**Mini-Project Report**



|  |  |
| --- | --- |
| Student Name: | Gavali Deshabhakt Nagnath |
| Roll Number: | 202CD005 |
| Department: | Mathematical and Computational Sciences |
| Specialization: | Computational and Data Science |
| Subject: | Big Data Analytics |
| Topic: | House Price Prediction Using Machine Learning |
| Course Instructor: | Dr. Pushparaj Shetty |
| Date: | 08/05/2021 |

***Index***

|  |  |  |
| --- | --- | --- |
| ***Chapter No.*** | ***Chapter*** | ***Page No.*** |
| I | Introduction & Methodology | 3 |
| II | Data pre-processing | 8 |
| III | Generating data required for further analysis | 13 |
| IV | Finding Answers | 19 |
| V | Conclusion & References | 30 |
|  | Entire Python Code | 31 |

Chapter – I

**Introduction & Methodology**

**Introduction:**

This project is about analyzing the co-authorship network database gathered from Scopus. Scopus is a website which provide us many ways of filtering the data. As per the question requirements we need to filter data accordingly on Scopus itself and using python program when we required to do so. So, we have to go through numerous preprocessing steps that are required to generate a data frame before actually initializing our analyzing steps.

Now let’s introduce you with the Scopus layout through which we actually got our database. Scopus is affiliate with our institute central library where we need to mention the name of document category we searching for, in our case it was ‘Artificial Intelligence’, which prompt us to various other subcategories such as open access, year, author name, subject area, document type, source title, publication stage, keyword, affiliation, funding sponsor, country, source type and language. Each of these categories will allow us for further refining of the data set.

So finally after filtration on Scopus we did some further database refining using ‘pandas’ library of python like dropping of some unnecessary columns, renaming of the columns for easy access and order in which they appear, which ultimately give us this final structure of database which includes every author available on Scopus from 15 different countries i.e. ‘Australia’ , ’Canada’, ’China’, ‘France’, ‘Germany’, ‘India’, ‘Iran’, ‘Italy’, ‘Japan’, ‘Netherland’, ‘South Korea’, ‘Spain’, ‘Taiwan’, ‘UK’, ‘US’, with around 59k records and around 120k unique authors. Our database is created by merging of database of 15 countries into one single database with column names as follows-

***‘Authors’***- This column contains the name of multiple authors separated by comma in one single string format.

***‘Title’***- It contains information of the topic on which authors work upon, each with unique name.

***‘Year’***- Give information of year of publication.

***‘Cited by’***- It shows how many have referred the author’s publication for their works.

***‘Country’***- It represent the country from where the publication is done.

***‘Sponsors’***- It shows whether or not the authors got the sponsorship for their work.

With this information in our complete database, we are in good position to analyze our database the problem statements given to us. We use ‘python’ and some of the libraries such as ‘pandas’, ‘matplotlib’, ‘NumPy’, ‘\_pickel’ etc. for better representation and analysis.

**Methodology:**

Now we will see the methodology that we have adopted to complete this project.

**Step 1:** Downloading database from scopus. We downloaded database separately for each country and again for funding details.

**Step 2:** Preprocessing of Main database:

In this process we removed some unnecessary attributes which are not required for our analysis and then we merged all the separate databases of countries in to one single CSV file.

**Step 3:** Generating Necessary Data Required for Further Analysis

In this process we have done operations to generate necessary data for further analysis i.e., to solve questions.

This step includes sub steps

1. Generating Authors list

2. Creating Python dictionary with Author name as key and his/her corresponding database as value

3. Creating list of Indian authors

4. Generating Foreign Authors list

5. Creating a dictionary with foreign author as key and number of papers published by him with Indian authors as value

**Step 4:** Finding answers of given questions.

Chapter - II

**Downloading Database**

As discussed earlier we used Scopus repository to download database. Scopus repository allows user to download database with maximum 2000 entries directly from their website and for databases with more than 2000 entries, they send a link to provided email and address and then one can download the database from that link. Also, in Scopus repository 20000 is the cap on entries that can be downloaded. This means that the user cannot download database with more than 20000 entries. Also, as the number of entries in the selected database increases, the unavailability of corresponding data attributes decreases. This means that if we select a large database for downloading then we might not receive some of the selected data attributes in the download file. In our case this attribute was ‘funding details’. When we selected ‘funding details’ attribute for large database then the repository did not give corresponding data in the file. To overcome this difficulty, we downloaded databases corresponding to only funding details separately and then merged it to our main database.

We considered ‘artificial intelligence’ as a keyword and ‘engineering’ as filed area for downloading database. So, our query was as below (without double quotes)

”TITLE-ABS-KEY(Artificial Intelligence) AND ( LIMIT-TO ( DOCTYPE,"ar" ) OR LIMIT-TO ( DOCTYPE,"ENGI" ) OR LIMIT-TO ( DOCTYPE, “Artificial Intelligence" ) )”

Now after analyzing the questions that we’re supposed to answer, we concluded that the data should be downloaded country-wise, meaning if we download data country wise then we will be able to add a country column in respective countries’ database and merge those together (row-wise) to get main database.

In country-wise databases, we found that If two authors publish a paper together and they are from different countries then the same paper will be present in databases of both countries’. This makes our work a lot simpler, because then we can analyze the database ‘as per’ country also.

So, with all this in mind we used following steps to download the database.

1. Registration on scopus with institute email id.
2. Verifying email id and logging on to scopus
3. Selecting search parameter as keyword (in our case it was ‘Artificial Intelligence’)
4. Selecting document type as ‘Article’
5. Selecting subject area as ‘Engineering’
6. Then we got the query as mentioned above.
7. Using above query, we downloaded database country-wise. E.g. if we’re to download database of India then the query would be something like this

TITLE-ABS-KEY(Artificial Intelligence) AND ( LIMIT-TO ( AFFILCOUNTRY,"India" ) ) AND ( LIMIT-TO ( DOCTYPE,"ar" ) ) AND ( LIMIT-TO ( SUBJAREA,"ENGI" ) )

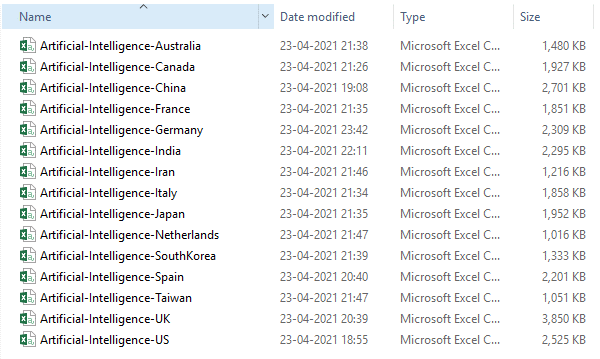
1. Changing country name and downloading corresponding data

We these steps we downloaded data country-wise but there was no ‘funding details’ attribute in any of the database. As we needed that attribute to find number of grants given to our filed ‘Artificial Intelligence’, we repeated same steps but used following query,

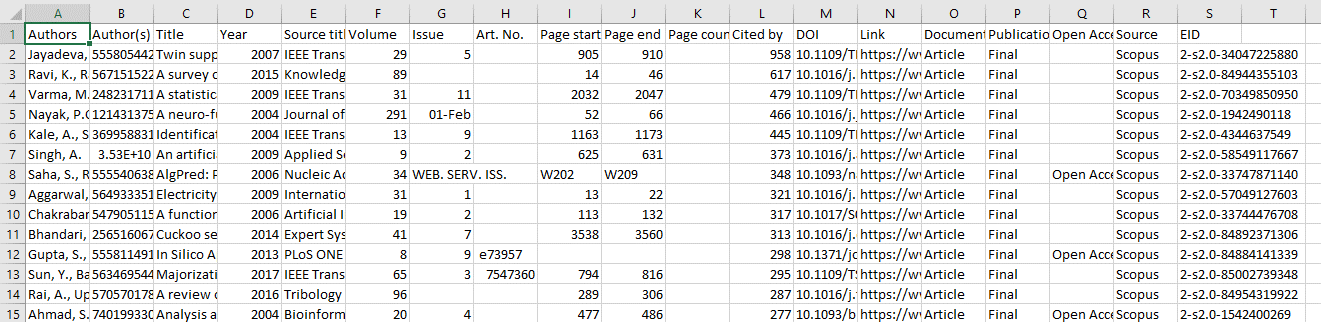
TITLE-ABS-KEY(Artificial Intelligence) AND ( LIMIT-TO ( DOCTYPE,"ar" ) OR LIMIT-TO ( DOCTYPE,"ENGI" ) OR LIMIT-TO ( DOCTYPE,"Artificial Intelligence" ) ) AND ( LIMIT-TO ( FUND-SPONSOR,"National Natural Science Foundation of China" ) OR LIMIT-TO ( FUND-SPONSOR,"National Institutes of Health" ) OR LIMIT-TO ( FUND-SPONSOR,"U.S. Department of Health and Human Services" ) OR LIMIT-TO ( FUND-SPONSOR,"National Science Foundation" ) OR LIMIT-TO ( FUND-SPONSOR,"European Commission" ) )

At the end of these steps we had following files at our hand,

1. **Main database files**

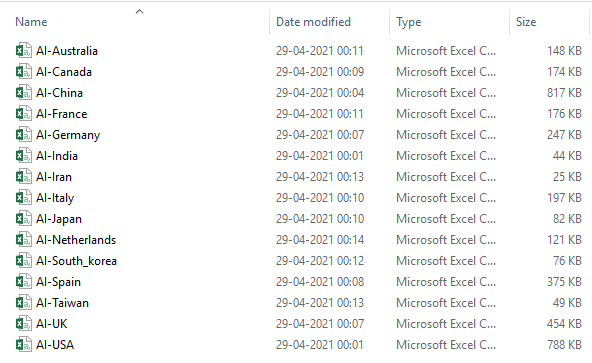


These files had following attributes,

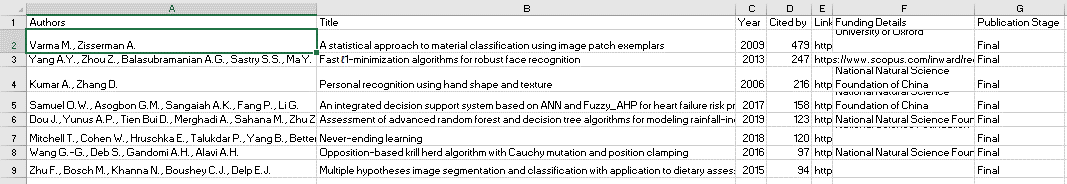


\*We can see from above image that there is no ‘funding details’ column in downloaded database.

1. **Database files with Funding Column containing funding column**



These files had following attributes



\*We can see from above image that there is a ‘funding details’ column in downloaded database.

Chapter – III

**Data Pre-processing**

1. **Preprocessing of Main database**

As discussed in earlier chapter we had downloaded database separately for each country and again for funding details. Also, the separated databases had some unnecessary attributes. So, these databases cannot be used directly for data analysis. That’s why we have to do preprocessing on those.

For data-preprocessing and also data analysis, we used python programming language. Python has many libraries which makes data analysis much easier. We imported the database of each countries using ‘read\_csv()’ function of pandas library and did pre-processing on it. Following is the block of code that we used to do so.

base\_dir = '/mnt/General\_Stuff/Study Stuff/Documents/CDS/Sem-II/BigDataAnalytics/BigData-Programs/Mini-Project/Data'

This base\_dir variable contains a string which path to database directory.

dir\_lst = []

for data\_file in os.listdir(base\_dir):

dir\_lst.append(os.path.join(base\_dir,data\_file))

Using this block of code merged filename.extension in database directory with base\_dir variable and stored the resultant string into a python list.

e.g. merging ‘Artificial-Intelligence-India.csv' with base\_dir variable gives

“/mnt/General\_Stuff/Study Stuff/Documents/CDS/Sem-II/BigDataAnalytics/BigData-Programs/Mini-Project/Data/Artificial-Intelligence-India.csv”

this string. We are doing this for every file in database directory and appending it to python list (i.e. dir\_lst). Thus, the python list will contain strings which are paths of databases of each country.

After getting python to databases we imported those using pandas’s read\_csv() function and removed un-necessary columns from them using drop() function. Then we also added a country column in respective database containing corresponding country name.

We removed following columns/attributes from database,

'Author(s) ID', 'Source title', 'Volume', 'Issue', 'Art. No.', 'Page start', 'Page end', 'Page count', 'DOI', 'Link', 'Document Type', 'Publication Stage', 'Open Access', 'Source', 'EID'

Also, we filled all empty cells in database with 0.

For doing all this we have used following block of code.

country\_lst = ['Australia','Canada','China','France','Germany','India','Iran','Italy','Japan','Netherlands','South Korea','Spain','Taiwan','United Kingdom', 'United States']

df\_lst = []

for data\_file, country in zip(dir\_lst, country\_lst):

print(country)

df\_tmp = pd.read\_csv(data\_file)

df\_tmp = df\_tmp.drop(['Author(s) ID' ,'Source title','Volume', 'Issue', 'Art. No.', 'Page start', 'Page end', 'Page count', 'DOI', 'Link', 'Document Type', 'Publication Stage','Open Access', 'Source', 'EID'],axis='columns')

df\_tmp = df\_tmp.fillna(0)

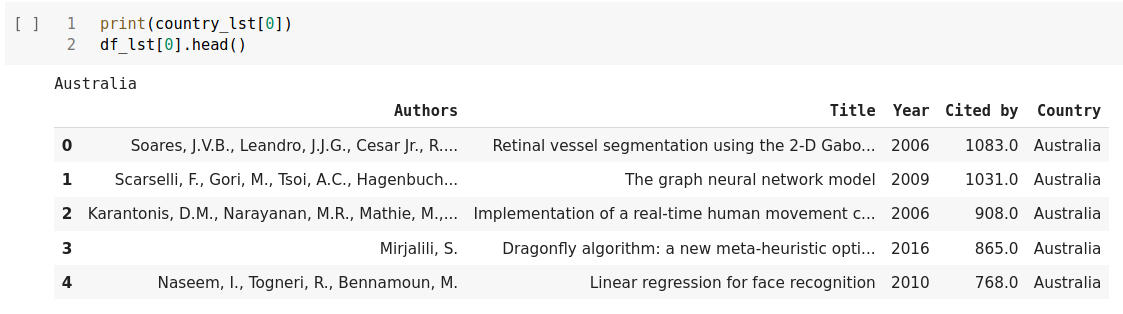
df\_tmp['Country'] = country

df\_lst.append(df\_tmp)

As explained above this code does following things,

1. Takes a database from database directory
2. Removes un-necessary attributes from it
3. Fills all empty cells with 0
4. Adds country column with corresponding country name
5. Appends the resulting database to python list.

At the end of this block we will have a python list containing filtered database of all countries.

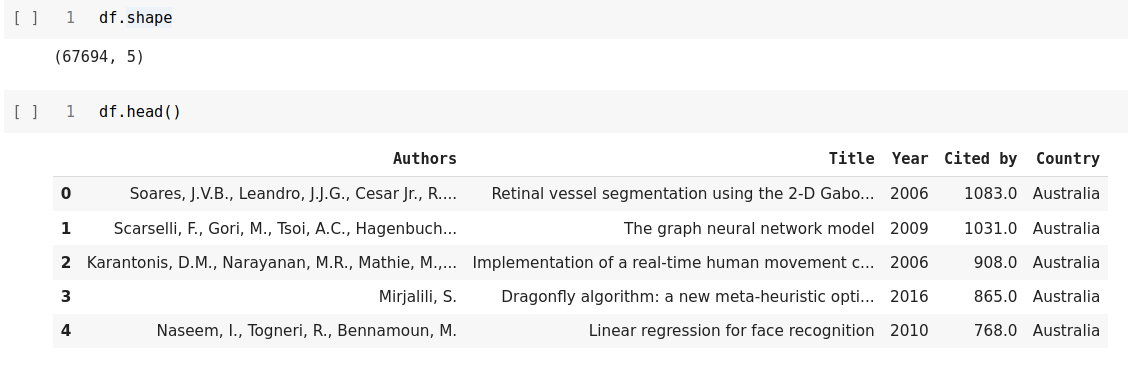


\*here we can see the df\_lst python list contains database of each country at different indices.

But we cannot use this python list for data analysis directly. We have to consolidate data in a single pandas dataframe. For that we used concat() function of pandas.

df = pd.concat(df\_lst)

The ‘df’ variable now contains database of all countries in consolidated form. We can see that in below image.



So, our final database has **total 67694 entries**. It contains data of **15-countries**. The names of those countries are as follows:

1. Australia
2. Canada
3. China
4. France
5. Germany
6. India
7. Iran
8. Italy
9. Japan
10. Netherlands
11. South Korea
12. Spain
13. Taiwan
14. United Kingdom
15. United States

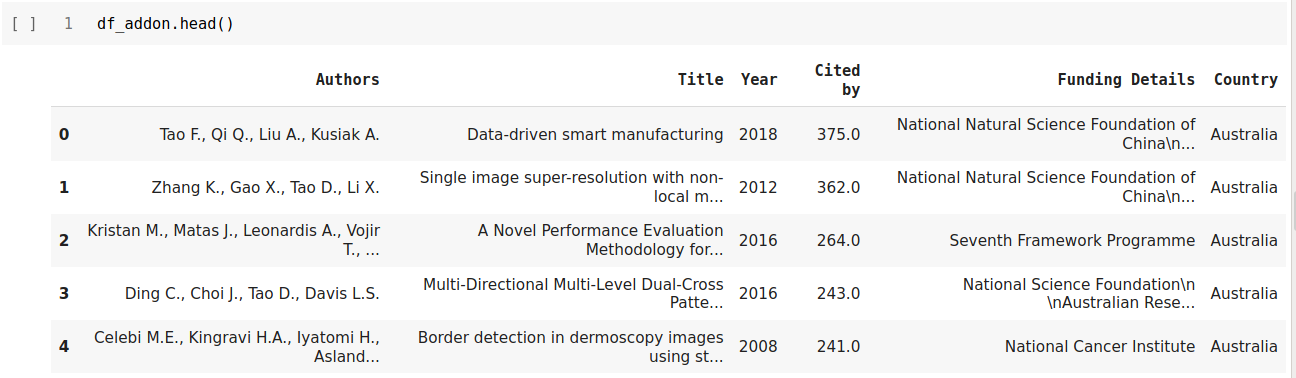
But this database does not have ‘Funding Details’ attribute in it. So, in order to add this attribute to our database, we used following procedure.

1. Changed base\_dir variable to path of database with sponsorship details directory

i.e.

base\_dir = '/mnt/General\_Stuff/Study Stuff/Documents/CDS/Sem-II/BigDataAnalytics/BigData-Programs/Mini-Project/Data\_with\_sponsor\_details'

1. Used same data pipeline as above for filtering data.
2. At the end of second step we obtained a pandas dataframe with sponsorship column in it.



1. Then we added a ‘Funding Details’ column in our main database and initialized it to zero using following code.

df['Funding\_Details'] = 0

1. After adding funding details column, we compared two databases by title and changed funding details accordingly.

titles\_old = list(df.Title)

titles\_new = list(df\_addon.Title)

count = 1

for i in range(len(titles\_new)):

for j in range(len(titles\_old)):

if titles\_new[i]==titles\_old[j]:

df.iloc[j,-1] = df\_addon.iloc[i,-2]

print(f'{count}\t{titles\_new[i]}')

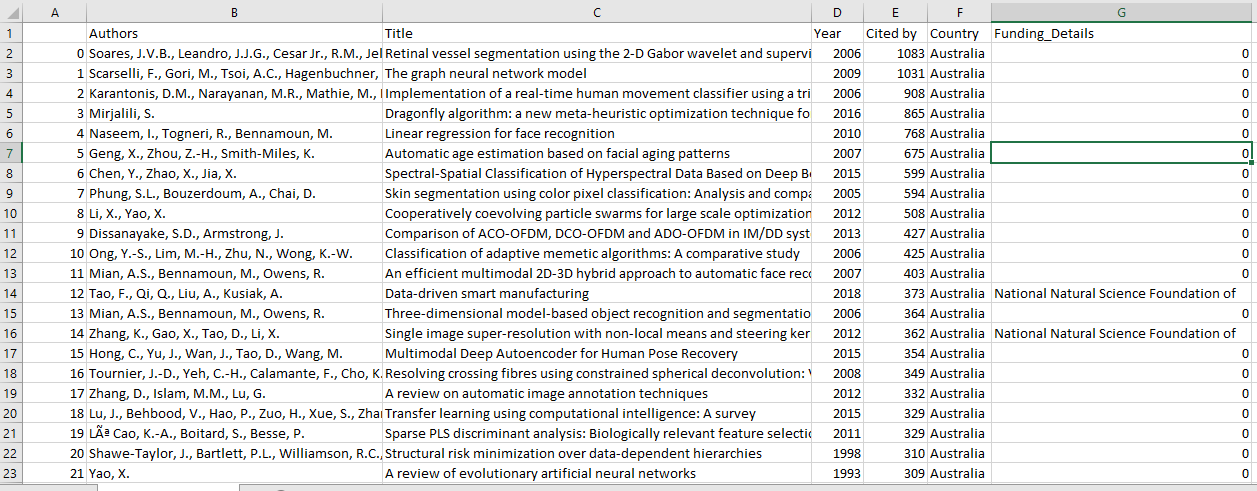
count += 1

This block of code creates 2- python lists containing titles from main and additional database and compares those titles one by one. If we find a match then we just replacing value at funding details column in main database with value at funding details column in additional database corresponding to matched title

After completing all the above steps we generated a dataframe which suits our need. This dataframe was then stored in csv file using to\_csv() function of panadas library, so that it can be used for further ananlysis easily.

df.to\_csv('Complete\_database.csv')

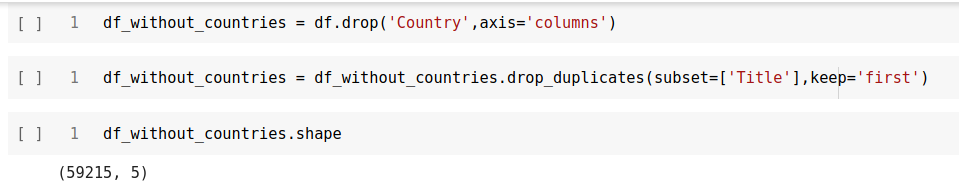
The resulting csv file had following attributes in it.



Chapter – III

**Generating Data Required for Further Analysis**

Now we’ll discuss about the files that we generated during further data-preprocessing phase. We imported the previously created csv file and stored it in panadas data-frame. One thing to note here is that, if two authors from different countries work together on a same research paper then that paper will be present corresponding to both countries. Thus, we need to tackle with these duplicate entries. For this we used drop\_duplicate() function of pandas. Here’s code of that,



Here we can see that the unique entries are 59,215 instead of 67,694. Now we’ll use this dataframe (i.e. df\_without\_countries) for further analysis.

Now we’ll generate files/data required for further analysis.

1. **Generating Authors list**

In database that we have almost all research papers are published by authors working together. So, authors column corresponding to each contains names of all the others who worked on that particular paper. But for data analysis we need names of individual authors. So, here we we’ll generate those names and store it in python list and in turn in a text file. For this we have used following code

1. First, we’ll extract collective authors from database



1. As we can see in output of above code, the authors\_lst holds authors names with end name and initial separted by commas. We’ll now generate individual authors name from authors\_lst python list.

The above code works as follow:

set\_authors = ['Jayadeva', 'Khemchandani, R.', 'Chandra, S.']

for i in range(len(authors\_lst)):

authors\_sub\_lst = authors\_lst[i].split(',')

authors\_sub\_lst\_mod = []

if authors\_sub\_lst == 'Jayadeva, Khemchandani, R., Chandra, S.'.split(','):

continue

for j in range(0,len(authors\_sub\_lst)-1,2):

authors\_sub\_lst\_mod.append(authors\_sub\_lst[j].strip()+','+authors\_sub\_lst[j+1])

for author in authors\_sub\_lst\_mod:

if(author not in set\_authors):

set\_authors.append(author)

print(f'{i}\t{author}')

1. Create a python list with name set\_authors
2. Go through the authors\_lst one by one, pick one string from it.
3. Split the string at ‘,’ and then merge the split string with increment of 2. (because end name and initials of author are also separated by ‘,’)
4. If the merged names are not in set\_authors then append it to set\_authors (this avoids repetitions of author names).

At the end of this code we’d have a python list containing names of all authors in our database.

As creating this list takes lot of time, so to avoid repetitive computation of this list we stored it in a text file using following code,

with open('Authors\_list.txt','w') as filehandle:

filehandle.writelines("%s\n" % author for author in set\_authors)

filehandle.close()

Also, we can re-use it whenever it is required in further analysis.

This file can be read using following code,

set\_authors = []

with open('Authors\_list.txt','r') as filehandle:

filecontents = filehandle.readlines()

for line in filecontents:

# remove linebreak which is the last character of the string

author = line[:-1]

# add item to the list

set\_authors.append(author)

**Note:**

Here we initialized the set\_authors with “'Jayadeva', 'Khemchandani, R.', 'Chandra, S.'” because this entry does not follow our assumption that the end name and initials are separated by ‘,’. (Mr. Jayadeva did not provide his initials)

1. **Creating Python dictionary with Author name as key and his/her corresponding database as value**

For doing this we used following code,

dct\_author\_database = {}

count=0

for author in set\_authors:

print(f'{count}\t{author}')

df\_auth = pd.DataFrame(columns = ['Authors', 'Title', 'Year', 'Cited\_by','Funding Details'] )

filt= df\_without\_countries['Authors'].str.contains(author, na=False)

df\_auth= df\_without\_countries.loc[filt,'Authors':].reset\_index(drop=True)

dct\_author\_database[author] = df\_auth

count += 1

This code does following things:

1. Creates a python dictionary with name dct\_author\_database
2. Go through each author in set\_authors list
3. Create an empty dataframe say df\_tmp
4. Go through df\_wihtout\_countries dataframe and append all rows containing authors name (picked at step 2) to the empty dataframe ie. df\_tmp.
5. And then store author name and df\_tmp as key value pair in dct\_author\_database dictionary.

At the end of this code we would have database of each other separated by author name. Storing it in dictionary makes it easy to fetch.

This thing can also be done using following code.

dct\_author\_database = {}

count = 0

for author in set\_authors:

print(f'{count}\t{author}')

df\_auth = pd.DataFrame(columns = ['Authors', 'Title', 'Year', 'Cited\_by'] )

for authors in authors\_lst:

if author in authors:

df\_tmp = df\_without\_countries[df\_without\_countries.Authors==authors]

df\_auth = df\_auth.append(df\_tmp,ignore\_index=True)

dct\_author\_database[author] = df\_auth

count += 1

Generating this dictionary takes around 50-60 minutes. To avoid this repetitive computation, we’ll store this library to a txt file. For that we’ll use ‘pickle’ library.

Here’s code for that,

with open('Author\_database\_dictonary.txt','wb') as file:

file.write(pickle.dumps(dct\_author\_database))

file.close()

To read this file and get back our dictionary we can use following code,

dct\_author\_database\_from\_file = {}

with open('Author\_database\_dictonary.txt','rb') as file:

dct\_author\_database\_from\_file = pickle.load(file)

file.close()

With this code we can load the dictionary at any point in further data analysis, without much computation and save our time.

1. **Creating list of Indian authors**

authors\_from\_ind\_database = list(df[df.Country=='India']['Authors'].unique())

set\_of\_authors\_from\_indian\_database = ['Jayadeva', 'Khemchandani, R.', 'Chandra, S.']

count = 0

for authors in authors\_from\_ind\_database:

authors\_sub\_lst = authors.split(',')

if authors\_sub\_lst == 'Jayadeva, Khemchandani, R., Chandra, S.'.split(','):

continue

authors\_sub\_lst\_mod = []

for i in range(0,len(authors\_sub\_lst)-1,2):

authors\_sub\_lst\_mod.append(authors\_sub\_lst[i].strip()+','+authors\_sub\_lst[i+1])

for author in authors\_sub\_lst\_mod:

if(author not in set\_of\_authors\_from\_indian\_database):

set\_of\_authors\_from\_indian\_database.append(author)

print(f'{count}\t{authors}')

count += 1

This code follows same algorithm as code used for generating set\_authors list in 1st bullet.

We then saved this file to text file using following code,

with open('India\_authors\_list.txt','w') as filehandle:

filehandle.writelines("%s\n" % author for author in set\_of\_authors\_from\_indian\_database)

filehandle.close()

But, as we know that this list might contain names of some foreign authors, that’s why we checked list manually and removed foreign authors names from it. (it took time because the list generated had around 11K names in it, so we divided it among our team)

After removing foreign authors names from the text file, we can load it using following command.

set\_of\_indian\_authors\_from\_file = []

with open('Indian\_authors\_list.txt','r') as filehandle:

filecontents = filehandle.readlines()

for line in filecontents:

# remove linebreak which is the last character of the string

author = line[:-1]

# add item to the list

set\_of\_indian\_authors\_from\_file.append(author)

1. **Generating Foreign Authors list**

As we have a Indian authors list and a list of all authors it’s fairly easy to generate foreign authors list from it using following code.

set\_of\_foreign\_authors = []

count = 0

for author in set\_authors:

if author not in set\_of\_indian\_authors\_from\_file:

set\_of\_foreign\_authors.append(author)

print(f'{count}\t{author}')

count += 1

Storing it in a text file:

with open('Foreign\_authors\_list.txt','w') as filehandle:

filehandle.writelines("%s\n" % author for author in set\_of\_foreign\_authors\_from\_file)

filehandle.close()

Reading from Text file:

set\_of\_foreign\_authors\_from\_file = []

with open('Foreign\_authors\_list.txt','r') as filehandle:

filecontents = filehandle.readlines()

for line in filecontents:

# remove linebreak which is the last character of the string

author = line[:-1]

# add item to the list

set\_of\_foreign\_authors\_from\_file.append(author)

1. **Creating a dictionary with foreign author as key and number of papers published by him with Indian authors as value**

We used this dictionary to find out author with highest co-authorship with Indian author. The dictionary can be generated using following code,

dct\_foreign\_author\_coauth\_count = {}

for foreign\_author in set\_of\_foreign\_authors\_from\_file:

dct\_foreign\_author\_coauth\_count[foreign\_author] = 0

count = 0

for foreign\_author in set\_of\_foreign\_authors\_from\_file:

row\_foreign\_auth\_list = dct\_author\_database\_from\_file[foreign\_author].iloc[:,0].values

for indian\_author in set\_of\_indian\_authors\_from\_file:

for authors in row\_foreign\_auth\_list:

if indian\_author in authors:

dct\_foreign\_author\_coauth\_count[foreign\_author] += 1

print(f'{count} {foreign\_author} {dct\_foreign\_author\_coauth\_count[foreign\_author]}')

count += 1

s

Storing it in file:

with open('Foreign\_auth\_and\_their\_publication\_count\_with\_india\_authors\_dct.txt','wb') as file:

file.write(pickle.dumps(dct\_foreign\_author\_coauth\_count))

file.close()

Reading it from file:

dct\_foreign\_author\_coauth\_count\_from\_file = {}

with open('Foreign\_auth\_and\_their\_publication\_count\_with\_india\_authors\_dct.txt','rb') as file:

dct\_foreign\_author\_coauth\_count\_from\_file = pickle.load(file)

file.close()

These are all the files that we’ll require in finding answers to our questions. In next chapter we’ll use these files for further analysis.

Chapter – IV

**Finding Answers**

In this chapter we’ve found answers of all the questions asked in our min-project pdf. For this we’ve used all the previously generated files. First, we created a function to sort dictionaries. This function is used in lot of next steps. Its definition is as follows,

def sort\_dict(dct,parameter,order='Ascending'):

if parameter=='key':

if(order=='reverse'):

sorted\_tuples = sorted(dct.items(),key=lambda item:item[0],reverse=True)

return {k:v for k,v in sorted\_tuples}

else:

sorted\_tuples = sorted(dct.items(),key=lambda item:item[0])

return {k:v for k,v in sorted\_tuples}

else:

if(order=='reverse'):

sorted\_tuples = sorted(dct.items(),key=lambda item:item[1],reverse=True)

return {k:v for k,v in sorted\_tuples}

else:

sorted\_tuples = sorted(dct.items(),key=lambda item:item[1])

return {k:v for k,v in sorted\_tuples}

This function takes 3 arguments as input,

1. Python dictionary
2. Parameter for sorting (either key or value)
3. Order of sorting (by default its ascending)

This function returns a sorted python dictionary as per provided parameters.

Now coming to questions,

1. **Highest cited author and his h-index (from the world)**

For finding this we’ve used following code,

author\_with\_highest\_citations = ""

max\_citations = 0

for author in set\_authors:

cites = dct\_author\_database[author]['Cited\_by'].sum()

if max\_citations<cites:

author\_with\_highest\_citations = author

max\_citations = cites

df\_of\_highest\_cited\_author = dct\_author\_database[author\_with\_highest\_citations]

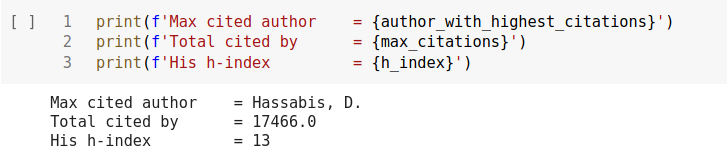
rows,cols = df\_of\_highest\_cited\_author.shape

# h\_index = min(rows,df\_of\_highest\_cited\_author['Cited\_by'].min())

avg\_citations\_of\_author\_with\_highest\_citations = max\_citations/rows

h\_index = min(rows,avg\_citations\_of\_author\_with\_highest\_citations)

Its output is as follows,



Here we can see that highest cited author is **‘Hassabis, D.’** and his total citations and h-index are as in output.

1. **Highest publication author**

For finding this we’ve used following code,

author\_with\_highest\_publication = ""

max\_publication\_count = 0

for author in set\_authors:

rows, columns = dct\_author\_database[author].shape

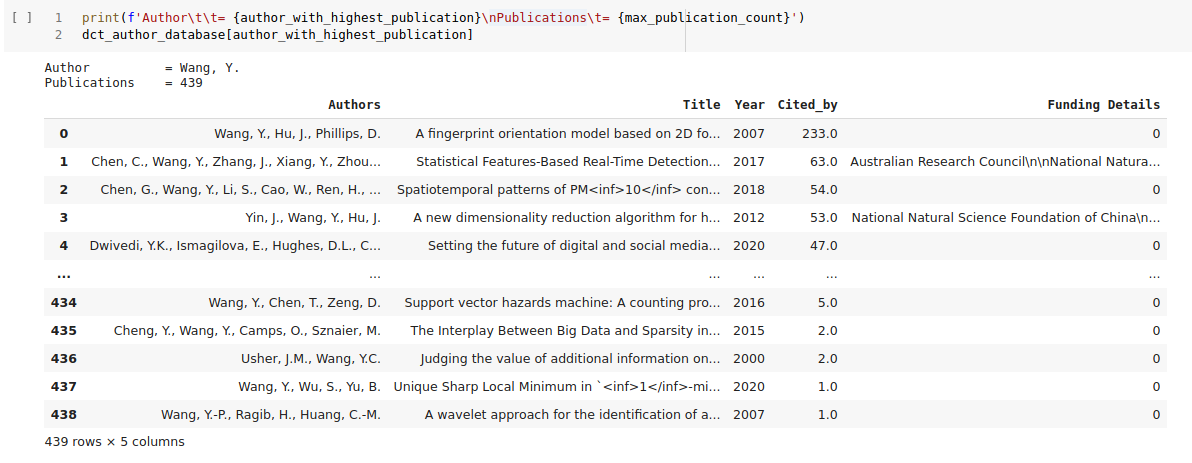
if rows>max\_publication\_count:

max\_publication\_count=rows

author\_with\_highest\_publication = author

# print(f'{author} \t{rows}')

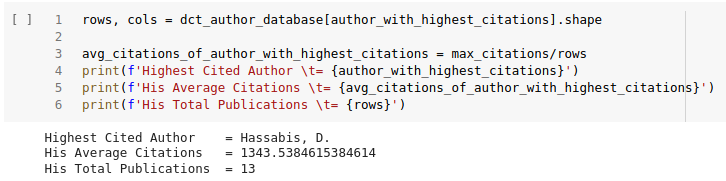
Its output is a below,



From output we can see that ‘**Wang Y.’** has highest publications **(total 439)**.

1. **Highest cited authors avg. citations, and the country name**

We already found highest cited author in first question. Now we can simply make use of our dct\_author\_database dictionary to find out his average citations and for finding country we can simply use google.

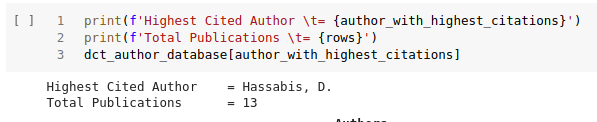


From output we can see that **the total publications of our highest cited author are 13** and his average citations are **around 1344.**

And from google we found that the author is from **United Kingdom.**

**d. Total number of publications of the highest cited author**

Answered in previous question itself.



**e. Total publication in year**

Our database contains information from 1964 to 2021. The total publication year-wise can be found using following code.

year\_lst = sorted(list(df['Year'].unique()))

country\_lst = list(df['Country'].unique())

df\_without\_duplicates = df.drop\_duplicates(subset=['Authors','Title'],keep='first')

dct\_df\_per\_year\_publications = {}

for year in year\_lst:

dct\_df\_per\_year\_publications[year], cols = df\_without\_duplicates[df\_without\_duplicates.Year==year].shape

dct\_df\_per\_year\_publications = sort\_dict(dct\_df\_per\_year\_publications,'Values','reverse')

Its output is as below,

1988: 531,

1993: 485,

1990: 457,

1991: 394,

1992: 357,

1987: 351,

1986: 165,

1985: 117,

1984: 70,

1983: 25,

1982: 23,

1977: 17,

1980: 13,

1973: 12,

1978: 12,

1979: 12,

1981: 11,

1974: 8,

1975: 8,

1976: 8,

1971: 7,

1972: 6,

1970: 3,

1962: 1,

1963: 1,

1964: 1,

1965: 1,

1968: 1,

1969: 1}

{2020: 6222,

2019: 4688,

2018: 4122,

2016: 3341,

2014: 3159,

2015: 3117,

2017: 3049,

2021: 2688,

2013: 2277,

2004: 2191,

2012: 2010,

2008: 1933,

2009: 1565,

2011: 1559,

2010: 1542,

2006: 1464,

2007: 1401,

2003: 1357,

2005: 1346,

2000: 793,

1989: 734,

1999: 730,

2001: 726,

1997: 718,

1996: 713,

1994: 703,

2002: 702,

1998: 697,

1995: 601,

**f. Total citation per year**

We can use same algorithm as above to find total cites per year,

dct\_citations\_per\_year = {}

for year in year\_lst:

dct\_citations\_per\_year[year]=df\_without\_duplicates[df\_without\_duplicates.Year==year]['Cited\_by'].sum()

dct\_citations\_per\_year = sort\_dict(dct\_citations\_per\_year,'Values','reverse')

dct\_citations\_per\_year

Its output is as below,

1990: 9800.0,

1993: 8807.0,

1991: 8453.0,

1988: 8385.0,

1987: 5439.0,

1986: 3652.0,

1980: 2616.0,

1985: 1930.0,

1977: 1644.0,

2021: 1568.0,

1979: 997.0,

1984: 812.0,

1971: 446.0,

1973: 256.0,

1978: 252.0,

1976: 215.0,

1983: 208.0,

1982: 202.0,

1975: 186.0,

1970: 83.0,

1981: 76.0,

1972: 49.0,

1974: 33.0,

1963: 17.0,

1962: 16.0,

1969: 4.0,

1964: 2.0,

1965: 2.0,

1968: 0.0}

{2008: 86548.0,

2016: 85426.0,

2015: 81343.0,

2005: 78325.0,

2004: 75544.0,

2007: 73896.0,

2006: 73652.0,

2014: 69065.0,

2009: 68433.0,

2018: 66871.0,

2013: 63245.0,

2017: 61390.0,

2010: 61118.0,

2012: 56435.0,

2011: 55280.0,

2019: 42281.0,

2000: 36085.0,

2003: 32783.0,

1999: 28128.0,

1998: 26193.0,

2002: 25418.0,

2001: 25176.0,

1997: 21638.0,

2020: 21386.0,

1995: 18774.0,

1994: 18694.0,

1996: 16943.0,

1989: 12072.0,

1992: 10540.0,

**g. Author(country) having highest co-authorship with Indian authors**

Its code as below,

mx\_pubs\_with\_indian\_authors = 0

foreign\_auth\_corr\_to\_mx\_pubs\_with\_indian\_authors = ""

for author in set\_of\_foreign\_authors:

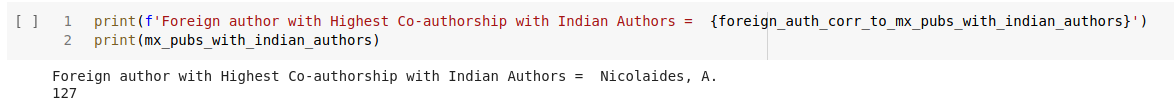
pubs = dct\_foreign\_author\_coauth\_count[author]

if mx\_pubs\_with\_indian\_authors < pubs:

mx\_pubs\_with\_indian\_authors = pubs

foreign\_auth\_corr\_to\_mx\_pubs\_with\_indian\_authors = author

Its output is as below,



It turns out that author **Nicolaides, A.** has highest co-authorship with Indian authors and he is from **"Vascular Screening and Diagnostic Centre, University of Nicosia, Nicosia, Cyprus"** (from google)

**h. Highest cited author from India and the university**

For this we make use of our previously created Indian authors list and dct\_author\_database dictionary. Its code goes as below,

max\_cites\_of\_indian\_author = 0

highest\_cited\_indian\_author = ""

for author in set\_of\_indian\_authors:

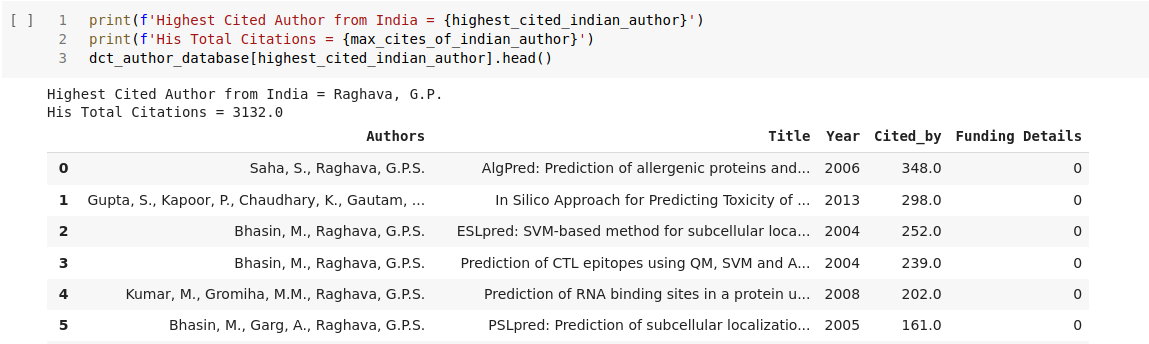
cites = dct\_author\_database[author]['Cited\_by'].sum()

if max\_cites\_of\_indian\_author < cites:

max\_cites\_of\_indian\_author = cites

highest\_cited\_indian\_author = author

Its output is as follow,



1. **Comparative year wise article publication analysis of India, China and USA**

For this question we first separated each countries database from our main database using following code.

df\_india = df[df.Country=='India'].copy().reset\_index(drop=True)

df\_china = df[df.Country=='China'].copy().reset\_index(drop=True)

df\_usa = df[df.Country=='United States'].copy().reset\_index(drop=True)

Then we created dictionary with country name as key and publication count as value. For this we use following code.

Then we made x and y variables as follow,

x\_data = [list(dct\_india\_year\_publications.keys()),list(dct\_china\_year\_publications.keys()),list(dct\_usa\_year\_publications.keys())]

y\_data = [list(dct\_india\_year\_publications.values()),list(dct\_china\_year\_publications.values()),list(dct\_usa\_year\_publications.values())]

Now coming to comparison part, we used line and bar graphs for comparison.

1. **Bar graph – Publication count vs Year**

Code for plotting bar graph is as follow,

# plt.rcParams['figure.figsize'] = [20,10]

fig = plt.figure(figsize=[20,10])

X = np.arange(len(year\_lst))

X = X + year\_lst[0]

fig = fig.add\_axes([0,0,1,1])

fig.bar(X + 0.00, list(dct\_india\_year\_publications.values()), label='India', color='r', width =0.25)

fig.bar(X + 0.25, list(dct\_china\_year\_publications.values()), label='China', color='g', width =0.25)

fig.bar(X + 0.50, list(dct\_usa\_year\_publications.values()), label='United States', color='b', width =0.25)

fig.legend(loc='upper left',fontsize=18)

plt.xticks(rotation = 'vertical')

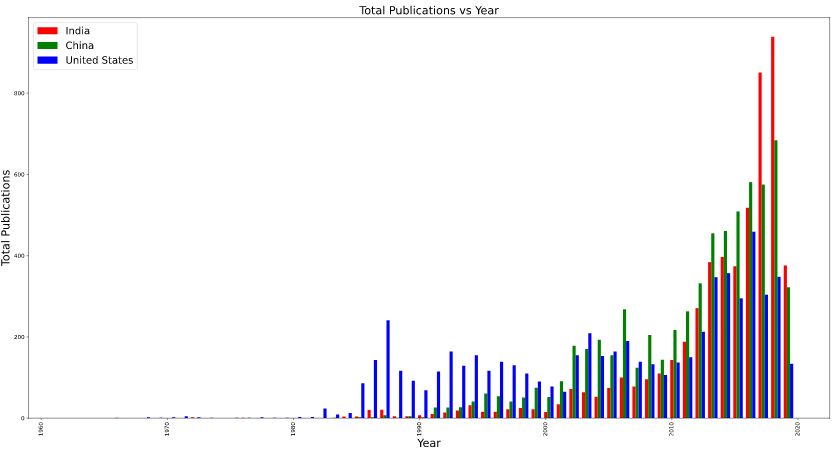
plt.title('Total Publications vs Year',fontsize=20)

plt.xlabel('Year',fontsize=20)

plt.ylabel('Total Publications',fontsize=20)

plt.show()

We get following bar chart as output,



From bar graph we can see variation in publication year-wise for all 3-countries. India has highest publications in year 2019, China in 2017 and USA in 1988.

1. **Line Graph – Publication Count vs Year**
2. **Rough curve**

We first plotted the x and y data directly and got below curve

Code:

fig = plt.figure(figsize=[20,10])

plt.plot(x\_data[0], y\_data[0], label='India', color='r')

plt.plot(x\_data[1], y\_data[1], label='China', color='g')

plt.plot(x\_data[2], y\_data[2], label='USA', color='b')

plt.xlabel('Year',fontsize=20)

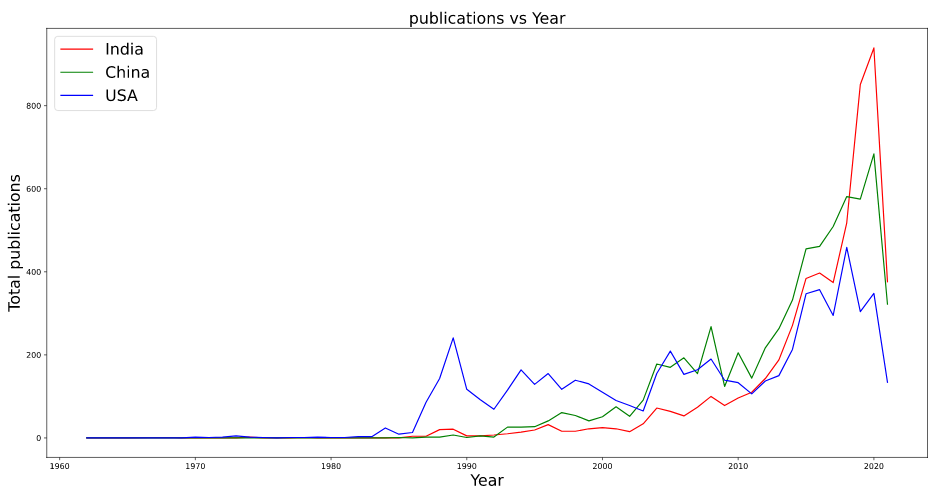
plt.ylabel('Total publications',fontsize=20)

plt.title('Publications vs Year',fontsize=20)

plt.legend(loc='upper left',fontsize=20)

plt.show()

Output:



As we can see, the plot is not at all smooth, but it follows same trajectory as bar graph.

To generate smoother plot, we use interpolate library from scipy package.

1. **Smooth plot**

Code:

y\_new = []

x\_new = []

for i in range(3):

x\_new\_tmp = np.linspace(year\_lst[0],year\_lst[0]+len(x\_data[0]),1000)

x\_new.append(x\_new\_tmp)

spline = interpolate.make\_interp\_spline(x\_data[i], y\_data[i])

y\_new.append(spline(x\_new\_tmp))

This code generates around 1000 points in the give interval using interpolation function. Using these new x and y we plotted new line graph.

Code:

fig = plt.figure(figsize=[20,10])

plt.plot(x\_new[0], y\_new[0], label='India', color='r')

plt.plot(x\_new[1], y\_new[1], label='China', color='g')

plt.plot(x\_new[2], y\_new[2], label='USA', color='b')

plt.xlabel('Year',fontsize=20)

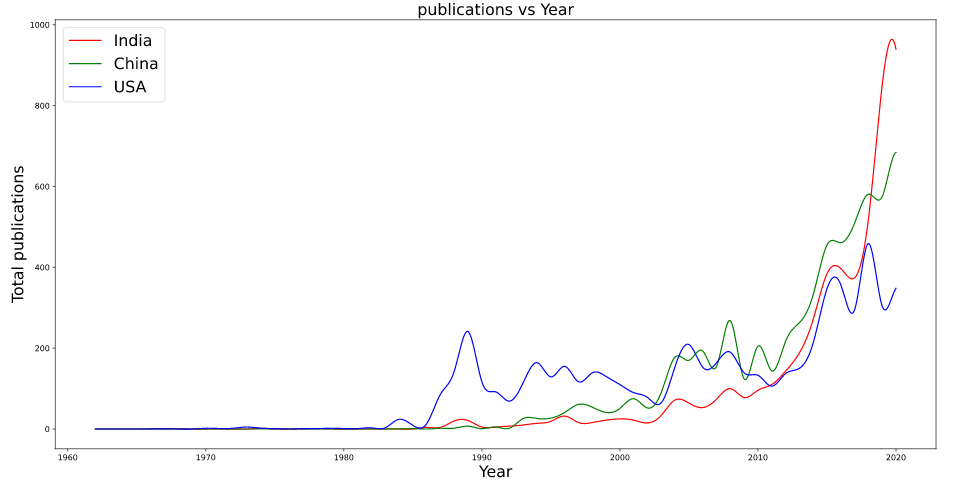
plt.ylabel('Total publications',fontsize=20)

plt.title('Publications vs Year',fontsize=20)

plt.legend(loc='upper left',fontsize=20)

plt.show()

Output:



Thus we obtained a smoother curve and here we can clearly see that it follows same trajectory as bar graph.

1. **Total number of grants given to the field**

Grant is basically receiving funding from external organizations or institutions. In our case it was very easy to calculate grants, because we had already added funding column in our database. We then simply compared that ‘funding details’ column with zero and find out number of grants.

Code:

grants, cols = df\_without\_duplicates[df\_without\_duplicates['Funding\_Details']!='0'].shape

Output:



As we can see from output, the grants received by field are **4,771**.

1. **Country wise total number of publications**

This is again simply because we had list of countries and also in our database there is country column. Using these two things we generated a dictionary with key as country name and value as publication count. For this we use following code,

Code:

dct\_country\_publications = {}

for country in country\_lst:

rows, columns = df[df.Country==country].shape

dct\_country\_publications[country] = rows

dct\_country\_publications = sort\_dict(dct\_country\_publications,'Value','reverse')

Output:

{'United Kingdom': 8994,

'China': 6401,

'United States': 6104,

'India': 5383,

'Germany': 5186,

'Spain': 4759,

'Canada': 4486,

'Japan': 4324,

'Italy': 4214,

'France': 4133,

'Australia': 3361,

'South Korea': 3026,

'Iran': 2720,

'Taiwan': 2430,

'Netherlands': 2173}

Chapter – V

**Conclusion and References**

**Conclusion:**

1. With this mini-project we had hands-on experience of data analysis using python.
2. We learned following things,
3. How to download database from scopus
4. What should we keep in mind while downloading database
5. How to pursue problem properly so that it can be solved easily
6. We also had hands on experience on python libraries such as matplotlib, pandas, NumPy, SciPy, \_pickle, etc.
7. Also, we got to know what tasks a person has to perform as a data scientist.

**References:**

1. [Scopus](https://www.scopus.com/search/form.uri?display=basic)
2. [Pandas Documentation](https://pandas.pydata.org/docs/)
3. [Numpy Documentation](https://numpy.org/doc/1.20/)
4. [Matplotlib Documentation](https://matplotlib.org/)
5. [GeeksForGeeks](https://www.geeksforgeeks.org/)
6. [StackOverFlow](https://stackoverflow.com/)