Data Channel For Movie website(Douban, Rotten Tomato)

Group Member： Tairan Ren, Xingyi Liu, Wenhu Cheng, Yi Zheng

1.Descriptions:

The programme does three things:

1. requesting data from websites, for the purposes of practices, both urllib and request are implemented.

   files: requests_douban.py, urlib_request.py

2. transforming the data: original data were stored in the format of json files, for them to be more readable, all the data files were transformed into csv format. All the nested columns exploded, unwanted columns removed, duplicates removed etc.,.

   files: all_json_to_csv.py ==> ods_process_csv.py

3. then building a mini size Datawarehouse, from the ods files, using dimensional data to categorize all the data and do the counts, in the top level then add the functions to draw the graph and visualize everything.

   files: dim_to_ads.py ==> ads_graph.py/adsshow.ipynb

   Codes are in different files for easier debugging and categorization of where the functions are most used.

   In the end, a main loop file is created, to import all the programme before the demo case file, where further discussion of the data were made, two dimension concatenated etc,.

   lib reliances:

   pandas: for its dataframe and read_csv

   matplotlib: for graphing

   urlib and request: to request for data

   json: for reading json files

2. Design Process:

2.0 Special helper Function：

```
import os

FILE_PATH = {'data':
['csv_rotten','csv_douban','dim','ads','ods','rotten','douban']}

def create_data():
    '''
    the fucntion that creates the folders system
    '''
```

```
    keys = list(FILE_PATH.keys())

    if not os.path.exists(keys[0]):
        print('creating level 1 folder')
        os.makedirs(keys[0] + '/')

    for i in FILE_PATH[keys[0]]:
        if not os.path.exists(keys[0] + '/'+ i +'/'):
            print(f'creating level 2 {i+"/"}')
            try:
                os.makedirs(keys[0] + '/'+ i +'/')
            except Exception as e:
                print(e)

def file_check(filename):
    '''
    the fucntion check if the file exists, not applied
    '''
    return os.path.isfile(filename)

def change_os_path():
    '''
    the function change the current operating address
    '''
    path ,_ = os.path.split(os.path.realpath(__file__))
    print(path)
    if os.getcwd() != path:
        os.chdir(path)
        print('changed')
```

This to set the operation path to the current path and create the folders

2.1 Request module:

with three major functions:

1. The function that create the request, get and return the content:

```
# create and return the request from url
def create_request(page:str,genre):

    base_url = f'https://movie.douban.com/j/chart/top_list?type=
{genre}&interval_id=100:90&action=&'
    # the starting point and each scroll
    param = {
        'start':page,
        'limit':20,
    }
    # using UA to simulate  a user surfing internet
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36'
    }
    # using timeout param to wait for connection, and retry when one request
fails
```

```
    flag = True
    while flag :
        try:
            response =
requests.get(url=base_url,params=param,headers=headers,timeout=2)
            flag = False
        except Exception as e:
            print(e.args)
            print('retrying')
            time.sleep(1)

    return response
```

2. the function that store the response to json format files:

```
# create file and store the data in it, and will automaticly create file names
def download(data_json:str,filename:str):
    filename = filename + '.json'
    with open(filename,'w',encoding='utf-8') as fp:
        json.dump(data_json,fp=fp,ensure_ascii=False)
    print('done')
```

3. the main loop:

```
def request_all(pages = 5):
    # 32 as the website is using numbers to mark types, and there are 32 in
total
    # page as in the scroll times, default at 5
    for i in range(1,32):
        for j in range(pages):
            response = create_request(page=0,genre= i)
            list_data = data_toJson(response=response)

 #download(data_json=list_data,file_name=FILEPATH+GENRE[i]+f'_{i}'+'.json')
            download(data_json=list_data,filename= FILEPATH+'douban/' +
f'_{i}_{j}')
                #time.sleep(1)
```

Interestingly, Douban is using numbers to mark types, and there are 32 in total, page as in the scroll times, default at 5, while rotten tomato is using names and a very special paging system: ['page=1','after=Mjk%3D','after=NTk%3D','after=ODk%3D','after=MTE5'] (probably hashed or encoded integer?), other than that, the codes for rotten tomato is using urllib.request, which is fundamentally the same, check the codes for more details.

2.2 Json To Csv module:

Taking the json data requested from rotten tomato as an example:

```
{
    "grids":Array[1],
    "pageInfo":Object{...}
}
// if expand the grids map, and inside that Array. its nested data of movies:
```

```
{
            "id":"movies_at_home_genres:action~sort:audience_highest", // this
is the searching id, not real data
            "list":[
                {
                   "audienceScore":{
                        "score":"100",
                        "sentiment":"positive"
                    },
                    "criticsScore":{
                        "certifiedAttribute":"criticscertified",
                        "score":"91",
                        "sentiment":"positive"
                    },
                    "fallbackPosterUrl":"......",
                    "id":"cc68d6bb-6d2c-388e-959f-d241b4bad34a",
                    "isVideo":true,
                    "emsId":"cc68d6bb-6d2c-388e-959f-d241b4bad34a",
                    "mediaUrl":"/m/raging_fire",
                    "mpxId":"1928806467555",
                    "posterUri":"......",
                    "releaseDateText":"Streaming Nov 23, 2021",
                    "title":"Raging Fire",
                    "trailerUrl":"......",
                    "type":"Movie"
                },
```


Since its so nested, the codes starts from cleaning process, the main loop is :

```
def json_to_csv_rotten():
    # get the headers from each movie data
    headers_etl = movies_headers('rotten')

    for i in range(len(ulr.GENRE)): # ulr.GENRE = All the types, this is silly i
know
        for j in range(5):
            # pointing which file processing
            print(f'{ulr.GENRE[i]} {j}')
            # get the movie type, since the orginial didn't include that
            file_type = ulr.GENRE[i]
            # read the file
            content =
read_json(file_name=f'data/rotten/{ulr.GENRE[i]}_{j}.json')
            # get the list of movies, break the grids
            content = movies_dicts_rotten(content)
            # give none value to empty spot
            content = etl_rotten(content,headers_etl)
            # trans form them to list of data for dataframe
            list_data = movies_data(content,type=file_type,web='rotten')
            # create dataframe, this is also checking if headers are matching
the list
            # they some how changed the structure recently
            df = create_dataframe(list_data)
            # prevent headers from multipling it self
            if i == 0 and j == 0:
```

```
                headers = headers_etl
            else:
                headers = None
            # write to file

write_csv(filename=f'data/csv_rotten/ods_rot.csv',headers=headers,dataframe=df)
```

the major functions, includes, extracting headers, empty value transformation and prepare the data for pandas pandas dataframe, which requiring headers of all the rows and data in list.

1. braking the nested json:

```
def movies_dicts_rotten(content:dict):
    # this function returns the part where all the value are stored
    #{grids:[{'id':xxx,'list':[{......}]}]}
    layerone = content.get('grids') # it is a nested structures with both dict
and list
    list_movies = layerone[0]['list'] # it is another list with all movies in
the given range
    #for movie in list_movies: there cases that some movies doesn't align with
others
    return list_movies
```

2. getting headers, since the header structures for two website are quite different:

```
def movies_headers(web:str):
    # this function returns the part where all the value are stored
    # it is a dict in list structure
    # cont = [{},{}] WEB_SITES = ['rotten','douban']
    if web == WEB_SITES[1]:
        content = read_json(file_name='data/douban/_1_0.json')
        headers = []
        for key in content[0].keys():
            headers.append(key)
    if web == WEB_SITES[0]:
        content = read_json(file_name=f'data/rotten/{ulr.GENRE[0]}_0.json')
        content = movies_dicts_rotten(content)
        headers = []
        for key in content[0].keys():
            headers.append(key)
        #headers.append('type')
    return sorted(headers) #with all the fields as dicts
```

3. empty tranformation:

```
# adding none value to the data's that has no value in it
def etl_rotten(c:list,headers:list):
    result = []
    for movie_dict in c:
        for key in headers:
            if not (key in movie_dict) :
                movie_dict.update({key:'None'})
        sorted_dict = dict(sorted(movie_dict.items(), key=lambda x: x[0]))
        result.append(sorted_dict)
    return result
```

4. In the format of lists:

```
def movies_data(content,type,web):
    # uses pandas to store the data in csv prepared format
    movies = []
    for i in range(len(content)):
        values = content[i].values()
        movie = []
        for value in values:
            movie.append(value)
        if web == WEB_SITES[0]:
            movie[-1] = type
        movies.append(movie)
    return movies
```

the outcome is:

audienceScore	criticsScore	emsId	fallbackPosterUrl	id	isVideo	mediaUrl	mpxId	posterUri	releaseDateText	title	trailerUrl	type
{'score': '100', 'sentiment': 'positive'}	{'certifiedAttribute': 'criticscertified', 'score': '91', 'sentiment': 'positive'}	cc68d6bb-6d2c-388e-959f-d241b4bad34a	//images.fandango.com/cms/assets/5d84d010-	cc68d6bb-6d2c-388e-959f-d241b4bad34a	TRUE	/m/raging_fire	1.93E+12	https://resizing.flixster.com	Streaming Nov 23, 2021	Raging Fire	...	action

Sadly, it is stll very nested and with many unwanted columns.

2.3 the mini data warehouse

From the csv data files, the idea is to transform all the data into a three layer structures, from there on, all the data are directly operatable:

ods: original or operational	**do the transform and clean the data**
dim: dimensional	**using dimension to group and count the data**
ads: application	**data ready for drawing**

each level contains less data than the previous, in the top level ads, data should be ready for drawing and in the simplest way of showing one target, with one key, one data.

2.3.0 naming principle:

this is very important for automation:

the ods file will have the param time in the front, in this case it's set to 7_ for conveniences, since it is using appending, and from there on follows the principle:

```
time_layer_keyColumn1_keyColumn2(if any)_operator(if any)_whichWeb.csv
```

for layers above ods, there is currently no need to add time to it, since the io type is changed to writing, which should be align with ods, but in this case, data are in small amount, so for the sake of conveniences:

```
layer_keyColumn1_keyColumn2(if any)_operator(if any)_whichWeb.csv
# in example:
ads_audience_score_grouped_rotten.csv
```

2.3.1 ods:

since there are still nested and unwanted columns, further cleaning is necessary, the main function:

```
def ods_rot():
    # read_csv
    df_rot = pd.read_csv('data/csv_rotten/ods_rot.csv')
    # drop unwanted columns: (ems, mpx are newly added)
    df_rot = etl_remove_list(df_rot,['Url','is','Uri','ems','mpx'])
    # extract month from orginal data as it is a dimension but needs processing
    df_rot = adding_month(df_rot,'releaseDateText','rotten')
    # explode the nested columns
    df_rot = adding_explode_audience(df_rot)
    df_rot = adding_explode_critic(df_rot)
    # write the data to csv
    df_rot.to_csv('data/ods/ods_etled_rotten.csv',encoding='utf-8',index=False)
```

the major function here is the drop columns, extracting month from time, and explode:

1. drop columns:

```
# this function loop throhgh the list of unwanted columns and drop them
def etl_remove_list(df,columns:list):
    headers = df.columns.values
    bad_header = []
    for i in columns:
        for header in headers:
            if i in header:
                bad_header.append(str(header))

    print(f'droping: {bad_header}' )
    df_copy = df.drop(bad_header,axis=1)
    return df_copy
```

2. adding month (in the case of two website, different method were added):

```
# extract the month column from time column
def adding_month(df,header,website):
    month = []
    df_copy = df
```

```
    if website == 'douban':
        return adding_month_douban(df,header,month,df_copy)
    if website == 'rotten':
        return adding_month_rotten(df,header,month,df_copy)
# the time format : release_date : 2006/6/3
def adding_month_douban(df,header,month,df_copy):
    for string in df[header]:
        if len(string)> 4:
            list_time = string.split('-')
            month.append(list_time[1])
        else:
            month.append(None)

    print(f'adding month column')
    df_copy['month'] = month
    return df_copy
# the time format : releaseDate : Streaming Nov 23, 2021
def adding_month_rotten(df,header,month,df_copy):
    for string in df[header]:
        if len(string) > 0:
            list_time = string.split(' ')
            month.append(list_time[1])
        else:
            month.append(None)
    print('adding month column')
    df_copy['month'] = month
    return df_copy
```

3. exploding,

```
# the method that explode the audience score
def adding_explode_audience(df):
    value = []

    for string in df['audienceScore']:

        if len(string) > 2:
            end_index = string.find(",",0,len(string))
            value.append(string[10:end_index])
        else:
            value.append(None)

    df_copy = df
    df_copy['audience_score'] = value
    df_copy = df_copy.drop('audienceScore',axis = 1)
    return df_copy
```

2.3.2 dim & ads:

since the ods data created as following:

id	releaseDateText	title	type	month	audience_score	critics_score

id	releaseDateText	title	type	month	audience_score	critics_score
cc68d6bb-6d2c-388e-959f-d241b4bad34a	Streaming Nov 23, 2021	Raging Fire	action	Nov	'100'	'91'

For this project at the moment, two major dimension were selected: month, type and concatenate two of them to form new keys. Further expansion could include audience_score and critics_score

In this case, three dimension were selected, month, type and audience_score

Showing the key month, and the generalized method of concatenation:

1. single key method, including directly to ads level codes, this is the 1.0 version:

```
def dimension_month_ads():
    dim_to_file('ods_etled_douban.csv','month')
    dim_to_file('ods_etled_rotten.csv','month')
    # will groupby key and write to new ads files
    ads_oped_data('dim_month_rotten.csv','grouped','month')
    ads_oped_data('dim_month_douban.csv','grouped','month')
    # the special function only for sorting the rotten
    sort_dict_rotten('data/ads/ads_month_grouped_rotten.csv')
```

the more generalized 2.0 version:

```
def dimension_general(web :str, type = 'audience_score' ):
    '''
    input : web : data source
            type : the dimension column
    write to the file created automatically
    '''
    def serach_headers(filename):
        '''
        the inner fuction that loops the header, search the matching columns
        and return the selected columns.
        '''
        columns = []
        filepath = FILE_PATH_ODS
        df = read_csv(FILE_PATH_ODS + filename)
        headers = list(df.columns)
        for i in headers:
            if type in i:
                columns.append(i)
        if len(columns) > 1:
            print(columns)
            num = int(input('which one do you mean, from 0: '))
            return columns[num]
        else:
            return columns[0]
    if web == 'douban':
        filename = 'ods_etled_douban.csv'
        score = 'score'
    elif web == 'rotten' :
        filename = 'ods_etled_rotten.csv'
        score = 'aduience_score'
```

```
    else:
        # for future expansion
        pass
    key = serach_headers(filename)
    return dim_to_file(filename,key) , key
```

In 2.0 version Functions are all generalized, dropping is just like in the previous, so major functions including toFile, and counting:

```
# create the dim data, dimension data with id
def dim_to_file(filename:str,dimension:str):
    filepath = 'data/ods/'
    df = read_csv(filepath+filename)
    # erase duplication, risky, but for now I think it works
    df_dropped = drop_duplicate(df,'id')
    # drop the other columns, only leave the key and the counts
    df_month = dimension_drop(df_dropped,dimension)
    print('printing first 5 rows:' )
    print(df_month[0:5])
    if filename[10:len(filename)] == 'douban.csv':
        encoder = 'utf-8-sig'
    else:
        encoder = 'utf-8'
    #print(encoder)

 df_month.to_csv(f'data/dim/dim_{dimension}_{filename[10:len(filename)]}',encodi
ng=encoder,index=False)
```

For now, the dim file is like this:

id	month
cc68d6bb-6d2c-388e-959f-d241b4bad34a	Nov
87a94713-52e0-32f7-9122-a66784618320	Dec
...	...

And in ads level, ready for graphing, after counting by month:

month	movie_num
Jan	256
Feb	150
...	...

Then the two key combine method the 1.0 version, with fixed month and type dimension:

```
# concact two of them to form new keys
def dimension_monthAndType_ads(month):
    # get the dataframe for pointed columns
```

```
    df_type = read_csv('data/dim/dim_type_rotten.csv')
    df_month = read_csv('data/dim/dim_month_rotten.csv')
    # join on id so that they will be in the same df
    df_merge = df_type.merge(df_month,how='inner',on='id')
    # create new columns and add none value if there is no value in it
    df_merge['month_type']=df_merge.apply(lambda x:str(x['month'])+"_"+x['type']
if (x['month'] != None) else np.nan,axis=1)
    # group by and count
    df_merge = groupby_count(df_merge,'month_type')
    # rename the header
    df_merge['counts'] = df_merge['id']
    # drop unwanted duplicated columns
    df_merge = df_merge.drop(['id','type','month'],axis=1)
    # write to file
    df_merge.to_csv(f'data/dim/dim_montyAndType_rotten.csv',encoding='utf-8')
    # upgrade file to ads level
    ads_monthType('data/dim/dim_montyAndType_rotten.csv',month)
# the main method to write ads data to file in ads level
def ads_oped_data(filename:str,optype:str,key:str):
    filepath= 'data/dim/'
    df = read_csv(filename= filepath+filename)
    df_grouped = groupby_count(df,key)
    oped_write_csv(df_grouped,filename,optype,key)
```

For now, the data look like this:

month_type	counts
Apr_action	15
...	...

the helper picking function will loop the data and return the selected data of selected month:

```
def ads_monthType(filename:str,month_key):
    df_merge = read_csv(filename)
    month = []
    value = []
    for row in df_merge.itertuples():
        if month_key in getattr(row,'month_type'):
            month.append(getattr(row, 'month_type'))
            value.append(getattr(row, 'counts'))

    df_new = pd.DataFrame({'month_type':month,'counts':value})
    oped_write_csv(df =
df_new,filename=filename,optype='counts',key=f'monthType{month_key}')
```

the search result:

month_type	counts
Mar_action	16

month_type	counts
Mar_adventure	12
...	...

Obviously, it could be more generalized and enabling theoretically do any two columns, in this case it's the reverse of type and month, this is also 2.0 generalized should work on both dimensions:

```
# the general version therotically this method should be able to deal with any
two columns
def dimension_concact_ads(type_one,type_two,key_column = None):
    df_type_one = read_csv(f'data/dim/dim_{type_one}_rotten.csv')
    df_type_two = read_csv(f'data/dim/dim_{type_two}_rotten.csv')
    df_merge = df_type_one.merge(df_type_two,how='inner',on='id')
    df_merge[f'{type_one}_{type_two}'] = df_merge.apply(lambda
x:str(x[type_one])+"_"+x[type_two] if ((x[type_one] != None) and (x[type_two] !=
None)) else np.nan,axis=1)
    df_merge = groupby_count(df_merge,f'{type_one}_{type_two}')
    df_merge['counts'] = df_merge['id']
    df_merge = df_merge.drop(['id',type_one,type_two],axis=1)

 df_merge.to_csv(f'data/dim/dim_{type_one}_{type_two}_rotten.csv',encoding='utf-
8')

 ads_concacted_two(f'data/dim/dim_{type_one}_{type_two}_rotten.csv',key_column)
    print(df_merge[0:5])
```

And generalized helper function:

```
# the generalized helper function
def ads_concacted_two(filename:str,key):
    df_merge = read_csv(filename)
    list_str = filename.split('_')
    col_one = []
    col_two = []
    for row in df_merge.itertuples():
        if key in getattr(row,f'{list_str[1]}_{list_str[2]}'):
            col_one.append(getattr(row, f'{list_str[1]}_{list_str[2]}'))
            col_two.append(getattr(row, 'counts'))

    df_new =
pd.DataFrame({f'{list_str[1]}_{list_str[2]}':col_one,'counts':col_two})
    oped_write_csv(df =
df_new,filename=filename,optype='counts',key=f'{list_str[1]}_{list_str[2]}_{key}
')
```

The helper functions, it's groupby and write to file that matters:

a. group by function:

```
# one of the ads data methods groupby and count, it first arrange the columns and
count how many rows
# in sql: select count(x) from a groupby x
def groupby_count(df:pd.DataFrame,column:str):
    df_group = df.groupby(column).count().reset_index()
    #print(df_group[0:5])
    return df_group
```

b. the write function:

```
# ads write to csv, will automatically generate file names
def oped_write_csv(df:pd.DataFrame,filename:str,optype:str,key:str):

    if filename == 'douban':
        encoder = 'utf-8-sig'
    else:
        encoder = 'utf-8'

 df.to_csv(f'data/ads/ads_{key}_{optype}_{filename[-10:len(filename)]}',encoding
=encoder,index=False)
```

c. the special function to sort the months, this was only created because groupby will auto sort alphabetically, which is not what we want:

```
# the extra method only for rotten tomato to sort the data based on month string
def sort_dict_rotten(file:str):
# for the case rotten tomato data doesn't has a sorted months
    df = read_csv(file)
    dict_movie = {}
    result = []
    for i in range(0, len(df)):
        for j in range(0, df.shape[1]):
            result.append(df.iloc[i][j])

    month = result[::2]
    value = result[1::2]

    for i in range(len(month)):
        dict_movie.update({month[i]:value[i]})
    result = {}
    for i in range(len(MONTH)):
        value = dict_movie[MONTH[i]]
        result.update({MONTH[i]:value})

    df_sorted =
pd.DataFrame({'month':result.keys(),'movie_num':result.values()})
    df_sorted.to_csv(file,encoding='utf-8',index=False)
```

2.4 the main loop for the whole programme so far:

from all the files discussed above, the main loop so far is as follows:

```
def main():
    # change the op path to file address:
    osh.change_os_path()
    # create the data folders system:
    osh.create_data()
    # get the request and store the data in orginial json files:
    print('requesting data from douban')
    reqd.request_all()
    print('requesting data from rotten tomato')
    urlr.requests_allgenre_pagefive_rotten()
    # do the process of cleaning data,adding mark to distingusih every time
    print('processing json files to csv files')
    jtc.json_to_csv_douban('7_')
    jtc.json_to_csv_rotten('7_')
    # do the further cleaning, and store everything in ods layer, shsould align
with the previous
    print('creating ods layer files')
    opc.ods_douban('7_')
    opc.ods_rot('7_')
    # do the dimensional data modelling
    print('creating dimensional files with single dimension')
    for i in ['douban','rotten']:
        for j in ['month','type','score']:
            created_filename , key = dta.dimension_general(i,j)
            print(created_filename + ' created')
            dta.ads_oped_data(created_filename,key)
            print('ads upgraded')
    # the further concactenation is in the adsshow.ipynb

if __name__ == '__main__':
    main()
```

the programme starting point was also addressed in each of the files.

2.5 Jupyter notebook

then it's the graphing part:

this the easy part , only two functions created, which will generate bar graph and line graph:

```
# the bar graph generator
# the bar graph generator
def graph_bar(filename,columns,barh = None, plot = None, sort = False):
    '''
    filename : filename
    columns : key
    barh : whether horizantal or not
    plot : line graph or not
    sort : sort by second column or not
    '''
    df = read_csv(f'data/ads/{filename}')
```

```
    if sort :
        header = list(df)
        #df = df.sort_values(by='id',ascending=True)
        df = df.sort_values(by=header[1],ascending=True)
    df_table = df.pivot_table(columns=columns)
    y_data = df[f'{columns}_'] = df.iloc[:,-1]
    x_data = df[columns]
    if barh == True:
        plt.barh(x_data,y_data,0.5,color =
['paleturquoise','mediumturquoise','lightseagreen','turquoise','aquamarine'])
    elif barh == False:
        plt.bar(x_data,y_data,0.5,color =
['darkslategray','teal','cadetblue','steelblue','darkcyan'])
    elif plot == True:

 plt.plot(x_data,y_data,0.5,marker='o',mfc='orange',ms=5,mec='c',lw=1.0,ls="-
",c='green' )

    plt.xticks(rotation=270)
    plt.tight_layout()
    plt.show()
    return df_table

# the multi line graph generator
def multi_line(filenames:list,columns = None):

    color =
['darkgray','rosybrown','darkgoldenrod','darkslategray','midnightblue']
    for i in range(len(filenames)):
        nameList = filenames[i].split('_')
        columns = nameList[3]
        df = read_csv(f'data/ads/{filenames[i]}')
        y_data = df[f'{columns}_'] = df.iloc[:,-1]
        x_data = df[columns]
        flag = i
        if flag > len(color):
            flag -= len(color)
        plt.plot(x_data,y_data,label=
columns,marker='o',mfc='orange',ms=5,mec='c',lw=1.0,ls="-",c= color[flag],  )

    plt.xticks(rotation=270)
    plt.tight_layout()
    plt.show()
```

for the actual codes that draw the graphs, since they are just calling functions, please refer to adsshow.ipynb.

3. Future plans:

For now the codes takes no user input, and the amount of data is relatively small, while the dimension is restricted to two, although with theoretic ability of expansion, yet nothing were tested.

An interesting situation is that if the front end codes were changed, it is possible that url will lost its function, and more or less columns will be modifed to their dataformat, which will then pose challenges to this programme.

These codes have achieved a certain degree of automation, so the next step should be to further encapsulate, create and use the main loop to encapsulate all the codes into a file, and then apply all the args to be user inputs and use tools such as pyinstaller to package them。

It would be a lot better if we decide to add databases to store the data rather then using files, that would enable sql queries and avoid using IO so many times.

Meanwhile there are certain risky methods were implemented, such as the drop duplicates, which is bad for future expansion.

-- Tairan Ren

4. Reflections:

What I learned most useful from this project are two main tricks:
First, How to get data from a website: we need to use the requests package, during which we may need to stimulate that we are a real person in case we get denied, and the second is how to clean all these data to useable ones since we need to group them. And also along the way I have a peek at how to use JSON, which is completely new to me.

– WENHU CHENG

In this project, I learned how to crawl data from websites and visualize them as the form of chart. And in the process of implement, definitely I met a lot of problems. Such as data crawling, data cleaning and visualization. I overcome them with the help of my teammates and website. I would like to talk more about the part of data slicing. I was absolutely new to this area. But now, I can conclude it with 3 steps. Firstly, remove the columns of data which we don't need. Secondly, we need to slice out the specific data which we want form pieces of data. Last one, we need to add or replace the data which we sliced out based on the original form to create a new form.

-- YI ZHENG

With the help of my teammates, I learned a lot from our project. After data collections and data cleaning, I learned how to use two functions and matplotlib packages in Python to do the data analysis and data visualization. And I learned a lot about how to exact marketing insights on this data-driven approach. First, we can analysis the relationship between the number of newly released movies and month of rotten tomato and douban, and conclude that movies are highly influenced by holidays. Second, we can obtain what’s the most popular type of movie in rotten tomato and douban.

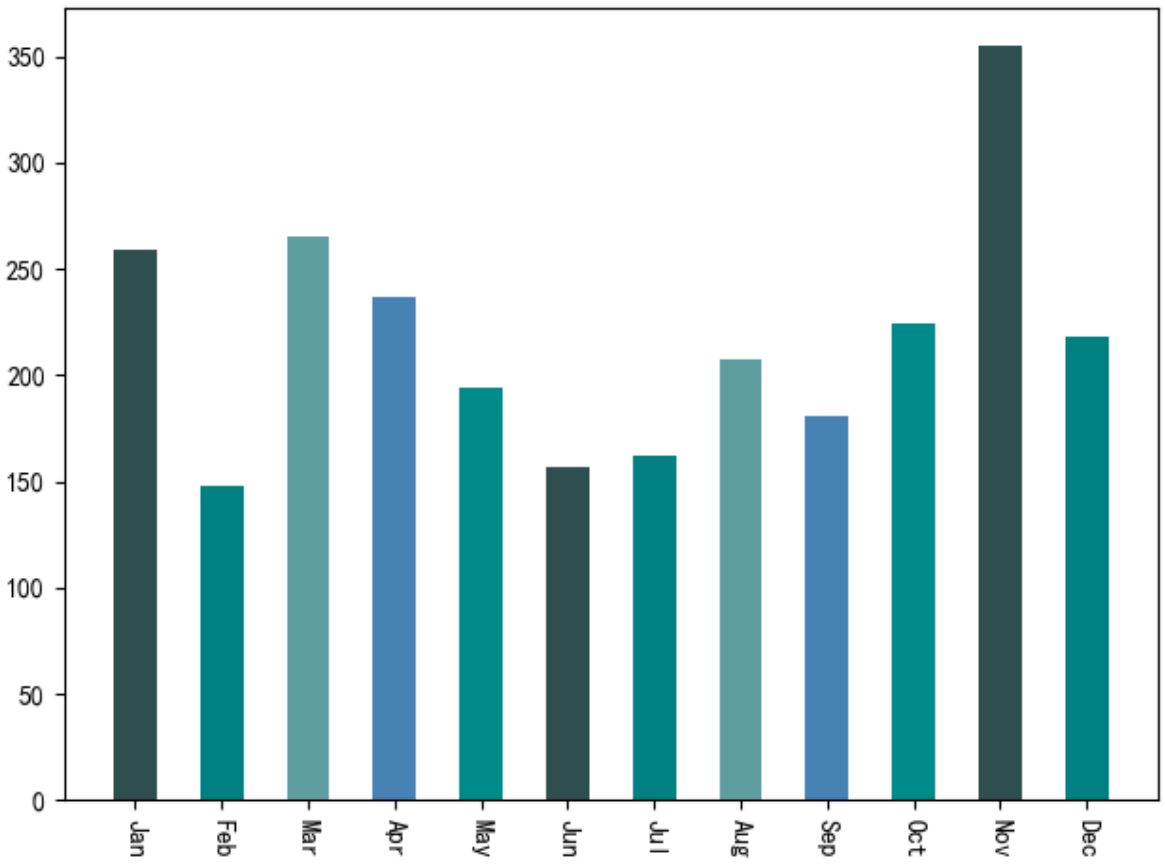
-- XINGYI LIU

5. Results:

5.1 The month bar graph :

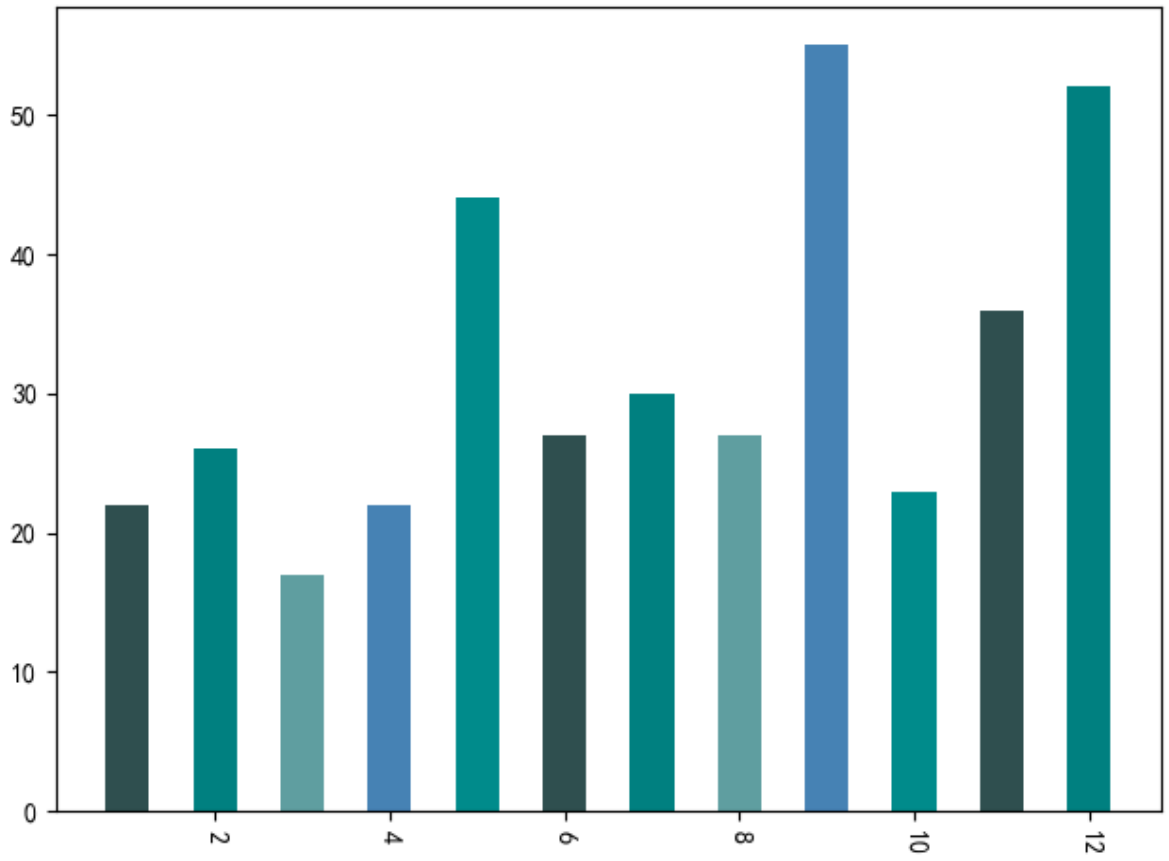
rotten tomato:

month	movie_num
Jan	256
Feb	150
Mar	267
Apr	236
May	194
Jun	157
Jul	160
Aug	206
Sep	182
Oct	222
Nov	362
Dec	209



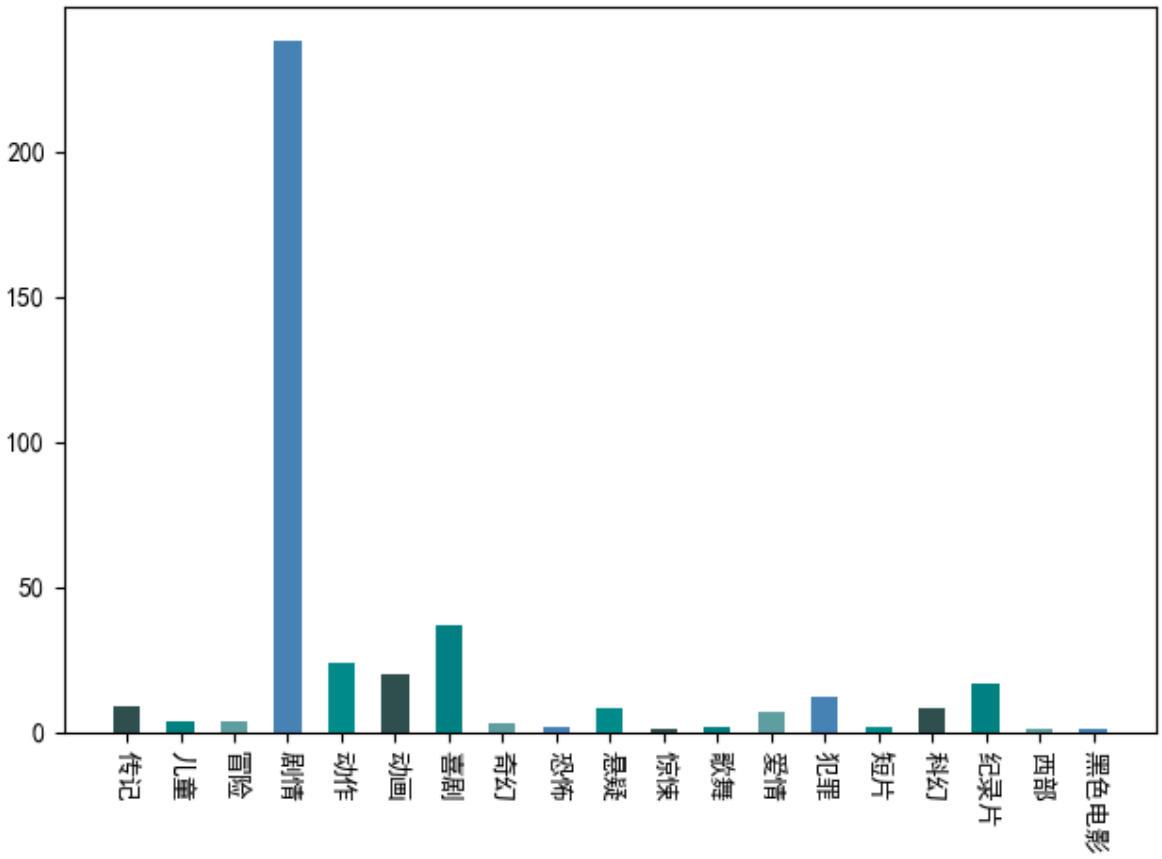
douban (it has a smaller database):

month	num
1	22
2	26
3	17
4	22
5	44
6	27
7	30
8	27
9	55
10	23
11	36
12	52

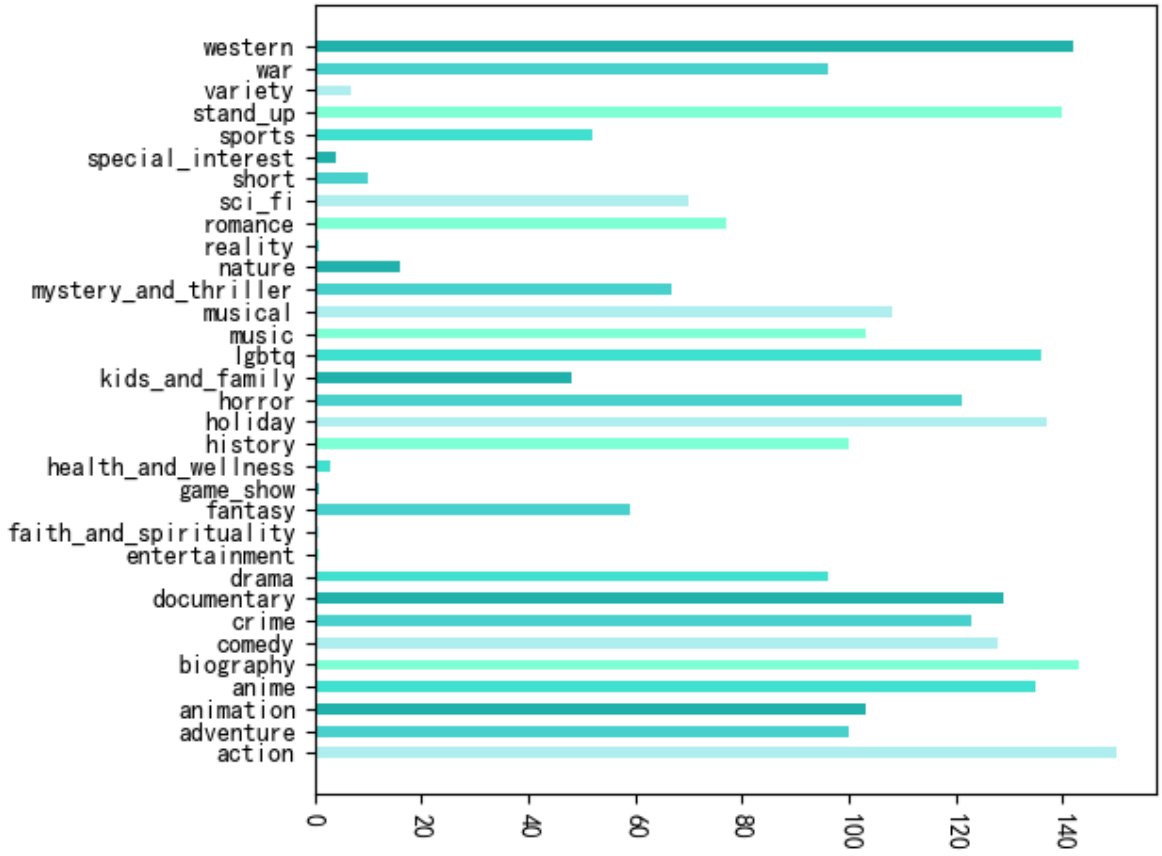


5.2 The type graph:

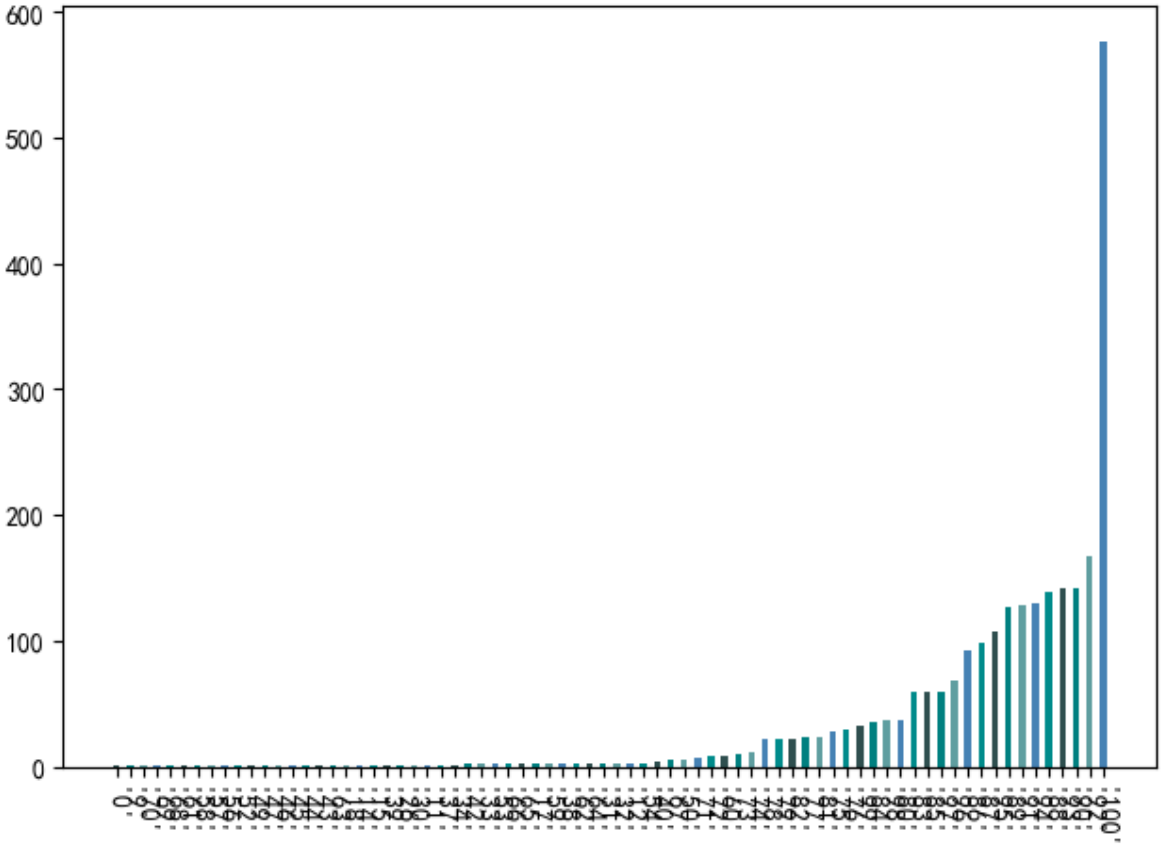
douban, obviously biased ,but not surprising as they are famous for in favor of story type :



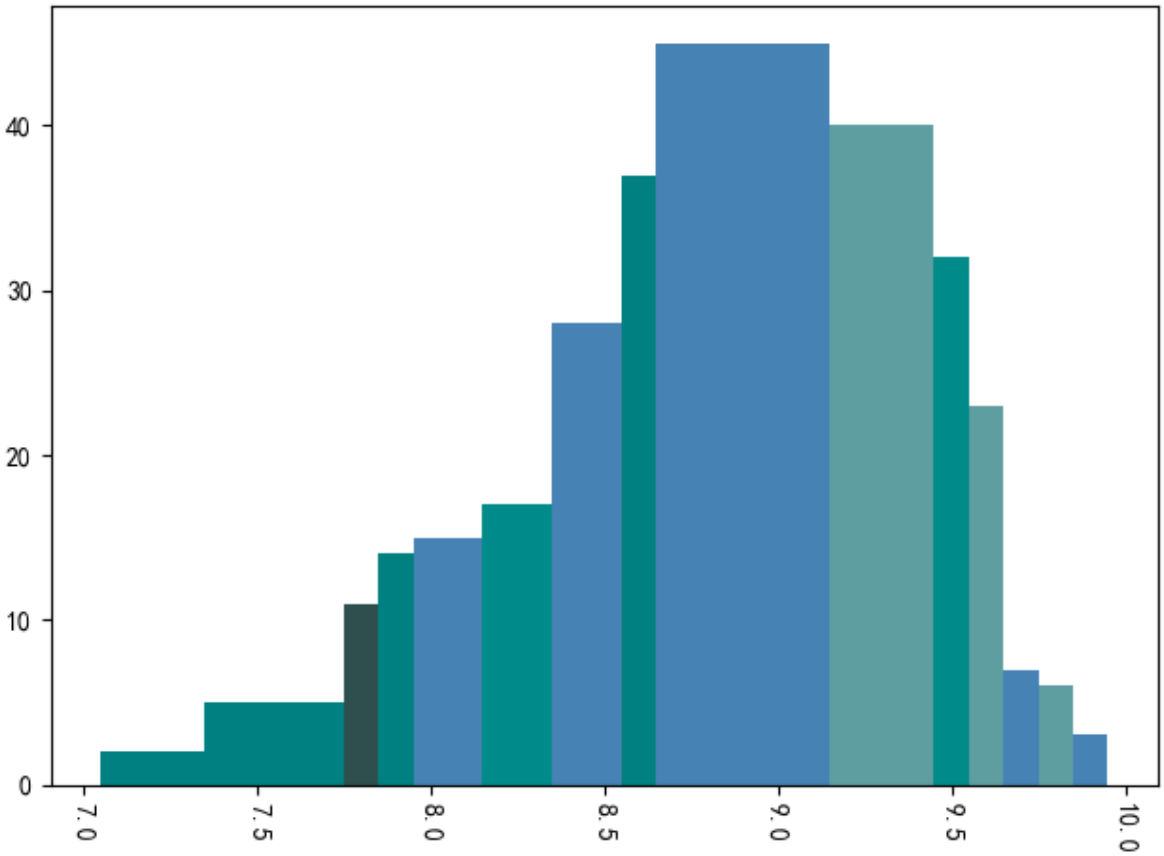
rotten tomato:



and the scores from rotten totmato:



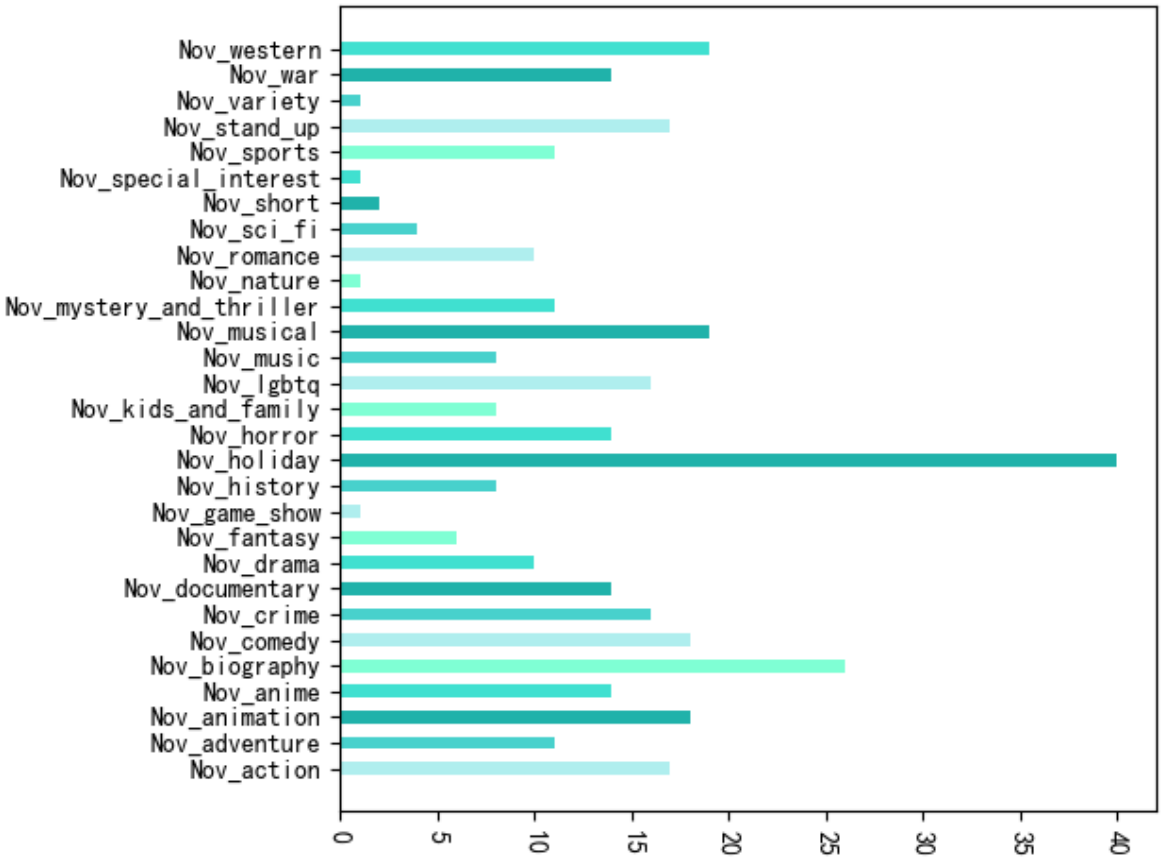
scores from douban:



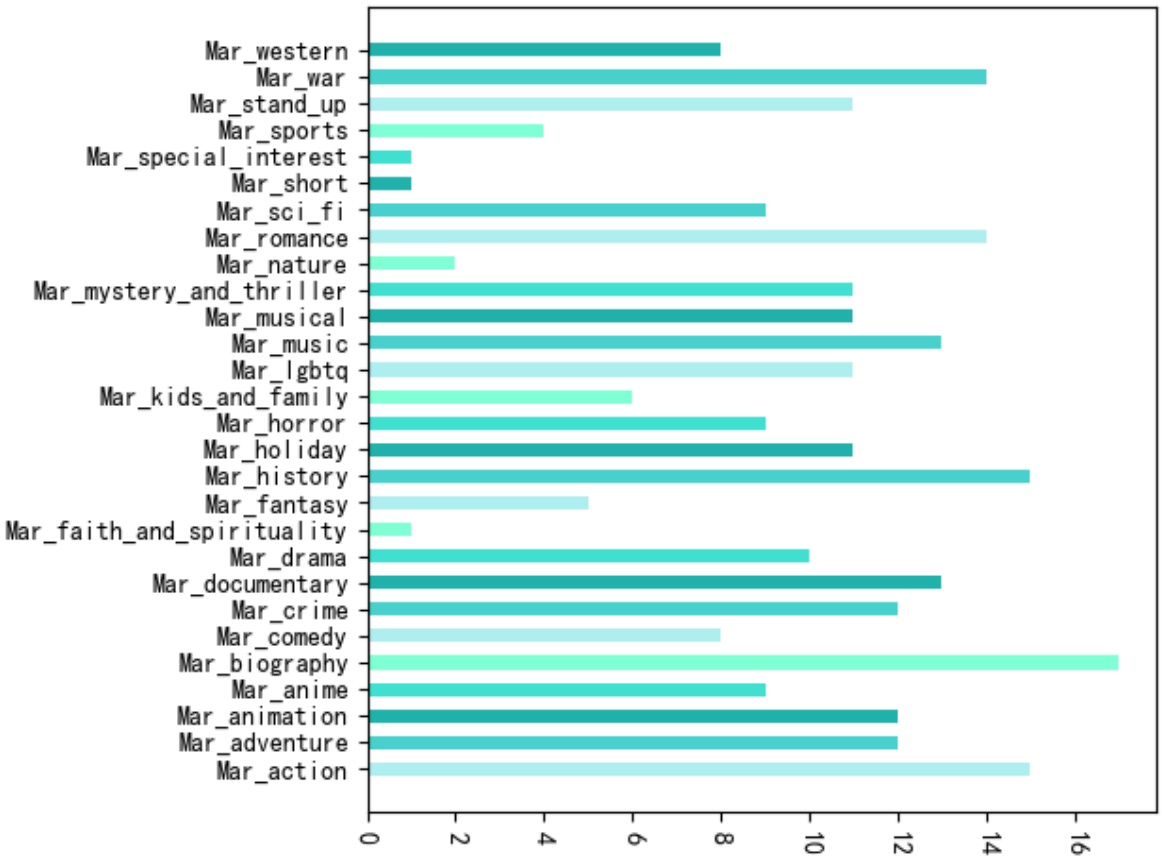
5.3 the two columns with selections:

since douban is so biased, these examples are made from rotten tomato only

Nov:



Mar:

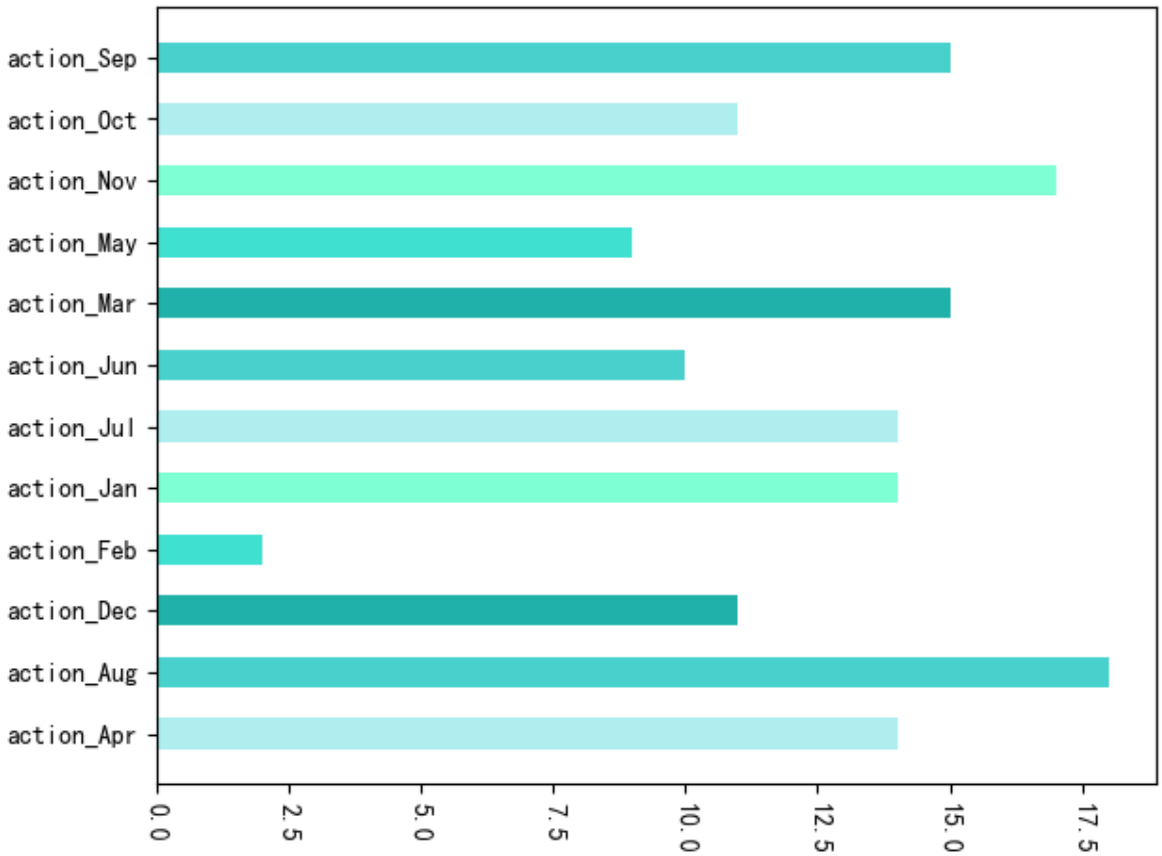


and codes to generate any month:

```
import dim_to_ads as tdm
tdm.dimension_concact_ads('month','type','rotten','Mar')
graph_bar('ads_month_type_Mar_counts_rotten.csv','month_type',barh=True)
```

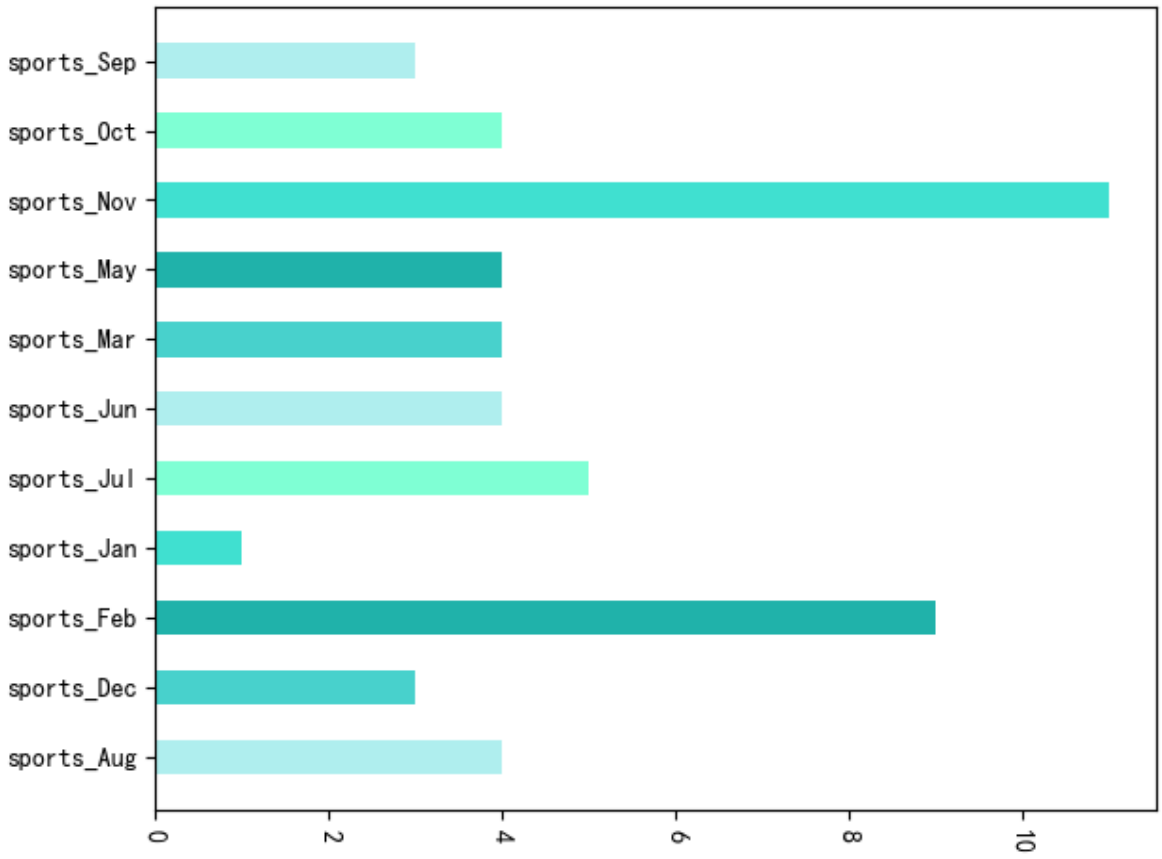
using generalize method to create reverse version:

```
tdm.dimension_concact_ads('type','month','rotten','action')
graph_bar('ads_type_month_action_counts_rotten.csv','type_month',barh=True)
```



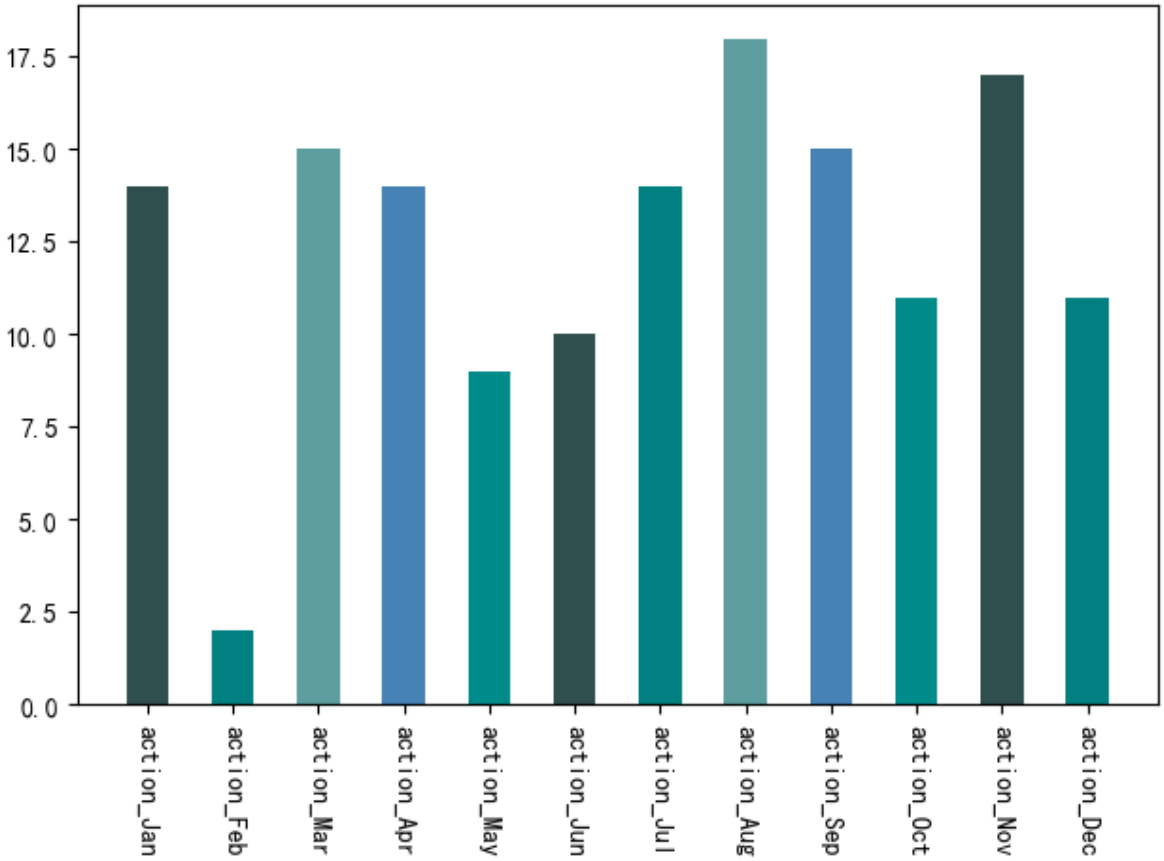
and of course any type:

```
import dim_to_ads as tdm
tdm.dimension_concact_ads('type','month','action')
graph_bar('ads_type_month_action_counts_rotten.csv','type_month',barh=True)
```



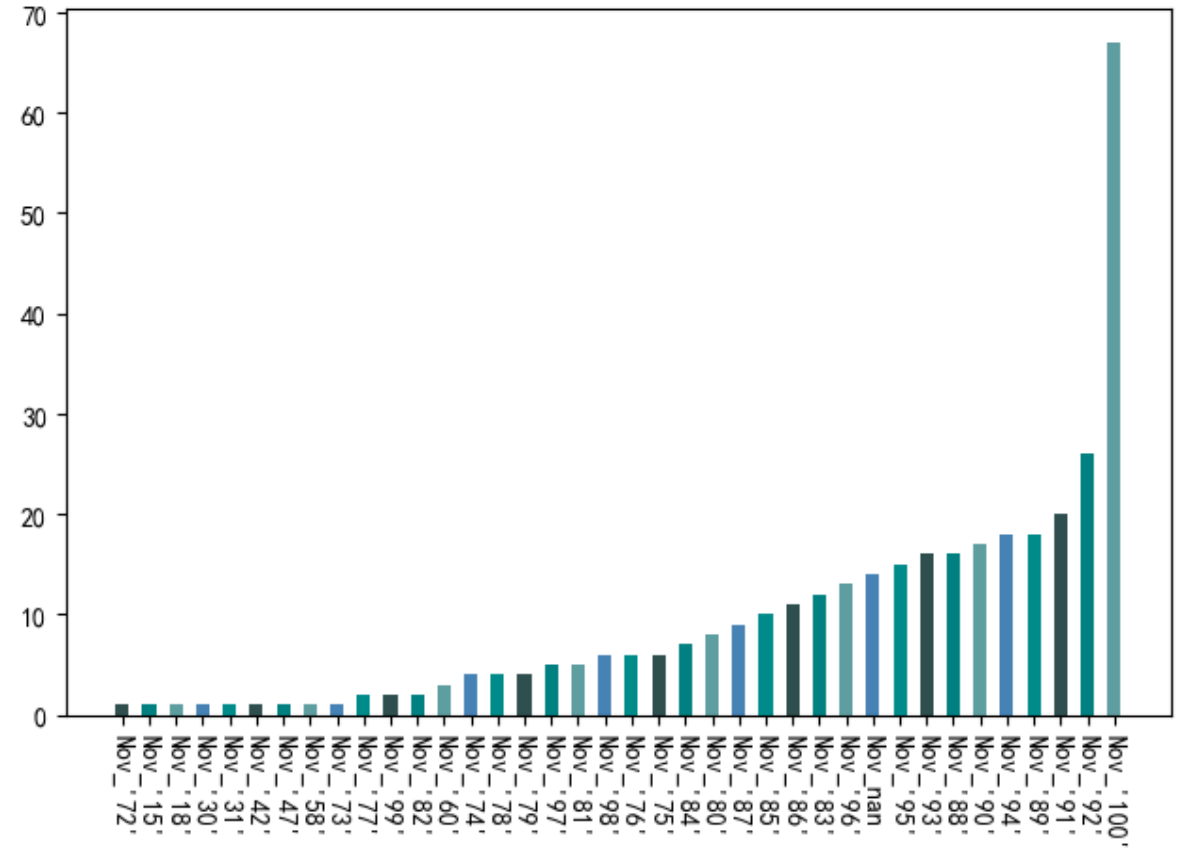
the above graph is unsorted, then sorted method and in line graph as follows:

```
tdm.sort_ads_type_month('data/ads/ads_type_month_action_counts_rotten.csv','acti
on')
graph_bar('ads_type_month_action_counts_rotten.csv','type_month',plot=True)
```

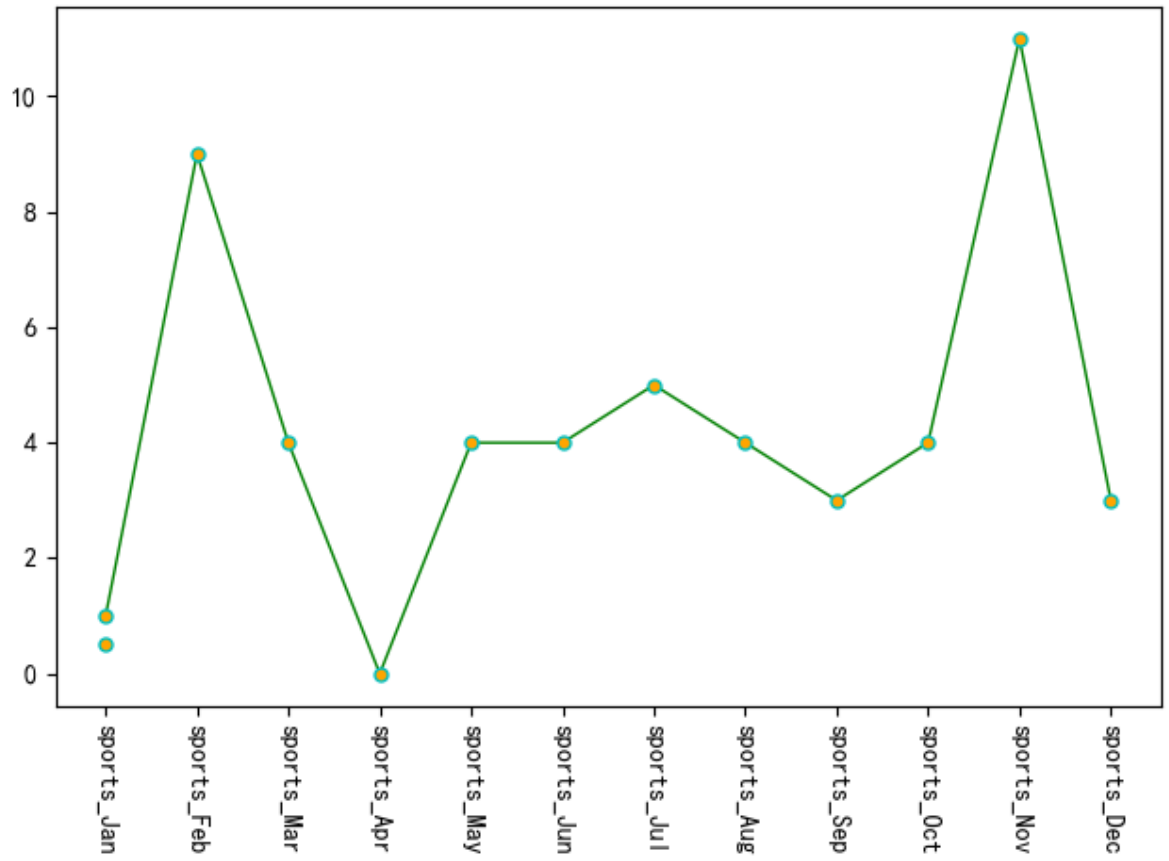


and ofcource any other kind:

```
# and month scores:
tdm.dimension_concact_ads('month','audience_score','rotten','Nov')
graph_bar('ads_month_audience_score_Nov_counts_rotten.csv','month_audience_score
',barh=False,sort=True)
```

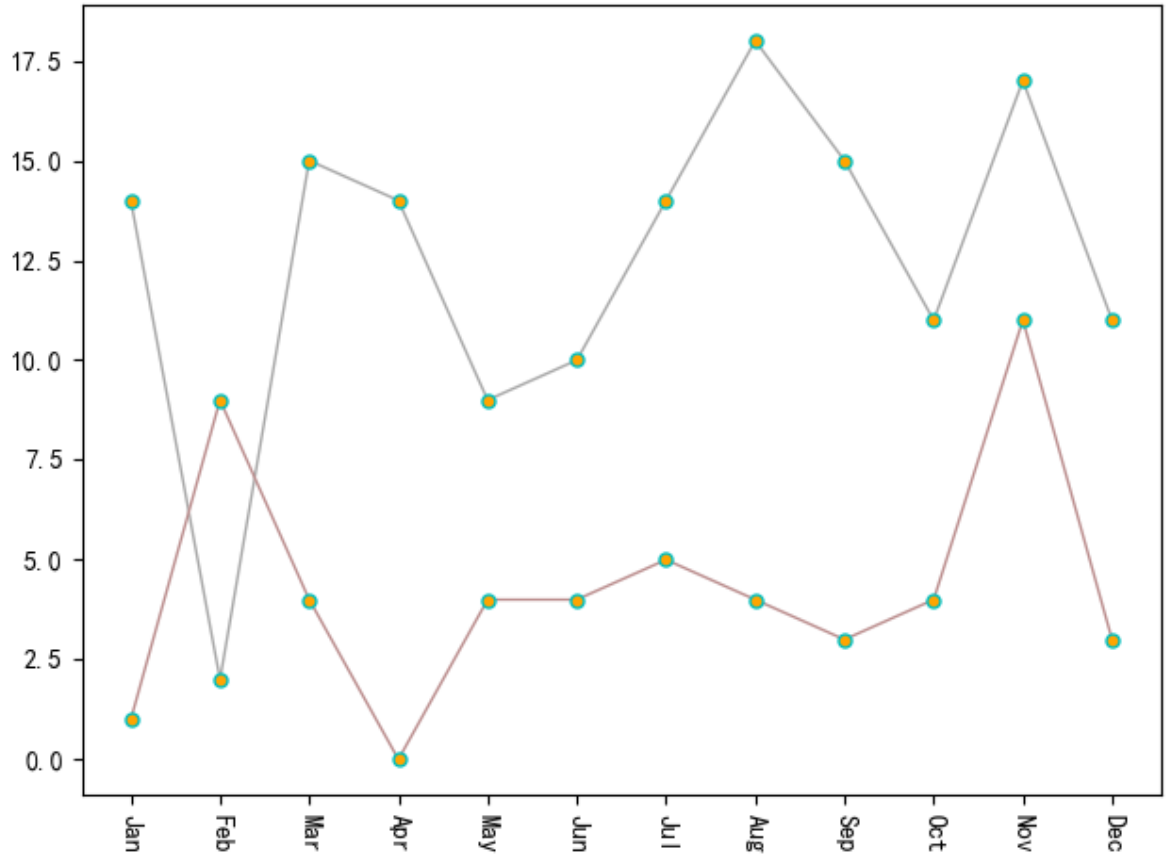


and line graph:



and two line in one graph:

```
# and adding the two kinds together:
multi_line(['ads_type_month_action_counts_rotten.csv','ads_type_month_sports_cou
nts_rotten.csv'])
```



6. Acknowledgements:

Learned a lot from pandas : https://pandas.pydata.org/docs/

and lambda: https://www.geeksforgeeks.org/python-lambda/

and the two data source:

douban : https://movie.douban.com/chart

rotten : https://www.rottentomatoes.com/browse/movies_at_home/sort:popular?page=1

for the encoding : https://blog.csdn.net/toshibahuai/article/details/79034829