Introducing R and Graphing Pizza Patents: part 1

This is the first part of a two part article on using R and the ggplot2 package to visualise patent data. In a previous article we looked at visualising pizza related patent activity in Tableau Public. In this article we look at how to plot our pizza dataset using the ggplot2 package in RStudio. You do not need to know anything about R to follow this article. You will however need to install [RStudio Desktop](http://www.rstudio.com/products/rstudio/) for your operating system (see below).

Part 1 will introduce the basics of handling data in R in preparation for plotting and will then use the quick plot or qplot function in ggplot2 to start graphing elements of the pizza patents dataset.

Part 2 will go into more depth on handling data in R and the use of ggplot2.

ggplot2 is an implementation of the theory behind the Grammar of Graphics. The theory was originally developed by Leland Wilkinson and reinterpreted with considerable success by Hadley Wickham at Rice University and RStudio. The basic idea behind the Grammar of Graphics is that any statistical graphic can be built using a set of simple layers consisting of:

1. A dataset containing the data we want to see (e.g x and y axes and data points)
2. A geometric object (or geom) that defines the form we want to see (points, lines, shapes etc.) known as a geom. Multiple geoms can be used to build a graphic (e.g, points and lines etc.).
3. A coordinate system (e.g. a grid, a map etc.).

On top of these three basic components, the grammar includes statistical transformations (or stats) describing the statistics to be applied to the data to create a bar chart or trend line. The grammar also describes the use of faceting (trellising) to break a dataset down into smaller components (see Part 2).

A very useful article explaining this approach is [Hadley Wickham's 2010 A Layered Grammar of Graphics](http://vita.had.co.nz/papers/layered-grammar.pdf) (preprint) and is recommended reading.

The power of this approach is that it allows us to build complex graphs from simple layers while being able to control each element and understand what is happening. One way to think of this is as stripping back a graph to its basic elements and allowing you to decide what each element (layer) should contain and look like. In short, you get to decide what your graphs look like.

ggplot2 contains two main functions:

1. qplot (quick plot)
2. ggplot()

The main difference between the two is that quick plot makes assumptions for you and, as the name suggests, is used for quick plots. In contrast, with ggplot we build graphics from scratch with helpful defaults that give us full control over what we see.

In this article we will start with qplot and increasingly merge into developing plots by adding layers in what could be called a ggplot kind of way. We will develop that further in the Part 2.

## Getting Started with R

This article assumes that you are new to using R. You do not need any knowledge of programming in R to follow this article. While you don't need to know anything about R to follow the article, you may find it helpful to know that :

1. R is a statistical programming language. That can sound a bit intimidating. However, R can handle lots of other tasks a patent analyst might need such as cleaning and tidying data or text mining. This makes it a good choice for a patent analyst.
2. R works using packages (libraries) for performing tasks such as importing files, manipulating files and graphics. There are around 6,819 packages and they are open source (mainly it seems under the MIT licence). If you can think of it there is probably a package that meets (or almost meets) your analysis needs.
3. Packages contain functions that do things such as read\_csv() to read in a comma separated file.
4. The functions take arguments that tell them what you want to do, such as specifying the data to graph and the x and y axis e.g. qplot(x = , y = , data = my dataset).
5. If you want to learn more, or get stuck, there are a huge number of resources and free courses out there and RStudio lists some of the main resources on their website [here](http://www.rstudio.com/resources/training/online-learning/). With R you are never alone.

R is best learned by doing. The main trick with R is to install and load the packages that you will need and then to work with functions and their arguments. Given that most patent analysts are likely to be unfamiliar with R we will adopt the simplest approach possible to make sure it is clear what is going on at each step.

The first step is to install R and RStudio desktop for your operating system by following the links and instructions [here](http://www.rstudio.com/products/rstudio/download/). Follow this very useful [Computerworld article](http://www.computerworld.com/article/2497143/business-intelligence/business-intelligence-beginner-s-guide-to-r-introduction.html?page=2) to become familiar with what you are seeing. You may well want to follow the rest of that article. Inside R you can learn a lot by installing the Swirl package that provides interactive tutorials for learning R. Details are provided in the resources at the end of the article.

The main thing you need to do to get started other than installing R and RStudio is to open RStudio and install some packages.

In this article we will use four packages:

1. readr to quickly read in the pizza patent dataset as an easy to use data table.
2. dplyr for quick addition and operations on the data to make it easier to graph.
3. ggplot2 or Grammar of Graphics 2 as the tool we will use for graphing.
4. ggthemes provides very useful additional themes including Tufte range plots, the Economist and Tableau and can be accessed through [CRAN](http://cran.r-project.org/web/packages/ggthemes/index.html) or [Github](https://github.com/jrnold/ggthemes).

## Getting Started

If you don't have these packages already then install each of them below by pressing command and Enter at the end of each line. As an alternative select ***Packages > Install*** in the pane displaying a tab called Packages. Then enter the names of the packages one at a time without the quotation marks.

install.packages("readr")  
install.packages("dplyr")  
install.packages("ggplot2")  
install.packages("ggthemes")

Then make sure the packages have loaded to make them available. Press command and enter at the end of each line below (or, if you are feeling brave, select them all and then click the icon marked Run).

library(readr)  
library(dplyr)  
library(ggplot2)  
library(ggthemes)

You are now good to go.

## About the pizza patent dataset

The pizza patents dataset is a set of 9,996 patent documents from the WIPO Patentscope database that make reference somewhere in the text to the term pizza. Almost everybody likes pizza and this is simply a working dataset that we can use to learn how to work with different open source tools. This will also allow us over time to refine our understanding of patent activity involving the term pizza and hone in on actual pizza related technology. In previous walkthroughs we divided the pizza dataset into a set of distinct data tables to enable analysis and visualisation using Tableau Public. You can download that dataset in .csv format [here](https://github.com/poldham/opensource-patent-analytics/blob/master/2_datasets/pizza_medium_clean/pizza_medium_clean.zip?raw=true). These data tables are:

1. pizza (the core set)
2. applicants (a subdataset divided and cleaned on applicant names)
3. inventors (a subdataset divided and cleaned on inventor names)
4. ipc\_class (a subdataset divided on ipc class names names)
5. applicants\_ipc (a child dataset of applicants listing the IPC codes)

In this article we will focus on the pizza table as the core set. However, you may want to experiment with other sets.

## Reading in the Data

We will use the readr package to rapidly read in the pizza set to R (for other options see the in depth articles on reading in .csv and Excel files and the recent Getting your Data into R RStudio [webinar](http://www.rstudio.com/resources/webinars/)). readr is nice and easy to use and creates a data table that we can easily view.

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

We now have a data table with the data. We can inspect this data in a variety of ways:

***1. View***

See a separate table in a new tab. This is useful if you want to get a sense of the data or look for column numbers.

View(pizza)

***2. head (for the bottom use tail)***

head allows you to see the top few rows or using tail the bottom few rows.If you would like to see more rows add a number after the dataset name e.g. `head(pizza, 20).

head(pizza)

***3. dimensions***

This allows us to see how many rows there are (9996) and how many columns(31)

dim(pizza)

***4. Summary***

Provides a summary of the dataset columns including quick calculations on numeric fields and the class of vector.

summary(pizza)

***5.The class of R object***

class() is one of the most useful functions in R because it tells you what kind of object or vectors you are dealing with. R vectors are normally either character, numeric, or logical (TRUE, FALSE) but classes also include integers and factors. Most of the time patent data is of either the character type or a date.

class(pizza)

***4. str - See the structure***

As you become more familiar with R the function str() becomes one of the most useful for examining the structure of your data. For example, using str we can see whether an object we are working with is a simple vector, a list of objects or a list that contains a set of data frames (e.g.) tables. If things don't seem to be working then class and str will often help you to understand why not.

str(pizza)

These options illustrate the range of ways that you can view the data before and during graphing. Mainly what will be needed is the column names but we also need to think about the column types.

If we inspect this data using str(pizza) we will see that the bulk of the fields are character fields. One feature of patent data is that it rarely includes actual numeric fields (such as counts). Most fields are character fields such as names or alphanumeric values (such as publication numbers e.g. US20151234A1). Sometimes we see counts such as citing documents or family members but most of the time our fields are character fields or dates. A second common feature of patent data is that some fields are concatenated. That is the cells in a column contain more than one value (e.g. multiple inventor or applicant names etc.).

We will walk through how to deal with these common patent data issues in R in other articles. For now, we don't need to worry about the form of data except that it is normally best to select a column (variable) that is not concatenated with multiple values to develop our counts. So as a first step we will quickly create a numeric field from the publication\_number field in pizza.

## Creating a numeric field

To create a numeric field for graphing we will need to do two things

1. add a column
2. assign each cell in that column a value that we can then count.

The most obvious field to use as the basis for counting in the pizza data is the publication\_number field because typically this contains unique alphanumeric identifiers.

To create a numeric field we will use the dplyr package. dplyr and its sister package tidyr are some of the most useful packages available for working in R and come with a handy [RStudio Cheatsheet](http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) and [webinar](http://www.rstudio.com/resources/webinars/archives/#). To see what the functions in dplyr are then click on its name in the packages pane.

Just for future reference the main functions are:

1. filter (to select rows in a data)
2. select (to select the columns you want to work with)
3. mutate (to add columns based on other columns)
4. arrange (to sort)
5. group\_by( to group data)
6. count (to easily summarise data on a value)

dplyr's mutate function allows us to add a new column based on the values contained in one or more of the other columns in the dataset. We will call this new variable n and we could always rename it in the graphs later on. There are quite a variety of ways of creating counts in R but this is one of the easiest. The mutate function is really very useful and worth learning.

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

What we have done here is to tell R that we want to use the mutate() function. We have then passed it a series of arguments consisting of:

1. our dataset = pizza
2. n = the result of the function sum() which is the sum of publication\_number giving the value 1 to each number.
3. pizza <- this tells R to create an object (a data frame) called pizza containing the results. If you take a look in the Environment pane you will now see that pizza has 32 variables. Note that we have now modified the data we imported into R although the original data in the file remains the same. If we now use View(pizza) we will see a new column called n with a value of 1 for each entry.

### Renaming Columns

We will be doing quite a lot of work with the publication\_country\_name field, so let's make our lives a bit easier by renaming it with the dplyr function rename(). We will also do the same for the publication\_country\_code and publication\_year. Note that it is easy to create labels for graphs with ggplot so we don't need to worry about renaming column names too much. We can rename them again later if saving the file to a new .csv file.

pizza <- rename(pizza, pubcountry = publication\_country\_name, pubcode = publication\_country\_code,   
 pubyear = publication\_year)

### Selecting Columns for plotting

We could now simply go ahead and work with pizza. However, for datasets with many columns or requiring different kinds of counts it can be much easier to simply select the columns we want to work with to reduce clutter. We can use the select() function from dplyr to do this.

p1 <- select(pizza, pubcountry, pubcode, pubyear, n)

Note that dplyr will exclude columns that are not mentioned when using select. This is one of the purposes of select as a function. For that reason you will probably want to rename the object (in this case as p1). If we used the name pizza for the object our original table would be reduced to the 4 columns specified by select. Type ?select in the console for further details.

We now have a data frame with 9,996 rows and 4 variables (columns). Use View(p1) or simply enter p1 into the console to take a look.

## Creating Counts

To make life even easier for ourselves we can use function count() from dplyr to group the data onto counts by different variables for graphing. Note that we could defer counting until later, however, this is a good opportunity to learn more about dplyr.

Let's go ahead and construct some counts using p1. At the same time we will use quick plot (qplot) for some exploratory plotting of the results. In the course of this R will show error warnings in red for missing values. We will be ignoring the warning because they are often R telling us things it things we need to know.

### Total by Year

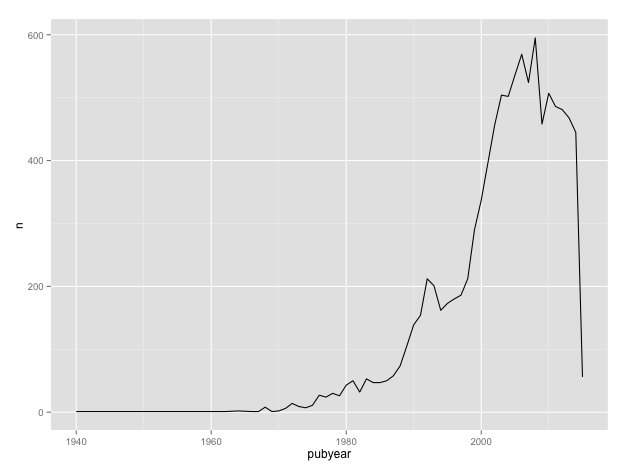
What if we wanted to know the overall total for our sample data by publication year. Try the following.

pt <- count(p1, pubyear, wt = n)

If we now view pt (either by using View(pt), noting the capital V, or clicking pt in the Environment pane) we will see that R has dropped the country columns to present us with an overall total by year in n. We now have a general overview of the data for graphing.

Let's go ahead and quickly plot that using the qplot() function.

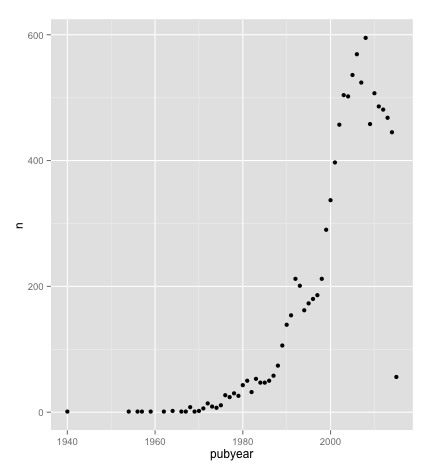
qplot(x = pubyear, y = n, data = pt, geom = "line")



What we have done here is used qplot(), provided the x and y axis columns, the data table to use, and then the type of graph we want to see as geom = "line". We will explain how this works in more detail below, the point here is that it is easy to gain a quick visual of our data.

What this tells us is that we have a very low number of records in the sample from 1940 to around 1970 and that the data appears to fall off a cliff as we move closer to the present. This is helpful because it provides us with some clues on tasks later in the graphing process. To investigate further try changing geom = "line" to geom = "point". In this case we could also simply remove the geom = "line" because "point"" (for scatter plots), is the default setting for qplot and ggplot. Note that the default colour for plotting is black.

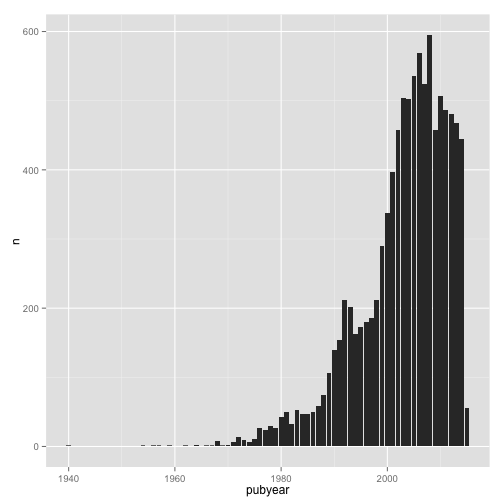
qplot(x = pubyear, y = n, data = pt, geom = "point")



What we now see is that we have a single data point in 1940, zero entries up until the late 1960s and that we have a data cliff that in 2015. We will come back to this below.

If we wanted to create a bar chart of the same data using qplot we would use the following. Note that for qplot we have added the geometric object we want to see as geom = "bar" and then the statistic we want to use as stat = "identity".

qplot(x = pubyear, y = n, data = pt, geom = "bar", stat = "identity")



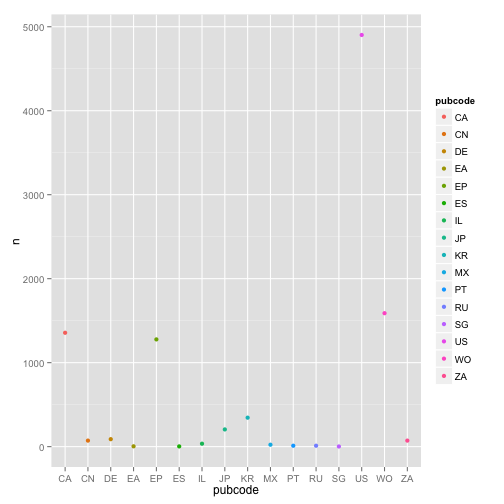
### Count by Country

We now want to gain a quick view of the overall data by the number of records per country that we will call pc. Following the same logic we used above, we specify the column to sum on (using wt = n).

pc <- count(p1, pubcountry, pubcode, wt = n)

To quickly plot the data we will use the pubcode (country code) as the value for the x axis to avoid squished country names. We will also add an argument for the color by specifying color = pubcode.

qplot(pubcode, n, data = pc, colour = pubcode)



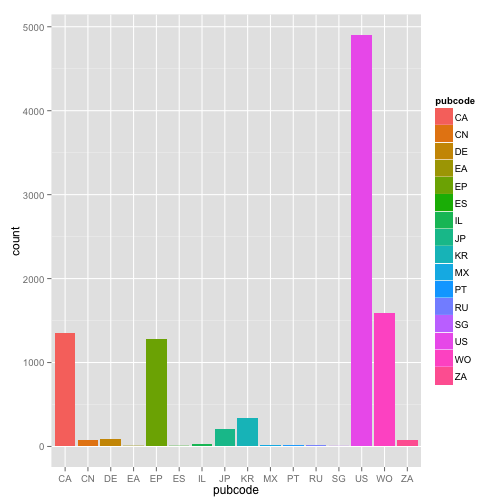
Note here that we have shortened the code in two ways:

1. We have removed the x = and y =, because qplot will know that we are specifying x and then y before we reach the data argument.
2. We have not specified geom = "point" for a scatter plot because "point" is the default for data with an x and y axis specified (if only the x axis is specified it is a bar).

This simply illustrates that when we are familiar with the different arguments we need we can then drop some of the formalities.

We can create a coloured bar chart as follows (see ?geom\_bar for helpful examples)

qplot(x = pubcode, data = pc, weight = n, geom = "bar", fill = pubcode)



We now have a quick overview of the count of pizza documents by country using the two letter country codes. This helps us because we can see that some countries have very low scores while one country (the US) dominates the data. In patent analysis we typically confront a situation where major patent offices (the US, the European Patent Office, Japan and the Patent Cooperation Treaty) dominate the data resulting in compressed data in graphs. In normal circumstances we might want to drop some of this data or we might want to split the data into separate datasets. We will discuss this in part 2 of this article.

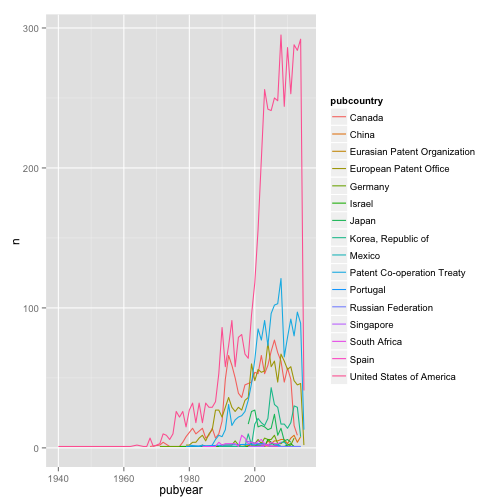
### Count by Country and Year

Having gained an insight into the overall data and individual country scores we now want to take a look at the trends by country. To do that we will create a new table object that we will call pcy and specify the pubcountry, pubcode, and pubyear with the wt for summing as before.

pcy <- count(p1, pubcountry, pubcode, pubyear, wt = n)

We now want to gain a quick view of trends by country, we will add colour (as color = pubcountry) to help distinguish between countries in a line plot. Note that in ggplot2 colour refers to the colour of lines or outlines while fill refers to the colour for an object such as a bar.

qplot(pubyear, n, data = pcy, geom = "line", colour = pubcountry)



We now see a line graph with a set of trend lines for each country with sparse results at the beginning and a data cliff in recent years. We will come back to this.

## Going further with qplot

We have now created three data tables containing counts for plotting to get an insight into our pizza data:

1. pt = publication totals by year
2. pc = total records by publication country
3. pcy = trends by publication country and year.

These plots are rough and ready exploratory tools to give us an idea of what our data looks like and to think about what issues we might want to address in the data and how to do that. However what we have learned about R is:

1. Packages such as readr allow us to easily read in our data. See also readxl and the standard read.csv() in the default utils package.
2. dplyr allows us to wrangle the data by adding values using mutate, renaming columns using rename, selecting the data we want to use with select, and using count to easily generate different types of count for aspects of the data. We will cover the very useful filter() function for rows in part 2.
3. qplot allows us to generate quick graphs by specifying the columns for the x and y axis, the data we want to see and some details of how we want to see the data with the default being a scatter plot.

What we have learned about the pizza data is:

1. We have some sparse results early on in the dataset and a data cliff that is most marked in 2015.
2. Some countries have very limited activity, while others dominate the data.
3. Plotting country trends reveals that some countries are squished to the bottom by the dominance of US data and there appears to be variation across countries in the start of a data cliff.

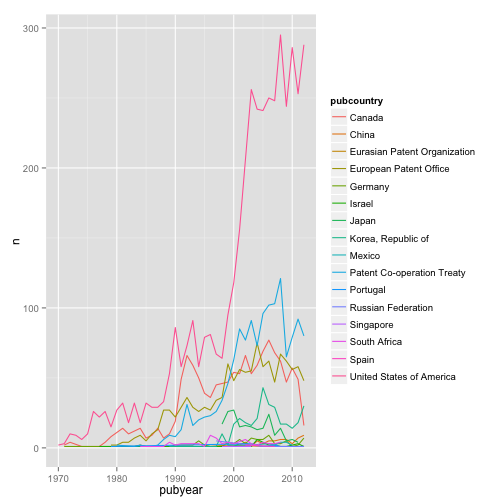
To round off the use of qplot, let's work on some of the issues identified in the data. We won't fix everything, as we will come on to that in part 2, but we will generate a quick graph that makes more sense to our potential audience.

One of the main issues noted above is sparse results and a data cliff. The sparse early results will reflect a lack of use of the term pizza and/or a lack of access to full text data for that period in the patent database. The data cliff will result from two factors. First, a patent application is typically only published at least two years after it was originally filed. This introduces a lag time of at least two years into data on patent trends. Second, databases can vary in their coverage of data from a particular country or how often data is updated for a particular country.

The key issue here is arriving at a more accurate view. In particular, we do not want to mislead readers into thinking that there has been a massive collapse in the use of the term pizza in patents in recent years. To do that with qplot we can limit the x axis to take out the sparse data and to pull back a sensible distance from the data cliff. In terms of the patent data cliff two years may be sufficient although three years is more sensible.

To limit the data we will specify the x axis using xlim (the y axis equivalent is ylim). To specify a data range we will enclose the years we want separated by a comma in brackets with c (for combine or concatenate) at the beginning.

qplot(pubyear, n, data = pcy, geom = "line",   
 colour = pubcountry,   
 xlim = c(1970, 2012))

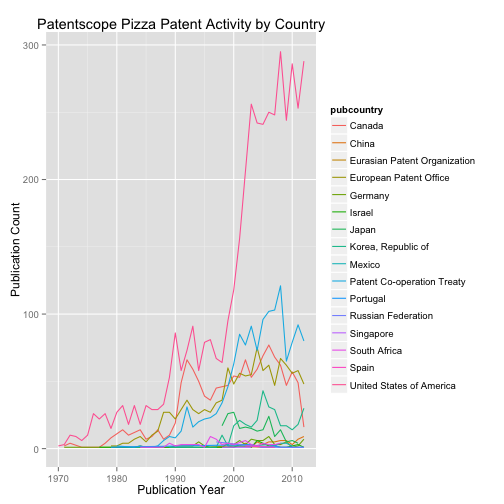


R will throw a warning about removing rows that we will ignore. Note that what we have done here is limited the data displayed on the x axis. We have not removed the data from our pcy table. We will demonstrate how to edit data out using filter in part 2.

That cleans up the plot a little bit. However, we would probably want to add a note to the explanation on the presence of a data cliff and data availability issues.

To finish off we will also add some labels. There are a couple of ways to do this with qplot. The first is to specify each label using xlab = x label, ylab = y label, main = title as below. As the code is becoming long we will indent it.

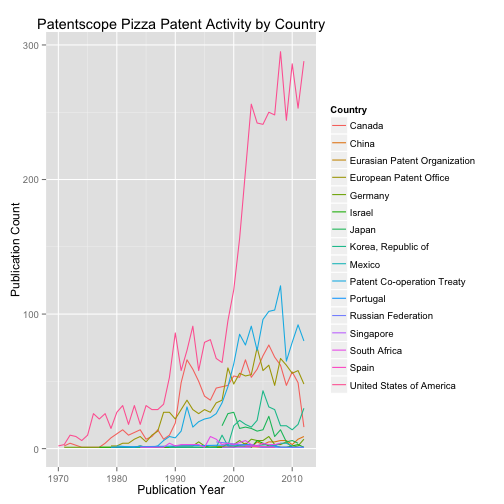
qplot(pubyear, n, data = pcy, geom = "line", colour = pubcountry,   
 xlim = c(1970, 2012),   
 xlab = "Publication Year",   
 ylab = "Publication Count",   
 main = "Patentscope Pizza Patent Activity by Country"  
 )



# Dealing with Legends

We could now do with addressing the legend that reads pubcountry

qplot(pubyear, n, data = pcy, geom = "line", colour = pubcountry,   
 xlim = c(1970, 2012),   
 xlab = "Publication Year",   
 ylab = "Publication Count",   
 main = "Patentscope Pizza Patent Activity by Country") +  
 labs(colour = "Country")



When we run this code we will now see the word Country for the legend.

Note here that whereas in the code so far we have written this as a long string, in the code above we have used + labs(colour = "Country"). This basically adds a layer to our plot that says that the label for the variable we have chosen for the colour (pubcountry) should be called Country.

In practice, there are a number of ways to modify legends. These include using scale or modifying the original data frame. For more on scale options see Winston Chang's excellent Cookbook for R [pages](http://www.cookbook-r.com/Graphs/Legends_(ggplot2)/#using-scales). There is almost always more than one way to achieve the same goal in R.

We now have a reasonable plot. However, let's clarify two problems in our data to avoid misleading readers.

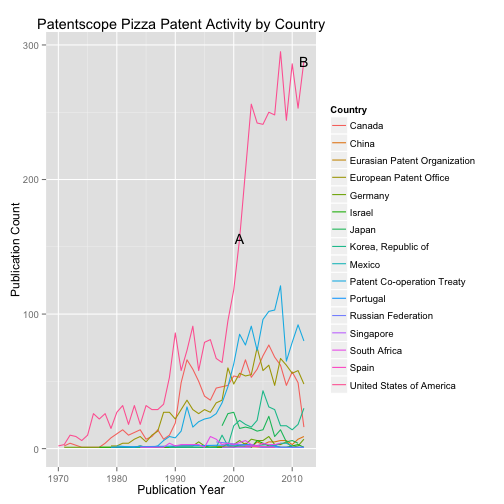
1. There is a sharp spike in US patent activity between 1998 and 2002. There are a number of possible explanations for this. However, in the majority of patent data a key explanation is that prior to 2001 the US only published patent documents when they were granted. In contrast most countries published documents at the application and the grant stage. Because Patentscope aggregates publication data the spike is not as marked as it probably would be if we used full publication records (with kind codes etc.).

US practice has two consequences. First, the data will underestimate actual activity in terms of demand for patent rights in the US prior to 2001 because we only see patent grants. Second, between 2000 and 2001 we normally see an apparent jump in demand that suggests a leap in activity. In reality this is a reporting effect from a change in USPTO publishing practice. While this is rarely mentioned, possibly because of a lack of awareness of the change, it is actually good practice to make a note of this to avoid accidentally misleading readers. 2. We have pulled the data back from the data cliff arising from the lack of published data. It is a good idea to notify readers about this as part of an approach that seeks to tell the truth and avoid misleading the reader with the data (as discussed in Tufte's classic The Visual Display of Quantitative Information.

### Annotating Plots

We can do this either in the accompanying text or in the plot itself. Most of the time it will make sense to do this in the accompanying text. However, if we choose to make a note in the plot we can do this in a number of ways. First, we could add some text using +annotate().

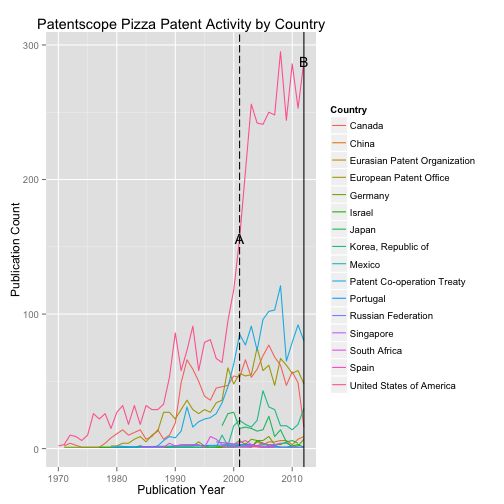
qplot(pubyear, n, data = pcy, geom="line",   
 color = pubcountry,   
 xlim = c(1970, 2012),   
 xlab = "Publication Year",   
 ylab = "Publication Count",   
 main = "Patentscope Pizza Patent Activity by Country") +  
 labs(colour="Country") +   
 annotate("text", x = 2001, y = 156, label = "A") +   
 annotate("text", x = 2012, y = 288, label = "B")



To annotate the plot we have added two layers using annotate(). In both cases we have specified the type of annotation as "text" and the year as the relevant year on the x axis. Then we have looked up the relevant value for the position of the text on the y axis for those years. We could then add an explanation of A and B in the text of the document. More information is available from the [ggplot2 web pages](http://docs.ggplot2.org/0.9.3/index.html) for [annotate](http://docs.ggplot2.org/0.9.3/annotate.html) and also see [geom\_text](http://docs.ggplot2.org/0.9.3/geom_text.html)

A second option would be to draw some lines on the plot. For demonstration we will draw a solid line at 2012 and specify a different line type for 2001. To draw a vertical line we use the function [geom\_vline](http://docs.ggplot2.org/0.9.3/geom_vline.html) where vline stands for vertical line (for horizontal lines us [geom\_hline](http://docs.ggplot2.org/0.9.3/geom_hline.html). We then specify the xintercept point and optionally the color and linetype that we want to see.

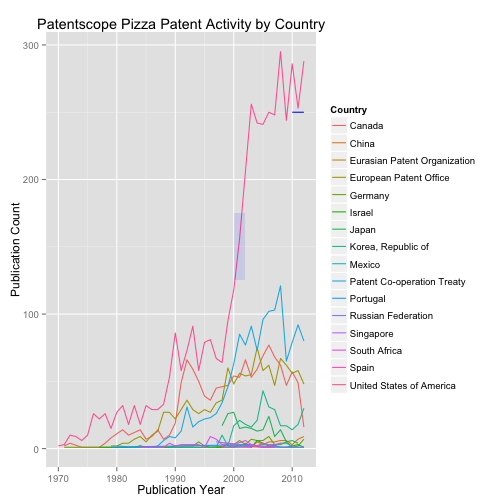
qplot(pubyear, n, data = pcy, geom = "line", color = pubcountry,   
 xlim = c(1970, 2012), xlab = "Publication Year",   
 ylab = "Publication Count",   
 main = "Patentscope Pizza Patent Activity by Country") +   
 labs(colour = "Country") +   
 annotate("text", x = 2001, y = 156, label = "A") +   
 annotate("text", x = 2012, y = 288, label = "B") +   
 geom\_vline(xintercept=2012) +   
 geom\_vline(xintercept=2001, linetype = "longdash")



In this case we have simply added two lines at the relevant years and distinguished the first line by defining the line type. Other options for drawing lines that are worth exploring are [geom\_line](http://docs.ggplot2.org/0.9.3/geom_line.html) and [geom\_abline](http://docs.ggplot2.org/0.9.3/geom_abline.html). However, for those following Tufte's rules in the Visual Display of Quantitative Information, note that the line additions are interfering with our appreciation of the data itself. Put bluntly, they are getting in the way and overwhelming the plot.

There are a number of things that we could do here. 1. We could revert to the labels and perhaps change the size and colour of the text. 2. We could use shorter lines and reduce the length of the line. Alternatively we might want to use a shaded layer instead or a line to avoid obtrusive text. As described in detail in Winston Chang's R Graphics Cookbook we can achieve this by changing the annotations. First we will add a shaded area to the period 2000 to 2002 to demarcate the start and end points of the change in US practice (in terms of transitions from one situation to the other). Second we will add a short horizontal line in the period 2010 to 2012 as the marker for discussion of the data cliff.

qplot(pubyear, n, data = pcy, geom = "line", color = pubcountry,   
 xlim = c(1970, 2012),   
 xlab = "Publication Year",   
 ylab = "Publication Count",   
 main = "Patentscope Pizza Patent Activity by Country"  
 ) +   
 labs(colour = "Country") +   
 annotate("rect", xmin = 2000, xmax = 2002, ymin = 125,   
 ymax = 175, alpha = .1, fill = "blue") +  
 annotate("segment", x=2010, xend = 2012, y = 250,   
 yend = 250, colour="blue")



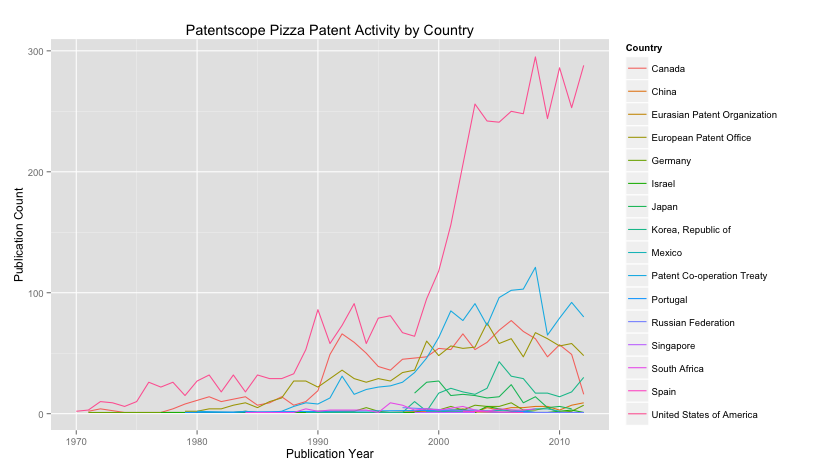
As this demonstrates, there are a variety of options that are available for annotating plots... although as the present example reveals they need to be handled with cautions. However, we now have a more or less reasonable line graph. However, to finish off this article maybe we would like to try different visual themes. Let's try that.

### Applying a theme

ggplot2 has a number of built in themes that are accessed by calling the function theme. However, note that some of these options are being deprecated. In addition, there is the excellent ggthemes package created by Jeffrey Arnold that we loaded earlier.

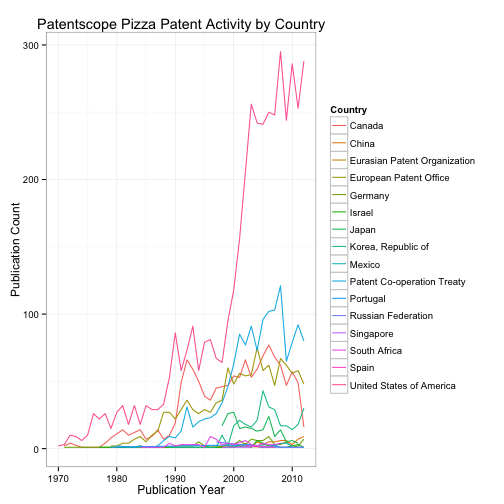
To make life a little easier for ourselves let's create an object with the basic settings for our line graph. We will leave the annotations off because we would want to decide on their form depending on our theme and add them later on.

line <- qplot(pubyear, n, data = pcy, geom = "line", color = pubcountry,   
 xlim = c(1970, 2012),   
 xlab = "Publication Year",   
 ylab = "Publication Count",   
 main = "Patentscope Pizza Patent Activity by Country") +   
 labs(colour = "Country")  
line



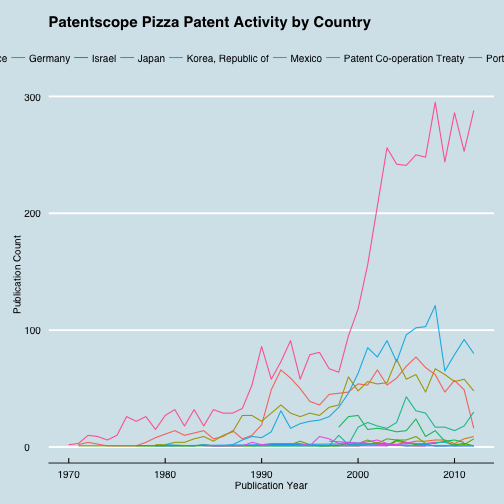
To add a theme try typing + theme at the end of our line object. Then select an option. In this example we have simply chosen black and white theme-bw().

line + theme\_bw()



Some of us may at one time or another have aspired to write for The Economist. Let's see what happens when we try the Economist magazine theme by adding + theme\_economist.

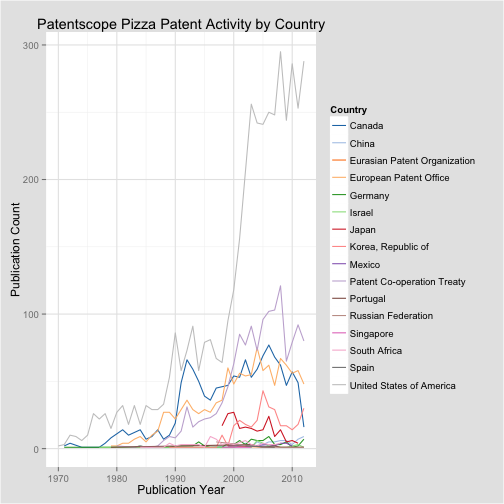
line + theme\_economist() + theme(legend.text=element\_text(size=10))



Here we might want to make some adjustments to the legend size (see above) but it's not bad.

In a previous article we used Tableau Public. We can more or less reproduce a Tableau plot as follows.

line + theme\_igray() + scale\_colour\_tableau("tableau20")



In this case we have added theme\_igray() to grey the background rather than the plot area. We have also specified scale\_colour\_tableau. Note that in some case a range of colour palettes are available. The default in the Tableau theme is the "tableau10"" scheme with ten colours. This means, in this case, if we do not specify "tableau20" we will not see all of the lines because we don't have enough colours.

There are a large number of themes to choose from in the ggthemes package and these are well worth exploring, including the use of color blind friendly palettes as also discussed by Winston Chang in the R Graphics Cookbook with examples [here](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/). See [Jeffrey Arnold's Github home page](https://github.com/jrnold/ggthemes) for details on how to use ggthemes and also the ggthemes packages details.

A reader familiar with R and ggplot2 will have noticed that as we added elements to the qplot graphics using + we are effectively starting to use the ggplot functions to control what we see on a graphic. More accurately, we are increasingly adding layers to the graphic with tighter and tighter control over what we are seeing. That is we are moving away from using qplot as an exploratory graphics tool towards the kind of controls we might use for a publication quality graphic (always bearing in mind we are graphing patent documents mentioning pizza).

Bear in mind when using R in general that there is normally more than one way to get where you want to go. Thus, there will be faster ways to get to the Tableau plot we have just produced. The way we have described above is one way, not necessarily the best way. What is important however, is that we can clearly see the steps involved in building up a graphic.

Our last Tableau style plot is a good place to move onto our final topic in Part 1. That is, saving our work.

## Saving a data.frame and Saving a plot

We have covered saving a .csv file in a detailed article. If we wanted to save on of our new tables on of the fastest ways to do that is using the readr write\_csv() function. We go into there options in the article Importing CSV files into R and a second article on Excel files in R. But, as the code below shows, it is very easy to write a file in R. You may want to use getwd() to get the working directory first so that you will know where the file will be stored.

write\_csv(pcy, "pcy.csv")

We can save the current plot easily in a range of formats sing the ggsave function. We can also save plots from the Plot tab next to the Files tab in a variety of formats.

ggsave("plot.png", width = 5, height = 5)

A copy of the last plot will now appear in the working directory (use getwd() in the console to see your working directory). The name of the file and the width and height can of course be changed. ggsave can save to a range of formats including eps/ps, tex (pictex), pdf, jpeg, tiff, png, bmp, svg and wmf (windows only). If you find that your plot does not save as you expect (e.g. it is crunched) then try the Plot window options instead. For text outlining and other operations for publication quality graphics consider using the free [GNU Impage Manipulation Tool or GIMP](http://www.gimp.org).

If we wanted to go back one plot and save that we could use last\_plot(). Give it a try.

last\_plot()

Note here that we can only go back one plot (not one plot, then one plot and so on). So, if you have something you want to save, it is probably best to do that as you go.

## Round Up

In this article we have shown how to get started with R using patent data by focusing on graphing data using the ggplot2 package and the qplot function for our pizza patents. We started this article by looking at ways to easily:

1. import data using readr
2. manipulate data using dplyr

In part 2, we will go into more detail on dplyr as a key tool for wrangling patent data, notably using select, filter and mutate.

In the second part of the article we focused on developing some basic graphs of our pizza patent data using qplot in ggplot2. As we moved through we increasingly began to use the + sign and to add more and more layers to control what we were seeing. In the process we are effectively starting to use the grammar of graphics approach of layered graphics which is where the ggplot sister function to qplot excels. In part 2 we will go into more detail on the use of ggplot.

In thinking about using R for patent analysis, one of the challenges is that it can be maddeningly frustrating to import actual data (rather than training or toy datasets) and to use the range of functions and arguments. Learning and using R requires patience. On the other hand, the reason that we are focusing on R, and RStudio as a user interface, is that no other open source tool offers this range of functions, the ability to operate at scale and options in terms of analysis and visualisation. A complete open source solution for patent analytics is probably offered by a combination of R with Python (notably for data cleaning and text mining). However, it makes sense to start with R.

A second reason for favouring R over other tools are the extensive resources and communities that exist around R as an ecosystem. If you get stuck, and this happens quite a lot, someone else has been there before you. With R you are never alone.

We will end with some useful resources to take you further in visualising patent data with ggplot2 both as sources of inspiration and for when you get stuck.

### Useful Resources

1. [RStudio Cheatsheet](http://www.rstudio.com/wp-content/uploads/2015/05/ggplot2-cheatsheet.pdf)
2. [R Graphics Cookbook](http://www.cookbook-r.com/Graphs/) by Winston Chang
3. [Hadley Wickham 2010 A Layered Grammar of Graphics preprint article](http://vita.had.co.nz/papers/layered-grammar.pdf)
4. [Hadley Wickham ggplot2 book from Amazon](http://www.amazon.co.uk/Hadley-Wickham/e/B002BOA9GI/ref=sr_tc_2_0?qid=1435678538&sr=1-2-ent)
5. Swirl tutorials (install.packages("swirl")) and [Github repository](https://github.com/swirldev/swirl_courses)
6. [ggplot2 online help topics](http://docs.ggplot2.org/0.9.3/index.html#)
7. [R-Bloggers on ggplot2](http://www.r-bloggers.com/search/ggplot2)
8. [Stack Overflow questions and answers on ggplot2](http://stackoverflow.com/questions/tagged/ggplot2)
9. [YouTube ggplot 2 videos](https://www.youtube.com/results?search_query=ggplot2)