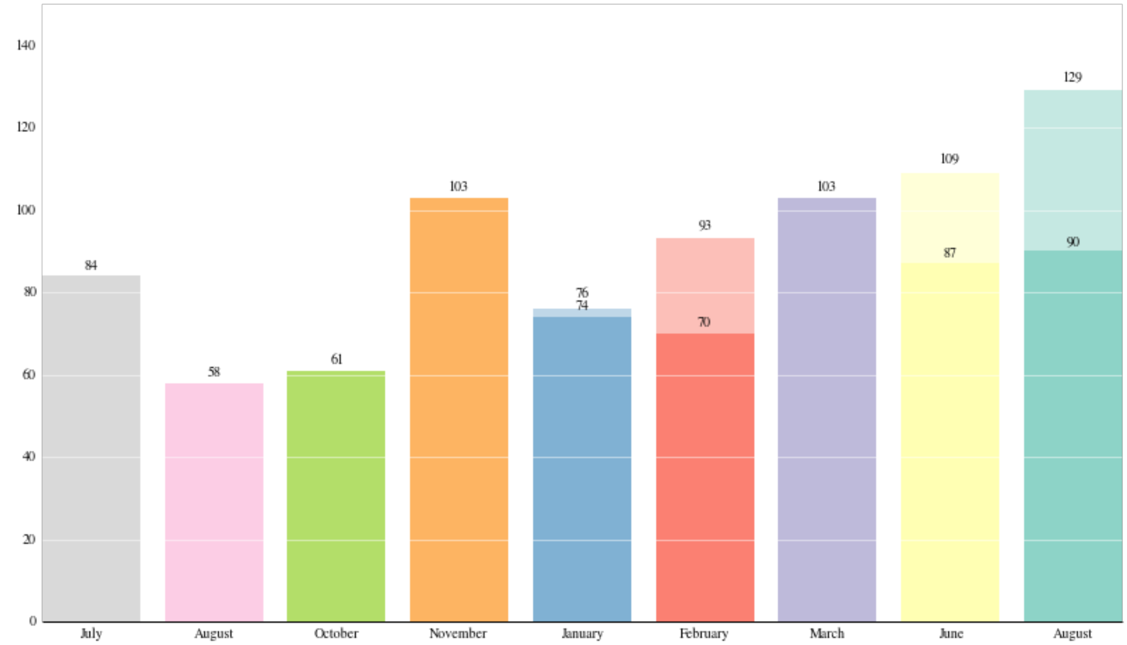
```
In [63]: events = ['July', 'August', 'October',
                   'November', 'January', 'February',
                   'March', 'June', 'August']

         # format: (attendees, waitlisted)
         rsvps = [(84, 0), (58, 0), (61, 0), (103, 0), (74, 2), (70, 23), (103, 0), (87, 22), (90, 39)]
         attendees = [ elem[0] for elem in rsvps ]
         waitlisted = [ elem[1] for elem in rsvps ]
         overall = [ sum(elem) for elem in rsvps ]

         x_pos = np.arange(len(events))
         box_colors = brewer2mpl.get_map('Set3', 'Qualitative', len(events), reverse=True).mpl_colors
```

... then define plotting function (reusable code).

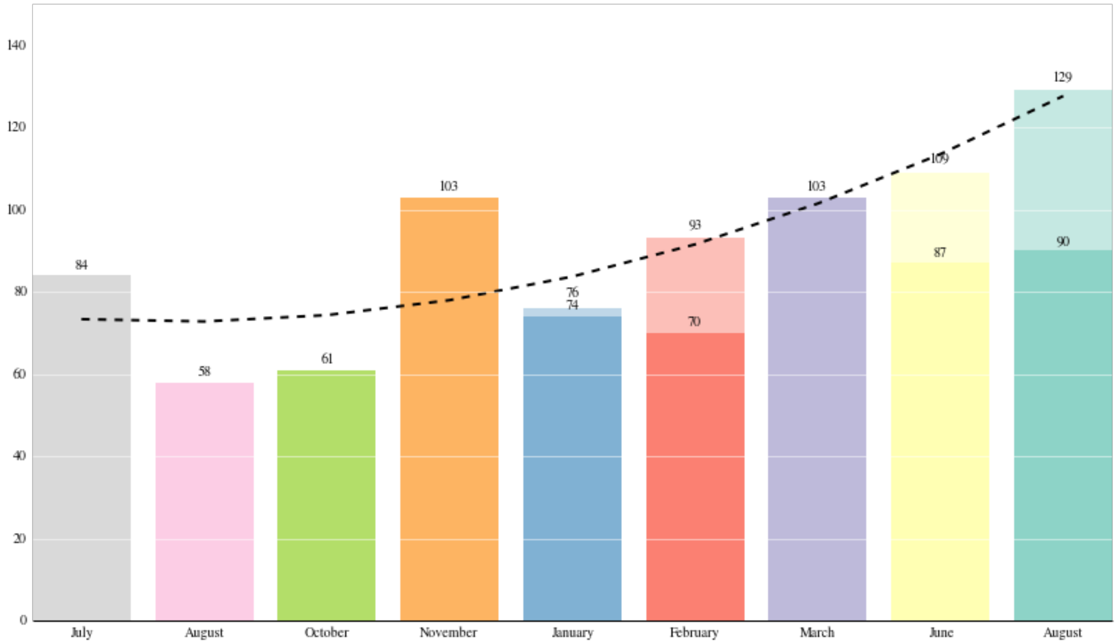
```
In [64]: def plot_attendees(show_waitlisted=False, trendline=False, save=None):
             plt.figure(figsize=(14, 8))
             plt.ylim([0,150])
             plt.xticks(x_pos, events)
             ax = plt.subplot(111)

             # Remove top axes
             ax.spines['top'].set_visible(False)
             ax.spines['right'].set_visible(False)
             ax.spines['left'].set_visible(False)

             # remove ticks
             ax.yaxis.set_ticks_position('none')
             ax.xaxis.set_ticks_position('none')

             ax.grid(axis='y', color='white', linestyle='-')

             for i, rsvp, color in zip(x_pos, attendees, box_colors):
                 ax.bar(i, rsvp, align='center', color=color, linewidth=0)
                 ax.annotate("{}".format(rsvp), (i, rsvp + 1), va="bottom", ha="center")

             if show_waitlisted:
                 for i, rsvp, color in zip(x_pos, waitlisted, box_colors):
                     ax.bar(i, rsvp, align='center', color=color, linewidth=0, bottom=attendees[i], alp
                     if rsvp != 0:
                         ax.annotate("{}".format(overall[i]), (i, rsvp + attendees[i] + 2), va="bottom"

             if trendline:
                 z = np.polyfit(x_pos, overall, 2)
                 p = np.poly1d(z)
                 plt.plot(x_pos, p(x_pos), "k--")

             if save:
                 plt.savefig(save, dpi=200)

             plt.show()
             plt.close()

In [65]: plot_attendees(save="hist-attendees")
```



```
In [66]: plot_attendees(show_waitlisted=True, save="hist-attendees-waitlisted")
```



```
In [67]: plot_attendees(show_waitlisted=True, trendline=True, save="hist-attendees-waitlisted-trendline
```



## Playing with Meetup.com API

It turns out you can get meta-info from the meetups via API. You just need a key to query it. So let's do the best you can do when you have data of users: *stalk people*.

So let's get the data:

```
In [ ]: %%bash
        key=$(cat meetup_apikey.txt)
        curl "http://api.meetup.com/2/members?order=name&group_urlname=Munich-Datageeks&offset=0&format=
```

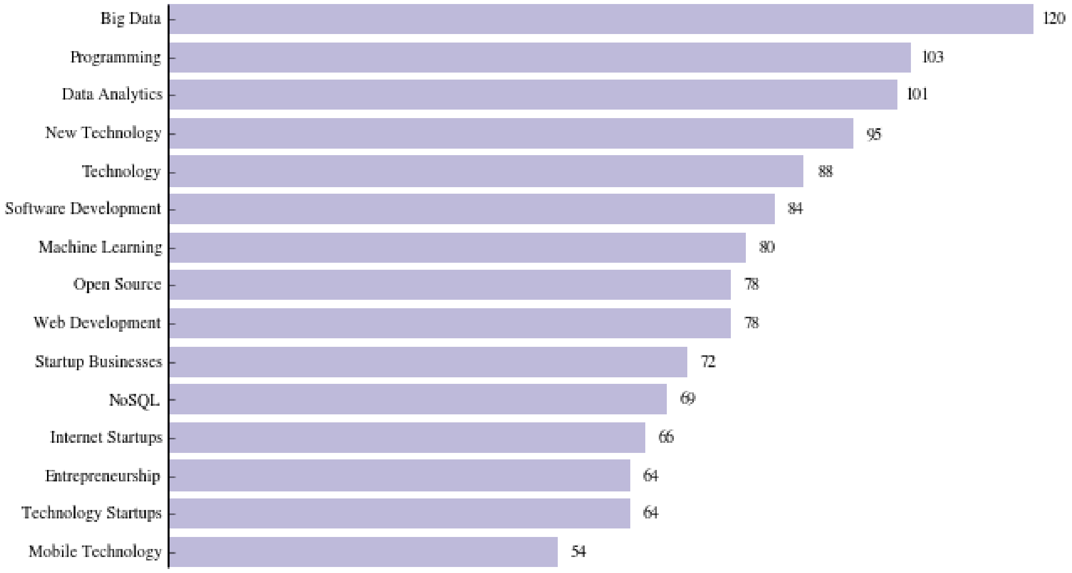
Let's first create a nice function to graph stuff again.

```
In [111]: def plot_count_graph(data_tuples, path, color_index=0):
              plt.figure(figsize=(10, 6))
              ax = plt.subplot(111)

              color = brewer2mpl.get_map('Set3', 'Qualitative', 5).mpl_colors[color_index]

              y_pos = np.arange(len(data_tuples))
              counts = [ j  for i, j in data_tuples ]
              plt.yticks(y_pos, [ i  for i, j in data_tuples ])
              plt.xlim([0, counts[-1] + 10])

              # remove border of plot - should be possible to do it in an easier way!
              ax.spines['top'].set_visible(False)
              ax.spines['right'].set_visible(False)
              ax.spines['bottom'].set_visible(False)
              ax.xaxis.set_ticks([])
              ax.get_yaxis().tick_left()

              padding = 3 if counts[-1] > 100 else 0.5
              for i, count in enumerate(counts):
```

```
            ax.barh(i, count, align='center', linewidth=0, color=color)
            ax.annotate(str(count), (count + padding, i), va="center", ha="center")

        plt.grid(axis='x', color='white', linestyle='-')

        plt.savefig(path, dpi=300, bbox_inches='tight')
        plt.show()
        plt.close()
```

With the previously defined function we are going to write code to get the TOP-15 interests of the Datageeks community:

```
In [124]: import json
          from operator import itemgetter
          from collections import Counter

          interest_counter = Counter()

          with open('members.json') as f:
              member_data = json.load(f)
              for user in member_data['results']:
                  for interest in user['topics']:
                      interest_counter[interest['name']] += 1

          plot_count_graph(interest_counter.most_common()[0:15][::-1], "interests", color_index=2)
```



1.1.1 Let's keep stalking people

I am quite interested in what people write on their bio. Let's see if there are common terms.

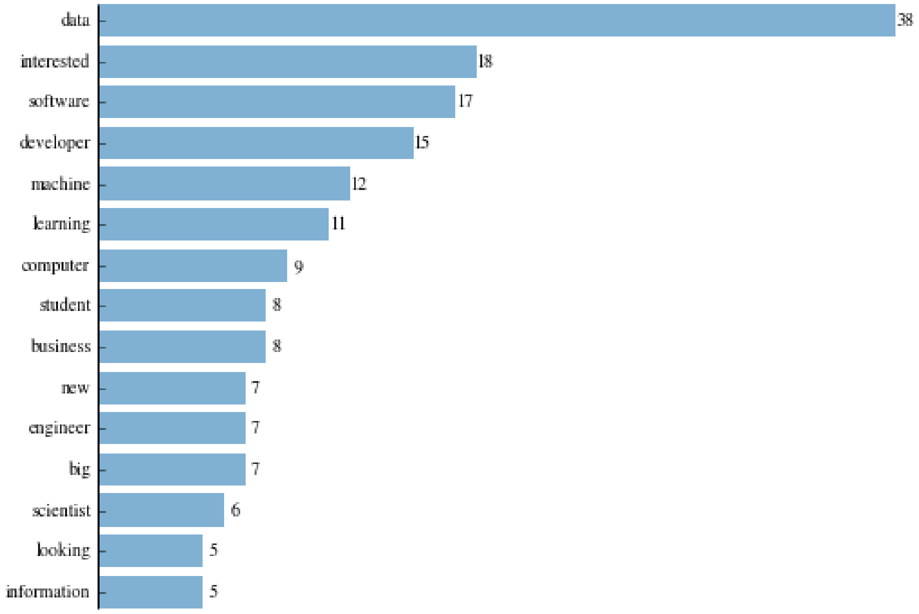
First, preparing some helper functions and getting ready.

```
In [ ]: from sklearn.feature_extraction.text import CountVectorizer
        from itertools import tee, izip
```

```
import nltk
from nltk.corpus import stopwords

# in order to be able to use the NLTK stopwords corpus,
# you need to download it first: http://www.nltk.org/data.html
nltk.download("stopwords")

# itertools recipe
def pairwise(iterable):
    "s -> (s0,s1), (s1,s2), (s2, s3), ..."
    a, b = tee(iterable)
    next(b, None)
    return izip(a, b)

def get_unigrams(bio):
    unigrams_all = bio.strip().replace(',', '').replace('@', '').lower().split(' ')
    filter_func = lambda x: x and x not in stopwords.words('english') + ['hi', "i'm"]
    return filter(filter_func, unigrams_all)
```

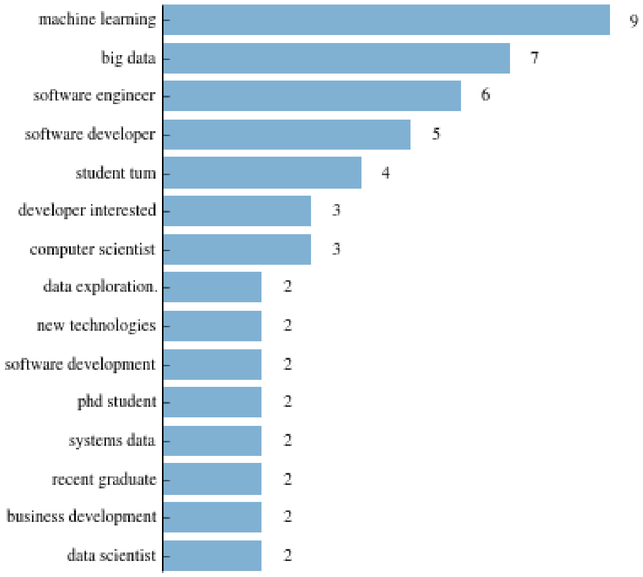
Now, let's have a look at the most common unigrams.

```
In [115]: wordcounts_unigrams = Counter()
          # we still have the variable "member_data" in our scope
          for member in member_data['results']:
              if 'bio' in member:
                  # this is just a quick and easy tokenization which is good enough for our purpose
                  # for more sophisticated tokenization: http://www.nltk.org/api/nltk.tokenize.html
                  unigrams = get_unigrams(member['bio'])
                  wordcounts_unigrams.update(unigrams)

          # now we can reuse the plot_count_graph function defined earlier
          plot_count_graph(wordcounts_unigrams.most_common()[:15][::-1], "unigrams", color_index=4)
```



Well, that's nice but not really that much telling. We expected **data** to be at the top and it looks like plotting the graph based on bigrams would make more sense. So let's to just that.

```
In [120]: wordcounts_bigrams = Counter()
          for member in member_data['results']:
              if 'bio' in member:
                  unigrams = get_unigrams(member['bio'])
                  bigrams = [ " ".join(pair) for pair in pairwise(unigrams) ]
                  wordcounts_bigrams.update(bigrams)

          plot_count_graph(wordcounts_bigrams.most_common()[:15][::-1], "bigrams", color_index=4)
```



Another interesting data source of meetup.com is the groups and the data attached to them, like number of members, rating, city, and country. The API allows you to get 200 groups for one specific topic, e.g. "Machine Learning". I am going to get data for three topics: Machine Learning, Data Science, and Big Data.

```
In []: %%bash
       key=$(cat meetup_apikey.txt)
       curl "http://api.meetup.com/2/groups.json/?topic=machine-learning&offset=0&order=members&key=${ke
       curl "http://api.meetup.com/2/groups.json/?topic=data-science&order=members&key=${key}" > ds_grou
       curl "http://api.meetup.com/2/groups.json/?topic=big-data&order=members&key=${key}" > bd_groups.
```

Now get the data in the format we want it to be.

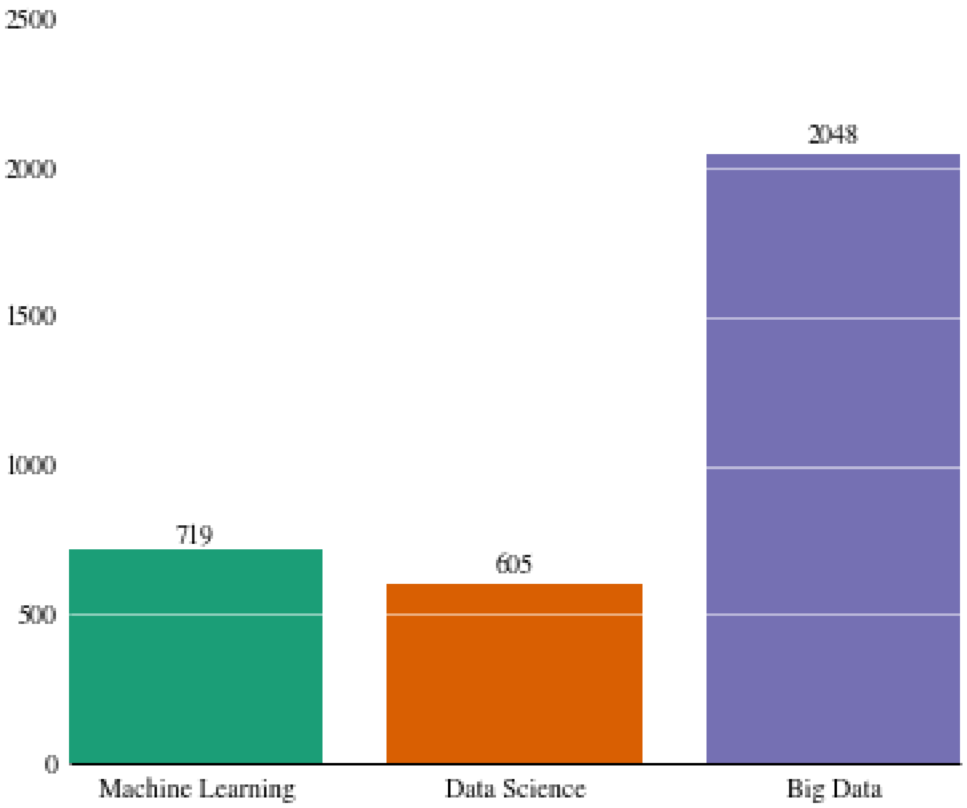
```
In [75]: from collections import defaultdict

         def get_group_data(path):
             data = defaultdict(dict)
             with open(path) as f:
                 group_data = json.load(f)

             for group in group_data["results"]:
                 # I am assuming that the name is unique
                 name = group["name"]
                 data[name]["city"] = group["city"]
                 data[name]["country"] = group["country"]
                 data[name]["rating"] = group["rating"]
                 data[name]["members"] = group["members"]

             # the api response only gives you data about the top 200 groups
             # hence, we need to get the total number of groups this way
             total_count = group_data["meta"]["total_count"]
             return data, total_count

         ml_data, ml_total_count = get_group_data("ml_groups.json")
         ds_data, ds_total_count = get_group_data("ds_groups.json")
         bd_data, bd_total_count = get_group_data("bd_groups.json")
```

This kind of structured data really screams pandas DataFrame at me :)

```
In [76]: ml_df = pd.DataFrame.from_dict(ml_data, orient=’index’)
         ds_df = pd.DataFrame.from_dict(ds_data, orient=’index’)
         bd_df = pd.DataFrame.from_dict(bd_data, orient=’index’)
```

Hmm…which of the three topics is more popular?

```
In [77]: # code very similar to "plot_attendees" function - should be refactored (once I have time)
         plt.figure(figsize=(6, 5))
         topics = ["Machine Learning", "Data Science", "Big Data"]
         counts = [ml_total_count, ds_total_count, bd_total_count]
         colors = brewer2mpl.get_map(’Dark2’, ’Qualitative’, len(topics)).mpl_colors

         x_pos = np.arange(3)
         plt.xticks(x_pos, topics)
         ax = plt.subplot(111)

         # Remove top axes
         ax.spines[’top’].set_visible(False)
         ax.spines[’right’].set_visible(False)
         ax.spines[’left’].set_visible(False)

         # remove ticks
         ax.yaxis.set_ticks_position(’none’)
         ax.xaxis.set_ticks_position(’none’)
         ax.grid(axis=’y’, color=’white’, linestyle=’-’)

         for i, topic, color in zip(x_pos, topics, colors):
                 ax.bar(i, counts[i], align=’center’, color=color, linewidth=0)
                 ax.annotate("{}".format(counts[i]), (i, counts[i] + 20), va="bottom", ha="center")

         plt.savefig("group_total_counts", dpi=200)
         plt.show()
         plt.close()
```



In the following, I will only analyze the "Machine Learning" groups. The same could be done for all the other group data of course.

List all the groups in Munich which have "Machine Learning" as one of their topics (remember that we only consider the top 200 groups based on number of members).

```
In [78]: ml_df[ml_df.city == u'München']
```

```
Out[78]:                      city  rating  members country
         Munich Datageeks  München    4.65      618      DE

         [1 rows x 4 columns]
```

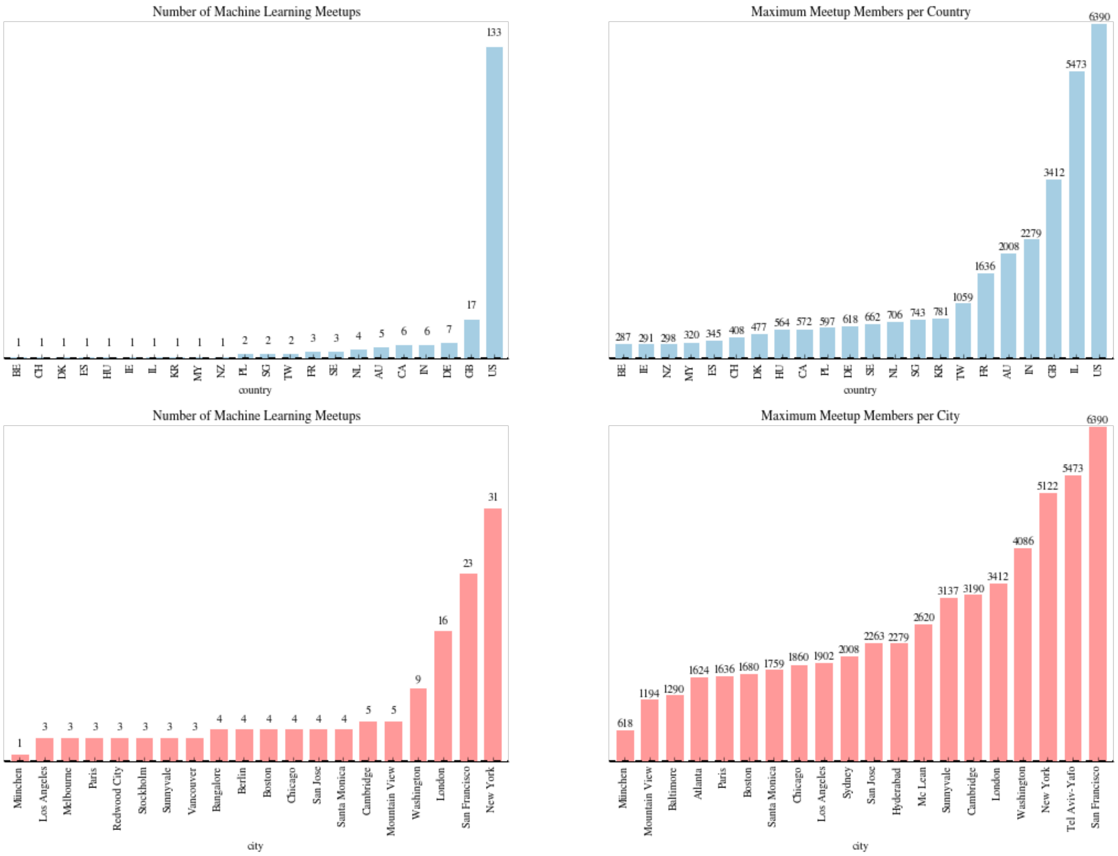
Now let's just get a quick overview of the data to know what we're dealing with here.

```
In [79]: ml_df.describe()
```

```
Out[79]:             rating      members
         count  200.000000   200.000000
         mean     4.372400   915.615000
         std      0.913214   973.938911
         min      0.000000   275.000000
         25%      4.427500   378.750000
         50%      4.560000   543.000000
         75%      4.710000  1004.250000
         max      5.000000  6390.000000

         [8 rows x 2 columns]
```

Next up, let's see where Meetups are quite popular - first on a country basis, then on a city basis.

In [83]:
```
# group dataframe by city or country
ml_df_city = ml_df.groupby("city")
ml_df_country = ml_df.groupby("country")

def apply_plot_settings(counts):
    ax = plt.gca()
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.spines['left'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    plt.yticks([])
    plt.ylim([0, counts[-1] + 10])

    padding = 3 if counts[-1] > 100 else 0.5
    for i, count in enumerate(counts):
        # the i + 0.65 is a little weird and I think necessary because of the pandas plot wrap
        # well, it works
        ax.annotate("{}".format(count), (i + 0.65, count + padding), va="bottom", ha="center")

# now the plotting starts
plt.figure(figsize=(18, 12))

plt.subplot(221)
apply_plot_settings(ml_df_country.count().rating.order())
ml_df_country.count().rating.order().plot(kind='bar', title='Number of Machine Learning Meetups

plt.subplot(222)
apply_plot_settings(ml_df_country.members.max().order())
ml_df_country.members.max().order().plot(kind='bar', title='Maximum Meetup Members per Country

plt.subplot(223)
# we want to have Munich in the plot :)
city_count_top = ml_df_city.count().rating.order()[-19:]
city_count_top[u'München'] = ml_df_city.count().rating[u"München"]
apply_plot_settings(city_count_top.order())
city_count_top.order().plot(kind='bar', title='Number of Machine Learning Meetups', grid=False

plt.subplot(224)
# we want to have Munich in the plot :)
city_members_top = ml_df_city.members.max().order()[-19:]
city_members_top[u'München'] = ml_df_city.members.max()[u"München"]
apply_plot_settings(city_members_top.order())
city_members_top.order().plot(kind='bar', title='Maximum Meetup Members per City', grid=False,
plt.savefig('country_city_stats', dpi=200)
```



Wow! These plots really shed a bad light on Munich as it looks like Munich is always in the last position. Time to change that by looking at what cities we are ahead of :)

First, how do we rank within Germany?

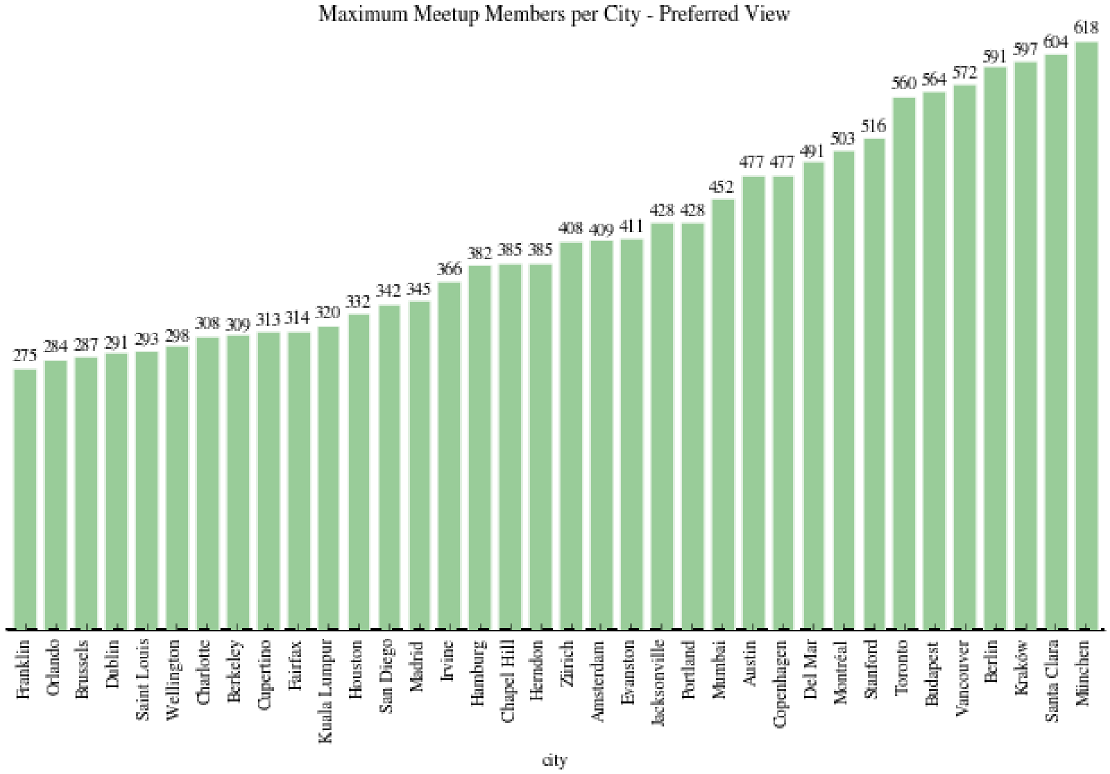
```
In [81]: ml_df[ml_df.country == "DE"].sort(columns=["members"], ascending=False)[["members", "city"]]
```

```
Out[81]:                                        members     city
         Munich Datageeks                           618  München
         Berlin Machine learning group              591   Berlin
         SoundCloud Tech Meetups                    407   Berlin
         Algorithms & Data Challenges Berlin        394   Berlin
         Big Data & NoSQL Meetup Hamburg            382  Hamburg
         12min.me                                   311  Hamburg
         Data Science Berlin                        302   Berlin

         [7 rows x 2 columns]
```

Now, what other cities do we surpass in number of members?

```
In [90]: max_members_munich = ml_df_city.members.max()[u"München"]
         surpassed = ml_df_city.members.max().order()[ml_df_city.members.max().order() <= max_members_mu
         print "Number of surpassed cities: {}".format(surpassed.count())
         plt.figure(figsize=(11, 6))
         apply_plot_settings(surpassed)
         surpassed.plot(kind='bar', title='Maximum Meetup Members per City - Preferred View', grid=Fals
         plt.savefig('surpassed_cities', dpi=200)
```

```
Number of surpassed cities: 36
```



1.2 What else could we explore?

- compare number of members per city with number of residents (or maybe even with number of IT professionals)
- analyze the "rating" column in more detail, e.g. "Are ratings in USA better than in Germany?"
- we have Google Analytics activated for our Meetup web page, so we could dig into that

1.3 Inspiration and Reference

Some interesting liks regarding visualization of data and Data Science in general:

- https://drive.google.com/folderview?id=0BxYkKyLxfsNVd0xicUVDS1dIS0k&usp=sharing
- http://nbviewer.ipython.org/5357268
- http://nbviewer.ipython.org/urls/raw.github.com/cs109/content/master/lec_03_statistical_graphs.ipynb