Module 3

Data Analysis and Visualisations

To justify that our selection is being representative: We use MathSciNet which indices top level journals worldwide (https://www.ams.org/journals/ (https://www.ams.org/journals/)) and is internationally known and respected. To the best of our knowledge, there are no biases in not indexing all parts of the world, or any other issues, and we fully trust the accuracy, sufficiency and credibility of the data we retrieved from MathSciNet.
We will start by loading all the librarieswe will need, some additional libraries will be loaded futher on towards the end of the notebook to draw the maps, but for now we need those below. If you are not sure that you have all of them installed, please use the `requirements.txt` provided in the general project directory with the following code:
`pip install -r requirements.txt`

In [1]:
```
import sys
import time
import re
import matplotlib as plt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
%matplotlib inline
import cufflinks as cf
```

Then we load the `.csv` files poduced by the Data Preparation Notebook.

In [2]:
```
merged_df = pd.read_csv('full.csv') # the whole data
gap_df = pd.read_csv('gap.csv')     # GAP citations
pac_df = pd.read_csv('pac.csv')     # GAP Packages citations
```

We can choose plotting style according to preference.

In [3]:
```
print(plt.style.available) # prints a list of what we have available
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background',
'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright',
'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgri
d', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seab
orn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-whit
e', 'seaborn-whitegrid', 'tableau-colorblind10']
```

In [4]:
```
plt.style.use('fivethirtyeight') # we set the style as per our choice
```

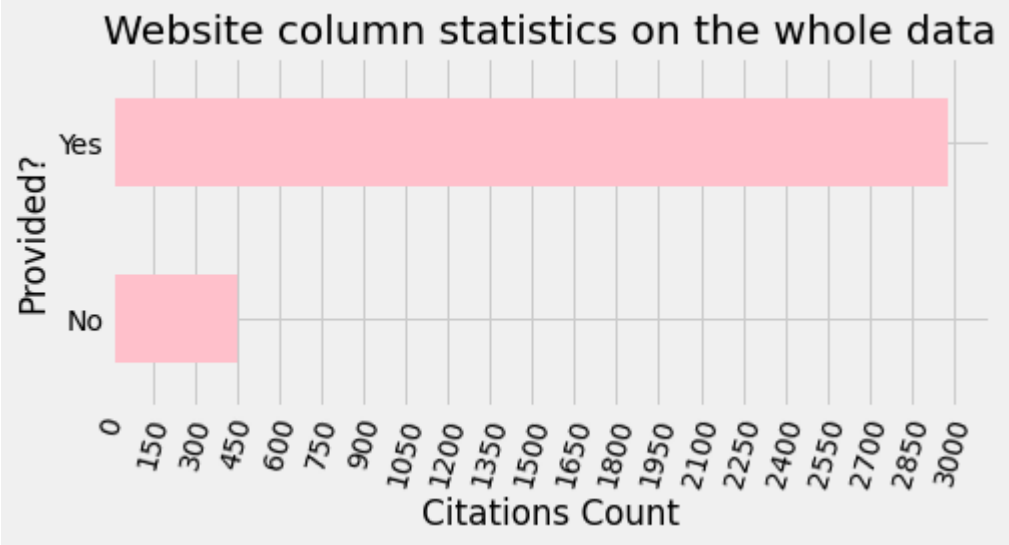
In [5]:
```
pd.options.display.float_format = "{:,.2f}".format # to show only two digits afte
```

Now we start analysing single or combined features of the data, using tables and visyalisations.

Website

We are starting with simple visualisations. The code below counts the occurances of the two values for the Website column, then displays the result in a horizontal bar chart.

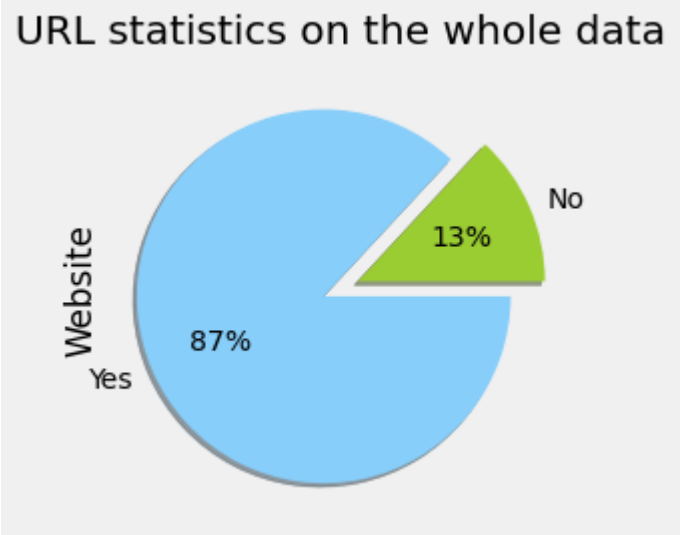
In [6]:
```
web_chart = merged_df['Website'].value_counts()
web_chart = web_chart.sort_values(ascending=True)
web_chart.plot(kind="barh", figsize=(7, 3),
               title='Website column statistics on the whole data',
               color='pink')
plt.xticks(range(0, 3100, 150))
plt.xticks(rotation=75)
plt.xlabel('Citations Count')
plt.ylabel('Provided?')
plt.show()
```



From approximately 3500 entries in our full dataset, 3000 GAP citations have provided some sort of Website. It is good that the majority of citers have done soe, because the presence of website reference in a software citation is vital, especially when a reader of citing publication wishes to learn more about the software, its versions, releases, price and usage.

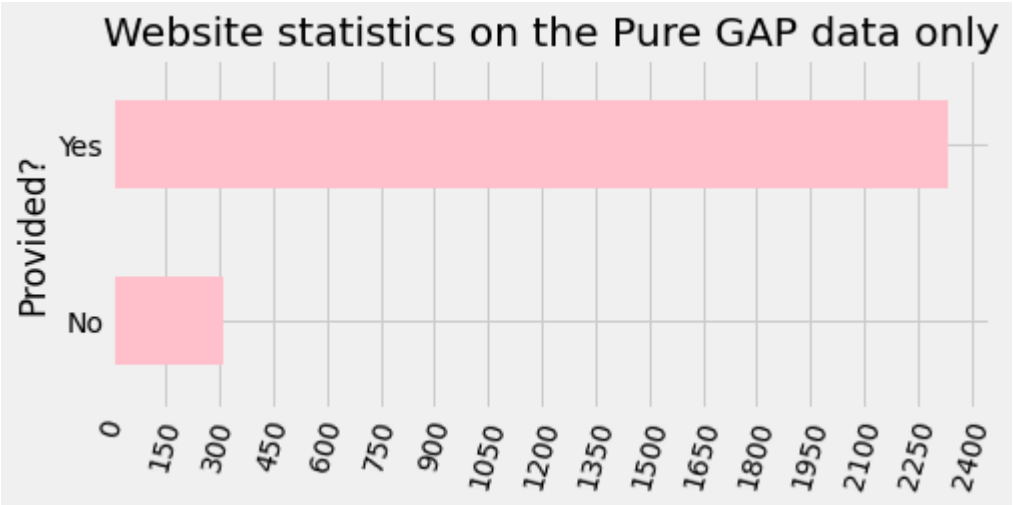
Below I created another representation of the same data, this time in a pie chart. Here we do not see the exact count but percentage ratios are displayed in each slice of the pie.

In [7]:
```
colors = ['yellowgreen', 'lightskyblue']
explode = (0.1, 0.1)
web_chart.plot(kind="pie", colors=colors, shadow=True,
               title='URL statistics on the whole data',
               autopct='%1.0f%%', explode=explode)
plt.show()
```



Below we have the same type of visualisation but for the two subsets of our data. First is the website statistics for GAP software citations and then for GAP Package citations only.

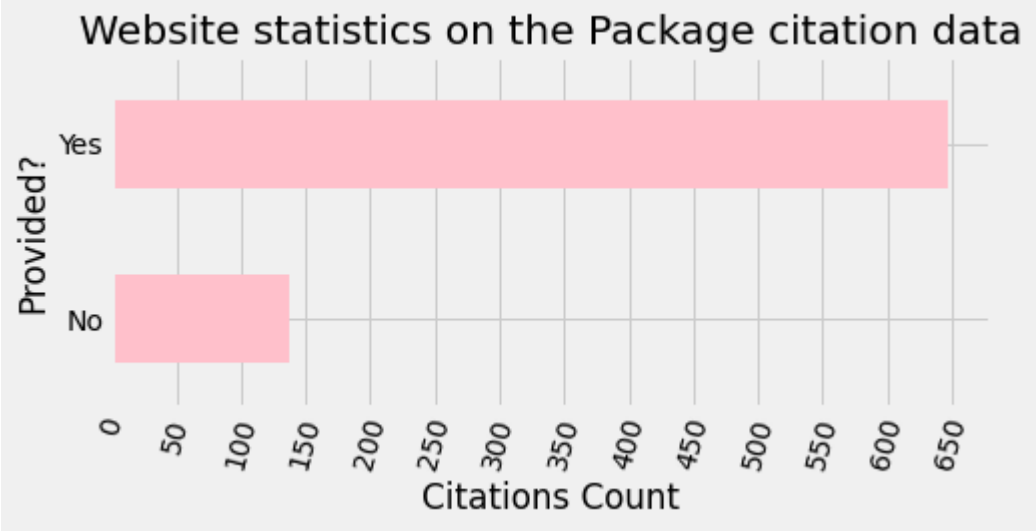
In [8]:
```
web_gap = gap_df['Website'].value_counts()
web_gap = web_gap.sort_values(ascending=True)
web_gap.plot(kind="barh", figsize=(7, 3),
             title='Website statistics on the Pure GAP data only',
               xlabel='Provided?', ylabel='Count', color='pink')
plt.xticks(range(0, 2500, 150))
plt.xticks(rotation=75)
plt.show()
```



GAP Package is based on GAP and sometimes similar, but in reality it is a separate piece of

software and as we will see in the raw data Package citations are slightly different. For isntace most of them provide a website reference which is different from the official https://www.gap-system.org/ (https://www.gap-system.org/) . Another difference is observed in the versions - most package citations either do not mention version or if they do it is version of the package used, not of the GAP release it is based on.

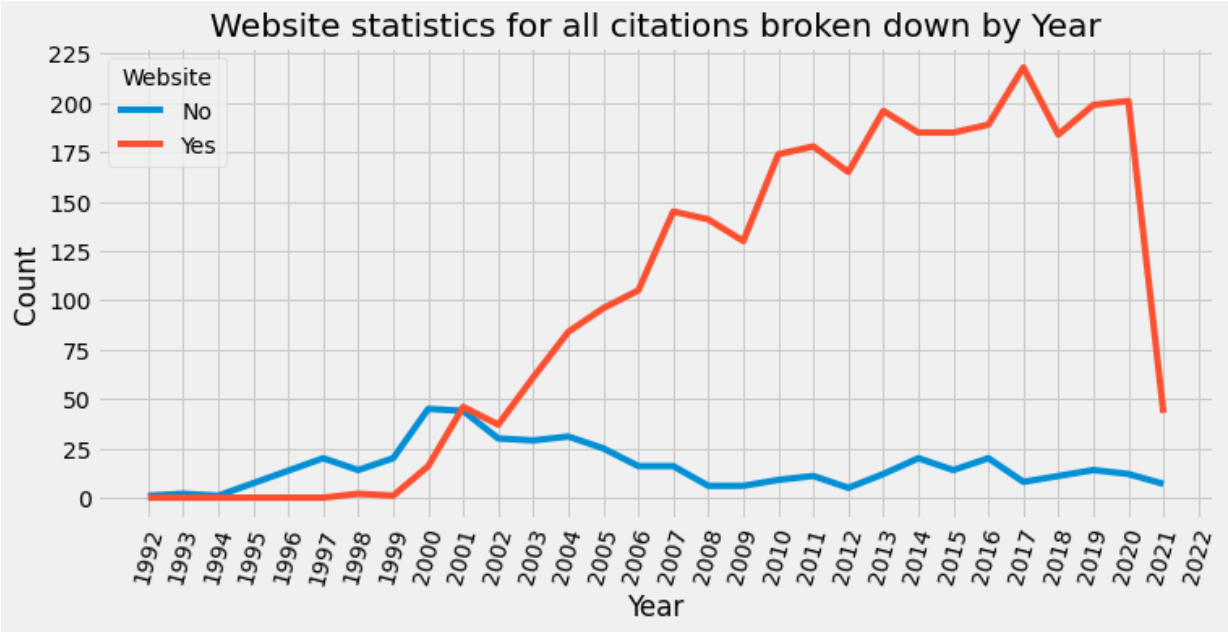
In [9]:
```
web_pac = pac_df['Website'].value_counts()
web_pac = web_pac.sort_values(ascending=True)
web_pac.plot(kind="barh", figsize=(7, 3),
             title='Website statistics on the Package citation data',
            color='pink')
plt.xticks(range(0, 700, 50))
plt.xticks(rotation=75)
plt.xlabel('Citations Count')
plt.ylabel('Provided?')
plt.show()
```



Website statistics grouped by Year, for the full dataset - this allows us to compare the number of publications who did and did not provide Website in their citations, for each year.

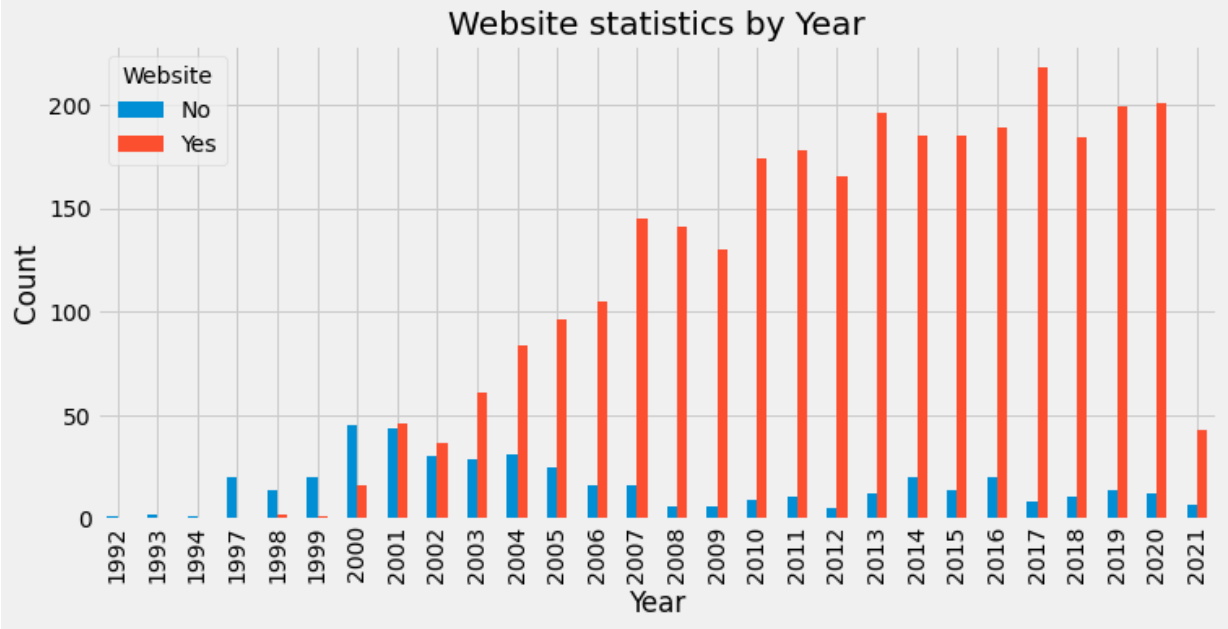
From the beginning of data gathering in 1992, to 2001 the number of citations lacking website reference was prevailing. After 2001 we are observing two drastic changes in the data, first the number of citations significantly increases and second most of the new citations provide a website, while the number of those lacking website decreases and remains low.

In [10]:
```
site_year = merged_df.groupby(['Website', 'Year'])['Year'].count().unstack('Websi
site_year_df = pd.DataFrame(data=site_year)
site_year_df.plot(kind='line', ylabel='Count', figsize=(11, 5),
                  title='Website statistics for all citations broken down by Year
                 )

plt.xticks(np.arange(1992, 2023, 1))
plt.xticks(rotation=75)
plt.yticks(np.arange(0, 250, 25))
plt.show()
```



Below we can see the same statistics represented in a bar chart.

In [11]:
```
site_year_df.plot(kind="bar", subplots=False, figsize=(11, 5),
                  title='Website statistics by Year',
                  xlabel='Year', ylabel='Count')
plt.show()
```



- We can see that in the beginning, up to 2000 the number of citations without a website reference was much higher. Then this trend reverses. On this barchart we can also observe the sharp increase of recorded citations altogether starting from 1997 and reaching a pique in 2017.

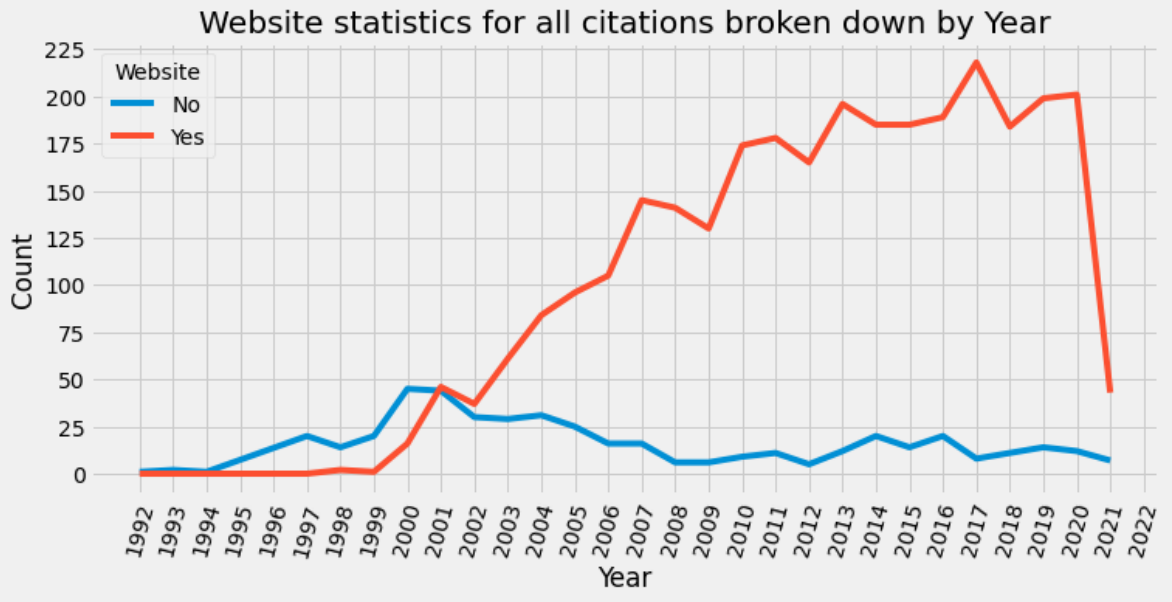
Website statistic for selected dataset

First we create a list containing the names of our 3 dataframes.
Then we employ that list as a dropdown in an interactive line chart. Depending on user selection, the specified data-frame is visualised.

In [12]:
```
df_list = ['Full Data', 'GAP Data', 'Package Data']
```

In [13]:
```
@interact
def jour_df_selector(Dataset = df_list):
    if Dataset == 'GAP Data':
        site_year = gap_df.groupby(['Website', 'Year'])['Year'].count().unstack('
        site_year_df = pd.DataFrame(data=site_year)
        site_year_df.plot(kind='line', ylabel='Count', figsize=(11, 5),
                          title='Website statistics for GAP software citations br
                         )

        plt.xticks(np.arange(1992, 2023, 1))
        plt.xticks(rotation=75)
        plt.yticks(np.arange(0, 250, 25))

        return plt.show()
    if Dataset == 'Full Data':
        site_year = merged_df.groupby(['Website', 'Year'])['Year'].count().unstac
        site_year_df = pd.DataFrame(data=site_year)
        site_year_df.plot(kind='line', ylabel='Count', figsize=(11, 5),
                          title='Website statistics for all citations broken dowr
                         )

        plt.xticks(np.arange(1992, 2023, 1))
        plt.xticks(rotation=75)
        plt.yticks(np.arange(0, 250, 25))

        return plt.show()
    if Dataset == 'Package Data':
        site_year = pac_df.groupby(['Website', 'Year'])['Year'].count().unstack('
        site_year_df = pd.DataFrame(data=site_year)
        site_year_df.plot(kind='line', ylabel='Count', figsize=(11, 5),
                          title='Website statistics for GAP Package citations brc
                         )

        plt.xticks(np.arange(1992, 2023, 1))
        plt.xticks(rotation=75)
        plt.yticks(np.arange(0, 250, 25))

        return plt.show()
```

Dataset | Full Data

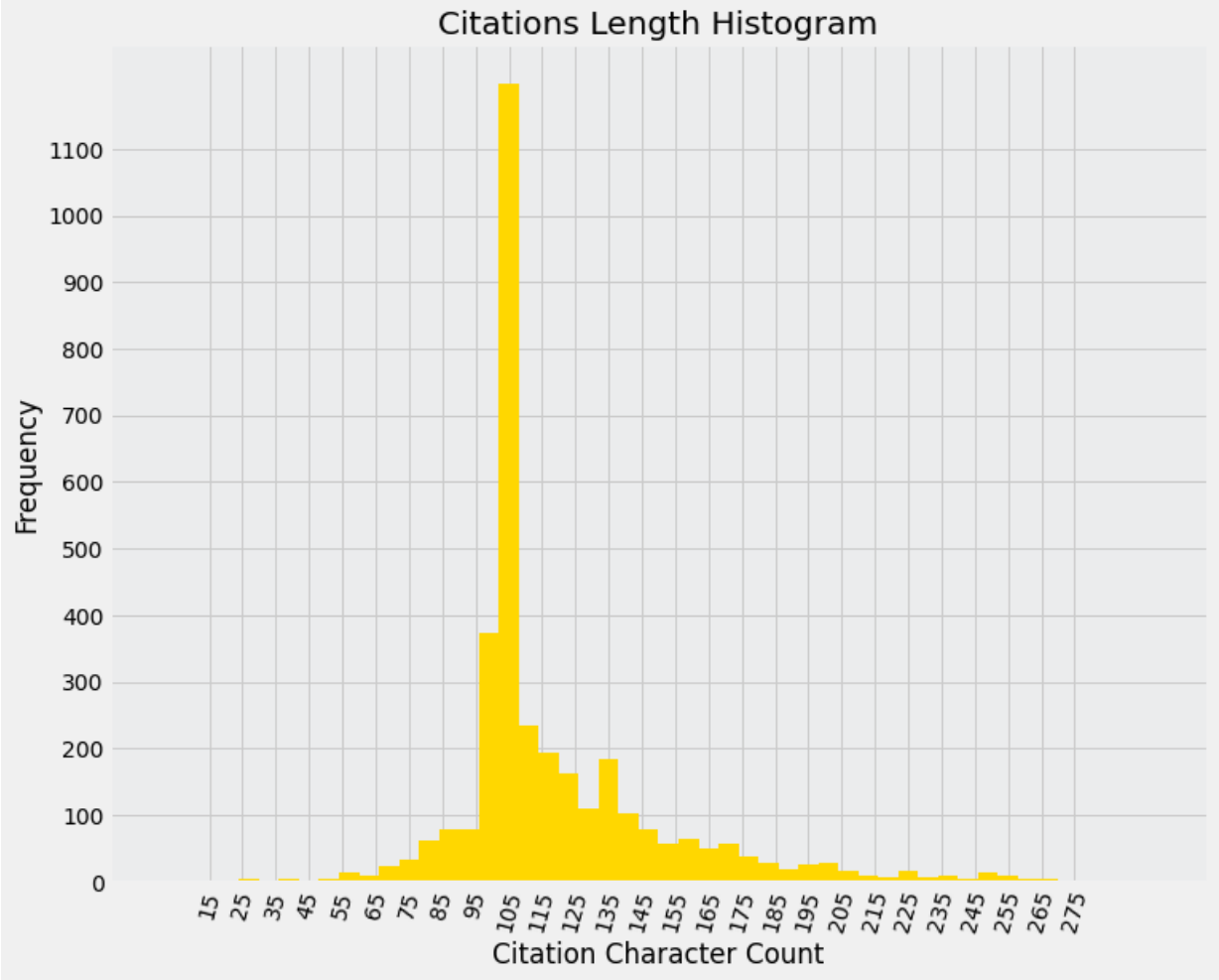


Using this tool we can focus on the Package citations only, where the good practice of mentioning a website only really increased in 2005, which is 4 years later than when it happened with the pure GAP citations.

Length

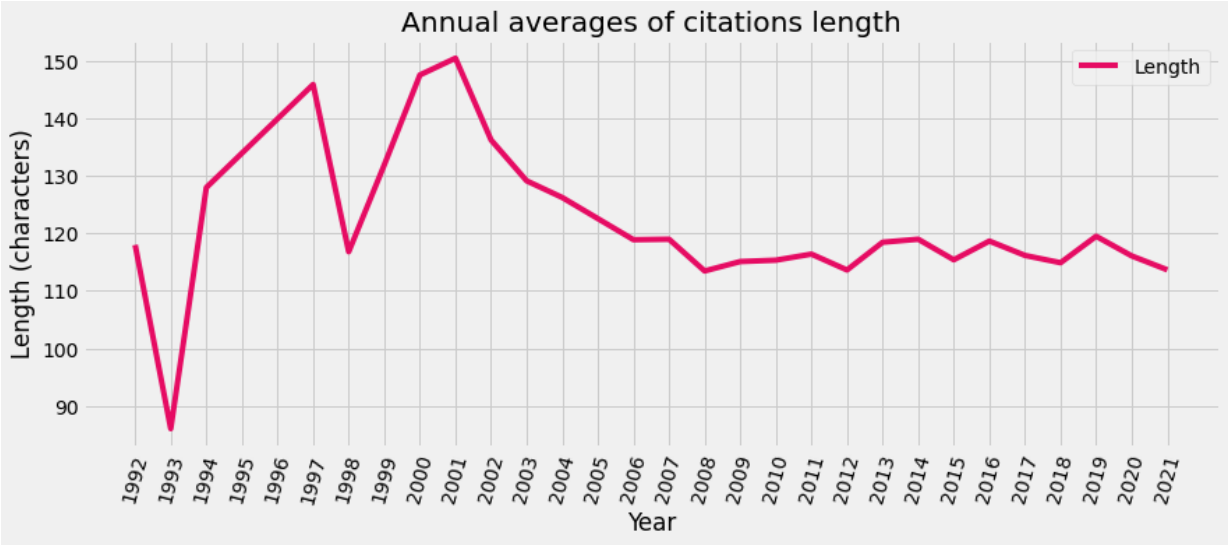
Here we visualise the `Length` column we created, using a Histogram.
Looking at it, we can see that most citations are around 105 characters long - more than 1100 records.

In [14]:
```
cit_len = merged_df['Length'].plot(kind='hist', bins=50,
                           figsize=(11,9), stacked=True,
                           range=(0, 300),
                           color='gold',
                           title='Citations Length Histogram',
                           xlabel="length")
cit_len.set_facecolor('#ebeced')
plt.xticks(np.arange(15, 285, 10))
plt.xticks(rotation=75)
plt.yticks(np.arange(0, 1200, 100))
plt.xlabel('Citation Character Count')
plt.show()
```



We can also draw a line chart displaying the average length per year. For this purpose we group our data by year and take the mean of the length column, then transform the result to a pandas dataframe and plot it.

In [15]:
```
length_average = merged_df.groupby('Year')['Length'].mean()
length_average = pd.DataFrame(data=length_average)
length_average.plot(kind='line', figsize=(13, 5),
         title='Annual averages of citations length',
         xlabel='Year', ylabel='Length (characters)', color='#e60b63')
plt.xticks(range(1992, 2022, 1))
plt.xticks(rotation=75)
plt.show()
```

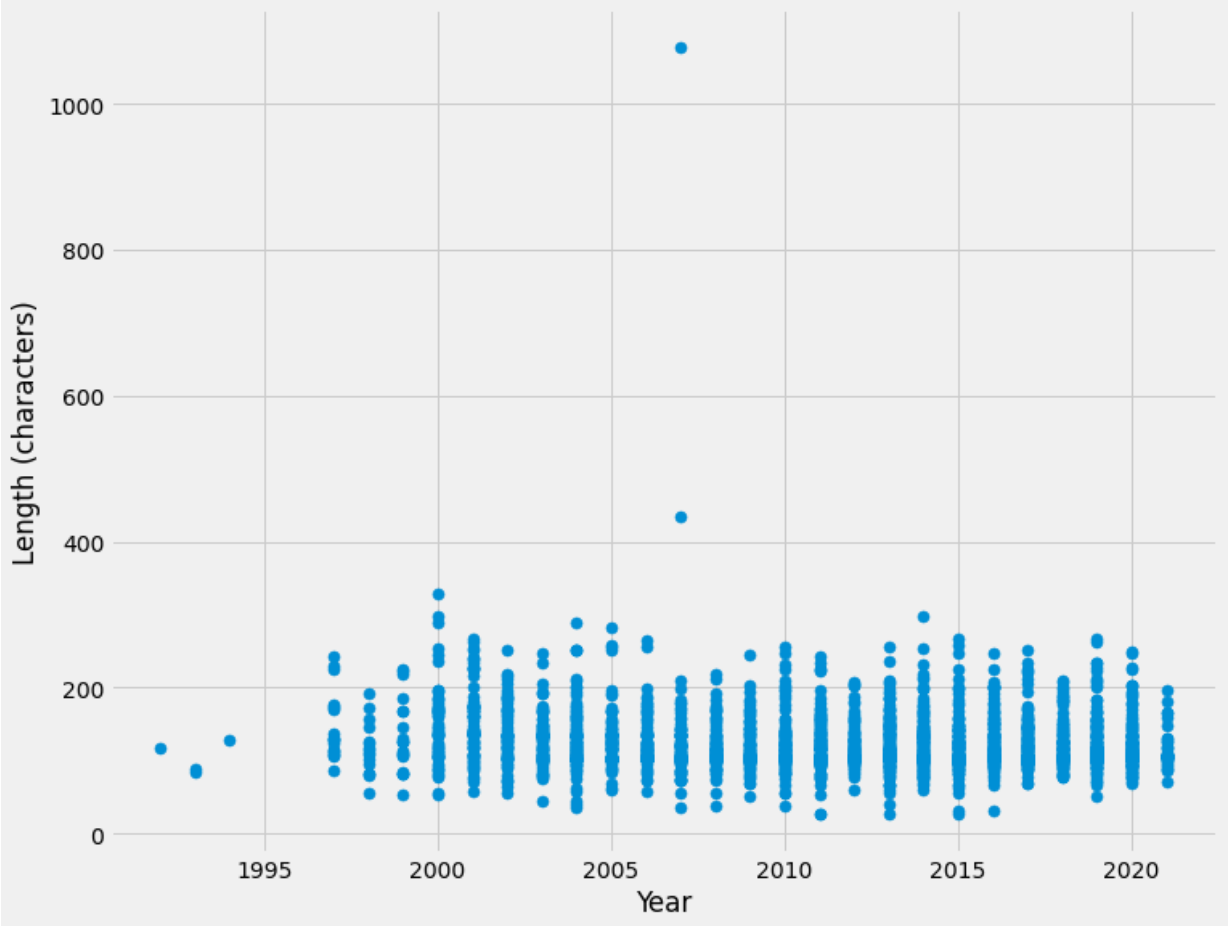


- In the first years, up to 2000 we only have very few records per year, therefore the average is volatile and strongly fluctuating. After that it decreases and starts stabilizing between 2000 and 2005. From 2005 onwards until today the average GAP Citations length remains stable in the range between 110 and 120 characters. This stable trend in the last 16 years proves that a widespread, potentially global standard for designing GAP citations has formed.

Scatter plot displaying length of citations by year.

Here we simply plot length by year taken directly from the main dataset. We are using scatterplot because there are too many data points to be displayed otherwise.

In [16]:
```
merged_df.plot(kind='scatter', y = 'Length', x = 'Year', figsize=(11,9),
               s=50)
plt.ylabel('Length (characters)')
plt.show()
```



In [17]:
```
merged_df[merged_df['Length']>400] # we isolate the outliers
```

Out[17]:

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Website	Length
271	2299794	Assmann, B. and Eick, B.	Math. Comp.	2007	article	20F16 (20-04)	The GAP Group. GAP– Groups, Algorithms and Prog...	Unknown	Yes	43
752	2351139	Cameron, P. and Cilleruelo, J. and Serra, O.	Rev. Mat. Iberoam.	2007	article	05D10	[1] Datskovsky, B. A.: On the number of monoch...	4.3	Yes	107

- We have two outliers, one of them is an error and the other one is citing GAP but mentioning both St Andrews and Braunschweig universities, along with their full contact details, which is making it too long. We will remove these two entries to make the scatter plot more clear.

In [18]:
```
print(merged_df.iloc[758]['Citation'])
print(merged_df.iloc[272]['Citation'])
```

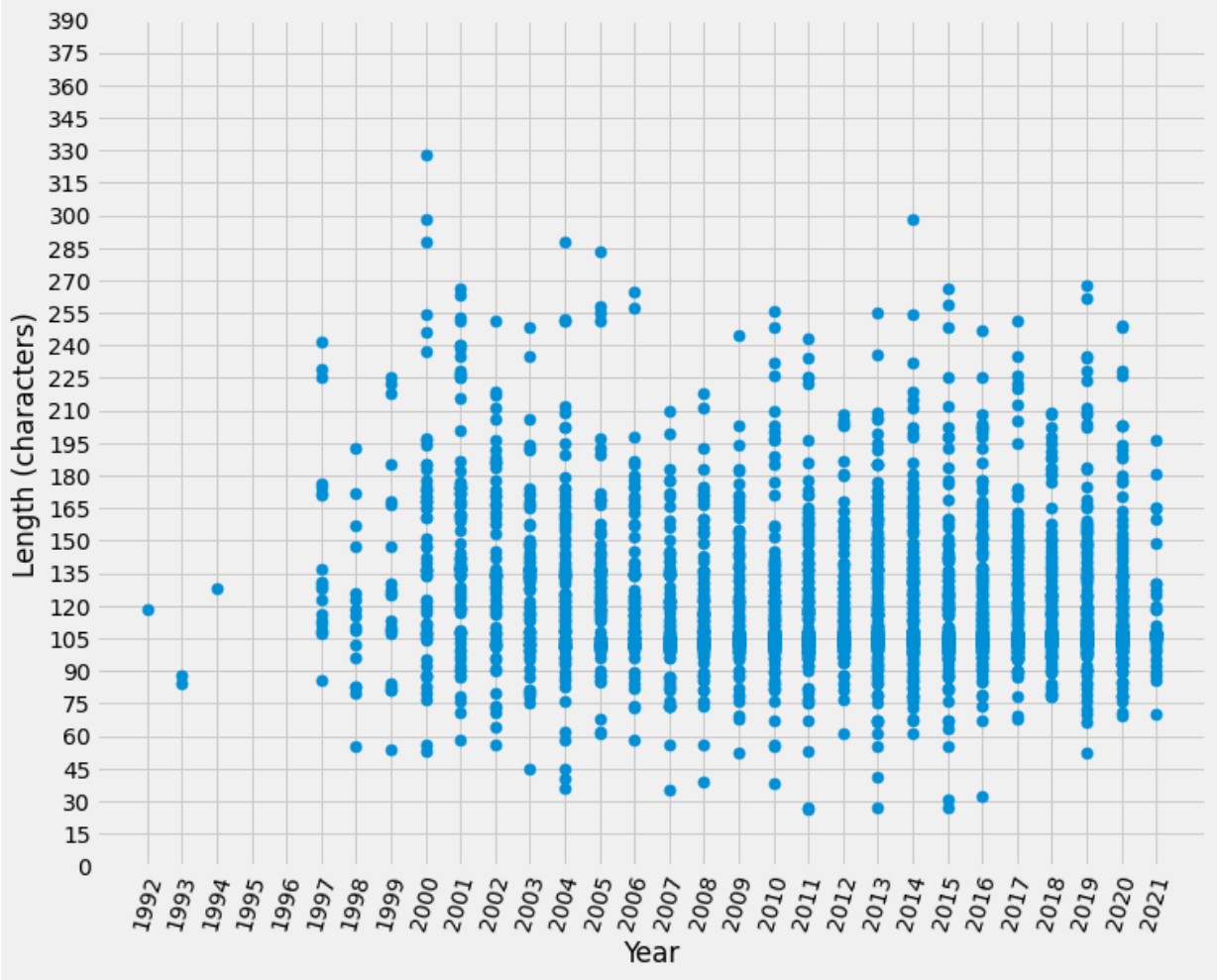
```
The GAP group, GAP-groups, algorithms, and programming, Version 4.4.12, 2008, h
ttp://www.gap-system.org. (http://www.gap-system.org.)
B. Assmann, Guarana—Applications of Lie methods in computational group theory,
 2006, A GAP 4 package, see [23].
```

In [19]:
```
outliers = merged_df[merged_df['Length']>400].index
len_scatter = merged_df.drop(outliers)
len_scatter.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3428 entries, 0 to 3429
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   MR                3428 non-null   int64
 1   Author            3428 non-null   object
 2   Journal           3428 non-null   object
 3   Year              3428 non-null   int64
 4   Publication Type  3428 non-null   object
 5   MSC               3428 non-null   object
 6   Citation          3428 non-null   object
 7   Version           3428 non-null   object
 8   Website           3428 non-null   object
 9   Length            3428 non-null   int64
 10  Accuracy Score    3428 non-null   int64
dtypes: int64(4), object(7)
memory usage: 227.6+ KB
```

Below is the updated plot. Without the outliers, the result is much more "zoomed in" and we can focus and study the data in more detail.

In [20]:
```
len_scatter.plot(kind='scatter', y = 'Length', x = 'Year', figsize=(11,9),
                 s=50, xticks = range(1992, 2022, 1),
                 yticks = range(0, 400, 15))
plt.xticks(rotation=75)
plt.ylabel('Length (characters)')
plt.show()
```



We observe that the length of citations widely varies during each year. The highest concentration of datapoints is around 105 (citation length) but they range from 30 to 330. Let us investigate the data and find out some more information about the shortest and longest citations.

First we will define a function initially used in the data preparation notebook, so we can display and analyse separate entries from our dataset.

In [21]:
```
# Python lectures by Dr Konovalov
# https://studres.cs.st-andrews.ac.uk/CS2006/Lectures/Python/L08-dataset.pdf
# slightly modified so it can return all citations with the specified MRN
# on the other hand the result is a dataframe and if we want to read the full cit
def get_c(mrno):
    r = merged_df[merged_df['MR'] == mrno]
    return r
```

Then we create two subsets, shorter than 30 characters and longer than 300 characters.

In [22]:
```
sma = merged_df[merged_df['Length'] < 30]
big = merged_df[merged_df['Length'] > 300]
print(len(sma))
print(len(big))
```

```
4
3
```

In [23]:
```
sma
```

Out[23]:

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Webs
3393	3063476	García Pillado, C. and González, S. and Martín...	J. Algebra Appl.	2013	article	94B60 (20C05 94B25)	http://www.gap-system.org/.	Unknown	Y
3394	2904286	Garsia-Pil'yado, K. and Gonsales, S. and Marko...	Fundam. Prikl. Mat.	2011	article	94B60 (20C05)	http://www.gap-system.org/.	Unknown	Y
3397	3451663	Garsia-Pil'yado, K. and Gonsales, S. and	Fundam. Prikl. Mat.	2015	article	94A60 (20D99)	http://www.gap-system.org/.	Unknown	Y

Afer a quick glance we can easily distinguish that the shortest citations in our records contain simply just the official GAP website and no other information.
Another important feature is that 3 out of the 4 are articles by the same group of authors. It seems at the time of citing, these authors were under the impression that providing only a website is good

enough.
It is a fun fact to note that all the shortest citations even got one accuracy point, by our Accuracy Score calculator, because after all they do mention the GAP website, even though it is the only bit of information they have.

In [24]:
```
big
```

Out[24]:

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Website	Leng
271	2299794	Assmann, B. and Eick, B.	Math. Comp.	2007	article	20F16 (20-04)	The GAP Group. GAP–Groups, Algorithms and Prog...	Unknown	Yes	4
752	2351139	Cameron, P. and Cilleruelo, J. and Serra, O.	Rev. Mat. Iberoam.	2007	article	05D10	[1] Datskovsky, B. A.: On the number of monoch...	4.3	Yes	10
3426	1801202	Shaw, R.	Des. Codes Cryptogr.	2000	incollection	51E14	L.H. Soicher, GRAPE: a system for computing wi...	Package	Yes	3

The middle row with index 752 which is over 1000 characters long is just an error along the pipeline - accidentaly all citations of the publication got into the GAP citation text, so we can rule that out.

In [25]:
```
print(merged_df.loc[758]['Citation'])
```

```
The GAP group, GAP-groups, algorithms, and programming, Version 4.4.12, 2008, http://www.gap-system.org. (http://www.gap-system.org.)
```

The first citation with index 272 is so long because it mentions the two main bases of GAP, namely University of St Andrews and Universität Braunschweig along with full contact details for both.

In [26]:
```
print(merged_df.loc[272]['Citation'])
```

```
B. Assmann, Guarana–Applications of Lie methods in computational group theory, 2006, A GAP 4 package, see [23].
```

The last citation is for the GRAPE GAP package, but it also refers to a journal including all its details such as name, volume, page etc

In [27]:
```
print(merged_df.loc[3426]['Citation'])
```

```
L.H. Soicher, GRAPE: a system for computing with graphs and groups, in: L. Fink
elstein and W.M. Kantor, eds., Groups and Computation, DIMACS Series in Discret
e Mathematics and Theoretical Computer Science Vol. 11, AMS, (1993) pp. 287–29
1. GRAPE is available from http://www-gap.dcs.st-and.ac.uk/gap/Share/grape.htm
l. (http://www-gap.dcs.st-and.ac.uk/gap/Share/grape.html.)
MR1235810
```

Now we can extend our review and look up citations shorter than 75 characters. Scrolling along the resulting filtered dataset we see that these are mostly citations which mention the software name, along with a website or a version but not both. Others are GAP Package citations mentioning the package name along with its release year and sometimes its specific version. Looking at the `Author` and `Journal` columns we cannot see any commonly repeating names, therefore we cannot conclude that too long or too short because of the author(s) who created them or the journal they come from.

In [28]:
```
sma1 = merged_df[merged_df['Length'] < 70]
sma1
```

Out[28]:

	MR	Author	Journal	Year	Publication Type	MSC	
21	2726552	Abdollahi, A. and Khosravi, H.	J. Algebra Appl.	2010	article	20F45 (20F18)	W. Nickel, NQ, 1998, A r
28	3297734	Abdollahi, A.	J. Group Theory	2015	article	20F45	W. Nickel, NQ, 1998, A r
250	2176489	Ash, A. and Pollack, D. and Sinnott, W.	J. Number Theory	2005	article	11F80	GAP4, Version: 4.3
273	2358616	Assmann, B. and Linton, S.	J. Algebra	2007	article	20F40 (20F05)	W. Nickel, NQ, 1998, A r
347	2864860	Balagović, M. and	J. Pure Appl.	2012	article	16S99 (16D90 16G10	J. Miche

In [29]:
```
print(merged_df.loc[880]['Citation'])
print(merged_df.loc[2240]['Citation'])
```

```
GAP. "GAP–Groups, Algorithms and Programming, Version 4.4.10, 2007." The GAP Gr
oup. http://www.gap-system, (http://www.gap-system,) org.
The GAP Group, GAP–Groups, Algorithms, and Programming, Version 4.4.12, http://
www.gap-system.org, (http://www.gap-system.org,) 2008.
```

Version

We will use StrictVersion on `ver` series below to sort the versions semantically, because we need

them to be displayed in chronological order.

In [30]:
```
from distutils.version import StrictVersion
```

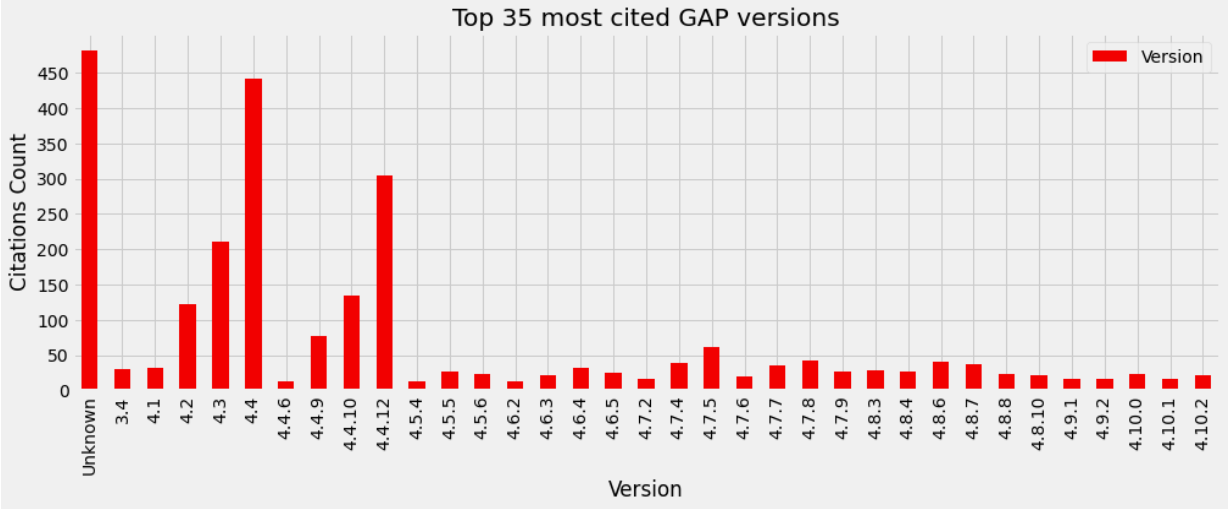
In [31]:
```
ver = gap_df['Version'].value_counts()
ver = ver.head(35)
ver = ver.to_frame()
ver = ver.drop(['Unknown']) # we remove Unknown row.
ver = ver.reindex(index=pd.Index(sorted(ver.index, key=StrictVersion)))
ver_return = pd.DataFrame([[482]], columns=['Version'], index=['Unknown']) # we c
ver = pd.concat([ver_return, ver])
ver
```

Out[31]:

	Version
Unknown	482
3.4	30
4.1	32
4.2	122
4.3	211
4.4	441
4.4.6	13
4.4.9	77
4.4.10	135
4.4.12	304
4.5.4	13
4.5.5	27
4.5.6	24
4.6.2	13
4.6.3	22
4.6.4	32
4.6.5	25
4.7.2	17
4.7.4	39
4.7.5	62
4.7.6	19
4.7.7	36
4.7.8	42
4.7.9	27
4.8.3	28
4.8.4	26
4.8.6	40
4.8.7	37
4.8.8	23

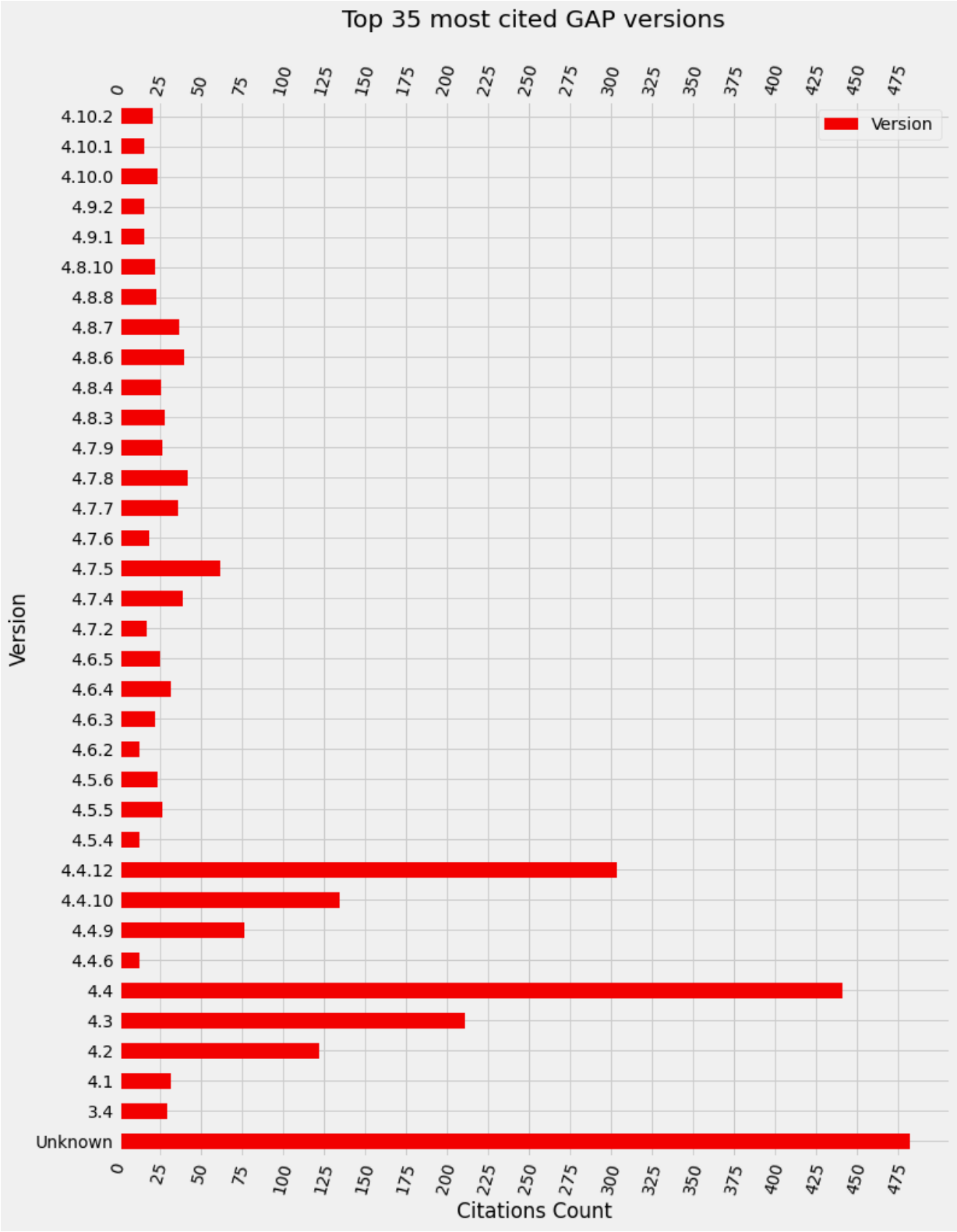
	Version
4.8.10	22
4.9.1	16
4.9.2	16
4.10.0	24
4.10.1	16
4.10.2	21

In [32]:
```
ver = gap_df['Version'].value_counts()
ver = ver.head(35)
ver = ver.to_frame()
ver = ver.drop(['Unknown'])
ver = ver.reindex(index=pd.Index(sorted(ver.index, key=StrictVersion)))
ver_return = pd.DataFrame([[482]], columns=['Version'], index=['Unknown'])
ver = pd.concat([ver_return, ver])
#ver = ver.dropna()
ver.plot(kind="bar", figsize=(15, 5), title='Top 35 most cited GAP versions',
         xlabel='Version',
         ylabel='Citations Count', color='#f20000')
plt.yticks(np.arange(0, 500, 50))
plt.show()
```



Because we have many versions in the x axis, we might also visualise that in a horizontal bar chart for better readability.

In [33]:
```
ver.plot(kind="barh", figsize=(11, 15), title=('Top 35 most cited GAP versions' +
         ylabel='Version',
         xlabel='Citations Count', color='#f20000')
plt.ylabel('Version')
plt.xlabel('Citations Count')
plt.tick_params(axis="x", bottom=True, top=True, labelbottom=True, labeltop=True)
plt.xticks(np.arange(0, 500, 25))
plt.xticks(rotation=75)
plt.show()
```



- In this bar chart we can see that the most popular GAP versions cited are subversions of GAP 4, between 4.2 and 4.4

Interactive chart displaying number of records for each version for selected Year.

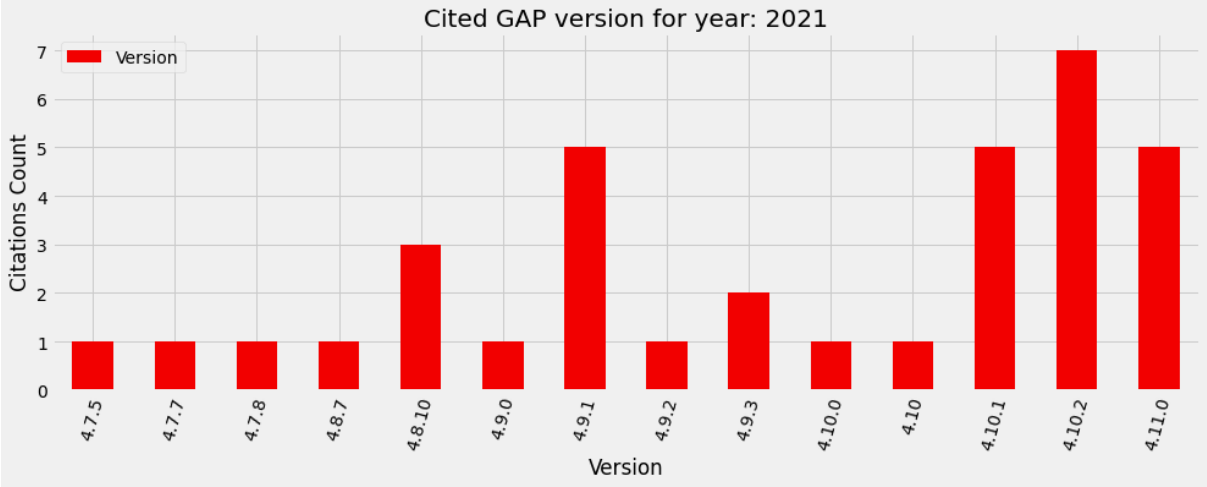
Creating and sorting a list of years fo our drop-down menu.

In [34]:
```
years_list = gap_df['Year'].unique()
years_list = np.sort(years_list)[::-1] #sort the NumPy array in descending order
#years_list
```

In this visualisation we will ommit the data for 'Unknown' version, because if it is present, due to its large number, bars for other versions are rendered to only several milimeters, which makes readability unacceptably poor. Nevertheless you can find a breakdown of `Unknown` version numbers by year in the next interactive visualisation, by selecing "Unknown" from the version dropdown.

In [35]:
```
@interact
def ver_by_year(Year=years_list):
    to_plot = gap_df[gap_df['Year']==Year]
    ver = to_plot['Version'].value_counts()
    ver = ver.to_frame()
    ver = ver.drop(['Unknown']) # thus we remove the Unknown version numbers
    ver = ver.reindex(index=pd.Index(sorted(ver.index, key=StrictVersion))) # sem
    ver.plot(kind="bar", figsize=(15, 5),
             title='Cited GAP version for year: ' + str(Year),
             xlabel='Version',
             ylabel='Citations Count', color='#f20000')
    plt.xticks(rotation=75)
    return plt.show()
```

Year 2021



The trend that we observe here is that in the yearly years very few GAP versions were cited, simply because not many existed yet. Later on after 2000 the number of different cited versions starts growing noticeably, because authors were citing the GAP version they used, which was not necessarily the latest version. As we have seen in the previous chart the most popular GAP versions are between 4.2 and 4.4
Another peculiarity we can find by looking at the years between 2010 and 2020 where we will find a wide variety of versions cited. This tells us that in the GAP community there was no unison or collective notion regarding what is the best version or what is the latest available version. On the other hand, different versions might be better suited for different purposes, we can never know for certain unless we study cases one by one. After we have seen data for all these years, we switch to 2021, the last available year in our dropdown list and suddenly we see a much clearer picture - all cited versions are between 4.1 and 4.9. This might be happening for several reasons

- older versions being discontinued in 2021
- general announcement on the official website and/or GAP community pages
- we are still only half-way into 2021 and we do not have the whole data yet - after the year ends, older version citations might appear in the data.

This interactive tool can be useful, not just for GAP but for any other software. It can be employed

to analyse numbers of cited versions for different years and draw conclusions.

Annual count of citations for selected Version

First a subset of the data is created to faciliate the research question.

In [36]:
```
vvv = gap_df.groupby(['Year', 'Version'])['Version'].count().unstack('Year').fill
vdf = pd.DataFrame(data=vvv)
vdf = vdf.sort_values(by='Version')
vdf
```

Out[36]:

Year	1992	1993	1994	1997	1998	1999	2000	2001	2002	2003	...	2012	2013	201
Version														
3.0	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
3.1	0.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
3.2	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
3.3	0.00	0.00	0.00	0.00	2.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
3.4	0.00	0.00	0.00	5.00	1.00	2.00	7.00	5.00	3.00	5.00	...	0.00	0.00	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4.9.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
4.9.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
4.9.2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
4.9.3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.0
Unknown	1.00	0.00	0.00	12.00	13.00	15.00	43.00	43.00	25.00	25.00	...	11.00	31.00	18.0

66 rows × 28 columns

Creating a list of versions for our drop-down menu. There are two words and when we try to use StrictVersion they cause an error. Therefore, to order them semantically, we first convert the data to a list, then remove the two words, then sort using the semantic version sorting algorithm. Finally we add back the two words to the list.

In [37]:
```
ver_list = gap_df['Version'].unique()
ver_list = ver_list.tolist()
ver_list.remove('Unknown')


# List of removed strings
words = ['Unknown']

# Sorting the ver_list
ver_list.sort(key=StrictVersion)

# Adding the removed strings back at the end of ver_list
ver_list = ver_list + words
```

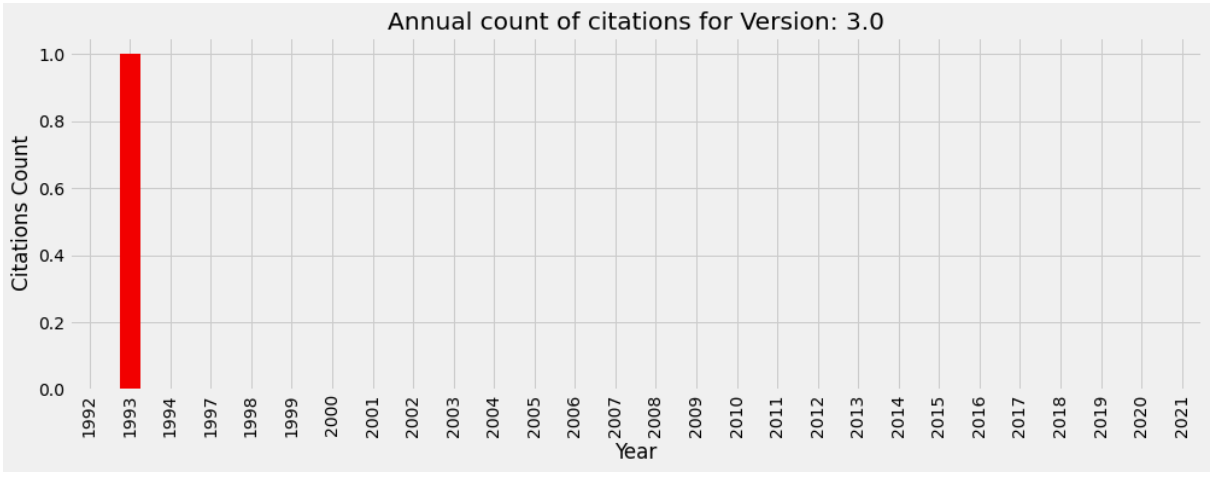
In [38]:
```
print(ver_list)
```

```
['3.0', '3.1', '3.2', '3.3', '3.4', '3.4.3', '3.4.4', '4.1', '4.2', '4.3', '4.
4', '4.4.2', '4.4.3', '4.4.4', '4.4.5', '4.4.6', '4.4.7', '4.4.9', '4.4.10',
'4.4.11', '4.4.12', '4.5', '4.5.3', '4.5.4', '4.5.5', '4.5.6', '4.5.7', '4.6',
'4.6.1', '4.6.2', '4.6.3', '4.6.4', '4.6.5', '4.6.9', '4.6.12', '4.7', '4.7.2',
'4.7.4', '4.7.5', '4.7.6', '4.7.7', '4.7.8', '4.7.9', '4.8', '4.8.1', '4.8.2',
'4.8.3', '4.8.4', '4.8.5', '4.8.6', '4.8.7', '4.8.8', '4.8.9', '4.8.10', '4.9.
0', '4.9', '4.9.1', '4.9.2', '4.9.3', '4.10.0', '4.10', '4.10.1', '4.10.2', '4.
11.0', '4.11', 'Unknown']
```

Then we employ the subset in an interactive visualisation, allowing users to specify which version's data they wish to display.

In [39]:
```
@interact
def ver_by_year(Version=ver_list):
    vdf.loc[Version].plot(kind="bar", figsize=(15, 5),
             title='Annual count of citations for Version: ' + Version,
             xlabel='Year',
             ylabel='Citations Count', color='#f20000')
    return plt.show()
```

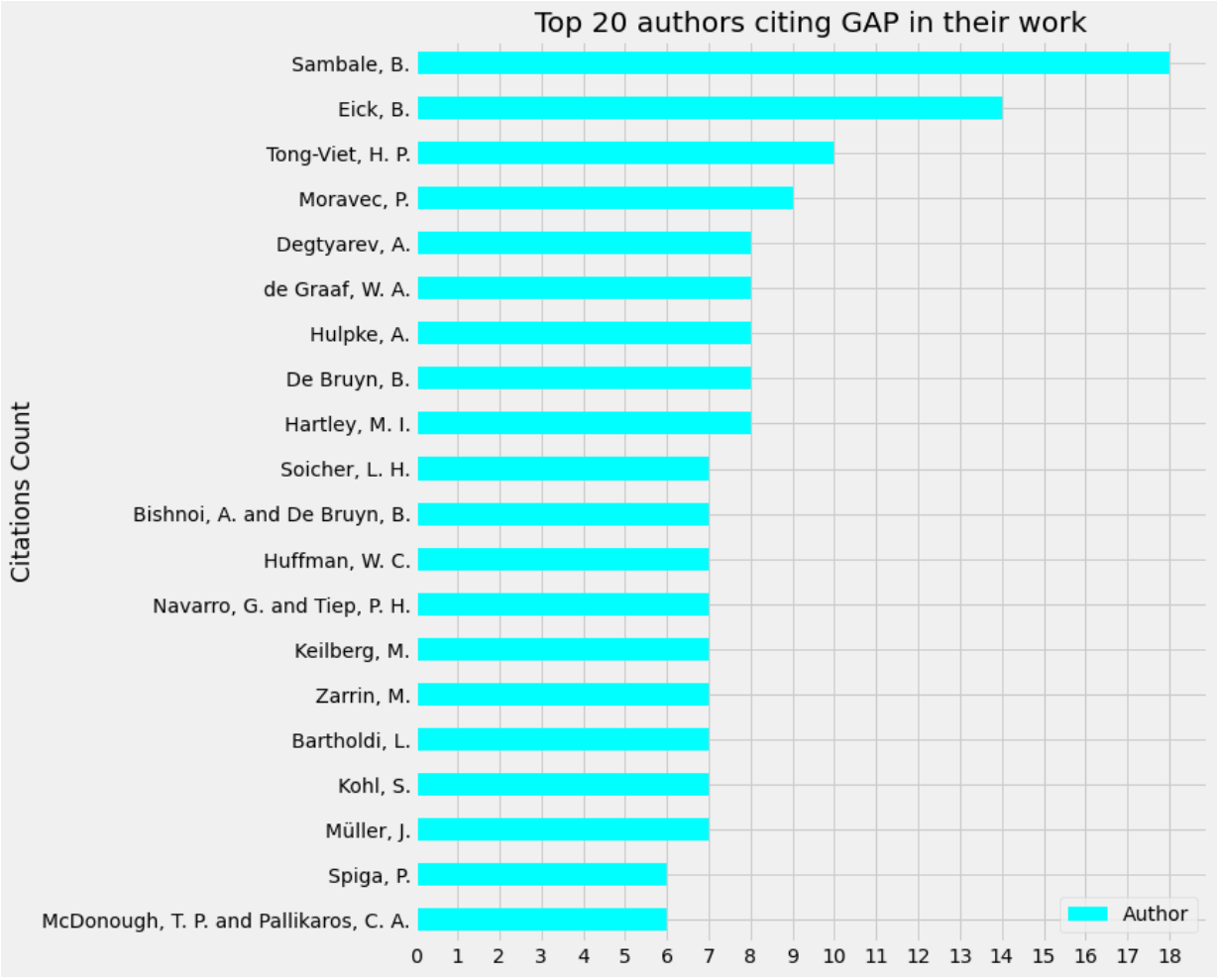
Version 3.0



Authors

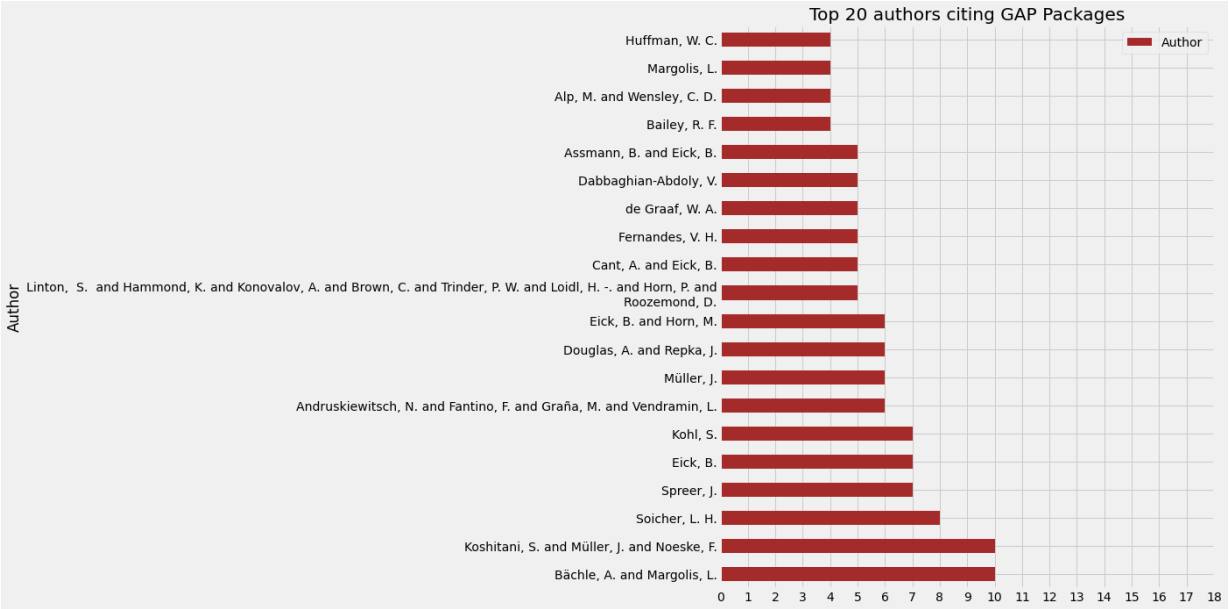
- Top 20 authors citing GAP and its packages in their work.

In [40]:
```
authors = gap_df['Author'].value_counts()
top = authors.head(20)
top = top.sort_values(ascending=True)
top = top.to_frame()
chart = top.plot(kind="barh",
                 figsize=(9,11), title='Top 20 authors citing GAP in their work',
                 xlabel='Author',
                 ylabel='Citations Count', color='cyan',
                  xticks=range(0, 19, 1))
plt.ylabel('Citations Count')
plt.show()
# how do we treat joint papers ?
```



- Top 20 authors citing GAP Packages

In [41]:
```
author = pac_df['Author'].value_counts()
top20 = author.head(20)
top20 = top20.sort_values(ascending=False)
top20 = top20.to_frame()


top20.plot(kind="barh",
                  figsize=(9,11), title='Top 20 authors citing GAP Packages',
                  xlabel='Author',
                  ylabel='Count', color='brown',
                  xticks=range(0, 19, 1))
plt.show()
```

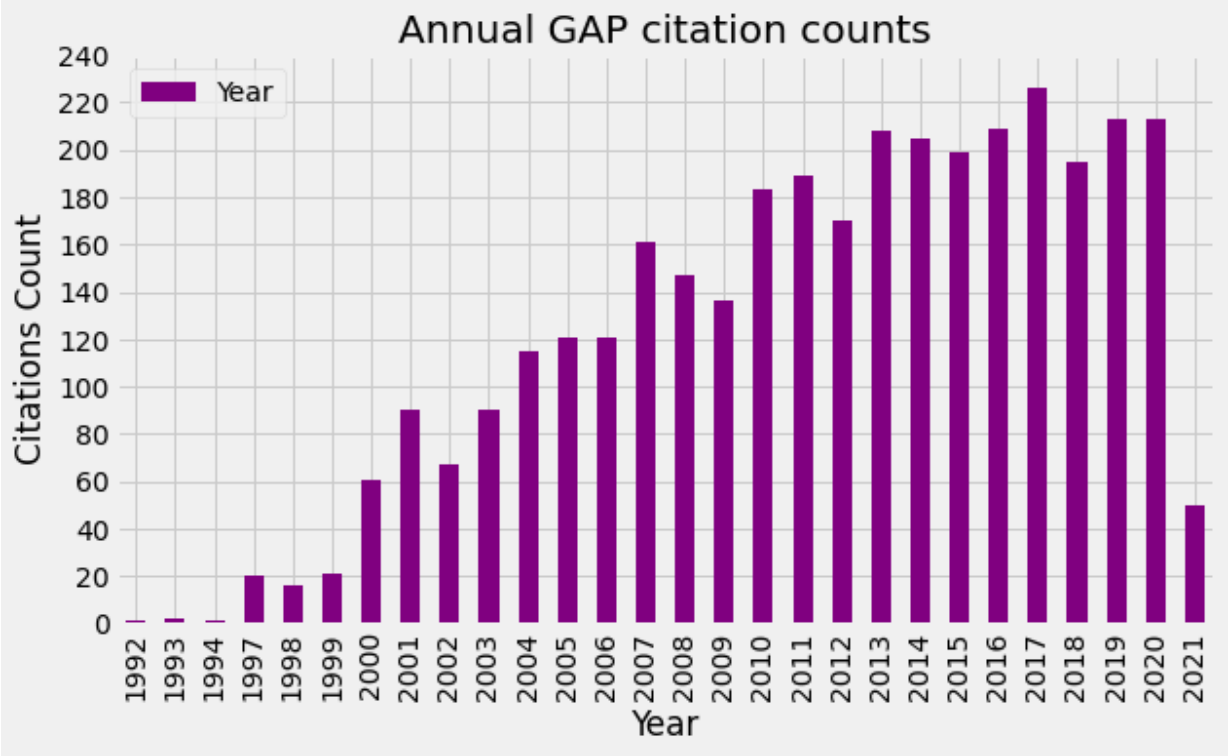


Year

Barchart dislaying number of GAP citations per year, with the bars sorted chronologically.

Here we simply use `value_counts()` to get number of entries per year, then plot it as a bar chart which allows us easy comparison between data points.

In [42]:
```
year_chrono = merged_df['Year'].value_counts()
chrono_df = year_chrono.to_frame()
chrono = chrono_df.sort_index()
chrono.plot(kind="bar", figsize=(9, 5), title='Annual GAP citation counts', xlabe
                ylabel='Citations Count', color='purple')
plt.yticks(np.arange(0,250, 20))
plt.show()
```



After looking at the chart we can see that the amount of GAP citations was relatively small upto 1999 and from 2000 onwards its started growing steady to reach around 200 per year in 2013. From there on it kept that level with around 250 in 2017 and 190 in 2018. The data for 2021 is still incomplete. We can conclude that GAP's popularity started to grow in the year of 2000.

Top 20 journals citing GAP, for selected dataset.

- Users can choose to see top 20 journals for the full data, for the GAP citations only or just for GAP Package citations.

In [43]:
```
@interact
def jour_df_selector(Dataset = df_list):
    if Dataset == 'GAP Data':
        top_journals = gap_df['Journal'].value_counts()
        top_20 = pd.DataFrame(top_journals.head(20))
        top_20 = top_20.reset_index()
        top_20.columns=['Journal', 'Citations count']
        return top_20
    if Dataset == 'Full Data':
        top_journals = merged_df['Journal'].value_counts()
        top_20 = pd.DataFrame(top_journals.head(20))
        top_20 = top_20.reset_index()
        top_20.columns=['Journal', 'Citations count']
        return top_20
    if Dataset == 'Package Data':
        top_journals = pac_df['Journal'].value_counts()
        top_20 = pd.DataFrame(top_journals.head(20))
        top_20 = top_20.reset_index()
        top_20.columns=['Journal', 'Citations count']
        return top_20
```

Dataset: Full Data

	Journal	Citations count
0	J. Algebra	499
1	Comm. Algebra	200
2	J. Symbolic Comput.	147
3	J. Algebra Appl.	111
4	Discrete Math.	106
5	Internat. J. Algebra Comput.	85
6	J. Group Theory	81
7	Des. Codes Cryptogr.	70
8	J. Pure Appl. Algebra	70
9	LMS J. Comput. Math.	61
10	Math. Comp.	55
11	Israel J. Math.	49
12	Experiment. Math.	48
13	J. Combin. Des.	46
14	European J. Combin.	44
15	J. Combin. Theory Ser. A	39
16	Electron. J. Combin.	38
17	Trans. Amer. Math. Soc.	37
18	Arch. Math. (Basel)	36

	Journal	Citations count
19	Algebra Colloq.	35

"J. Algebra" is the leader in all 3 charts with highest number of GAP citations. The top 5 are the same for Full data and GAP data with just 4th and 5th being reversed.
The top 5 for the Package citations is entirely different however, because GAP packages are cited by different journals and books.

Top 20 Journals citing GAP for selected time period

- Using this tool we can filter the data we have at our disposal and discover tendencies in citing GAP software over the years. For instance if we use `Up to` operator with the year of 1998 we will notice firstly that very few records are available and secondly they arem ostly British and American, with one Japanese journal. Then if we start increasing the year gradually, keeping the same operator, we will observe the appearance of new journal names from various countries around the world, such as France, Turkey, Switzerland, Russia, Israel and some international too.

In [44]:
```
@interact
def jou_by_year(Year=years_list,
                Operator=['Up to', 'For this year', 'After']):
    column='Year'
    if Operator == 'For this year':
        to_plot = merged_df[merged_df[column] == Year]
        top_journals = to_plot['Journal'].value_counts()
        top_20 = pd.DataFrame(top_journals.head(20))
        top_20 = top_20.reset_index()
        top_20.columns=['Journal', 'Citations count']
        return top_20
    if Operator == 'Up to':
        to_plot = merged_df[merged_df[column] <= Year]
        top_journals = to_plot['Journal'].value_counts()
        top_20 = pd.DataFrame(top_journals.head(20))
        top_20 = top_20.reset_index()
        top_20.columns=['Journal', 'Citations count']
        return top_20
    if Operator == 'After':
        to_plot = merged_df[merged_df[column] > Year]
        top_journals = to_plot['Journal'].value_counts()
        top_20 = pd.DataFrame(top_journals.head(20))
        top_20 = top_20.reset_index()
        top_20.columns=['Journal', 'Citations count']
        return top_20
```

Year 2015

Operator For this year

	Journal	Citations count
0	J. Algebra	34
1	Comm. Algebra	10
2	J. Pure Appl. Algebra	10
3	Australas. J. Combin.	10
4	Discrete Math.	7
5	LMS J. Comput. Math.	6
6	Des. Codes Cryptogr.	6
7	J. Group Theory	5
8	J. Algebra Appl.	5
9	Algebr. Represent. Theory	4
10	J. Combin. Theory Ser. A	4
11	Bull. Iranian Math. Soc.	3
12	Electron. J. Combin.	3
13	Glas. Mat. Ser. III	3
14	IEEE Trans. Inform. Theory	3
15	J. Combin. Des.	3

	Journal	Citations count
16	Proc. Indian Acad. Sci. Math. Sci.	3
17	Math. Z.	3
18	Groups Complex. Cryptol.	3
19	Israel J. Math.	3

Top 15 GAP citing Journals ordered by their average `delay`

- The delay column reflects the difference between the publication date and the release date of the cited GAP version (in years). This difference might be considered as delay in recognizing/picking up/ new GAP releases or it could just be personal preference of publication authors to older GAP releases.
  For this table we use `gap_df` dataset, because GAP versions do not apply to GAP Packages, they have their own version systems.

In [45]:
```
jou_year = gap_df.groupby(['Journal'])['Delay'].mean()
jou_year = pd.DataFrame(data=jou_year)
jou_year = jou_year.sort_values(by='Delay', ascending=False)
jou_year.head(15)
```

Out[45]:

Journal	Delay
Bull. Inst. Math. Acad. Sin. (N.S.)	14.00
Vietnam J. Math.	11.00
J. Inst. Math. Jussieu	10.00
Ergodic Theory Dynam. Systems	9.50
Bull. Belg. Math. Soc. Simon Stevin	8.60
Dissertationes Math.	8.00
J. Geom. Phys.	8.00
Kyoto J. Math.	8.00
Adv. Geom.	7.78
Math. Slovaca	7.50
Bull. Malays. Math. Sci. Soc.	7.50
Ars Combin.	7.36
J. Lie Theory	7.33
Sém. Lothar. Combin.	7.33
Ann. Sci. Éc. Norm. Supér. (4)	7.00

The most frequent type of work citing GAP

Looking at the data, it seems GAP is predominantly cited in articles - 3393 times. In the raw data there were several other publication types present, such as `book` `phdthesis` and `mastersthesis`, but they were filtered out durign the data cleaning process, simply because their entries had missing information in too many important columns, making them unfeasible for our further data analysis.

In [46]:
```
types_pub = pd.DataFrame(merged_df['Publication Type'].value_counts())
types_pub.columns = ['GAP Citations Count']
types_pub.index.names = ['Publication Type']
types_pub
```

Out[46]:

Publication Type	GAP Citations Count
article	3383
incollection	43
inproceedings	4

Interactive table, displaying publication types breakdown for selected year.

The above table, with added interactive element, which allow users to filter the data by year. In the case of our data this tool does not yield diversified results, but if re-used for other datasets, the tool might prove handy.

In [47]:
```
@interact
def jou_by_year(Year=years_list):
    to_plot = merged_df[merged_df['Year'] == Year]['Publication Type'].value_cour
    to_plot = pd.DataFrame(data=to_plot)
    to_plot.columns = ['GAP Citations Count for: ' + str(Year)]
    to_plot.index.names = ['Publication Type']
    return to_plot
```

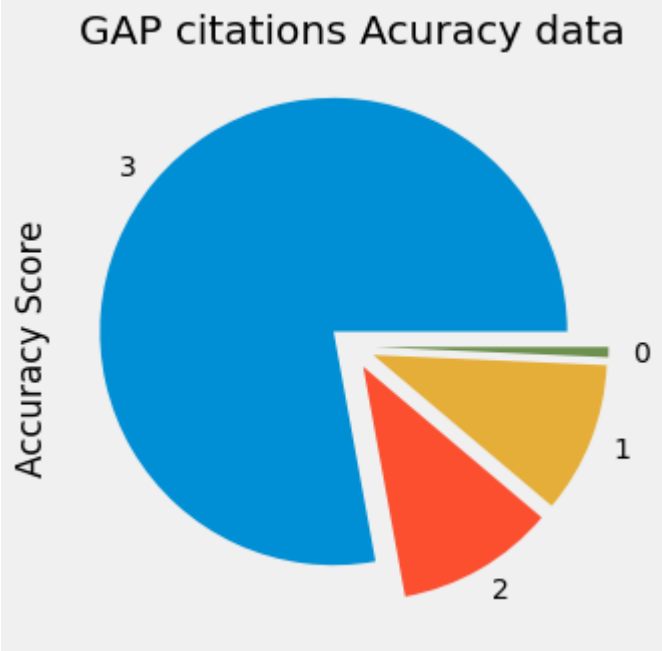
Year 2009

Publication Type	GAP Citations Count for: 2009
article	136

Accuracy

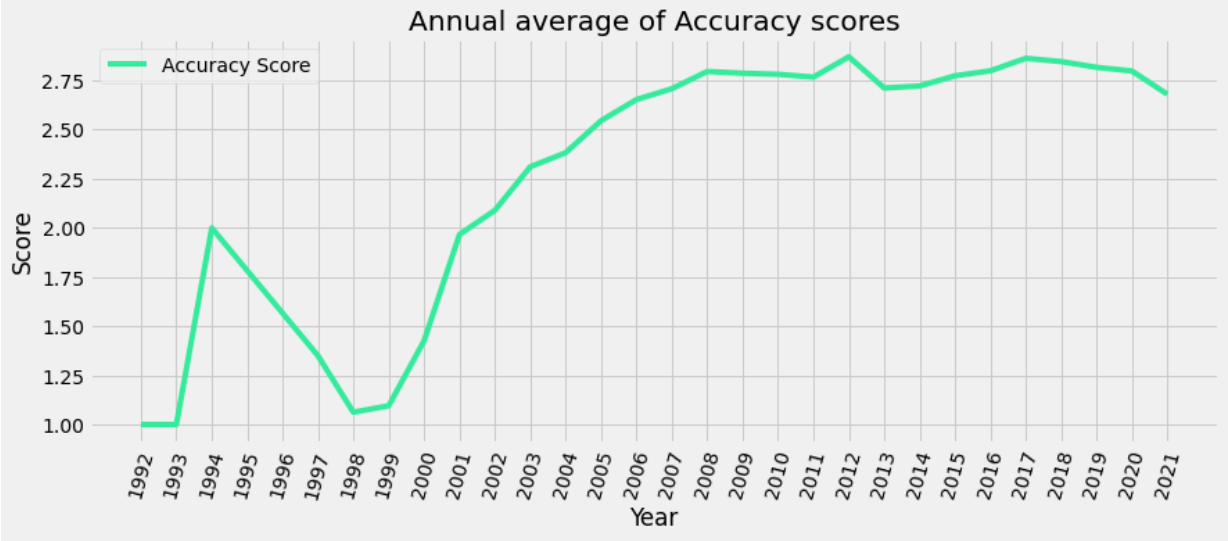
Fist we can display an overview by plotting the `Accuracy Score` coount of values for each score from 0 to 3.

In [48]:
```
acc = merged_df['Accuracy Score'].value_counts()
acc.plot(kind="pie", figsize=(5, 5), title='GAP citations Acuracy data',
         xlabel='Score',
         explode=(0.1, 0.1, 0.1, 0.1))
plt.show()
```



Then we group the data by year and take the mean of Accuracy Score for each group, to derive Annual average of Accuracy sores, which we plot in a line chart.

In [49]:
```
accuracy_mean = merged_df.groupby('Year')['Accuracy Score'].mean()
accuracy_mean = pd.DataFrame(data=accuracy_mean)
accuracy_mean.plot(kind='line', figsize=(13, 5),
        title='Annual average of Accuracy scores',
        xlabel='Year', ylabel='Score', color='#34eb9b')
plt.xticks(range(1992, 2022, 1))
plt.xticks(rotation=75)
plt.show()
```



In the line chart above we observe a spike in the Accuracy in 1993-4 then it falls again between 1995 and 1999. From there it starts rising again, to reach the maximum of 3 in late 2007. This level is kept until today.

Releases by year

We will use `gap_df` containing only GAP citations with versions provided, all other entries such as Package citations cannot be used in this part of the analysis because they do not have GAP Versions as such.

In [50]:
```
versions_cited = gap_df['Version'].unique()
versions_cited
```

Out[50]:
```
array(['Unknown', '4.7.5', '4.4.12', '4.7.8', '4.3', '4.4', '4.4.4',
       '4.6.4', '4.7.2', '4.6.5', '4.7.9', '4.8.8', '4.8.9', '4.4.10',
       '4.5.7', '4.7.4', '4.7.7', '4.2', '4.6.12', '4.1', '4.10.0',
       '4.8.6', '4.8.2', '4.10.2', '4.11.0', '4.4.2', '4.8.3', '4.8.7',
       '4.6', '4.9.1', '4.4.9', '4.8.4', '4.6.2', '4.9.3', '4.8.10',
       '4.5', '4.4.6', '4.4.7', '4.5.5', '4.4.11', '4.10.1', '4.6.3',
       '3.4', '4.8.5', '4.5.6', '4.11', '4.10', '4.9.2', '4.7.6', '4.6.9',
       '4.5.4', '3.4.4', '4.8', '4.9.0', '4.7', '3.3', '3.0', '4.5.3',
       '3.4.3', '4.6.1', '4.4.3', '4.8.1', '3.1', '4.4.5', '4.9', '3.2'],
      dtype=object)
```

Number of Citations by year of cited GAP release, in textual and graphic forms.

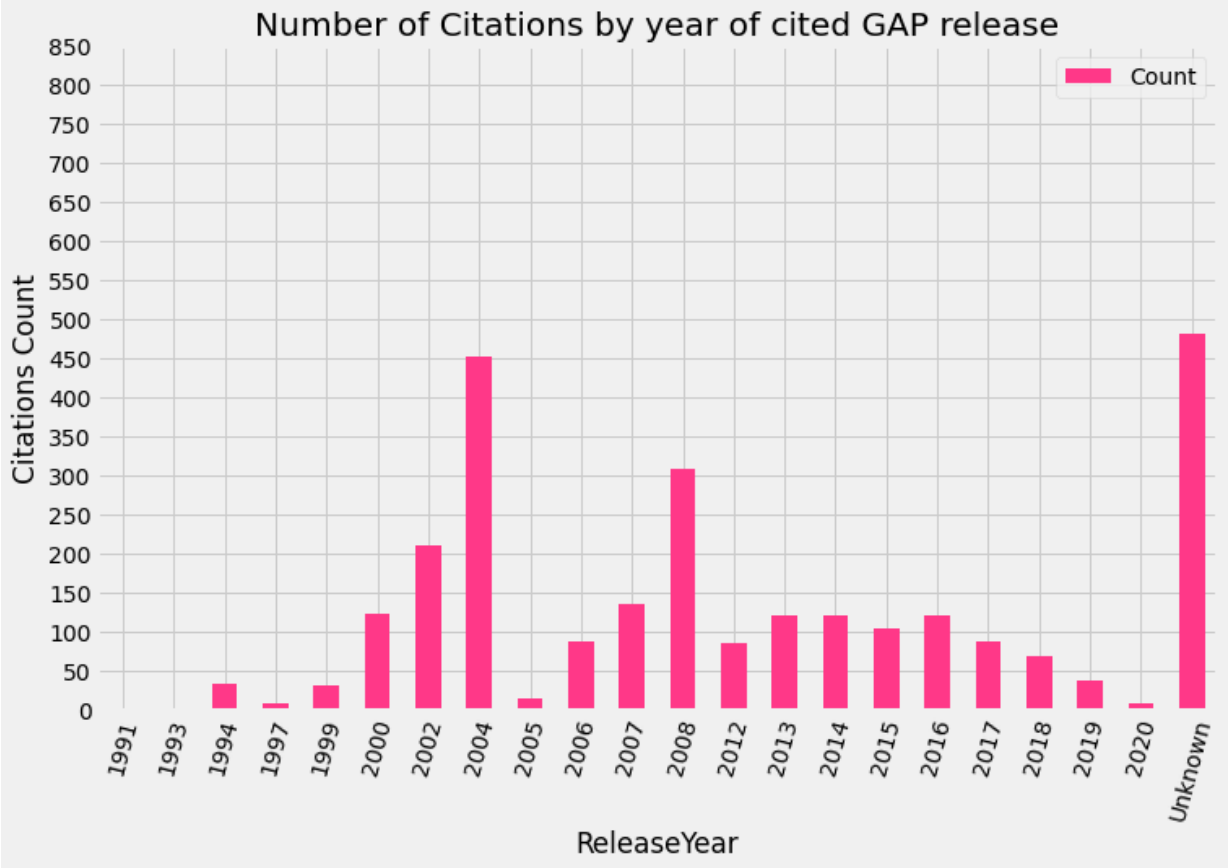
The table is sorted by citation count in descending order.

In [51]:
```
rel_years = gap_df['ReleaseYear'].value_counts()
rel_years = pd.DataFrame(data=rel_years)
rel_index = rel_years.index
rel_years.columns=['Count']
rel_years.index.names = ['ReleaseYear']
rel_years
```

Out[51]:

ReleaseYear	Count
Unknown	482
2004	451
2008	308
2002	211
2007	135
2000	122
2016	121
2014	120
2013	120
2015	105
2006	88
2017	88
2012	86
2018	68
2019	37
1994	33
1999	32
2005	14
1997	9
2020	9
1993	3
1991	3

In [52]:
```
rel_years = rel_years.sort_index()
rel_years.plot(kind='bar', figsize=(11, 7),
               title='Number of Citations by year of cited GAP release',
               color='#ff3888')
plt.xticks(rotation=75)
plt.yticks(range(0,900,50))
plt.ylabel('Citations Count')
plt.show()
```



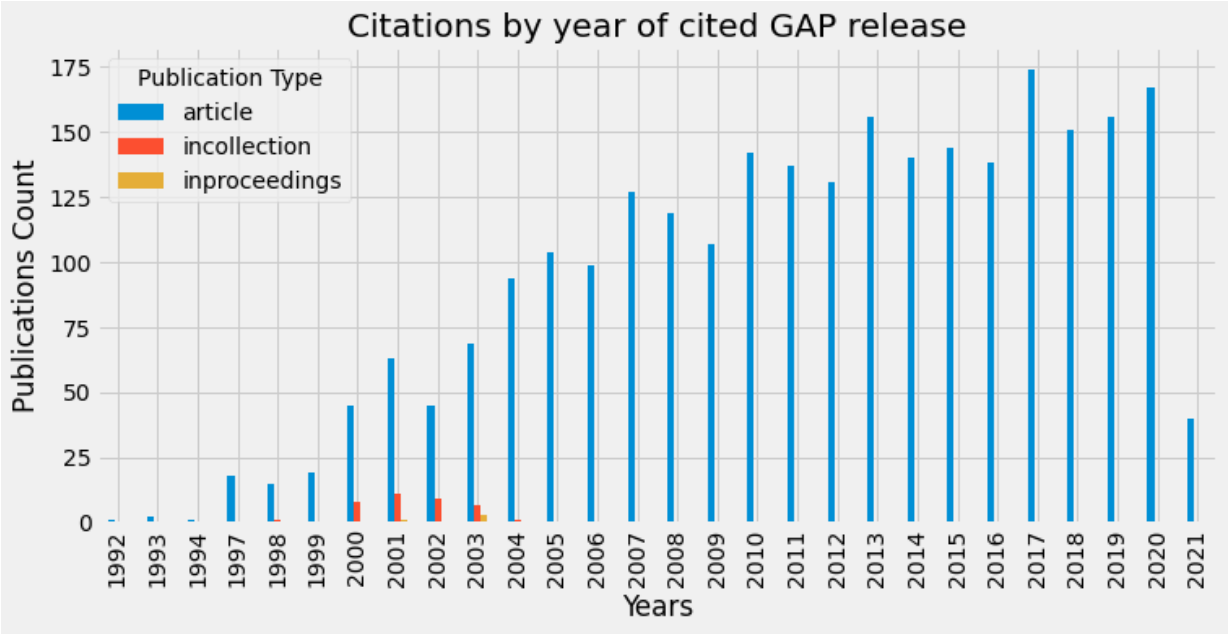
Types of GAP citing publications by year.

- Articles are the dominating type.

In [53]:
```
gap_df['Publication Type'].value_counts()
```

Out[53]:
```
article          2604
incollection       37
inproceedings       4
Name: Publication Type, dtype: int64
```

In [54]:
```
yea = gap_df.groupby(['Publication Type', 'Year'])['Publication Type'].count().ur
yea = pd.DataFrame(data=yea)
yea.plot(kind="bar", subplots=False,
         figsize=(11, 5),
         title='Citations by year of cited GAP release', xlabel='Years',
         ylabel='Publications Count')
plt.show()
```



Delay in picking up new GAP releases

Column `delay` is the result of release year minus publication year values.
Some `ReleaseYear` cells contain `Unknown` hence we have 1816 entries to work with on this query, which is enought to get a good notion of the trends.

In [55]:
```
gap_df[gap_df['ReleaseYear'] != 'Unknown'].count()
```

Out[55]:
```
MR                  2163
Author              2163
Journal             2163
Year                2163
Publication Type    2163
MSC                 2163
Citation            2163
Version             2163
Website             2163
Length              2163
Accuracy Score      2163
ReleaseYear         2163
Delay               2163
dtype: int64
```

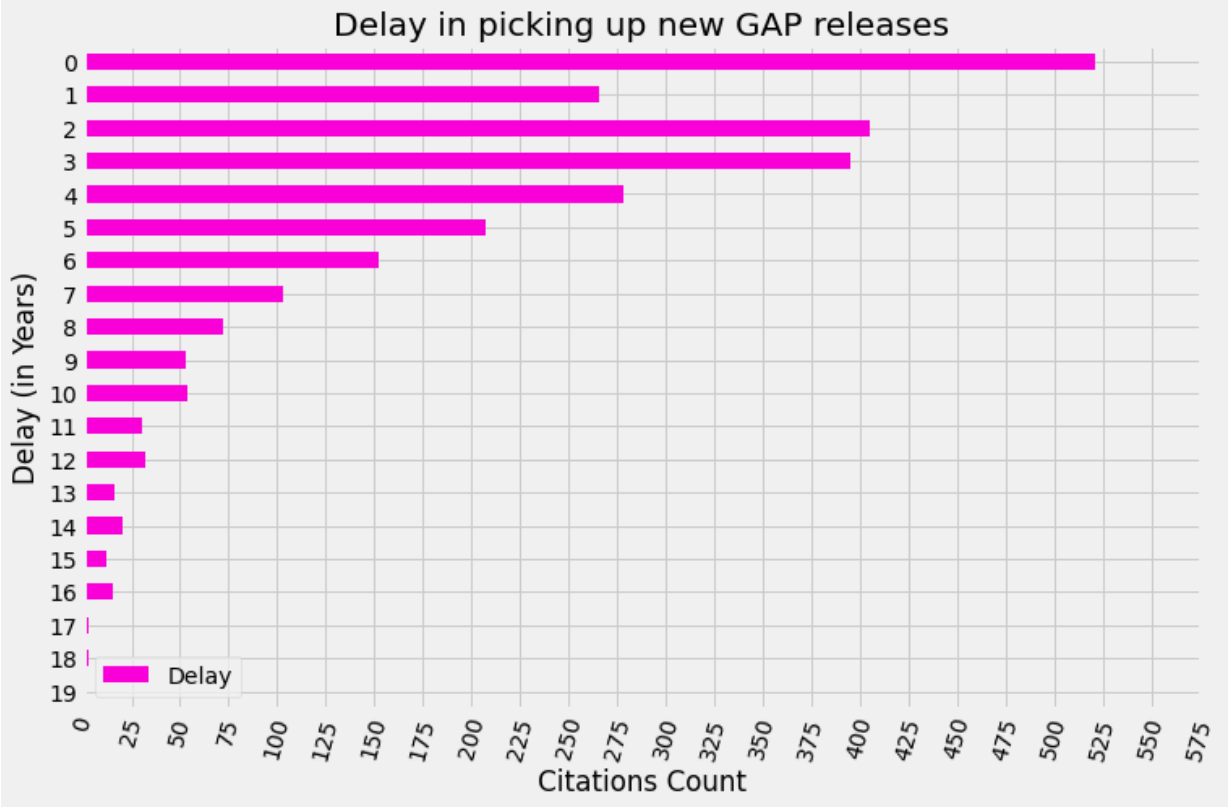
There are several negative values, potentially caused by errors in the data sources or somewhere along the pipeline. Their number is insignificant hence I will not investigate further. I will remove them from the `tail` subset of our data for better visualisation readability.

In [56]:
```
gap_df.loc[gap_df['Delay'] < 0]
```

1560	3422470	Keilberg, M.	Algebr. Represent. Theory	2015	article	16T05 (16W20 20D99)	GAP. GAP - Groups, Algorithms, and Programming...	4.8.4
1980	1958966	Moore, E. H. and Pollatsek, H.	Des. Codes Cryptogr.	2003	article	05B10 (20D60)	M. Schönert et al., GAP: Groups, Algorithms an...	4.4
2108	2810606	Niroomand, P. and Rezaei, R.	Comm. Algebra	2011	article	20P05 (20D99)	The GAP Group. (2008). GAP-Groups, Algorithms ...	4.11
2542	2372317	Wilde, T.	Comm.	2007	article	20C15	GAP (2005). The GAP	4.6

Below is the data in graphical form. The longest delays on record are 22 years but we only have a few such cases - less than 25. The most common delays are between 0 and 4 years, with 0 years leading the chart. We have 527 publications citing GAP versions in the same year these versions were released.

In [57]:
```
tail = gap_df['Delay'].value_counts()
tail = pd.DataFrame(data=tail)
tail = tail.sort_index(ascending=False)
tail.drop([-1,-5,-6, -9, -11], inplace=True) # to remove negative values
tail.plot.barh(figsize=(11,7), color='#fa00d9',
               title='Delay in picking up new GAP releases')
plt.xticks(range(0,600,25))
plt.xticks(rotation=75)
plt.xlabel('Citations Count')
plt.ylabel('Delay (in Years)')
plt.show()
```



More than 525 entries have 0 delay which means GAP versions were cited during the same year they were released, this represents a very good practice. Amongst the entries with positive delay those with 1 to 4 years delay are most common, having more than 200, 0 years being the most frequent.

Average delay by publication year

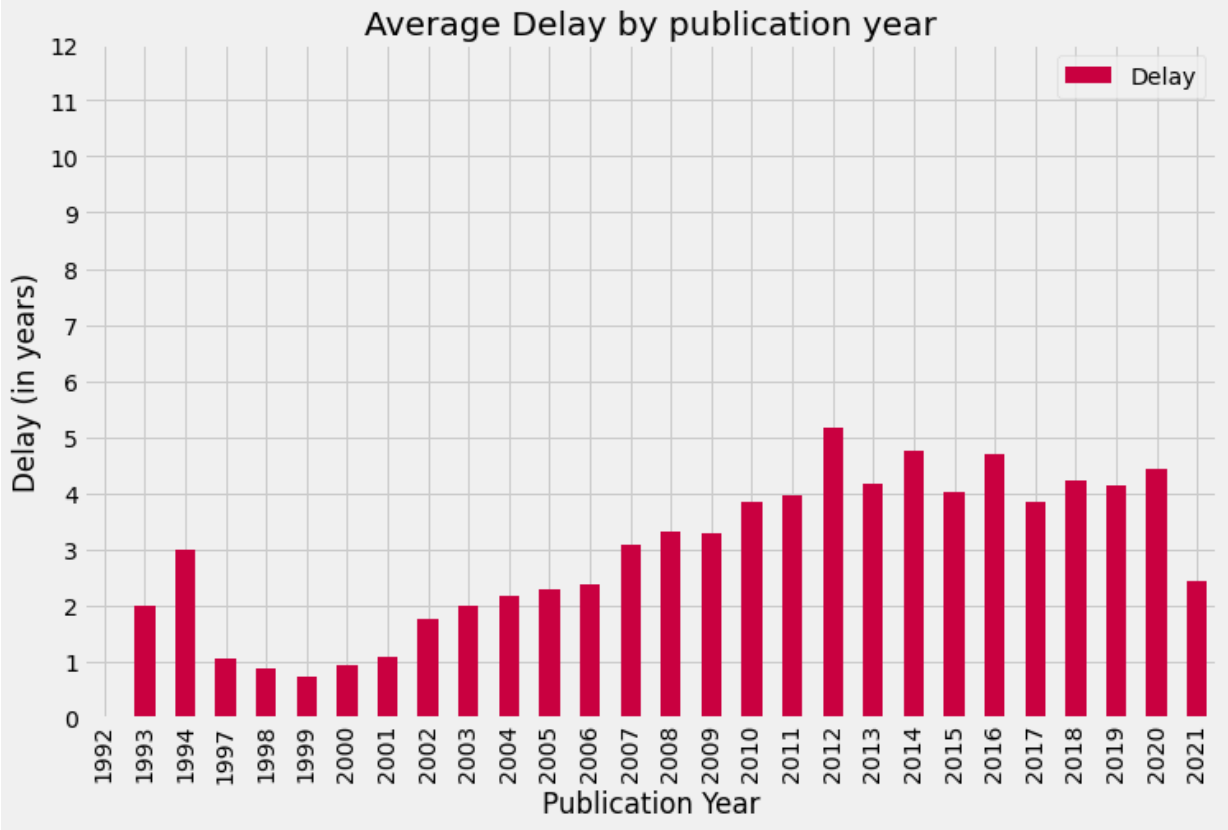
First we create a dataframe with the average delay for each publication year.
Then we plot it as a bar chart.

In [58]:
```
delay_average = gap_df.groupby(['Year'])['Delay'].mean()
delay_average = pd.DataFrame(data=delay_average)
#delay_average = delay_average.sort_values(by='Delay', ascending=False)
delay_average = delay_average.sort_index()
delay_average.to_csv('year_delay.csv', index=False, encoding='utf-8')
delay_average.head(5)
```

Out[58]:

Year	Delay
1992	0.00
1993	2.00
1994	3.00
1997	1.06
1998	0.88

In [59]:
```
delay_average.plot(kind="bar", color='#c90040', figsize=(11, 7),
                   title='Average Delay by publication year',
                   xlabel='Publication Year',
                   ylabel='Delay (in years)')
plt.yticks(range(0,13,1))
plt.show()
```



For instance, when looking at the chart we can deduce that the average Delay for all GAP Citing publications made in 2015 is almost 5 years – on average the cited GAP release during 2015 was

5 years old. Overall we observe significat delay average values across all years, which at first sight might seem concerning. The average delay is especially high in 2012, 2020 and 2021. This is not entirely a bad practice however, because of several possible reasons – authors:

- Prefer the older version
- Chose to use older version because it is better suited to their research needs
- Do not have access to a newer version
- Did not need a newer version
- Did not check if a new version exists

More Interactive Visualisations

Dataset timeline filter

Here we use `if` loops to introduce 3 operators: `Up To` , `For This Year` , `After` . Each of them combined with specified year allows users to filter the entire dataset.
As we will see, this filter can be used as a base for various other interactive visualisations.

In [60]:
```
years_list = gap_df['Year'].unique()
years_list = np.sort(years_list)[::-1] #sort the NumPy array in descending order
#years_list
```

In [61]:
```
@interact
def period_selector(Year=years_list,
                            operator=['Up to', 'For this year', 'After']):
    column='Year'
    if operator == 'For this year':
        return gap_df.loc[gap_df[column] == Year]
    if operator == 'Up to':
        return gap_df.loc[gap_df[column] <= Year]
    if operator == 'After':
        return gap_df.loc[gap_df[column] > Year]
```

Year 2021

operator Up to

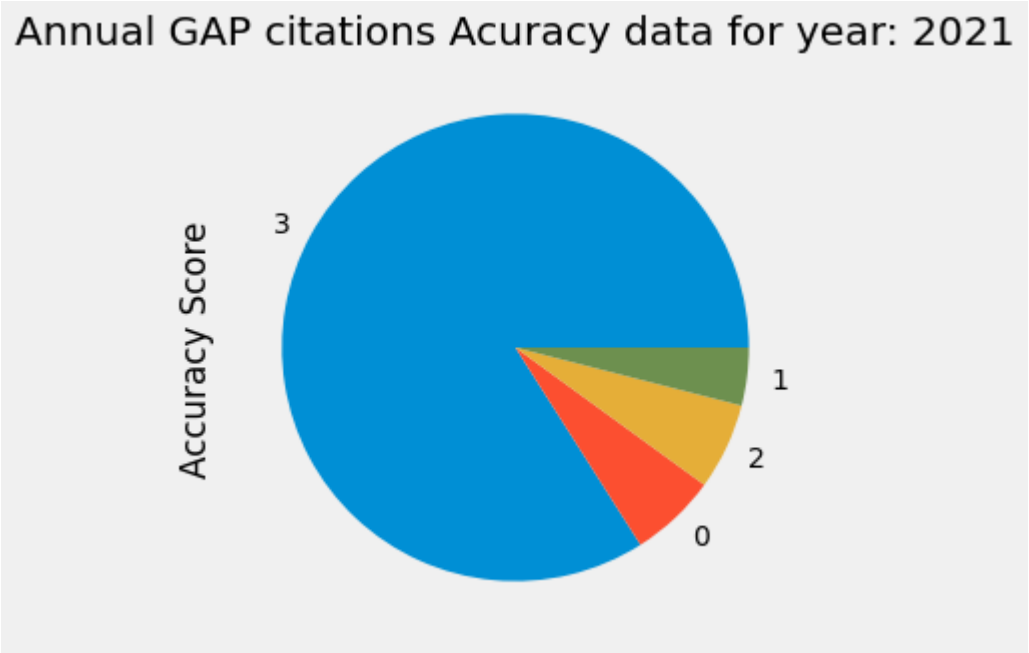
	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version
0	4056124	Abas, M. and Vetrík, T.	Theoret. Comput. Sci.	2020	article	05C25 (05C20 20F05)	GAP – Groups, algorithms, programming - a syst...	Unknown
1	3942387	Abbas, A. and Assi, A. and García-Sánchez, P. A.	Rev. R. Acad. Cienc. Exactas Fís. Nat. Ser. A ...	2019	article	13F20 (05E15 14H50)	The GAP Group: GAP —groups, algorithms, and pro...	4.7.5

Interactive pie chart displaying accuracy scores ratio for selected year

Here we combine the accuracy feature value counts with an year selector to filter the data.

In [62]:
```
@interact
def released_year_selector(Year = years_list):
    description='Select Year:'
    to_plot = merged_df[merged_df['Year'] == Year]
    to_plot = to_plot['Accuracy Score'].value_counts()
    to_plot.plot(kind="pie", figsize=(5, 5),
                 title='Annual GAP citations Acuracy data for year: ' + str(Year)
                 xlabel='Score',
                )
    return plt.show()
```

Year 2021



Version breakdown by year

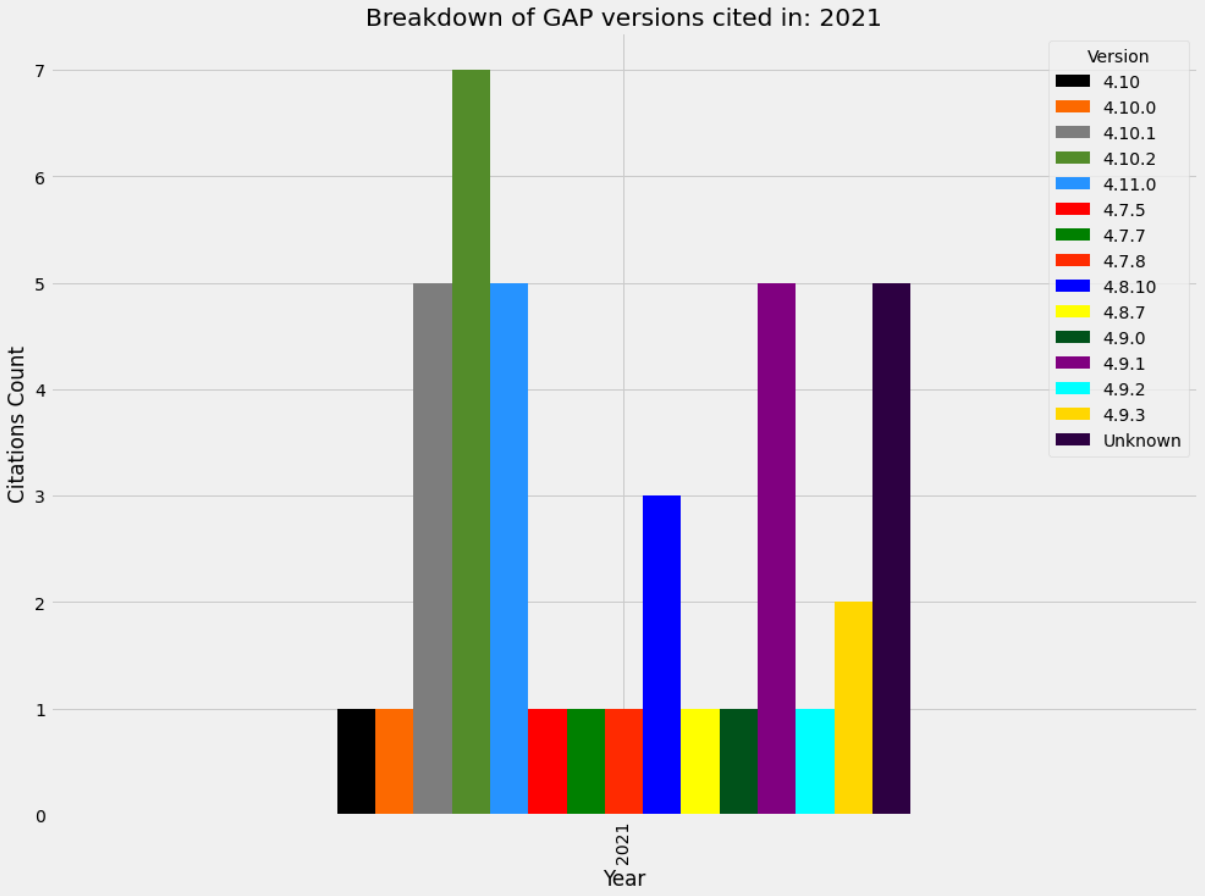
First we set a multi-color pallet.

In [63]:
```
my_colors = ['black', '#fc6900', '#7d7d7d', '#538c2a', '#2693ff', 'red',
             'green', '#ff2a00', 'blue', 'yellow', '#00521a',  'purple',
             'cyan', 'gold', '#2d0042', '#c7006a', '#660000', '#05f73e', '#adfc00
```

Then we group the data by version and year taking the version count.
We plot the result in a multi-color bar chart.
For som eyears version numbers exceed number of different colors but still distinction can be made.

In [64]:
```
@interact
def released_year_selector(Year = years_list):
    description='Select Year:'
    to_plot = gap_df[gap_df['Year'] == Year]
    to_plot = to_plot.groupby(['Version', 'Year'])['Version'].count().unstack('Ve
    to_plot.plot(kind="bar", figsize=(15, 11),
                         title='Breakdown of GAP versions cited in: ' + str(Year),
                 color=my_colors)
    plt.ylabel('Citations Count')
    return plt.show()
```

Year 2021



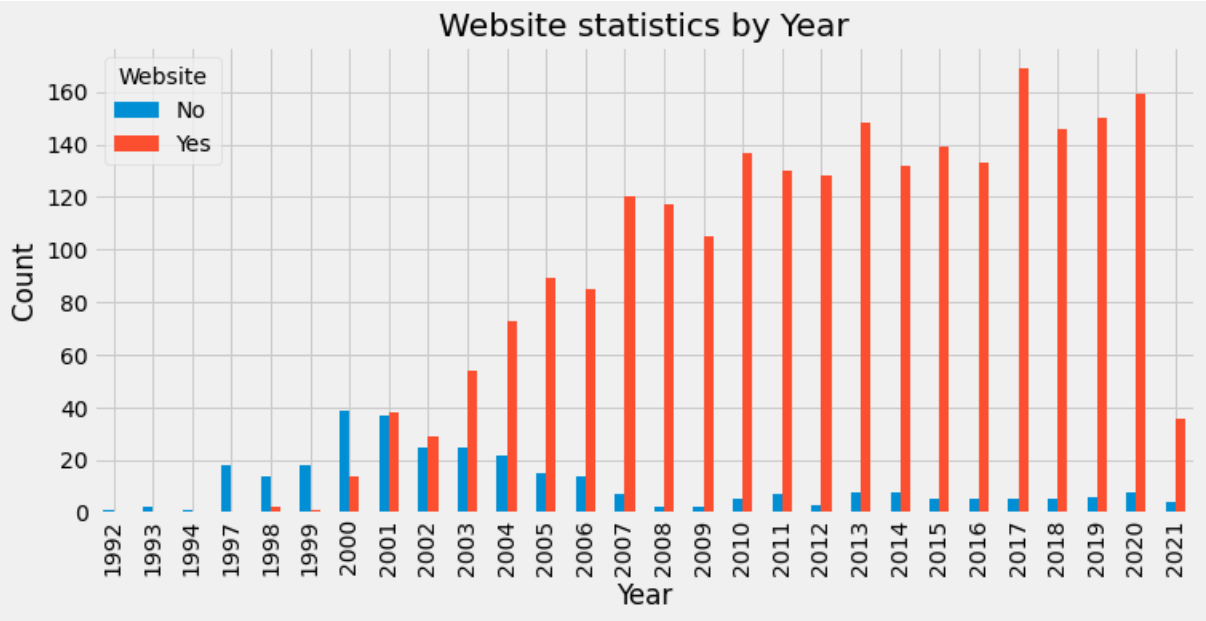
Interactive bar chart displaying website statistics, grouped by year, for selected time period.

Here we combine two already familiar elements: the website statistics bar chart and the time period selector. The resulting tool allows to view website data on, before or after a specified year.

In [65]:
```
@interact
def period_selector(Year=years_list, operator=['Up to', 'For this year', 'After']
    column='Year'
    if operator == 'For this year':
        site_year = gap_df.loc[gap_df[column] == Year]
        site_year = site_year.groupby(['Website', 'Year'])['Year'].count().unstac
        site_year_df = pd.DataFrame(data=site_year)
        site_year_df.plot(kind="bar", subplots=False, figsize=(11, 5),
                  title='Website statistics by Year',
                  xlabel='Year', ylabel='Count')
        return plt.show()
    if operator == 'Up to':
        site_year = gap_df.loc[gap_df[column] <= Year]
        site_year = site_year.groupby(['Website', 'Year'])['Year'].count().unstac
        site_year_df = pd.DataFrame(data=site_year)
        site_year_df.plot(kind="bar", subplots=False, figsize=(11, 5),
                  title='Website statistics by Year',
                  xlabel='Year', ylabel='Count')
        return plt.show()
    if operator == 'After':
        site_year = gap_df.loc[gap_df[column] > Year]
        site_year = site_year.groupby(['Website', 'Year'])['Year'].count().unstac
        site_year_df = pd.DataFrame(data=site_year)
        site_year_df.plot(kind="bar", subplots=False, figsize=(11, 5),
                  title='Website statistics by Year',
                  xlabel='Year', ylabel='Count')
        return plt.show()
```

Year 2021

operator Up to



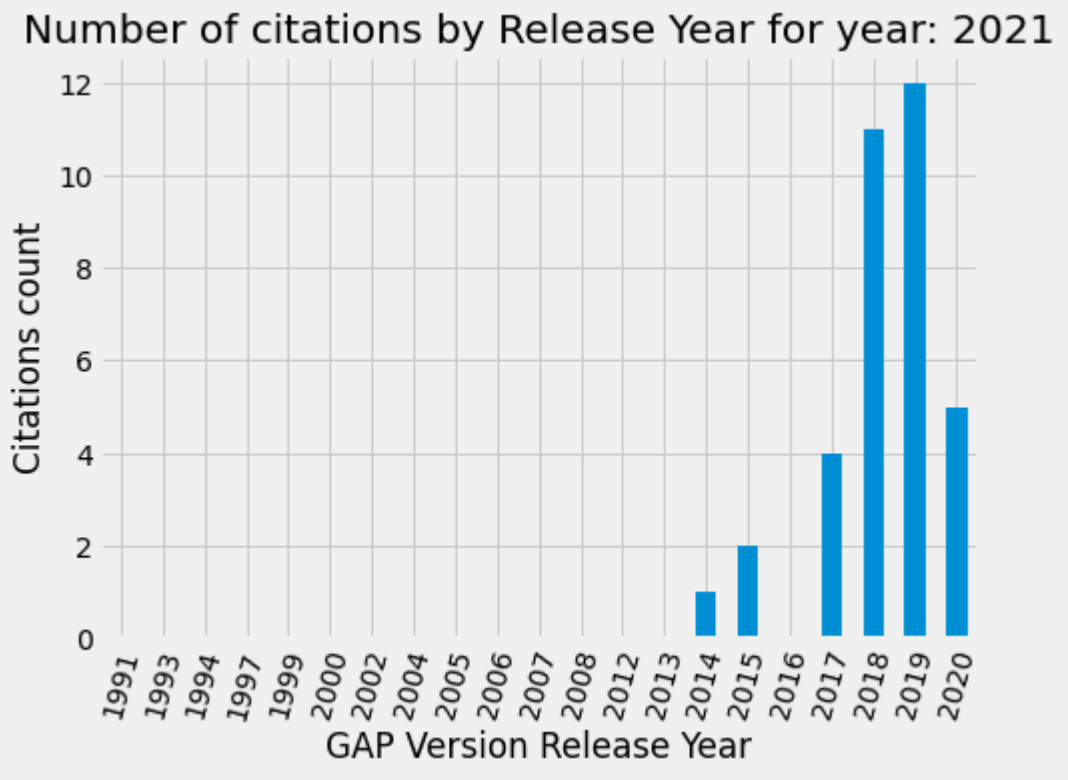
Interactive bar chart allowing user to select a publication year and then displaying numbers of GAP releases cited grouped in separate bars for each release year.

First we create a subset of our data, containing the number of citations per release year.
Then we combine that with the already familiar time filter which allows user to filter the displayed data by year. This tool can be used for in-depth analysis and investigations.

In [66]:
```
# we exclude entries with Unknown release year.
filtered_df = gap_df[gap_df['ReleaseYear'] != 'Unknown']
df3 = filtered_df.groupby(['Year', 'ReleaseYear'])['MR'].count().unstack('Year').
```

In [67]:
```
@interact
def year_selector(Year=years_list):
    description='Select Year:'
    to_plot = df3[Year]
    to_plot.plot.bar(figsize=(7, 5),
            title='Number of citations by Release Year for year: ' + str(Year))
    plt.xticks(rotation=75)
    plt.xlabel('GAP Version Release Year')
    plt.ylabel('Citations count')
    return plt.show()
```

Year 2021



Interactive bar chart allowing user to select Release Year and see the

number of citations grouped in bars - one for each Publication year

First we create a list of all Release years, for users to select from in the drop-down, sorted in descending order.

In [68]:
```
rel_list = filtered_df['ReleaseYear'].unique()
rel_list = np.sort(rel_list)
rel_list = np.sort(rel_list)[::-1]
```

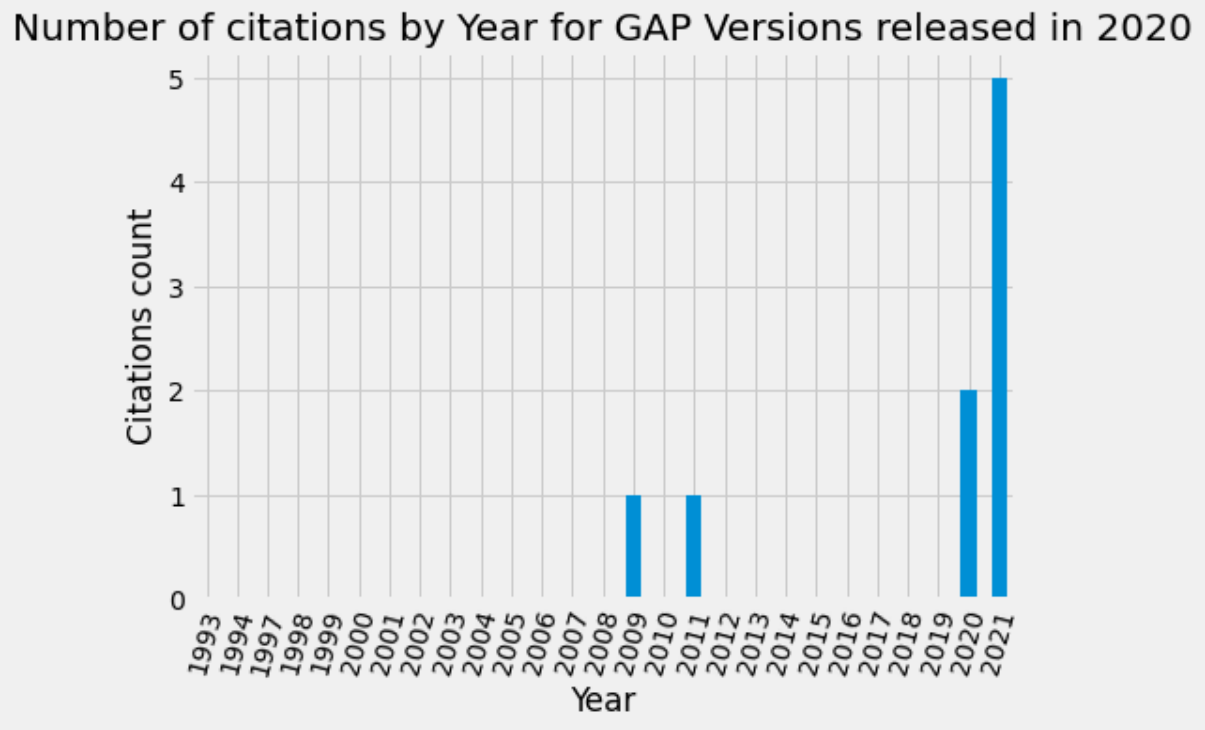
- Then we filter the data and create a new daraframe with a column for each Release year and a row for each year of publication with publication count in every intersecting cell.

In [69]:
```
df5 = filtered_df.groupby(['ReleaseYear', 'Year'])['MR'].count().unstack('Release
```

- Finally we visualise it, allowing users to select Release Year.
  Here we have created dynamic plot title by inserting a variable and concatenating it with one or more strings. Thus the title updates everytime the users changes their selection.

In [70]:
```
@interact
def released_year_selector(ReleaseYear = rel_list):
    description='Select Release Year:'
    to_plot1 = df5[ReleaseYear]
    to_plot1.plot.bar(figsize=(7, 5),
        title='Number of citations by Year for GAP Versions released in ' + Relea
    plt.xticks(rotation=75)
    plt.ylabel('Citations count')
    return plt.show()
```

ReleaseYear 2020



MSC Codes

Looking at the `MSC` column we have the primary MSC codes, followed by the secondary ones. The secondary MSCs are sometimes several and they are in brackets, in other entries there are no secondary MSCs stated, therefore we need to split the MSC data in two columns, before mapping the codes to a python dictionary to convert them to the corresponding full MSC Area names.

In [71]:
```
for index, row in merged_df.iterrows():
    print(row['MSC'])
```

```
05C25 (05C20 20F05)
13F20 (05E15 14H50)
13F20 (05E15 14H50)
05C25 (20B30 20E45)
20G40 (05C25)
20F45 (20D60 20F19)
20D60
20F99
20D15
20D60
20B30 (20D60)
20F05 (20F28)
16S50 (16P10 16U70)
20F45 (20F18)
20F45 (20F12)
20D45 (20D15)
20D60 (51E21)
20D45
20D60 (05C25)
```

The first 5 characters are always the primary MSC, hence we will separate them in a column. The rest of each cell represents Secondary MSC codes - there can be one, two, three or there can be no secondary MSCs at all, therefore we will further process this column.

In [72]:
```
merged_df['MSC Primary'] = [x[:5] for x in merged_df['MSC']]
merged_df['MSC Secondary'] = [x[5:] for x in merged_df['MSC']]
```

In [73]:
```
merged_df['MSC Primary'].head()
```

Out[73]:
```
0    05C25
1    13F20
2    13F20
3    05C25
4    20G40
Name: MSC Primary, dtype: object
```

In [74]:
```
merged_df['MSC Secondary'].head()
```

Out[74]:
```
0     (05C20 20F05)
1     (05E15 14H50)
2     (05E15 14H50)
3     (20B30 20E45)
4           (05C25)
Name: MSC Secondary, dtype: object
```

Now that we have split Primary and Secondary MSCs in two columns, we need the science area names corresponding to each MSC code.

- First we create the dictionary, using this PDF file provided by MathSciNet. Accessible on this link: https://mathscinet.ams.org/msnhtml/msc2020.pdf (https://mathscinet.ams.org/msnhtml/msc2020.pdf)

In [75]:
```
msc_text = """00 General and overarching topics; collections
01 History and biography
03 Mathematical logic and foundations
05 Combinatorics
06 Order, lattices, ordered algebraic structures
08 General algebraic systems
11 Number theory
12 Field theory and polynomials
13 Commutative algebra
14 Algebraic geometry
15 Linear and multilinear algebra; matrix theory
16 Associative rings and algebras
17 Nonassociative rings and algebras
18 Category theory; homological algebra
19 K-theory
20 Group theory and generalizations
22 Topological groups, Lie groups
26 Real functions
28 Measure and integration
30 Functions of a complex variable
31 Potential theory
32 Several complex variables and analytic spaces
33 Special functions
34 Ordinary differential equations
35 Partial differential equations
37 Dynamical systems and ergodic theory
39 Difference and functional equations
40 Sequences, series, summability
41 Approximations and expansions
42 Harmonic analysis on Euclidean spaces
43 Abstract harmonic analysis
44 Integral transforms, operational calculus
45 Integral equations
46 Functional analysis
47 Operator theory
49 Calculus of variations and optimal control; optimization
51 Geometry
52 Convex and discrete geometry
53 Differential geometry
54 General topology
55 Algebraic topology
57 Manifolds and cell complexes
58 Global analysis, analysis on manifolds
60 Probability theory and stochastic processes
62 Statistics
65 Numerical analysis
68 Computer science
70 Mechanics of particles and systems
74 Mechanics of deformable solids
76 Fluid mechanics
78 Optics, electromagnetic theory
80 Classical thermodynamics, heat transfer
81 Quantum theory
82 Statistical mechanics, structure of matter
83 Relativity and gravitational theory
85 Astronomy and astrophysics
```

```
86 Geophysics
90 Operations research, mathematical programming
91 Game theory, economics, social and behavioral sciences
92 Biology and other natural sciences
93 Systems theory; control
94 Information and communication, circuits
97 Mathematics education"""
```

With the code below we convert the text to a python dictionary, by taking the first two characters from each line - the numbers - as keys, and the rest of each line as value. Thus the keys, match what we currently have in the MSC columns of our Pandas dataframe and we can then proceed to mapping or in other words replacing the numbers with corresponding MSC area names.

In [76]:
```
msc_dict = {l[:2]: l[3:] for l in msc_text.splitlines()}
msc_dict
```

Out[76]:
```
{'00': 'General and overarching topics; collections',
 '01': 'History and biography',
 '03': 'Mathematical logic and foundations',
 '05': 'Combinatorics',
 '06': 'Order, lattices, ordered algebraic structures',
 '08': 'General algebraic systems',
 '11': 'Number theory',
 '12': 'Field theory and polynomials',
 '13': 'Commutative algebra',
 '14': 'Algebraic geometry',
 '15': 'Linear and multilinear algebra; matrix theory',
 '16': 'Associative rings and algebras',
 '17': 'Nonassociative rings and algebras',
 '18': 'Category theory; homological algebra',
 '19': 'K-theory',
 '20': 'Group theory and generalizations',
 '22': 'Topological groups, Lie groups',
 '26': 'Real functions',
 '28': 'Measure and integration',
 '30': 'Functions of a complex variable',
 '31': 'Potential theory',
 '32': 'Several complex variables and analytic spaces',
 '33': 'Special functions',
 '34': 'Ordinary differential equations',
 '35': 'Partial differential equations',
 '37': 'Dynamical systems and ergodic theory',
 '39': 'Difference and functional equations',
 '40': 'Sequences, series, summability',
 '41': 'Approximations and expansions',
 '42': 'Harmonic analysis on Euclidean spaces',
 '43': 'Abstract harmonic analysis',
 '44': 'Integral transforms, operational calculus',
 '45': 'Integral equations',
 '46': 'Functional analysis',
 '47': 'Operator theory',
 '49': 'Calculus of variations and optimal control; optimization',
 '51': 'Geometry',
 '52': 'Convex and discrete geometry',
 '53': 'Differential geometry',
 '54': 'General topology',
 '55': 'Algebraic topology',
 '57': 'Manifolds and cell complexes',
 '58': 'Global analysis, analysis on manifolds',
 '60': 'Probability theory and stochastic processes',
 '62': 'Statistics',
 '65': 'Numerical analysis',
 '68': 'Computer science',
 '70': 'Mechanics of particles and systems',
 '74': 'Mechanics of deformable solids',
 '76': 'Fluid mechanics',
 '78': 'Optics, electromagnetic theory',
 '80': 'Classical thermodynamics, heat transfer',
 '81': 'Quantum theory',
 '82': 'Statistical mechanics, structure of matter',
```

```
 '83': 'Relativity and gravitational theory',
 '85': 'Astronomy and astrophysics',
 '86': 'Geophysics',
 '90': 'Operations research, mathematical programming',
 '91': 'Game theory, economics, social and behavioral sciences',
 '92': 'Biology and other natural sciences',
 '93': 'Systems theory; control',
 '94': 'Information and communication, circuits',
 '97': 'Mathematics education'}
```

Processing the Primary column.

We will need the first two digits of each MSC code only, so we can map it to our dictionary and convert it to a science area name.

In [77]:
```
merged_df['MSC Primary'] = [x[:2] for x in merged_df['MSC Primary']] # taking fir
merged_df["MSC Primary"] = merged_df["MSC Primary"].map(msc_dict) # converting di
merged_df["MSC Primary"].head() # instepc result
```

Out[77]:
```
0                       Combinatorics
1                  Commutative algebra
2                  Commutative algebra
3                       Combinatorics
4    Group theory and generalizations
Name: MSC Primary, dtype: object
```

Processing the Secondary column.

We will use several Python string methods:
First we apply a Regex expression which strips all unnecessary characters, then takes the first two characters of each word (in our case each MSC code).
Then we map the results to our dictionary.
Finally we use `.groupby(level=0).agg(list)` to group the Series by index and put them back into lists.

In [78]:
```
merged_df["MSC Secondary"] = (
    merged_df["MSC Secondary"]
    .str.extractall(r"[ \(](\w{2})")[0]
    .map(msc_dict)
    .groupby(level=0).agg(list)
)
```

We need to check the data type for the column and the amount of `NaN` values, to be taken care of.

In [79]:
```
type(merged_df["MSC Secondary"][1])
```

Out[79]:
```
list
```

In [80]:
```
merged_df["MSC Secondary"].isnull().values.sum()
```

Out[80]:
```
665
```

We fill ne `NaN` cells with the string `No` . Then we can use `lambda` with `pd.unique` to remove duplicates.

In [81]:
```
merged_df["MSC Secondary"] = merged_df["MSC Secondary"].fillna('No')
merged_df["MSC Secondary"] = merged_df["MSC Secondary"].apply(lambda x: list(pd.u
merged_df["MSC Secondary"].head(3)
```

Out[81]:
```
0    [Combinatorics, Group theory and generalizations]
1                  [Combinatorics, Algebraic geometry]
2                  [Combinatorics, Algebraic geometry]
Name: MSC Secondary, dtype: object
```

In [82]:
```
type(merged_df["MSC Secondary"][1])
```

Out[82]:
```
list
```

- We will create a new dataframe solely for MSC Areas statistics.

In [83]:
```
msc_df = merged_df["MSC Primary"].value_counts()
msc_df = msc_df.to_frame()
msc_df = msc_df.reset_index()
msc_df.columns = ['MSC Area', 'Primary Count']
msc_df.head() # this is the base, now we start adding to it
```

Out[83]:

	MSC Area	Primary Count
0	Group theory and generalizations	1785
1	Combinatorics	420
2	Associative rings and algebras	201
3	Number theory	124
4	Nonassociative rings and algebras	122

We will add another column reflecting the MSC Areas which are only in Secondary but not Primary (in the same row).

- Using `.explode` to further analyse the `MSC Secondary` column, splitting each list in each cell into separate rows.

In [84]:
```
kin = merged_df.explode('MSC Secondary')
kin
```

Out[84]:

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Website	I
0	4056124	Abas, M. and Vetrík, T.	Theoret. Comput. Sci.	2020	article	05C25 (05C20 20F05)	GAP – Groups, algorithms, programming - a syst...	Unknown	Yes	
0	4056124	Abas, M. and Vetrík, T.	Theoret. Comput. Sci.	2020	article	05C25 (05C20 20F05)	GAP – Groups, algorithms, programming - a syst...	Unknown	Yes	
1	3942387	Abbas, A. and Assi, A. and García-Sánchez, P. A.	Rev. R. Acad. Cienc. Exactas Fís. Nat. Ser. A ...	2019	article	13F20 (05E15 14H50)	Delgado, M., García-Sánchez, P.A., Morais, J.:...	Package	Yes	
1	3942387	Abbas, A. and Assi, A. and García-Sánchez, P. A.	Rev. R. Acad. Cienc. Exactas Fís. Nat. Ser. A ...	2019	article	13F20 (05E15 14H50)	Delgado, M., García-Sánchez, P.A., Morais, J.:...	Package	Yes	
2	3942387	Abbas, A. and Assi, A. and García-Sánchez, P. A.	Rev. R. Acad. Cienc. Exactas Fís. Nat. Ser. A ...	2019	article	13F20 (05E15 14H50)	The GAP Group: GAP —groups, algorithms, and pro...	4.7.5	Yes	
...	...	...	...	...	...	...	...	...	...	
3426	1801202	Shaw, R.	Des. Codes Cryptogr.	2000	incollection	51E14	L.H. Soicher, GRAPE: a system for computing wi...	Package	Yes	
3427	2558870	Sidki, S. N.	J. Algebra	2009	article	16S50 (15B33 16P90)	L. Bartholdi, Functionally recursive groups, h...	Unknown	Yes	
3427	2558870	Sidki, S. N.	J. Algebra	2009	article	16S50 (15B33 16P90)	L. Bartholdi, Functionally recursive groups, h...	Unknown	Yes	
3428	2824780	Tian, Y. and Lu, C.	J. Ind. Manag. Optim.	2011	article	90C20 (90C11 90C46)	X. Sun, C. Liu, D. Li and J. Gao, On duality g...	Unknown	Yes	

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Website	I
3429	1981371	Zhurtov, A. K.	Sibirsk. Mat. Zh.	2003	article	20F99	Schönert M. et al., Groups, Algorithms and Pro...	Unknown	Yes	

4562 rows × 13 columns

In [85]:
```
kin['MSC Secondary'].iloc[1]
```

Out[85]:
```
'Group theory and generalizations'
```

- This is how we count the occurences of one MSC Area as secondary, in our dataset.

In [86]:
```
kin['MSC Secondary'].str.count("Combinatorics").sum()
```

Out[86]:
```
469
```

Now we can add the other coulmn which counts each occurance of an MSC area as Secondary in our main dataframe `merged_df` .

In [87]:
```
msc_df['Secondary Count'] = [kin['MSC Secondary'].str.count(x).sum() for x in msc
```

In [88]:
```
msc_df.head()
```

Out[88]:

	MSC Area	Primary Count	Secondary Count
0	Group theory and generalizations	1785	1739
1	Combinatorics	420	469
2	Associative rings and algebras	201	200
3	Number theory	124	203
4	Nonassociative rings and algebras	122	113

We will add a third column to display the count of appearances of each MSC area, as secondary only.

In [89]:
```
msc_df_sec = kin[kin['MSC Primary'] != kin['MSC Secondary']] # checking it is dif
msc_df_sec = msc_df_sec[msc_df_sec['MSC Secondary'] != 'No'] # excluding citatior
```

In [90]:
```
sec = msc_df_sec['MSC Secondary'].value_counts()
sec_df = pd.DataFrame(data=sec)
sec_df = sec_df.reset_index()
sec_df.columns = ['MSC Area', 'Only As Secondary']
sec_df
```

Out[90]:

	MSC Area	Only As Secondary
0	Group theory and generalizations	543
1	Combinatorics	274
2	Computer science	131
3	Number theory	117
4	Associative rings and algebras	87
5	Algebraic geometry	78
6	Manifolds and cell complexes	65
7	Geometry	64
8	Information and communication, circuits	52
9	Convex and discrete geometry	46
10	Commutative algebra	44
11	Nonassociative rings and algebras	43
12	Algebraic topology	35
13	Order, lattices, ordered algebraic structures	30
14	Linear and multilinear algebra; matrix theory	28
15	Category theory; homological algebra	28
16	Field theory and polynomials	26
17	Quantum theory	24
18	Topological groups, Lie groups	20
19	Numerical analysis	16
20	Functions of a complex variable	16
21	K-theory	13
22	Dynamical systems and ergodic theory	12
23	Operations research, mathematical programming	12
24	Mathematical logic and foundations	10
25	Probability theory and stochastic processes	10
26	General algebraic systems	9
27	Differential geometry	8
28	Statistics	6
29	Global analysis, analysis on manifolds	6
30	Operator theory	6

	MSC Area	Only As Secondary
31	Functional analysis	5
32	Special functions	5
33	Biology and other natural sciences	3
34	Partial differential equations	3
35	Several complex variables and analytic spaces	3
36	Ordinary differential equations	3
37	Abstract harmonic analysis	3
38	Measure and integration	2
39	General and overarching topics; collections	2
40	Statistical mechanics, structure of matter	2
41	History and biography	2
42	Difference and functional equations	2
43	Sequences, series, summability	1
44	Fluid mechanics	1
45	Mathematics education	1
46	Calculus of variations and optimal control; op...	1
47	Harmonic analysis on Euclidean spaces	1
48	General topology	1
49	Optics, electromagnetic theory	1

We merge the last subset with the existing data-frame, so it becomes third column there. We also fill null values and fix data type and remove the `_merge` column.

In [91]:
```
new_msc_df = pd.merge(msc_df, sec_df, on='MSC Area', how='left', indicator=True)
```

In [92]:
```
new_msc_df = new_msc_df.drop(columns="_merge")
new_msc_df['Only As Secondary'] = new_msc_df['Only As Secondary'].fillna(0)
new_msc_df['Only As Secondary'] = new_msc_df['Only As Secondary'].astype(int)
```

MSC Areas classified by number of uses as Primary, Secondary and Secondary only, in GAP Citations

Here is the result of our work.

In [93]:
```
new_msc_df.head(43)
```

Out[93]:

	MSC Area	Primary Count	Secondary Count	Only As Secondary
0	Group theory and generalizations	1785	1739	543
1	Combinatorics	420	469	274
2	Associative rings and algebras	201	200	87
3	Number theory	124	203	117
4	Nonassociative rings and algebras	122	113	43
5	Algebraic geometry	121	155	78
6	Information and communication, circuits	100	73	52
7	Manifolds and cell complexes	92	118	65
8	Geometry	90	90	64
9	Convex and discrete geometry	67	73	46
10	Computer science	40	148	131
11	Commutative algebra	35	63	44
12	Field theory and polynomials	34	38	26
13	Algebraic topology	26	51	35
14	Category theory; homological algebra	24	43	28
15	Dynamical systems and ergodic theory	18	21	12
16	Linear and multilinear algebra; matrix theory	16	29	28
17	K-theory	13	14	13
18	Operations research, mathematical programming	10	18	12
19	Order, lattices, ordered algebraic structures	8	31	30
20	General algebraic systems	8	10	9
21	Quantum theory	8	30	24
22	Differential geometry	8	9	8
23	Topological groups, Lie groups	7	22	20
24	Statistics	6	12	6
25	Functions of a complex variable	5	18	16
26	Numerical analysis	4	16	16
27	Several complex variables and analytic spaces	4	3	3
28	Functional analysis	4	6	5
29	Harmonic analysis on Euclidean spaces	4	3	1
30	Probability theory and stochastic processes	4	13	10
31	Mathematical logic and foundations	3	11	10
32	Global analysis, analysis on manifolds	3	7	6

	MSC Area	Primary Count	Secondary Count	Only As Secondary
33	Biology and other natural sciences	3	5	3
34	Systems theory; control	2	1	0
35	Partial differential equations	2	5	3
36	Special functions	2	6	5
37	General and overarching topics; collections	2	2	2
38	Ordinary differential equations	1	4	3
39	Game theory, economics, social and behavioral ...	1	0	0
40	Mechanics of particles and systems	1	1	0
41	Mechanics of deformable solids	1	1	0
42	Operator theory	1	6	6

MSC Areas classified by number of uses as Primary in our GAP Citation database

`Group theory and generalizations` with 1791 citations is the most frequently used to categorise GAP citations, followed by `Combinatorics` with 420 and `Associative rings and algebras` with 201 citations. This chart encompasses the whole data across all years. The next visualisation is interactive and allows the user to filter this same chart by year.

In [94]:
```
pri = merged_df['MSC Primary'].value_counts()
pri = pd.DataFrame(data=pri)
pri = pri.reset_index()
pri.columns = ['MSC Areas', 'Count']
pri
```

Out[94]:

	MSC Areas	Count
0	Group theory and generalizations	1785
1	Combinatorics	420
2	Associative rings and algebras	201
3	Number theory	124
4	Nonassociative rings and algebras	122
5	Algebraic geometry	121
6	Information and communication, circuits	100
7	Manifolds and cell complexes	92
8	Geometry	90
9	Convex and discrete geometry	67
10	Computer science	40

Interactive table displaying most popular MSC Areas used as primary for selected year

Here we combine the already familiar year filter with counts of unique Primary MSC areas.

In [95]:
```
@interact
def year_priMSC(Year=years_list):
    description='Select Year:'
    to_plot = merged_df[merged_df['Year'] == Year]
    pri = to_plot['MSC Primary'].value_counts()
    pri = pd.DataFrame(data=pri)
    pri = pri.reset_index()
    pri.columns = ['MSC Areas', 'Count']
    return pri
```

Year 2021

	MSC Areas	Count
0	Group theory and generalizations	20
1	Nonassociative rings and algebras	10
2	Combinatorics	6
3	Associative rings and algebras	6
4	Order, lattices, ordered algebraic structures	3
5	Commutative algebra	2
6	Information and communication, circuits	1
7	Category theory; homological algebra	1
8	Algebraic geometry	1

Interactive line chart displaying number of uses of selected MSC Area as Primary in GAP Citations

Here we need to combine with `Year` columns, hence we use the full data, group it by `MSC Primary` and `Year`, then take the records count. We `unstack` on MSC Primary which results in a column for each MSC area.

In [96]:
```
df9 = merged_df.groupby(['MSC Primary', 'Year'])['MR'].count().unstack('MSC Prima
msc_list = df9.columns
df9.head()
```

Out[96]:

MSC Primary	Algebraic geometry	Algebraic topology	Associative rings and algebras	Biology and other natural sciences	Category theory; homological algebra	Combinatorics	Commutative algebra	Co
Year								
1992	0.00	0.00	1.00	0.00	0.00	0.00	0.00	
1993	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1994	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1997	1.00	0.00	0.00	0.00	0.00	0.00	1.00	
1998	1.00	0.00	0.00	0.00	0.00	2.00	0.00	

5 rows × 43 columns

The variable which will be changed by users in this interactive plot is MSC Area, hence we use the `msc_list` .

In [97]:
```
@interact
def msc_sel(MSC = msc_list):
    to_plot3 = df9[MSC]
    to_plot3.plot.line(figsize=(11, 5),
                title='Number of citations per year for "' + MSC + '" as Primary
                                xlabel='Year', ylabel='Citations count')
    plt.xticks(range(1992,2021,1))
    plt.xticks(rotation=75)
    return plt.show()
```

MSC: Algebraic geometry

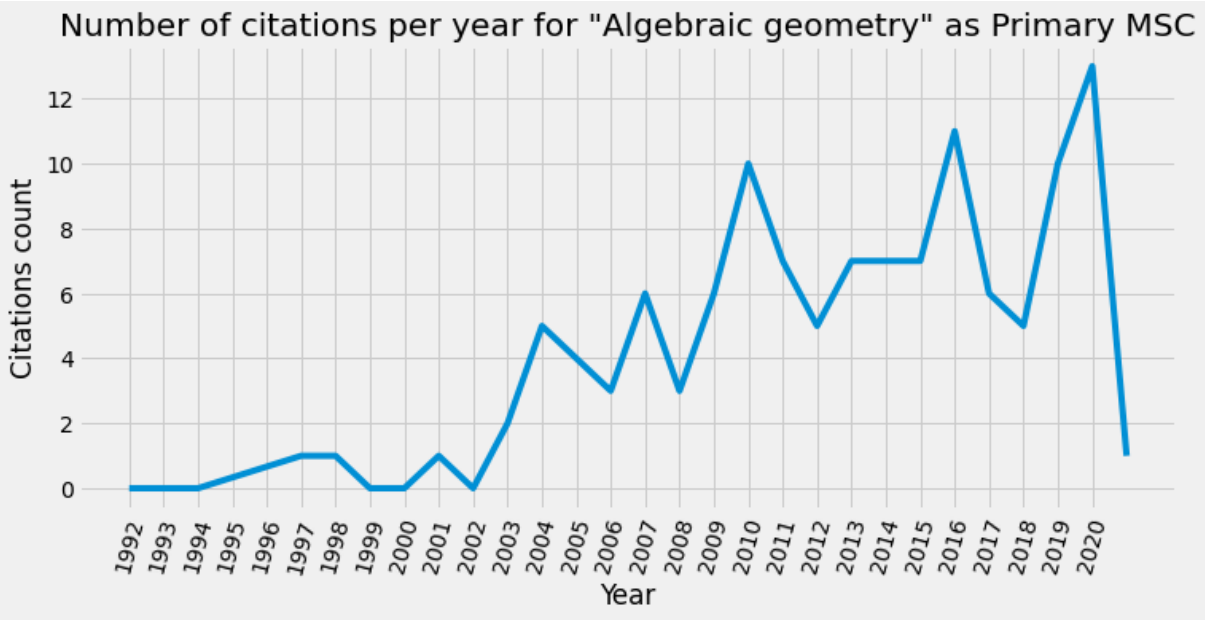


Table displaying Top 15 MSC Areas or groups, used as Secondary Only in GAP Citations

- 'No' represents the number of citations where no Secondary areas were stated. `Group theory and generalizations` is the most popular as secondary MSC area, used on its own or combined with others. The number of citations not stating secondary MSC is also high - 665 and takes the second place.

In [98]:
```
sec = merged_df['MSC Secondary'].value_counts()
sec = pd.DataFrame(data=sec)
sec = sec.reset_index()
sec.columns = ['MSC Areas', 'Count']
sec = sec.head(15)
sec
```

Out[98]:

	MSC Areas	Count
0	[Group theory and generalizations]	1148
1	[No]	665
2	[Combinatorics]	209
3	[Combinatorics, Group theory and generalizations]	112
4	[Associative rings and algebras, Group theory ...	100
5	[Number theory]	80
6	[Algebraic geometry]	64
7	[Nonassociative rings and algebras]	58
8	[Group theory and generalizations, Computer sc...	56
9	[Associative rings and algebras]	48
10	[Manifolds and cell complexes]	46

Interactive table displaying most popular MSC areas used separately or in groups as secondary, for selected year

In [99]:
```
@interact
def year_priMSC(Year=years_list):
    description='Select Year:'
    to_plot = merged_df[merged_df['Year'] == Year]
    sec = to_plot['MSC Secondary'].value_counts()
    sec = pd.DataFrame(data=sec)
    sec = sec.reset_index()
    sec.columns = ['MSC Areas', 'Count']
    return sec
```

Year 2021

	MSC Areas	Count
0	[Group theory and generalizations]	17
1	[No]	6
2	[Nonassociative rings and algebras, Group theo...	5
3	[Combinatorics, Geometry]	4
4	[Combinatorics, Convex and discrete geometry]	3
5	[Group theory and generalizations, Topological...	2
6	[Associative rings and algebras]	2
7	[Algebraic geometry]	2
8	[Associative rings and algebras, Nonassociativ...	2
9	[Commutative algebra, Algebraic geometry, Asso...	1

After selecting all years one by one, `Group theory and generalizations` is definitely the most commonly MSC Area used as secondary, followed by `Combinatorics` . The other one in top 3 is `No` which represents citatiosn without secondary MSC areas being stated.

Number of uses as Secondary, for selected MSC areas, by Year

We repeat the code we used to visualise Primary MSCs, but for Secondary.
The list of sciences here is slightly different because some of the Secondary MSCs are grouped, therefore we create a new list formed by the columns of the new subset, creted below.

In [100]:
```
df1 = kin.groupby(['MSC Secondary', 'Year'])['MR'].count().unstack('MSC Secondary
df1.head()
```

Out[100]:

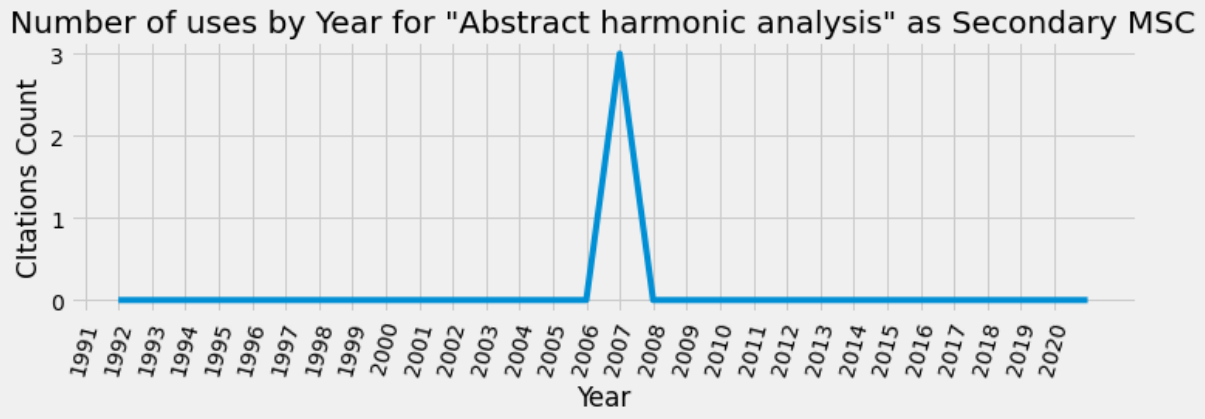
MSC Secondary	Abstract harmonic analysis	Algebraic geometry	Algebraic topology	Associative rings and algebras	Biology and other natural sciences	Calculus of variations and optimal control; optimization	Category theory; homological algebra	Comb
Year								
1992	0.00	0.00	0.00	1.00	0.00	0.00	0.00	
1993	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1994	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1997	0.00	1.00	0.00	0.00	0.00	0.00	0.00	
1998	0.00	1.00	0.00	0.00	0.00	0.00	0.00	

5 rows × 54 columns

In [101]:
```
sec_list = df1.columns # updated selection list for Secondary MSCs
```

In [102]:
```
@interact
def msc_sel(SeconaryMSC = sec_list):
    to_plot3 = df1[SeconaryMSC]
    to_plot3.plot.line(figsize=(11, 3),
                title='Number of uses by Year for "' + SeconaryMSC + '" as Secondary MSC',
                                xlabel='Year', ylabel='CItations Count')
    plt.xticks(np.arange(1991, 2021, 1.0))
    plt.xticks(rotation=75)
    return plt.show()
```

SeconaryM… Abstract harmonic analysis



Number of uses as Primary, for selected single MSC area, by Year

In [103]:
```
df11 = kin.groupby(['MSC Primary', 'Year'])['MR'].count().unstack('MSC Primary').
df11.head()
```

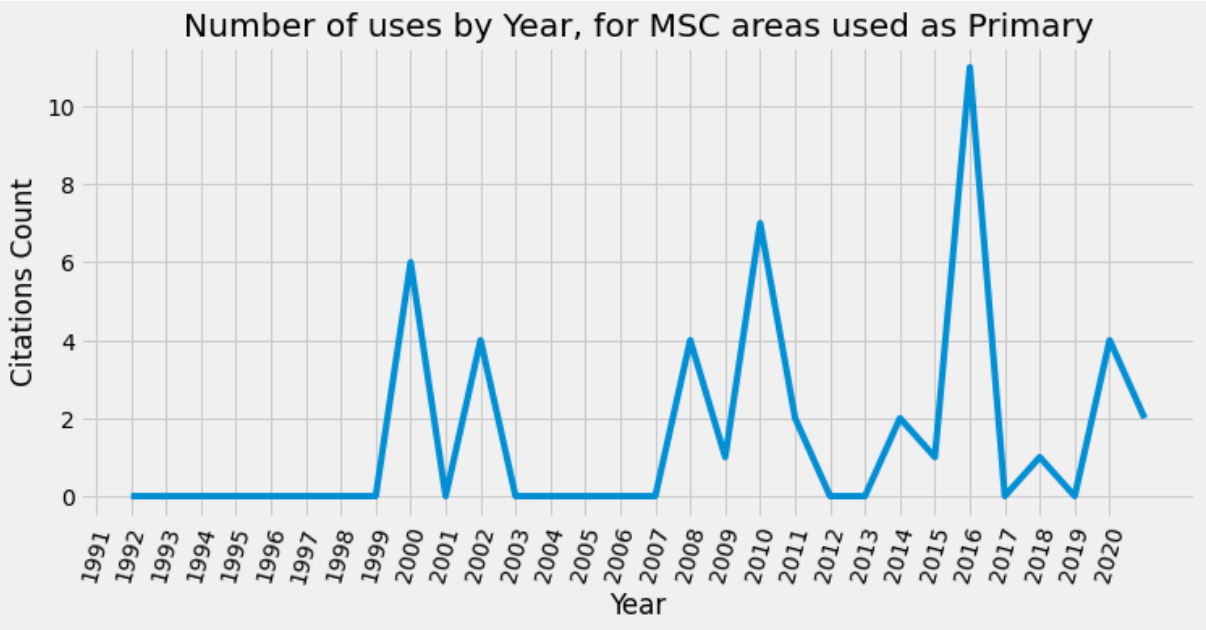
Out[103]:

MSC Primary	Algebraic geometry	Algebraic topology	Associative rings and algebras	Biology and other natural sciences	Category theory; homological algebra	Combinatorics	Commutative algebra	Co
Year								
1992	0.00	0.00	2.00	0.00	0.00	0.00	0.00	
1993	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1994	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
1997	3.00	0.00	0.00	0.00	0.00	0.00	2.00	
1998	1.00	0.00	0.00	0.00	0.00	2.00	0.00	

5 rows × 43 columns

In [104]:
```
@interact
def year_priMSC(MSC = df11.columns):
    description='Select Year:'
    to_plot = df11[MSC]
    to_plot.plot.line(figsize=(11, 5),
                title='Number of uses by Year, for MSC areas used as Primary',
                                xlabel='Year', ylabel='Citations Count')
    plt.xticks(np.arange(1991, 2021, 1.0))
    plt.xticks(rotation=75)
    return plt.show()
```

MSC: Category theory; homological al



Correlation between Journal and Accuracy Score

We will analyse perform analysis to find out if such correlation exists or not.

Top 35 journals by number of GAP citations

First we create a table with journal names as index and the citations count for each.
Journal of Algebra has the most GAP citations - almost 500.

In [105]:
```
top35 = merged_df.groupby('Journal')['MR'].count()
top35 = top35.sort_values(ascending=False)
top35 = pd.DataFrame(data=top35)
top35.columns=['GAP Citations Count']
top35.head(35)
```

Out[105]:

Journal	GAP Citations Count
J. Algebra	499
Comm. Algebra	200
J. Symbolic Comput.	147
J. Algebra Appl.	111
Discrete Math.	106
Internat. J. Algebra Comput.	85
J. Group Theory	81
Des. Codes Cryptogr.	70
J. Pure Appl. Algebra	70
LMS J. Comput. Math.	61
Math. Comp.	55
Israel J. Math.	49
Experiment. Math.	48
J. Combin. Des.	46
European J. Combin.	44
J. Combin. Theory Ser. A	39
Electron. J. Combin.	38
Trans. Amer. Math. Soc.	37
Arch. Math. (Basel)	36
Algebra Colloq.	35
Adv. Math.	32
Semigroup Forum	31
J. Algebraic Combin.	29
Algebr. Represent. Theory	27
Linear Algebra Appl.	26
Bull. Iranian Math. Soc.	25
Forum Math.	24
Discrete Comput. Geom.	23
Sibirsk. Mat. Zh.	22
Pacific J. Math.	21

Journal	GAP Citations Count
Exp. Math.	21
Monatsh. Math.	21
J. Knot Theory Ramifications	21
Ars Math. Contemp.	20
IEEE Trans. Inform. Theory	19

A table displaying the top 35 Journals by sum of their accuracy scores.

- The result is heavily impacted by the number of citations by each journal, therefore we will try to find the average in the next section.

In [106]:
```
fil_df = merged_df.groupby('Journal')['Accuracy Score'].sum()
fil_df = pd.DataFrame(data=fil_df)
fil_df = fil_df.sort_values(by='Accuracy Score', ascending=False)
fil_df.columns=['Total Accuracy Score']
fil_df.head(35)
```

Out[106]:

Journal	Total Accuracy Score
J. Algebra	1300
Comm. Algebra	541
J. Symbolic Comput.	354
J. Algebra Appl.	313
Discrete Math.	292
Internat. J. Algebra Comput.	220
J. Group Theory	205
J. Pure Appl. Algebra	195
Des. Codes Cryptogr.	188
LMS J. Comput. Math.	177
Israel J. Math.	128
J. Combin. Des.	127
Math. Comp.	126
Experiment. Math.	123
European J. Combin.	116
J. Combin. Theory Ser. A	108
Electron. J. Combin.	107
Arch. Math. (Basel)	103
Trans. Amer. Math. Soc.	93
Algebra Colloq.	93
Adv. Math.	89
Semigroup Forum	86
J. Algebraic Combin.	76
Algebr. Represent. Theory	75
Bull. Iranian Math. Soc.	72
Linear Algebra Appl.	70
Forum Math.	69
Monatsh. Math.	61
Pacific J. Math.	59
Discrete Comput. Geom.	58

Journal	Total Accuracy Score
J. Knot Theory Ramifications	57
Exp. Math.	56
Bull. Aust. Math. Soc.	55
IEEE Trans. Inform. Theory	54
Ars Math. Contemp.	52

Average Accuracy score journals citing GAP, but still ordered by number of GAP citations.

In [107]:
```
# First we create a list of journals and their average accuracy scores.
jou_lis = fil_df.index
mea_lis = []
for i in jou_lis:
    mea_score = merged_df[merged_df['Journal']==i]['Accuracy Score'].mean()
    mea_lis.append(i)
    mea_lis.append(str(mea_score))
```

In [108]:
```
# Then we transform it to pandas dataframe and reshape it so each row has journal
mea_lis_df = pd.DataFrame(data=mea_lis)
mea_lis_df.columns=['Journal']
mea_lis_df = pd.DataFrame(mea_lis_df.Journal.values.reshape(-1,2),columns=['Jourr
```

After looking in the tabular representation below, of average accuracy scores for journals arranged by number of GAP citations, we can see that Journals with highest number of GAP citations do not have perfect accuracy score but are not far off the mark, with averages ranging from 2.4 to 2.89 for the top 35.

In [109]:
```
mea_lis_df.head(35)
```

Out[109]:

	Journal	Average Score
0	J. Algebra	2.6052104208416833
1	Comm. Algebra	2.705
2	J. Symbolic Comput.	2.4081632653061225
3	J. Algebra Appl.	2.81981981981982
4	Discrete Math.	2.7547169811320753
5	Internat. J. Algebra Comput.	2.588235294117647
6	J. Group Theory	2.5308641975308643
7	J. Pure Appl. Algebra	2.7857142857142856
8	Des. Codes Cryptogr.	2.6857142857142855
9	LMS J. Comput. Math.	2.901639344262295
10	Israel J. Math.	2.6122448979591835
11	J. Combin. Des.	2.760869565217391
12	Math. Comp.	2.290909090909091
13	Experiment. Math.	2.5625
14	European J. Combin.	2.6363636363636362
15	J. Combin. Theory Ser. A	2.769230769230769
16	Electron. J. Combin.	2.8157894736842106
17	Arch. Math. (Basel)	2.861111111111111
18	Trans. Amer. Math. Soc.	2.5135135135135136
19	Algebra Colloq.	2.657142857142857
20	Adv. Math.	2.78125
21	Semigroup Forum	2.774193548387097
22	J. Algebraic Combin.	2.6206896551724137
23	Algebr. Represent. Theory	2.7777777777777777
24	Bull. Iranian Math. Soc.	2.88
25	Linear Algebra Appl.	2.6923076923076925
26	Forum Math.	2.875
27	Monatsh. Math.	2.9047619047619047
28	Pacific J. Math.	2.8095238095238093
29	Discrete Comput. Geom.	2.5217391304347827
30	J. Knot Theory Ramifications	2.7142857142857144
31	Exp. Math.	2.6666666666666665
32	Bull. Aust. Math. Soc.	2.8947368421052633
33	IEEE Trans. Inform. Theory	2.8421052631578947

	Journal	Average Score
34	Ars Math. Contemp.	2.6

In the tail, we find that journals with fewer GAP citations have relatively lower accuracy averages of 1 to 2. This can be explained by with the continuous improvement principle.

In [110]:
```
mea_lis_df.tail(20)
```

Out[110]:

	Journal	Average Score
268	J. Gökova Geom. Topol. GGT	2.0
269	Nonlinearity	2.0
270	Math. Biosci. Eng.	2.0
271	Arch. Ration. Mech. Anal.	2.0
272	J. Ind. Manag. Optim.	2.0
273	J. Amer. Statist. Assoc.	2.0
274	Adv. Comput. Math.	2.0
275	SIAM J. Sci. Comput.	2.0
276	J. Funct. Anal.	1.0
277	Ann. Sci. École Norm. Sup. (4)	1.0
278	Electron. J. Differential Equations	1.0
279	Mat. Zametki	1.0
280	J. Amer. Math. Soc.	1.0
281	Math. Oper. Res.	1.0
282	Abh. Math. Sem. Univ. Hamburg	1.0
283	Numer. Math.	1.0
284	Order	1.0
285	Bull. Soc. Math. France	1.0
286	J. Nonlinear Sci.	1.0
287	New Zealand J. Math.	0.0

A subject-specific journal is a base for a community involving its authors, readers, reviewers, editors, a group of scientists, who often work together and share views, ideas, and practices, including software citation. Therefore, the more times a journal cites certain software the more likely it is that the accuracy of its citations improves all the authors who write for this journal adopt the same, improved set of principles for citing that software.

Packages

Questions to be answered:

- What is the proportion of GAP Packages citation against the total number of entries we have?
- How did the number of GAP Packages citation changes over the years, compared to the number of pure GAP citations?
- When authors cite a package, how often do they also separately cite GAP?

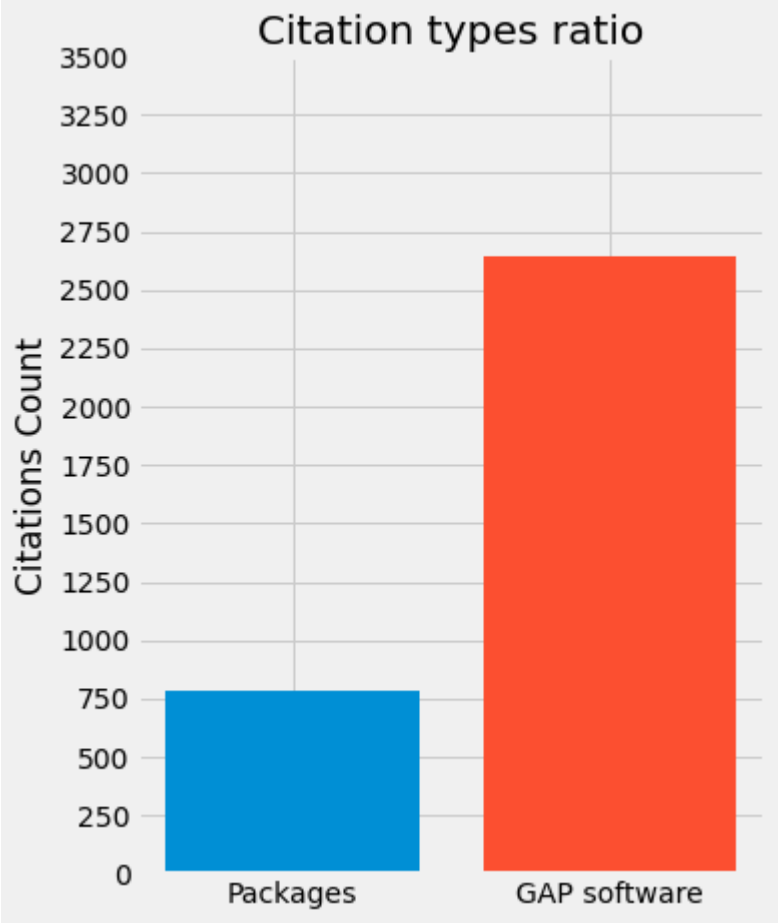
First we find the number of Packages agains number of all citations.

In [111]:
```
pac = merged_df[merged_df['Version'] == 'Package']['MR'].count()
full = merged_df[merged_df['Version'] != 'Package']['MR'].count()
print(pac, '/', full)
```

```
785 / 2645
```

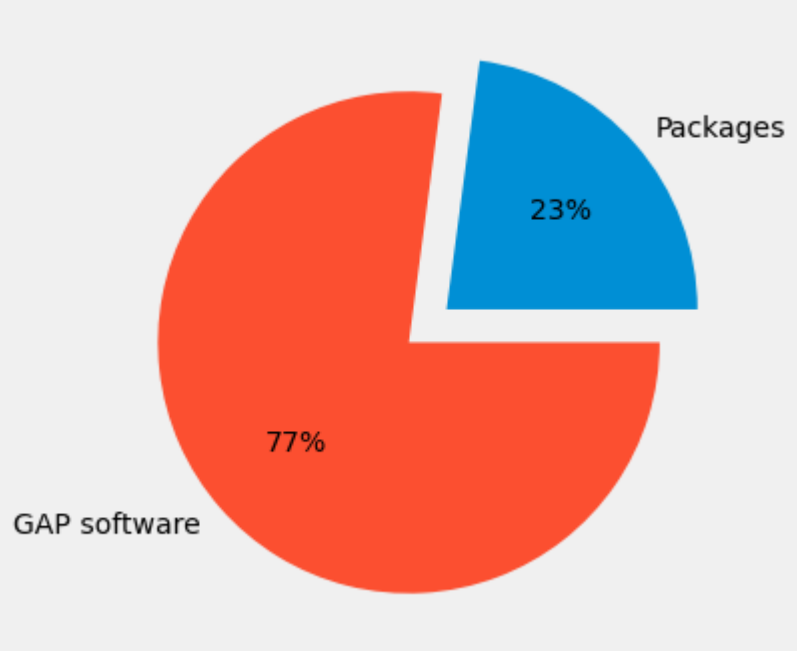
Then we can plot the information as two bars in a chart, easy to compare.

In [112]:
```
labels = ['Packages', 'GAP software'] # we label the bars
data = [pac, full] # we prepare data instances
fig = plt.figure(figsize =(5, 7)) # we set figure size
plt.bar(height = pac, x = 'Packages') # we assign data instances to eahc bar to c
plt.bar(height = full, x = 'GAP software') # we assign data instances to eahc bar
plt.yticks(np.arange(0,3700,250)) # we fine-tune the y axis ticks
plt.ylabel('Citations Count')
plt.title('Citation types ratio') # we name our plot
plt.show()
```



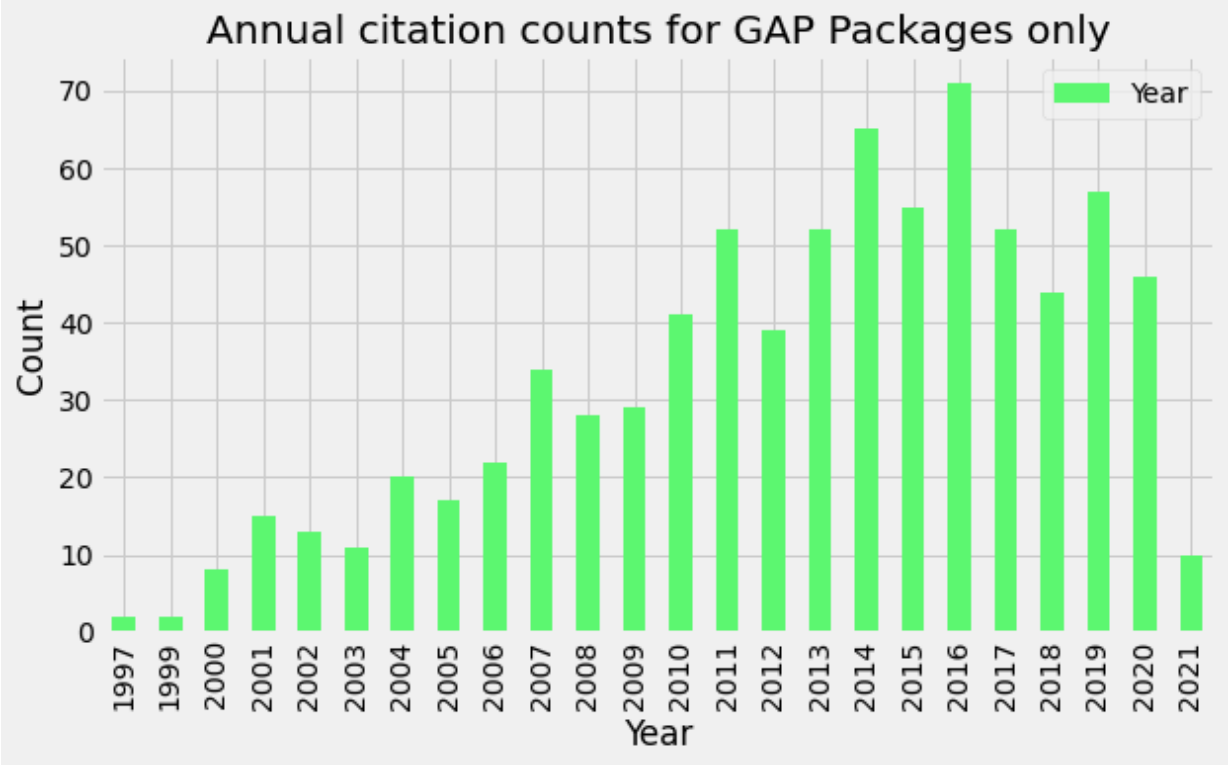
We can also plot it in a pie chart, which will allow viewers to see the percentage ratios.

In [113]:
```
labels = ['Packages', 'GAP software']
data = [pac, full]
fig = plt.figure(figsize =(5, 7))
plt.pie(data, labels = labels, explode = (0.1, 0.1), autopct='%1.0f%%')
plt.show()
```



Below we will plot the annual citation count for GAP Packages only, by reusing the code compiled for the full data and applying it on the `pac_df` subset. Numbers of Packages citations follow the same tendency as pure GAP citations, which is to increase rapidly starting from 2000. We can see that GAP packages start being cited since 1997 compared to 1992 for the main GAP software, which indicates Packages probably did not exist until 1997.

In [114]:
```
pac_year = pac_df['Year'].value_counts()
chrono_df = pac_year.to_frame()
chrono_pac = chrono_df.sort_index()
chrono_pac.plot(kind="bar", figsize=(9, 5), title='Annual citation counts for GAP
            xlabel='Year',
            ylabel='Count', color='#5cf770')
plt.yticks(np.arange(0,80, 10))
plt.show()
```



Package + GAP checker

The GAP website recommends to authors who cite GAP packages to also cite GAP in a separate reference. This is an excellent citation practice and we will try to find out how many authors did that.

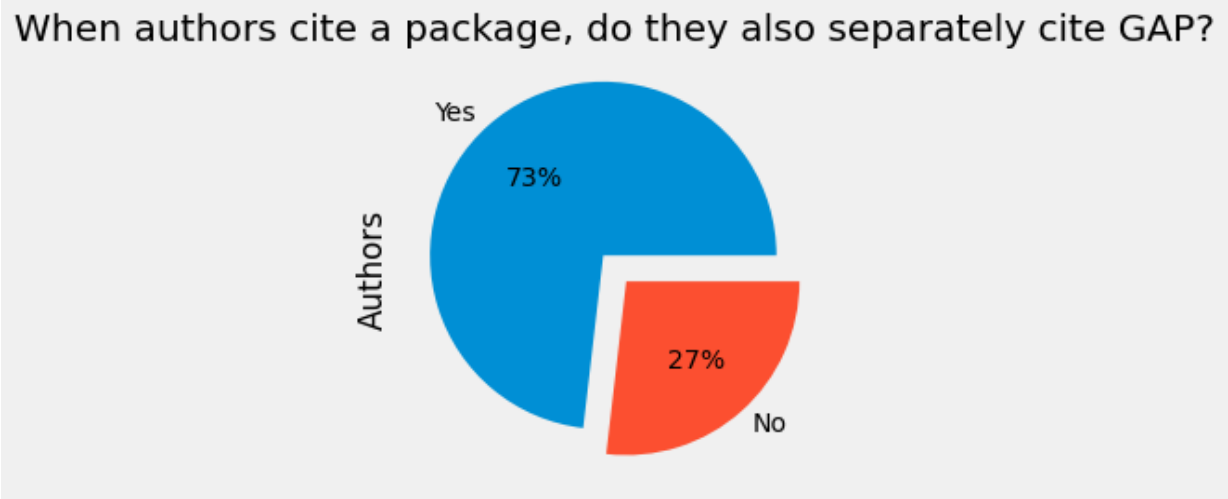
We have studied the dataframe and identified many cases when authors do cite GAP after they cited a package. This mostly occurs in the next record, but not always. Considering that we need to perform the check for rows having the same MR number, we can use the following custom function, and apply it to the `Version` column of our data grouped by `MR`.

In [115]:
```
def good_practice_check(s):
    s=set(s)
    if len(s.difference(['Package']))>0:
        if 'Package' in s:
            return 'Yes'
        return np.NaN
    return 'No'
```

In [116]:
```
filtered = merged_df.groupby('MR')['Version'].apply(good_practice_check)
filtered = filtered.dropna()
#filtered = pd.DataFrame(data=filtered)
filtered.columns=['Practice']
filtered.name='Authors'
filtered.value_counts()
```

Out[116]:
```
Yes    466
No     171
Name: Authors, dtype: int64
```

In [117]:
```
filtered.value_counts().plot.pie(
    title='When authors cite a package, do they also separately cite GAP?',
    explode = (0.1, 0.1), autopct='%1.0f%%')

plt.show()
```



To summarize from 636 publications (unique MR numbers in our dataset) citing GAP Packages, 465 or 73% cite GAP software separately after citing the package and 171 or 27% cite only GAP Package without citing GAP in a separate reference. It is great to see that the larger portion of authors did follow GAP guidelines and cited in two separate references.

Country

To justify that our selection is being representative: We use MathSciNet which indices top level journals worldwide (https://www.ams.org/journals/ (https://www.ams.org/journals/)) and is internationally known and respected. To the best of our knowledge, there are no biases in not indexing all parts of the world, or any other issues, and we fully trust the accuracy, sufficiency and credibility of the data we retrieved from MathSciNet.

Countries data was kindly provided by Dr Alexander Konovalov, who is part of the GAP team. The countries file provided is in `.json` format, therefore we need to load it first and then have a first look at it, using the `pprint` library.

In [118]:
```
import json
from pprint import pprint

with open('countries.json') as data:
    con = json.load(data)

pprint(con)
```

```
{'countries': {'1085773': ['Germany', 'Germany', 'Germany'],
               '1089065': ['Germany', 'Germany'],
               '1112157': ['Germany', 'Germany'],
               '1128656': ['Australia'],
               '1136226': ['England'],
               '1176516': ['Australia'],
               '1176715': ['Germany', 'Germany'],
               '1179271': ['Germany'],
               '1185021': ['Germany', 'Germany'],
               '1195429': ['Germany'],
               '1203878': ['England', 'England'],
               '1213836': ['Germany'],
               '1215552': ['The Netherlands', '48109', 'The Netherlands'],
               '1228206': ['France'],
               '1231011': ['New Zealand'],
               '1235810': ['England'],
               '1239007': ['Australia', 'Australia'],
               '1241915': ['England'],
               '1243835': ['Germany'],
```

What we have is a dictionary with only one key and one, extremely large value where all the tupples MR-countries are located. This large value is what we need to proceed. We will access it using the dictionary method `.values()` then we will convert it to a list in order to exrtract it. Finally we will convert the resulting list back to a dictionary, which this time has equally balanced keys and values.

In [119]:
```
con_data = con.values() # we access the data we need
values_list = list(con_data) # we convert it to list so we can extract it from th
countries = dict(values_list[0]) # we convert it back to python dictionary format
countries
```

Out[119]:
```
{'4056124': ['Slovakia', 'South Africa'],
 '3942387': ['France', 'France', 'Spain'],
 '3354065': ['Iran', 'Iran'],
 '3646312': ['Iran', 'Iran', 'Germany'],
 '1864795': ['Iran'],
 '2287843': ['Iran', 'Iran', 'Iran'],
 '2175389': ['Iran', 'Iran', 'Iran', 'Iran'],
 '2149067': ['Iran', 'Iran'],
 '2293309': ['Iran', 'Iran'],
 '2330635': ['Iran', 'Iran', 'Iran'],
 '2458407': ['Iran', 'Iran', 'Iran'],
 '2416600': ['Iran'],
 '2371966': ['Iran'],
 '2650379': ['Iran', 'Iran'],
 '2671230': ['Iran', 'Iran', 'Scotland', 'New Zealand'],
 '2387528': ['Iran', 'Iran', 'Iran'],
 '2424280': ['Iran', 'Iran', 'Iran'],
 '2514858': ['Iran', 'Iran'],
 '2546325': ['Iran', 'Iran'],
```

Our new dictionary has the MR numberers as keys and as values - a list of all countries corresponding to the authors for each MR number. Now we need to add the countries data to our existing dataframe. Before we do that we will convert the MR column to string so its data-type matches our dictionary. Then we map it storing the result in the new coulmn `Countries` .

In [120]:
```
merged_df['MR'] = merged_df['MR'].astype(str)
```

In [121]:
```
merged_df['Countries'] = merged_df['MR'].map(countries)
```

The next step is to break the lists of countries so we have one country in each cell. We will use the already familiar `.explode` method on our dataframe. Once done this will help us to further process this new column, because we have two issues still in it:

- all the entries where country should be US are numbers
- there are some null values, where our MR number was not present in the dictionary, or the value corresponding to our MR number was an empty list

In [122]:
```
countries_bloom = merged_df.explode('Countries')
```

In [123]:
```
nulaa = pd.isnull(countries_bloom['Countries'])
nulaa.value_counts()
#countries_bloom[countries_bloom['Countries'] == '']
```

Out[123]:
```
False    7166
True       61
Name: Countries, dtype: int64
```

In [124]:
```
countries_bloom[countries_bloom['Countries'].isnull()]
```

Out[124]:

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Website
54	1416724	Adler, A.	Proc. London Math. Soc. (3)	1997	article	14G35 (11F32 11G20 14H45 20B25 20C34)	M. Schönert et al, GAP: groups, algorithm and ...	Unknown	No
384	2608962	Barraclough, R. W.	LMS J. Comput. Math.	2010	article	20C15 (20-04 20C34)	The GAP Group. GAP —Groups, Algorithms, and Pro...	4.3	Yes
483	2171593	Bessis, D.	J. Math. Sci. (N.Y.)	2005	incollection	55Q20 (55-04 57M05)	D. Bessis and J. Michel, VKCURVE, software pac...	Package	Yes
488	1932707	Bhattacharyya, B.	Trans. Amer. Math. Soc.	2003	article	46L37 (46L55)	The GAP Group, Aachen, St. Andrews, GAP - Grou...	Unknown	Yes
518	3836249	Bishnoi, A. and De Bruyn, B.	J. Algebraic Combin.	2018	article	51E25 (05E30 51E12)	The GAP Group, GAP —Groups, Algorithms, and Pro...	4.7.5	Yes
...	...	...	...	...	...	...	...	...	...
3238	3003995	Ugolini, S.	J. Number Theory	2013	article	11G20	GAP Group, GAP - groups, algorithms and progra...	Unknown	Yes
3302	2372317	Wilde, T.	Comm. Algebra	2007	article	20C15	GAP (2005). The GAP Group, Groups, Algorithms ...	4.6	Yes
3354	4119478	Zhong, T. and Tan, Y.	Front. Math. China	2020	article	20D60 (20E34)	The GAP Group. GAP —Groups, Algorithms, and Pro...	4.10.0	Yes
3365	3864855	Amiri, M.	Internat. J. Algebra Comput.	2018	article	20D45 (20B05)	The Gap group, Gap-groups, algorithms, and pro...	4.4.10	Yes

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Website
3387	2081627	Dougherty, R. and Faber, V. and Murphy, M.	Discrete Comput. Geom.	2004	article	52B70	F. Lutz. Gap-program BISTELLAR. Available at h...	Unknown	Yes

61 rows × 14 columns

We will fill null values with a string 'No Country Data'.

In [125]:
```
countries_bloom.loc[(countries_bloom['Countries'].isnull(), 'Countries')] = 'No C
```

Then we replace all numeric cells with "United States" string because they are US postcodes corresponding to different states, but for not at least we will not need to filter our data by US stae, just the Country is enough.

In [126]:
```
countries_bloom.loc[(countries_bloom['Countries'].str.isnumeric(), 'Countries')]
```

In [127]:
```
countries_bloom[countries_bloom['Countries'] == 'United States']
```

Out[127]:

	MR	Author	Journal	Year	Publication Type	MSC	Citation	Version	Websit
46	3449941	Aboras, M. and Vojtěchovský, P.	Comm. Algebra	2016	article	20N05	The GAP Group, GAP - Groups, Algorithms, and P...	4.5.7	Ye
46	3449941	Aboras, M. and Vojtěchovský, P.	Comm. Algebra	2016	article	20N05	The GAP Group, GAP - Groups, Algorithms, and P...	4.5.7	Ye
47	3449941	Aboras, M. and Vojtěchovský, P.	Comm. Algebra	2016	article	20N05	Nagy, G. P., Vojtěchovský, P. LOOPS: Computing...	Package	Ye
47	3449941	Aboras, M. and Vojtěchovský, P.	Comm. Algebra	2016	article	20N05	Nagy, G. P., Vojtěchovský, P. LOOPS: Computing...	Package	Ye
51	2986074	Adan-Bante, E. and Harris, J. M.	Bol. Soc. Mat. Mexicana (3)	2011	article	20G40 (20E45)	T. G. Group, GAP - Groups, Algorithms, and Pro...	4.4.10	Ye
...	...	...	...	...	...	...	...	...	.
3409	3503387	Kiers, C. and O'Neill, C. and Ponomarenko, V.	Comm. Algebra	2016	article	20M14 (11A51)	Delgado, M., García-Sanchez, P., Morais, J. (2...	Package	Ye
3409	3503387	Kiers, C. and O'Neill, C. and Ponomarenko, V.	Comm. Algebra	2016	article	20M14 (11A51)	Delgado, M., García-Sanchez, P., Morais, J. (2...	Package	Ye
3413	3750929	Long, D. D. and Thistlethwaite, M. B.	Exp. Math.	2018	article	22E40 (20H10)	Groups, Algorithms, Programming – A System for...	Unknown	Ye
3413	3750929	Long, D. D. and Thistlethwaite, M. B.	Exp. Math.	2018	article	22E40 (20H10)	Groups, Algorithms, Programming – A System for...	Unknown	Ye
3428	2824780	Tian, Y. and Lu, C.	J. Ind. Manag. Optim.	2011	article	90C20 (90C11 90C46)	X. Sun, C. Liu, D. Li and J. Gao, On duality g...	Unknown	Ye

1155 rows × 14 columns

Maps

First of all, we need to prepare our data, so that our country names match exactly those in the Geographical `geojson` file, we will use to draw the borders on the choropleth map.

As we can see, after scrolling through the Geographical data, some country names in our citations dataframe are different from those in the `json` file with coordinates. We need them to be the same, so we will update our country names where necessary.

In [128]:
```
import json
from pprint import pprint

with open('world-countries.json') as data:
    geo = json.load(data)

pprint(geo)
```

```
{'features': [{'geometry': {'coordinates': [[[61.210817, 35.650072],
                                             [62.230651, 35.270664],
                                             [62.984662, 35.404041],
                                             [63.193538, 35.857166],
                                             [63.982896, 36.007957],
                                             [64.546479, 36.312073],
                                             [64.746105, 37.111818],
                                             [65.588948, 37.305217],
                                             [65.745631, 37.661164],
                                             [66.217385, 37.39379],
                                             [66.518607, 37.362784],
                                             [67.075782, 37.356144],
                                             [67.83, 37.144994],
                                             [68.135562, 37.023115],
                                             [68.859446, 37.344336],
                                             [69.196273, 37.151144],
                                             [69.518785, 37.608997],
                                             [70.116578, 37.588223],
                                             [70.270574, 37.735165],
                                             [70.376304, 38.138396],
```

In [129]:
```
countries_bloom.replace('United States', "United States of America", inplace = Tr
countries_bloom.replace('England', "United Kingdom", inplace = True)
countries_bloom.replace('Peoples Republic of China', "China", inplace = True)
countries_bloom.replace('Taiwan (R.O.C.)', "Taiwan", inplace = True)
countries_bloom.replace('Scotland', "United Kingdom", inplace = True)
countries_bloom.replace('The Netherlands', "Netherlands", inplace = True)
countries_bloom.replace('Wales', "United Kingdom", inplace = True)
countries_bloom.replace('Republic of Korea', "South Korea", inplace = True)
countries_bloom.replace('Serbia', "Republic of Serbia", inplace = True)
countries_bloom.replace('Socialist Republic of Vietnam', "Vietnam", inplace = Tru
countries_bloom.replace('Bosnia-Hercegovina', "Bosnia and Herzegovina", inplace =
```

Then where prepare the exact data we will map, which is the number of participations in GAP citing papers of authors from each country, as explained below. For this purpose we simply take the count of each unique value in the "Countries" column from the `exploded` data. Then we convert it to a pandas dataframe, add index and sort in descending order so we have a chart, name the columns and finally we export to `.csv` file which is then loaded in our choropleth map.

In [130]:
```
to_map = countries_bloom['Countries'].value_counts()
c_data = pd.DataFrame(data=to_map)
c_data = c_data.reset_index()
c_data.columns=['Country', 'Citations']
c_data.to_csv('citations_countries.csv', index=False, encoding='utf-8')
```

- It is very important to note that, because most papers citing GAP have multiple authors, we cannot assign a single country to one citation. Therefore, the table below represents the total count of each authors occurance in the data, converted to their country of origin. In other words the table is a result of the exploded data, which is broken down to a row for each country. For instance if a citation had 3 american authors it was broken into 3 lines having US as country each - resulting in +3 points for the US country count, not just one. The map further down is drawn from this table hence it also represents the same type of data.

  To summarise the citations column below represents the count of every time an author originating from this country participated in a paper citing the GAP software.

In [131]:
```
c_data.head(35)
```

Out[131]:

	Country	Citations
0	United States of America	1155
1	Germany	856
2	United Kingdom	802
3	Spain	452
4	Iran	384
5	Italy	379
6	Australia	282
7	China	255
8	Canada	186
9	Russia	176
10	Portugal	163
11	France	160
12	Belgium	147
13	Japan	139
14	Ireland	138
15	Hungary	128
16	Argentina	106
17	New Zealand	96
18	Brazil	96
19	Croatia	89
20	India	88
21	South Africa	82
22	Netherlands	70
23	Turkey	65
24	Mexico	63
25	No Country Data	61
26	Poland	60
27	Israel	59
28	Czech Republic	49
29	Austria	38
30	Slovenia	36
31	Romania	27
32	Switzerland	26
33	Taiwan	25

	Country	Citations
34	Belarus	24

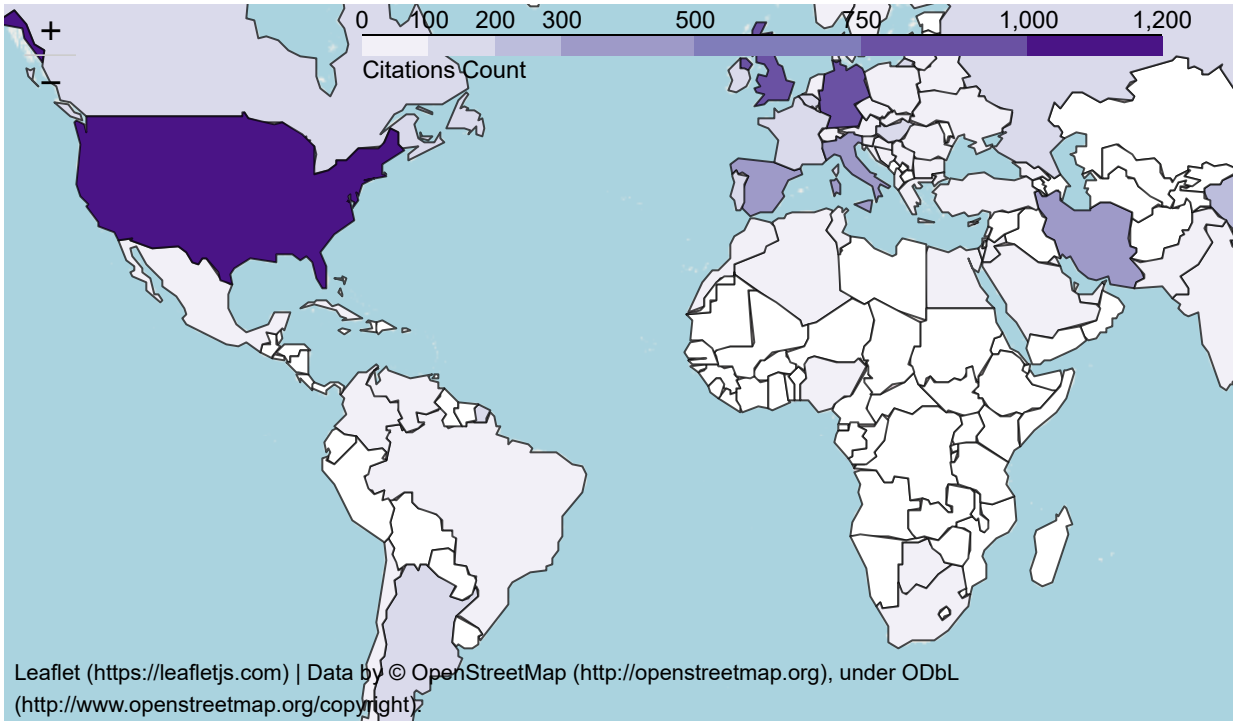
After inspecting the whole table, made of 71 rows, we discover that most entries have lower counts. For instance, all entries after 35th position are below 25, then everything between 25th and 17th is between 25 and 100, then between 17th and 4th all entries are ranging from 100 to ~450. Finally the top 3 are above 800.
Having noted all that, we need to partition our bins accordingly.
After many tests with different bins and colour schemes, we decided on the following bins `[0, 100, 200, 300, 500, 750, 1000, 1200]` and a single colour scheme `Purples`. The stronger the purple colour saturation is the higher number of citations we have for the country on the map.

GAP Citations Count Map

- I have placed comments along with the code, explaining each step in the process of creatign our map.

In [132]:
```
import pandas as pd
import os
import folium
import branca
from folium.features import GeoJsonPopup, GeoJsonTooltip
# we define our data which will dictate the choropleth colour scheme
country_count = pd.read_csv('citations_countries.csv')
# we add the geographical coordinates data
country_geo = os.path.join('world-countries.json')
# we define the base map - in our case the whole world
# here we also set the initial view, layout type and initial zoom level
citations_map = folium.Map(location=[0, 0], tiles="OpenStreetMap", zoom_start=2)

# tooltips
tooltip = GeoJsonTooltip(
    fields=["Country", "Citations"],
    aliases=["Country", "Citations"],
    localize=True,
    sticky=False,
    labels=True,
    style="""
        background-color: #F0EFEF;
        border: 2px solid black;
        border-radius: 3px;
        box-shadow: 3px;
    """,
    max_width=800,
)

# now we start building on top of the base
# in our case it will be one choropleth layer
choropleth = folium.Choropleth(
 geo_data=country_geo,
 name='choropleth',
 data=country_count,
 columns=['Country', 'Citations'], # columns to be used from our data
 key_on='feature.properties.name', # matching elements from the `geojson` data
 fill_color='Purples', # main colour scheme
 fill_opacity=1, # main colour scheme opacity
 line_opacity=0.7, # borders opacity
 legend_name="Citations Count",
    nan_fill_color="white", # missing values color
    nan_fill_opacity=1, # missing values opacity
    highlight=True, # `mouse on` highliht feature for each country
    bins=[0, 100, 200, 300, 500, 750, 1000, 1200], # legend bins
    tooltip = tooltip
).add_to(citations_map)



# layers filter, in case we add more layers later on
folium.LayerControl().add_to(citations_map)

# we load the map in the notebook
citations_map
```

Out[132]:



In [133]:
```
country_count["Citations"].quantile()
```

Out[133]:
```
23.0
```

GAP Citations Accuracy Score Map

For the next map, we will make another subset of our data, having only two columns - Country and its average accuracy score.
Once we have what we need, we export it to a `CSV` file, which is then loaded by our map drwaing library `folium` and used to colour different countries accordingly.

In [134]:
```
acc = countries_bloom.groupby(['Countries'])['Accuracy Score'].mean()
acc = pd.DataFrame(data=acc)
acc = acc.sort_values(by='Accuracy Score', ascending=False)
acc = acc.reset_index()
acc.to_csv('accuracy_countries.csv', index=False, encoding='utf-8')
acc
```

Out[134]:

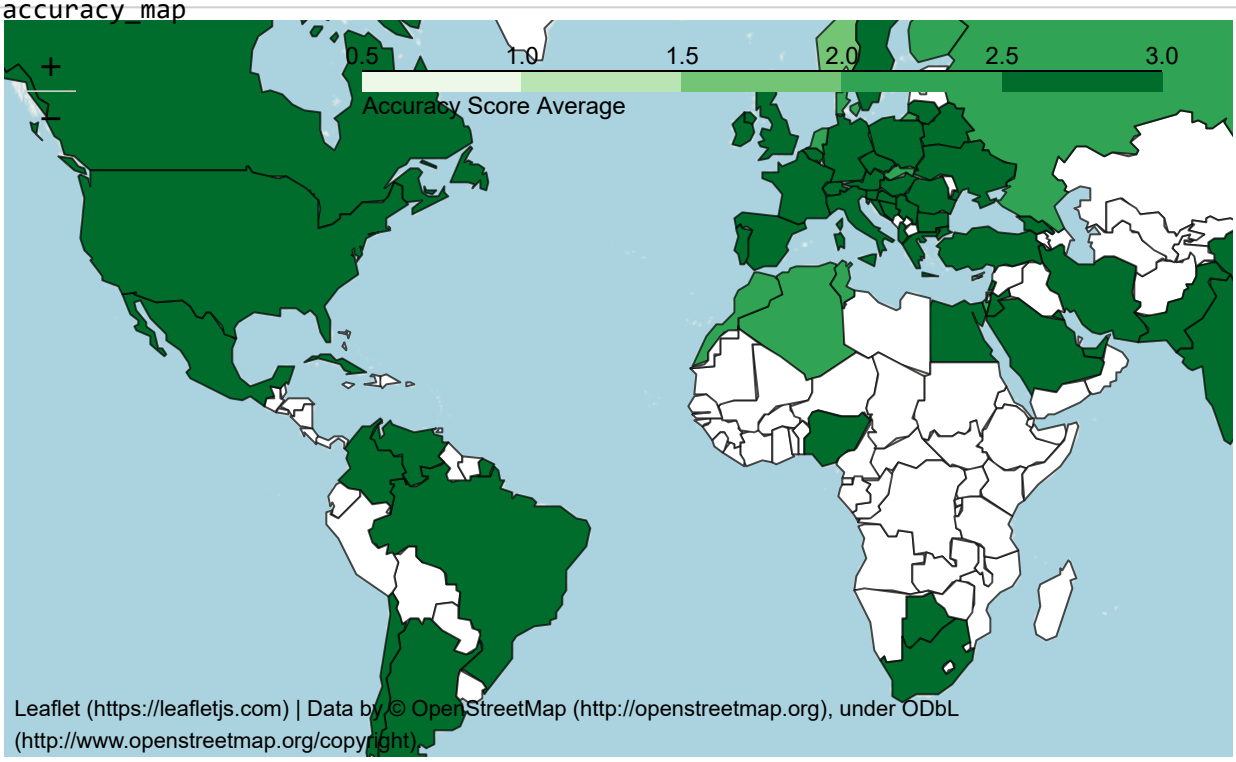
	Countries	Accuracy Score
0	Albania	3.00
1	Egypt	3.00
2	Taiwan	3.00
3	Romania	3.00
4	Republic of Serbia	3.00
...	...	...
66	Denmark	2.20
67	Slovakia	2.13
68	Tunisia	2.00
69	Cyprus	1.86
70	Norway	1.67

71 rows × 2 columns

In [135]:
```
# we define our data which will dictate the choropleth colour scheme
accuracy_mean = pd.read_csv('accuracy_countries.csv')
# we add the geographical coordinates data
country_geo = os.path.join('world-countries.json')
# we define the base map - in our case the whole world
# here we also set the initial view, layout type and initial zoom level
accuracy_map = folium.Map(location=[0, 0], tiles="OpenStreetMap", zoom_start=2)

tooltip = GeoJsonTooltip(
    fields=["Countries", "Accuracy Score"],
    aliases=["Countries", "Accuracy Score"],
    localize=True,
    sticky=False,
    labels=True,
    style="""
        background-color: #F0EFEF;
        border: 2px solid black;
        border-radius: 3px;
        box-shadow: 3px;
    """,
    max_width=800,
)

popup = GeoJsonPopup(
    fields=["Countries", "Accuracy Score"],
    aliases=["Countries", "Accuracy Score"],
    localize=True,
    labels=True,
    style="background-color: yellow;",
)

# now we start building on top of the base
# in our case it will be one choropleth layer
choropleth = folium.Choropleth(
    geo_data=country_geo,
    name='choropleth1',
    data=accuracy_mean,
    columns=['Countries', 'Accuracy Score'], # columns to be used from our data
    key_on='feature.properties.name', # matching elements from the `geojson` data
    fill_color='Greens', # main colour scheme
    fill_opacity=1, # main colour scheme opacity
    line_opacity=0.7, # borders opacity
    legend_name="Accuracy Score Average",
    nan_fill_color="white", # missing values color
    nan_fill_opacity=1, # missing values opacity
    highlight=True, # `mouse on` highliht feature for each country
    bins=[0.5, 1, 1.5, 2, 2.5, 3], # legend bins
    tooltip=tooltip,
    popup=popup
).add_to(accuracy_map)



# layers filter, in case we add more layers later on
folium.LayerControl().add_to(accuracy_map)
```

```
# we load the map in the notebook (we can also save it as HTML file)
accuracy_map
```

Out[135]:



Average GAP Citations Length Map

Similar to the precious map, we create a dataframe, with two columns - Country and its average Lenght.
Then we use it as data input to draw our third map.

In [136]:
```
length_av = countries_bloom.groupby(['Countries'])['Length'].mean()
length_av = pd.DataFrame(data=length_av)
length_av = length_av.sort_values(by='Length', ascending=False)
length_av = length_av.reset_index()
length_av.to_csv('length_countries.csv', index=False, encoding='utf-8')
length_av.head()
```

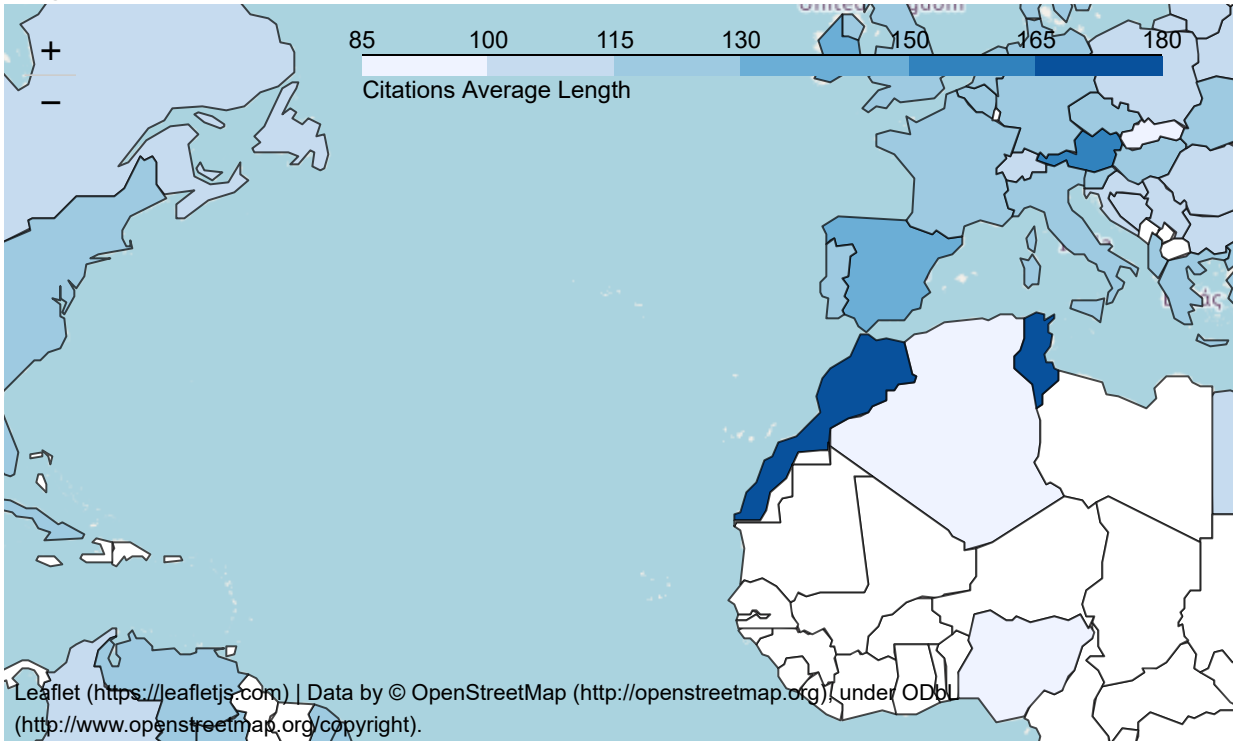
Out[136]:

	Countries	Length
0	Morocco	179.67
1	Tunisia	177.00
2	Cyprus	176.71
3	Pakistan	158.29
4	Austria	157.76

In [137]:
```
# we define our data which will dictate the choropleth colour scheme
accuracy_mean = pd.read_csv('length_countries.csv')
# we add the geographical coordinates data
country_geo = os.path.join('world-countries.json')
# we define the base map - in our case the whole world
# here we also set the initial view, layout type and initial zoom level
length_map = folium.Map(location=[0, 0], tiles="OpenStreetMap", zoom_start=2)

tooltip = GeoJsonTooltip(
    fields=["Countries", "Length"],
    aliases=["Countries", "Length"],
    localize=True,
    sticky=False,
    labels=True,
    style="""
        background-color: #F0EFEF;
        border: 2px solid black;
        border-radius: 3px;
        box-shadow: 3px;
    """,
    max_width=800,
)

popup = GeoJsonPopup(
    fields=["Countries", "Length"],
    aliases=["Countries", "Length"],
    localize=True,
    labels=True,
    style="background-color: yellow;",
)

# now we start building on top of the base
# in our case it will be one choropleth layer
choropleth = folium.Choropleth(
    geo_data=country_geo,
    name='choropleth1',
    data=accuracy_mean,
    columns=['Countries', 'Length'], # columns to be used from our data
    key_on='feature.properties.name', # matching elements from the `geojson` data
    fill_color='Blues', # main colour scheme
    fill_opacity=1, # main colour scheme opacity
    line_opacity=0.7, # borders opacity
    legend_name="Citations Average Length",
    nan_fill_color="white", # missing values color
    nan_fill_opacity=1, # missing values opacity
    highlight=True, # `mouse on` highliht feature for each country
    bins=[85, 100, 115, 130, 150, 165, 180], # legend bins
    tooltip=tooltip,
    popup=popup
).add_to(length_map)



# layers filter, in case we add more layers later on
folium.LayerControl().add_to(length_map)
```

```
# we load the map in the notebook (we can also save it as HTML file)
length_map
```

Out[137]:



In [138]:

```
citations_map.save("Number of GAP Citations Map.html")
accuracy_map.save("GAP Citations average accuracy Map.html")
length_map.save("GAP Citations average length Map.html")
```