CS 109A/STAT 121A/AC 209A/CSCI E-109A: Homework 1

Harvard University
Fall 2017
Instructors: Pavlos Protopapas, Kevin Rader, Rahul Dave, Margo Levine

---

INSTRUCTIONS

WARNING: There is web page scraping in this homework. It takes about 40 minutes. **Do not wait till the last minute** to do this homework.

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit. There is an important CAVEAT to this. DO NOT run the web-page fetching cells again. (We have provided hints like `# DO NOT RERUN THIS CELL WHEN SUBMITTING` on some of the cells where we provide the code). Instead load your data structures from the JSON files we will ask you to save below. Otherwise you will be waiting for a long time. (Another reason to not wait until the last moment to submit.)
- Do not include your name in the notebook.

Homework 1: Rihanna or Mariah?

Billboard Magazine puts out a top 100 list of "singles" every week. Information from this list, as well as that from music sales, radio, and other sources is used to determine a top-100 "singles" of the year list. A **single** is typically one song, but sometimes can be two songs which are on one "single" record.

In this homework you will:

1. Scrape Wikipedia to obtain infprmation about the best singers and groups from each year (distinguishing between the two groups) as determined by the Billboard top 100 charts. You will have to clean this data. Along the way you will learn how to save data in json files to avoid repeated scraping.
2. Scrape Wikipedia to obtain information on these singers. You will have to scrape the web pages, this time using a cache to guard against network timeouts (or your laptop going to sleep). You will again clean the data, and save it to a json file.
3. Use pandas to represent these two datasets and merge them.
4. Use the individual and merged datasets to visualize the performance of the artists and their songs. We have kept the amount of analysis limited here for reasons of time; but you might enjoy exploring music genres and other aspects of the music business you can find on these wikipedia pages at your own leisure.

You should have worked through Lab0 and Lab 1, and Lecture 2. Lab 2 will help as well.

As usual, first we import the necessary libraries. In particular, we use Seaborn (http://stanford.edu/~mwaskom/software/seaborn/) to give us a nicer default color palette, with our plots being of large (`poster`) size and with a white-grid background.

```
In [1]:  %matplotlib inline
         import numpy as np
         import scipy as sp
         import matplotlib as mpl
         import matplotlib.cm as cm
         import matplotlib.pyplot as plt
         import pandas as pd
         import time
         pd.set_option('display.width', 500)
         pd.set_option('display.max_columns', 100)
         pd.set_option('display.notebook_repr_html', True)
         import seaborn as sns
         sns.set_style("whitegrid")
         sns.set_context("poster")
```

Q1. Scraping Wikipedia for Billboard Top 100.

In this question you will scrape Wikipedia for the Billboard's top 100 singles.

Scraping Wikipedia for Billboard singles

We'll be using BeautifulSoup (http://www.crummy.com/software/BeautifulSoup/), and suggest that you use Python's built in `requests` library to fetch the web page.

1.1 Parsing the Billboard Wikipedia page for 1970

Obtain the web page at http://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_1970 (http://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_1970) using a HTTP GET request. From this web page we'll extract the top 100 singles and their rankings. Create a list of dictionaries, 100 of them to be precise, with entries like

`{'url': '/wiki/Sugarloaf_(band)', 'ranking': 30, 'band_singer': 'Sugarloaf', 'title': 'Green-Eyed Lady'}`.

If you look at that web page, you'll see a link for every song, from which you can get the `url` of the singer or band. We will use these links later to scrape information about the singer or band. From the listing we can also get the band or singer name `band_singer`, and `title` of the song.

HINT: look for a table with class `wikitable`.

You should get something similar to this (where songs is the aforementioned list):

```
songs[2:4]

[{'band_singer': 'The Guess Who',
  'ranking': 3,
  'title': '"American Woman"',
  'url': '/wiki/The_Guess_Who'},
 {'band_singer': 'B.J. Thomas',
  'ranking': 4,
  'title': '"Raindrops Keep Fallin\' on My Head"',
  'url': '/wiki/B.J._Thomas'}]
```

In [2]:
```python
from IPython.display import IFrame, HTML
```

In [3]:
```python
import requests
from bs4 import BeautifulSoup
```

In [ ]:
```python
# Getting the wikipedia page for 1970
req_1970 = requests.get("http://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_1970")
page = req_1970.text
```

In [ ]:
```python
soup_1970 = BeautifulSoup(page, 'html.parser')
```

In [ ]:
```python
# Getting the correct table
# Since it is guaranteed to be uinique, use find rather than find_all
table_1970 = soup_1970.find("table", "wikitable", "sortable")
```

In [ ]:
```python
# Extract all the rows
rows_1970 = [row for row in table_1970.find_all("tr")]
```

In [ ]:
```python
columns = ["ranking", "title", "band_singer", "url"]
```

```
In [ ]: values_1970 = []
        for row in rows_1970[1:]:
            value_row = []

            # Getting all the values
            for v in row.find_all("td"):
                value_row.append(v.get_text())
            value_row[0] = int(value_row[0])

            # Getting the singer url- the last url is the singer url
            urls = row.find_all("a")
            value_row.append(urls[-1]['href'])

            values_1970.append(tuple(value_row))
```

```
In [ ]: songs_1970 = [{col: val for col, val in zip(columns, values)} for values in values_1970]
        songs_1970[2:4]
```

1.2 Generalize the previous: scrape Wikipedia from 1992 to 2014

By visiting the urls similar to the ones for 1970, we can obtain the billboard top 100 for the years 1992 to 2014. (We choose these later years rather than 1970 as you might find music from this era more interesting.) Download these using Python's `requests` module and store the text from those requests in a dictionary called `yearstext`. This dictionary ought to have as its keys the years (as integers from 1992 to 2014), and as values corresponding to these keys the text of the page being fetched.

You ought to sleep a second (look up `time.sleep` in Python) at the very least in-between fetching each web page: you do not want Wikipedia to think you are a marauding bot attempting to mount a denial-of-service attack.

HINT: you might find range and string-interpolation useful to construct the URLs .

```
In [ ]: year_list = list(range(1992, 2015))
        year_list
        address_pre = "http://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_"
        yearstext = dict();
        for this_year in year_list:
            this_address = address_pre + str(this_year)
            req = requests.get(this_address)
            page = req.text
            yearstext[this_year] = page

            time.sleep(1.1)
```

1.3 Parse and Clean data

Remember the code you wrote to get data from 1970 which produces a list of dictionaries, one corresponding to each single. Now write a function `parse_year(the_year, yeartext_dict)` which takes the year, prints it out, gets the text for the year from the just created `yearstext` dictionary, and return a list of dictionaries for that year, with one dictionary for each single. Store this list in the variable `yearinfo`.

The dictionaries **must** be of this form:

```
{'band_singer': ['Brandy', 'Monica'],
  'ranking': 2,
  'song': ['The Boy Is Mine'],
  'songurl': ['/wiki/The_Boy_Is_Mine_(song)'],
  'titletext': '" The Boy Is Mine "',
  'url': ['/wiki/Brandy_Norwood', '/wiki/Monica_(entertainer)']}
```

The spec of this function is provided below:

```
In [ ]:
"""
Function
--------
parse_year

Inputs
------
the_year: the year you want the singles for
yeartext_dict: a dictionary with keys as integer years and values the downloaded web pages
    from wikipedia for that year.

Returns
-------

a list of dictionaries, each of which corresponds to a single and has the
following data:

Eg:

{'band_singer': ['Brandy', 'Monica'],
  'ranking': 2,
  'song': ['The Boy Is Mine'],
  'songurl': ['/wiki/The_Boy_Is_Mine_(song)'],
  'titletext': '" The Boy Is Mine "',
  'url': ['/wiki/Brandy_Norwood', '/wiki/Monica_(entertainer)']}

A dictionary with the following data:
    band_singer: a list of bands/singers who made this single
    song: a list of the titles of songs on this single
    songurl: a list of the same size as song which has urls for the songs on the single
        (see point 3 above)
    ranking: ranking of the single
    titletext: the contents of the table cell
    band_singer: a list of bands or singers on this single
    url: a list of wikipedia singer/band urls on this single: only put in the part
        of the url from /wiki onwards


Notes
-----
See description and example above.
"""
```

```python
In [4]: import re

columns = ["ranking", "titletext", "song", "songurl", "band_singer", "url"]

def parse_year(the_year, yeartext_dict):
    page = yeartext_dict[the_year]
    soup = BeautifulSoup(page, 'html.parser')
    table = soup.find("table", "wikitable", "sortable")
    rows = [row for row in table.find_all("tr")]
    info_all = []
    for row in rows[1:]:
        info_row = []
        # Check if the rank is in <td> or <th>
        info_th = row.find("th")
        info_td = row.find_all("td")
        if info_th:
            rank = int(info_th.get_text())
            ind = 0;
        else:
            rank = int(info_td[0].get_text()) # First item is ranking
            ind = 1;
            print("Ranking in td %d %d" % (the_year, rank))
        info_row.append(rank) # Append ranking

        # Next are song's title and url
        titletext = info_td[ind].get_text()
        info_row.append(titletext) # Append titletext
        titletext_splt = titletext.split(" / ")
        title_w_url = info_td[ind].find_all("a")
        song = []
        songurl = []
        # Some songs are partially linked with url, so splitted title gives the most complete song name(s)
        # See Rank #1 song in 1996 Billboard
        for t in titletext_splt:
            if (t != '' and t != ' '): # Get rid of empty name
                song.append(t.replace('"', ''))
                if title_w_url:
                    for v in title_w_url:
                        t_partial = v.get_text()
                        if (t_partial in t) or (t in t_partial):
                            songurl.append(v['href'])
                            break
                else:
                    songurl.append(None)
        info_row.append(song) # Append song
        info_row.append(songurl) # Append song url

        # Next are singer and url
        ind += 1
        singer_text = info_td[ind].get_text()
        singer_splt = re.split(" featuring | and |, ", singer_text)
        singer_w_url = info_td[ind].find_all("a")
        band_singer = []
        url = []
        # Some singers' name may be partially linked with url too (not in 1992-2014 though)
        # Just to be general here
        for s in singer_splt:
            if (s != '' and s != ' '): # Get rid of empty name
                band_singer.append(s)
                if singer_w_url:
                    for v in singer_w_url:
                        s_partial = v.get_text()
                        if (s_partial in s) or (s in s_partial):
                            url.append(v['href'])
                            break
                else:
                    url.append(None)
        info_row.append(band_singer) # Append band_singer
        info_row.append(url) # Append singer url
        # Add everything into info_all
        info_all.append(tuple(info_row))

    yearinfo = [{col: val for col, val in zip(columns, info)} for info in info_all]
    return(yearinfo)
```

```
In [ ]:  parse_year(1997, yearstext)[2]
```

```
In [ ]:  # Create a dictionary with key = year and value = the list of dictionary for that year
         yearinfo = dict()
         for y in range(1992, 2015):
             this_yearinfo = parse_year(y, yearstext)
             yearinfo[y] = this_yearinfo
```

Helpful notes

Notice that some singles might have multiple songs:

```
{'band_singer': ['Jewel'],
  'ranking': 2,
  'song': ['Foolish Games', 'You Were Meant for Me'],
  'songurl': ['/wiki/Foolish_Games',
   '/wiki/You_Were_Meant_for_Me_(Jewel_song)'],
  'titletext': '" Foolish Games " / " You Were Meant for Me "',
  'url': ['/wiki/Jewel_(singer)']}
```

And some singles don't have a song URL:

```
{'band_singer': [u'Nu Flavor'],
  'ranking': 91,
  'song': [u'Heaven'],
  'songurl': [None],
  'titletext': u'"Heaven"',
  'url': [u'/wiki/Nu_Flavor']}
```

Thus there are some issues this function must handle:

1. There can be more than one band_singer as can be seen above (sometimes with a comma, sometimes with "featuring" in between). The best way to parse these is to look for the urls.
2. There can be two songs in a single, because of the way the industry works: there are two-sided singles. See https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_1997 (https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_1997) for an example. You can find other examples in 1998 and 1999.
3. The titletext is the contents of the table cell, and retains the quotes that Wikipedia puts on the single.
4. If no song anchor is found (see the 24th song in the above url), assume there is one song in the single, set songurl to [None] and the song name to the contents of the table cell with the quotes stripped (ie song is a one-element list with this the titletext stripped of its quotes).

As a check, we can do this for 1997. We'll print the first 5 outputs: parse_year(1997, yearstext)[:5]

This should give the following. Notice that the year 1997 exercises the edge cases we talked about earlier.

```
[{'band_singer': ['Elton John'],
  'ranking': 1,
  'song': ['Something About the Way You Look Tonight',
   'Candle in the Wind 1997'],
  'songurl': ['/wiki/Something_About_the_Way_You_Look_Tonight',
   '/wiki/Candle_in_the_Wind_1997'],
  'titletext': '" Something About the Way You Look Tonight " / " Candle in the Wind 1997 "',
  'url': ['/wiki/Elton_John']},
 {'band_singer': ['Jewel'],
  'ranking': 2,
  'song': ['Foolish Games', 'You Were Meant for Me'],
  'songurl': ['/wiki/Foolish_Games',
   '/wiki/You_Were_Meant_for_Me_(Jewel_song)'],
  'titletext': '" Foolish Games " / " You Were Meant for Me "',
  'url': ['/wiki/Jewel_(singer)']},
 {'band_singer': ['Puff Daddy', 'Faith Evans', '112'],
  'ranking': 3,
```

```
                      'song': ["I'll Be Missing You"],
                      'songurl': ['/wiki/I%27ll_Be_Missing_You'],
                      'titletext': '" I\'ll Be Missing You "',
                      'url': ['/wiki/Sean_Combs', '/wiki/Faith_Evans', '/wiki/112_(band)']},
                     {'band_singer': ['Toni Braxton'],
                      'ranking': 4,
                      'song': ['Un-Break My Heart'],
                      'songurl': ['/wiki/Un-Break_My_Heart'],
                      'titletext': '" Un-Break My Heart "',
                      'url': ['/wiki/Toni_Braxton']},
                     {'band_singer': ['Puff Daddy', 'Mase'],
                      'ranking': 5,
                      'song': ["Can't Nobody Hold Me Down"],
                      'songurl': ['/wiki/Can%27t_Nobody_Hold_Me_Down'],
                      'titletext': '" Can\'t Nobody Hold Me Down "',
                      'url': ['/wiki/Sean_Combs', '/wiki/Mase']}]
```

Save a json file of information from the scraped files

We do not want to lose all this work, so let's save the last data structure we created to disk. That way if you need to re-run from here, you don't need to redo all these requests and parsing.

DO NOT RERUN THE HTTP REQUESTS TO WIKIPEDIA WHEN SUBMITTING.

*We **DO NOT** need to see these JSON files in your submission!*

In [5]:
```python
import json
```

In [ ]:
```python
# DO NOT RERUN THIS CELL WHEN SUBMITTING
fd = open("data/yearinfo.json","w")
json.dump(yearinfo, fd)
fd.close()
del yearinfo
```

Now let's reload our JSON file into the yearinfo variable, just to be sure everything is working.

In [6]:
```python
# RERUN WHEN SUBMITTING
# Another way to deal with files. Has the advantage of closing the file for you.
with open("data/yearinfo.json", "r") as fd:
    yearinfo = json.load(fd)
```

1.4 Construct a year-song-singer dataframe from the yearly information

Let's construct a dataframe `flatframe` from the `yearinfo`. The frame should be similar to the frame below. Each row of the frame represents a song, and carries with it the chief properties of year, song, singer, and ranking.

	year	band_singer	ranking	song	songurl	url
0	1992	Boyz II Men	1.0	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men
1	1993	Whitney Houston	1.0	I Will Always Love You	/wiki/I_Will_Always_Love_You#Whitney_Houston_v...	/wiki/Whitney_Houston
2	1994	Ace of Base	1.0	The Sign (song)	/wiki/The_Sign_(song)	/wiki/Ace_of_Base
3	1995	Coolio	1.0	Gangsta's Paradise	/wiki/Gangsta%27s_Paradise	/wiki/Coolio
4	1996	Los del Río	1.0	Macarena (song)	/wiki/Macarena_(song)	/wiki/Los_del_R%C3%ADo
5	1997	Elton John	1.0	Something About the Way You Look Tonight	/wiki/Something_About_the_Way_You_Look_Tonight	/wiki/Elton_John
6	1998	Next (group)	1.0	Too Close (Next song)	/wiki/Too_Close_(Next_song)	/wiki/Next_(group)
7	1999	Cher	1.0	Believe (Cher song)	/wiki/Believe_(Cher_song)	/wiki/Cher

To construct the dataframe, we'll need to iterate over the years and the singles per year. Notice how, above, the dataframe is ordered by ranking and then year. While the exact order is up to you, note that you will have to come up with a scheme to order the information.

Check that the dataframe has sensible data types. You will also likely find that the year field has become an "object" (Pandas treats strings as generic objects): this is due to the conversion to and back from JSON. Such conversions need special care. Fix any data type issues with `flatframe`. (See Pandas [astype (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.astype.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.astype.html) function.) We will use this `flatframe` in the next question.

(As an aside, we used the name `flatframe` to indicate that this dataframe is flattened from a hierarchical dictionary structure with the keys being the years.)

In [7]:
```python
# Create the DataFrame
flatframe = pd.DataFrame()
joinlist = ['band_singer', 'song', 'songurl', 'url']
for y in range(1992, 2015):
    yearframe = pd.DataFrame.from_records(yearinfo[str(y)])
    yearframe['year'] = y
    del yearframe['titletext']
    # Join the list of various lenth into one string before storing in the DataFrame
    # This makes data analysis easilier (otherwise can't use function like value_counts())
    yearframe[joinlist] = yearframe[joinlist].applymap(lambda l: ', '.join(filter(None, l)))
    flatframe = flatframe.append(yearframe)
flatframe.head(8)
```

Out[7]:

	band_singer	ranking	song	songurl	url	year
0	Boyz II Men	1	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men	1992
1	Sir Mix-a-Lot	2	Baby Got Back	/wiki/Baby_Got_Back	/wiki/Sir_Mix-a-Lot	1992
2	Kris Kross	3	Jump	/wiki/Jump	/wiki/Kris_Kross	1992
3	Vanessa Williams	4	Save the Best for Last	/wiki/Save_the_Best_for_Last	/wiki/Vanessa_L._Williams	1992
4	TLC	5	Baby-Baby-Baby	/wiki/Baby-Baby-Baby	/wiki/TLC_(band)	1992
5	Eric Clapton	6	Tears in Heaven	/wiki/Tears_in_Heaven	/wiki/Eric_Clapton	1992
6	En Vogue	7	My Lovin' (You're Never Gonna Get It)	/wiki/My_Lovin%27_(You%27re_Never_Gonna_Get_It)	/wiki/En_Vogue	1992
7	Red Hot Chili Peppers	8	Under the Bridge	/wiki/Under_the_Bridge	/wiki/Red_Hot_Chili_Peppers	1992

In [8]:
```python
# Rearange the DataFrame columns and re-index based on sorting
flatframe = flatframe[['year', 'band_singer', 'ranking', 'song', 'songurl', 'url']]
flatframe = flatframe.sort_values(by=['ranking', 'year'])
flatframe.index = range(len(flatframe))
flatframe.head(8)
```

Out[8]:

	year	band_singer	ranking	song	songurl	url
0	1992	Boyz II Men	1	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men
1	1993	Whitney Houston	1	I Will Always Love You	/wiki/I_Will_Always_Love_You#Whitney_Houston_v...	/wiki/Whitney_Houston
2	1994	Ace of Base	1	The Sign	/wiki/The_Sign_(song)	/wiki/Ace_of_Base
3	1995	Coolio, L.V.	1	Gangsta's Paradise	/wiki/Gangsta%27s_Paradise	/wiki/Coolio, /wiki/L.V._(singer)
4	1996	Los del Río	1	Macarena (Bayside Boys Mix)	/wiki/Macarena_(song)	/wiki/Los_del_R%C3%ADo
5	1997	Elton John	1	Something About the Way You Look Tonight, Cand...	/wiki/Something_About_the_Way_You_Look_Tonight...	/wiki/Elton_John
6	1998	Next	1	Too Close	/wiki/Too_Close_(Next_song)	/wiki/Next_(group)
7	1999	Cher	1	Believe	/wiki/Believe_(Cher_song)	/wiki/Cher

Who are the highest quality singers?

```
In [9]:  # For multiple singers, find the first singer (main).
         # This makes the counts more accurate
         flatframe['first_singer'] = flatframe['band_singer'].apply(lambda n: n.split(', ')[0])
         flatframe.head(8)
```

Out[9]:

	year	band_singer	ranking	song	songurl	url	first_singer
0	1992	Boyz II Men	1	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men	Boyz II Men
1	1993	Whitney Houston	1	I Will Always Love You	/wiki/I_Will_Always_Love_You#Whitney_Houston_v...	/wiki/Whitney_Houston	Whitney Houston
2	1994	Ace of Base	1	The Sign	/wiki/The_Sign_(song)	/wiki/Ace_of_Base	Ace of Base
3	1995	Coolio, L.V.	1	Gangsta's Paradise	/wiki/Gangsta%27s_Paradise	/wiki/Coolio, /wiki/L.V._(singer)	Coolio
4	1996	Los del Río	1	Macarena (Bayside Boys Mix)	/wiki/Macarena_(song)	/wiki/Los_del_R%C3%ADo	Los del Río
5	1997	Elton John	1	Something About the Way You Look Tonight, Cand...	/wiki/Something_About_the_Way_You_Look_Tonight...	/wiki/Elton_John	Elton John
6	1998	Next	1	Too Close	/wiki/Too_Close_(Next_song)	/wiki/Next_(group)	Next
7	1999	Cher	1	Believe	/wiki/Believe_(Cher_song)	/wiki/Cher	Cher

Here we show the highest quality singers and plot them on a bar chart.

1.5 Find highest quality singers according to how prolific they are

What do we mean by highest quality? This is of course open to interpretation, but let's define "highest quality" here as the number of times a singer appears in the top 100 over this time period. If a singer appears twice in a year (for different songs), this is counted as two appearances, not one.

Make a bar-plot of the most prolific singers. Singers on this chart should have appeared at-least more than 15 times. (HINT: look at the docs for the pandas method `value_counts`.)

```
In [10]:  singer_appear_count = flatframe['first_singer'].value_counts()
          singer_appear_count = singer_appear_count[singer_appear_count >= 15].sort_values()
```

```
In [11]:  sns.reset_defaults()
          sns.set_context("notebook")

          count = singer_appear_count.values
          num = np.arange(len(singer_appear_count))
          singer_appear_count.plot.barh()
          ax = plt.gca()
          plt.ylabel("Appearence")
          for c, n in zip(count, num):
              plt.annotate(str(c), xy=(c+0.1, n), va='center')
          xt = np.arange(0, count.max(), 5)
          plt.xticks(xt, [''] * len(xt))
          plt.grid(axis='x', color='white', linestyle='-')
          ax.tick_params(axis='both', which='both', length=0)
          sns.despine(left=True, bottom=True)

          plt.show()
```
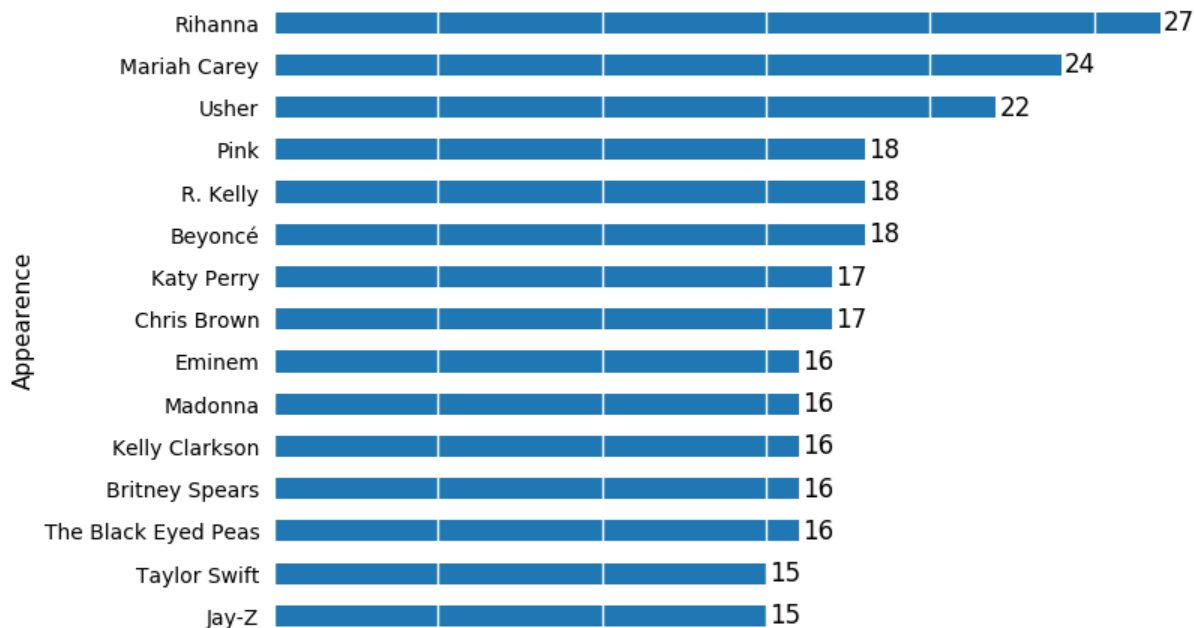

1.6 What if we used a different metric?

What we would like to capture is this: a singer should to be scored higher if the singer appears higher in the rankings. So we'd say that a singer who appeared once at a higher and once at a lower ranking is a "higher quality" singer than one who appeared twice at a lower ranking.

To do this, group all of a singers songs together and assign each song a score `101 - ranking`. Order the singers by their total score and make a bar chart for the top 20.

```
In [12]:  singer_group = flatframe.groupby('first_singer')
```

```
In [13]:  score_dict = dict()
          for name, group in singer_group:
              score_dict[name] = 101 * len(group) - group['ranking'].agg(np.sum)
          scoreframe = pd.DataFrame.from_dict(score_dict, orient='index')
          scoreframe.columns = ['score']
          scoreframe = scoreframe.sort_values(by='score', ascending=False)
```

```
In [14]:  score_top20 = scoreframe[:20].sort_values(by='score')
          score = score_top20['score']
```

In [15]:
```python
sns.reset_defaults()
sns.set_context("notebook")

num = np.arange(len(score_top20))
score_top20.plot.barh()

ax = plt.gca()
plt.ylabel("Quality Score")
for s, n in zip(score, num):
    plt.annotate(str(s), xy=(s+10, n), va='center')
xt = np.arange(0, score.max(), 200)
plt.xticks(xt, [''] * len(xt))
plt.grid(axis='x', color='white', linestyle='-')
ax.legend_.remove()
ax.tick_params(axis='both', which='both', length=0)
sns.despine(left=True, bottom=True)

plt.show()
```
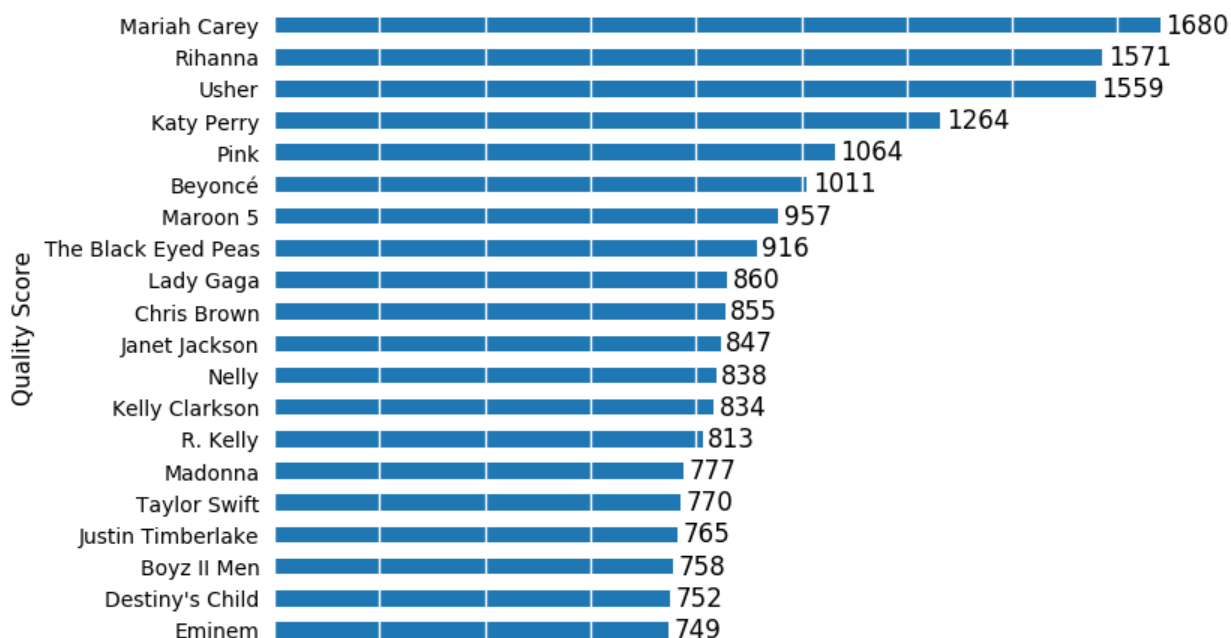

1.7 Do you notice any major differences when you change the metric?

How have the singers at the top shifted places? Why do you think this happens?

Some singers move up into the top list with less than 15 times appearence in the Billboard 100. This is simply because their songs rank on average higher (i.e. higher score) in the Billboard 100

Q2. Scraping and Constructing: Information about Artists, Bands and Genres from Wikipedia

Our next job is to use those band/singer urls we collected under `flatframe.url` and get information about singers and/or bands.

Scrape information about artists from wikipedia

We wish to fetch information about the singers or groups for all the winning songs in a list of years.

Here we show a function that fetches information about a singer or group from their url on wikipedia. We create a cache object `urlcache` that will avoid redundant HTTP requests (e.g. an artist might have multiple singles on a single year, or be on the list over a span of years). Once we have fetched information about an artist, we don't need to do it again. The caching also helps if the network goes down, or the target website is having some problems. You simply need to run the `get_page` function below again, and the `urlcache` dictionary will continue to be filled.

If the request gets an HTTP return code different from 200, (such as a 404 not found or 500 Internal Server Error) the cells for that URL will have a value of 1; and if the request completely fails (e.g. no network connection) the cell will have a value of 2. This will allow you to analyse the failed requests.

Notice that we have wrapped the call in whats called *an exception block*. We try to make the request. If it fails entirely, or returns a HTTP code thats not 200, we set the status to 2 and 1 respectively.

```
In [ ]: urlcache={}
```

```
In [ ]: def get_page(urls):
            if not (urls is None):
                for url in urls.split(', '):
                    #print(url)
                    # Check if URL has already been visited.
                    if (url not in urlcache) or (urlcache[url]==1) or (urlcache[url]==2):
                        time.sleep(1)
                        # try/except blocks are used whenever the code could generate an exception (e.g. division by zero).
                        # In this case we don't know if the page really exists, or even if it does, if we'll be able to reach
                        try:
                            r = requests.get("http://en.wikipedia.org%s" % url)

                            if r.status_code == 200:
                                #print("Success")
                                urlcache[url] = r.text
                            else:
                                print(r.status_code)
                                urlcache[url] = 1
                        except:
                            print("Fail")
                            urlcache[url] = 2
            return urlcache[url]
```

We sort the `flatframe` by year, ascending, first. Think why.

```
In [ ]: flatframe=flatframe.sort_values('year')
        flatframe.head(8)
```

Pulling and saving the data

```
In [ ]: # DO NOT RERUN THIS CELL WHEN SUBMITTING
        # Here we are populating the url cache
        # subsequent calls to this cell should be very fast, since Python won't
        # need to fetch the page from the web server.
        # NOTE this function will take quite some time to run (about 30 mins for me), since we sleep 1 second before
        # making a request. If you run it again it will be almost instantaneous, save requests that might have failed
        # (you will need to run it again if requests fail..see cell below for how to test this)
        flatframe["url"].apply(get_page)
```

You may have to run this function again and again, in case there were network problems. Note that, because there is a "global" cache, it will take less time each time you run it. Also note that this function is designed to be run again and again: it attempts to make sure that there are no unresolved pages remaining. Let us make sure of this: *the sum below should be 0, and the boolean True.*

```
In [ ]: # DO NOT RERUN THIS CELL WHEN SUBMITTING
        print("Number of bad requests:",np.sum([(urlcache[k]==1) or (urlcache[k]==2) for k in urlcache])) # no one or 0's)
        print("Did we get all urls?", len(flatframe.url.unique())==len(urlcache)) # we got all of the urls
```

```
In [ ]: len(urlcache)
```

Let's save the `urlcache` to disk, just in case we need it again.

```
In [ ]: # DO NOT RERUN THIS CELL WHEN SUBMITTING
        with open("data/artistinfo.json","w") as fd:
            json.dump(urlcache, fd)
        del urlcache
```

2.1 Extract information about singers and bands

From each page we collected about a singer or a band, extract the following information:

1. If the page has the text "Born" in the sidebar on the right, extract the element with the class `.bday`. If the page doesn't contain "Born", store `False`. Store either of these into the variable `born`. We want to analyze the artist's age.
2. If the text "Years active" is found, but no "born", assume a band. Store into the variable `ya` the value of the next table cell corresponding to this, or `False` if the text is not found.

Put this all into a function `singer_band_info` which takes the singer/band url as argument and returns a dictionary `dict(url=url, born=born, ya=ya)`.

The information can be found on the sidebar on each such wikipedia page, as the example here shows:

Write the function `singer_band_info` according to the following specification:

```
In [ ]:  """
         Function
         --------
         singer_band_info

         Inputs
         ------
         url: the url
         page_text: the text associated with the url

         Returns
         -------
         A dictionary with the following data:
             url: copy the input argument url into this value
             born: the artist's birthday
             ya: years active variable

         Notes
         -----
         See description above. Also note that some of the genres urls might require a
         bit of care and special handling.
         """
```

```
In [17]:  def singer_band_info(url, page_text):
              info_dict = {'url': url}
              soup = BeautifulSoup(page_text, 'html.parser')
              infobox = soup.find("table", "infobox")
              if infobox: # See B-Rock and the Bizz in Billboard 1997 rank 68 - no box
                  rows = [row for row in infobox.find_all("tr")]
                  rows_text = []
                  for row in rows:
                      has_text = row.find("th")
                      if has_text:
                          rows_text.append(has_text.get_text())
                      else:
                          rows_text.append('')
                  # Extract born and years active
                  if 'Born' in rows_text:
                      born_ind = rows_text.index('Born')
                      born_text = rows[born_ind].find("td").get_text()
                      born_splt = born_text.split("(")
                      if len(born_splt) > 1: # See Donna Lewis in Billboard 1997 Rank 64 - has born tag but no date
                          born = born_splt[1].split(")")[0] # Get a format of xxxx-xx-xx
                          if len(born) != 10:
                              born = False
                      else:
                          born = False
                  else:
                      born = False
                  info_dict['born'] = born
                  if 'Years active' in rows_text:
                      ya_ind = rows_text.index('Years active')
                      ya = rows[ya_ind].find("td").get_text()
                  elif 'Years\xa0active' in rows_text:
                      ya_ind = rows_text.index('Years\xa0active') # Some of the Years active string are different
                      ya = rows[ya_ind].find("td").get_text()
                  else:
                      ya = False
                  if ya and ('\n' in ya):
                      ya = ya.replace('\n', '')
                  info_dict['ya'] = ya
              else:
                  info_dict['born'] = False
                  info_dict['ya'] = False
              return info_dict
```

```
In [18]:  if '' in urlcache:
              del urlcache['']
```

2.2 Merging this information in

Iterate over the items in the singer-group dictionary cache `urlcache`, run the above function, and create a dataframe from there with columns

url, born, and ya. Merge this dataframe on the url key with `flatframe`, creating a rather wide dataframe that we shall call `largedf`. It should look something like this:

	year	band_singer	ranking	song	songurl	url	born	ya
0	1992	Boyz II Men	1.0	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men	False	1985-prese
1	1992	Boyz II Men	37.0	It's So Hard to Say Goodbye to Yesterday	/wiki/It%27s_So_Hard_to_Say_Goodbye_to_Yesterday	/wiki/Boyz_II_Men	False	1985-prese
2	1992	Boyz II Men	84.0	Uhh Ahh	/wiki/Uhh_Ahh	/wiki/Boyz_II_Men	False	1985-prese
3	1993	Boyz II Men	12.0	In the Still of the Night (1956 song)	/wiki/In_the_Still_of_the_Night_(1956_song)#Bo...	/wiki/Boyz_II_Men	False	1985-prese
4	1994	Boyz II Men	3.0	I'll Make Love to You	/wiki/I%27ll_Make_Love_to_You	/wiki/Boyz_II_Men	False	1985-prese

Notice how the born and ya and url are repeated every time a different song from a given band is represented in a row.

In [19]:
```python
flatframe['born'] = False
flatframe['ya'] = False
flatframe.head(8)
```

Out[19]:

	year	band_singer	ranking	song	songurl	url	first_singer	born	ya
0	1992	Boyz II Men	1	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men	Boyz II Men	False	False
1	1993	Whitney Houston	1	I Will Always Love You	/wiki/I_Will_Always_Love_You#Whitney_Houston_v...	/wiki/Whitney_Houston	Whitney Houston	False	False
2	1994	Ace of Base	1	The Sign	/wiki/The_Sign_(song)	/wiki/Ace_of_Base	Ace of Base	False	False
3	1995	Coolio, L.V.	1	Gangsta's Paradise	/wiki/Gangsta%27s_Paradise	/wiki/Coolio, /wiki/L.V._(singer)	Coolio	False	False
4	1996	Los del Río	1	Macarena (Bayside Boys Mix)	/wiki/Macarena_(song)	/wiki/Los_del_R%C3%ADo	Los del Río	False	False
5	1997	Elton John	1	Something About the Way You Look Tonight, Cand...	/wiki/Something_About_the_Way_You_Look_Tonight...	/wiki/Elton_John	Elton John	False	False
6	1998	Next	1	Too Close	/wiki/Too_Close_(Next_song)	/wiki/Next_(group)	Next	False	False
7	1999	Cher	1	Believe	/wiki/Believe_(Cher_song)	/wiki/Cher	Cher	False	False

In [20]:
```python
# Instantiate a empty list to store all the info (born and ya)
info_cache = {}
```

In [21]:
```python
url_list = flatframe['url'].tolist()
for urls in flatframe['url']:
    if urls:
        url = urls.split(', ')[0]
        if url in urlcache:
            if url not in info_cache:
                page_text = urlcache[url]
                info = singer_band_info(url, page_text)
                info_cache[url] = info
                #print("%s Born: %s Years Active: %s" % (url, info['born'], info['ya']))
                ind = flatframe.index[flatframe['url'].apply(lambda s: s.split(', ')[0].find(url) == 0)].tolist()
                flatframe.loc[ind, ['born', 'ya']] = [info['born'], info['ya']]
        else:
            print("No such url!: %s" % url)
```

```
In [22]:  flatframe.sort_values(by='ranking').head(8)
```

Out[22]:

	year	band_singer	ranking	song	songurl	url	first_singer	born	ya
0	1992	Boyz II Men	1	End of the Road	/wiki/End_of_the_Road	/wiki/Boyz_II_Men	Boyz II Men	False	1985–present
22	2014	Pharrell Williams	1	Happy	/wiki/Happy_(Pharrell_Williams_song)	/wiki/Pharrell_Williams	Pharrell Williams	1973-04-05	1992–present
21	2013	Macklemore, Ryan Lewis, Wanz	1	Thrift Shop	/wiki/Thrift_Shop	/wiki/Macklemore, /wiki/Ryan_Lewis, /wiki/Wanz	Macklemore	1983-06-19	2000–present
20	2012	Gotye, Kimbra	1	Somebody That I Used to Know	/wiki/Somebody_That_I_Used_to_Know	/wiki/Gotye, /wiki/Kimbra	Gotye	1980-05-21	2001–present
19	2011	Adele	1	Rolling in the Deep	/wiki/Rolling_in_the_Deep	/wiki/Adele	Adele	1988-05-05	2006–present
18	2010	Kesha	1	Tik Tok	/wiki/Tik_Tok	/wiki/Kesha	Kesha	1987-03-01	2005–present
17	2009	The Black Eyed Peas	1	Boom Boom Pow	/wiki/Boom_Boom_Pow	/wiki/The_Black_Eyed_Peas	The Black Eyed Peas	False	1992–present(Hiatus: 2007–2009; 2012–2015)
15	2007	Beyoncé	1	Irreplaceable	/wiki/Irreplaceable	/wiki/Beyonc%C3%A9	Beyoncé	1981-09-04	1997–present

2.3 What is the age at which singers achieve their top ranking?

Plot a histogram of the age at which singers achieve their top ranking. What conclusions can you draw from this distribution of ages?

HINT: You will need to do some manipulation of the born column, and find the song for which a band or an artist achieves their top ranking. You will then need to put these rows together into another dataframe or array to make the plot.

```
In [23]:  ff_w_born = flatframe.copy()
          ff_w_born = ff_w_born[ff_w_born['born'] != False]
          # Get the birth year to calculate age
          ff_w_born['byear'] = ff_w_born['born'].apply(lambda s: int(s.split('-')[0]))
```

```
In [24]:  singer_w_born = ff_w_born.groupby('first_singer')
          singer_w_born_dict = dict()

          for name, group in singer_w_born:
              top_rank = group['ranking'].agg(np.min)
              top_group = group[group['ranking'] == top_rank]
              top_year = top_group['year'].agg(np.max)
              byear = top_group['byear'].tolist()[0] # Index doesn't matter because they are all the same
              top_age = top_year - byear
              singer_w_born_dict[name] = top_age

          singer_top_frame = pd.DataFrame.from_dict(singer_w_born_dict, orient='index')
          singer_top_frame.columns = ['top_age']
          singer_top_frame = singer_top_frame.sort_values(by='top_age')
          singer_top_frame.head(8)
```

Out[24]:

|	top_age
Lil' Romeo	12
Lil' Bow Wow	13
Sammie	13
JoJo	14
LeAnn Rimes	16
Tevin Campbell	16
Jibbs	16
Lorde	17

```
In [25]: sns.reset_defaults()
         sns.set_style("whitegrid")
         sns.set_context("notebook")

         top_age = singer_top_frame['top_age']
         singer_top_frame.plot.hist(bins=24)

         ax = plt.gca()
         plt.title("Age at which singer reach top ranking");
         plt.ylabel("Frequency")
         plt.xlabel("Age")
         xt = np.arange(10, 60, 5)
         plt.xticks(xt)
         ax.legend_.remove()

         age_percent = [top_age.quantile(.1), top_age.quantile(.25), top_age.mean(), top_age.quantile(.75), top_age.quantile(.9
         for a in age_percent:
             plt.axvline(a, 0, .9, color='r', ls='--');
             plt.annotate('%.1f' % a, xy=(a-1, 62), va='center')

         plt.show()
```
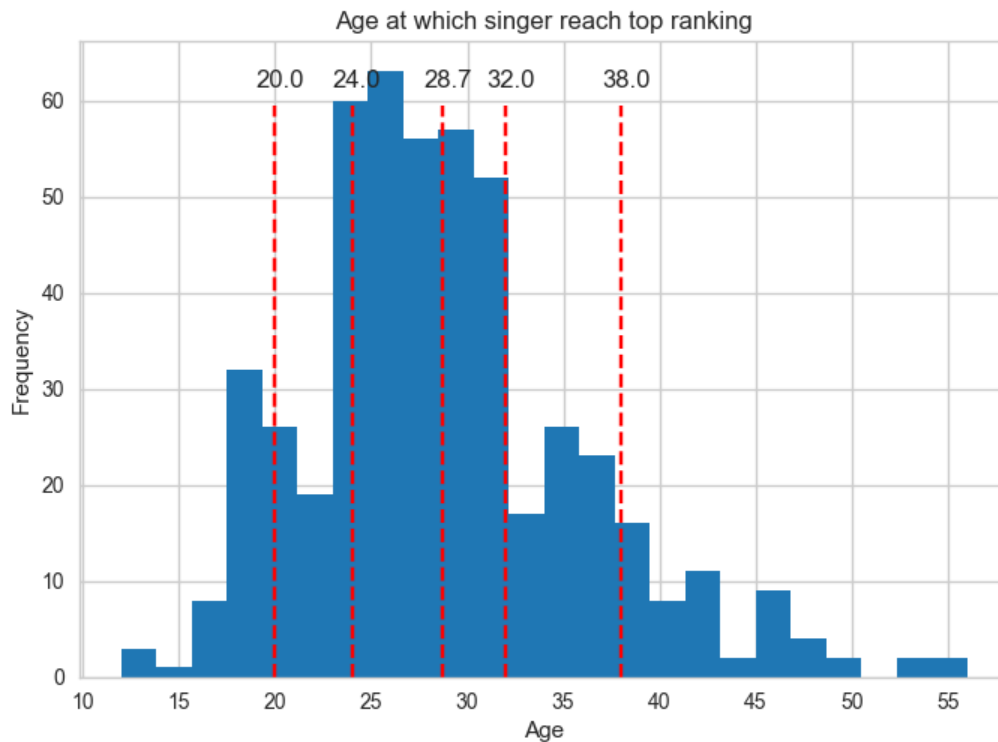

Conclusion: Most of the singers reach their personal top ranking at the age between 20 and 38 (80%), with 50% of them are within the age of 24 to 32 (50%)

2.4 At what year since inception do bands reach their top rankings?

Make a similar calculation to plot a histogram of the years since inception at which bands reach their top ranking. What conclusions can you draw?

```
In [26]: bandframe = flatframe.copy()
         bandframe = bandframe[(bandframe['born'] == False) & (bandframe['ya'] != False)]
         # Get the year of band formation
         bandframe['inception'] = bandframe['ya'].apply(lambda s: int(s.split('-')[0][:4]))
         band_w_ya = bandframe.groupby('first_singer')
         band_dict = dict()

         for name, group in band_w_ya:
             top_rank = group['ranking'].agg(np.min)
             top_group = group[group['ranking'] == top_rank]
             top_year = top_group['year'].agg(np.max)
             inception = top_group['inception'].tolist()[0] # Index doesn't matter because they are all the same
             top_age = top_year - inception
             band_dict[name] = top_age

         band_top_frame = pd.DataFrame.from_dict(band_dict, orient='index')
         band_top_frame.columns = ['top_age']
         band_top_frame = band_top_frame.sort_values(by='top_age')
```

```
In [27]: sns.reset_defaults()
         sns.set_style("whitegrid")
         sns.set_context("notebook")

         top_age = band_top_frame['top_age']
         num = np.arange(len(band_top_frame))
         band_top_frame.plot.hist(bins=24)

         ax = plt.gca()
         plt.title("Years since formation to reach top ranking");
         plt.ylabel("Frequency")
         plt.xlabel("Years")
         xt = np.arange(0, 50, 5)
         plt.xticks(xt)
         ax.legend_.remove()

         age_percent = [top_age.quantile(.5), top_age.quantile(.8)]
         for a in age_percent:
             plt.axvline(a, 0, .9, color='r', ls='--');
             plt.annotate('%.1f' % a, xy=(a-1, 78), va='center')

         plt.show()
```


Years since formation to reach top ranking
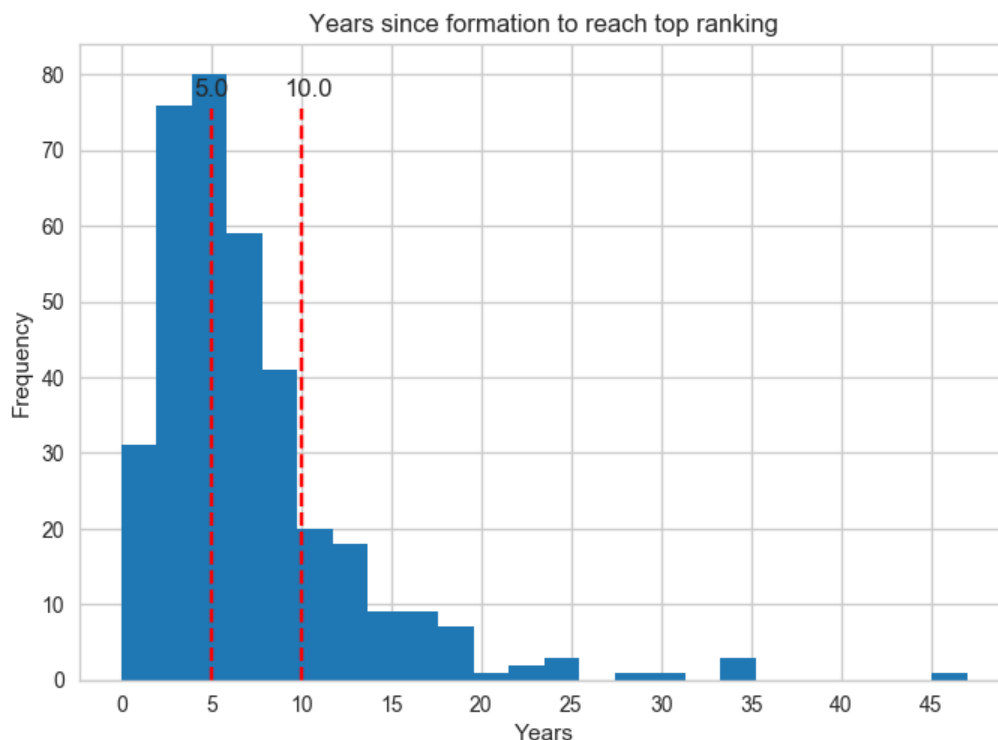
Conclusion: Most of the bands reach their top ranking within 10 years of their formation (80%). 50% of the bands achieve this in 5 years.