Creating a Patent Infographic with R

In this chapter we will use RStudio to prepare patent data for visualisation in an infographic using online software tools.

Infographics are a popular way of presenting data in a way that is easy for a reader to understand without reading a long report. Infographics are well suited to presenting summaries of data with simple messages about key findings. A good infographic can encourage the audience to read a detailed report and is a tool for engagement with audiences during presentations of the findings of patent research.

Some patent offices have already been creating infographics as part of their reports to policy makers and other clients. The Instituto Nacional de Propiedade Industrial (INPI) in Brazil produces regular two page Technology Radar[[1]](#footnote-1) (Radar Tecnologico) consisting of charts and maps that briefly summarise more detailed research on subjects such as Nanotechnology in Waste Management[[2]](#footnote-2). WIPO Patent Landscape Reports[[3]](#footnote-3), which go into depth on patent activity for a particular area, are accompanied by one page infographics that have proved very popular such as the infographic accompanying a recent report on assistive devices[[4]](#footnote-4).

A growing number of companies are offering online infographic software services such as infogr.am[[5]](#footnote-5), easel.ly[[6]](#footnote-6) piktochart.com[[7]](#footnote-7), canva.com[[8]](#footnote-8) or venngage.com[[9]](#footnote-9) to mention only a selection of the offerings out there. The Cool Infographics website[[10]](#footnote-10) provides a useful overview of available tools.

One feature of many of these services is that they are based on a freemium model. Creating graphics is free but the ability to export files and the available formats for export of your masterpiece (e.g. high resolution or .pdf) often depend on upgrading to a monthly account at varying prices. In this chapter we test drive infogr.am[[11]](#footnote-11) as a chart friendly service, albeit with export options that depend on a paid account.

This chapter is divided into two sections.

1. In part 1 we focus on using RStudio to prepare patent data for visualisation in infographics software using the dplyr, tidyr and stringr packages. This involves dealing with common problems with patent data such as concatenated fields, white space and creating counts of data fields.
2. In part 2 we produce an infographic from the data using infogr.am[[12]](#footnote-12).

## Getting Started

To start with we need to ensure that RStudio and R for your operating system are installed by following the instructions on the RStudio website here[[13]](#footnote-13). Do not forget to follow the link to also install R for your operating system[[14]](#footnote-14).

When working in RStudio it is good practice to work with projects. This will keep all of the files for a project in the same folder. To create a project go to File, New Project and create a project. Call the project something like infographic. Any file you create and save for the project will now be listed under the Files tab in RStudio.

R works using packages (libraries) and there are around 7,490 of them for a whole range of purposes. We will use just a few of them. To install a package we use the following. Copy and paste the code into the Console and press enter.

install.packages("readr") # read in .csv files `readxl` for excel files  
install.packages("dplyr") # wrangle data  
install.packages("tidyr") # tidy data  
install.packages("stringr") # work with text strings  
install.packages("ggplot2") # for graphing

Packages can also be installed by selecting the Packages tab and typing the name of the package.

To load the package (library) use the following or check the tick box in the Packages pane.

library(readr)  
library(dplyr)  
library(tidyr)  
library(stringr)  
library(ggplot2)

We are now ready to go.

## Load a .csv file using readr

We will work with the pizza\_medium\_clean dataset in the online Github Manual repository[[15]](#footnote-15). If manually downloading a file remember to click on the file name and select Raw to download the actual file.

We can use the easy to use read\_csv() function from the readr package to quickly read in our pizza data directly from the Github repository. Note the raw at the beginning of the filename.

pizza <- read\_csv("https://raw.githubusercontent.com/poldham/opensource-patent-analytics/master/2\_datasets/pizza\_medium\_clean/pizza.csv")

readr will display a warning for the file arising from its efforts to parse publication dates on import. We will ignore this as we will not be using this field.

As an alternative to importing directly from Github download the file and enter the path in quotes (you must use the full path, e.g. C: etc.). For additional arguments (controls) look up the help for the function using ?read\_csv.

pizza\_read <- read\_csv("yourfilepath")

readr and readxl (for Excel files) are quite new. For more complex data see the Manual online articles on reading csv files in R[[16]](#footnote-16) and read excel files in R`[[17]](#footnote-17) packages for importing Excel.

## Viewing Data

We can view data in a variety of ways.

1. In the console:

pizza

## Source: local data frame [9,996 x 31]  
##   
## applicants\_cleaned  
## (chr)  
## 1 NA  
## 2 Ventimeglia Jamie Joseph; Ventimeglia Joel Michael; Ventimeglia Thomas Jose  
## 3 Cordova Robert; Martinez Eduardo  
## 4 Lazarillo De Tormes S L  
## 5 NA  
## 6 Depoortere, Thomas  
## 7 Frisco Findus Ag  
## 8 Bicycle Tools Incorporated  
## 9 Castiglioni, Carlo  
## 10 NA  
## .. ...  
## Variables not shown: applicants\_cleaned\_type (chr),  
## applicants\_organisations (chr), applicants\_original (chr),  
## inventors\_cleaned (chr), inventors\_original (chr), ipc\_class (chr),  
## ipc\_codes (chr), ipc\_names (chr), ipc\_original (chr), ipc\_subclass\_codes  
## (chr), ipc\_subclass\_detail (chr), ipc\_subclass\_names (chr),  
## priority\_country\_code (chr), priority\_country\_code\_names (chr),  
## priority\_data\_original (chr), priority\_date (chr),  
## publication\_country\_code (chr), publication\_country\_name (chr),  
## publication\_date (date), publication\_date\_original (chr),  
## publication\_day (int), publication\_month (int), publication\_number  
## (chr), publication\_number\_espacenet\_links (chr), publication\_year (int),  
## title\_cleaned (chr), title\_nlp\_cleaned (chr),  
## title\_nlp\_multiword\_phrases (chr), title\_nlp\_raw (chr), title\_original  
## (chr)

1. In Environment click on the blue arrow to see in the environment. Keep clicking to open a new window with the data.
2. Use the View() command (for data.frames and tables)

View(pizza)

If possible use the View() command or environment. The difficulty with the console is that large amounts of data will simply stream past.

## Identifying Types of Object

We often want to know what type of object we are working with and more details about the object so we know what to do later. Here are some of the most common commands for obtaining information about objects.

class(pizza) ## type of object  
names(pizza) ## names of variables  
str(pizza) ## structure of object  
dim(pizza) ## dimensions of the object

The most useful command in this list is str() because this allows us to access the structure of the object and see its type. str() is particularly useful because we can see the names of the fields (vectors) and their type. Most patent data is a character vector with dates forming integers.

## Working with Data

We will often want to select aspects of our data to focus on a specific set of columns or to create a graph. We might also want to add information, notably numeric counts.

The dplyr package provides a set of very handy functions for selecting, adding and counting data. The tidyr and stringr packages are sister packages that contain a range of other useful functions for working with our data. We have covered some of these in other chapters on graphing using R but will go through them quickly and then pull them together into a function that we can use across our dataset.

### Select

In this case we will start by using the select() function to limit the data to specific columns. We can do this using their names or their numeric position (best for large number of columns e.g. 1:31). In dplyr, unlike most R packages, existing character columns do not require "".

pizza\_number <- select(pizza, publication\_number, publication\_year)

We now have a new data.frame that contains two columns. One with the year and one with the publication number. Note that we have created a new object called pizza\_number using <- and that after select() we have named our original data and the columns we want. A fundamental feature of select is that it will drop columns that we do not name. So it is best to create a new object using <- if you want to keep your original data for later work.

### Adding data with mutate()

mutate() is a dplyr function that allows us to add data based on existing data in our data frame, for example to perform a calculation. In the case of patent data we normally lack a numeric field to use for counts. We can however assign a value to our publication field by using sum() and the number 1 as follows.

pizza\_number <- mutate(pizza\_number, n = sum(publication\_number = 1))

When we view pizza\_number we now have a value of 1 in the column n for each publication number. Note that in patent data a priority, application, publication or family number may occur multiple times and we would want to reduce the dataset to distinct records. For that we would use n\_distinct(pizza\_number$publication\_number) from dplyr or unique(pizza\_number$publication\_number) from base R. Because the publication numbers are unique we can proceed.

### Counting data using count()

At the moment, we have multiple instances of the same year (where a patent publication occurs in that year). We now want to calculate how many of our documents were published in each year. To do that we will use the dplyr function count(). We will use the publication\_year and add wt = (for weight) with n as the value to count.

pizza\_total <- count(pizza\_number, publication\_year, wt = n)  
pizza\_total

## Source: local data frame [58 x 2]  
##   
## publication\_year n  
## (int) (dbl)  
## 1 1940 1  
## 2 1954 1  
## 3 1956 1  
## 4 1957 1  
## 5 1959 1  
## 6 1962 1  
## 7 1964 2  
## 8 1966 1  
## 9 1967 1  
## 10 1968 8  
## .. ... ...

When we now examine pizza\_total, we will see the publication year and a summed value for the records in that year.

This raises the question of how we know that R has calculated the count correctly. We already know that there are 9996 records in the pizza dataset. To check our count is correct we can simply use sum and select the column we want to sum using $.

sum(pizza\_total$n)

## [1] 9996

So, all is good and we can move on. The $ sign is one of the main ways of subsetting to tell R that we want to work with a specific column (the others are "[" and "[[").

### Renaming a field using rename()

Next we will use rename() from dplyr to rename the fields. Note that understanding which field require quote marks can take some effort. In this case renaming the character vector publication\_year as "pubyear" requires quotes while renaming the numeric vector "n" does not.

pizza\_total <- rename(pizza\_total, pubyear = publication\_year, publications = n)

### Make a quickplot with qplot()

Using the qplot() function in ggplot2 we can now draw a quick line graph. Note that qplot() is unusual in R because the data (pizza\_total) appears after the coordinates. We will specify that we want a line using geom = (if geom is left out it will be a scatter plot). This will give us an idea of what our plot might look like in our infographic and actions we might want to take on the data.

qplot(x = pubyear, y = publications, data = pizza\_total, geom = "line")

The plot reveals a data cliff in recent years. This normally reflects a lack of data for the last 2-3 years as recent documents feed through the system en route to publication.

It is a good idea to remove the data cliff by cutting the data 2-3 years prior to the present. In some cases two years is sufficient, but 3 years is a good rule of thumb.

We also have long tail of data with limited data from 1940 until the late 1970s. Depending on our purposes with the analysis we might want to keep this data (for historical analysis) or to focus in on a more recent period.

We will limit our data to specific values using the dplyr function filter().

For more details on graphing in R see the online qplot[[18]](#footnote-18) and gglot2[[19]](#footnote-19) articles of the manual.

### Filter data using filter()

In contrast with select() which works with columns, filter() in dplyr works with rows. In this case we need to filter on the values in the pubyear column. To remove the data prior to 1990 we will use the greater than or equal to operator >= on the pubyear column and we will use the less than or equal to <= operator on the values after 2012.

One strength of filter() in dplyr is that it is easy to filter on multiple values in the same expression (unlike the very similar filter function in base R). The use of filter() will also remove the 30 records where the year is recorded as NA (Not Available). We will write this file to disk using the simple write\_csv() from readr. To use write\_csv() we first name our data (pizza\_total) and then provide a file name with a .csv extension. In this case and other examples below we have used a descriptive file name bearing in mind that Windows systems have limitations on the length and type of characters that can be used in file names.

pizza\_total <- filter(pizza\_total, pubyear >= 1990, pubyear <= 2012)  
write\_csv(pizza\_total, "pizza\_total\_1990\_2012.csv")  
pizza\_total

## Source: local data frame [23 x 2]  
##   
## pubyear publications  
## (int) (dbl)  
## 1 1990 139  
## 2 1991 154  
## 3 1992 212  
## 4 1993 201  
## 5 1994 162  
## 6 1995 173  
## 7 1996 180  
## 8 1997 186  
## 9 1998 212  
## 10 1999 290  
## .. ... ...

When we print pizza\_total to the console we will see that the data now covers the period 1990-2012. When using filter() on values in this way it is important to remember to apply this filter to any subsequent operations on the data (such as applicants) so that it matches the same data period.

To see our .csv file we can head over to the Files tab and, assuming that we have created a project, the file will now appear in the list of project files. Clicking on the file name will display the raw unformatted data in RStudio.

### Simplify code using pipes %>%

So far we have handled the code one line at a time. But, one of the great strengths of using a programming language is that we can run multiple lines of code together. There are two basic ways that we can do this.

We will create a new temporary object df to demonstrate this.

1. The standard way

df <- select(pizza, publication\_number, publication\_year)  
df <- mutate(df, n = sum(publication\_number = 1))  
df <- count(df, publication\_year, wt = n)  
df <- rename(df, pubyear = publication\_year, publications = n)  
df <- filter(df, pubyear >= 1990, pubyear <= 2012)  
qplot(x = pubyear, y = publications, data = df, geom = "line")

The code we have just created is six lines long. If we select all of this code and run it in one go it will produce our graph.

One feature of this code is that each time we run a function on the object total we name it at the start of each function (e.g. mutate(total...)) and then we overwrite the object.

We can save quite a lot of typing and reduce the complexity of the code using the pipe operator introduced by the the magrittr package and then adopted in Hadley Wickham's data wrangling and tidying packages we are using.

1. Using pipes %>%

Pipes are now a very popular way of writing R code because they simplify writing R code and speed it up. The most popular pipe is %>% which means "this" then "that". In this case we are going to create a new temporary object df1 by first applying select to pizza, then mutate, count, rename and filter. Note that we only name our dataset once (in select()) and we do not need to keep overwriting the object.

df1 <- select(pizza, publication\_number, publication\_year) %>% mutate(n = sum(publication\_number = 1)) %>%   
 count(publication\_year, wt = n) %>% rename(pubyear = publication\_year, publications = n) %>%   
 filter(pubyear >= 1990, pubyear <= 2012) %>% qplot(x = pubyear, y = publications,   
 data = ., geom = "line") %>% print()

In the standard code we typed df nine times to arrive at the same result. Using pipes we typed df1 once. Of greater importance is that the use of pipes simplifies the structure of R code by introducing a basic "this" then "that" logic which makes it easier to understand.

One point to note about this code is that qplot() requires us to name our data (in this case df1). However, df1 is actually the final output of the code and does not exist as an input object before the final line is run. So, if we attempt to use data = df1 in qplot() we will receive an error message. The way around this is to use . in place of our data object. That way qplot() will know we want to graph the outputs of the earlier code. Finally, we need to add an explicit call to print() to display the graph (without this the code will work but we will not see the graph).

If we now inspect the structure of the df1 object (using str(df1)) in the console, it will be a list. The reason for this is that it is an object with mixed components, including a data.frame with our data plus additional data setting out the contents of the plot. As there is no direct link between R and our infographics software this will create problems for us later because the infographics software won't know how to interpret the list object. So, it is generally a good idea to use a straight data.frame.

## Harmonising data

One challenge with creating multiple tables from a baseline dataset is keeping track of subdatasets. At the moment we have two basic objects we will be working with:

1. pizza - our raw dataset
2. pizza\_total - created via pizza\_number limited to 1990\_2012.

In the remainder of the chapter we will want to create some additional datasets from our pizza dataset. These are:

1. Country trends
2. Applicants
3. International Patent Classification (IPC) Class
4. Phrases
5. Google
6. Google IPC
7. Google phrases

We need to make sure that any data that we generate from our raw dataset matches the period for the pizza\_total dataset. If we do not do this there is a risk that we will generate subdatasets with counts for the raw pizza dataset.

To handle this we will use filter() to create a new baseline dataset with an unambiguous name.

pizza\_1990\_2012 <- rename(pizza, pubyear = publication\_year) %>% filter(pubyear >=   
 1990, pubyear <= 2012) %>% print()

## Source: local data frame [8,262 x 31]  
##   
## applicants\_cleaned applicants\_cleaned\_type  
## (chr) (chr)  
## 1 NA People  
## 2 Lazarillo De Tormes S L Corporate  
## 3 NA People  
## 4 Depoortere, Thomas People  
## 5 Frisco Findus Ag Corporate  
## 6 Bicycle Tools Incorporated Corporate  
## 7 Castiglioni, Carlo People  
## 8 NA People  
## 9 Bujalski, Wlodzimierz People  
## 10 Ehrno Flexible A/S; Stergaard, Ole Corporate; People  
## .. ... ...  
## Variables not shown: applicants\_organisations (chr), applicants\_original  
## (chr), inventors\_cleaned (chr), inventors\_original (chr), ipc\_class  
## (chr), ipc\_codes (chr), ipc\_names (chr), ipc\_original (chr),  
## ipc\_subclass\_codes (chr), ipc\_subclass\_detail (chr), ipc\_subclass\_names  
## (chr), priority\_country\_code (chr), priority\_country\_code\_names (chr),  
## priority\_data\_original (chr), priority\_date (chr),  
## publication\_country\_code (chr), publication\_country\_name (chr),  
## publication\_date (date), publication\_date\_original (chr),  
## publication\_day (int), publication\_month (int), publication\_number  
## (chr), publication\_number\_espacenet\_links (chr), pubyear (int),  
## title\_cleaned (chr), title\_nlp\_cleaned (chr),  
## title\_nlp\_multiword\_phrases (chr), title\_nlp\_raw (chr), title\_original  
## (chr)

In this case we start with a call to rename() to make this consistent with our pizza\_total table and then use a pipe to filter the data on the year. Note here that when filtering raw data on a set of values it is important to inspect it first to check that the field is clean (e.g. not concatenated).

We are now in a position to create our country trends table.

## Country Trends using spread()

There are two basic data formats: long and wide. Our pizza dataset is in long format because each column is a variable (e.g. publication\_country) and each row in publication\_country contains a country name. This is the most common and useful data format.

However, in some cases, such as infogr.am our visualisation software will expect the data to be in wide format. In this case each country name would become a variable (column name) with the years forming the rows and the number of records per year the observations. The key to this is the tidyr() function spread().

As above we will start off by using select() to create a table with the fields that we want. We will then use mutate() to add a numeric field and then count up that data. To illustrate the process run this code (we will not create an object).

select(pizza\_1990\_2012, publication\_country\_name, publication\_number, pubyear) %>%   
 mutate(n = sum(publication\_number = 1)) %>% count(publication\_country\_name,   
 pubyear, wt = n) %>% print()

## Source: local data frame [223 x 3]  
## Groups: publication\_country\_name [?]  
##   
## publication\_country\_name pubyear n  
## (chr) (int) (dbl)  
## 1 Canada 1990 19  
## 2 Canada 1991 49  
## 3 Canada 1992 66  
## 4 Canada 1993 59  
## 5 Canada 1994 50  
## 6 Canada 1995 39  
## 7 Canada 1996 36  
## 8 Canada 1997 45  
## 9 Canada 1998 46  
## 10 Canada 1999 47  
## .. ... ... ...

When we run this code we will see the results in long format. We now want to take our publication\_country\_name column and spread it to form columns with n as the values.

In using spread note that it takes a data argument (pizza\_1990\_2012), a key (publication\_country\_name), and value column (n). We are using pipes so the data only needs to be named in the first line. For additional arguments see ?spread().

country\_totals <- select(pizza\_1990\_2012, publication\_country\_name, publication\_number,   
 pubyear) %>% mutate(n = sum(publication\_number = 1)) %>% count(publication\_country\_name,   
 pubyear, wt = n) %>% spread(publication\_country\_name, n) %>% print()

## Source: local data frame [23 x 17]  
##   
## pubyear Canada China Eurasian Patent Organization  
## (int) (dbl) (dbl) (dbl)  
## 1 1990 19 NA NA  
## 2 1991 49 NA NA  
## 3 1992 66 NA NA  
## 4 1993 59 NA NA  
## 5 1994 50 NA NA  
## 6 1995 39 NA NA  
## 7 1996 36 1 NA  
## 8 1997 45 NA NA  
## 9 1998 46 NA NA  
## 10 1999 47 2 2  
## .. ... ... ... ...  
## Variables not shown: European Patent Office (dbl), Germany (dbl), Israel  
## (dbl), Japan (dbl), Korea, Republic of (dbl), Mexico (dbl), Patent  
## Co-operation Treaty (dbl), Portugal (dbl), Russian Federation (dbl),  
## Singapore (dbl), South Africa (dbl), Spain (dbl), United States of  
## America (dbl)

We now have data in wide format.

In some cases, such as infogr.am, visualisation software may expect the country names as rows and the year as column names. We can modify our call to spread() by replacing the publication\_country\_name with pubyear. Then we will write the data to disk for use in our infographic.

country\_totals <- select(pizza\_1990\_2012, publication\_country\_name, publication\_number,   
 pubyear) %>% mutate(n = sum(publication\_number = 1)) %>% count(publication\_country\_name,   
 pubyear, wt = n) %>% spread(pubyear, n) %>% print()

## Source: local data frame [16 x 24]  
## Groups: publication\_country\_name [16]  
##   
## publication\_country\_name 1990 1991 1992 1993 1994 1995 1996  
## (chr) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)  
## 1 Canada 19 49 66 59 50 39 36  
## 2 China NA NA NA NA NA NA 1  
## 3 Eurasian Patent Organization NA NA NA NA NA NA NA  
## 4 European Patent Office 22 29 36 29 26 29 27  
## 5 Germany 2 2 2 2 5 2 1  
## 6 Israel NA NA 1 NA NA 1 1  
## 7 Japan NA NA NA NA NA NA NA  
## 8 Korea, Republic of NA NA NA 1 NA NA 1  
## 9 Mexico NA NA NA NA NA NA NA  
## 10 Patent Co-operation Treaty 8 13 31 16 20 22 23  
## 11 Portugal NA NA NA NA NA NA NA  
## 12 Russian Federation NA NA NA NA NA NA NA  
## 13 Singapore NA NA NA NA NA NA NA  
## 14 South Africa 2 3 3 3 3 1 9  
## 15 Spain NA NA NA NA NA NA NA  
## 16 United States of America 86 58 73 91 58 79 81  
## Variables not shown: 1997 (dbl), 1998 (dbl), 1999 (dbl), 2000 (dbl), 2001  
## (dbl), 2002 (dbl), 2003 (dbl), 2004 (dbl), 2005 (dbl), 2006 (dbl), 2007  
## (dbl), 2008 (dbl), 2009 (dbl), 2010 (dbl), 2011 (dbl), 2012 (dbl)

write\_csv(country\_totals, "pizza\_country\_1990\_2012.csv")

To restore the data to long format we would need to use gather() as the counterpart to spread(). gather takes a dataset, a key for the name of the column we want to gather the countries into, a value for the numeric count (in this case n), and finally the positions of the columns to gather in. Note here that we need to look up the column positions in country\_totals (e.g. using View()) or count the columns using ncol(country\_totals).

gather(country\_totals, year, n, 2:24) %>% print()

## Source: local data frame [368 x 3]  
## Groups: publication\_country\_name [16]  
##   
## publication\_country\_name year n  
## (chr) (chr) (dbl)  
## 1 Canada 1990 19  
## 2 China 1990 NA  
## 3 Eurasian Patent Organization 1990 NA  
## 4 European Patent Office 1990 22  
## 5 Germany 1990 2  
## 6 Israel 1990 NA  
## 7 Japan 1990 NA  
## 8 Korea, Republic of 1990 NA  
## 9 Mexico 1990 NA  
## 10 Patent Co-operation Treaty 1990 8  
## .. ... ... ...

The combination of spread and gather work really well to prepare data in formats that are expected by other software. However, one of the main issues we encounter with patent data is that our data is not tidy because various fields are concatenated.

## Tidying data - Separating and Gathering

In patent data we often see concatenated fields with a separator (normally a ;). These are typically applicant names, inventor names, IPC codes, or document numbers (priority numbers, family numbers). We need to tidy this data prior to data cleaning (such as cleaning names) or to prepare for analysis and visualisation. For more on the concept of tidy data read Hadley Wickham's Tidy Data article[[20]](#footnote-20).

To tidy patent data we will typically need to do two things.

1. Separate the data so that each cell contains a unique data point (e.g. a name, code or publication number). This normally involves separating data into columns.
2. Gathering the data back in. This involves transforming the data in the columns we have created into rows.

Separating data into columns is very easy in tools such as Excel. However, gathering the data back in to separate rows is remarkably difficult. Happily, this is very easy to do in R with the tidyr package.

The tidyr package contains two functions that are very useful when working with patent data. The first of these is separate().

Here we will work with the applicants\_cleaned field in the pizza dataset. This field contains concatenated names with a ; as the separator. For example, on lines 1\_9 there are single applicant names or NA values. However, on lines 10 and line 59 we see:

Ehrno Flexible A/S; Stergaard, Ole  
Farrell Brian; Mcnulty John; Vishoot Lisa

The problem here is that when we are dealing with thousands of lines of applicant names we don't know how many names might be concatenated into each cell as a basis for separating the data into columns.

The first option we will try is to separate the applicants\_cleaned field into an arbitrarily high number of columns.

### Separate

In the first step we use the tidyr separate() function. Ideally the field we want to separate is the first column because we will be using the numeric positions of the columns. In the case of the pizza1 data the applicants\_cleaned field is already the first column. If we wanted to move a column to the first position we could use select() as follows. To illustrate this we will create two new temporary objects, p1 and p2.

p1 <- select(pizza, 2:31, 1) #moves column 1 to the end (column 31)  
p2 <- select(p1, 31, 1:30) #moves column 31 to the first column

If we print p1 to the console applicants\_cleaned will now appear as the final column. If we print p2 it will have been restored to the first column. Working with the numeric positions of columns can take a little while to get used to but in the first code chunk we are saying columns 2 to 31 and then column 1. In the second chunk we reverse this to say column 31 and then columns 1:30.

Next we use separate() on pizza\_1990\_2012. This is followed by the unquoted name of the column we want to separate and the number of columns we want to separate the applicants into (1:30). Here we have chosen an arbitrary 30 columns. We then specify the separator with the ;. The next two arguments are for what to do with any extra data and the direction to fill cells. We use fill = "right" because separate() will throw an error if the cells do not divide into the same number of pieces.

pizza1 <- separate(pizza\_1990\_2012, applicants\_cleaned, 1:30, sep = ";", extra = "merge",   
 fill = "right") %>% print()

## Source: local data frame [8,262 x 60]  
##   
## 1 2 3 4 5 6  
## (chr) (chr) (chr) (chr) (chr) (chr)  
## 1 NA NA NA NA NA NA  
## 2 Lazarillo De Tormes S L NA NA NA NA NA  
## 3 NA NA NA NA NA NA  
## 4 Depoortere, Thomas NA NA NA NA NA  
## 5 Frisco Findus Ag NA NA NA NA NA  
## 6 Bicycle Tools Incorporated NA NA NA NA NA  
## 7 Castiglioni, Carlo NA NA NA NA NA  
## 8 NA NA NA NA NA NA  
## 9 Bujalski, Wlodzimierz NA NA NA NA NA  
## 10 Ehrno Flexible A/S Stergaard, Ole NA NA NA NA  
## .. ... ... ... ... ... ...  
## Variables not shown: 7 (chr), 8 (chr), 9 (chr), 10 (chr), 11 (chr), 12  
## (chr), 13 (chr), 14 (chr), 15 (chr), 16 (chr), 17 (chr), 18 (chr), 19  
## (chr), 20 (chr), 21 (chr), 22 (chr), 23 (chr), 24 (chr), 25 (chr), 26  
## (chr), 27 (chr), 28 (chr), 29 (chr), 30 (chr), applicants\_cleaned\_type  
## (chr), applicants\_organisations (chr), applicants\_original (chr),  
## inventors\_cleaned (chr), inventors\_original (chr), ipc\_class (chr),  
## ipc\_codes (chr), ipc\_names (chr), ipc\_original (chr), ipc\_subclass\_codes  
## (chr), ipc\_subclass\_detail (chr), ipc\_subclass\_names (chr),  
## priority\_country\_code (chr), priority\_country\_code\_names (chr),  
## priority\_data\_original (chr), priority\_date (chr),  
## publication\_country\_code (chr), publication\_country\_name (chr),  
## publication\_date (date), publication\_date\_original (chr),  
## publication\_day (int), publication\_month (int), publication\_number  
## (chr), publication\_number\_espacenet\_links (chr), pubyear (int),  
## title\_cleaned (chr), title\_nlp\_cleaned (chr),  
## title\_nlp\_multiword\_phrases (chr), title\_nlp\_raw (chr), title\_original  
## (chr)

We now have a data frame with 8,262 rows and 60 columns. When pizza1 is printed we will now see a set of numeric columns with NA in most cells and the name Stergaard, Ole in row 10 of column 2.

Note that while this works well there can be some inconsistency where the underlying data has a semicolon in the name where it should be a ,. As a result some of the names will be incorrectly split. We will simply live with this for this exercise.

### Gathering

The second step is to use the tidyr() function gather(). This will gather the columns we specify into rows. gather() involves specifying a key value pair. We can introduce the key (a numeric value) if we don't have one by specifying a column name and gather will create it for us. In this case we use n. Then we specify the value - the column that we want to gather the names into - we will call that applicants. Then we specify the columns to gather by their numeric position. Finally, where there are NA (Not Available) values we specify na.rm = TRUE to remove them.

pizza1 <- gather(pizza1, n, applicants, 1:30, na.rm = TRUE) %>% print()

## Source: local data frame [12,133 x 32]  
##   
## applicants\_cleaned\_type applicants\_organisations  
## (chr) (chr)  
## 1 Corporate Lazarillo De Tormes S L  
## 2 People NA  
## 3 Corporate Frisco Findus Ag  
## 4 Corporate Bicycle Tools Incorporated  
## 5 People NA  
## 6 People NA  
## 7 Corporate; People Ehrno Flexible A/S  
## 8 People NA  
## 9 People NA  
## 10 People NA  
## .. ... ...  
## Variables not shown: applicants\_original (chr), inventors\_cleaned (chr),  
## inventors\_original (chr), ipc\_class (chr), ipc\_codes (chr), ipc\_names  
## (chr), ipc\_original (chr), ipc\_subclass\_codes (chr), ipc\_subclass\_detail  
## (chr), ipc\_subclass\_names (chr), priority\_country\_code (chr),  
## priority\_country\_code\_names (chr), priority\_data\_original (chr),  
## priority\_date (chr), publication\_country\_code (chr),  
## publication\_country\_name (chr), publication\_date (date),  
## publication\_date\_original (chr), publication\_day (int),  
## publication\_month (int), publication\_number (chr),  
## publication\_number\_espacenet\_links (chr), pubyear (int), title\_cleaned  
## (chr), title\_nlp\_cleaned (chr), title\_nlp\_multiword\_phrases (chr),  
## title\_nlp\_raw (chr), title\_original (chr), n (chr), applicants (chr)

We now have a data.frame with 12,133 rows and 32 columns. The reason for this is that gather has created new rows containing the individual applicant names. If we use View(pizza1) we will see that tidyr has created our applicants column at the end of the data.frame (column 32).

### Trimming with stringr

However, if we now inspect the bottom of the column by subsetting into it using $ we will see that a lot of the names have a leading whitespace space. This results from the separate exercise where the ; is actually ;space. Take a look at the last few rows of the data using tail().

tail(pizza1$applicants, 10)

## [1] " Ruengruglikit, Chada"   
## [2] " King"   
## [3] " Yahoo! Inc"   
## [4] " Rutgers, The State University Of New Jersey"  
## [5] " Lanette Marie"   
## [6] " Schaich, Karen"   
## [7] " Langdon"   
## [8] " Wellgen, Inc"   
## [9] " Shaffer Manufacturing Corp"   
## [10] " Susan Patricia"

We can address this problem using a function from the stringr package str\_trim(). Trimming whitespace is important because it affects how names will rank at a later stage. For example " Dibble, James W" will be treated as a separate name from "Dibble, James W". We have a choice with str\_trim() on whether to trim the whitespace on the right, left or both. Here we will choose both.

Because we are seeking to modify an existing column (not to create a new vector or data.frame) we will use $ to select the column and as the data for the str\_trim() function. That will apply the function to the applicants column in pizza1.

pizza1$applicants <- str\_trim(pizza1$applicants, side = "both")

We can tie the steps so far together using pipes into the following simpler code that we will put in the temporary object pizza4.

pizza2 <- rename(pizza, pubyear = publication\_year) %>% filter(pubyear >= 1990,   
 pubyear <= 2012) %>% separate(applicants\_cleaned, 1:30, sep = ";", extra = "merge",   
 fill = "right") %>% gather(n, applicants, 1:30, na.rm = TRUE)  
pizza2$applicants <- str\_trim(pizza2$applicants, side = "both")  
tail(pizza2$applicants)

## [1] "Lanette Marie" "Schaich, Karen"   
## [3] "Langdon" "Wellgen, Inc"   
## [5] "Shaffer Manufacturing Corp" "Susan Patricia"

Note that when using str\_trim() we use subsetting to modify the applicants column in place. There is possibly a more efficient way of doing this with pipes but this appears difficult because the data.frame needs to exist for the str\_trim() to act on in place or we end up with a vector of applicant names rather than a data.frame.

### Calculating the number of separators

We now have some working code that will separate out our names, gather it back in and then trim it. However, rather than using an arbitrary number in separate() to ensure accuracy it would be very helpful if we knew the maximum number of names that the applicants, inventors or IPC code fields breaks into in a given dataset. This is important because in some case the number of applicants or other concatenated data may exceed our arbitrary maximum.

The code below is a small function that starts by counting the number of separators (sep) in a column (col) using the str\_count() function from stringr. In this case some of the fields are NA. In R, where a vector contains NA values R will always return NA as the answer. So, we use na.omit() to remove NA from the calculation (note that we are using pipes so we name our data only once). Then we create a separate object that calculates the maximum value max(). We need to oblige R to do this as an integer by placing the max() function inside as.integer().

The final name in the concatenated applicants name will not possess a separator at the end. If we don't address this then our function will undercount the names. A simple way to accommodate this is to add +1 at the end of the calculation. Note that cells containing only 1 name will not be counted. However, with the exception of NAs the minimum value will always be 1 and we are seeking the maximum value, so this is fine.

To load this function simply select the text and copy it into your console.

field\_count <- function(data, col = "", sep = "[[:alnum:]]+") {  
 library(stringr)  
 library(dplyr)  
 field\_count <- str\_count(data[[col]], pattern = sep) %>% na.omit()  
 n <- as.integer(max(field\_count) + 1) %>% print()  
}

Head to the Environment tab and you should see it under Functions. Next run the code as follows. Note that we are running the count on our base dataset pizza.

n <- field\_count(pizza\_1990\_2012, "applicants\_cleaned", sep = ";")

## [1] 18

This tells us that the maximum number of actors in the applicants\_cleaned column in pizza\_1990\_2012 is 18. We can now rerun our original code and instead of using an arbitrary number we can use the value of n as 1:n. We will create and export an applicants table using this value.

### Creating an Applicants Table

We will now do some tidying up using select() and arrange().

Remember that the gather() function requires a key value pair. This introduced a column called n into the data. We now want to drop this as it is not always a meaningful count. We also want to move our new applicants column from the last column in the dataset to the first. To achieve this we will add a line to the code to move our applicants column to column 1 using select() and drop column n by not naming it (column 31). Finally, we will use arrange() to sort the applicants in alphabetical order.

We will save this in a new object called applicants.

applicants <- separate(pizza\_1990\_2012, applicants\_cleaned, 1:n, sep = ";",   
 extra = "merge", fill = "right") %>% gather(n, applicants, 1:n, na.rm = TRUE) %>%   
 select(32, 1:30) %>% arrange(applicants)  
applicants$applicants <- str\_trim(applicants$applicants, side = "both")  
applicants

## Source: local data frame [12,133 x 31]  
##   
## applicants applicants\_cleaned\_type  
## (chr) (chr)  
## 1 Ab Agri Limited Corporate  
## 2 Adler Scott A Corporate; People  
## 3 Ahopelto, Timo Corporate; People  
## 4 Aimee Corporate; People  
## 5 Ajmera Tejus People  
## 6 Ajmera Tejus People  
## 7 Ajmera Tejus Corporate; People  
## 8 Ajmera Tejus Corporate; People  
## 9 Albanese Mary Elizabeth Corporate; People  
## 10 Alfano, Vincenzo People  
## .. ... ...  
## Variables not shown: applicants\_organisations (chr), applicants\_original  
## (chr), inventors\_cleaned (chr), inventors\_original (chr), ipc\_class  
## (chr), ipc\_codes (chr), ipc\_names (chr), ipc\_original (chr),  
## ipc\_subclass\_codes (chr), ipc\_subclass\_detail (chr), ipc\_subclass\_names  
## (chr), priority\_country\_code (chr), priority\_country\_code\_names (chr),  
## priority\_data\_original (chr), priority\_date (chr),  
## publication\_country\_code (chr), publication\_country\_name (chr),  
## publication\_date (date), publication\_date\_original (chr),  
## publication\_day (int), publication\_month (int), publication\_number  
## (chr), publication\_number\_espacenet\_links (chr), pubyear (int),  
## title\_cleaned (chr), title\_nlp\_cleaned (chr),  
## title\_nlp\_multiword\_phrases (chr), title\_nlp\_raw (chr), title\_original  
## (chr)

We will want to create a plot with the applicants data in our infographic software. For that we need to introduce a field to count on. We might also want to establish a cut off point based on the number of records per applicant.

In this code we will simply print the applicants ranked in descending order. The second to last line of the code provides a filter on the number of records. This can be changed after inspecting the data.

library(tidyr)  
library(dplyr)  
applicant\_count <- select(applicants, applicants, publication\_number) %>%  
 mutate(n = sum(publication\_number = 1)) %>%  
 count(applicants, wt = n) %>%  
 arrange(desc(n)) %>%  
 filter(n >= 1) %>% #applicant records filter  
 print()

## Source: local data frame [6,178 x 2]  
##   
## applicants n  
## (chr) (dbl)  
## 1 Graphic Packaging International, Inc 154  
## 2 Kraft Foods Holdings, Inc 132  
## 3 Google Inc 123  
## 4 Microsoft Corporation 88  
## 5 The Pillsbury Company 83  
## 6 General Mills, Inc 77  
## 7 Nestec 77  
## 8 The Procter & Gamble Company 59  
## 9 Pizza Hut, Inc 57  
## 10 Yahoo! Inc 54  
## .. ... ...

If we inspect applicant count using View(applicant\_count) we have 6,178 rows. That is far too many to display in an infographic. So, next we will filter the data on the value for the top ten (64). Then we will write the data to a .csv file using the simple write\_csv() (as an alternative use write.csv() with the appropriate arguments).

applicant\_count <- select(applicants, applicants, publication\_number) %>% mutate(n = sum(publication\_number = 1)) %>%   
 count(applicants, wt = n) %>% arrange(desc(n)) %>% filter(n >= 64) %>% print()

## Source: local data frame [7 x 2]  
##   
## applicants n  
## (chr) (dbl)  
## 1 Graphic Packaging International, Inc 154  
## 2 Kraft Foods Holdings, Inc 132  
## 3 Google Inc 123  
## 4 Microsoft Corporation 88  
## 5 The Pillsbury Company 83  
## 6 General Mills, Inc 77  
## 7 Nestec 77

write\_csv(applicant\_count, "pizza\_applicants\_1990\_2012.csv")

When we inspect applicant\_count we will see that Graphic Packaging International is the top result with 154 results with Google ranking third with 123 results followed by Microsoft. This could suggest that Google and Microsoft are suddenly entering the market for online pizza sales or pizza making software or, as is more likely, that there are uses other uses of the word pizza in patent data that we are not aware of.

As part of our infographic we will want to explore this intriguing result in more detail. We can do this by creating a subdataset for Google using filter().

## Selecting applicants using filter()

As we saw above, while select() functions with columns, filter() from dplyr works with rows. Here we will filter the data to select the rows in the applicants column that contain Google Inc. and then write that to a .csv for use in our infographic.

google <- filter(applicants, applicants == "Google Inc")  
write\_csv(google, "google\_1990\_2012.csv")

Note that the correct result for the period 1990 to 2012 for Google is 123 records from 191 records across the whole pizza dataset. The correct result will be achieved only where you use the filtered, separated and trimmed data we created in applicants.

## Generating IPC Tables

In the next step we will want to generate two tables containing International Patent Classification (IPC) data. IPC codes and the Cooperative Patent Classification (CPC, not present in this dataset) provide information on the technologies involved in a patent document. The IPC is hierarchical and proceeds from the general class level to the detailed group and subgroup level. Experience reveals that the majority of patent documents receive more than one IPC code with a trends towards increasing use of IPC codes to more fully describe the technological aspects of patent documents.

The pizza dataset contains IPC codes on the class and the subclass level in concatenated fields. One important consideration in using IPC data is that the descriptions are long and can be difficult for non-specialists to grasp. This can make visualising the data difficult and often requires manual efforts to edit labels for display.

We now want to generate three IPC tables.

1. A general IPC table for the pizza dataset
2. A general IPC table for the Google dataset
3. A more detailed IPC subclass table for the Google dataset

For ease of presentation in an infographic we will use the ipc\_class field. For many patent analytics purposes this will be too general. However it has the advantage of being easy to visualise.

To generate the table we can use a generic function based on the code developed for dealing with the applicants data.

library(dplyr)  
library(tidyr)  
library(stringr)  
patent\_count <- function(data, col = "", col1 = "", n\_results = n\_results, sep = "[[:alnum:]]+") {  
 i <- str\_count(data[[col]], pattern = sep) %>% na.omit()  
 i <- as.integer(max(i) + 1)  
 p\_count <- select\_(data, col, col1) %>% separate\_(col, 1:i, sep = sep, fill = "right") %>%   
 mutate(n = sum(col1 = 1)) %>% gather(x, col, 1:i, na.rm = TRUE)  
 p\_count$col <- str\_trim(p\_count$col, side = "both")  
 select(p\_count, col, n) %>% count(col, wt = n) %>% arrange(desc(n)) %>%   
 .[1:n\_results, ] %>% print()  
}

The patent\_count() function ties together the field\_count() and the code we developed for applicants. It contains variations to make it work as a function. The function takes four arguments

1. col = the concatenated column that we want to split and gather back in
2. col1 = a column for generating counts (in this dataset the publication\_number)
3. n\_results = the number of results we want to see in the new table (typically 10 or 20 for visualisation)
4. sep = the separator to use to separate the data in col. With patent data this is almost always ;

To generate the ipc\_class data we can do the following and then write the file to .csv. Note that we have set the number of results n\_results to 10.

pizza\_ipc\_class <- patent\_count(data = pizza\_1990\_2012, col = "ipc\_class", col1 = "publication\_number",   
 n\_results = 10, sep = ";")

## Source: local data frame [10 x 2]  
##   
## col n  
## (chr) (dbl)  
## 1 A21: Baking 2233  
## 2 A23: Foods Or Foodstuffs 1843  
## 3 B65: Conveying 1383  
## 4 G06: Computing 1326  
## 5 A47: Furniture 932  
## 6 H04: Electric Communication Technique 747  
## 7 H05: Electric Techniques Not Otherwise Provided For 613  
## 8 F24: Heating 512  
## 9 A61: Medical Or Veterinary Science 318  
## 10 G07: Checking 226

write\_csv(pizza\_ipc\_class, "pizza\_ipcclass\_1990\_2012.csv")

Note that this dataset is based on the main pizza\_1990\_2012 dataset (including cases where no applicant name is available). The reason we have not used the applicants dataset is because that dataset will duplicate the IPC field for each split of an applicant name. As a result it will over count the IPCs by the number of applicants on a document name. As this suggests, it is important to be careful when working with data that has been tidied because of the impact on other counts.

This problem does not apply in the case of our Google data because the only applicant listed in that data is Google (excluding co-applicants). We can therefore safely use the Google dataset to identify the IPC codes.

google\_ipc\_class <- patent\_count(data = google, col = "ipc\_class", col1 = "publication\_number",   
 n\_results = 10, sep = ";")

## Source: local data frame [10 x 2]  
##   
## col n  
## (chr) (dbl)  
## 1 G06: Computing 105  
## 2 H04: Electric Communication Technique 17  
## 3 G01: Measuring 14  
## 4 G09: Educating 11  
## 5 G10: Musical Instruments 7  
## 6 A63: Sports 1  
## 7 G08: Signalling 1  
## 8 NA NA  
## 9 NA NA  
## 10 NA NA

write\_csv(google\_ipc\_class, "google\_ipcclass\_1990\_2012.csv")

There are only 7 classes and as we might expect they are dominated by computing. We might want to dig into this in a little more detail and so let's also create an IPC subclass field.

google\_ipc\_subclass <- patent\_count(data = google, col = "ipc\_subclass\_detail",   
 col1 = "publication\_number", n\_results = 10, sep = ";")

## Source: local data frame [10 x 2]  
##   
## col  
## (chr)  
## 1 G06F: Electric Digital Data Processing  
## 2 G06Q: Data Processing Systems Or Methods, Specially Adapted For Administrat  
## 3 G01C: Measuring Distances, Levels Or Bearings  
## 4 G09B: Educational Or Demonstration Appliances  
## 5 G10L: Speech Analysis Or Synthesis  
## 6 H04W: Wireless Communication Networks  
## 7 G09G: Arrangements Or Circuits For Control Of Indicating Devices Using Stat  
## 8 H04B: Transmission  
## 9 H04L: Transmission Of Digital Information, E.G. Telegraphic Communication  
## 10 H04M: Telephonic Communication  
## Variables not shown: n (dbl)

write\_csv(google\_ipc\_subclass, "google\_ipcsubclass\_1990\_2012.csv")

We now have the data on technology areas that we need to understand our data. The next and final step is to generate data from the text fields.

### Phrases Tables

We will be using data from words and phrases in the titles of patent documents for use in a word cloud in our infographic. It is possible to generate this type of data in R directly using the tm and NLP packages. Our pizza dataset already contains a title field broken down into phrases using Vantagepoint software and so we will use that. We will use the field title\_nlp\_multiword\_phrases as phrases are generally more informative than individual words. Once again we will use our general patent\_count() function although experimentation may be needed with the number of phrases that visualise well in a word cloud.

pizza\_phrases <- patent\_count(data = pizza\_1990\_2012, col = "title\_nlp\_multiword\_phrases",   
 col1 = "publication\_number", n\_results = 15, sep = ";")

## Source: local data frame [15 x 2]  
##   
## col n  
## (chr) (dbl)  
## 1 Food Product 179  
## 2 Microwave Ovens 137  
## 3 Making Same 48  
## 4 conveyor Oven 46  
## 5 Crust Pizza 44  
## 6 microwave Heating 41  
## 7 Bakery Product 40  
## 8 pizza Box 40  
## 9 Microwave Cooking 39  
## 10 Pizza Oven 37  
## 11 pizza Dough 35  
## 12 Cook Food 34  
## 13 Baked Product 33  
## 14 Related Method 32  
## 15 Food Item 29

write\_csv(pizza\_phrases, "pizza\_phrases\_1990\_2012.csv")

Now we do the same with the Google data.

google\_phrases <- patent\_count(data = google, col = "title\_nlp\_multiword\_phrases",   
 col1 = "publication\_number", n\_results = 15, sep = ";")

## Source: local data frame [15 x 2]  
##   
## col n  
## (chr) (dbl)  
## 1 Digital Map System 10  
## 2 conversion Path Performance Measures 9  
## 3 Mobile Device 8  
## 4 Search Results 8  
## 5 Geographical Relevance 4  
## 6 Local Search Results 4  
## 7 Location Prominence 4  
## 8 Network Speech Recognizers 4  
## 9 Processing Queries 4  
## 10 Search Query 4  
## 11 aspect-Based Sentiment Summarization 3  
## 12 authoritative Document Identification 3  
## 13 Business Listings Search 3  
## 14 Content Providers 3  
## 15 indexing Documents 3

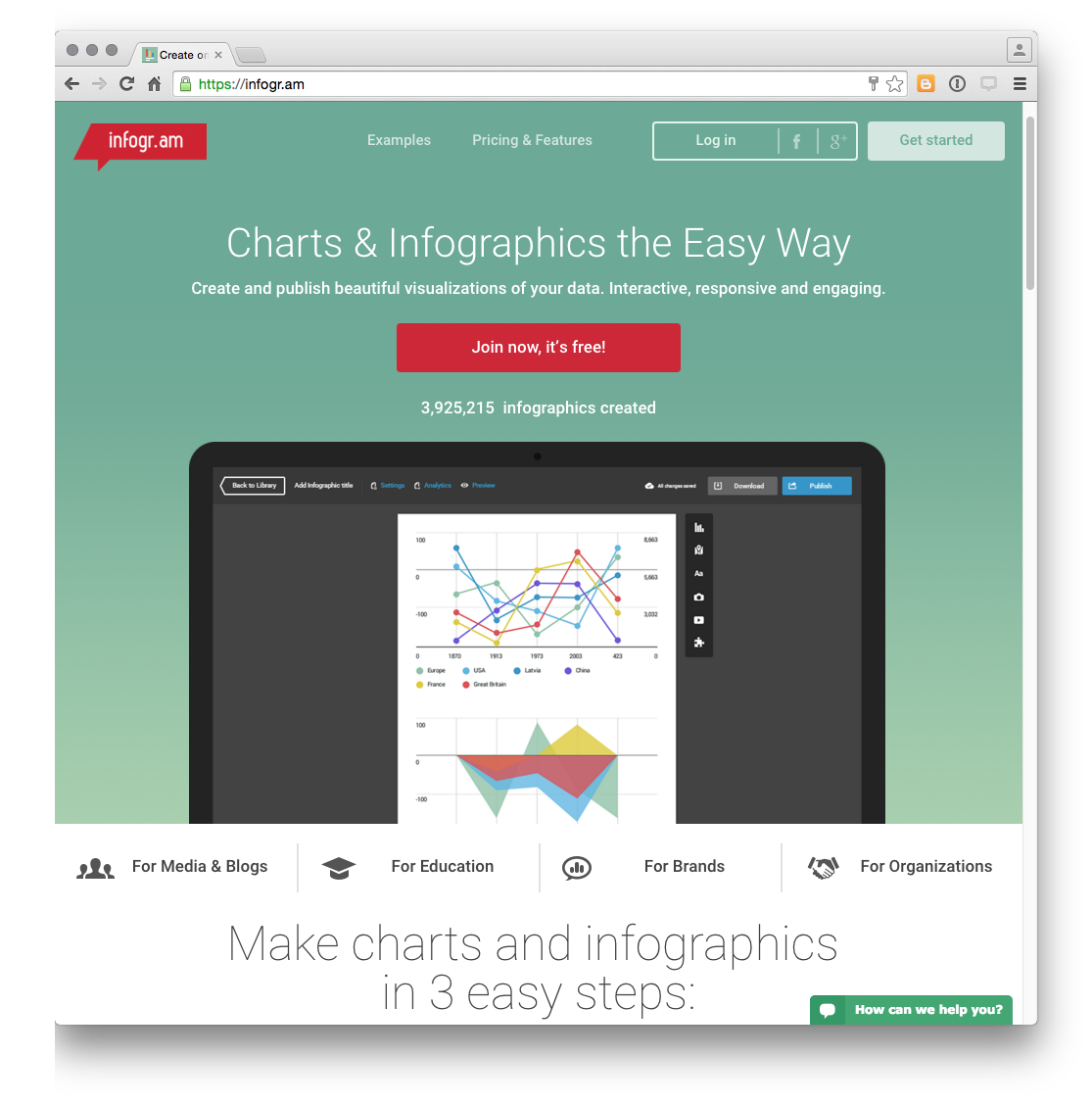
write\_csv(google\_phrases, "google\_phrases\_1990\_2012.csv")

We now have the following .csv files.

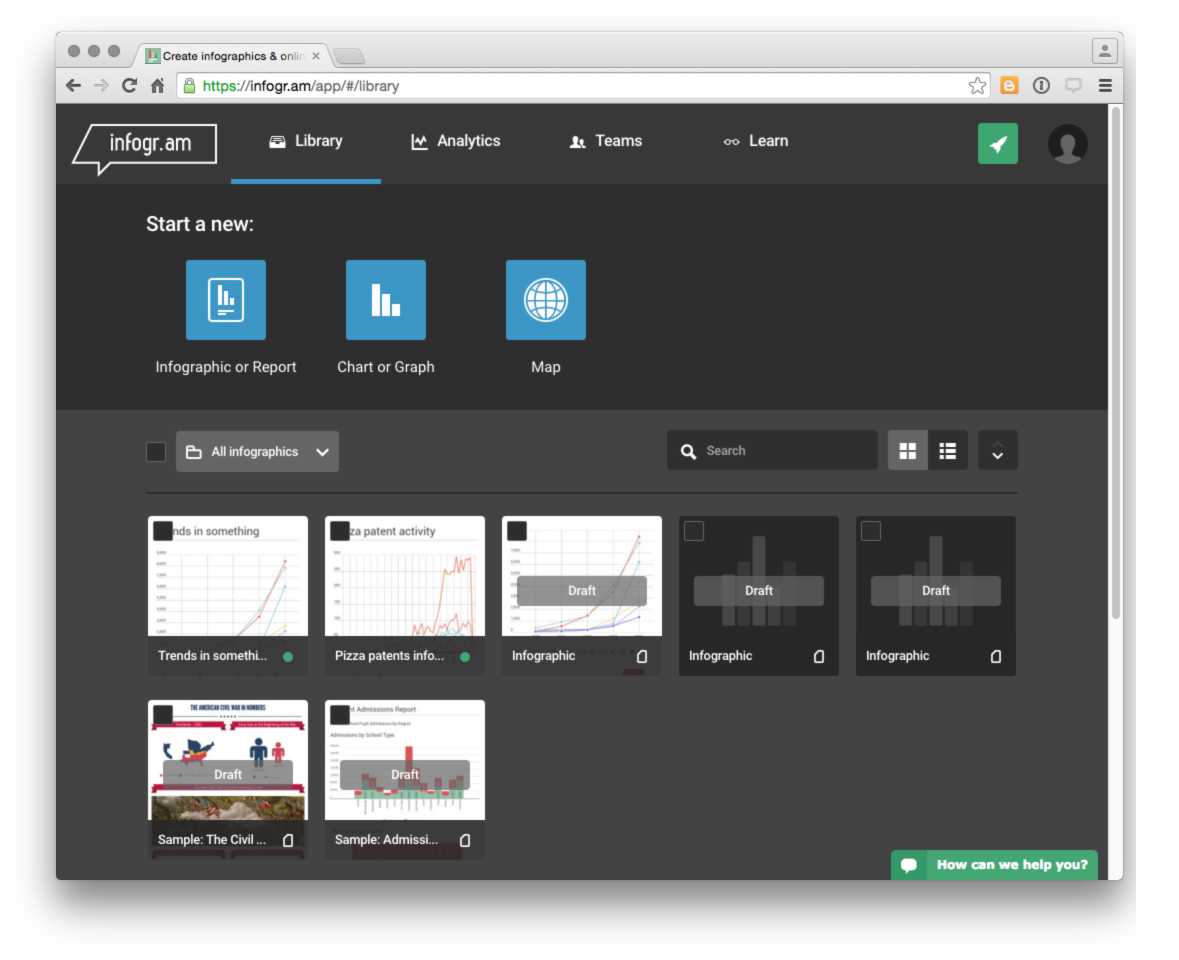
1. pizza\_total\_1990\_2012
2. pizza\_country\_1990\_2012
3. pizza\_applicants\_1990\_2012
4. pizza\_ipcclass\_1990\_2012
5. pizza\_phrases\_1990\_2012
6. Google\_1990\_2012
7. Google\_ipclass\_1990\_2012
8. Google\_ipcsubclass\_1990\_2012
9. Google\_phrases-1990\_2012

## Creating an infographic in infogr.am

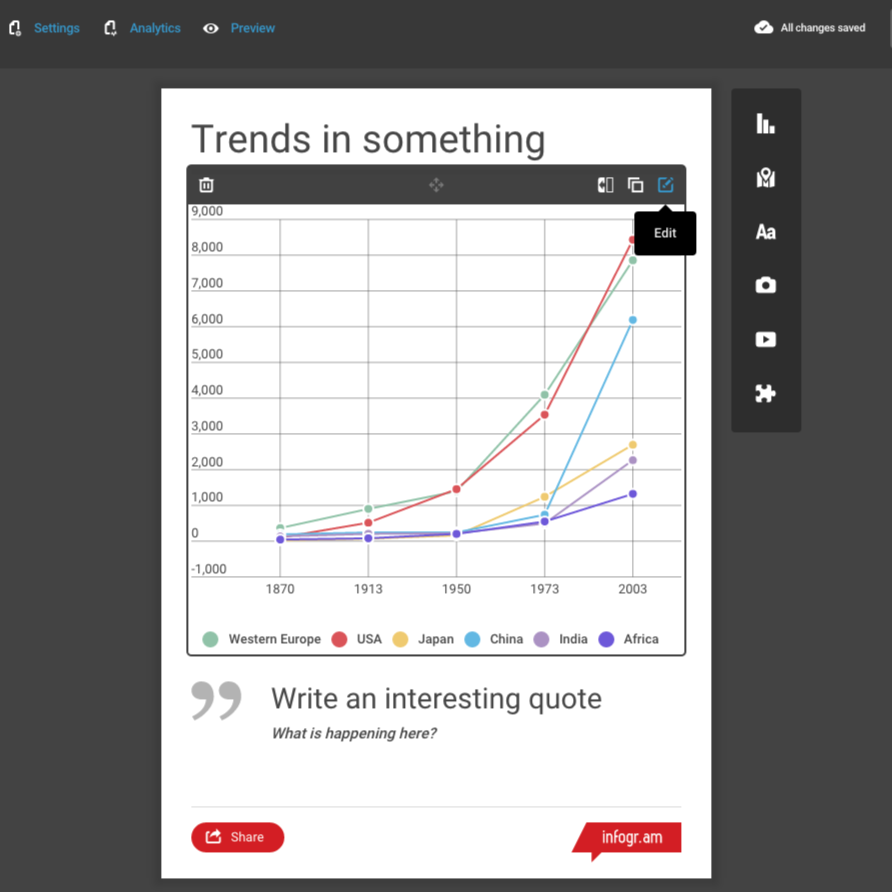
We first need to sign up for a free account with infogr.am[[21]](#footnote-21)



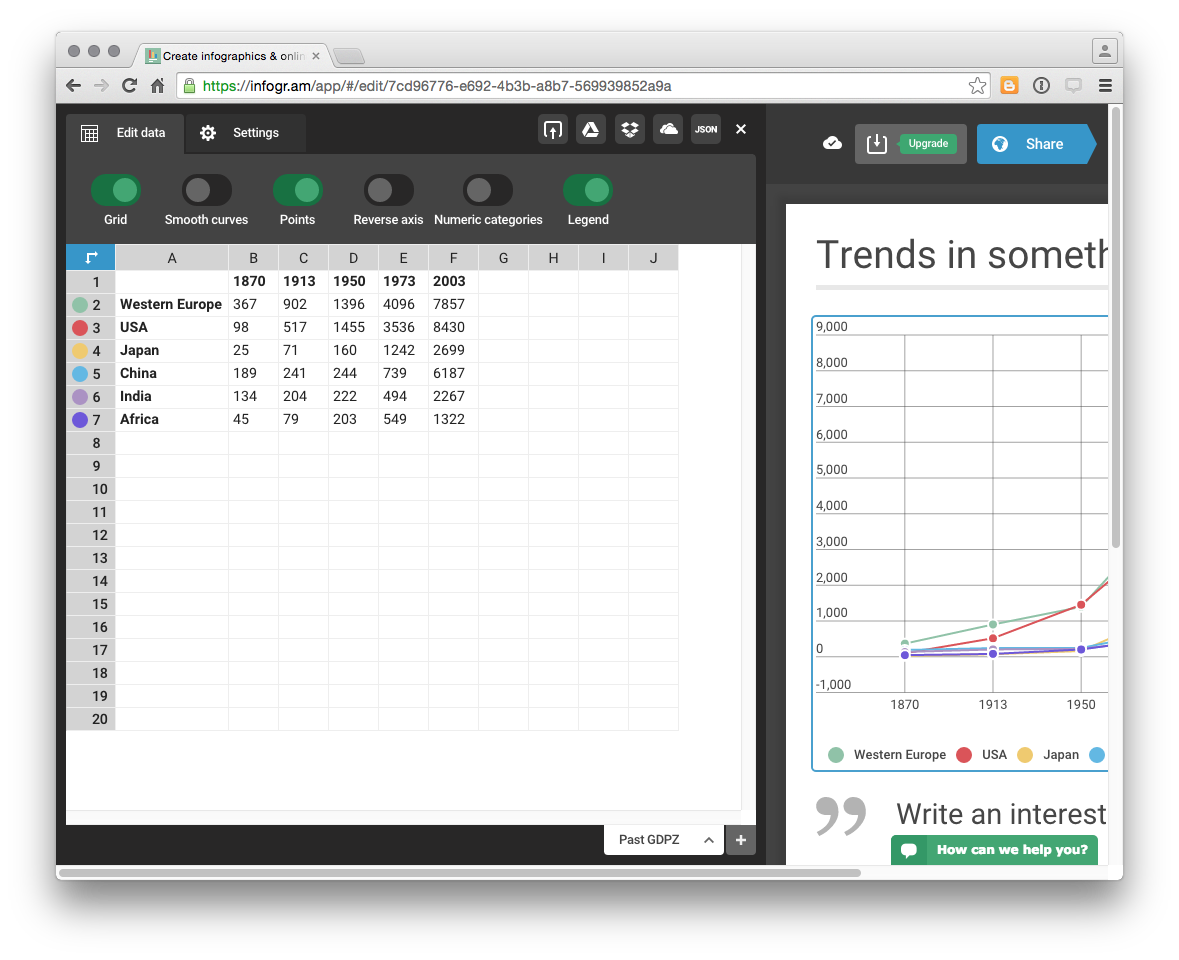
We will then see a page with some sample infographics to provide ideas to get you started.



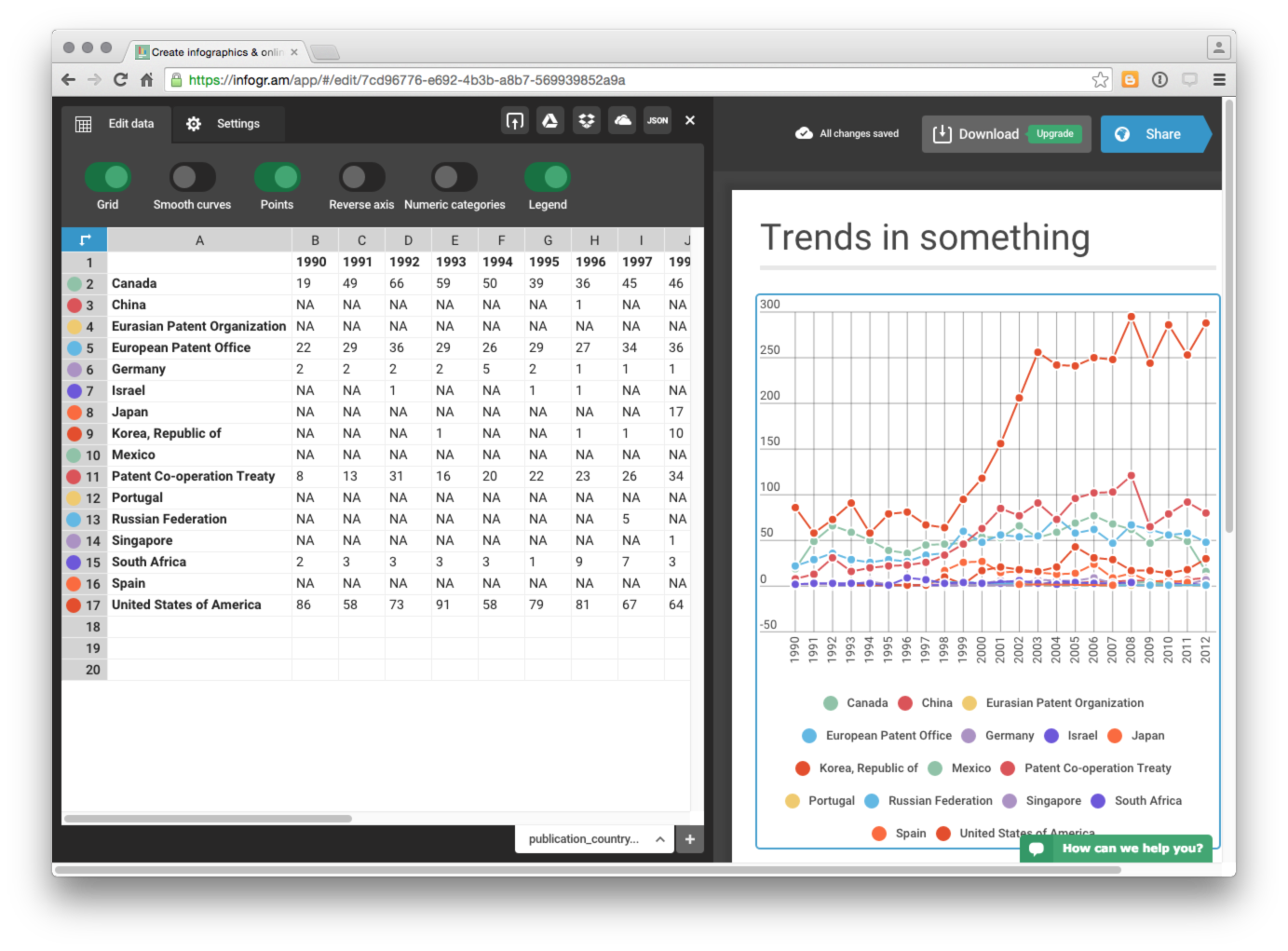
Click on one of the infograms with a graph such as Trends in Something and then click inside the graph box itself and select the edit button in the top right.



This will open up a data panel with the toy data displayed.

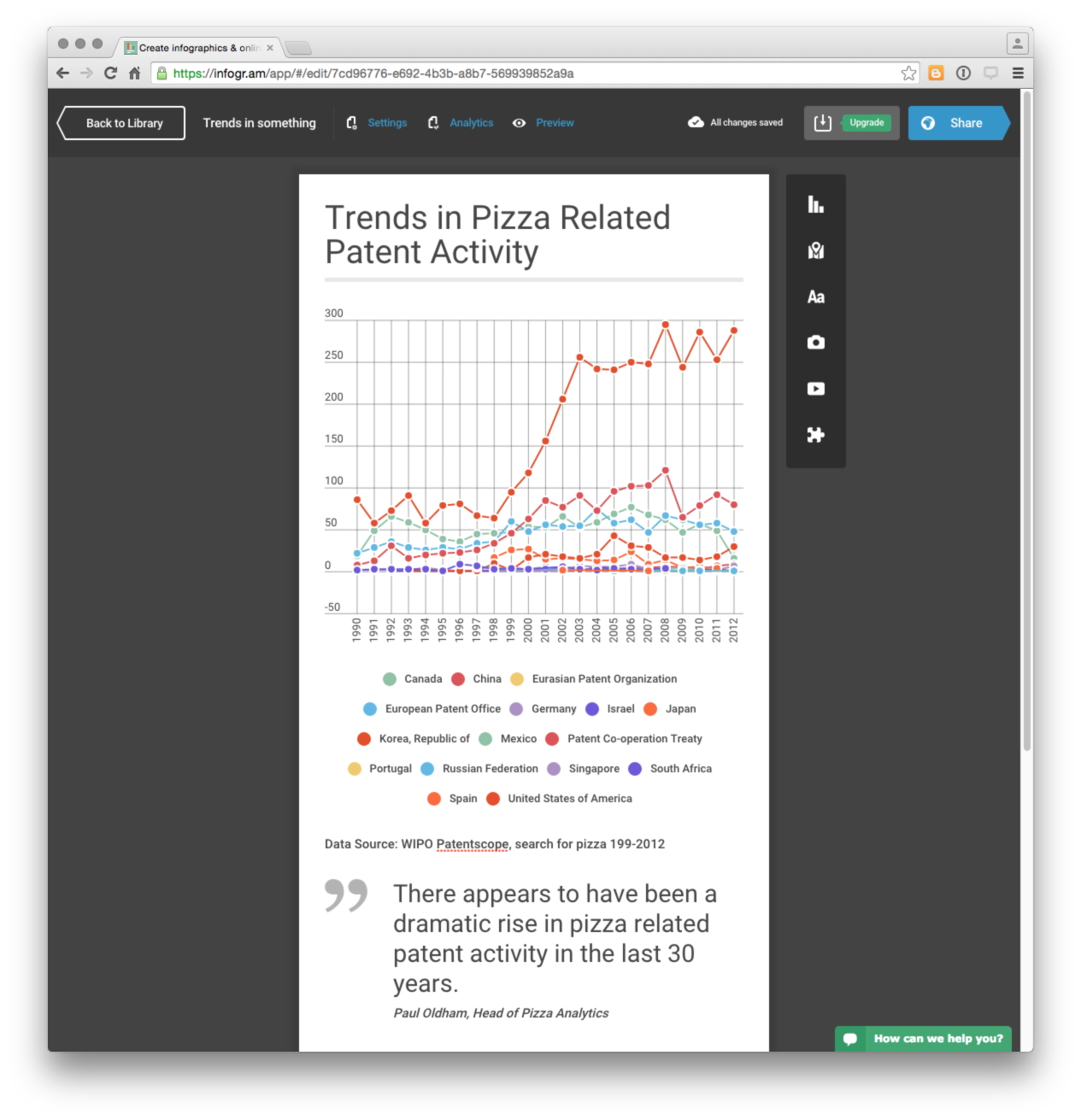


We want to replace this data by choosing the upload button and selecting our pizza\_country\_1990\_2012.csv file.



We now have a decent looking graph for our country trends data where we can see the number of records per country and year by hovering over the relevant data points. While some of the countries with low frequency data are crunched at the bottom (and would be better displayed in a separate graph), hovering over the data or over a country name will display the relevant country activity. We will therefore live with this.

We now want to start adding story elements by clicking on the edit button in the title. Next we can start adding new boxes using the menu icons on the right. Here we have changed the title, added a simple body text for the data credit and then a quote from someone describing themselves as the Head of Pizza Analytics.



Next we need to start digging into the data using our IPC, applicants and phrases data.

To work with our IPC class data we will add a bar chart and load the data. To do this select the graph icon in the right and then Bar. Once again we will choose edit and then load our pizza\_ipcclass\_1990\_2012 dataset. Then we can add a descriptive text box. We can then continue to add elements as follows:

1. applicants bar chart
2. pizza phrases by selecting graph and word cloud
3. Google ipc-subclass
4. Google word cloud.

One useful approach to developing an infographic is to start by adding the images and then add titles and text boxes to raise key points. In infogram new text boxes appear below existing boxes but can be repositioned by dragging and dropping boxes onto each other.

One nice feature of infogram is that it is easy to share the infographic with others through a url, an embed code or on facebook or via twitter.

At the end of the infographic it is a good idea to provide a link where the reader can obtain more information.... such as the full report or the underlying data. In this case we will add a link to the Tableau workbook on pizza patent activity that we developed in an earlier chapter.

Our final infographic should look something like this online version[[22]](#footnote-22).

### Round Up

In this chapter we have concentrated on using R to tidy patent data in order to create an online infographic using free software. Using our trusty pizza patent data from WIPO Patentscope we walked through the process of wrangling and tidying patent data first using short lines of code that we then combined into a function. As this introduction to tidying data in R has hopefully revealed, R and packages such as dplyr, tidyr and stringr provide very useful tools for working with patent data... and they are free and well supported.

In the final part of the chapter we used the data we had generated in RStudio to create an infographic using infogr.am that we then shared online. Infogram is just one of a number of online infographic services and it is well worth trying other services such as easel.ly[[23]](#footnote-23) to find a service that meets your needs.

As regular users of R will already know, it is already possible to produce all of these graphics (such as word clouds) directly in R using tools such as ggplot2, plotly and word clouds using packages such as wordcloud. Some of these topics have been covered in other chapters and for more on text mining and word clouds in R see this recent article on R-bloggers[[24]](#footnote-24). None of the infographic services we viewed appeared to offer an API that would enable a direct connection with R. There also seems to be a gap in R's packages where infographics might sit with this 2015 R-bloggers article[[25]](#footnote-25) providing a walk through on how to create a basic infographic.

1. <http://www.inpi.gov.br/menu-servicos/informacao/radares-tecnologicos> [↑](#footnote-ref-1)
2. <http://www.inpi.gov.br/menu-servicos/arquivos-cedin/n08_radar_tecnologico_nano_residuos_versao_resumida_ingles_atualizada_20160122.pdf> [↑](#footnote-ref-2)
3. <http://www.wipo.int/patentscope/en/programs/patent_landscapes/> [↑](#footnote-ref-3)
4. <http://www.wipo.int/export/sites/www/patentscope/en/programs/patent_landscapes/reports/documents/assistivedevices_infographic.pdf> [↑](#footnote-ref-4)
5. <https://infogr.am/app/#/library> [↑](#footnote-ref-5)
6. <http://www.easel.ly> [↑](#footnote-ref-6)
7. <https://magic.piktochart.com/templates> [↑](#footnote-ref-7)
8. <https://www.canva.com/create/infographics/> [↑](#footnote-ref-8)
9. <https://venngage.com> [↑](#footnote-ref-9)
10. <http://www.coolinfographics.com/tools/> [↑](#footnote-ref-10)
11. <https://infogr.am/app/#/library> [↑](#footnote-ref-11)
12. <https://infogr.am/app/#/library> [↑](#footnote-ref-12)
13. <https://www.rstudio.com/products/rstudio/download/> [↑](#footnote-ref-13)
14. <https://cran.rstudio.com> [↑](#footnote-ref-14)
15. <https://github.com/poldham/opensource-patent-analytics/tree/master/2_datasets> [↑](#footnote-ref-15)
16. <http://poldham.github.io/reading-csv-files-in-R/> [↑](#footnote-ref-16)
17. <http://poldham.github.io/reading-writing-excel-files-R/> [↑](#footnote-ref-17)
18. <http://poldham.github.io/ggplot_pizza_patents_part1/> [↑](#footnote-ref-18)
19. <http://poldham.github.io/ggplot_pizza_patents_part2j/> [↑](#footnote-ref-19)
20. <http://vita.had.co.nz/papers/tidy-data.pdf> [↑](#footnote-ref-20)
21. <https://infogr.am/> [↑](#footnote-ref-21)
22. <https://infogr.am/trends_in_something> [↑](#footnote-ref-22)
23. <https://www.easel.ly> [↑](#footnote-ref-23)
24. <http://www.r-bloggers.com/building-wordclouds-in-r/> [↑](#footnote-ref-24)
25. <http://www.r-bloggers.com/r-how-to-layout-and-design-an-infographic/> [↑](#footnote-ref-25)