

# Book Ratings Analysis and Recommendation System

Claire Salloum

## Problem

My initial goal with this project was to explore a dataset scraped from a large, popular website. My initial problem was modest: can I scrape enough usable data from a site to perform exploratory and predictive analysis. I chose to scrape data from Audible.com.

Audible is an online purveyor of audio books, owned by Amazon. It contains over 150,000 English books, organized by category (eg. Mystery, Self Development, Science and Technology).

## Data

### *Data Acquisition*

I acquired the data using a two-stage scraping program. First I scraped a search results page, which contained all English products. I retrieved Title and URL for all books scraped.

```
pg = range(0, 1000)
for n in pg:
    print str(n) + ' ',
    myst =
'http://www.audible.com/search/ref=a_search_c8_2_srchPg?field_language=91781770
11&searchSize=50&searchPage='+str(n)
    page = requests.get(myst)
    soup = BeautifulSoup(page.content)
    spn1 = soup.find_all('div', {'class': 'adbl-prod-title'})
    for r in spn1:
        book = {}
        try:
            book['title'] = r.find('a').get_text().strip()
            for link in r.find_all('a', {'class': 'adbl-link'}):
                book['url'] = link.get('href')
        except:
            pass
        book_urls.append(book)
```

I ran this code twice because the number of urls returned was significantly smaller than the 150,000 books that the Audible website was reporting. After both runs, I combined the resulting output and, after deleting duplicates, had 26,677 urls. I decided to move on to the next stage of scraping, hoping that this would be enough to get started with data exploration.

I used the output from the above code to create a dataframe of URLs, which I then used to scrape data from individual book pages.

```

base_url = 'http://www.audible.com'

for url in urls.url[41600:41700]:
    url = base_url + url
    page = requests.get(url)

    soup = BeautifulSoup(page.content)
    spn = soup.find_all('div', {'id': 'adbl_page_content'})

    for r in spn:
        book = {}
        try:
            book['category'] = r.find('div', {'class':
'adbl-pd-breadcrumb'}).get_text().strip()
            book['title'] = r.find('h1',
{'class': 'adbl-prod-h1-title'}).get_text().strip()
            book['id'] = r.find('meta',
{'itemprop': 'productID'}).attrs['content']
            book['url'] = url
            book['author'] = r.find('span',
{'class': 'adbl-prod-author'}).get_text().strip()
            book['ratings'] = r.find('li', {'class':
'adbl-ratings-row'}).get_text().strip()
            book['summary'] = r.find('div', {'class':
'adbl-content'}).get_text().strip()
            book['length'] = r.find('span', {'class':
'adbl-run-time'}).get_text().strip()
            bks.append(book)

        except:
            continue

```

The above code generated a dictionary of books with the following features:

- Category
- Title
- ID
- url
- Author
- Ratings
- Summary
- Length

### *Challenges*

The above was the result of much trial and error. I encountered a major challenge running my scrapping programs for long periods of time. I found that it worked best when I ran it for a

maximum of 500 runs. Therefore, I spent a lot of time watching the program and updating the values for the range in the for loop.

I attempted to set up an Amazon EC2 instance, in the hopes that through a separate server, I could let the program run un-supervised. The EC2 instance set-up was another time-consuming challenge, which in the end, I did accomplish. However, at that point, I had moved on to the next stage of my Audible data analysis, and it was no longer as important to my project to continue scraping.

### *Initial exploration*

My initial data exploration revealed a couple of important things.

1. My original feature set did not include year or length. My initial exploration highlighted that future analysis would benefit from having more numeric features. I therefore re-wrote the script to include year and length.
2. The second stage of data exploration showed that most books were dated after 2010. I realized I had pulled the copyright year for the audiobook, which would likely not be useful, as very old books could have new audio versions. Year was not used in future analysis.
3. The ID feature was a combination of values that looked like this: "isbn:9781591799870" and others which looked like this: "B00F58OU7Q". I assumed that the first type was the isbn, which would be useful in comparing this dataset to other book databases. However, when I compared these values to known book isbn's, I realized that they were not consistent. The ID feature, was therefore not used in further analysis.

After adjusting for the above, I cleaned the book dataset (see Appendix 1 for this script) in the following ways:

- Clean author column to only include first author. Parse into first, middle and last names.
- Divide category breadcrumb into two columns: cat1 and cat2
- Clean ID
- Convert length (ie run time of audio book) to one digit (hours)
- Separate ratings column into two columns: rating(float) and n\_rating(number of ratings, int)

### **Data Visualization and Statistical Methods**

Figure 1 shows the initial breakdown of the three numeric categories in the book dataset. Rating is the average rating given by Audible users. It is a score out of 5. N\_Rating is the number of ratings per book. Run is the run time of the audiobook in hours.

	rating	n_rating	run
<b>count</b>	26674.000000	26674.000000	24422.000000
<b>mean</b>	3.985634	178.758379	9.159754
<b>std</b>	0.508375	669.659337	5.917993
<b>min</b>	1.000000	3.000000	1.016667
<b>25%</b>	3.700000	9.000000	5.466667
<b>50%</b>	4.000000	31.000000	8.516667
<b>75%</b>	4.300000	116.000000	11.566667
<b>max</b>	5.000000	36495.000000	153.983333

Fig 1. Initial visualization of numeric data

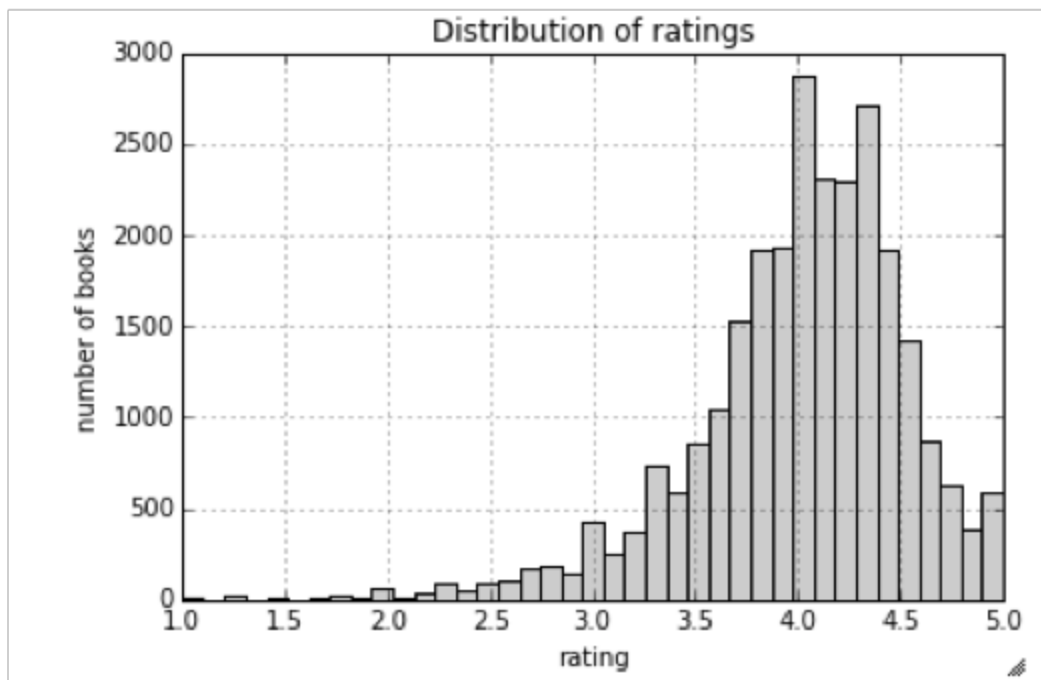


Fig 2. Histogram of ratings

Figure 2 reveals that Audible users tend to be pretty generous with their ratings. The vast majority of users rated 3 or above.

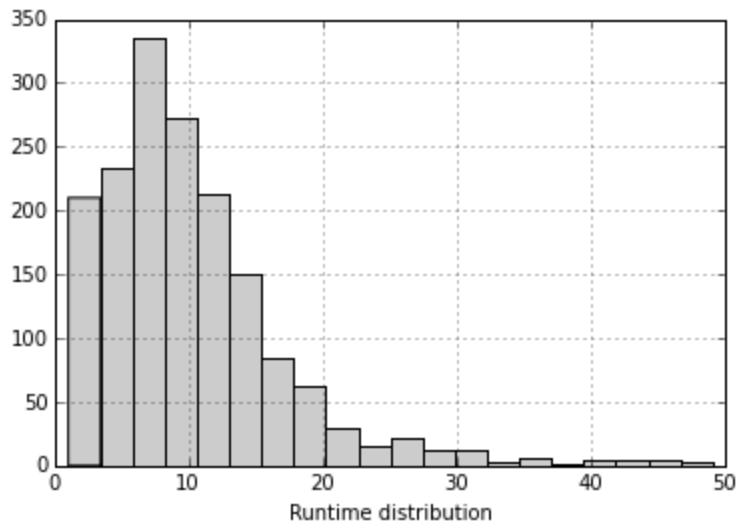


Fig 3. Histogram of run time (hours)

The majority of books were less than 20 hours long, as shown in Figure 3, with a mean of 9 hours.

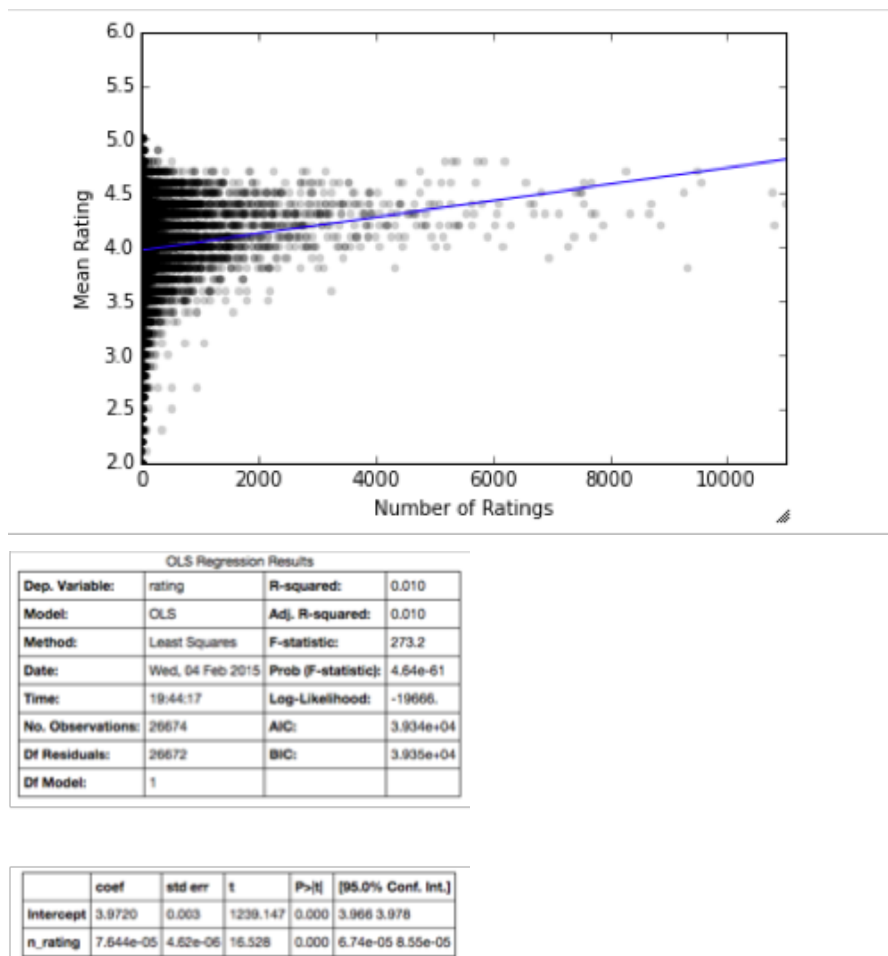


Fig. 4 Number of Ratings by Mean Rating

Figure 4 highlights that as the number of ratings increased, so did the mean rating given.

## Acquiring new data

At this stage I revised my project goal. My new goal was to use the data I had acquired from Audible to write a simple book recommendation system. In order to achieve this, I would need user-based ratings. The ratings I had from Audible were an average value taken from Audible users. Getting access to the individual user accounts for Audible would be difficult or likely impossible for me. Therefore, I started looking for other sources of user ratings of books.

I spent some time exploring Goodreads.com, which is a social network where people track and rate the books they have read. I attempted another scraping script and also experimented with the Goodreads API. In the interest of time, however, these were both abandoned.

Thanks to some outside help (Thanks, Dylan!), I acquired and downloaded a large set of user ratings and book information called BX Books.

BX books consists of:

- 1,031,175 ratings
- 271,379 books
- 278,858 users

A quick exploratory analysis revealed the following age distribution shown in Figure 5.

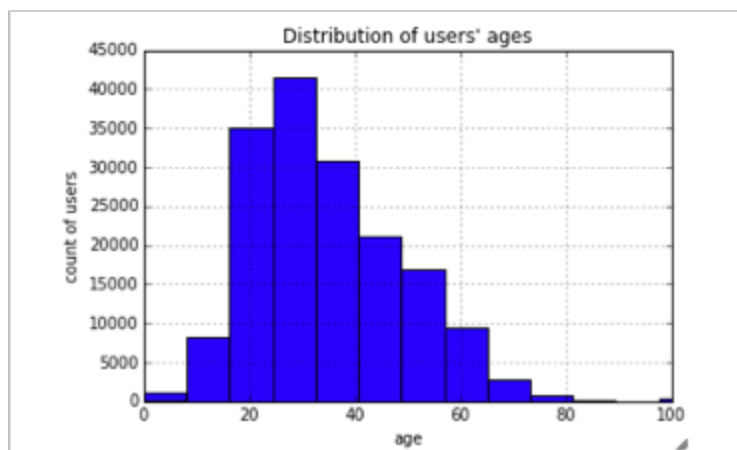


Fig. 5 Histogram of user ages from BX Books

I hoped to take the books and user reviews from BX books to supplement the Audible dataset with user ratings.

However, when I combined the two datasets, the overlap in common books was only 7000 books, as illustrated in figure 6.

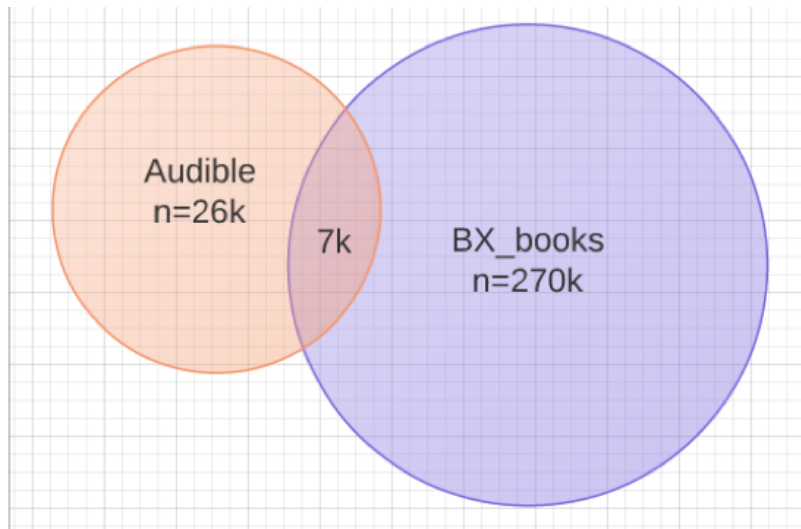


Fig. 6 Books which are common to both Audible and BX Books

In order to continue with my goal of a recommendation system, I decided to focus solely on the BX Books dataset moving forward. My reasoning was that I predicted recommendations would be more useful if there was a larger library of data to draw from.

### **Recommendation system**

I took the data from the BX books set and generated a list of the most rated and most controversial books. This was defined as books with ratings whose standard deviation was greater than 4 and the number of ratings was greater than 500. An excerpt of this list is shown in Figure 7.

I took the top 10 most-rated books from this list and used it in a subsequent recommendation system detailed below.

	user_rating		
	size	mean	std
title			
A Time to Kill	549	3.122040	4.036065
Angels & Demons	670	3.708955	4.152591
Bridget Jones's Diary	815	3.527607	4.016417
Divine Secrets of the Ya-Ya Sisterhood: A Novel	740	3.437838	4.055124
Girl with a Pearl Earring	526	4.218631	4.122380
Harry Potter and the Chamber of Secrets (Book 2)	556	5.183453	4.470784
Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))	575	4.895652	4.575678
Interview with the Vampire	508	3.551181	4.049903
Life of Pi	664	4.088855	4.215850
Snow Falling on Cedars	662	3.252266	4.038557
The Da Vinci Code	898	4.642539	4.377689
The Joy Luck Club	555	3.055856	4.032196
The Lovely Bones: A Novel	1295	4.468726	4.230463
The Notebook	650	3.560000	4.177524
The Red Tent (Bestselling Backlist)	723	4.334716	4.268839
The Secret Life of Bees	774	4.447028	4.359541
Timeline	552	3.750000	4.117930
To Kill a Mockingbird	510	4.700000	4.594158

Fig. 6 Most rated and controversial books

The recommendation system I created creates a new user and compares it to the BX books users. The new user is asked to rate 15 of the most rated and controversial books. These ratings are used to find similar users. The script for the recommendation system is printed in Appendix 2. Example output is shown in Figure 6.

Out[559]:


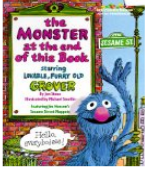


	Author	Title	Year	Image
23096	Laura Vaccaro Seeger	The Hidden Alphabet	2003	
29744	Jon Stone	The Monster at the End of This Book (Jellybean Books(TM).)	1999	
42990	William Shakespeare	Complete Works of William Shakespeare	1994	
130457	Paul Rezendes	Tracking & the Art of Seeing: How to Read Animal Tracks & Sign	1999	

Fig. 6 Example output from recommendation system.



## Business applications

Many online book (text and audio) sellers currently use recommendation systems. The system I created is limited by the type and amount of books and users contained in BX books. It is also limited by the features used - the user ratings. The system could be improved by using a more reliable or at least more contemporary set of books and users. I would like to explore using ratings from goodreads users, because they might be a more reliable set of raters (given the social aspect of the site, the users' ratings are perhaps more thoughtful). I also would be interested in incorporating features such as book genre to the system.

Despite the limitations of my recommendation system, it could have the makings of a simple web application: users could consult the Recommend4me (working title) app when having trouble choosing their next book. The more users use the system, the better the recommendations would get. An additional feature, allowing users to input and rate their most recently read book could help keep the app up-to-date.

## Conclusion

What started as an exploratory mission into web-scraping ended up producing a basic book recommendation system. I managed to get fairly comfortable with scraping, cleaning and manipulating data, despite some technical setbacks. My recommendation system is rudimentary, but I'm inspired to continue to tweak and improve it over time. Thanks to my instructors, Dylan and Haski (did you read to the end?) for all your help and patience.

## Appendix 1:

Script to clean imported book data

```
def clean_text(df):

#Categories:
    Cat1 = []
    Cat2 = []
    for text in df.category:
        try:
            line = text.decode('ascii', 'ignore').replace('\n',
'').replace('>', ',')
            cat1 = (line.split(',')[1]).strip()
            cat2 = (line.split(',')[2]).strip()
            Cat1.append(cat1)
            Cat2.append(cat2)
        except:
            cat1 = ''
            cat2 = ''
            Cat1.append(cat1)
            Cat2.append(cat2)
```

```

df['cat1'] = Cat1
df['cat2'] = Cat2

#Ratings:
Rating = []
Nrating = []
for text in df.ratings:
    try:
        rating = text.split('\n')[0]
        nrating = text.split('\n')[1]
        if len(nrating.strip('(').split()) != 0:
            nrating = nrating.strip('(').split()[0]
        else:
            nrating = -1
        Rating.append(float(rating))
        Nrating.append(int(nrating))
    except:
        rating = -1
        nrating = -1
        Rating.append(float(rating))
        Nrating.append(int(nrating))

df['rating'] = Rating
df['n_rating'] = Nrating

#IDs
isbn = []
for text in df.id:
    try:
        if text.startswith('isbn:'):
            isbn_n = int(text.replace('isbn:', ''))
        else:
            isbn_n = text
        isbn.append(isbn_n)
    except:
        isbn_n = 'NaN'
        isbn.append(isbn_n)
df['isbn'] = isbn

#Authors
#using nameparser HumanName
author_last = []
author_first = []
author_middle = []
author_name = []
for author in df.author:
    try:
        guy1 = (author.split(', '))[0]

```

```

        except:
            guy1 = author

    name = HumanName(guy1)

    try:
        last = str(name.last)
    except:
        pass
    try:
        first = str(name.first)
    except:
        pass
    try:
        middle = str(name.middle)
    except:
        pass
    author_last.append(last)
    author_first.append(first)
    author_middle.append(middle)
    author_name.append(guy1)
df['first'] = author_first
df['middle'] = author_middle
df['last'] = author_last
df['author'] = author_name

#pub_year from summary

pub_year = []

for text in df.summary:
    year = 0
    t = text.split()
    try:
        for word in t[-30:]:
            if (word.startswith('(c)') or (word.startswith('(P)')):
                year = int(word[-4:])
    except:
        pass
    pub_year.append(year)
df['year'] = pub_year

# Run time

run_time = []
for line in df.length:
    s = line.split('and')
    if len(s) == 2:

```

```

        hr = float(s[0].split()[0])
        mn = (float(s[1].split()[0]))/60
        time = hr + mn
    elif (len(t) == 1) & (t[0][-3:-1] == 'hr'):
        hr = float(t[0].split()[0])
        mn = 0
        time = hr + mn
    else:
        time = -1

    run_time.append(time)
df['run'] = run_time

```

## Appendix 2: Recommend4me

```

def loadBookRatings():
    numBooks = 0
    bookNames = {}
    bookRatings = {}
    for n in ratings.index:
        bookID = ratings.isbn.loc[n]
        bookName = ratings.title.loc[n]
        bookNames[bookID] = bookName

        userID = ratings.userID.loc[n]
        rating = ratings.user_rating.loc[n]
        #timestamp = ratings.run.loc[n]

        if (not(bookRatings.has_key(userID))):
            bookRatings[userID] = {}
        bookRatings[userID][bookName] = float(rating)

    numBooks += 1

    return (bookRatings, bookNames, numBooks)

```

```
(bookRatings, bookNames, numBooks) = loadBookRatings()
print "Our data has " + repr(numBooks) + " dimensions."
```

```
#### Using Haski's Pearson function
```

```
def pearson(X, Y):
    n = len(X)
    if n == 0: return 0

    sumX = 0.0
    sumY = 0.0
    sumX2 = 0.0
    sumY2 = 0.0
    sumXY = 0.0

    for i in range(n):
        x = float(X[i])
        y = float(Y[i])
        sumX += x
        sumY += y
        sumXY += x * y
        sumX2 += x * x
        sumY2 += y * y

    numerator = sumXY - ( (sumX * sumY) / n )
    denominatorTerm1 = sumX2 - ( (sumX * sumX) / n )
    denominatorTerm2 = sumY2 - ( (sumY * sumY) / n )
    denominator = (denominatorTerm1 * denominatorTerm2) ** 0.5
    if (denominator == 0.0):
        return 0.0
    else:
        return numerator / denominator
```

```
#### Using Haski's Find Similar Users function
```

```
def findSimilarUsers(targetUserID, allRatings):
    similarUsers = []
    targetUserRatings = allRatings[targetUserID]
    for user in allRatings:
        # Exclude myself:
        if (user != targetUserID):
            userSimilarity = 0
            # Find mutually rated movies (if any)
            targetUserRatingsArray = []
            otherUserRatingsArray = []
            for movie in targetUserRatings:
```

[illegible]

```

# Compute the averages
recommendationCandidates = []

for candidate in candidateTotals:
    (totalScore, totalSim, sourceRatings) = candidateTotals[candidate]
    recommendationCandidates.append((totalScore / totalSim, totalScore,
                                     candidate, sourceRatings))

recommendationCandidates.sort(reverse=True)

#return recommendationCandidates[0 : numRecommendations]
return [(b[2]) for b in recommendationCandidates[0 : numRecommendations]]

def recommend4me(newID):
    #create new user profile
    d = {}
    for i, row in most_rated.iterrows():
        d[row['title']] = int(raw_input(row['title'] + ': '))

    bookRatings[newID] = d

    #find similar users
    similarUsers = findSimilarUsers(newID, bookRatings)

    #get recommendation candidates
    candidates = getRecommendationCandidates(newID, similarUsers, bookRatings,
10)

    candidates_db =
BX_books[BX_books.title.isin(candidates)].drop_duplicates('title')

    #present candidates
    df = candidates_db[['author', 'title', 'pub_year']]
    df.columns = ['Author', 'Title', 'Year']
    df['Image'] = candidates_db['imageM'].map(lambda x: '</img>'.format(x) )
    pd.set_option('max_colwidth', 300)
    return HTML(df.to_html(escape=False))

```