Walter Thornton working with Dwayne Kennemore

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 pl
import pandas as pd
import time
import requests
import urllib.request
from bs4 import BeautifulSoup
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`.

In [2]:
```
#retrieves our data from wikipedia
with urllib.request.urlopen('https://en.wikipedia.org/wiki/Billboard_Year-End_Hot
   wiki_chart = response.read()
soup = BeautifulSoup(wiki_chart, "lxml")
tables = soup.find('table', attrs={'class':'wikitable sortable'})

#scrapes the page for relevant information
song_list = []
tr_list = tables.find_all('tr')
for tr in tr_list:
    td_list = tr.find_all('td')
    if td_list == [] :
        td_list = []
    else :
        ranking = td_list[0].get_text()
        title = td_list[1].get_text()
        band_singer = td_list[2].get_text()
        soup_of_link = BeautifulSoup(str(td_list), "lxml")
        url = td_list[2].a["href"]
        #creates a dictionary entry for the scraped data
        dict_entry = {'band_singer' : band_singer,
        'ranking' : ranking,
        'title' : title,
        'url' : url}
        #appends our new dictionary to our song_list
        song_list.append(dict_entry)
```

In [3]:
```
#sanity check
song_list[2:4]
```

Out[3]:
```
[{'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'}]
```

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'}]
```

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.

In [4]:
```
#retrieves full webpages from wikipedia saving each to a dictionary keyed by year
yearstext = {}
for year in range(1992, 2015):
    with urllib.request.urlopen(f'https://en.wikipedia.org/wiki/Billboard_Year-En
        year_text = {year : response.read()}
        yearstext.update(year_text)
        time.sleep(1)
```

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

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 [5]:
```
"""
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 
    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
        (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 par
        of the url from /wiki onwards


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





def parse_year(the_year, yeartext_dict):
    year = the_year
    yearinfo = []
    song = []
    songurl = []
    band_singer = []
    title = []
    url = []
    title_text = ''
```

```
        i = 0
        title_string = ''
        band_singer = ''
        soup = BeautifulSoup(yearstext[year], "lxml")
        tables = soup.find('table', attrs={'class':'wikitable sortable'})
        #iterates through tree structure, scraping our data
        tr_list = tables.find_all('tr')
        for tr in tr_list:
            td_list = tr.find_all('td')
            if td_list == [] :
                td_list = []
            else :
                ranking = tr.th.string
                links = tr.td.findAll('a')
                number_of_links = len(links)
                if number_of_links == 0:
                    songurl = [None]
                    title_text = [a['title']]
                    song.append(a['title'] )
                else :
                    i = 0
                    for a in tr.td.findAll('a') :
                        song.append(a['title'] )
                        title_string = '\"' + a['title'] + '\"'
                        if i == 0 :
                            title_text = title_string
                            i = i + 1
                        else :
                            title_text = title_text + ' / ' + title_string
                            i = i + 1
                        songurl.append(a['href'])
                title = song
                #finds next td tag
                tr.td.findNext('td')
                temp = len(tr.td.findNext('td').findAll('a'))
                if temp == 0:
                    singer_url = [None]
                    band_singer = tr.td.findNext('td').string
                    band_singer = [band_singer]
                elif temp == 1:
                    singer_url = tr.td.findNext('td').a['href']
                    singer_url = [singer_url]
                    band_singer = tr.td.findNext('td').a.string
                    band_singer = [band_singer]
                else:
                    singer_url = []
                    band_singer = []
                    for a in tr.td.findNext('td').findAll('a'):
                        webpage = a['href']
                        singer_url.append(webpage)
                        band_singer.append(a.string)
                #creates dictionary entry
                dict_entry = {'band_singer' : band_singer,
                'ranking' : ranking,
                'song' : title, 'songurl': songurl, 'titletext' : title_text,
                'url' : singer_url}
                #appends new dictionary to our list
                yearinfo.append(dict_entry)
                songurl = []
                song = []
                title_string = ''
                title_text = ''
        return(yearinfo)
```

In [6]:
```
#parse each year
years_dict = {}
for year in range(1992, 2015):
    years_dict.update({year : parse_year(year, yearstext)})
```

In [7]:
```
#assign the complete dictionary back to yearinfo
yearinfo = years_dict
```

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.

In [8]:
```
# sanity check
parse_year(1997, yearstext)[:5]
```

Out[8]:
```
[{'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 Win
d 1997"',
  'url': ['/wiki/Elton_John']},
 {'band_singer': ['Jewel'],
  'ranking': '2',
  'song': ['Foolish Games', 'You Were Meant for Me (Jewel song)'],
  'songurl': ['/wiki/Foolish_Games',
   '/wiki/You_Were_Meant_for_Me_(Jewel_song)'],
  'titletext': '"Foolish Games" / "You Were Meant for Me (Jewel song)"',
  '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']}]
```

```
[{'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 i
n 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 [9]:
```
import json
```

In [10]:
```
# DO NOT RERUN THIS CELL WHEN SUBMITTING
fd = open("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 [11]:
```
# RERUN WHEN SUBMITTING
# Another way to deal with files. Has the advantage of closing the file for you.
with open("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

In [12]:
```
# turns our list into a list of dictionaries where each entry contains a year key
rows = []
for year in yearinfo.keys():
    for song in yearinfo[year]:
        song['year'] = year
        rows.append(song)
```

```
In [13]: # sanity check
         rows
```

```
Out[13]: [{'band_singer': ['Boyz II Men'],
           'ranking': '1',
           'song': ['End of the Road'],
           'songurl': ['/wiki/End_of_the_Road'],
           'titletext': '"End of the Road"',
           'url': ['/wiki/Boyz_II_Men'],
           'year': '1992'},
          {'band_singer': ['Sir Mix-a-Lot'],
           'ranking': '2',
           'song': ['Baby Got Back'],
           'songurl': ['/wiki/Baby_Got_Back'],
           'titletext': '"Baby Got Back"',
           'url': ['/wiki/Sir_Mix-a-Lot'],
           'year': '1992'},
          {'band_singer': ['Kris Kross'],
           'ranking': '3',
           'song': ['Jump'],
           'songurl': ['/wiki/Jump'],
           'titletext': '"Jump"',
```

In [14]:
```
# TO RERUN THIS CELL, you must restart and clear all output!!!

# the following code removes lists from inside of our dictionaries inside of our
# for each individual list item
band_singer = []
songurl = ''
title_text = ''
singer_url = []
starting_length = len(rows)
for dics in rows:
    if starting_length == 0:
        break
    dict_add = {}
    # checks if our dictionary contains lists longer than one element
    if len(dics['band_singer']) > 1:
        i = 0
        j = len(dics['band_singer'])

        for value in dics['band_singer']:
            # new dictionary entry to append to our list
            dict_add = {'band_singer' : dics['band_singer'][i],
            'ranking' : dics['ranking'],
            'song' : dics['song'], 'songurl': dics['songurl'], 'titletext' : dics
            'url' : dics['url'][i], 'year' : dics['year']}
            rows.append(dict_add)
            i = 1 + i
            j = j - 1
    starting_length = starting_length - 1

rows2 = []
band_singer = []

# loops through our list again, removing duplicate entries
for dics in rows:
    if len(dics['band_singer']) == 1 or len(dics['band_singer']) > 5:
        rows2.append(dics)

# turns all remaining one element lists into strings
for row in rows2:
    for key in row:
        row[key] = str(row[key])
        row[key] = row[key].strip("[]")
        row[key] = row[key].strip("'")
```

In [190]:
```
rows2
```

```
  'titletext': '"Baby Got Back"',
  'url': '/wiki/Sir_Mix-a-Lot',
  'year': '1992'},
 {'band_singer': 'Kris Kross',
  'ranking': '3',
  'song': 'Jump',
  'songurl': '/wiki/Jump',
  'titletext': '"Jump"',
  'url': '/wiki/Kris_Kross',
  'year': '1992'},
 {'band_singer': 'Vanessa Williams',
  'ranking': '4',
  'song': 'Save the Best for Last',
  'songurl': '/wiki/Save_the_Best_for_Last',
  'titletext': '"Save the Best for Last"',
  'url': '/wiki/Vanessa_L._Williams',
  'year': '1992'},
 {'band_singer': 'TLC',
  'ranking': '5',
  'song': 'Baby-Baby-Baby',
```

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) 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 [15]:
```
# create dataframe
flatframe = pd.DataFrame(rows2)


# check datatypes of dataframe columns
flatframe['year'].dtype
```

Out[15]:
```
dtype('O')
```

In [16]:
```
# sanity check
flatframe
```

8	Color Me Badd	9	All 4 Love	/wiki/All_4_Love	"All 4
9	Jon Secada	10	Just Another Day (Jon Secada song)	/wiki/Just_Another_Day_(Jon_Secada_song)	Anoth (Jon S
10	Shanice	11	I Love Your Smile	/wiki/I_Love_Your_Smile	"I Lov
11	Mr. Big	12	To Be with You	/wiki/To_Be_with_You	"To B
12	Right Said Fred	13	"I'm Too Sexy"	/wiki/I%27m_Too_Sexy	"I
13	Michael Jackson	14	Black or White	/wiki/Black_or_White	"B
14	Billy Ray		Achy		

In [17]:
```
# convert year column to int
flatframe['year'] = flatframe['year'].astype(np.uint16)
```

Who are the highest quality singers?

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`.)

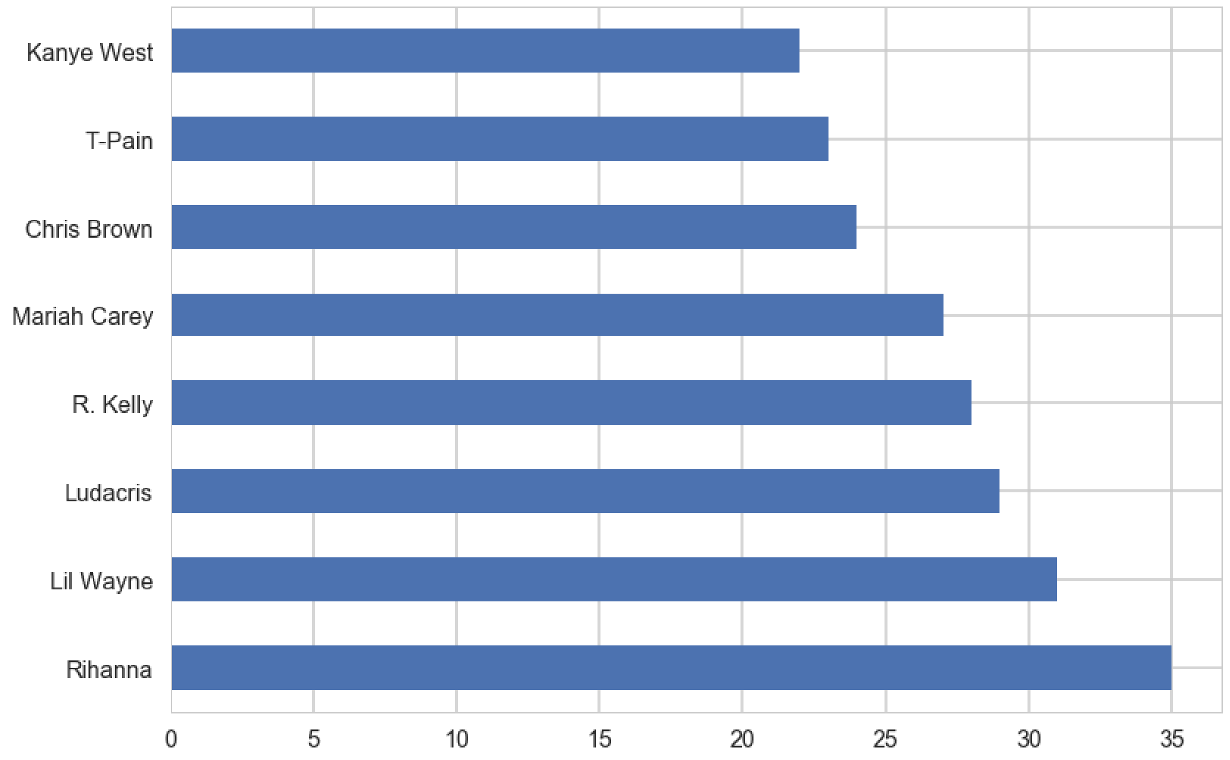
In [18]:
```
# count artist appearences in top 100



artist_count = flatframe["band_singer"].value_counts()
```

```
In [19]: # plot top artists
         artist_count[:8].plot.barh()
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1c89dc5b978>
```



1.6 What if we used a different metric?

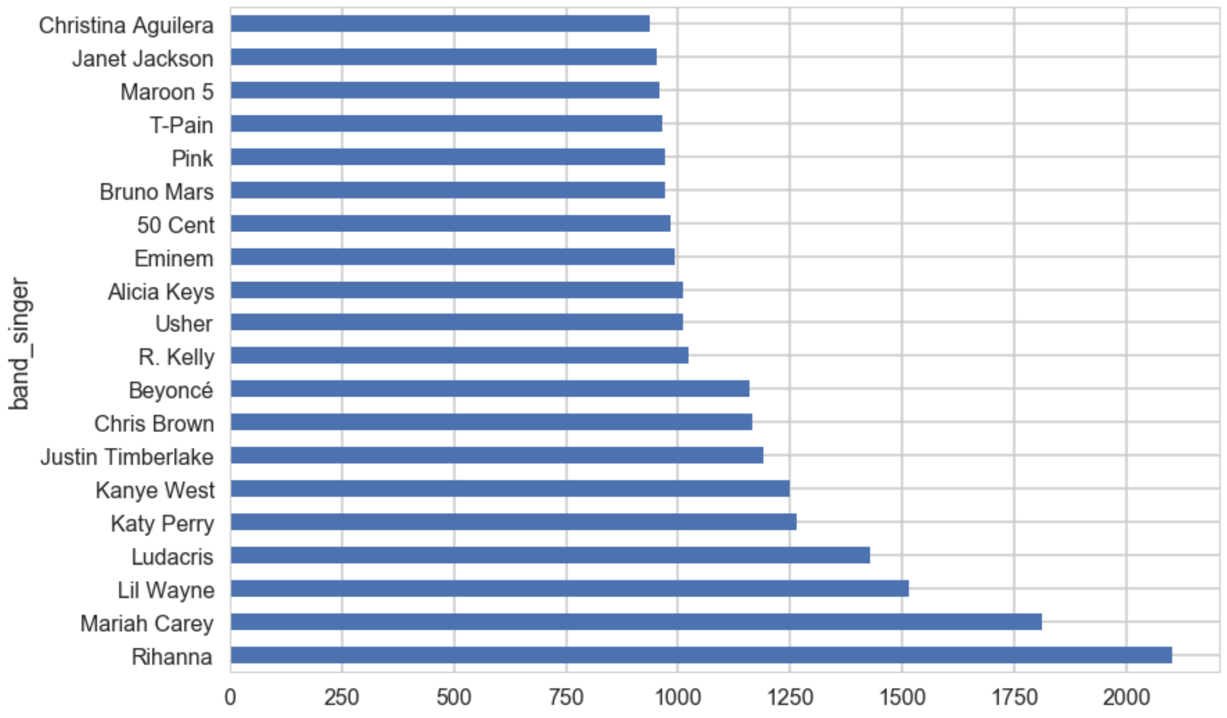
In [20]:
```
scored_df = pd.DataFrame(np.array(flatframe[['band_singer', 'song', 'ranking']]))
scored_df.columns = ['band_singer', 'song', 'ranking']
# scored_df.groupby(['band_singer', 'song'])['ranking']

scored_df['ranking'] = scored_df['ranking'].apply(lambda x: 101 - int(x))

#display(scored_df)

scored_df.groupby('band_singer')['ranking'].aggregate(np.sum).sort_values(ascendi
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c89e109128>
```



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.

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?

In [21]:
```
"""
While Rihanna keeps her place at the head of the pack (by a sizeable margin, it s
Mariah Carey gains materially over R. Kelly, Ludacris and Lil Wayne. The reason f
Mariah's songs appear in the top 100 less frequently than those other three, the
the relative rank of her songs while the former disregards this and looks at only
relative rank also gives Katy Perry a place near the top, too.
"""
```

Out[21]:
```
"\nWhile Rihanna keeps her place at the head of the pack (by a sizeable margin,
it should be noted), \nMariah Carey gains materially over R. Kelly, Ludacris an
d Lil Wayne. The reason for this is that while\nMariah's songs appear in the to
p 100 less frequently than those other three, the latter chart weighs\nthe rela
tive rank of her songs while the former disregards this and looks at only frequ
ency. Including\nrelative rank also gives Katy Perry a place near the top, to
o.\n"
```

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 [22]:
```
urlcache={}
```

In [23]:
```
def get_page(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 exceptio
        # In this case we don't know if the page really exists, or even if it doe
        try:
            r = requests.get("http://en.wikipedia.org%s" % url)

            if r.status_code == 200:
                urlcache[url] = r.text
            else:
                urlcache[url] = 1
        except:
            urlcache[url] = 2
    return urlcache[url]
```

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

In [24]:
```
# sort the flatframe by year
flatframe = flatframe.sort_values('year')
```

Pulling and saving the data

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 [25]:
```
# 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), sin
# making a request. If you run it again it will be almost instantaneous, save req
# (you will need to run it again if requests fail..see cell below for how to test
flatframe["url"].apply(get_page)
```

In [27]:
```
# DO NOT RERUN THIS CELL WHEN SUBMITTING
print("Number of bad requests:",np.sum([(urlcache[k]==1) or (urlcache[k]==2) for
print("Did we get all urls?", len(flatframe.url.unique())==len(urlcache)) # we go
```

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

In [28]:
```
# DO NOT RERUN THIS CELL WHEN SUBMITTING
with open("artistinfo.json","w") as fd:
    json.dump(urlcache, fd)
del urlcache
```

In [29]:
```
# RERUN WHEN SUBMITTING
with open("artistinfo.json") as json_file:
    urlcache = json.load(json_file)
```

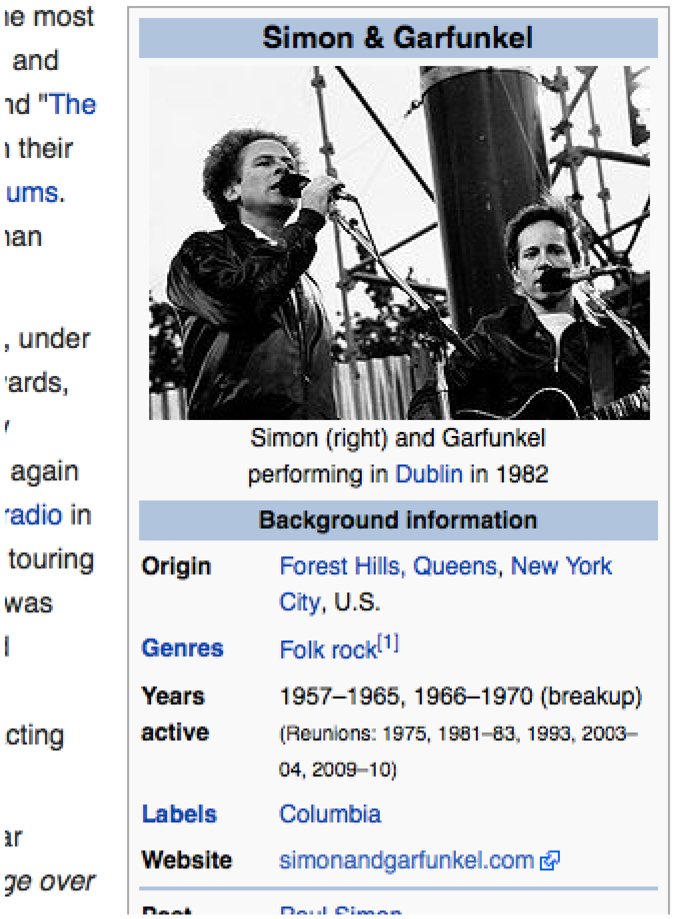
2.1 Extract information about singers and bands

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:



ıe most
and
ıd "The
ı their
ums.
ıan

, under
'ards,
'
again
'adio in
touring
was
l

cting

ar
ge over

Simon & Garfunkel

Simon (right) and Garfunkel performing in Dublin in 1982

Background information

Origin	Forest Hills, Queens, New York City, U.S.
Genres	Folk rock[1]
Years active	1957–1965, 1966–1970 (breakup) (Reunions: 1975, 1981–83, 1993, 2003–04, 2009–10)
Labels	Columbia
Website	simonandgarfunkel.com

.

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

In [30]:
```
"""
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.
"""


def singer_band_info(url, page_text):
    bday_dict = {}
    bday = ''
    ya = ''
    # soupify our webpage
    soup = BeautifulSoup(page_text[url], "lxml")
    tables = soup.find('table', attrs={'class':'infobox vcard plainlist'})
    if (tables == None):
        tables = soup.find('table', attrs={'class':'infobox biography vcard'})
    bday = tables.find('span', {'class': 'bday'})
    if bday:
        bday = bday.get_text()[:4]
        bday_dict = {'url' : url, 'born' : bday, 'ya' : ya}
    else:
        ya = False
        for tr in tables.find_all('tr'):
            if hasattr(tr.th, 'span'):
                if hasattr(tr.th.span, 'string'):
                    if tr.th.span.string == 'Years active':
                        if hasattr(tr.th, 'span'):
                            #ya = tr.td.string
                            ya = tr.td.text   #DK add
                            bday = 'False'
                            bday_dict = {'url' : url, 'born' : 'False', 'ya' : ya
    return(bday_dict)
```

In [32]:
```
# create additional dictionary with all artists additional information with their
list_of_addit_dicts = []
bday_dict = {}
for url in urlcache.keys():
    try:
        bday_dict = singer_band_info(url, urlcache)
        list_of_addit_dicts.append(bday_dict)
    except:
        break
```

In [35]:
```
#use the additional dictionary to create additional dataframe

additional_df = pd.DataFrame(list_of_addit_dicts)
```

In [37]:
```
#merge the two dataframes

largedf = pd.merge(flatframe, additional_df, left_on='url', right_on='url', how="
```

In [211]:
```
largedf
```

9	Boyz II Men	2	One Sweet Day	/wiki/One_Sweet_Day	" Sw D
10	Boyz II Men	30	4 Seasons of Loneliness	/wiki/4_Seasons_of_Loneliness	"4 Seas Lonelin
11	Boyz II Men	30	A Song for Mama	/wiki/A_Song_for_Mama	"A Song Ma
12	Boyz II Men	96	4 Seasons of Loneliness	/wiki/4_Seasons_of_Loneliness	"4 Seas Lonelin
13	Sir Mix-a-Lot	2	Baby Got Back	/wiki/Baby_Got_Back	"Baby Ba
14	Kris Kross	3	Jump	/wiki/Jump	"Ju
15	[illegible]	[illegible]	Warm It	[illegible]	"War

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.

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?

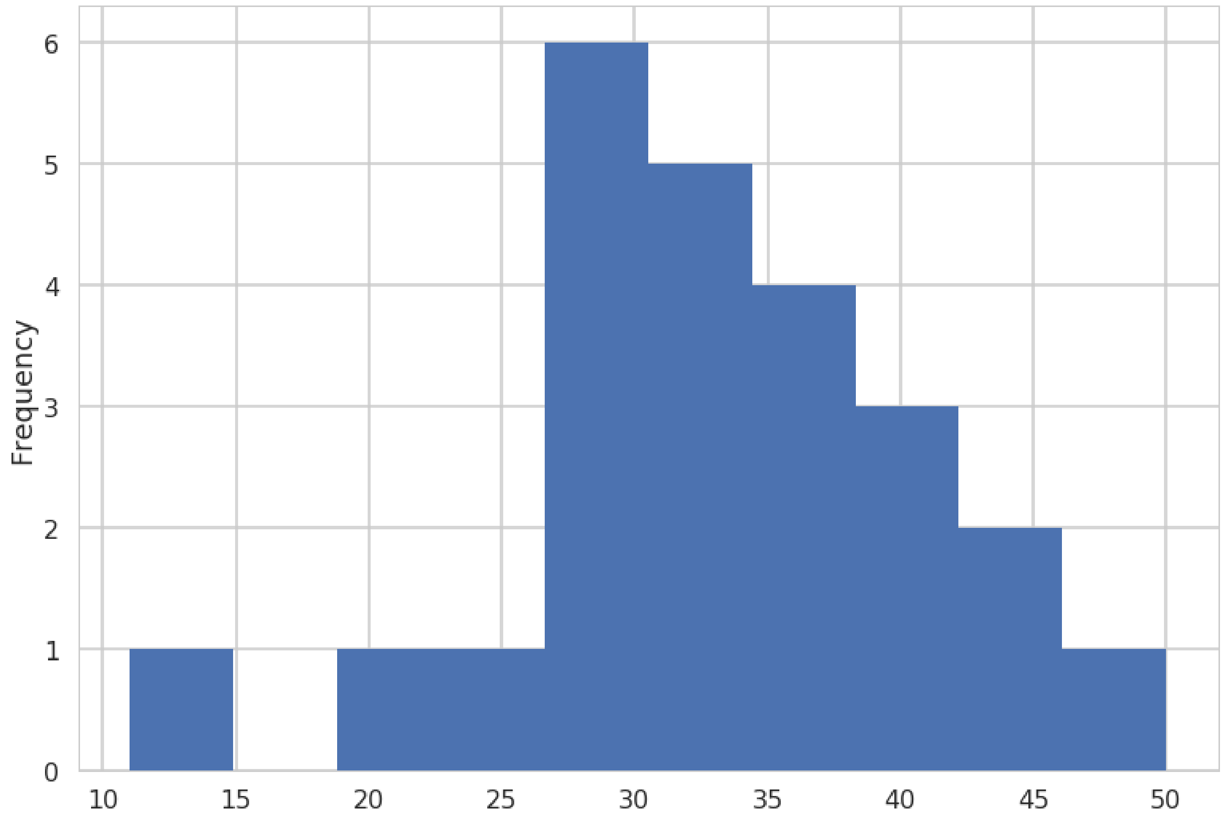
HINT: You will need to do some manipulation of the `born` column, and find the song for which a band or an artist achieves their top ranking. You will then need to put these rows together into another dataframe or array to make the plot.

In [231]:
```
new_df = pd.DataFrame(np.array(largedf[['year', 'band_singer', 'ranking', 'born']]
new_df.columns = ['year', 'band_singer', 'ranking', 'born']
new_df['born'] = pd.to_numeric(new_df['born'], errors='coerce').fillna(0).astype(
new_df['year_minus_born'] = new_df['year'] - new_df['born']
sorted_df = new_df.sort_values(['band_singer', 'ranking']);
filtered_df = sorted_df.drop_duplicates(subset='band_singer', keep='first')
filtered_df = filtered_df.query('born > 0')
print(filtered_df)
filtered_df.groupby('band_singer')['year_minus_born'].aggregate(np.sum).sort_valu
```

```
     year      band_singer ranking  born year_minus_born
160  1992        Amy Grant      52  1960              32
231  1992     Bonnie Raitt     100  1949              43
194  1995      Bryan Adams      16  1959              36
99   1992    CeCe Peniston      20  1969              23
184  1998      Celine Dion      13  1968              30
49   1996    Color Me Badd      54  1985              11
259  1992       Don Henley      22  1947              45
133  1997      Elton John        1  1947              50
60   1992   George Michael      26  1963              29
154  1992           Hammer      46  1962              30
247  1998            Janet       6  1966              32
242  1994    Janet Jackson      12  1966              28
188  1992      Karyn White      63  1965              27
203  1992   Kathy Troccoli      79  1958              34
225  1996      Keith Sweat      10  1961              35
252  1992  Luther Vandross      41  1951              41
221  2006     Mary J. Blige     11  1971              35
167  1994    Michael Bolton      32  1953              41
62   1992  Michael Jackson      14  1958              34
260  1992      Patty Smyth      22  1957              35
199  1992      Paula Abdul      73  1962              30
234  1993     Peabo Bryson      18  1951              42
127  1994     Richard Marx      25  1963              31
254  1992          Ya Kid K      38  1972              20
```

Out[231]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29d267ac18>
```

```
/opt/anaconda3/lib/python3.6/site-packages/matplotlib/font_manager.py:1297: Use
rWarning: findfont: Font family ['sans-serif'] not found. Falling back to DejaV
u Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```



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

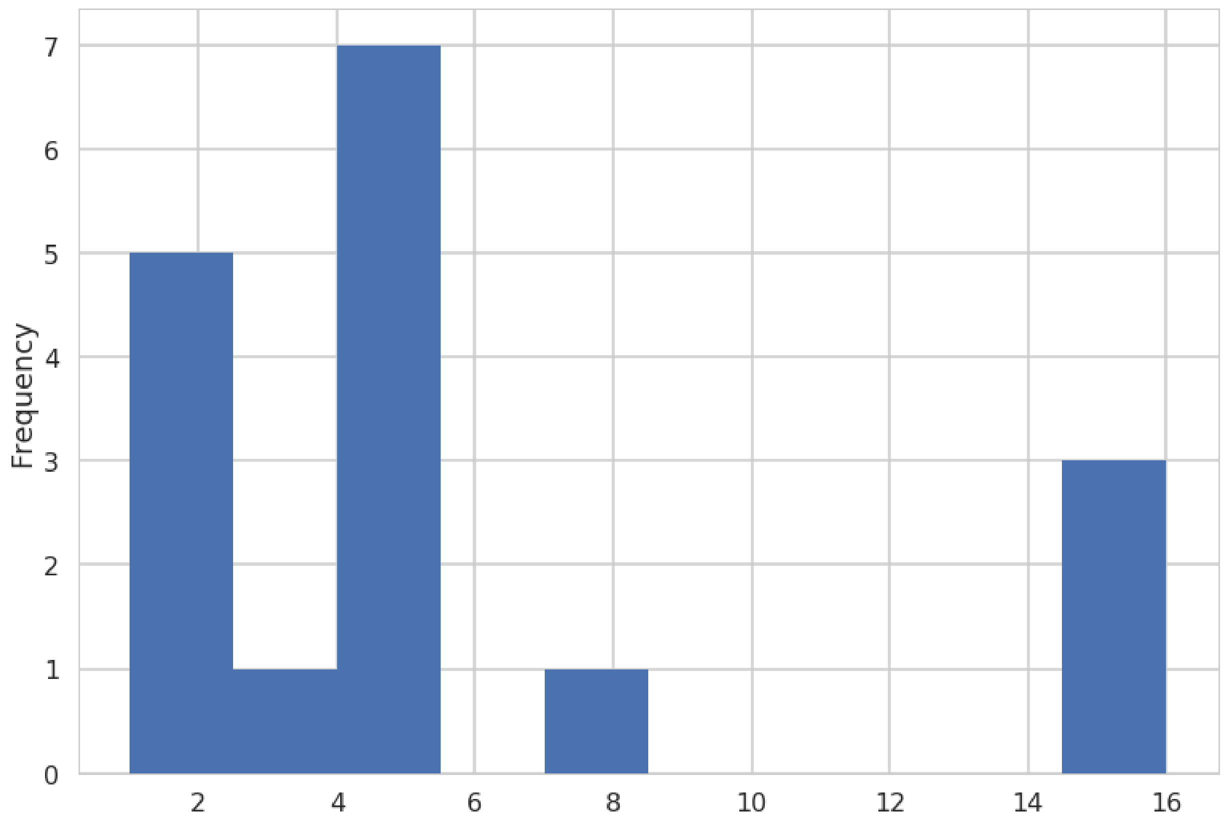
Make a similar calculation to plot a histogram of the years since inception at which bands reach their top ranking. What conclusions can you draw?

In [241]:
```
new_df_2 = pd.DataFrame(np.array(largedf[['year', 'band_singer', 'ranking', 'ya']]
new_df_2.columns = ['year', 'band_singer', 'ranking', 'ya']
new_df_2['ya'] = new_df_2['ya'].str[:4]
new_df_2['ya'] = pd.to_numeric(new_df_2['ya'], errors='coerce').fillna(0).astype(
new_df_2['year_minus_ya'] = new_df_2['year'] - new_df_2['ya']
sorted_df_2 = new_df_2.sort_values(['band_singer', 'ranking']);
filtered_df_2 = sorted_df_2.drop_duplicates(subset='band_singer', keep='first')
filtered_df_2 = filtered_df_2.query('ya > 0')
filtered_df_2 = filtered_df_2.query('year_minus_ya < 40')
print(filtered_df_2)
filtered_df_2.groupby('band_singer')['year_minus_ya'].aggregate(np.sum).sort_valu
```

```
     year                    band_singer ranking    ya year_minus_ya
139  1993           Arrested Development      31  1988             5
0    1992                    Boyz II Men       1  1985             7
235  1992                Charles & Eddie      90  1990             2
230  1993                        Das EFX      86  1988             5
206  1993                    Def Leppard     100  1977            16
39   1994                       En Vogue      14  1989             5
208  1993                           Jade      13  1991             2
14   1992                     Kris Kross       3  1991             1
189  1992  Marky Mark and the Funky Bunch      69  1989             3
56   1992                        Mr. Big      12  1988             4
202  1992                         N2Deep      78  1990             2
201  1993                          Snap!      25  1989             4
27   1995                            TLC       2  1990             5
255  1992                    Technotronic      38  1988             4
196  1992                       The Cure      71  1976            16
256  1992         The New Power Generation      25  1990             2
120  1992                             U2      57  1976            16
```

Out[241]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29d30f8eb8>
```

```
/opt/anaconda3/lib/python3.6/site-packages/matplotlib/font_manager.py:1297: Use
rWarning: findfont: Font family ['sans-serif'] not found. Falling back to DejaV
u Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```


Frequency
0
1
2
3
4
5
6
7
2
4
6
8
10
12
14
16